Model Slicing for Supporting Complex Analytics with Elastic Inference Cost and Resource Constraints

Shaofeng Cai[†], Gang Chen[§], Beng Chin Ooi[†], Jinyang Gao[‡]

[†]*National University of Singapore* {shaofeng, ooibc}@comp.nus.edu.sg

[§]*Zhejiang University* cg@zju.edu.cn

[‡]*Alibaba Group* jinyang.gjy@alibaba-inc.com

ABSTRACT

Deep learning models have been used to support analytics beyond simple aggregation, where deeper and wider models have been shown to yield great results. These models consume a huge amount of memory and computational operations. However, most of the large-scale industrial applications are often computational budget constrained. In practice, the peak workload of inference service could be 10x higher than the average cases, with the presence of unpredictable extreme cases. Lots of computational resources could be wasted during off-peak hours and the system may crash when the workload exceeds system capacity. How to support deep learning services with dynamic workload cost-efficiently remains a challenging problem. In this paper, we address the challenge with a general and novel training scheme called *model slicing*, which enables deep learning models to provide predictions within the prescribed computational resource budget dynamically. *Model slicing* could be viewed as an elastic computation solution without requiring more computational resources. Succinctly, each layer in the model is divided into *groups* of contiguous block of basic components (i.e. neurons in dense layers and channels in convolutional layers), and then partially ordered relation is introduced to these groups by enforcing that groups participated in each forward pass always starts from the *first* group to the *dynamically-determined rightmost* group. Trained by dynamically indexing the rightmost group with a single parameter *slice rate*, the network is engendered to build up group-wise and residual representation. Then during inference, a sub-model with fewer groups can be readily deployed for efficiency whose computation is roughly quadratic to the width controlled by the *slice rate*. Extensive experiments show that models trained with *model slicing* can effectively support on-demand workload with elastic inference cost.

PVLDB Reference Format:
Shaofeng Cai, Gang Chen, Beng Chin Ooi, Jinyang Gao. Model Slicing for Supporting Complex Analytics with Elastic Inference Cost and Resource Constraints . *PVLDB*, 13(2): 86-99, 2019.
DOI: https://doi.org/10.14778/3364324.3364325

Proceedings of the VLDB Endowment, Vol. 13, No. 2
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3364324.3364325

1. INTRODUCTION

Database management systems (DBMS) have been widely used and optimized to support OLAP-style analytics. In present-day applications, more and more data-driven machine learning based analytics have been grafted into DBMS to support complex analysis (e.g., stock prediction, disease progression analysis) and/or to enable predictive query and system optimization. To better understand the data and decipher the information that truly counts in the era of Big Data with its ever-increasing data size and complexity, many advanced large-scale machine learning models have been devised, from million-dimension linear models (e.g., Logistic Regression [40], feature selection [55]) to complex models like Deep Neural Networks [30]. To meet the demand for more complex analytic queries, OLAP database vendors have integrated Machine Learning (ML) libraries into their systems (e.g., SQL Server pymssql[1], DB2 python_ibm_db[2] and etc). It is widely recognized that the integration of ML analytics into data systems yields seamless effects since the ML task is treated as an operator of the query plan instead of an individual black-box system on top of data systems. Naturally, a higher-level abstraction provides more space for optimization. For example, query planning [42, 31], lazy evaluation [57], materialization [55] and operator optimization [1] could be considered in a fine-grained manner.

Cost and accuracy are always the two most crucial criteria considered for analytic tasks. Lots of research on approximate query processing have been conducted [33, 4] to provide faster yet approximate analytical query results in modern large-scale analytical database systems, while such a trade-off is not equally well researched for modern ML analytic tasks, particularly deep neural network models. There are two characteristics of the inference cost of analytic tasks for deep neural network models. Firstly, with the development of high-end hardware and large-scale datasets, recent deep models are growing deeper [30, 16] and wider [53, 51]. State-of-the-art models have been designed with up to hundreds of layers and tens of millions of parameters, which leads to a dramatic increase in the inference cost. For instance, a 152-layer ResNet [16] with over 60 million parameters requires up to 20 Giga FLOPs for the inference of one single 224×224 image. The surging computational cost severely affects the viability of many deep models in industry-scale applications. Secondly, for most of the analytic tasks, the workload is usually not constant, e.g., the

[1]https://docs.microsoft.com/en-us/sql/connect/python/pymssql/python-sql-driver-pymssql
[2]https://github.com/ibmdb/python-ibmdb

number of images per query for person re-id [58] service in peak hours could be five times more than the workload in the off-peak hours. Therefore, such a trade-off should be naturally supported in the inference phase rather than the training phase: using one single deep model with fixed inference cost to support the peak workload could lead to huge amounts of resources wasting in off-peak hours, and may not be able to handle the unexpected extreme workload. How to trade off the accuracy and cost during deep model inference remains a challenging problem of great importance.

Existing model architecture re-design [25, 20] or model compression [14, 15, 35] methods are not able to handle elastic inference satisfactorily, and we shall use an application example to highlight the challenges. Singles′ Day shopping festival[3] around 11 November was introduced by Taobao.com and is now becoming one of the biggest online shopping festivals around the world. In 2018, the Singles′ Day festival generated close to 30 billion dollars of sales in one single day and had attracted hundreds of millions of users from more than 200 different countries. The peak level of trade rate reached 0.256 million per second, and 42 million processing in the database in the first half hour. In Singles′ Day, the search traffic of the e-commerce search engine increases about three times than in a common day, and could be 10x in its first hour. Meanwhile, the workload of most other services in Alibaba such as OLTP transaction may also hit the peak at the same time [3], and consequently, it is not possible to scale up the service by acquiring more hardware resources from Alibaba Cloud. The system degradation is often executed in two simple and naive approaches: First, some costly deep learning models are replaced by simple GBDT [6, 28] models; Second, the size of the candidate items for ranking is reduced. The search accuracy suffers dramatically due to the system degradation in such a coarse-grained manner. With a deep learning model supporting elastic inference cost, the system degradation management can become more fine-grained where the inference cost and accuracy trade-off per query sample can be dynamically determined based on the current system workload.

In this paper, instead of constructing small models based on each individual workload requirement, we propose and address a related but slightly different research problem: developing a general framework to support deep learning models with elastic inference cost. We base the framework on a pay-as-you-go model to support dynamic trade-offs between computation cost and accuracy during inference time. That is, dynamic optimization is supported based on system workload, availability of resources and user requirements.

An ML model abstraction with elastic inference cost would greatly benefit the optimization of the system design for complex analytics. We shall examine the problem from a fresh system perspective and propose our solution – *model slicing*, a general network training mechanism supporting elastics inference cost, to satisfy the run-time memory and computation budget dynamically during the inference phase. The crux of our approach is to decompose each layer of the model into groups of a contiguous block of basic components, i.e. neurons in dense layers and channels in convolutional layers, and facilitate *group residual learning* by imposing partially ordered relation on these groups. Specifically, if one group participates in the forward pass of model computation, then all of its preceding groups in this layer are also activated under such a structural constraint. Therefore, we can use a single parameter *slice rate* r to control the proportion of groups participated in the forward pass during inference. We empirically share the slice rate among all layers in the network; thus the computational resources required can be regulated precisely by the *slice rate*.

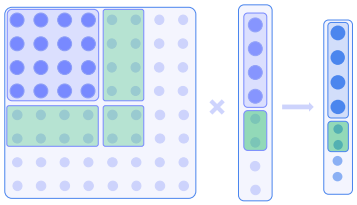


Figure 1: *Model slicing*: slice a sub-layer that is composed of preceding groups of the full layer controlled by the *slice rate* r during each forward pass. Only the activated parameters and groups of the current layer are required in memory and participate in computation. We illustrate a dense layer with *slice rate* $r = 0.5$ (activated groups highlighted in blue) and $r = 0.75$ (additional groups involved in green).

The *slice rate* is structurally the same concept as *width multiplier* [20] which controls the width of the network. However, instead of training only one fixed narrower model as in [20], we train the network in a dynamic manner to enhance the representation capacity of all the subnets it subsumes. For each forward pass during training, as illustrated in Figure 1, we sample the *slice rate* from a distribution F predetermined in the *Slice Rate Scheduling Scheme*, and train the corresponding sub-layers. The main challenges of training one model that supports inference at different widths include: how to determine proper candidate subnets (i.e. scheduling the *slice rate*) for each training iteration; and more importantly, how to stabilize the scale of output for each component (i.e. neurons or channels) as the the number of input components varies. Independent to our work, *Slimmable Neural Network* [52] (*SlimmableNet*) also proposes to train a single network executable at different widths. In [52], candidate subnets are considered to be equally important during training, by *statically* scheduling *all* subnets for every training pass and incorporating a set of batch normalization [26] (BN) layers into each layer, one for each candidate sub-layer, to address the output scale instability issue. In contrast, we consider the importance of the subnets to be different in *model slicing* (e.g., the full and the base network are the two most important subnets), and propose to dynamically schedule the training accordingly; besides the multi-BN solution, we further propose a more efficient solution with the group normalization[50] layer (GN) to prevent the scale instability, which works in accordance with the dynamic group-wise training and engenders the *group residual representation*. We shall provide more discussions on Section 3.

