sensAI: ConvNets Decomposition via Class Parallelism for Fast Inference on Live Data

Guanhua Wang [1] Zhuang Liu [1] Brandon Hsieh [1] Siyuan Zhuang [1]
Joseph Gonzalez [1] Trevor Darrell [1] Ion Stoica [1]

Abstract

Convolutional Neural Networks (ConvNets) enable computers to excel on vision learning tasks such as image classification, object detection. Recently, real-time inference on live data is becoming more and more important. From a system perspective, it requires fast inference on each single, incoming data item (e.g. 1 image). Two main-stream distributed model serving paradigms – data parallelism and model parallelism – are not necessarily desirable here, because we cannot further split a single input data piece via data parallelism, and model parallelism introduces huge communication overhead. To achieve live data inference with low latency, we propose sensAI, a novel and generic approach that decouples a CNN model into disconnected subnets, each is responsible for predicting certain class(es). We call this new model distribution paradigm as class parallelism. Experimental results show that, sensAI achieves up to 18x faster inference on single input data item with no or negligible accuracy loss on CIFAR-10, CIFAR-100 and ImageNet-1K datasets.

1 Introduction

Convolution Neural Networks (CNNs) have recently succeeded in many computer vision tasks, such as image classification (Krizhevsky et al., 2012; Simonyan & Zisserman, 2015; He et al., 2016) and object detection (Ren et al., 2015; Girshick et al., 2014). As both model size (Hu et al., 2018; Real et al., 2019) and image resolution (Deng et al., 2009; Lin et al., 2015) grow larger, the serving time of a single ConvNet increases drastically. Thus, distributed model serving is adopted to accelerate the process by running a single CNN over multiple GPUs or machines simultaneously (Paszke et al., 2017; Abadi et al., 2016). Conventional distributed approaches are data parallelism (Chen et al., 2015; Li et al., 2014) and model parallelism (Lee et al., 2014; Dean et al., 2012). In data parallelism, each GPU has a full copy of the model and does inference independently on a subset of the whole input data. Model parallelism adopts a different approach: each GPU only maintains a portion of the whole model, and communicates intermediate results (e.g. feature-maps) during each round of model serving.

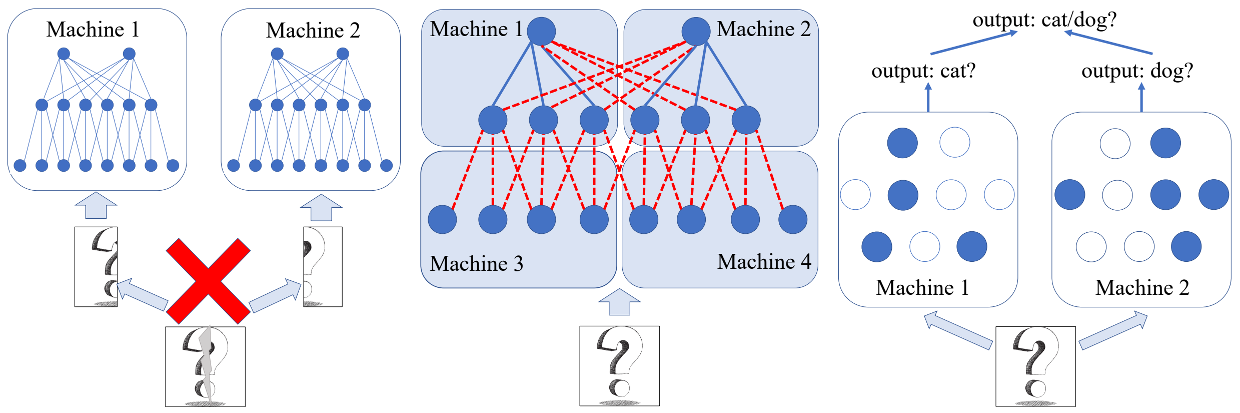
Making faster decision on live data is becoming increasingly important. In cases like autonomous driving (Paden et al., 2016; Badue et al., 2019), once the camera captures a frame of image that contains pedestrians, it may save people's lives if the stop decision can be made slightly faster. Other application scenarios like automatic stock trading using machine learning, right now is happening in giant banks like JP Morgan (Porzecanski, 2019) and Goldman Sachs (Jennings, 2018; Horwitz, 2020). If one party can make the trading decision several milliseconds earlier than the others, it can bring in huge amount of profits. From a system perspective, making faster decision on live data means faster model serving on each incoming, atomic data item (e.g. a single image, a stock's instantaneous price).

Neither the conventional data parallelism nor model parallelism can achieve faster serving on single data item. It is infeasible to split an atomic input piece further for data parallelism (shown in Fig. 1(a)). Model parallelism introduces huge communication overhead for transferring intermediate results (like feature-maps shown as red-dashed lines in Fig. 1(b)) among the GPUs in use. To achieve faster inference on single data item, we propose `sensAI`, a novel and generic approach that distributes a single CNN into disconnected subnets, and achieves decent serving accuracy with negligible communication overhead (1 float value).

`sensAI` achieves this extremely low communication overhead in distributed model serving by adopting a new concept: *class parallelism*, which decouples a classification ConvNet into multiple binary classifiers for independent, in-parallel inference (shown as Fig. 1(c)). The intuition behind *class parallelism* is, within a CNN, different neurons (i.e. channels) are responsible for predicting different classes, and typically only a subset of neurons is crucial for predicting

[1]University of California, Berkeley. Correspondence to: Guanhua Wang <guanhua@cs.berkeley.edu>.

Proceedings of the 4^{th} MLSys Conference, San Jose, CA, USA, 2021. Copyright 2021 by the author(s).



(a) Data Parallelism (b) Model Parallelism (c) Class Parallelism

Figure 1. Comparison of distributed model serving on single data piece among data parallelism, model parallelism and class parallelism.

one specific class probability (Yu et al., 2018). *Class parallelism* can also be used with data parallelism together by duplicating the whole set of binary classifiers.

For image classification tasks with a relatively small number of classes *N*, e.g., CIFAR-10 (Krizhevsky, 2009), we achieve *class parallelism* by pulling out *N* binary classifiers from a pretrained N-way classification CNN. And we use all these binary classifiers to perform faster, in-parallel inference by taking the maximum confidence output from these models to determine the predicted class. For harder classification tasks with more classes, e.g., ImageNet-1K (Russakovsky et al., 2015) with 1000 classes, instead of decoupling a given CNN into N binary classifiers, we divide the image classes into k groups, with each group containing m classes ($m \times k = N$). For each group of classes, we distill a m-way classifier from the base model. And we combine the outputs from those k smaller m-way classifiers to obtain the target N-way classification results.

In `sensAI`, we trade more computation resources (e.g. more GPUs) for faster inference speed on single data item. `sensAI` achieves decent scalability with *class parallelism*. Extensive experimental results on CIFAR-10/100 (Krizhevsky, 2009) and ImageNet-1K (Russakovsky et al., 2015) datasets show that, compared with baseline of single-GPU and multi-GPU via model parallelism, sensAI achieves up to 18x speedup for single image serving time. We also demonstrate `sensAI`'s effectiveness on more efficient CNNs like MobileNet-V2 (Sandler et al., 2018) and ShuffleNet-V2 (Ma et al., 2018). Despite that the aggregated size of all our decoupled binary/grouped classifiers may be similar or larger than the size of the base model, the FLOPs and model size are significantly reduced on each device (e.g. a GPU). Additionally, different from traditional model parallelism, `sensAI`'s tiny classifiers on each device can be concurrently executed without blocking each other. Our main goal is to reduce inference latency on live data, and `sensAI`'s *class parallelism* is a generic and simple method to speed up model serving on single data item.

2 Related Work

Data and model parallelism: To boost up model serving speed, data parallelism (Goyal et al., 2017; Jia et al., 2019; Wang et al., 2020; Or et al., 2020) and model parallelism (Kim et al., 2016; Dean et al., 2012; Huang et al., 2019) are widely adopted for in-parallel CNN inference (Yu & Chowdhuryo, 2020). In data-parallel model serving, each machine (or GPU) maintains a replica of the entire CNN model, and process a partition of input data (Gu et al., 2019; Peng et al., 2019). Another common parallelism strategy is model parallelism (Dean et al., 2012), where a CNN model is split into disjoint subsets on multiple devices (Kim et al., 2016; Lee et al., 2014). For each mini-batch of model serving, a large amount of intermediate results (e.g. feature-maps) need to be transferred among the GPUs that hold adjacent model partitions.

