TESLA: TASK-WISE EARLY STOPPING AND LOSS AGGREGATION FOR DYNAMIC NEURAL NETWORK INFERENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

For inference operations in deep neural networks on end devices, it is desirable to deploy a single pre-trained neural network model, which can dynamically scale across a computation range without comprising accuracy. To achieve this goal, Incomplete Dot Product (IDP) has been proposed to use only a subset of terms in dot products during forward propagation. However, there are some limitations, including noticeable performance degradation in operating regions with low computational costs, and essential performance limitations since IDP used hand-crafted profile coefficients. In this paper, we extend IDP by proposing new training algorithms involving a single profile, which may be trainable or pre-determined, to significantly improve the overall performance, especially in operating regions with low computational costs. Specifically, we propose the Task-wise Early Stopping and Loss Aggregation (TESLA) algorithm, which is showed in our 3-layer multilayer perceptron on MNIST that achieves 94.7% accuracy on average, and especially outperforms the original IDP by 32% when only 10% of dot products we used. We further introduce trainable profile coefficients, with which TESLA further improves the accuracy to 95.5% without specifying coefficients in advance. We also apply TESLA to the VGG-16 model on CIFAR-10 and it achieves 80% accuracy using only 20% of dot products, where the original IDP does not work. Finally, we visualize the learned representations at different dot product percentages by class activation map and show that, by applying TESLA, the learned representations can adapt over a wide range of operation regions.

1 INTRODUCTION

Inference operations in deep neural networks on end devices, such as mobile phones, embedded sensors, IoT devices, etc., have recently received increasing attention including McMahan et al. (2016), Howard et al. (2017), and Teerapittayanon et al. (2017). In such applications, it is desirable to deploy a single pre-trained CNN model on end devices, while allowing multiple operating regions to meet different power consumption, latency, and accuracy requirements. To achieve this goal, McDanel et al. (2017a) proposed the incomplete dot product (IDP) operation, where only a subset of terms is used in dot products of forward propagation. From now on, x% DP, where $\leq x \geq 100$, means the x% of terms used in dot products. As illustrated in Figure 1, 50% DP means half of filters are used during forward propagation, and only half of the output channels are retained. To reduce the deviation induced by IDP, filters are prioritized from most important to the least important by pre-determined monotonically non-increasing profile coefficients (say, $\gamma_1, ..., \gamma_N$) during training. Therefore, IDP can be applied at inference time with dynamically-adjusted degrees of completeness (specified by the percentage of terms being used) to trade off accuracy slightly for lowered power consumption and reduced latency. Specifically, VGG-16 model with 50% DP achieves 70% in accuracy on the CIFAR-10 dataset compared to the standard network achieves only 35% accuracy when using the reduced channel set.

While the original IDP design seems promising, there are two limitations. First, since the training process aims at optimizing the loss function computed using all weights of the model (100% DP), there will be a mismatch between training and testing. It is no surprise that inference performance significantly decreases in low DP percentages and thus narrow the dynamic computation range. To