The *model slicing* training scheme can be scrutinized under the perspective of residual learning [16, 17] and knowledge distillation [18]. Under the random training process of *model slicing*, *groups* of each layer need to build up the representation increasingly, where the preceding groups carry the most fundamental information and the following groups

[3]https://en.wikipedia.org/wiki/Singles%27_Day

the residual representation relatively. Structurally, the final learned network is an ensemble of G subnets, with G being the number of groups, each corresponds to one *slice rate*. The parameters of these subnets are tied together and during each forward training pass, one subnet uniquely indexed by the slice rate is selected and trained. We conjecture that the accuracy of the resulting full trained network should be comparable to the network trained conventionally. Meanwhile, smaller subnets gradually distill knowledge from larger subnets as the training progresses, and thus can achieve comparable or even higher accuracy than their counterparts individually trained. Consequently, we can provide the same functionality of an ensemble of models with only one model by width slicing.

The proposed training scheme has many advantages over existing methods on various issues such as model compression, model cascade and anytime prediction. First, *model slicing* is readily applicable to existing neural networks, requiring no iterative retraining or dedicated library/hardware support as compared with most compression methods [15, 35]. Second, instead of training a set of models and optimize the scheduling of these models with different accuracy-efficiency trade-offs as is in conventional model cascade [27, 47], *model slicing* provides the same functionality of producing an approximate low-cost prediction with one single model. Third, the structure of the model trained with *model slicing* naturally supports applications where the model is required to give prediction within a given computational budget dynamically, e.g., anytime prediction [22, 21].

Our main technical contributions are:

- We develop a general training and inference framework *model slicing* that enables deep neural network models to support complex analytics with the trade-off between accuracy and inference cost/resource constraints on a per-input basis.
- We formally introduce the group residual learning of *model slicing* to general neural network models and further convolutional and recurrent neural networks. We also study the training details of *model slicing* and their impact in depth.
- We empirically validate through extensive experiments that neural networks trained with *model slicing* can achieve performance comparable to an ensemble of networks with one single model and support fluctuating workload with up to 16x volatility. Example applications are also provided to illustrate the usability of *model slicing*. The code is available at GitHub [4], which has been included in [38].

The rest of the paper is organized as follows. Section 2 provides a literature survey of related works. Section 3 introduces *model slicing* and how it can be applied to various deep learning models, including Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs) and etc. We then show how *model slicing* can support fine-grained system degradation management for present industrial deep learning services and we also provide an illustrating application of cascade ranking in Section 4. Experimental evaluations of *model slicing* are given in Section 5, under prevailing natural language processing and computer vision tasks on public benchmark datasets. Visualizations and detailed discussions of the results are also provided. Section 6 concludes the paper and points out some further research directions.

[4]https://github.com/ooibc88/modelslicing

2. RELATED WORK

2.1 Resource-aware Model Optimization

Many recent works directly devise networks [22, 48, 2] that are more economical in producing predictions. SkipNet [48] incorporates reinforcement learning into the network design, which guides the gating module whether to bypass the current layer for each residual block. SkipNet can provide predictions more efficiently yet in a less controlled manner inherently. In MoE [41], a gating network is introduced to select a smaller number of networks out a mixture-of-experts which consists of up to thousands of networks during inference for each sample. This kind of model ensemble approach aims to scale up the model capacity without introducing much overhead, while our approach enables every single model trained to scale down and support elastic inference cost.

MSDNet [22] supports classification with computational resource budgets at test time by inserting multiple classifiers into a 2D multi-scale version of DenseNet [23]. By early-exit into a classifier, MSDNet can provide predictions within given computation constraints. ANNs [21] adopts a similar design strategy of introducing auxiliary classifiers with Adaptive Loss Balancing, which supports the trade-off between accuracy and computational cost by using the intermediate features. [36] also develops a model that can successively improve prediction quality with each iteration but this approach is specific to segmenting videos with RNN models. These methods can largely alleviate the computational efficiency problem. However, they are highly specialized networks, which restrict their applicability. Functionally, models trained with *model slicing* also reuse intermediate features and support progressive prediction but with width slicing. *Model slicing* works similarly to these networks yet is more efficient, flexible and general.

2.2 Model Compression

Reducing the model size and computational cost has become a central problem in the deployment of deep learning solutions in real-world applications. Many works have been proposed to resolve the challenges of growing network size and surging resource expenditure incurred, mainly memory and computation. The mainstream solutions are to compress networks into smaller ones, including low-rank approximation [12], network quantization [10, 14, 15], weight pruning [15, 14], network sparsification on different level of structure [49, 35] etc.

To this end, many model compression approaches attempt to reduce the model size on the trained networks. [12] reduces model redundancy with tensor decomposition on the weight matrix. [10] and [15] instead propose to quantize the network weights to save storage space. HashNet [7] also proposes to hash network weights into different groups and sharing weight values within each group. These techniques are effective in reducing model size. For instance, [15] achieves up to 35x to 49x compression rates on AlexNet [30]. Although a considerable amount of storage can be saved, these techniques can hardly reduce run-time memory or inference

time, and they typically need a dedicated library and/or hardware support.

Many studies propose to prune weights, filters or channels in the networks. These approaches are generally effective because typically, deep networks are highly redundant in model representation. [14, 15] iteratively prune unimportant connections of small weights in trained neural networks. [44] further guides the sparsification of neural networks during training by explicitly imposing sparse constraints over each weight with a gating variable. The resulting networks are highly sparse, which can be stored compactly in a sparse format. However, the speedup of inference time of these methods depend heavily on dedicated sparse matrix operation libraries or hardware, and the saving of run-time memory is again very limited since most of the memory consumption comes from the activation maps instead of these weights. [49, 35] reduce the model size more radically by imposing regularization on the channel or filter and then prune the unimportant components. Like *model slicing*, channel and filter level sparsity can reduce the model size, run-time memory footprint and also lower the number of computational operations. However, these methods often require iterative fine-tuning to regain performance and support no inference time control.

2.3 Efficient Model Design

Instead of compressing existing large neural networks during or after training, recent works have also been exploring more efficient network design. ResNet [16, 17] proposes residual learning via an identity mapping shortcut and the efficient bottleneck structure, which enables the training of very deep networks without introducing more parameters. [45] shows that ResNet behaves like an ensemble of shallow networks and it can still function normally with a certain fraction of layers being removed. FractalNet [32] contains a series of the duplication of the fractal architecture with interacting subpaths. FractalNet adopts drop-path training which randomly selects certain paths during training, allowing for the extraction of fixed-depth subnetworks after training without significant performance loss. To some extent, these network architectures can support on-demand workload by slicing subnets layer-wise or path-wise. However, these methods are not generally applicable to other networks and the accuracy significantly drops when shortening or narrowing the network.

Many recent works focus on designing lightweight networks. SqueezeNet [25] reduces parameters and computation with the fire module. MobileNet [20] and Xception [9] utilize depth-wise and point-wise convolution for more parameter efficient convolutional networks. ShuffleNet [56] proposes point-wise group convolution with channel shuffle to help the information flowing across channels. These architectures scrutinize the bottleneck in conventional convolutional neural networks and search for more efficient transformation, reducing the model size and computation greatly.

3. MODEL SLICING

We aim to provide a general training scheme for neural networks to support on-demand workload with elastic inference cost. More specifically, the target is to enable the neural network to produce prediction within prescribed computational resources budget for each input instance, and meanwhile maintain the accuracy.

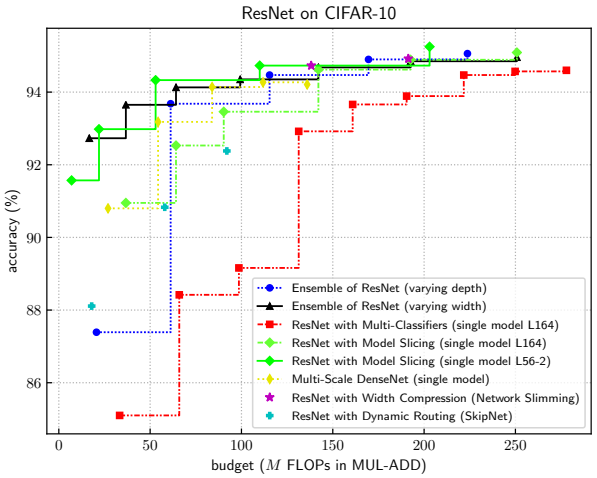


Figure 2: Classification accuracy w.r.t. inference FLOPs of ResNet trained with *model slicing* against ensemble, compression and other baselines on the CIFAR-10 dataset.

Existing methods of model compression, model ensemble and anytime prediction models can partially address this problem, but each has its limitations. Model compression methods such as *network slimming* [35] which compresses channel width each layer, produce efficient models while they typically take longer training time for iterative pruning and retraining, and more importantly, have no control over resources required during inference. Model ensemble methods, e.g., the ensemble of varying depth or width networks, support inference time resources control by scheduling the model for the immediate prediction task. However, deploying an ensemble of the models multiply the amount of disk storage and memory consumption; further, scheduling of these models is a non-trivial task to the system in deployment. Many works [22, 21, 36] instead exploit intermediate features for faster approximate prediction. For instance, Multi-Scale DenseNet [22] (MSDNet) inserts multiple classifiers into the model and thus supports anytime prediction by early-exit on a classifier.

Our *model slicing* also exploits and reuses intermediate features produced by the model while sidesteps the aforementioned problems. The key idea is to develop a general training and inference mechanism called *model slicing* which slices a narrower subnet for faster computation. With *model slicing*, neural networks are able to dynamically control the width of the subnet and thus regulate the computational resource consumption with one single parameter *slice rate*. In Figure 2, we illustrate by comparing the accuracy-efficiency trade-offs of ResNet trained with different approaches. We can observe that model ensemble methods are strong baselines which trade off accuracy for lower inference cost and that the Ensemble of ResNet with varying width performs better than varying depth. This finding indicates the superiority of width slicing over depth slicing, which is corroborated by the rapid loss in accuracy of ResNet with Multi-Classifiers (single model) in Figure 2. We will show that trained with *model slicing*, one single model is able to provide inference performance comparable to the ensemble of varying width networks. Therefore, *model slicing* is an ideal solution for neural networks to support elastic inference cost and resource constraints.