However, neither of them can be used to reduce inference latency on live data. Given that we cannot further split an atomic data piece, data parallelism is not applicable to speed-up inference on single input data item (Li et al., 2020; Jayarajan et al., 2019). Although we can increase parallelism via model parallelism, given that the inference time of a single input piece is extremely short (e.g. several milliseconds), the systematic networking communication time for transmitting huge intermediate results (e.g. feature-maps) are often several orders of magnitude higher than the inference time of an image (Dean et al., 2012; Paszke et al., 2017; Jeaugey, 2017; Lee et al., 2014), making model parallelism unlikely to speed-up single image inference. `sensAI` proposes a new parallelism method, called *class parallelism*, which decouples a CNN model into disconnected subnets. Only a single float value (i.e. the confidence level of predicting an input piece belonging to one specific class) communication is needed in the end. This new approach of independent, in-parallel inference allows us to further accelerate inference on single input piece. In addition, our *class parallelism* can also be adopted together with data parallelism (by replicating the whole set of binary classifiers) to further reduce

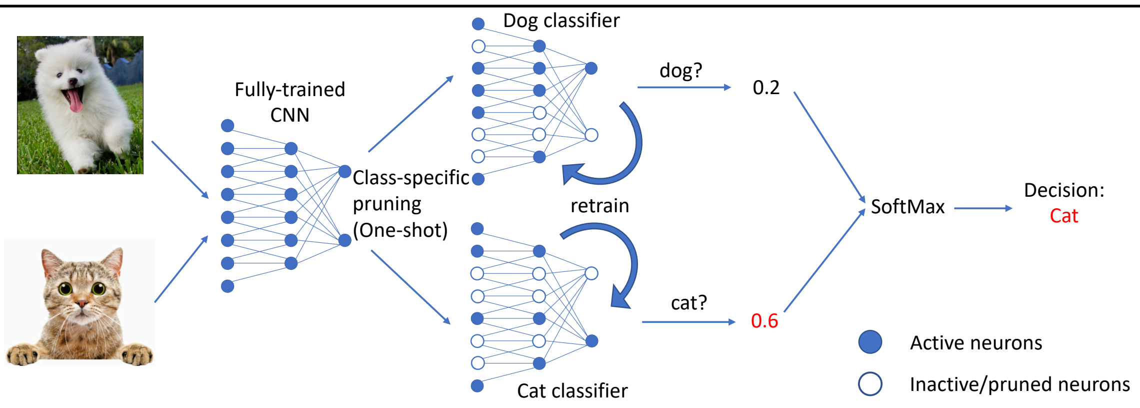


Figure 2. sensAI workflow for binary, in-parallel inference.

model serving latency on batches of input data.

Class-specific neuron analysis: Zhou et al. (Zhou et al., 2018) point out that unit ablation on a fully trained CNN model will only decrease inference accuracy on certain class, and then analyze the correlation between units ablation and its impacted class. Yu et al. (Yu et al., 2018) show the possibility of decoupling a 10-way CNN model into ten binary classifiers. However, even these literature points out that the neurons belonging to certain class of images can be relatively independent, `sensAI` is the first approach to propose the concept of *class parallelism* and use it for in-parallel model inference.

Network pruning: Over-parameterization is a well-known attribute of CNNs (Denton et al., 2014; Ba & Caruana, 2014). To reduce memory footprints and computational cost, network pruning (Han et al., 2015; Li et al., 2017; Liu et al., 2017) gains attention and is recognized as an effective way to improve computational efficiency while maintaining decent accuracy. `sensAI` also adopts network pruning technique to pull out *class-parallel* models from the original CNN. Different from existing class-agnostic pruning methods, `sensAI` uses one-shot, class-specific pruning. And `sensAI` can combine class-agnostic pruning schemes (Liu et al., 2017; Han et al., 2015; 2016) to further shrink down the size of our binary models.

One-Vs-All (OVA) reduction: OVA machine learning model reduction is a general approach which reduces a multi-class learning problem into multiple simpler problems solvable with binary classifiers (Galara et al., 2011; Beygelzimer et al., 2016). Rifkin et al. (Rifkin & Klautau, 2004) and Beygelzimer et al. (Beygelzimer et al., 2005) demonstrate OVA's effectiveness via both experiments and theoretical arguments. Another line of work combines OVA with Error-Correcting Output Codes (ECOC) to further boost model serving accuracy (Dietterich & Bakiri, 1995; Kong & Dietterich, 1995; Deng et al., 2010). Different from traditional OVA approaches and ECOC extension which train binary classifiers with *predefined* model structure (Anand et al., 1995; Dietterich & Bakiri, 1995), `sensAI` learns different model structures from fully-trained base model for different binary classification tasks, which achieves better serving accuracy with less redundant binary models.

3 Method

In `sensAI`, we trade more computational resources (e.g. more GPUs) for fast inference on single data item. Our main goal is to reduce model serving latency (not FLOPs or model size) on single input piece. `sensAI` proposes *class parallelism*, which decouples a CNN model into binary classifiers for each class. For dataset with too many classes, we design grouped classifiers, that each is responsible for prediction of multiple classes. And our grouped classifiers can achieve arbitrary degree of scalability from single machine to number of machines equivalent to image classes. In this section, we first present `sensAI` high-level workflow. Then we describe how each component works and discuss the corresponding design in details.

3.1 Workflow overview

We assume to have a normally fully-trained N-way CNN classifier where N is the total number of classes. As a toy example shown in Fig. 2, sensAI decouples a CNN model for faster, in-parallel inference via the following 3 steps: class-specific pruning, retraining and combining results back to original N-way predictions. Given a pre-trained CNN, we first conduct one-shot, class-specific pruning to pull out binary or grouped (i.e. multi-class) classifiers from the base model. Second, to recover possibly lost serving accuracy, we retrain each binary/grouped classifier independently. Third, we deploy each binary/grouped classifier on a single GPU for faster, in-parallel inference on live data.

3.2 Class-specific pruning

Here we first discuss how to distill binary classifiers from the pre-trained base model. Then we extend it to pull out grouped (multi-class) classifiers from the base model.

3.2.1 Binary classifiers:

In the process of distilling binary classifiers from a fully-trained model, we would like to identify the neurons that are important for predicting each specific class. Here we use activation-based criteria to determine the importance of neurons for each class. After feeding all input images of one class to the fully-trained model, we collect activation statistics for each neuron (i.e. channel), and based on that statistics we determine which neurons to keep or prune for obtaining binary classifier for that class. For example, if our criterion is Average Percentage of Zeros (APoZ (Hu et al., 2017)), we prune out the neurons that have larger number of zeros in their activation maps when taking that certain class as input. For the final classification layer, we only keep the prediction head of the class of interest.

How to obtain such binary classifiers via class-specific pruning is crucial to the success of our method, as it is responsible for identifying sub-network that performs well for predicting each class, from which we will combine the outputs to get the final prediction results. Here we examine three activation-based pruning criteria, namely APoZ (Hu et al., 2017), average activations (Avg) (Yu et al., 2018), and a hybrid criterion we proposed which combines the above two. The APoZ criterion is already explained above. The average activation criterion is similar to APoZ except it uses the mean absolute value of a channel instead of the average percentage of zeroes.

Our hybrid policy is defined as follows:

$$\Psi(N^c_{i,j}) = \frac{\sum^{D_c} \Phi(A^c_{i,j} < \theta_1)}{D_c} \tag{1}$$

$$where\ \Theta(A^c_{i,j}) < \theta_2 \tag{2}$$

where $N^c_{i,j}$ denotes the j-th neuron in i-th layer evaluating on the c-th class of images, and its corresponding output featuremaps are $A^c_{i,j}$. D_c refers to the number of images belongs to the c-th class. $\Phi(\cdot)$ calculates the average percentage of featuremap values smaller than θ_1 for each image in the set of the c-th class.

First, instead of calculating average percentage of zeros (APoZ), we generalize it as $\Psi(\cdot)$, which calculates the average percentage of featuremap values less than threshold θ_1. Then we generate the initial candidate list of neurons need to be pruned by setting a percentage threshold on $\Psi(\cdot)$. Second, for each prune candidate, we evaluate its corresponding featuremaps' mean absolute value $\Theta(A^c_{i,j})$. If the mean absolute value is larger than the threshold θ_2 we set, we exclude it from the prune candidate list. *The key insight of this hybrid policy is to avoid pruning neurons that generate featuremaps with high percentage of near-zero values, and have very large non-zero values at the same time.*

3.2.2 *Grouped classifiers:*

For simple classification tasks like 10-class CIFAR-10 dataset (Krizhevsky, 2009), it is feasible to distill from pre-trained model into 10 binary classifiers to achieve parallelism across 10 GPUs or machines. However, for tasks with a huge amount of classes, such as 1000-way ImageNet-1K classification dataset (Russakovsky et al., 2015), pulling

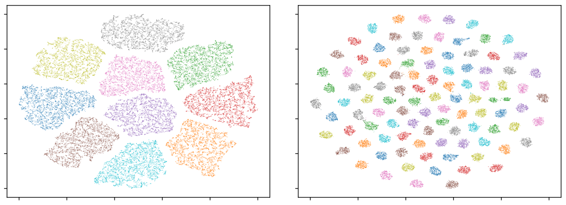

(a) CIFAR-10 (b) CIFAR-100

Figure 3. t-SNE visualization for feature representation of training images using pre-trained VGG-19 on CIFAR-10 and CIFAR-100.

out 1000 binary classifiers for *class parallelism* can be unrealistic, as maintaining 1000 binary models can be expensive and parallelism over 1000 GPUs may not be practical in many circumstances.

