SplitNet: Learning to Semantically Split Deep Networks for Parameter Reduction and Model Parallelization

Juyong Kim [*1] **Yookoon Park** [*1] **Gunhee Kim** [1] **Sung Ju Hwang** [2 3]

Abstract

We propose a novel deep neural network that is both lightweight and effectively structured for model parallelization. Our network, which we name as *SplitNet*, automatically learns to split the network weights into either a set or a hierarchy of multiple groups that use disjoint sets of features, by learning both the class-to-group and feature-to-group assignment matrices along with the network weights. This produces a tree-structured network that involves no connection between branched subtrees of semantically disparate class groups. SplitNet thus greatly reduces the number of parameters and required computations, and is also embarrassingly model-parallelizable at test time, since the evaluation for each subnetwork is completely independent except for the shared lower layer weights that can be duplicated over multiple processors, or assigned to a separate processor. We validate our method with two different deep network models (ResNet and AlexNet) on two datasets (CIFAR-100 and ILSVRC 2012) for image classification, on which our method obtains networks with significantly reduced number of parameters while achieving comparable or superior accuracies over original full deep networks, and accelerated test speed with multiple GPUs.

1. Introduction

Recently, deep neural networks have shown impressive performances on a multitude of tasks, including visual recognition (Krizhevsky et al., 2012; Szegedy et al., 2015; He et al., 2016), speech recognition (Hinton et al., 2012), and natural language processing (Bengio et al., 2003; Sutskever et al., 2014). However, such remarkable performances are achieved at the cost of increased computational complexity at both training and test time compared to traditional machine learning models including shallow neural networks. This increased complexity of deep networks can be problematic if the model and the task size becomes very large (*e.g.* classifying tens of thousands of object classes), or the application is time-critical (*e.g.* real-time object detection).

There exist various solutions to tackle this complexity issue. One way is to reduce the number of model parameters; this could be achieved either by training a new smaller network while maintaining similar behavior as the original network (Ba & Caruana, 2014; Hinton et al., 2014), or by disconnecting unnecessary weights through pruning or sparsity regularization (Reed, 1993; Han et al., 2015; Collins & Kohli, 2014; Wen et al., 2016; Alvarez & Salzmann, 2016). Another approach to speed up deep networks is distributed machine learning (Dean et al., 2012; Chilimbi et al., 2014; Zhang et al., 2015); however most research effort has been made on the systems and optimization sides, without consideration of the ways to obtain network structure that is intrinsically scalable.

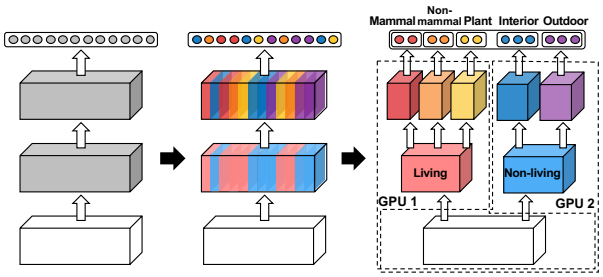
In this work, we aim to learn a deep neural network that not only reduces the number of model parameters, but also is effectively structured for model parallelization. How can we then achieve these seemingly disjoint goals in a single, unified learning framework? We focus on the observation that as the number of classes increases, semantically disparate classes (or tasks) can be represented by largely disjoint sets of features. For example, features describing *animals* may be quite different from those describing *vehicles*, whereas low-level features such as stripes, dots, and colors are likely be shared across all classes. It implicates that we can cluster classes into mutually exclusive groups based on the features they use. Such grouping of concepts based on semantic proximity also agrees with the way that our brain stores semantic concepts, where semantically related concepts activate similar part of the brain (Huth et al., 2012), in a highly localized manner.

Based on this intuition, we propose a novel deep network architecture named *SplitNet* that automatically performs

[*]Equal contribution [1]Seoul National University, Seoul, South Korea [2]UNIST, Ulsan, South Korea [3]AITrics, Seoul, South Korea. Correspondence to: Sung Ju Hwang <sjhwang@unist.ac.kr>.

Proceedings of the 34^{th} International Conference on Machine Learning, Sydney, Australia, PMLR 70, 2017.

Codes available at http://vision.snu.ac.kr/projects/splitnet.



(a) Base Network (b) Splitting Deep Neural Network (c) Our Network(*SplitNet*)

Figure 1. **Concept**. Our network automatically learns to split the classes and associated features into multiple groups at multiple network layers, obtaining a tree-structured network. Given a base network in (a), (b) our algorithm optimizes network weights as well as class-to-group and feature-to-group assignments. Colors indicate the group assignments of classes and features. (c) After learning to split the network, the model can be distributed to multiple GPUs to accelerate training and inference.

deep splitting of the network into a set of subnetworks or a hierarchy of subnetworks sharing common lower layers, such that classes in each group share a common subset of features, that is completely disjoint from the features for other class groups. We train such a network by additionally learning classes-to-group assignments and feature-to-group assignments along with the network weights, while regularizing them to be disjoint across groups. Figure 1 illustrates the key idea of our proposed approach.

Our splitting algorithm is fairly generic, and is applicable to any general deep neural networks including feed-forward and convolutional networks. We test our method with ResNet (He et al., 2016; Zagoruyko & Komodakis, 2016) and AlexNet (Krizhevsky et al., 2012) on CIFAR-100 and ILSVRC 2012 datasets, on which our networks respectively achieve 32% and 56% parameter reduction, while obtaining comparable or even superior performance to the full base network. We further perform test-time parallelization of our network with multiple GPUs, where it achieved $1.73\times$ speedup to naive parallelization method with 2 GPUs. Our contributions in this work are threefold.