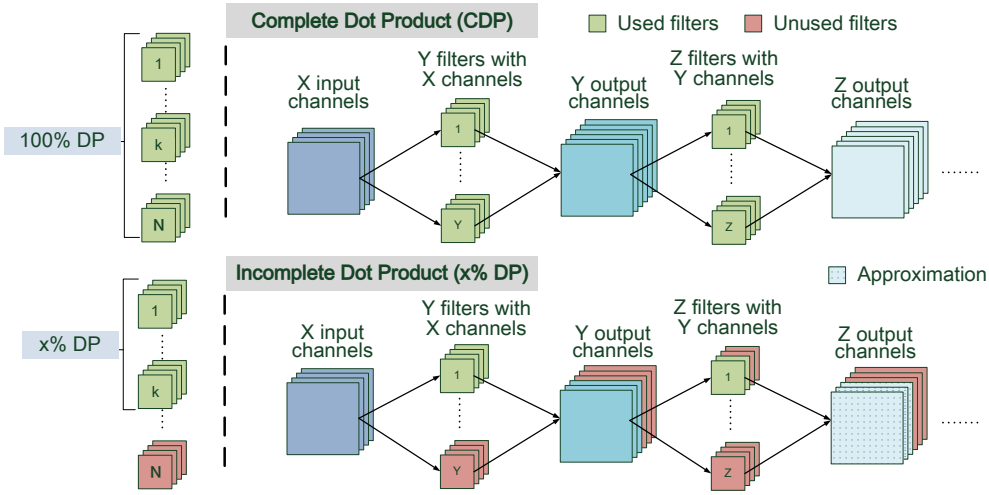


Figure 1: Visualize incomplete dot product

mitigate this problem, the original IDP design utilizes the multiple-profile training strategy, where different profiles can be specified to focus on different dot product ranges. In such a multiple-profile training process, however, certain subset of weights will be freezed in each training stage corresponding to the profile being focused, and hence the overall performance may not be fully optimized. Besides, each profile needs to maintain a separate first and last layer for adjusting to its own dot product range, resulting in additional memory overhead. The second limitation relates to the pre-determined nature of profile coefficients. While there are multiple ways to set the profile based on different dynamic range requirements, the original IDP design did not focus on finding a single "best" profile that leads to the best performance. Instead, they use multiple hand-crafted profile coefficients, which make the system design less general among different applications, and hence may limit the overall performance of the system.

To reduce the mismatch between training and testing performances, we propose the Task-wise Early Stopping and Loss Aggregation (TESLA) algorithm, in which multiple loss functions are computed in different DP percentages. By gradually aggregating these loss functions in decreasing order of DP percentages as the objective function to be optimized, TESLA significantly improves testing performances in low DP percentages without compromising accuracy in medium to high DP percentages. The loss functions can also be aggregated in random order of DP percentages to make a variant of TESLA, called Randomized TESLA (R-TESLA), which enables better performances under pre-specified operating regions of end devices. Moreover, we relax the constraint of pre-determined profile coefficients and propose the alternate training procedure (ATP) to alternately train the profile coefficients along with weights of the model. By introducing trainable profile coefficients, customization among different applications can be achieved in a more generalized way, and the overall performance can also be further improved. This paper has made two major contributions: (1) We propose the Task-wise Early Stopping and Loss Aggregation (TESLA) algorithm and Randomized TESLA that can achieve dynamic scaling over a computation range in neural network inference without compromising accuracy. (2) We also propose the Alternate Training Procedure (ATP) that can learn the profile coefficients and the model weights simultaneously without the need of manual configuration of the profile coefficients.

2 Incomplete Neural Networks

Incomplete dot product (IDP) is a novel mechanism proposed by McDanel et al. (2017a) that can be applied to a hidden layer of MLPs or deep CNN models to dynamically lower the inference costs by computing only a subset of terms in dot products during forward propagation. By introducing a set of non-increasing coefficients γ_i, referred to as a profile, to the channels during training, the channels will be ordered implicitly in non-increasing order from the most important to the least important. By simply dropping out less important channels at inference time, it suffices to train

and deploy a single network, while still supporting different levels of computation scaling without compromising accuracy significantly. In this section, we briefly introduce the main concepts of IDP.

2.1 INCOMPLETE DOT PRODUCT OPERATION

Mathematically, for an IDP fully-connected layer with input dimension N and output dimension M, the j-th output component y_j is computed as

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

for $j \in \{1, 2, ..., M\}$, where x_i is the i-th input component, w_{ji} is the weight corresponding to the j-th output component and the i-th input component, and γ_i is the i-th profile coefficient.

Similar expression can be derived for the IDP operation applied to a convolutional layer of CNN, as illustrated in Figure 1. For an IDP convolutional layer with number of input channels N and number of output channels M, the j-th output channel $\mathbf{y}_j$ is computed as

$$\mathbf{y}_j = \gamma_j \sum_{i=1}^{N} \mathbf{f}_{ji} * \mathbf{x}_i, \tag{2}$$

for $j \in \{1, 2, ..., M\}$, where $\mathbf{f}_{ji} * \mathbf{x}_i$ denotes the convolution operation of the i-th input channel $\mathbf{x}_i$ and the i-th channel of the j-th filter $\mathbf{f}_{ji}$, and γ_j is the profile coefficient for the j-th filter. Note that, instead of applying profile coefficients depthwise on each filter before convolution as is the case in the original IDP design, we multiply each γ_j to each output channel after a complete convolution to produce $\mathbf{y}_j$. These two approaches, however, are equivalent with negligible difference induced by the first hidden layer. Since the output channels $\mathbf{y}_j$'s become input channels $\mathbf{x}_i$'s to the next layer, applying γ_j's to $\mathbf{y}_j$'s is equivalent to applying them into the convolution operation in the next layer.