In this case, instead of distilling binary classifiers per class, we first divide all N classes into k groups, each with m classes ($m \times k = N$), and then use similar pruning techniques to distill a multi-class classifier for each group. The grouping assignment can be done randomly or based on some priors on similarity among the classes. Here we evaluate two grouping approaches: random grouping, and our novel grouping method called *nearby grouping* for grouping similar classes.

In our *nearby grouping*, we first evaluate the similarity of feature representation for images belonging to different classes, and then group the classes that share high similarity. As shown in Fig. 3, we use t-distributed stochastic neighbor embedding (t-SNE) (van der Maaten & Hinton, 2008) to visualize distance of feature representation (collected from feature-vector before the last fully-connected layer when passing training images into a fully-trained VGG-19) for all images in both CIFAR-10 (Fig. 3(a)) and CIFAR-100 (Fig.3(b)). In Fig. 3, the data points with same color represent the feature-vector of images belong to the same class. We then adopt k-means clustering (MacQueen, 1967) (as we describe in Appendix. A) over feature representation of images to group multiple classes that are close to each other (*nearby grouping*), as t-SNE ensures nearby classes will be close in projected space by minimizing the Kullback-Leibler divergence between joint distribution of points in input and projected space (Linderman & Steinerberge, 2019).

In group-wise pruning, we feed all the images in one class group (instead of a single class) and collect the statistics for pruning. For the final classification layer, we keep all m predictions of classes in that group. We add another head to indicate the "negative" samples, which refers to the input images do not fall into this group of classes. Therefore, each grouped classifier becomes an $(m + 1)$-way classification model.

Note that even though our grouping technique is designed for the special case of too many classes, it is also applicable to problems with fewer classes, to accommodate the need for different levels of parallelism. It also works decently even in the case that the class number cannot be divided by the grouping number. In reality, we can always ensure the class difference between largest and smallest class-group to be $<= 1$ (by splitting classes as evenly as possible), thus the imbalance effect should be minimal, especially when total number of classes is large.

3.3 Retraining

For binary classifiers, we impose a retraining process to regain the possibly lost serving performance. For each binary model, we form a new retraining dataset, which consists of all the positive samples (i.e. images belong to that class), and images from the rest of classes as negative training samples. We also balance the positive and negative samples by keeping the number of images from two groups to be the same when loading images for each epoch of retraining. Each binary classifier is then trained with binary cross-entropy (BCE) loss on its own retraining dataset.

For grouped classifiers, the retraining process is roughly the same, except now we have $(m + 1)$-way classification (instead of binary classification) for each grouped model, with m positive and 1 negative classes probability heads. Here we also balance the retraining dataset so that we draw equal number of images from $m + 1$ classes in each epoch of retraining. Each grouped classifier is trained with the conventional multi-way cross-entropy (CE) loss. Note that cross-entropy loss also works perfectly even with imbalanced dataset, in which case the output produced by the network still represents its estimated probabilities.

3.4 Combine results back to N-way predictions

After getting all the retrained binary models, we combine their outputs together for the original N-way inference tasks. We simply select the maximum probability (e.g. 0.6 in Fig. 2) of being positive across all binary classifiers' outputs to determine the N-way classification result (e.g. "cat" decision in Fig. 2).

In the case of grouped classifiers, instead of outputting a single scalar (the probability of being a specific class), each grouped classifier outputs a $m + 1$ dimensional probability vector that sums up to 1. The first m scalars correspond to the probability of being each of the m classes, and the last one refers to the probability of not belonging to the classes in this group. Collecting the first m scalars from all k grouped models, we have $m \times k = N$ probability back again. We simply take the maximum among those N probabilities as the predicted class.

4 Experiments

4.1 Datasets and Models

We evaluate sensAI model serving performance mainly using two standard types of CNN models, namely ResNet (He et al., 2016) and VGGNet (Simonyan & Zisserman, 2015), on three different datasets: CIFAR-10, CIFAR-

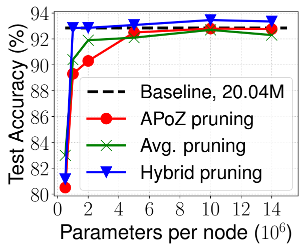


Figure 4. Pruning methods comparison: APoZ, Avg, hybrid (VGG-19, CIFAR-10).

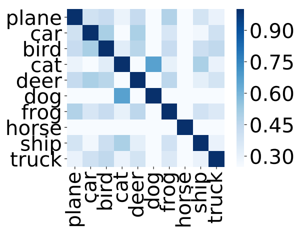


Figure 5. Similarity among binary classifiers measured by IoU on channels.

100 (Krizhevsky, 2009) and ImageNet-1K (Russakovsky et al., 2015). We also verify sensAI's effectiveness on more efficient networks like MobileNet-V2 (Sandler et al., 2018) and ShuffleNet-V2 (Ma et al., 2018). We report model-size and FLOPs reduction, the actual time speed-up when running in parallel, and the test accuracy performance (Top-1 accuracy) change.

We use G3 instance (NVIDIA Tesla M60 GPU) on Amazon Web Services (AWS) for all experiments. For each binary/grouped classifier we have, we assign it on a single GPU with no sharing among other workloads.

4.2 CIFAR-10 results

For CIFAR-10, we mainly evaluate two standard types of CNNs: VGG-19 with batch normalization and ResNet-164, which are different in both network type (VGG v.s. ResNet) and model depth (19 v.s. 164). We also test sensAI improvements on two more efficient CNNs: MobileNet-V2 and ShuffleNet-V2 in Section 4.2.5. We re-run the training process with standard hyper-parameter setting [1] (in total 164 epochs, learning rate starts with 0.1, decay by 0.1 at 81, 122 epochs), and get test accuracy of these baseline models: 92.85% for VGG-19, 94.79% for ResNet-164, 94.24% for MobileNet-V2 and 93.46% for ShuffleNet-V2.

In this section, we first evaluate three popular channel-level pruning metrics, namely APoZ, Avg, and our hybrid policy as we discussed in Sec. 3.2.1. Since hybrid pruning gives the best results, we adopt hybrid pruning policy for all the following experiments on different datasets (e.g. CIFAR-10/100). Second, with one-shot class-specific pruning and retraining, we show the maximum model size reduction we achieved for both VGG-19 and ResNet-164, which leads to 2-6x speedup over 1-GPU baseline for single image inference at no accuracy loss. Third, we further conduct performance comparison between sensAI and baseline with model parallelism using the same number of GPUs, where sensAI achieves up to 18x latency reduction for per-image serving. Fourth, we compare the test accuracy between training small binary classifiers with pre-defined structure (OVA) and sensAI's binary models distilled from pre-trained base model. Fifth, we further demonstrate sensAI's effective-

[1]Borrowed from multiple popular github repositories like *bearpaw/pytorch-classification* and *kuangliu/pytorch-cifar*

ness on more efficient neural networks like ShuffleNet-V2 and MobileNet-V2. Lastly, we conduct some statistical analysis over our distilled binary classifiers.

4.2.1 *Pruning policy comparison*

Here we evaluate three popular structured (i.e. channel-level) pruning metrics on feature-maps: APoZ (Hu et al., 2017), Avg (Yu et al., 2018), and our hybrid policy, which are described in Sec. 3.2.1. We compare the final inference accuracy performance using these three mechanisms on VGG-19 using CIFAR-10 dataset.

For implementation of our hybrid neuron pruning method, as defined in Sec. 3.2.1 as Equation 1 and 2, we collect neurons with higher percentage ($\Psi(\cdot) \geq x$) of featuremap values less than threshold value θ_1 and then combining with average activation pruning. Therefore we have three hyper-parameter to tune: $\geq x$ percentage of value less than θ_1, average activation threshold θ_2. The thresholds are determined by a grid search on a separate validation set, which are not layer-specific, but global in the network. The pruning rate of each layer is thus determined by the thresholds.

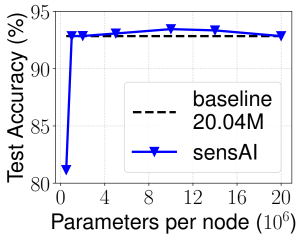
In Fig. 4, hybrid pruning generally performs better than either APoZ or Avg pruning: it reaches highest test accuracy when the size of distilled binary classifiers are similar using different pruning techniques. Therefore, we adopt our proposed hybrid pruning policy as the default class-specific pruning metric for all the following `sensAI` experiments.

4.2.2 *sensAI evaluation on VGG-19 and ResNet-164*

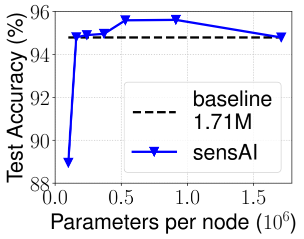
We evaluate `sensAI` performance in three aspects: number of parameters in each binary model, FLOPs cost, end-to-end time saving for model inference. For retraining on pruned binary classifiers over fully-trained baseline models, we limit our retrain epochs to be 80, and borrow similar learning rate decay methodology as illustrated before.