3.1 Model Slicing for Neural Networks

We start by introducing *model slicing* to fully-connected layer (dense layer) for general neural networks. Each dense layer in the neural network transforms via a weight matrix $\mathbf{W} \in \mathbb{R}^{N \times M}$: $\mathbf{y} = \mathbf{W}\mathbf{x}$, where $\mathbf{x} = [x_1, x_2, \ldots, x_M]$, a M-dimension input vector, corresponds to M input neurons and $\mathbf{y} = [y_1, y_2, \ldots, y_N]$, N output neurons correspondingly. Details such as the bias and non-linearity are omitted here for brevity. As illustrated in Figure 1, a gating variable is *implicitly* introduced to impose a structural constraint on each input neuron x_j:

$$y_i = \sum_{j=1}^{M} w_{ij}(\alpha_j \cdot x_j) \tag{1}$$

Each gating variable α_j thus controls the participation of the corresponding neuron x_j in each forward pass during both training and inference. Formally, the structural constraint is obtained by imposing partial ordered relation on these gating variables:

$$\forall i \forall j (i < j \wedge \alpha_j = 1 \rightarrow \alpha_i = 1) \tag{2}$$

which requires that the set of activated neurons during each forward pass forms a contiguous block starting from the *first* neuron. Based on the relation, we further divide these neurons into G ordered groups, i.e. $\mathbf{x} = [\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_G}]$, each group corresponds to a contiguous block of neurons. We denote the index of the rightmost neuron of the *first* i *groups* as g_i, and the corresponding sub-layer as Sub-layer-r_i, where the *slice rate* $r_i = \frac{g_i}{M}, (0 < r_i \leq 1)$. Then the set of groups participated in the current forward pass can be determined by indexing the rightmost group $\mathbf{x_i}$, and the set of neurons involved corresponds to $\{x_1, x_2, \ldots, x_{g_i}\}$. Note that the group number G is a pre-defined hyper-parameter, which could be set from 1 (the original layer) to M (each component forms a group).

Empirically, the *slice rate* is shared among all the layers in the network and we denote the subnet of first i groups in each layer as Subnet-r_i. Thus the width of the whole network can be regulated by the single parameter r. As illustrated in Figure 1, only the sliced part of the weight matrix and components are activated and required to reside in memory for inference in the current forward pass. We denote the computational operation required by the full network as C_0, then the computational operation required by the subnet of *slice rate* r is roughly $r^2 \times C_0$. Therefore, the run-time computational resources limit C_t can be dynamically satisfied by restricting *slice rate* r by:

$$r \leq min(\sqrt{\frac{C_t}{C_0}}, 1) \tag{3}$$

Consequently, a subnet can be readily sliced and deployed out of the network trained with *model slicing* whose disk storage and run-time memory consumption are also roughly quadratic to the *slice rate* r. Besides satisfying the run-time computational constraint, another primary concern is how to maintain the performance of these subnets. To this end, we propose the *model slicing* training in Algorithm 1. For each training pass, a list of slice rate $\mathbf{L}_t$ is sampled from the predefined slice rate list $\mathbf{L}$ by a scheduling scheme $\mathcal{F}$, and the corresponding subnets are optimized under the current training batch. We shall elaborate on the scheduling scheme in Section 3.4.

Algorithm 1: Training with Model Slicing.

Input: model $\mathbf{W}_0$, slice rate list $\mathbf{L}$, scheduling scheme $\mathcal{F}$, training iteration T, *criterion*, *optimizer*.
Upgrade layers to support *model slicing*:
$\mathbf{W}_0 \leftarrow upgrade\_model(\mathbf{W}_0, \mathbf{L})$
for *iteration* t **from** 0 **to** $T-1$ **do**
  Generate next batch of data and label: $(\mathbf{x}_t, \mathbf{y}_t)$
  Generate the current training slice rate list:
  $\mathbf{L}_t \leftarrow next\_slice\_rate\_batch(\mathbf{L}, \mathcal{F})$
  Initialize model gradient $\mathbf{W}_g \leftarrow 0$
  for *slice rate* $r \in \mathbf{L}_t$ **do**
    Forward Subnet-r: $\hat{\mathbf{y}} \leftarrow forward(\mathbf{W}_t, r, \mathbf{x}_t)$
    Compute Loss: $loss \leftarrow criterion(\mathbf{y}_t, \hat{\mathbf{y}})$
    Accumulate gradient:
    $\mathbf{W}_g \leftarrow \mathbf{W}_g + loss.backward()$
  end
  Update model
  $\mathbf{W}_{t+1} \leftarrow optimizer.update(\mathbf{W}_t, \mathbf{W}_g)$
end

Notice that the parameters of all subnets are tied together and any subnet indexed by a slice rate r_i subsumes all smaller subnets. The structural constraint of model slicing is reminiscent of residual learning [16, 17], where the Subnet-r_1 (the base network) carries the base representation. With the new input group $\mathbf{x_i}$ introduced as i grows, each y_j is optimized to learn from finer input details and thus the *group residual presentation*. We shall provide more discussions on this effect in Section 3.5. From the viewpoint of knowledge distillation [18], the Subnet-r_G (Subnet-1.0) maintains the capacity of the full model and as the training progresses, each Subnet-r_i gradually distills the representation from larger subnets and transfers the knowledge to smaller ones. Under this training scheme, we conjecture that the full network can maintain the accuracy, or possibly improve due to the regularization and ensemble effect; and in the meantime, the subnets can gradually pick up the performance by distilling knowledge from larger subnets.

3.2 Convolutional Neural Networks

Model slicing is readily applicable to convolutional neural networks in a similar manner. The most fundamental operation in CNNs comes from the convolutional layer which can be constructed to represent any given transformation $\mathcal{F}_{conv} : \mathbf{X} \rightarrow \mathbf{Y}$, where $\mathbf{X} \in \mathbb{R}^{M \times W_{in} \times H_{in}}$ is the input with M channels of size $W_{in} \times H_{in}$, $\mathbf{Y} \in \mathbb{R}^{N \times W_{out} \times H_{out}}$ the output likewise. Denoting $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_M]$ and $\mathbf{Y} = [\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_N]$ in vector of channels, the parameter set associated with each convolutional layer is a set of filter kernels $\mathbf{K} = [\mathbf{k}_1, \mathbf{k}_2, \ldots, \mathbf{k}_N]$. In a way similar to the dense layer, *model slicing* for the convolutional layer can be represented as:

$$\mathbf{y}_i = \mathbf{k}_i * \mathbf{X} = \sum_{j=1}^{M} \mathbf{k}_i^j * (\alpha_j \cdot \mathbf{x}_j) \tag{4}$$

where $*$ denotes convolution operation, $\mathbf{k}_i^j$ is a 2D spatial kernel associated with i_{th} output channel $\mathbf{y}_i$ and convolves on j_{th} input channel $\mathbf{x}_j$. Consequently, treating channels in

convolutional layers analogously to neurons in dense layers, *model slicing* can be directly applied to CNNs with the same training scheme.

Nonetheless, the output scale instability issue arises when applying *model slicing* to CNNs. Specifically, each convolutional layer is typically coupled with a batch normalization layer [26] to normalize outputs in the batch dimension, which stabilizes the mean and variance of input channels received by channels in the next layer. In the implementation of Equation 5, each batch-norm layer normalizes outputs with the batch mean μ and variance σ^2 and keeps records of running estimates of them which will be used directly after training. Here, γ and β are learnable affine transformation parameters of this batch-norm layer associated with each channel. However, with model slicing, the number of inputs received by a given output channel is no longer fixed, which is instead determined by the slice rate r_i during each forward pass. Consequently, the mean and variance of the batch-norm layer on the output fluctuate drastically; thus *one single set* of the running estimates is unable to stabilize the distribution of the output channel.

$$\hat{\mathbf{y}} = \frac{\mathbf{y}_{in} - \mu}{\sqrt{\sigma^2 + \epsilon}}; \mathbf{y}_{out} = \gamma\hat{\mathbf{y}} + \beta \tag{5}$$

We propose to address this issue with Group Normalization [50], an adaptation to Batch-norm. Group-norm divides channels into groups and normalizes channels in the same way as is in Equation 5 with the only difference that the mean and variance are calculated dynamically within each group. Formally, given the total number of groups G , the mean μ_i and variance σ_i^2 of i-th group are estimated within the set of channels in Equation 6 and shared among all the channels in the i-th group for normalization.

$$\mathcal{S}_i = \{\mathbf{x}_j | floor(\frac{j-1}{G}) = i\} \tag{6}$$

Group-norm normalizes channels group-wise instead of batch-wise, avoiding running estimates of the batch mean and variance in batch-norm whose error increases rapidly as the batch size decreases. Experiments in [50], which is also validated by our experiments on various network architectures, show that the accuracy of group-norm is relatively stable with respect to the batch size and group number. Besides stabling the scale, another benefit of group-norm is that it engenders the group-wise representation, which is in line with the *group residual learning* effect of model slicing training. To introduce model slicing to CNNs, we only need to replace batch-norm with group-norm and slice the normalization layers together with convolutional layers at the granularity of the group.