To compute IDP with a target dot product percentage, a truncated version of Eq. 1 or Eq. 2 replaces the original computation to keep only a subset of the beginning terms. As for the case with all terms are kept, we refer to such operations as complete dot product (CDP) or 100% DP, interchangeably. Note that in the training process in the original IDP design, only CDP is used.

2.2 MULTIPLE-PROFILE INCOMPLETE NEURAL NETWORKS

In the work of McDanel et al. (2017a), several profile coefficients are proposed and applied in a pre-determined manner. When only a single profile is applied to the model, the trade-off between computation range and performance in high DP percentage regions is also demonstrated. Generally, the faster the profile coefficients decrease, the larger computation range can be achieved, at the expense of a performance degradation in high DP percentage regions. To cover a larger computation range while maintaining the performance in high DP percentage regions, McDanel et al. (2017a) further introduced the multiple-profile incomplete neural networks (MP-IDP), where different profiles can be specified to focus on different DP ranges. During training, all the specified profiles are applied in increasing order of their operating DP ranges. When a profile is applied, only weights corresponding to its operating DP range will be updated, leaving weights corresponding to lower DP percentages freezed since they have been trained in previous stages, and weights corresponding to higher IDP percentages set to zeros since they will be trained in later stages. In such a stage-by-stage training process, the overall performance may not be fully optimized.

3 TASK-WISE EARLY STOPPING AND LOSS AGGREGATION

As discussed in Section 2, in the original IDP design, only CDP is used during training but IDP can be applied during inference. This leads to a noticeable degradation in performance at inference time, especially in low DP percentages. To mitigate this problem, we propose the Task-wise Early Stopping and Loss Aggregation (TESLA) algorithm. In this paper, a *task* is defined as the learning process that uses only a subset of weights specified by a DP percentage to learn the optimal representations of the network. The design of TESLA is described as follows.

3.1 TASK-WISE EARLY STOPPING

Tasks with different DP percentages may have different learning difficulties and converging rates. Consider two tasks, one of which applies 40% DP (task 1) and the other 70% DP (task 2). If we optimize these two tasks separately with the same hyper-parameters (learning rate, number of epoches, etc.), it may happen that the model for task 1 ended-up overfit the training data, while the model for task 2 still underfit the training data. To simplify the learning procedure for all tasks, we keep all hyper-parameters unchanged except the numbers of epoches, which are controlled by the early stopping mechanism that halts the training process if the model performance has not been improved for a specific number of iterations. With task-wise early stopping, we are able to optimize all the tasks sequentially, with each task initializes its model using the weights that have been optimized for all previous tasks. In the above two-task example, we can first try to optimize task 2 using 70% DP, and then, after meeting some early-stopping criterion, switch to task 1 using 40% DP. Note that, since 40% DP use the first 40% of terms, which is exactly a subset of terms used in 70% DP, the optimizing process for task 1 may destroy the optimality of task 2. To deal with this problem, some kinds of loss aggregation is needed in order to learn the shared representations for all the tasks.