Fig. 6(a), 7(a), 8(a) depict the results of applying *class parallism* on VGG-19. Surprisingly, with only one-shot pruning and retraining, we can reduce number of parameters by 20x (Fig. 6(a)), FLOPs by 24x (Fig. 7(a)) at no test accuracy loss, which leads to **6x** serving time reduction per image (Fig. 8(a)).

Similar results are achieved on ResNet-164 which are shown in Fig. 6(b), 7(b), and 8(b). Compared with base model, each binary classifier in `sensAI` achieves model size reduction by 11x (Fig.6(b)), FLOPs reduction by 11x (Fig.7(b)) without test accuracy loss, which leads to **2x** reduction of serving latency for single image (Fig.8(b)). The less inference time saving for ResNet-164 model is mainly due to the model depth is too large. Only reducing the width (i.e. channels) has less effects on the end-to-end time saving, since the depth of the model dominates the model inference time. It is widely-believed that model pruning should not reduce the model depth (Liu et al., 2017; Li et al., 2017; Liu

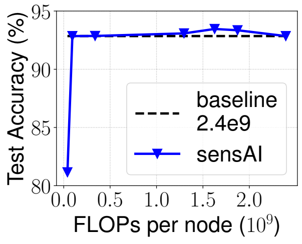


(a) VGG-19

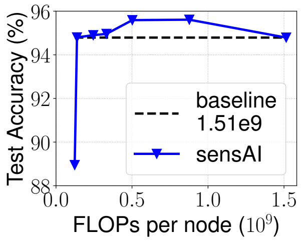


(b) ResNet-164

Figure 6. Number of parameters v.s. test accuracy comparison of VGG-19 and ResNet-164 on CIFAR10.

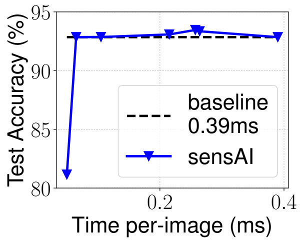


(a) VGG-19

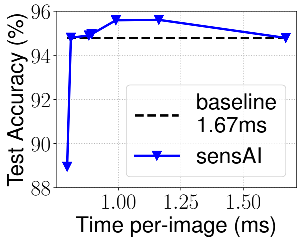


(b) ResNet-164

Figure 7. FLOPs v.s. test accuracy comparison of VGG-19 and ResNet-164 on CIFAR10.



(a) VGG-19



(b) ResNet-164

Figure 8. Per-image inference time v.s. test accuracy comparison of VGG-19 and ResNet-164 on CIFAR10.

et al., 2019). With this in mind, reducing end-to-end model serving time by 2x on ResNet-164 is still decent.

The intuition behind this high ratio of single-shot pruning is: *we simplify the inference task from 10-way classification to 1-way.* Thus, for each binary model, the amount of inactive neurons we can prune is much more than traditional, class-agnostic pruning over original 10-way classification model. Besides the one-shot, class-specific pruning method we use, we can further shrink down our binary model size by combining with class-agnostic pruning methods (Liu et al., 2017; Han et al., 2015; 2016) for further iterative pruning.

Additionally, we achieve better test accuracy when maintaining a little more neurons in each binary classifiers. On VGG-19 in Fig.6(a), we get higher test accuracy when each binary model size is in the range of 2M$\sim$14M. For ResNet-164 in Fig.6(b), better test accuracy can be achieved when average binary model size is between 0.16M and 0.91M.

4.2.3 *Model parallelism v.s. sensAI*

As our main goal is achieving fast inference on single data piece, data parallelism is not applicable. Therefore, we compare per-image inference speed between model parallelism

	Model	Test acc. (%)	Avg. per-GPU model size (10^6)	Per-image infer. time (ms)
MP-baseline (10 GPUs)	VGG-19	92.85	2.00	1.12
	ResNet-164	94.79	0.17	2.33
sensAI	VGG-19	92.86	1.01	0.06
	ResNet-164	94.79	0.16	0.81

Table 1. Comparison between baseline with model parallelism (MP) and sensAI using 10 GPUs.

and sensAI by using the same number of GPUs (10 GPUs).

As suggested by recent literature (Narayanan et al., 2019; Huang et al., 2019) on model parallelism, we evenly split both VGG-19 and ResNet-164 models over 10 GPUs to balance work, and test the performance of single image inference. The following baseline experiments with model parallelism on CIFAR-100 and ImageNet-1K obey the same rule. In Table 1, adopting model parallelism to baseline model actually perform worse than single GPU baseline, single image inference time is 0.73 ms longer on VGG-19 and 0.66 ms longer on ResNet-164 than using single GPU. Thus, compared with baseline with model parallelism using the same number of GPUs as `sensAI`, `sensAI` achieves **18x** speedup on VGG-19 and $\sim$**3x** speedup on ResNet-164 for per-image serving time without test accuracy loss.

The reason for model parallelism's bad performance on single image serving is twofold: First, model parallelism splits layers of weights across multiple GPUs, which leads to huge communication overhead for transferring intermediate results (e.g. feature-maps) (Dean et al., 2012). Second, model parallelism also under-utilizes computational resources on GPUs, which is mainly due to the sequential dependency of underlying neural networks that introduces more hard synchronization barriers among GPUs holding different model partitions (Lepikhin et al., 2021). For instance, in Fig. 1(b), machine 1 and 2 cannot start computation until they completely receive outputs from machine 3 and 4. In contrast, the binary classifiers of sensAI can independently process the input data item without blocking each other.

4.2.4 *OVA v.s. sensAI*

One-Vs-All (OVA) reduction strategy has been studied for decades (Galara et al., 2011; Beygelzimer et al., 2016; Dietterich & Bakiri, 1995). Since both sensAI and OVA decouple a multi-classification problem into a bunch of binary classification problems, here we conduct a direct comparison between training multiple pre-defined small binary models (OVA) and pulling out binary models from pre-trained base model (sensAI) on CIFAR-10 dataset.

For OVA, we directly train small CNNs like ResNet-20 from scratch, each of which is responsible for binary classification task on a single class in CIFAR-10 dataset. We also borrow the same training recipe mentioned in Sec. 4.2. We use the same per-class dataset as we used for `sensAI` binary model retraining to train each ResNet-20 as OVA's binary classifier, and impose calibration among OVA binary classifiers to

	Test Accuracy %	Average binary model size (10^6)
OVA	91.65	0.27
sensAI	94.79	0.16
	94.90	0.24

Table 2. Comparison between OVA and sensAI with 10 GPUs.

obtain the possibly best serving accuracy. Then we use these OVA binary classifiers in the same way as `sensAI`'s in-parallel inference setting. For `sensAI`, we just directly use two groups of binary classifiers pulling out from ResNet-164 mentioned in Section 4.2.2, one group with smaller average model size than OVA's ResNet-20, the other group with similar average model size as ResNet-20.

In Table 2, with similar binary model size, as `sensAI` with average model size of 0.24M parameters and OVA's ResNet-20 with 0.27M parameters, our binary classifiers pulling from ResNet-164 outperform OVA's ResNet-20 binary classifiers by **3.25%** on test accuracy. With `sensAI` binary models (average size of 0.16M parameters) which are smaller than OVA ones, `sensAI` still outperforms OVA by $\sim$3% on test accuracy for CIFAR-10 dataset. The conclusion here is: comparing with OVA's training small binary models with the *same* and *pre-defined* structure, `sensAI` learns different binary model structures from a big base model for different binary classification tasks, which achieves better serving accuracy at less computational cost.

4.2.5 *sensAI improvements on efficient CNNs*

Above we evaluate `sensAI` on standard CNNs like VGG and ResNet series. Here we explore the performance gain that `sensAI` can achieve on efficient neural networks like ShuffleNet-V2 (Ma et al., 2018) and MobileNet-V2 (Sandler et al., 2018).

Since these efficient models are small in number of parameters and designed to be run on single device, we report `sensAI`'s speed-up over single GPU baseline. As shown in Table 3, sensAI achieves 5x model size reduction, 3.5x FLOPs reduction on MobileNet-V2, and 4x reduction for both model size and FLOPs on ShuffleNet-V2, which leads $\sim$**2x** time reduction for per-image inference on both MobileNet-V2 and ShuffleNet-V2. It seems unfair to compare sensAI using 10 GPUs whereas baseline models only use single GPU. However, as Sec. 4.2.3 indicating that, splitting small and efficient models with model parallelism across multiple GPUs will definitely perform worse than using single GPU for single image serving.

4.2.6 *Binary models analysis*

Here we conduct simple analysis over the best sets of binary classifiers (i.e. smallest model sizes without test accuracy loss) we get from VGG-19, ResNet-164, MobileNet-V2 and ShuffleNet-V2 on CIFAR-10 dataset.