3.3 Recurrent Neural Networks

Model slicing can be readily applied to recurrent layers similarly to fully-connected layers. Take the vanilla recurrent layer expressed in Equation 7 for demonstration, the difference is that the output $\mathbf{h}_t$ is computed from two sets of inputs, namely $\mathbf{x}_t$ and $\mathbf{h}_{t-1}$.

$$\mathbf{h}_t = \sigma(\mathbf{W}_{hx}\mathbf{x}_t + \mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{b}_h) \tag{7}$$

Consequently, we can slice each input of the recurrent layer separately and adopt the same training scheme as fully-connected layers. Model slicing for recurrent layers of RNN variants such as GRU [8] and LSTM[19] works similarly. Dynamic slicing is applied to all input and output sets, including hidden/memory states and various gates, regulated by one single parameter *slice rate* r of each layer.

3.4 Slice Rate Scheduling Scheme

As shown in Algorithm 1, for each training pass of model slicing, a list of *slice rate* is sampled from a predetermined scheduling scheme $\mathcal{F}$, and then the corresponding subnets are trained under the current training batch. Formally, the random scheduling can be described as sampling the *slice rate* r from a Distribution $\mathcal{F}$. Denoting the list of valid *slice rate* r in order as $(r_1, r_2, \ldots, r_G)$, then we have:

$$\begin{cases} p(r_1) = F(\frac{r_1+r_2}{2}) = \int_{-\infty}^{\frac{r_1+r_2}{2}} f(r)dr, & i = 1 \\ p(r_i) = F(\frac{r_i+r_{i+1}}{2}) - F(\frac{r_{i-1}+r_i}{2}) = \int_{\frac{r_{i-1}+r_i}{2}}^{\frac{r_i+r_{i+1}}{2}} f(r)dr, & 1 < i < G \\ p(r_G) = 1 - F(\frac{r_{G-1}+r_G}{2}) = \int_{\frac{r_{G-1}+r_G}{2}}^{+\infty} f(r)dr, & i = G \end{cases} \tag{8}$$

where $f(r)$ is the probability density function, $F(r)$ the cumulative distribution function of $\mathcal{F}$ and $p(r_i)$ the probability of *slice rate* r_i being sampled. Thereby, the random scheduling $\mathcal{F}$ (e.g., the Uniform Distribution or the Normal Distribution) can be parameterized with a Categorical Distribution $Cat(G, p(r_1), p(r_2), \ldots, p(r_G))$, where each $p(r_i)$ denotes the relative importance of Subnet-r_i over other subnets. Further, the importance of these subnets should be treated differently. In particular, the full and the base network (i.e. Subnet-r_G and Subnet-r_1) should be the two most important subnets, because the full network represents the model capacity and the base network forms the basis for all the subnets. Based on this observation, we propose three categories of scheduling schemes:

- *Random scheduling*, where *each* of the slice rate is sampled from an $\mathcal{F}$ parameterized by $(p(r_1), \ldots, p(r_G))$.
- *Static scheduling*, where *all* valid slice rates are scheduled for the current training pass.
- *Random static scheduling*, where both a *fixed* set and a set of randomly sampled slice rates are scheduled.

For *random scheduling*, the importance of different subnets can be represented in the assigned probabilities, where we can assign higher sampling probabilities to more important subnets (e.g., the full and base network) during training. Likewise, for *random static scheduling*, we can include the important subnets in the fixed set and meanwhile assign proper probabilities to the remaining subnets. We shall evaluate these *slice rate* scheduling schemes in Section 5.1.2.

3.5 Group Residual Learning of Model Slicing

The *model slicing* training scheme structurally is reminiscent of residual learning proposed in ResNet [16, 17]. In ResNet, a shortcut connection of identity mapping is proposed to forward input to output directly: $\mathbf{y} = \mathbf{x} + \mathcal{F}_{conv}(\mathbf{x})$, where during optimization, the convolutional transformation only needs to learn the residual representation on top of input information $\mathbf{x}$, namely $\mathbf{y} - \mathbf{x}$. Analogously, networks trained with model slicing learn to accumulate the representation with additional groups introduced (group of neurons in dense layers and group of channels in convolutional layers), i.e. $\mathbf{y} = \sum_{i=1}^{G} \mathcal{F}_{conv\_i}(\mathbf{x_i})$.

To demonstrate the group residual learning effect in *model slicing*, we take the transformation in a fully-connected layer for example, and analyze the relationship between any two sub-layers of *slice rate* r_a and r_b with $r_a < r_b$. We have the transformation of Sub-layer-r_a as $\mathbf{y}_a = \mathbf{W}_a\mathbf{x}_a$ and the transformation of Sub-layer-r_b $[\tilde{\mathbf{y}}_a; \mathbf{y}_b] = \mathbf{W}_b[\mathbf{x}_a; \mathbf{x}_b]$ in block matrix multiplication as:

$$\begin{bmatrix}\tilde{\mathbf{y}}_a \\ \mathbf{y}_b\end{bmatrix} = \begin{bmatrix}\mathbf{W}_a & \mathbf{B} \\ \mathbf{C} & \mathbf{D}\end{bmatrix} \cdot \begin{bmatrix}\mathbf{x}_a \\ \mathbf{x}_b\end{bmatrix} = \begin{bmatrix}\mathbf{W}_a\mathbf{x}_a + \mathbf{B}\mathbf{x}_b \\ \mathbf{C}\mathbf{x}_a + \mathbf{D}\mathbf{x}_b\end{bmatrix} \quad (9)$$

Here, $\mathbf{x}_b$ is the supplementary input group introduced for Sub-layer-r_b and $\mathbf{y}_b$ is the corresponding output group. Generally $r_b - r_a \ll r_a$, then the *group residual representation learning* can be clarified from two angles. Firstly, the base representation of Sub-layer-r_b is $\tilde{\mathbf{y}}_a = \mathbf{W}_1\mathbf{x}_a + \mathbf{B}\mathbf{x}_b = \mathbf{y}_a + \mathbf{B}\mathbf{x}_b$, which is composed of the base representation $\mathbf{y}_a$ and the residual representation $\mathbf{B}\mathbf{x}_b$. Secondly, the newly-introduced output group $\mathbf{y}_b$ further forms the residual representation supplementary to the base representation $\tilde{\mathbf{y}}_a$. Higher model capacity is therefore expected of Subnet-r_b.

The justification for the *group residual learning effect* in *model slicing* is that as the training progresses, the base representation of $\mathbf{y}_a$ alone in Sub-layer-r_a has already been optimized for the learning task. Therefore, the supplementary group $\mathbf{y}_b$ introduced to Sub-layer-r_b gradually adapts to learn the residual representation, which is corroborated in the visualization in Section 5.5.1. Furthermore, this group residual learning characteristic provides an efficient way to harness the richer representation for Subnet-r_b based on Subnet-r_a by the simple approximation of $\tilde{\mathbf{y}}_a \approx \mathbf{y}_a$. With this approximation in every layer of the network, the most computationally heavy features of $\mathbf{W}_a\mathbf{x}_a$ could be reused without re-evaluating, thus the representation of Sub-layer-r_b can be updated by calculating only $\mathbf{C}\mathbf{x}_a + \mathbf{D}\mathbf{x}_b$ with a significantly lower computational cost.

We note that the *model slicing* training for *group residual representation* is applicable to the majority of neural networks. In addition, the *group residual learning* mechanism of *model slicing* is ideally suited for networks with layer transformation of multiple branches, e.g., group convolution [56], depth-wise convolution [20] and homogeneous multi-branch residual transformation of ResNeXt [51] etc.

4. EXAMPLE APPLICATIONS

In this section, we demonstrate how *model slicing* can benefit the deployment of deep learning based services. We use *model slicing* as the base framework to manage fine-grained system degradation for large scale machine learning services of dynamic workload. We also provide an example application of cascade ranking with *model slicing*.

4.1 Supporting Dynamic Workload Services

For a service with a dynamic workload, fine-grained system degradation management can be supported directly and efficiently with *model slicing*. Query samples come as a stream, and there is a dynamic latency constraint. Queries are usually batch-processed with vectorized computation for higher efficiency.

We design and implement an example solution to guarantee the latency and throughput requirement via *model slicing*. Given the processing time per sample for the full model t, to satisfy the dynamic latency constraint T and unknown query workload, we can build a mini-batch in every $T/2$ time, and utilize the rest $T/2$ time budget for processing: first examine the number of samples n in current batch, and choose the slice rate r satisfying $nr^2t \leq T/2$ (Equation 3) so that the processing time for this batch is within the budget $T/2$. Under such a system design, no computation resource is wasted as the total processing time per mini-batch is exactly the time interval of the batch input. Meanwhile, all samples can be processed within the required latency.

4.2 Implementing Cascade Ranking Application

Many information retrieval and data mining applications such as search and recommendation need to rank a large set of data items with respect to many user requests in an online manner. There are generally two issues in this process: 1). Effectiveness as how accurate the obtained results in the final ranked list are and whether there are a sufficient number of good results; and 2). Efficiency such as whether the results are obtained in a timely manner from the user perspective and whether the computational costs of ranking is low from the system perspective. For large-scale ranking applications, it is of vital importance to address both issues for providing good user experience and achieving a cost-saving solution.

Cascade ranking [46, 34] is a strategy designed for such a trade-off. It utilizes a sequence of prediction functions of different costs in different stages. It can thus eliminate irrelevant items (e.g., for a query) in earlier stages with simple features and models, while segregate more relevant items in later stages with more complicated features and models. In general, functions in early stages require low inference cost while functions in later stages require high accuracy.