Algorithm 1 Task-wise Early Stopping and Loss Aggregation, TESLA

Input: a task set in decreasing order, $T = L_i$; aggregation coefficient α
Initialization: $L_1^{obj} \leftarrow L_1$ and $i \leftarrow 2$
while $i < size(T)$ **do**
  TaskwiseEarlyStopping(L^{obj})
  $L_i^{obj} \leftarrow$ TaskwiseLossAggregation(L_{i-1}^{obj}, L_i)
  $i \leftarrow i + 1$
end while

Algorithm 2 Randomized TESLA, R-TESLA

Input: a task set in any order, $T = L_i$; allowable epoch, max_epoch; aggregation coefficient α
Initialization: $L_1^{obj} \leftarrow L_1$ and $i \leftarrow 0$
while $i < max_epoch$ **do**
  TaskwiseEarlyStopping(L_i^{obj}), which takes n_epochs
  Sample a task called L_k
  $L_i^{obj} \leftarrow$ TaskwiseLossAggregation(L_i^{obj}, L_k)
  $i \leftarrow i + n_epcohs$
end while

3.2 TASK-WISE LOSS AGGREGATION

Task-wise loss aggregation is therefore proposed in this paper to enable multiple tasks being optimized *incrementally* and *jointly* without freezing any weight. By introducing one task at a time, the aggregated loss function can be expressed as

$$L_1^{obj} = L_1 \ \text{ and } \ L_i^{obj} = \alpha \times L_i + (1 - \alpha) \times L_{i-1}^{obj} \ , \ \ \forall i = 2, \cdots, N, \tag{3}$$

where α is the aggregation coefficient shared by all subsequent tasks. In other words, the task-wise loss aggregation is the weighted sum of the current objective function and the loss function of the newly-added task. As a consequence, task-wise loss aggregation allows the errors of related tasks to be back-propagated jointly to construct a shared representation to be relevant to all tasks.

3.3 TESLA AND RANDOMIZED TESLA

Task-wise Early Stopping and Loss Aggregation, TESLA. We integrate task-wise early stopping and task-wise loss aggregation as TESLA to learn dynamic representations in neural networks. The entire training process optimizes all tasks in an arbitrary order. It is obvious that we have several options to order tasks in (i) increasing, (ii) decreasing, or (iii) random DP percentages. Recall that we add a non-increasing coefficients to prioritize terms in computing dot product, and thus the beginning

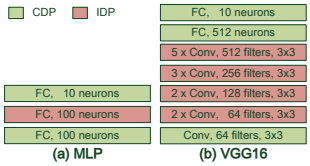


Figure 2: Used network structures.

Table 1: Hyper-parameters in Experiments

Experiment	MLP on MNIST	VGG-16 on CIFAR-10
Tasks at DP %	100, 70, 40, 10	100, 50, 20
Learning rate	0.001	0.004
Optimizer	Adam	SGD momentum = 0.9
Batch size	28	32
Aggregation coefficient	0.5	0.5
TESLA stopping criteria	not improve in 4 epochs	not improve in 4 epochs
R-TESLA stopping criteria	# epochs over 50	# epochs over 35
Initial weights	random	pre-trained on ImageNet

terms, e.g. at 10% DP, are more important than the terms at last 10% terms. Therefore, discarding the terms from the end is less harmful to the optimized parameters, so TESLA is designed to optimize tasks in decreasing order of DP percentages. The TESLA algorithm is shown in Algorithm 1.

Randomized Task-wise Early Stopping and Loss Aggregation, R-TESLA. Here *Randomized* means that tasks are optimized in random order. The benefits of R-TESLA is two fold. First, R-TESLA provides an opportunity to turn attention back to optimize a task which had been halted before, and allows to finetune the weights, which may have been contaminated by other tasks. Second, unlike TESLA that optimizes each task only once, R-TESLA allows each task to be optimized for multiple times, which can be specified by a customized task distribution derived from the behavioral statistics of users or the specification of hardware design. The detailed procedures of R-TESLA are in Algorithm 2.

3.4 Trainable Profile Coefficients

In this section we propose to learn profile coefficients along with weights of the model alternately. We initialize all coefficients as one and as long as any update of profile coefficients, we manually clip the coefficients to keep the non-increasing property. The alternate training procedure (ATP) relaxes the constraint of fixed coefficients and we demonstrate the feasibility of ATP in the experiment of the MLP model on MNIST dataset in Section 4.