We first analyze the similarity of the pruned binary classifiers, for VGG-19 on CIFAR-10. For each binary classifier,

	Model	Test acc. (%)	Model size per-GPU (10^6)	FLOPs per-GPU (10^9)	Per-image infer. time (ms)
Baseline	MobileNet-V2	94.24	2.30	0.56	0.48
	ShuffleNet-V2	93.46	1.26	0.27	0.23
sensAI	MobileNet-V2	94.27	0.45	0.16	0.22
	ShuffleNet-V2	93.50	0.31	0.07	0.12

Table 3. Comparison between efficient baseline model and sensAI.

we collect the index of its channels in the original baseline model. The similarity of two architectures is measured by intersection of unions (IoU) of the channel indices. In Fig. 5, we visualize the similarity of pairs of classes in a heatmap. Higher similarities are indicated by darker colors. We observe that the cat and dog binary classifiers share most channels among all pairs, which is intuitive since they are similar to human and their features might be more correlated. Similar correlation results are also found on other popular models like ResNet-164, MobileNet-V2 and ShuffleNet-V2, which are shown in Appendix B.

Second, we analyze model size variation in each group of 10 binary classifiers. We find that, the binary models usually have similar number of parameters. For example, for VGG-19, the binary models are all with $\sim$1M parameters (variance $<$0.1M), and $\sim$0.16M parameters (variance $<$0.02M) for ResNet-164. Binary models getting from efficient CNNs also have similar patterns, where all binary models are with $\sim$0.45M parameters (variance $<$0.05M) for MobileNet-V2, and $\sim$0.31M parameters (variance $<$0.04M) for ShuffleNet-V2. The intuition behind may because we evenly split the multi-classification task among all the binary classifiers. This decent feature guarantees our in-parallel inference can finish roughly at the same time (i.e. no stragglers) among the whole set of binary classifiers involved.

4.3 CIFAR-100 results

We discuss results for grouped classifiers on CIFAR-100 dataset. We evaluate two class-grouping methods: random grouping and *nearby grouping* described in Sec. 3.2.2. We test on two CNNs: VGG-19 and ResNet-110. We get test accuracy of these two baseline models as 71.95% for VGG-19, 72.44% for ResNet-110. We measured model size, FLOPs, and relative speed-ups of per-image inference time for these two baseline models (as Baseline/MP-baseline in Table 4).

As discussed in Sec. 4.2.6, to achieve good load balancing and also minimize end-to-end, in-parallel inference time, each of our grouped model picks the same number of classes, so that the grouped models may take roughly the same amount of time in forward propagation during in-parallel model serving. Therefore, we have two group size settings: in 10-group setting, each grouped classifier is responsible for model prediction of 10 classes. In 5-group case, each grouped classifier is serving for 20 classes.

We directly adopt hybrid pruning policy and learning rate decay settings during retraining the same as CIFAR-10 experiments. For retraining on CIFAR-100, since the small amount of training images per-class (500 images per-class) is more likely to cause overfitting, we reduce the number of retraining epochs to be 40, and collect the best set of grouped classifiers during the retraining session.

4.3.1 *sensAI speedup over 1-GPU baseline*

As shown in Table 4, here we only list the minimum average grouped model size we can get without losing too much test accuracy. Our novel *nearby grouping* performs better than random grouping regarding both test accuracy and reduction of single-image serving time.

Random grouping: with 5 groups, we can achieve model size reduction 4.6x/3.2x, FLOPs reduction 2.2x/2.6x, per-image inference speed-up 2.17x/1.64x over single GPU baseline on VGG-19/ResNet-110. For 10-group case, random grouping can reduce model size by 9x/7x, FLOPs by 6x/5x, which lead to **3.25x/1.87x** time reduction for per-image inference on VGG-19/ResNet-110, separately.

Nearby grouping: in contrast, with 5 groups, compared with 1-GPU baseline, our *nearby grouping* can achieve reduction in model size by 6.6x/5.3x, FLOPs by 5x/4x, per-image serving time by 2.44x/1.8x on VGG-19/ResNet-110. For 10-group case with negligible test accuracy loss (0.03% loss on ResNet-110), *nearby grouping* can achieve 12x/10x model size reduction, 11x/9x FLOPs reduction and **4.5x/2.06x** single-image serving time reduction on VGG-19/ResNet-110, respectively.

4.3.2 *sensAI speedup over model-parallel baseline*

We compare `sensAI` performance with model-parallel baseline using same amount of GPUs. Since sensAI with nearby grouping achieves better performance than random grouping in most cases, we mainly focus on the comparison results between model-parallel baseline (MP-baseline) and sensAI using *nearby grouping*.

In Table 4, even though model size and FLOPs per-GPU are significantly reduced, model-parallel baseline (MP-baseline) generally performs worse than 1-GPU baseline on single image inference time, which is due to intermediate results transfer and sequential dependency of different model partitions as illustrated in Sec. 4.2.3. We report the results of 5-GPU and 10-GPU cases here.

5-GPU: compared with 5-GPU MP-baseline, sensAI with 5-group achieves similar model size and FLOPs reduction on both VGG-19 and ResNet-110. However, these small grouped models in sensAI can run concurrently without blocking each other (whereas model parallelism does). sensAI can obtain **2.44/0.41=5.95x** and **1.8/0.78=2.3x** speedup for per-image serving on VGG-19/ResNet-110, separately.

10-GPU: similarly, compared with 10-GPU MP-baseline, sensAI with 10-group using nearby-grouping achieves

Type	Model	Grouping method	Test acc. (%)	Model size (10^6)	FLOPs (10^9)	Per-image infer. time speed-up (over 1-GPU Baseline)
Baseline (1 GPU)	VGG-19	N/A	71.95	20.02	2.39	1x
	ResNet-110	N/A	72.44	1.66	1.3	1x
MP-baseline (5 GPUs)	VGG-19	N/A	71.95	4.00	0.48	0.41x
	ResNet-110	N/A	72.44	0.33	0.29	0.78x
MP-baseline (10 GPUs)	VGG-19	N/A	71.95	2.00	0.24	0.33x
	ResNet-110	N/A	72.44	0.17	0.14	0.76x
sensAI	VGG-19	Random 5-group	71.75	4.28	1.06	2.17x
		Random 10-group	71.84	2.21	0.36	3.25x
		Nearby 5-group	72.07	3.01	0.47	2.44x
		Nearby 10-group	72.18	1.61	0.21	4.5x
	ResNet-110	Random 5-group	72.43	0.513	0.501	1.64x
		Random 10-group	72.14	0.24	0.233	1.87x
		Nearby 5-group	72.52	0.31	0.298	1.8x
		Nearby 10-group	72.41	0.167	0.143	2.06x

Table 4. Comparison between sensAI two grouping methods (random and nearby grouping) with 5-group (5 GPUs) 10-group (10 GPUs) v.s. baselines of 1-GPU and model parallism (MP) using same amount of GPUs (5, 10 GPUs) on CIFAR-100.

slightly lower numbers in model size and FLOPs per-GPU for VGG-19 and ResNet-110. For single image serving time, sensAI with 10-group achieves **4.5/0.33=13.64x** and **2.06/0.76=2.71x** speedup compared with its model-parallel counterparts on VGG-19 and ResNet-110, respectively.

4.4 ImageNet-1K results

Here we report experiment results of `sensAI` for VGG-19 and ResNet-50 on ImageNet-1K dataset. We use the same hybrid pruning policy as in our CIFAR-10/100 experiments. And we limit our retraining epochs to be 40, with learning rate delay by 0.1 at 20, 30 epoch numbers. We use pre-trained models as our baseline (shown as Baseline in Table 5). Since *nearby grouping* generally performs better than random grouping, here we apply *nearby grouping* strategy for 10 and 20 groups on ImageNet-1K dataset, each grouped classifier is responsible for 100 and 50 classes prediction, separately. And we report the best results (smallest model size with negligible test accuracy loss) here as Table 5.

4.4.1 sensAI speedup over 1-GPU baseline

For VGG-19, as summarized in Table 5, using 10-group classifiers in `sensAI`, comparing with single GPU baseline model, we reduce each grouped model size by 9x at no loss of test accuracy, which achieves FLOPs reduction by 6x and reduce inference time by 3.4x. With 20-group classifiers in `sensAI`, we can reduce each grouped model size by $\sim$17x, and achieves FLOPs reduction by 12x while reducing inference time by $\sim$**6x**(i.e. 5.87x) without test accuracy loss.

For ResNet-50, in Table 5, decoupling baseline model into 10 grouped classifiers with `sensAI`, we reduce each grouped model size by 9.4x, FLOPs by 8.4x and inference time by 2.16x at no loss of test accuracy. With 20 grouped classifiers, we reduce model size by 14x, FLOPs by 13x, per-image inference time by **3.08x** with moderate accuracy loss of 0.07%. Compared with deep ResNets (i.e. ResNet-110 and ResNet-164) we used for CIFAR-10/100 datasets, we show `sensAI` can indeed prune more neurons and achieve faster per-image inference performance for slightly shallower ResNet like ResNet-50.