One critical characteristic of cascade ranking is that the optimization target for each function may depend on all other functions in different stages [34]. For instance, given a positive item set $\{1, 2, ..., 7\}$ and we aim to build a cascade ranking solution with two stages, suppose that function in stage two mis-drop positive item $\{6, 7\}$, a function in stage one mis-drop $\{1, 6, 7\}$ is better than a function mis-drop $\{1, 2\}$, though the former has a higher error rate over the whole dataset (in the first case $\{2, 3, 4, 5\}$ are left while in the second case only $\{3, 4, 5\}$ are left). Lots of analysis are given in [46, 5, 34]. Therefore, we expect the prediction of positive items given by functions in different stages to be consistent so that the accumulated false negatives are minimized. Unfortunately, most implementations of the ranking/filtering function at each stage for cascade ranking use different model architectures with different parameters. The results of different models are thus unlikely to be consistent.

Model slicing would be an ideal solution for cascade ranking. Firstly, it provides the trade-off of model effectiveness and model efficiency with one single model. The ranking functions at different stages can be obtained by as simple as configuring the inference cost of the model. Secondly, as is corroborated in Section 5.5, the prediction results of *model slicing* sub-models are inherently correlated since the larger model is actually using the smaller model as the base of its model representation. We shall illustrate the effectiveness and efficiency of *model slicing* in comparison with the traditional model cascade solution in a cascade ranking simulation in Section 5.4.

5. EXPERIMENTS

We evaluate the performance of *model slicing* on state-of-the-art neural networks on two categories of public benchmark tasks, specifically evaluating *model slicing* for dense layers, i.e. fully-connected and recurrent layers on language modeling [37, 54, 39] in Section 5.2 and evaluating *model slicing* for convolutional layers on image classification [43, 16, 53] in Section 5.3. Experimental setups of *model slicing* are provided in Section 5.1; cascade ranking simulation of example applications and visualization on the *model slicing* training are given in Section 5.4 and Section 5.5 respectively.

5.1 Model Slicing Setup

5.1.1 General Setup and Baselines

The slice rate r_i corresponds to Subnet-r_i, which is restricted between a lower bound r_1 and 1.0. In the experiments, the networks trained with *model slicing* are evaluated with the slice rate list where r_i ranges from $r_1 = 0.25/0.375$ (corresponding to around 16x/7x the computational speedup) to 1.0 in every $\frac{1}{4}/\frac{1}{8}/\frac{1}{16}$ (the slice granularity). We apply *model slicing* to all the hidden layers except the input and output layers because both layers are necessary for the inference and further take a negligible amount of parameter and computation in the full network.

We compare *model slicing* primarily with two baselines. The first baseline is the full network trained without *model slicing* ($r_1 = 1.0$, *single model*), implemented by fixing r_1 to 1.0 during training. During inference, we slice the corresponding Sub-layer-r_i of each layer in the network for comparison. The second baseline is an ensemble of networks of varying width (*fixed models*). In addition to the above two baselines, we also compare model slicing with model compression (Network Slimming [35]), anytime prediction (multi-classifiers methods, e.g. MSDNet [22]) and efficient prediction (SkipNet [48]).

5.1.2 Slice Rate Scheduling Scheme

Table 1: Accuracy of VGG-13 trained with various training scheduling schemes on CIFAR-10. $|\mathcal{L}_t|$ denotes the number of slice rates scheduled for each training pass.

Scheme	Fixed	R-uniform-2	R-weighted-2	R-weighted-3	Static	R-min	R-max	R-min-max	Slimmable
$\|\mathcal{L}_t\|$	4	2	2	3	4	2	2	3	4
1.00	94.31	93.72	94.23	94.34	93.67	93.15	94.32	**94.35**	**94.41**
0.75	93.86	93.64	94.08	**94.20**	93.46	93.14	93.59	93.97	**94.29**
0.50	93.39	93.68	93.76	**93.92**	93.19	93.11	93.05	93.60	93.47
0.25	91.63	91.59	91.68	91.96	91.69	91.84	91.31	**92.10**	91.45

We evaluate the three slice rate scheduling schemes proposed in Section 3.4 with the slice rate list $(1.0, 0.75, 0.5, 0.25)$ in Table 1. Specifically, the baseline is the ensemble of fixed models (*fixed*). For *random scheduling*, we evaluate the uniform sampling (*R-uniform*) and the weighted random sampling (*R-weighted*, weight list $(0.5, 0.125, 0.125, 0.25)$); in particular, *R-uniform-k* and *R-weighted-k* denote random scheduling of k slice rates scheduled for each forward pass. For *static scheduling* (*Static*), the subnets are regarded as equally important and thus all slice rates are scheduled whose *computation* grows linearly with the number of subnets configured; For *random static scheduling*, we evaluate statically scheduling the base network (*R-min*), the full network (*R-max*) or both of these two subnet (*R-min-max*), and meanwhile uniformly sampling one remaining subnets. The detailed training settings are given in Section 5.3.2.

Table 1 shows that weighted sampling of *random scheduling* achieves higher accuracy than uniformly sampling with a comparable training budget; and training longer further improves the performance. In contrast, *static scheduling* performs consistently worse than the weighted *random scheduling* even though it takes more training rounds. The results corroborate our conjuncture that the base and the full network are of greater importance and thus should be scheduled more frequently during training.

We next evaluate the *random static scheduling*, which consists of statically scheduling the base and/or full network while uniformly sampling the remaining subnets. We observe that statically training the base (*R-min*) or the full (*R-max*) network helps to improve the corresponding subnets. Meanwhile, the performance of the neighboring subnets also improves, mainly due to the effect of knowledge distillation. We also compare *model slicing* with *SlimmableNet* [52] (*Slimmable*) that adopts *static scheduling* and multi-BN layers instead of one group-norm layer. The results shown in Table 1 reveal that *SlimmableNet* obtains higher accuracies in larger subnets, which may result from the longer training time; while smaller subnets perform worse than *model slicing* with *random scheduling*, e.g., *R-weighted* or *R-min-max*, mainly due to the lack of differentiation of varying importance of subnets in *static scheduling*. In the following experiments, we therefore evaluate *model slicing* with *R-weighted-3* for small datasets and *R-min-max* for larger datasets for reporting purpose.

5.1.3 The Lower Bound of Slice Rate

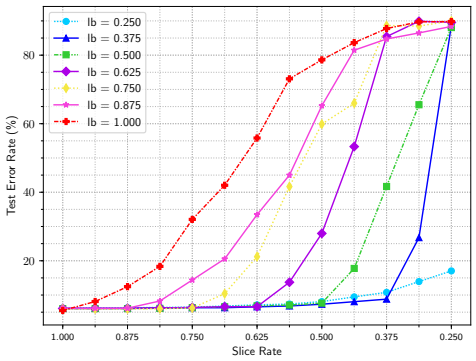


Figure 3: Illustration of the impact of the lower bound (*lb*) on VGG-13 trained with *model slicing* on CIFAR-10.

For each of the subnet, the computation resources required can be evaluated beforehand. The lower bound controls the width of the base network and thus should be set to Equation 3 under the computational resource limit. Figure 3 shows the accuracies of VGG-13 trained with different lower bounds. Empirically, the accuracy drops steadily as r_i decreases towards r_1 (the lower bound *lb*), and networks trained with different *lb*s perform rather close. Given a lower bound *lb*, however, the accuracy of the corresponding Subnet-*lb* is slightly higher than other Subnet-*lb*s, which is mainly because the base network is optimized more frequently. When the slice rate r_i decreases over the lower bound, the accuracy drops drastically. This phenomenon meets the expectation that further slicing the base network destroys the base representation, and thus the accuracy suffers significantly. The loss of accuracy is more severe for

convolutional neural networks, where the representation depends heavily on all channels of the base network. In the following experiments, we therefore evaluate lower bound 0.375/0.25 for small (e.g. CIFAR, PTB)/large (e.g. ImageNet) datasets respectively for reporting purpose, whose computational cost is roughly 14.1%/6.25% of the full network (i.e. 7.11x/16x speedup) and empirically can be adjusted readily according to the deployment requirement.

5.2 NNLM for Language Modeling

5.2.1 Language modeling task and dataset

The task of language modeling is to model the probability distribution over a sequence of words. Neural Network Language Modeling (NNLM) comprises both fully-connected and recurrent layers; we thus adopt NNLM to evaluate the effectiveness of *model slicing* for dense layers. NNLM [37, 54, 39] specifies the distribution over next word w_{t+1} given its preceding word sequence $w_{1:t} = [w_1, w_2, \ldots, w_t]$ with neural networks. Training of NNLM involves minimizing the negative log-likelihood (NLL) of the sequence: $NLL = -\sum_{t=1}^{T} logP(w_t|w_{1:t-1})$. Following the common practice for language modeling, we use perplexity (PPL) to report the performance: $PPL = exp(\frac{NLL}{T})$. We adopt the widely benchmarked English Penn Tree Bank (PTB) dataset and use the standard train/test/validation split by [37].

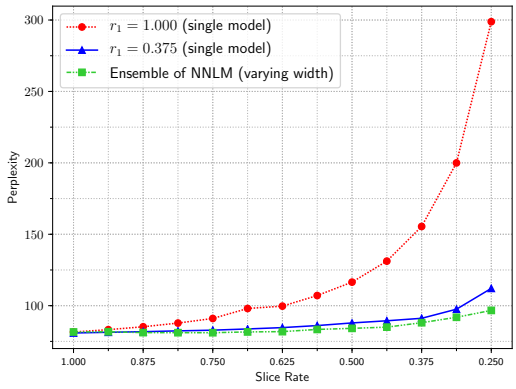


Figure 4: Results of NNLM trained w/o *model slicing*.