4 Experiments

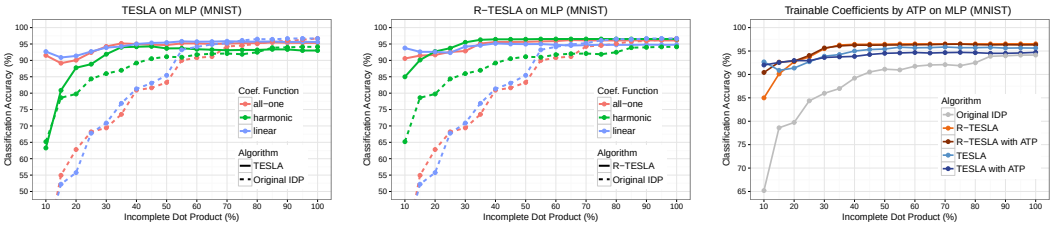
In this section, we demonstrate the effectiveness of using TESLA and R-TESLA to learn dynamic representations in MLP and CNN models, with the widely-used datasets MNIST and CIFAR-10, respectively. Figure 2 shows the network architectures in study. Here we compare TESLA and R-TESLA with the original IDP design proposed by McDanel et al. (2017a) over a range of dynamic scaling during inference. All hyper-parameters and experiment settings are summarized in Table 1.

4.1 Multilayer Perceptrons

First, we consider a 3-layer MLP model, in which the IDP operation is applied to the first hidden layer, as shown in Figure 2(a), and evaluate on the MNIST dataset. In this experiment, we define 4 tasks that optimize representations at 10%, 40%, 70%, and 100% DP, respectively. It is noteworthy that defining too many tasks in our experiment would not benefit much, since there must be a large amount of shared parameters among tasks which makes the model vulnerable to overfitting.

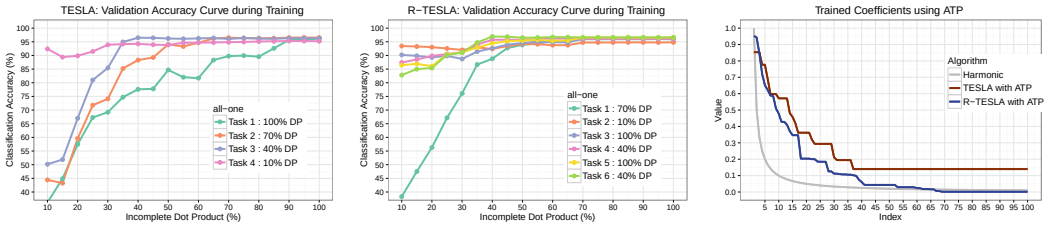
TESLA versus original IDP. We compare TESLA and the original IDP deign under various profiles. Figure 3(a) shows that TESLA supports larger dynamic computation range than original IDP in all cases. TESLA with linear profile keeps the performance in all DP percentages above 90% accuracy with an average 94.7%. As for the original IDP, we observe that, at 20% DP, the harmonic profile achieves an accuracy of 80%, but only 63% and 55% accuracy by the all-one and the linear profiles, respectively. Besides, Figure 4(a) depicts the performance improvement task by task. We observe that TESLA, after completing all the tasks, only performs about 1% worse in accuracy at 100% DP, with a significant improvment in accuracy from 50% to 90% at 10% DP, which is an acceptable trade-off under practical applications.

R-TESLA versus TESLA and original IDP. Next, we compare R-TESLA with TESLA and the original IDP together. Figure 3(b) shows that R-TESLA outperforms the original IDP by a large



(a) **(b)** **(c)**

Figure 3: A comparison between TESLA, R-TESLA, and ATP on MLP (MNIST)



(a) **(b)** **(c)**

Figure 4: Performance history of TESLA (a) and R-TESLA (b); learned profile coefficients by ATP (c).

margin. R-TESLA with the harmonic profile leads to the best average accuracy at 95.2%. By observing the optimizing progress in Figure 4(a) and 4(b), we find that TESLA achieves its best result after completing the last task thanks to its ordinal optimization. On the other hand, we cannot ensure that R-TESLA can make the ultimate model retains the best dynamic representations due to its random nature.