One thing worth pointing out is, for ImageNet-1K dataset, since the image resolution is much bigger compared with 32x32 CIFAR-10/100 datasets, `sensAI` can really achieve per-image inference time reduction at serveral millisecond level. For example, in 20-group with VGG-19 model setting, `sensAI` can reduce **7.79 ms** per-image inference latency over 1 GPU baseline at no accuracy loss.

Another thing worth noting is, since each of our grouped classifiers is retrained independently, given that ImageNet-1K dataset takes longer time to train than CIFAR-10/100, we can adopt data parallelism on each of our grouped classifier to further boost up retraining speed. We further extend our *class parallelism* to class-specific, concurrent model training with *zero* communication as Appendix C.

We also note that the aggregated number of parameters among the whole set of our grouped classifiers is larger than the base model. However, our main goal is to reduce serving latency on single image. And *Class parallelism* provides a generic way to reduce serving latency on single input piece. We can further reduce our grouped model size by adopting some class-agnostic pruning methods (Liu et al., 2017; Han et al., 2015; 2016) for further iterative pruning.

4.4.2 sensAI speedup over model-parallel baseline

We evenly split the base model onto the same number of GPUs that `sensAI` uses. Below we report the comparison between model-parallel baseline and our approach.

10-GPU: compared with 10-GPU MP-baseline in Table 5, even though sensAI with 10-group maintains higher number of model parameters and FLOPs per-GPU, sensAI achieves **7.5x** (3.4/0.45) and **3.7x** (2.16/0.58) latency reduction for single image serving on VGG-19 and ResNet-50, separately.

20-GPU: Compared with baseline of model parallel using 20 GPUs, sensAI with 20-group achieves time reduction by **13.5x** (5.87/0.43) and **5.4x** (3.08/0.57) for per-image inference on VGG-19 and ResNet-50, respectively.

Type	Model	Nearby Grouping	Test acc. (%)	Model size (10^6)	FLOPs (10^9)	Per-image infer. time speed-up (over 1-GPU Baseline)
Baseline (1 GPU)	VGG-19	N/A	74.21	143.68	117.99	1x
	ResNet-50	N/A	76.13	25.56	24.63	1x
MP-baseline (10 GPUs)	VGG-19	N/A	74.21	14.37	11.8	0.45x
	ResNet-50	N/A	76.13	2.56	2.47	0.58x
MP-baseline (20 GPUs)	VGG-19	N/A	74.21	7.2	5.9	0.43x
	ResNet-50	N/A	76.13	1.28	1.24	0.57x
sensAI	VGG-19	10-group	74.26	15.34	18.45	3.4x
		20-group	74.24	8.57	9.73	5.87x
	ResNet-50	10-group	76.17	2.71	2.93	2.16x
		20-group	76.06	1.83	1.89	3.08x

Table 5. Comparison between sensAI of 10-group (10 GPUs) and 20-group (20 GPUs) v.s. baselines of 1-GPU and model parallelism (MP) with 10 GPUs and 20 GPUs on ImageNet-1K.

5 Discussion

We discuss several issues we note regarding sensAI's *class parallelism* approach.

First, in *class parallelism*, we have a hard limitation: we can only achieve scalability up to the number of classes we have to classify. This may limit its usage in many cases. For example, it cannot distribute a CIFAR-10 inference task to over more than 10 machines (or GPUs). If the original problem is a binary classification itself, our method is not directly applicable.

Second, the aggregated number of parameters for the whole set of our binary/grouped classifiers maybe even be larger than the baseline model size (e.g. for CNNs on large dataset like ImageNet-1K). However, our main goal is to reduce inference latency (not FLOPs or model parameters) on single data piece. *Class parallelism* is a simple and effective approach for reducing inference latency on live data (i.e. single input data item), whereas other parallelism methods (e.g. data parallelism and model parallelism) cannot achieve. As we only do one-shot, class-specific pruning to reduce our binary/grouped model sizes, our model size reduction already outperforms state-of-the-art class-agnostic channel pruning techniques (Liu et al., 2019). We can potentially further reduce our binary/grouped model sizes by incorporating quantization (NVIDIA, 2020) or class-agnostic pruning methods (Liu et al., 2017; Han et al., 2015; 2016) for iterative pruning (Liu et al., 2019). We leave this as a future research direction.

Third, if the binary/grouped models have high variance in number of parameters, the in-parallel inference time using *class parallelism* will be determined by the binary/grouped model with most computation. Given binary model analysis in Sec.4.2.6 and the grouped models we designed to have even number of classes, we found that, our binary and grouped models mostly have similar number of parameters. This is possibly because we split original multi-classification task evenly to our binary/grouped classifiers. This feature of similar binary/grouped model sizes ensures that all the binary/grouped classifiers involved during in-parallel inference can finish model serving at roughly the same time.

Fourth, regarding the baseline, in most cases, the best performance of single image serving time is achieved when using single GPU. This is because transmitting intermediate results across GPUs can be expensive, and traditional model parallelism introduces hard synchronization barriers in our single-image serving scenario. As we focus on per-image serving case, data parallelism (Goyal et al., 2017; Wang et al., 2020) and pipeline parallelism (Huang et al., 2019; Narayanan et al., 2019) are not applicable here to mitigate serving latency. And our experimental results verify that, even though baseline with model parallelism can reduce the model size and FLOPs consumption on each device, the end-to-end latency of single image inference is much higher than using single GPU. The reason that sensAI can decouple and distribute the baseline model to multiple GPUs while achieving lower per-image inference latency is, we remove all possible communication overheads and synchronization barriers so that all our decoupled subnets can run in-parallel without blocking each other.

6 Conclusion

In sensAI, we propose a novel in-parallel model serving mechanism, called *class parallelism*, that enables fast model serving on live data (i.e. single data item). Experiments on CIFAR-10 dataset demonstrate its effectiveness: by pulling out binary classifiers from base model using class-specific pruning and retraining, sensAI achieves time reduction of per-image model serving by 6x on VGG-19, and 2x among ResNet-164, MobileNet-V2 and ShuffleNet-V2 over single GPU baseline. Compared with model parallel baseline using same amount of GPUs, sensAI achieves up to 18x time reduction for single image model serving. Similar results are achieved on CIFAR-100 and ImageNet-1K datasets with our grouped classifiers.

Acknowledgements

Guanhua Wang and Ion Stoica are supported by NSF CISE Expeditions Award CCF-1730628. This research is also supported by gifts from Amazon Web Services, Ant Group, CapitalOne, Ericsson, Facebook, Futurewei, Google, Intel, Microsoft, Nvidia, Scotiabank, Splunk and VMware.

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., and Zheng, X. Tensorflow: A system for large-scale machine learning. In *USENIX OSDI*, 2016.

Anand, R., Mehrotra, K., Mohan, C. K., and Ranka, S. Efficient classification for multiclass problems using modular neural networks. *IEEE Transactions on Neural Networks*, 6(1):117–124, 1995.

Ba, J. and Caruana, R. Do deep nets really need to be deep? In *NeurIPS*, 2014.

Badue, C., Guidolini, R., Carneiro, R. V., Azevedo, P., Cardoso, V. B., Forechi, A., Jesus, L., Berriel, R., Paixao, T., Mutz, F., Veronese, L., Oliveira-Santos, T., and Souza, A. F. D. Self-driving cars: A survey. In *arXiv:1901.04407*, 2019.

Beygelzimer, A., Langford, J., and Zadrozny, B. Weighted one-against-all. In *AAAI*, 2005.

Beygelzimer, A., III, H. D., Langford, J., and Mineiro, P. Learning reductions that really work. *Proceedings of the IEEE*, 104(1):136–147, 2016.

Chen, T., Li, M., Li, Y., Lin, M., Wang, N., Wang, M., Xiao, T., Xu, B., Zhang, C., and Zhang, Z. MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems. *arXiv preprint arXiv:1512.01274*, 2015.

Dean, J., Corrado, G. S., Monga, R., Chen, K., Devin, M., Le, Q. V., Mao, M. Z., Ranzato, M., Senior, A., Tucker, P., Yang, K., and Ng, A. Y. Large scale distributed deep networks. In *NeurIPS*, 2012.

Deng, H., Stathopoulos, G., and Suen, C. Y. Applying error-correcting output coding to enhance convolutional neural network for target detection and pattern recognition. In *International Conference on Pattern Recognition*, 2010.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009.

Denton, E. L., Zaremba, W., Bruna, J., LeCun, Y., and Fergus, R. Exploiting linear structure within convolutional networks for efficient evaluation. In *NeurIPS*, 2014.

Dietterich, T. G. and Bakiri, G. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2:263–286, 1995.

Galara, M., Fernandez, A., Barrenechea, E., Bustince, H., and Herrera, F. An overview of ensemble methods for binary classifiers in multi-class problems: Experimental study on one-vs-one and one-vs-all schemes. *Pattern Recognition*, 44:1761–1776, 2011.

Girshick, R., Donahue, J., Darrell, T., and Malik, J. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014.