5.2.2 NNLM configuration and training details

Following [37, 54, 39], the NNLM model in the experiments consists of an input embedding layer, two consecutive LSTM layers, an output dense layer and finally a softmax layer. The embedding dimension is 650 and both LSTM layers contain 640 units. In addition, a dropout layer with dropout rate 0.5 follows the embedding and two LSTM layers. The models are trained by truncated backpropagation through time for 35 time steps, minimizing NLL during training without any regularization terms with SGD of batch size 20. The learning rate is initially set to 20 and quartered in the next epoch if the perplexity does not decrease on the validation set. *Model slicing* applies to both recurrent layers and the output dense layer with output rescaling.

5.2.3 Results of Model Slicing on NNLM

Results in Figure 4 and Table 2 show that *model slicing* is effective to support on-demand workload with one single model only at the cost of minimum performance loss. The performance of the network trained without *model slicing* decreases drastically. With *model slicing*, the performance decreases steadily and stays comparable to the corresponding fixed models. In particular, the performance of the subnet is slightly better than the corresponding fixed model when the *slice rate* is near 1.0. For instance, as is shown in Table 2, the perplexity is 80.89 for the Subnet-r_G (the full network) while 81.58 for the full fixed model.

Table 2: Remaining percentage of computation (C_t), perplexity of NNLM on PTB w.r.t. the slice rate.

Slice Rate **r**	1.000	0.875	0.750	0.625	0.500	0.375	0.250
$\mathbf{C_t}$	100.0	76.56	56.25	39.06	25.00	14.06	6.250
NNLM-1.0	81.58	85.23	91.04	99.68	116.5	155.5	298.8
NNLM-0.375	**80.89**	81.79	82.86	84.65	87.92	91.17	112.1
NNLM-fixed	81.58	**81.66**	**81.78**	**81.83**	**84.13**	**88.08**	96.69

This validates our hypothesis that the regularization and ensemble effect could improve the full model performance. Further, the student-teacher knowledge distillation effect of the *group residual learning* facilitates the learning process by transferring and sharing representation, and thus helps maintain the performance of subnets.

5.3 CNNs for Image Classification

In this subsection, we evaluate *model slicing* for convolutional layers on image classification tasks, mainly focusing on representative types of convolutional neural networks. We first introduce dataset statistics for the evaluation. Then configurations of the networks and training details are introduced. Finally, we discuss and compare with baselines the results of *model slicing* training scheme for CNNs.

5.3.1 Datasets

We evaluate the results on CIFAR [29] and ImageNet-12 [11] image classification datasets.

The CIFAR [29] datasets consist of 32×32 colors scenery images. CIFAR-10 consists of images drawn from 10 classes. The training and testing sets contain $50,000$ and $10,000$ images respectively. Following the standard data augmentation scheme [16, 24, 23], each image is first zero-padded with 4 pixels on each side, then randomly cropped to produce 32×32 images again, followed by a random horizontal flip. We normalize the data using the channel means and standard deviations for data pre-processing.

The ILSVRC 2012 image classification dataset contains 1.2 million images for training and another 50,000 for validation from 1000 classes. We adopt the same data augmentation scheme for training images following the convention[16, 53, 23], and apply a 224×224 center crop to images at test time. The results are reported on the validation set following common practice.

5.3.2 CNN Architectures and Training Details

Model slicing dynamically slices channels within each layer in CNNs; thus we adopt three representative architectures differing mainly in the channel width for evaluation. The first architecture is VGG [43] whose convolutional layer is a plain 3×3 *conv* of medium channel width. The second architecture is the pre-activation residual network [17] (ResNet). ResNet is composed of the bottleneck block [17], denoting as B-Block ($conv1 \times 1 - conv3 \times 3 - conv1 \times 1$). We evaluate *model slicing* on ResNet of varying depth and width,

Table 3: Configurations of representative convolutional neural networks on CIFAR (left panel) and ImageNet (right panel) datasets. Building blocks are denoted as "[block, number of channels] × number of blocks".

Group	Output Size	VGG-13	ResNet-164	ResNet-56-2	Output Size	VGG-16	ResNet-50
conv1	32×32	[conv3×3, 64]×2	[B-Block, 16]×1	[B-Block, 16]×1	112×112	[conv3×3, 64]×3	[B-Block, 64]×1
conv2	32×32	[conv3×3, 128]×2	[B-Block, 16]×18	[B-Block, 16×2]×6	56×56	[conv3×3, 128]×3	[B-Block, 64]×3
conv3	16×16	[conv3×3, 256]×2	[B-Block, 32]×18	[B-Block, 32×2]×6	28×28	[conv3×3, 256]×3	[B-Block, 128]×4
conv4	8×8	[conv3×3, 512]×4	[B-Block, 64]×18	[B-Block, 64×2]×6	14×14	[conv3×3, 512]×3	[B-Block, 256]×6
conv5	8×8	-	-	-	7×7	[conv3×3, 512]×3	[B-Block, 512]×3
avgPool/FC	10	[avg8×8, 512]	[avg8×8, 64×4]	[avg8×8, 64×2×4]	1000	[512×7×7,4096,4096]	[avg7×7,512×4]
Dataset	-	CIFAR	CIFAR	CIFAR	-	ImageNet-12	ImageNet-12
Params	-	9.42M	1.72M	2.35M	-	138.36M	25.56M

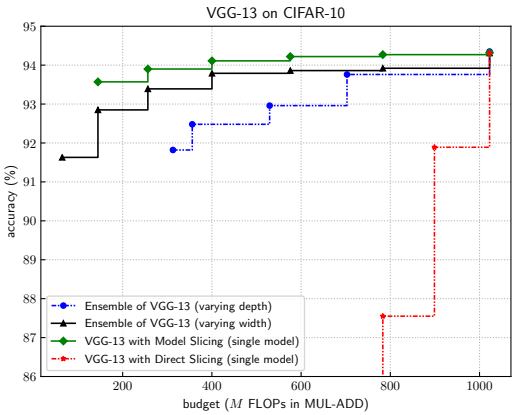


Figure 5: Classification accuracy w.r.t. inference FLOPs of VGG-13 trained with model slicing against other baselines on the CIFAR-10 dataset.

and denote the architecture adopted as ResNet-L, with L being the number of layers. The third architecture is Wide Residual Network [53], which is denoted as ResNet-L-k, with k being the widening factor of the channel width for each layer. Detailed configurations are summarized in Table 3.

To support *model slicing*, convolutional layers and the batch-norm layers are replaced with counterpart layers supporting *model slicing*. For both baseline and *model slicing* trained models, we train 300 epochs on CIFAR-10 with SGD of batch size 128 and initial learning rate 0.1, and 100 epochs on ImageNet-12 with SGD of batch size 128 and learning rate 0.01 with gradual warmup [16, 13]. The learning rate is divided by 10 at 50% and 75% of the total training epochs for CIFAR-10, and at 30%, 60% and 90% for ImageNet-12. Other training details follow the conventions [17, 53].

5.3.3 Results of Model Slicing on CNNs

Results of representative CNNs on CIFAR and ImageNet datasets are illustrated in Figure 2, Figure 5, and summarized in Table 4. In general, a CNN model trained with *model slicing* is able to produce prediction with elastic inference cost by dynamically scheduling a corresponding subnet whose accuracy is comparable to or even higher than its conventionally trained counterpart.

We compare the performance of *model slicing* with more baseline methods on ResNet in Figure 2. We can observe that ResNet-164 trained with *model slicing* (single model L164) achieves accuracies significantly higher than ResNet with Multi-Classifiers baseline, which confirms the superiority of *model slicing* over depth slicing. However, its performance is noticeably worse than the ensemble of ResNet

Table 4: Remaining estimated percentage of computation FLOPs (C_t)/parameter size (M_t), and accuracy of VGG-13, ResNet-164, ResNet-56-2 on CIFAR-10, and VGG-16, ResNet-50 on ImageNet w.r.t. the slice rate.

Slice Rate **r**	1.000	0.8750	0.7500	0.6250	0.500	0.375	0.2500
$\mathbf{C_t/M_t}$	100.0%	76.56%	56.25%	39.06%	25.00%	14.06%	6.25%
VGG-13-lb-1.0	94.31	87.55	67.93	44.18	21.37	12.23	10.19
VGG-13-fixed-models	94.31	93.92	93.86	93.79	93.39	92.85	91.63
VGG-13-lb-0.375	**94.32**	**94.27**	**94.22**	**94.11**	**93.90**	**93.57**	16.87
ResNet-164-lb-1.0	94.96	87.55	67.93	44.12	21.37	12.33	10.19
ResNet-164-fixed-models	94.96	94.85	**94.68**	**94.35**	**94.13**	**93.65**	92.73
ResNet-164-lb-0.375	**95.09**	**94.89**	94.62	93.46	92.53	90.95	16.83
ResNet-56-2-fixed-models	95.25	95.20	**95.17**	**95.01**	**94.52**	**94.04**	93.19
ResNet-56-2-lb-0.375	**95.37**	**95.25**	94.73	94.33	92.98	91.57	10.58
VGG-16-fixed-models	72.47	-	**70.73**	-	66.31	-	54.14
VGG-16-lb-0.25	**72.53**	-	70.69	-	**66.41**	-	**54.20**
ResNet-50-fixed-models	76.05	-	**74.73**	-	**72.02**	-	63.91
ResNet-50-lb-0.25	**76.08**	-	74.65	-	71.97	-	**63.98**

of varying width, especially in the lower budget prediction. This is mainly because the convolutional layer of ResNet-164 on CIFAR is narrow. In particular, the convolutional layer in conv1/conv2 comprises 16 channels (see Table 3) and thus with *slice rate* 0.375, only 6 channels remain for inference which leads to limited representational power. With twice the channel width, the single *model slicing* trained model ResNet-L56-2 achieves accuracies comparable to the strong ensemble baseline of varying depth/width, model width compression baseline *Network Slimming* [35], and achieves higher accuracies than SkipNet [48] in corresponding inference budgets and generally better accuracy-budget trade-offs than MSDNet [22]. This demonstrates that *model slicing* works more effectively for models of wider convolutional layers, e.g. the VGG-13, ResNet-L56-2 and ResNet-50. For instance, the accuracy is 93.57% for VGG-13-lb-0.375 with *slice rate* 0.375, which is 0.72% higher than its individually trained counterpart and takes around 14.06% of the computation of the full network (∼7.11x speedup). This is also confirmed in the wider network VGG-16 and ResNet-50 on the larger dataset ImageNet. Specifically, ResNet-50-lb-0.25 of *slice rate* 0.25 achieves slightly higher accuracy than the fixed model of the same width and takes only around 6.25% computation of the full network (∼16x speedup).