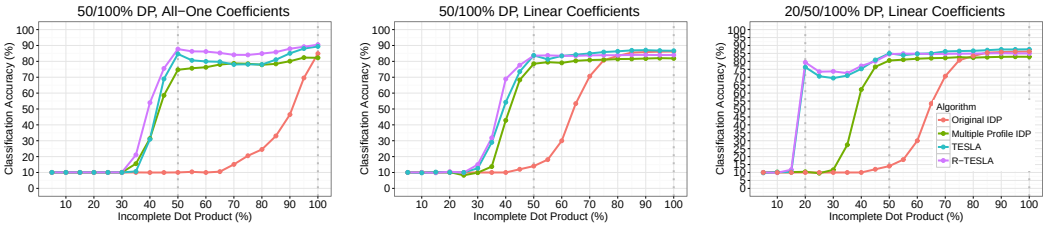
Learn profile coefficients by ATP. Figure 3(c) shows with the help of trainable coefficients both TESLA and R-TESLA further boost by 1% in average, and we find that the learned profile coefficients are similar to harmonic ones as shown in Figure 4(c). This may support why performance of harmonic coefficients is the best in the original IDP design. By allowing coefficients to be trainable, it is no longer to require pre-determining hand-crafted coefficients.

4.2 Convolutional Neural Networks

We choose the known VGG-16 model pre-trained on ImageNet to evaluate over CIFAR-10 dataset. The last few dense layers are replaced by a single 10-class classifier. We use the all-one and the linear profile coefficients to compare: (i) the original IDP design, (ii) multiple-profile IDP design (MP-IDP) as proposed in McDanel et al. (2017a), (iii) TESLA, and (iv) R-TESLA. The experimental results are summarized below.

Algorithms optimized at 50% and 100% DP. Applying all-one coefficient is equivalent to using the original VGG-16 network, with the only difference when using IDP operation. We can see from Figure 5(a) that the performance of original IDP drops much faster in this experiment compared to the previous MLP on MNIST since CIFAR-10 is a more challenging dataset. Despite the poor performance in the original IDP design at 50% DP, applying the MP-IDP does enlarge the computational range, with an increase in accuracy to 75%. Furthermore, both TESLA and R-TESLA boost the accuracy to reach 85%, and an even higher accuracy at 100% DP. With the linear profile coefficients as shown in Figure 5(b), TESLA, R-TESLA, and MP-IDP all achieve better performance in most dot product percentages and significantly wider computational ranges. It is also worth mentioning that using the linear profile generates a smoother performance curve between 50% and 100% DP compared to the all-one profile. This beneficial trait accomplished by using the linear profile guarantees that end-device users won't notice substantial performance drop when applying dynamic computational resource scaling.

Algorithms optimized at 20%, 50% and 100% DP. Compared with the previous results, Figure 5(c) shows that our proposed TESLA and R-TESLA greatly widens the computational ranges by



(a) all-one (at 50%, 100%) **(b)** linear (at 50%, 100%) **(c)** linear (at 20%, 50%, 100%)

Figure 5: A comparison with VGG-16 on CIFAR-10.

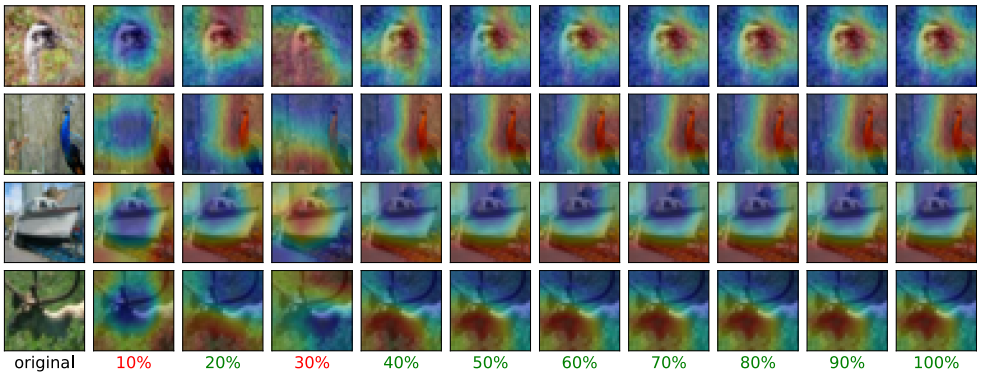


Figure 6: CAMs at different DP percentages. Red colored text means wrong prediction and green colored text means correct prediction.