Goyal, P., Dollar, P., Girshick, R., Noordhuis, P., Wesolowski, L., Kyrola, A., Tulloch, A., Jia, Y., and He, K. Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. *arXiv preprint arXiv:1706.02677*, 2017.

Gu, J., Chowdhury, M., Shin, K. G., Zhu, Y., Jeon, M., Qian, J., Liu, H., and Guo, C. Tiresias: A gpu cluster manager for distributed deep learning. In *USENIX NSDI*, 2019.

Han, S., Pool, J., Tran, J., and Dally, W. Learning both weights and connections for efficient neural network. In *NeurIPS*, 2015.

Han, S., Mao, H., and Dally, W. J. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In *ICLR*, 2016.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *CVPR*, 2016.

Horwitz, J. Goldman: AI tools have potential in finance beyond smart stock trading. `https://bit.ly/3ckkX3A`, 2020.

Hu, H., Peng, R., Tai, Y.-W., and Tang, C.-K. Network Trimming: A Data-Driven Neuron Pruning Approach towards Efficient Deep Architectures . *arXiv preprint arXiv:1607.03250*, 2017.

Hu, J., Shen, L., and Sun, G. Squeeze-and-excitation networks. In *CVPR*, 2018.

Huang, Y., Cheng, Y., Bapna, A., Firat, O., Chen, M. X., Chen, D., Lee, H., Ngiam, J., Le, Q. V., Wu, Y., and Chen, Z. Gpipe: Efficient training of giant neural networks using pipeline parallelism. In *NeurIPS*, 2019.

Jayarajan, A., Wei, J., Gibson, G., Fedorova, A., and Pekhimenko, G. Priority-based parameter propagation for distributed dnn training. In *MLSys*, 2019.

Jeaugey, S. Optimized inter-GPU collective operations with NCCL 2. `https://developer.nvidia.com/nccl`, 2017.

Jennings, D. G. Goldman Sachs Gambles Big in AI. `https://bit.ly/3jNLkCj`, 2018.

Jia, Z., Zaharia, M., and Aiken, A. Beyond data and model parallelism for deep neural networks. In *MLSys*, 2019.

Kim, J. K., Ho, Q., Lee, S., Zheng, X., Dai, W., Gibson, G. A., and Xing, E. P. Strads: A distributed framework for scheduled model parallel machine learning. In *ACM EuroSys*, 2016.

Kong, E. B. and Dietterich, T. G. Error-correcting output coding corrects bias and variance. In *ICML*, 1995.

Krizhevsky, A. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *NeurIPS*, 2012.

Lee, S., Kim, J. K., Zheng, X., Ho, Q., Gibson, G. A., and Xing, E. P. On model parallelization and scheduling strategies for distributed machine learning. In *NeurIPS*, 2014.

Lepikhin, D., Lee, H., Xu, Y., Chen, D., Firat, O., Huang, Y., Krikun, M., Shazeer, N., and Chen, Z. Gshard: Scaling giant models with conditional computation and automatic sharding. In *ICLR*, 2021.

Levy-Kramer, J. and Klaber, M. k-means-constrained 0.4.3. https://pypi.org/project/k-means-constrained/, 2020.

Li, H., Kadav, A., Durdanovic, I., Samet, H., and Graf, H. P. Pruning filters for efficient convnets. In *ICLR*, 2017.

Li, L., Jamieson, K., Rostamizadeh, A., Gonina, E., Ben-tzur, J., Hardt, M., Recht, B., and Talwalkar, A. A system for massively parallel hyperparameter tuning. In *MLSys*, 2020.

Li, M., Andersen, D. G., Park, J. W., Smola, A. J., Ahmed, A., Josifovski, V., Long, J., Shekita, E. J., and Su, B.-Y. Scaling distributed machine learning with the parameter server. In *USENIX OSDI 2014*, 2014.

Lin, T.-Y., Maire, M., Belongie, S., Bourdev, L., Girshick, R., Hays, J., Perona, P., Ramanan, D., Zitnick, C. L., and Dollar, P. Microsoft COCO: Common Objects in Context. *arXiv preprint arXiv:1405.0312*, 2015.

Linderman, G. C. and Steinerberge, S. Clustering with t-sne, provably. *SIAM Journal on Mathematics of Data Science*, 1, 2019.

Liu, Z., Li, J., Shen, Z., Huang, G., Yan, S., and Zhang, C. Learning efficient convolutional networks through network slimming. In *ICCV*, 2017.

Liu, Z., Sun, M., Zhou, T., Huang, G., and Darrell, T. Rethinking the value of network pruning. In *ICLR*, 2019.

Ma, N., Zhang, X., Zheng, H.-T., and Sun, J. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *CVPR*, 2018.

MacQueen, J. B. Some methods for classification and analysis of multivariate observations. In *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pp. 281–297. University of California Press, 1967.

Malinen, M. I. and Franti, P. Balanced k-means for clustering. In *Proceedings of the Joint IAPR International Workshop on Structural, Syntactic, and Statistical Pattern Recognition*, 2014.

Narayanan, D., Harlap, A., Phanishayee, A., Seshadri, V., Devanur, N. R., Ganger, G. R., Gibbons, P. B., and Zaharia, M. Pipedream: Generalized pipeline parallelism for dnn training. In *ACM SOSP*, 2019.

NVIDIA. TensorRT. https://developer.nvidia.com/tensorrt, 2020.

Or, A., Zhang, H., and Freedman, M. J. Resource elasticity in distributed deep learning. In *MLSys*, 2020.

Paden, B., Cap, M., Yong, S. Z., Yershov, D., and Frazzoli, E. A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Transactions on Intelligent Vehicles*, 1(1):33–55, 2016.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. Automatic differentiation in PyTorch. In *NeurIPS*, 2017.

Peng, Y., Zhu, Y., Chen, Y., Bao, Y., Yi, B., Lan, C., Wu, C., and Guo, C. A generic communication scheduler for distributed dnn training acceleration. In *ACM SOSP*, 2019.

Porzecanski, K. JPMorgan Commits Hedge Fund to AI in Technology Arms Race. https://bloom.bg/2P1JUpF, 2019.

Real, E., Aggarwal, A., Huang, Y., and Le, Q. V. Regularized Evolution for Image Classifier Architecture Search. In *AAAI*, 2019.

Ren, S., He, K., Girshick, R., and Sun, J. Faster r-cnn: Towards real-time object detection with region proposal networks. In *NeurIPS*, 2015.

Rifkin, R. and Klautau, A. In defense of one-vs-all classification. *Journal of Machine Learning Research*, 5: 101–141, 2004.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 2015.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, 2018.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

van der Maaten, L. and Hinton, G. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9, 2008.

Wang, G., Venkataraman, S., Phanishayee, A., Thelin, J., Devanur, N., and Stoica, I. Blink: Fast and Generic Collectives for Distributed ML. In *MLSys*, 2020.

Yang, W. pytorch-classification on CIFAR10/100 and ImageNet. https://github.com/bearpaw/pytorch-classification, 2017.

Yu, F., Qin, Z., and Chen, X. Distilling Critical Paths in Convolutional Neural Networks. In *NeurIPS CDNNRIA workshop*, 2018.

Yu, P. and Chowdhuryo, M. Salus: Fine-grained gpu sharing primitives for deep learning applications. In *MLSys*, 2020.

Zhou, B., Sun, Y., Bau, D., and Torralba, A. Revisiting the importance of individual units in cnns via ablation. *arXiv preprint arXiv:1806.02891*, 2018.

A K-MEANS CLUSTERING OVER T-SNE PROJECTED SPACE

After t-SNE (van der Maaten & Hinton, 2008) preprocessing on training images, we conduct k-means clustering (MacQueen, 1967) over t-SNE's output feature representations (FR), in order to get proper group arrangement for each class.

As summarized in Algorithm 1, we first calculate all the mean feature-representation per-class (A^c) over t-SNE output (X^c). Then we feed these mean FR per-class as the input data points for k-means clustering algorithm.

Algorithm 1 K-means clustering over t-SNE output

```
Input: total number of classes: $N$, number of groups: $k$,
       t-SNE output: $\{x_i^c\} \in X^c$ where c={0,1,..N-1}
for $m$ = 0 to N-1 do
    $a^m \leftarrow \sum_{n=0}^{|X^m|-1} x_n^m / \mid X^m \mid$; // Avg FR per class
end
Data: $\{a^c\} \in A^c$ for c={0,1,..N-1}, $Min_{size} \leftarrow \lfloor N/k \rfloor$,
      $Max_{size} \leftarrow \lceil N/k \rceil$
if $Min_{size} == Max_{size}$ then
    kmeans-balanced($A^c, N/k$)
else
    kmeans-constrained($A^c, Min_{size}, Max_{size}$)
end
```

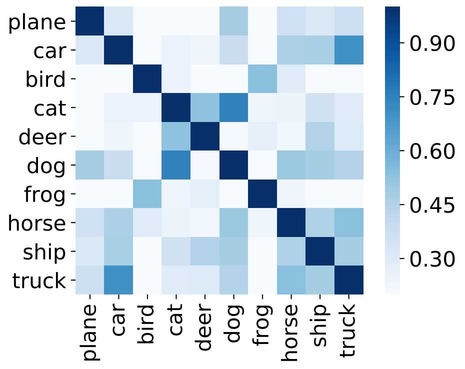
Here we may have two cases, if the number of classes N can be divided by the number of groups k, we can directly use *kmeans-balanced()* algorithm (Malinen & Franti, 2014). If the number of class is not divisible by the number of groups, we adopt another variation of k-means clustering algorithm called *kmeans-constrained()* (Levy-Kramer & Klaber, 2020) by passing-in Min_{size}, Max_{size} into the function, where $(Max_{size} - Min_{size}) \leq 1$ as we split the classes as evenly as possible (Section 3.2.2).