We can also notice in Figure 5, Table 4 that the accuracy of CNNs trained conventionally (lower bound lb=1.0) decreases drastically as more channel groups are sliced off. This shows that with conventional training, channel groups in the same convolutional layer are highly dependent on other groups in the representation learning such that slicing even one channel group off may impair the representation. With the *group residual representation learning* of *model slicing*, one single network can achieve accuracy comparable to the ensemble of networks of varying width with significantly less memory and computational operation.

5.4 Simulation of Cascade Ranking

We further simulate a cascade ranking scenario with six stages of classifiers. CIFAR-10 test dataset is adopted for illustration which contains ten types of items (classes) and 1000 items (images) for each type, and VGG-13 (see Table 3) is adopted as the baseline model. The classifier (model) is required to categorize each item into a type and then filter out all the items whose predicted category is not consistent with its previous type. Therefore, the cascade ranking pipeline will only keep items of consistent classification type in *all* the cascade models. Typically, the pipeline deploys smaller models in early stages to efficiently filter out irrelevant items, and larger but costlier models in subsequent stages for higher retrieval quality. The baseline solution is a cascade model of the baseline model of varying width, which is compared with the *model slicing* solution with corresponding sub-models sliced off the baseline model trained with *model slicing*. The parameter size and computation FLOPs of models at each stage are provided in Table 5.

Table 5: Simulation of cascade ranking with the cascade model and the model trained with *model slicing*. The *precision* shows the prediction accuracy of each classifier; the *aggregate recall* denotes the fraction of correctly classified items over the total number of items by each stage.

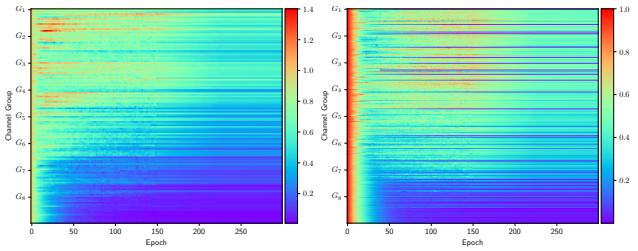
Stage/Classifier		1st	2nd	3rd	4th	5th	6th
Model Width (r)		0.375	0.500	0.625	0.750	0.875	1.000
Params (M)		1.33	2.36	3.68	5.30	7.21	9.42
FLOPs (M)		144.6	256.5	400.2	575.8	783.2	1022.5
Cascade Model	*precision*	92.85%	93.39%	93.79%	93.86%	93.92%	94.31%
	aggregate recall	92.85%	90.11%	88.62%	87.45%	86.70%	86.03%
Model Slicing	*precision*	93.57%	93.90%	94.11%	94.22%	94.27%	94.32%
	aggregate recall	93.57%	91.81%	89.47%	88.95%	88.76%	88.67%

Table 5 summarizes the results on *precision* and the *aggregate recall* of each stage. The results show two advantages of the *model slicing* solution over the conventional cascade model solution: firstly, in terms of effectiveness, the *model slicing* solution retrieves 88.67% correct items in total as compared with 86.03% of the conventional solution. The significantly higher *aggregate recall* is mainly because of the more consistent prediction between classifiers which we shall discuss and visualize in Section 5.5.3; secondly, in terms of efficiency, the conventional solution takes totally 29.3M parameters and 3182.8M FLOPs computation for the retrieval of each item, while *model slicing* solution only takes 9.42M parameters in one model and the computation could be greatly reduced with the computation reusage discussed in Section 3.5.

5.5 Visualization

5.5.1 Residual Learning Effect of Model Slicing

In CNNs trained with *model slicing*, each of the convolutional layers is followed by a group normalization layer to stabilize the scale of output with a scaling factor, i.e., γ in Equation 5. The scaling factor largely represents the importance of the corresponding channel. We therefore visualize the evolution of these scaling factors during *model slicing* training in Figure 6. Specifically, we take the first convolutional layers of conv3 and conv5 in VGG-13 (see Table 3), which corresponds to low and high level feature extractors. We can observe an obvious stratified pattern in Figure 6. Groups from G_1 to G_3 of the base network gradually learn scaling factors of the largest values. Meanwhile, from G_3 to G_8, the average scaling factor values gradually become smaller. This validates our assumption that *model slicing* training engenders *residual group learning*, where the base network learns the fundamental representation and following groups residually build up the representation.



(a) conv3 (b) conv5

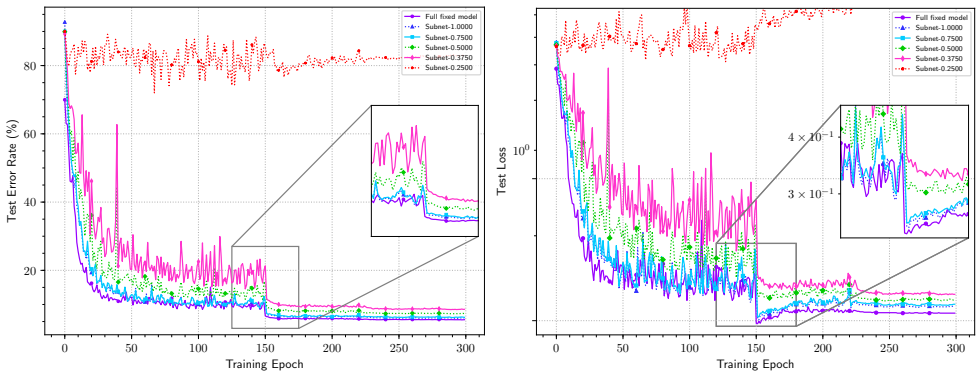
Figure 6: Visualization of channel scaling factors (γ from Equation 5) in scale as the training evolves, taken from the first convolutional layer of conv3, conv5 (Table 3) of VGG-13 trained on CIFAR-10 respectively. Brighter colors correspond to larger values.

5.5.2 Learning Curves of Model Slicing

Figure 7 illustrates learning curves of VGG-13 trained with *model slicing* compared with the full fixed model. Learning curves of the subnets of VGG-13 trained with *model slicing* reveal that the error rate drops faster in larger subnets and smaller subnets closely follow the larger subnets. This demonstrates the knowledge distillation effect, where larger subnets learn faster and gradually transfer the knowledge learned to smaller subnets. We notice that the final accuracy of subnets of a relatively larger slice rate approaches the full fixed model, which shows that the *model slicing* trained model can trade off accuracy for efficiency by inference with a smaller subnet with less memory and computation at the cost of a minor accuracy decrease.

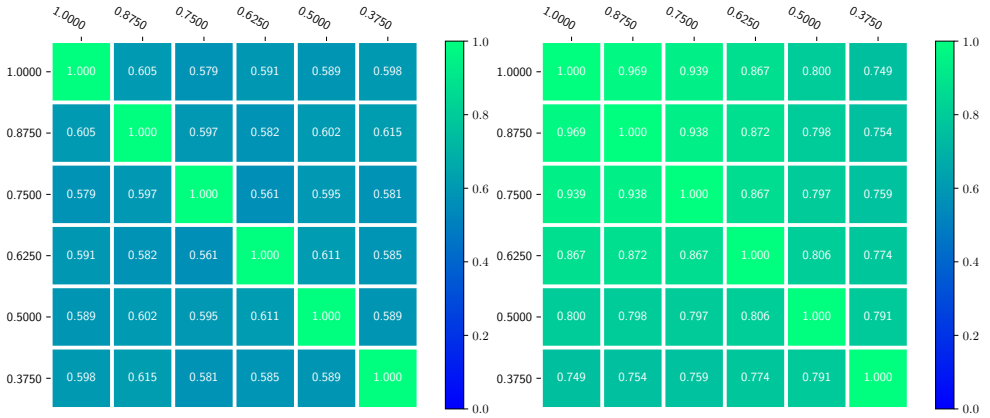
5.5.3 Prediction Consistency of Model Slicing

We also evaluate the consistency of prediction results between the subnets of the model trained with *model slicing*. Typically, the outputs are not the same for different models trained conventionally. However, trained with *model slicing*, the model of a larger slice rate incorporates models of lower slice rate as part of its representation. Consequently, the subnets sliced off the *model slicing* model are expected to produce similar predictions, and larger subnets could be able to correct wrong predictions of smaller models. Figure 8 shows the *inclusion coefficient* of wrongly predicted samples between each pair of models. The inclusion coefficient measures the fraction of the wrongly predicted samples of the larger model over those of the smaller model. It essentially measures the ratio of error overlapped between two models. Unsurprisingly, the prediction results of *model slicing* training is much more consistent than that of training different fixed models separately. Therefore, *model slicing* may not be ideal for applications such as model ensemble which typically requires diversity, but could be extremely useful for applications requiring consistent prediction such as cascade ranking where the accumulated error is expected to be minimized.