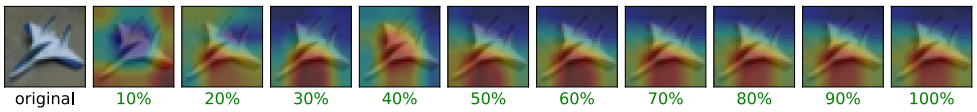


Figure 7: CAMs of a testing image that is correctly classified at all specified DP percentages.

making accuracy reaching 75% at 20% DP. We contribute this effect to applying TESLA and R-TESLA in decreasing order of dot product percentages. The representation learned at 100% DP drives the representation learned at 50% DP, which also makes the representation much easier to learn at 20% DP. Despite the usage of linear profile coefficients, there still exists a performance drop between 20% DP and 50% DP, we believe this can be alleviated if we optimized at more points. Compared to TESLA and R-TESLA, MP-IDP trains models in increasing order of DP percentages but when optimizing at more and lower DP percentages, no improvement is made by MP-IDP. We believe that this is due to the limited network capacity makes the network hard to train.

CAM visualization. We visualize what the convolutional filters of CNN learned at different DP percentages by deploying the Class Activation Mapping (CAM) technique introduced by Zhou et al. (2016a). A resulting CAM indicates how much each location contribute to the final class prediction. In this stage, we replace the max-pooling layers with average-pooling layers and train the VGG-16 network with linear coefficients optimized at 20%, 50%, 100% DP. From CAMs at different DP percentages, we found that the network made wrong prediction at 10% and 30% DP but made correct prediction at 20% DP as shown in Figure 6 since the representations at 20% DP are optimized. This finding implies that we can specify any DP percentages to be optimized for satisfying any requirements. We also notice that the CAMs at 10% DP are almost the same across several images, which indicates too limited capacity to capture patterns. We extensively test the VGG-16 on CIFAR-10 by using TESLA, R-TESLA and MP-IDP methods with all-one and linear coefficients optimized at different DP percentages. However, we also found that the harmonic coefficients did not work in this case. One possible reason is that harmonic coefficients decrease quickly and extremely close to zero when large numbers of filters in use, so the resulting representations are almost blank. This observation sparked our following experiment on making coefficients trainable.

5 RELATED WORK

Our work is rooted from IDP proposed by McDanel et al. (2017a), which, in addition to MLPs and regular CNNs, can also be used in conjunction with other variants of convolutional layers, such as separable convolution layer Howard et al. (2017) and binary convolutional layer McDanel et al. (2017b). As discussed throughout this paper, our work extends the original IDP design by proposing new training algorithms involving a single profile, which may be trainable or pre-determined, to significantly improve the overall performance, especially in low DP percentages.

Network pruning is a widely-studied area that also aims at compressing the CNN models. Early works of network pruning construct a threshold for dropping weights by information obtained from Hessian matrix or inverse Hessian matrix in LeCun et al. (1990); Hassibi & Stork (1993), which adds memory and computation costs. In most of the recent works, magnitude-based pruning and recovering are incorporated to compensate the potential loss incurred by inadequate pruning. For example, Guo et al. (2016) introduces the splicing operation to enable connection recovery, and Han et al. (2016) directly makes the network dense again. Li et al. (2016) also prune filters in CNNs based on magnitude, but the number of filters pruned away in each layer is decided by layer-wise sensitivity. Besides magnitude-based pruning, a Taylor expansion-based criterion is introduced in Molchanov et al. (2016) to approximate the change in the cost function induced by pruning. In addition to network pruning, some works focus on low-rank decomposition for network compression. For example, Denton et al. (2014) and Jaderberg et al. (2014) approximate the weight matrix into low-rank components by minimizing the reconstruction error. Yu et al. (2017) further decomposes the weight matrix into its low-rank and sparse component. Other works focus on grouping similar weights, such as quantization by Han et al. (2015), Gong et al. (2014), and Zhou et al. (2017) and weight sharing by Ullrich et al. (2017), aiming at reducing the level of redundancy and the required storage. Yet another approach introduces group sparsity regularizer to constrain the structure of the model in Wen et al. (2016), Zhou et al. (2016b), and Alvarez & Salzmann (2016).