B IOU SIMILARITY AMONG BINARY CLASSIFIERS ON CIFAR-10

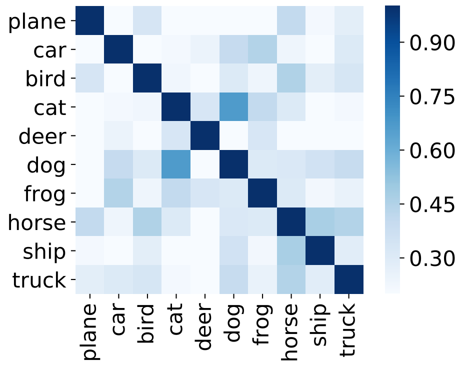
We measure the Intersection over Union (IoU) similarity comparison among the whole set of binary classifiers pulled from different models as ResNet-164, MobileNet-V2 and ShuffleNet-V2 on CIFAR-10 dataset.

Figure 9 depicts the IoU similarity for all three different kinds of models as ResNet-164 (Figure 9(a)), ShuffleNet-V2 (Figure 9(b)) and MobileNet-V2 (Figure 9(c)). They all show that the relatvent bianry classifiers we get from the original base model share high IoU similarity.

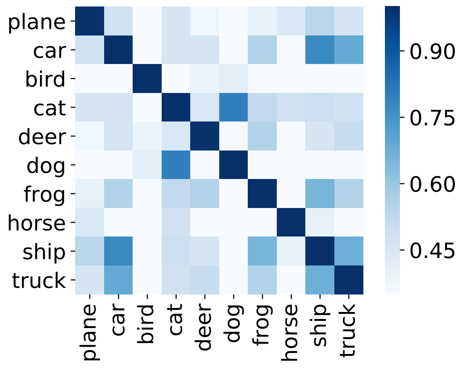
In Figure 9(a), the binary classifiers pulled from ResNet-164 indicates that, not only cat and dog binary classifiers



(a) ResNet-164



(b) ShuffleNet-V2



(c) MobileNet-V2

Figure 9. IoU similarity comparison of ResNet-164, ShuffleNet-V2 and MobileNet-V2 on CIFAR-10.

share high similarity, but also car and truck. Both pairs are intuitive since these two pairs also look similar from human perspective. Compared with ResNet-164, the binary classifiers collected from ShuffleNet-V2 in Figure 9(b) generally share less neurons. However, the cat-dog pair still share the highest similarity among the whole set of its binary classifiers. For binary classifiers getting from MobileNet-V2, in Figure 9(c), besides high similarity of cat-dog and car-truck pairs, car and ship may also be highly correlated. This is possibly due to the fact that they are all chunky and share similar cube-like shape from certain angles.

C Model training via Class Parallelism

C.1 Method

Training binary/grouped model in parallel is slightly different from class-parallel inference. Since we do not have a pre-trained base model, we need to first train the original N-way model for a certain number of epochs to allow the base model to learn the feature representations from the training data. Only based on this "half-baked" model we could distill subnets for different classes.

On this half-baked model, we conduct the same process to collect feature-maps from each neuron (i.e. channel) by passing per-class/per-group images as input. Then for each class or class group, based on neuron activation statistics, we use activation-based pruning metrics to prune the base model and get our binary/grouped ones. We then distribute each binary/grouped model onto a single GPU and train these subnets in parallel using the class-specific dataset we generated from the original training dataset. Once all the subnets training are done, we use the same procedure to combine these models' output for the original multi-way inference.

Note that in this class-parallel training process, the half-baked model is only trained using N-way full dataset for a small number of epochs, and for the rest of the training epochs, we employ in-parallel binary/grouped model training with *zero* communication needed. For example, for training on the CIFAR-10 dataset, the baseline models are typically trained for $\sim$164 epochs with proper learning rate decay recipes (Yang, 2017). We empirically train the half-baked base model for only 20 epochs, from which we distill our binary models. And the rest 144 epochs are trained with our class parallelism (i.e., each binary or grouped classifier is trained on one GPU concurrently at no communication cost). Since each binary/grouped classifier has much less computation consumption than the base model, we still achieve training speedup without model synchronization via class parallelism. After finishing training binary/grouped classifiers in parallel, we can use the same method to combine all the binary/grouped classifiers results for the original N-way classification results.

C.2 CIFAR-10 Evaluation

We evaluate in-parallel training via class parallelism using VGG-19 and ResNet-164 on CIFAR-10 dataset. For pulling out binary models from a half-baked 10-way model, we first conduct regular training of baseline 10-way model from scratch on full dataset for 20 epochs. Given that model converges fastest during the first tens of epochs, we collect the per-class feature-maps of each neuron over this half-baked model and use these feature-maps statistics for hybrid pruning to get our binary models.

Note that normally for training either VGG-19 or ResNet-164 to reach a decent test accuracy, it takes around 164 epochs (Yang, 2017). Since we train the full model for 20 epochs at first, there are still 164-20 = 144 epochs of training budget remaining. We measure per-iteration training time on base model and maximum of per-iteration training time from our binary models. Since first 20 epochs are the same for both `sensAI` and baseline model training, we report the total training time after 20 epochs to make a direct comparison of training speed between `sensAI` and baseline single GPU training.

We use 64 as mini-batch size for training. For `sensAI` experiments, without losing inference accuracy, we use binary models with $\sim$1M parameters for VGG-19, and $\sim$0.16M size binary models for ResNet-164. After 20 epochs of baseline model pre-training, we retrain the pruned binary models for the rest 144 epochs. Total training time reduction on VGG-19 is **11x**, which is **over linear scalibility**. The main reason for this is we reduce the model size to 1/20. It saves computation time in both forward and backward propagation, which leads to reduction in per-iteration time. On ResNet-164, we reduce training time by 3x. As discussed in Section 4.2.2, the model depth and skip layer connections are the dominant factors for its long training time.

We also note this may not be a fair comparison since `sensAI` uses 10 GPUs whereas baseline only uses 1 GPU. The main point is, we can incorporate `sensAI` to *further* boost up the training speed in data parallelism. More specifically, for each model replica involved in original data parallel training, we can further distribute it to multiple GPUs for concurrent, zero-communication class parallelism training.

After training, we also evaluate the number of parameters, FLOPs, and inference time during our class-parallel model serving. In general, comparing with binary model getting from fully-trained base-model, the binary models pulling out from half-baked model have lower top-1 test accuracy (decrease in the the range from 0.04% to 0.38%). Additionally, the size of binary models pulling out from half-baked base model is not evenly distributed, some of the binary models maybe 1.2x$\sim$2x the size of some other smaller binary models. It may due to the fact that the base model is not fully trained. Thus the half-baked model generates more noisy pruning signals such that we may only be able to get less efficient binary models. However, even in the case that some binary models are 1.2x$\sim$2x times larger than others, our end-to-end in parallel inference still has 2$\sim$5x speed up over the baseline of either single GPU or model parallelism.

The key reason is the binary models are already very tiny, and double the size does not make much difference on its end-to-end forward propagation time for live data inference.

D Code Repository

`sensAI` is an open-source project. The corresponding GitHub repository link is here: `https://github.com/GuanhuaWang/sensAI`

Since our approach is quite general and should be able to apply to other domains like Reinforcement Learning (RL), Natural Language Processing (NLP) models, we are willing to help if people need to "customize" it with their own input and model (Support team `sensaiworld.slack.com` or Email `guanhua@cs.berkeley.edu`).

We are now working on extending our ideas to semantic segmentation tasks in computer vision, control systems on drones with RL models, etc. We also explore adding fault tolerance feature on top of our approach and published a short paper here: `https://bit.ly/3e2tDOO`

Last, we would like to thank all our main contributors at UC Berkeley: Kenan Jiang, Kehan Wang, Jichan Chung, Fangyu Wu, Balaji Veeramani, Yaoqing Yang, Adarsh Karnati, Zihao Fan, Hank O'Brien, Yingxin Kang, Sahil Rao, for their efforts to this code base and help with debugging on AWS EC2 performance anomalies. We are also indebted to Kannan Ramchandran, Fisher Yu (ETH), Zhewei Yao, Gur-Eyal Sela, Vipul Gupta, Dequan Wang for their insightful comments on our previous drafts of the paper.