(a) Test Error (b) Loss

Figure 7: Test Error Rate and Loss curves of VGG-13 full fixed model and VGG-13 trained with *model slicing* ($r_1 = 0.375$) validated under different *slice rates* on CIFAR-10 dataset.



(a) fixed models (b) subnets of the model trained with *model slicing*

Figure 8: Heatmap of the *inclusion coefficient* of wrongly predicted samples between each pair of VGG-13 fixed models and sliced subnets of VGG-13 trained with *model slicing* ($r_1 = 0.375$) respectively on CIFAR-10 dataset.

6. CONCLUSIONS

Relatively few efforts have been devoted to neural networks dynamically providing predictions within memory and computational operation budget. In this paper, we propose *model slicing*, a general training framework supporting elastic inference cost for neural networks. The key idea of *model slicing* is to impose a structural constraint on basic components of each layer both during training and inference, and then regulate the width of the network with a single parameter *slice rate* during inference given the resource budget on a per-input basis. We have provided detailed analysis and discussion on training details of *model slicing* and evaluated *model slicing* through extensive experiments.

Results on NLP and vision tasks show that neural networks trained with *model slicing* can effectively support on-demand workload by slicing a subnet from the trained network dynamically. With *model slicing*, neural networks can achieve significant reduction of run-time memory and computation with comparable performance, e.g., 16x speedup with *slice rate* 0.25. Unlike conventional model compression methods where the computation reduction is limited, the required computation decreases quadratically to *slice rate*.

Model slicing also sheds light on the learning process of neural networks. Networks trained with *model slicing* engender *group residual learning* in each layer, where components in the base network learn the fundamental representation while the following groups build up the representation residually. Meanwhile, the learning process is reminiscent of knowledge distillation. During training, larger subnets learn faster and gradually transfer the representation to smaller subnets. Finally, *model slicing* is readily applicable to the model compression scenario by deploying a proper subnet.

7. ACKNOWLEDGMENTS

This research is supported by the National Research Foundation Singapore under its AI Singapore Programme [Award No. AISG-GC-2019-002] and Singapore Ministry of Education Academic Research Fund Tier 3 under MOE's official grant number MOE2017-T3-1-007.

8. REFERENCES

[1] M. Boehm, M. W. Dusenberry, D. Eriksson, A. V. Evfimievski, F. M. Manshadi, N. Pansare, B. Reinwald, F. R. Reiss, P. Sen, A. C. Surve, et al. Systemml: Declarative machine learning on spark. *PVLDB*, 9(13):1425–1436, 2016.

[2] S. Cai, Y. Shu, W. Wang, and B. C. Ooi. Isbnet: Instance-aware selective branching network. *arXiv preprint arXiv:1905.04849*, 2019.

[3] W. Cao, Y. Gao, B. Lin, X. Feng, Y. Xie, X. Lou, and P. Wang. Tcprt: Instrument and diagnostic analysis system for service quality of cloud databases at massive scale in real-time. In *Proceedings of the 2018 International Conference on Management of Data*, pages 615–627. ACM, 2018.

[4] S. Chaudhuri, B. Ding, and S. Kandula. Approximate query processing: No silver bullet. In *Proceedings of the 2017 ACM SIGMOD International Conference on Management of Data*, pages 511–519. ACM, 2017.

[5] R.-C. Chen, L. Gallagher, R. Blanco, and J. S. Culpepper. Efficient cost-aware cascade ranking in multi-stage retrieval. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 445–454. ACM, 2017.

[6] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794. ACM, 2016.

[7] W. Chen, J. Wilson, S. Tyree, K. Weinberger, and Y. Chen. Compressing neural networks with the hashing trick. In *International Conference on Machine Learning*, pages 2285–2294, 2015.

[8] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

[9] F. Chollet. Xception: Deep learning with depthwise separable convolutions. *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1610–02357, 2017.

[10] 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.

[11] 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*, pages 248–255. Ieee, 2009.

[12] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in neural information processing systems*, pages 1269–1277, 2014.

[13] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

[14] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[15] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143, 2015.

[16] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[17] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016.

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

[19] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[20] 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.

[21] H. Hu, D. Dey, M. Hebert, and J. A. Bagnell. Learning anytime predictions in neural networks via adaptive loss balancing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3812–3821, 2019.

[22] G. Huang, D. Chen, T. Li, F. Wu, L. Van Der Maaten, and K. Q. Weinberger. Multi-scale dense convolutional networks for efficient prediction. *arXiv preprint arXiv:1703.09844*, 2, 2017.

[23] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[24] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger. Deep networks with stochastic depth. In *European Conference on Computer Vision*, pages 646–661. Springer, 2016.

[25] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[26] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[27] D. Kang, J. Emmons, F. Abuzaid, P. Bailis, and M. Zaharia. Noscope: optimizing neural network queries over video at scale. *PVLDB*, 10(11):1586–1597, 2017.

[28] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu. Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems*, pages 3146–3154, 2017.

[29] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[30] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[31] A. Kumar, R. McCann, J. Naughton, and J. M. Patel. Model selection management systems: The next frontier of advanced analytics. *ACM SIGMOD Record*, 44(4):17–22, 2016.

[32] G. Larsson, M. Maire, and G. Shakhnarovich. Fractalnet: Ultra-deep neural networks without residuals. *arXiv preprint arXiv:1605.07648*, 2016.

[33] F. Li, B. Wu, K. Yi, and Z. Zhao. Wander join: Online aggregation via random walks. In *Proceedings of the 2016 International Conference on Management of Data*, pages 615–629. ACM, 2016.

[34] S. Liu, F. Xiao, W. Ou, and L. Si. Cascade ranking for operational e-commerce search. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1557–1565. ACM, 2017.

[35] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang. Learning efficient convolutional networks through network slimming. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, pages 2755–2763. IEEE, 2017.

[36] L. McIntosh, N. Maheswaranathan, D. Sussillo, and J. Shlens. Recurrent segmentation for variable computational budgets. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 1648–1657, 2018.

[37] T. Mikolov, M. Karafiát, L. Burget, J. Černockỳ, and S. Khudanpur. Recurrent neural network based language model. In *Eleventh Annual Conference of the International Speech Communication Association*, 2010.

[38] B. C. Ooi, K.-L. Tan, S. Wang, W. Wang, Q. Cai, G. Chen, J. Gao, Z. Luo, A. K. Tung, Y. Wang, et al. Singa: A distributed deep learning platform. In *Proceedings of the 23rd ACM international conference on Multimedia*, pages 685–688. ACM, 2015.

[39] O. Press and L. Wolf. Using the output embedding to improve language models. *arXiv preprint arXiv:1608.05859*, 2016.

[40] M. Richardson, E. Dominowska, and R. Ragno. Predicting clicks: estimating the click-through rate for new ads. In *Proceedings of the 16th international conference on World Wide Web*, pages 521–530. ACM, 2007.

[41] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

[42] X. Shi, B. Cui, G. Dobbie, and B. C. Ooi. Towards unified ad-hoc data processing. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, pages 1263–1274. ACM, 2014.

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

[44] S. Srinivas, A. Subramanya, and R. Venkatesh Babu. Training sparse neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 138–145, 2017.

[45] A. Veit, M. J. Wilber, and S. Belongie. Residual networks behave like ensembles of relatively shallow networks. In *Advances in Neural Information Processing Systems*, pages 550–558, 2016.

[46] L. Wang, J. Lin, and D. Metzler. A cascade ranking model for efficient ranked retrieval. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, pages 105–114. ACM, 2011.

[47] X. Wang, Y. Luo, D. Crankshaw, A. Tumanov, F. Yu, and J. E. Gonzalez. Idk cascades: Fast deep learning by learning not to overthink. *arXiv preprint arXiv:1706.00885*, 2017.

[48] X. Wang, F. Yu, Z.-Y. Dou, T. Darrell, and J. E. Gonzalez. Skipnet: Learning dynamic routing in convolutional networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 409–424, 2018.

[49] 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*, pages 2074–2082, 2016.

[50] Y. Wu and K. He. Group normalization. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 3–19, 2018.

[51] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He. Aggregated residual transformations for deep neural networks. In *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, pages 5987–5995. IEEE, 2017.

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

[53] S. Zagoruyko and N. Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

[54] W. Zaremba, I. Sutskever, and O. Vinyals. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014.

[55] C. Zhang, A. Kumar, and C. Ré. Materialization optimizations for feature selection workloads. *ACM Transactions on Database Systems (TODS)*, 41(1):2, 2016.

[56] X. Zhang, X. Zhou, M. Lin, and J. Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6848–6856, 2018.

[57] Y. Zhang, W. Zhang, and J. Yang. I/o-efficient statistical computing with riot. In *Data Engineering (ICDE), 2010 IEEE 26th International Conference on*, pages 1157–1160. IEEE, 2010.

[58] L. Zheng, L. Shen, L. Tian, S. Wang, J. Wang, and Q. Tian. Scalable person re-identification: A benchmark. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1116–1124, 2015.