While all the above techniques are promising in reducing the size of the networks, none of them supports dynamic adjustment during inference as IDP does. Furthermore, most of the above techniques involve retraining the model iteratively, resulting in computational overhead. In our proposed work, the goal of efficient inference with dynamic adjustment can be readily fulfilled by training a single model at once, and the effectiveness is expected to be further improved by incorporating with other techniques listed above.

6 CONCLUSION

In this paper, we extend the idea of incomplete dot product (IDP) by proposing the Task-wise Early Stopping and Loss Aggregation (TESLA) algorithm to significantly improve the performance of neural networks with dynamically computation regions at inference time without significantly compromising accuracy. A task is defined as the learning process that uses only a subset of weights specified by an DP percentage to learn the optimal representations of the network. By introducing non-increasing profile coefficients to prioritize weights or filters during training, TESLA can be used to optimize multiple tasks in decreasing order of DP percentages by aggregating the their loss functions. Additionally, we propose Randomized TESLA (R-TESLA) which optimizes tasks in random order, and show that both TESLA and R-TESLA outperform original IDP and multiple-profile IDP significantly. The visualization of the class activation maps (CAMs) provide a strong evidence that the representations learned by TESLA allow dynamically scaling across a computation range to meet various power consumption, latency and accuracy requirements on end devices.

REFERENCES

Jose M Alvarez and Mathieu Salzmann. Learning the number of neurons in deep networks. In *Advances in Neural Information Processing Systems*, pp. 2270–2278, 2016.

Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in Neural Information Processing Systems*, pp. 1269–1277, 2014.

Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*, 2014.

Yiwen Guo, Anbang Yao, and Yurong Chen. Dynamic network surgery for efficient dnns. In *Advances In Neural Information Processing Systems*, pp. 1379–1387, 2016.

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

Song Han, Jeff Pool, Sharan Narang, Huizi Mao, Enhao Gong, Shijian Tang, Erich Elsen, Peter Vajda, Manohar Paluri, John Tran, et al. Dsd: Dense-sparse-dense training for deep neural networks. 2016.

Babak Hassibi and David G. Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in Neural Information Processing Systems 5*, pp. 164–171. 1993.

Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*, 2014.

Yann LeCun, John S. Denker, and Sara A. Solla. Optimal brain damage. In *Advances in Neural Information Processing Systems 2*, pp. 598–605. 1990.

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

Bradley McDanel, Surat Teerapittayanon, and HT Kung. Incomplete dot products for dynamic computation scaling in neural network inference. 2017a.

Bradley McDanel, Surat Teerapittayanon, and H.T. Kung. Embedded binarized neural networks. In *Proceedings of the 2017 International Conference on Embedded Wireless Systems and Networks*, 2017b.

H Brendan McMahan, Eider Moore, Daniel Ramage, and Blaise Aguera y Arcas. Federated learning of deep networks using model averaging. 2016.

Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. 2016.

Surat Teerapittayanon, Bradley McDanel, and HT Kung. Distributed deep neural networks over the cloud, the edge and end devices. In *Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on*, pp. 328–339. IEEE, 2017.

Karen Ullrich, Edward Meeds, and Max Welling. Soft weight-sharing for neural network compression. *arXiv preprint arXiv:1702.04008*, 2017.

Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems*, pp. 2074–2082, 2016.

Xiyu Yu, Tongliang Liu, Xinchao Wang, and Dacheng Tao. On compressing deep models by low rank and sparse decomposition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7370–7379, 2017.

Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. Incremental network quantization: Towards lossless cnns with low-precision weights. *arXiv preprint arXiv:1702.03044*, 2017.

Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2921–2929, 2016a.

Hao Zhou, Jose M Alvarez, and Fatih Porikli. Less is more: Towards compact cnns. In *European Conference on Computer Vision*, pp. 662–677. Springer, 2016b.