- We propose a novel deep neural network named *SplitNet*, which is organized as a tree of disjoint subnetworks with greatly reduced the number of parameters and computations in terms of FLOPs, that obtains comparable accuracies to fully-connected networks.
- We propose an efficient algorithm for automatically training such a tree-structured network, which learns *class-to-group* and *feature-to-group* assignments along with the network weights.
- The networks trained by our approach are embarrassingly model-parallelizable; in distributed learning setting, we show that our networks scale well to an increased number of processors.

2. Related Work

Parameter reduction for deep neural networks. Achieving test-time efficiency is an active research topic in deep learning. One straightforward approach is to remove weak connections during the training, usually implemented using the ℓ_1-norm (Collins & Kohli, 2014). However, the ℓ_1-norm often results in a model that trades-off the accuracy with the efficiency. Han et al. (2015) presented an iterative weight pruning technique that repeatedly retrains the network while removing of weak connections, which achieves a superior performance over ℓ_1-regularization. Recently, the group sparsity using $\ell_{2,1}$-norm has been explored for learning a compact model. Alvarez & Salzmann (2016) applied (2,1)-norm regularization at each layer to eliminate the hidden units that are not shared across upper-level units, thus automatically deciding how many neurons to use at each layer. Wen et al. (2016) used the same group sparsity to select unimportant channels and spatial features in a CNN, and let the network to automatically decide how many layers to use. However, they assume that all classes share the same set of features, which is restrictive when the number of classes is very large. On the contrary, our proposed SplitNet enforces feature sharing only within a group of related classes, and thus semantically reduce the number of parameters and computations for large-scale problems. Recently, Shankar et al. (2016) also addressed the use of symmetrical split at mid-level convolutional layers in CNNs for architecture refinement. However, they did not learn the splits but predefine them, as opposed to SplitNet which learns semantic splits along with network weights.

Parallel and distributed deep learning. As deep networks and training data become increasingly larger, researchers are exploring parallelization and distribution techniques to speed up the training process. Most parallelization techniques exploit either 1) data parallelism, where the training data is distributed across multiple computational nodes, or 2) model parallelism, where the model parameters are distributed. Dean et al. (2012) used both data and model parallelism to train a large-scale deep neural network on a computing cluster with thousands of machines. For CNNs, Krizhevsky et al. (2012) used both data and model parallelism to train separate convolutional filters from disjoint datasets. Krizhevsky (2014) later proposed to vertically split the network, exploiting the different time/memory characteristics of the convolutional and fully connected layers. Chilimbi et al. (2014) proposed a server-client architecture where each client computes partial gradients of parameters, that are stored and communicated from the global model parameter server. Zhang et al. (2015) proposed a GPU-based distributed deep learning, with greatly reduced the communication overheads. However, all these are systems-based approaches that work under the assumption that the model structure is given and

fixed. Our approach, on the other hand, leverages semantic knowledge of class relatedness to learn a network structure that is well-fitted to a distributed machine learning setting.

Tree-structured deep networks. There have been some efforts to exploit hierarchical class structures for improving the performance of deep networks. To list a few recent work, Warde-Farley et al. (2014) proposed to group classes based on their weight similarity, and augmented the original deep network with the softmax loss for fine-grained classification for classifying classes within each group. Yan et al. (2015) proposed a convolutional network that combines predictions from separate sub-networks for coarse- and fine-grained category predictions, which share common lower-layers. Goo et al. (2016) exploited the hierarchical structure among the classes to learn common and discriminative features across classes, by adding in simple feature pooling layers. Murdock et al. (2016) generalized dropout to stochastically assign nodes to clusters which results in obtaining a hierarchical structure. However, all of these methods focus on improving the model accuracy, at the expense of increased computational complexity. On the contrary, SplitNet is focused on improving memory/time efficiency and parallelization performance of the model.

3. Approach

Given a base network, our goal is to obtain a tree-structured network that contains either a set or a hierarchy of subnetworks, where the leaf-level subnetworks are associated with a specific group of classes, as in Figure 1. Then what is the optimal way to split the classes? A key insight to our approach is that classes within each group should share features as much as possible, since grouping of disparate classes may result in learning redundant features over multiple groups, and may waste network capacity as a result. Thus, to maximize the utility of this splitting process, we need to cluster classes together into groups so that each group uses a subset of features that are completely disjoint from the ones used by other groups. One straightforward way to obtain such mutually exclusive groupings of classes is to leverage a semantic taxonomy, since semantically similar classes are likely to share features, whereas dissimilar classes are unlikely to do so. However, in practice, such semantic taxonomy may not be available, or may not agree with actual hierarchical grouping based on what features each class uses. Another simple approach is to perform (hierarchical) clustering on the weights learned in the original network, which is also based on actual feature uses. Yet, this grouping may be still suboptimal since the groups are highly likely to overlap, and is inefficient since it requires training the network twice.

In the following sections, we will describe how to learn such groups with disjoint set of classes and features, along with the network weights in a deep learning framework.

3.1. Problem Statement

Given a dataset $\mathcal{D} = \{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^N$, where $\boldsymbol{x}_i \in \mathbb{R}^d$ is an input data instance and $y_i \in \{1, \dots, K\}$ is a class label for K classes, our goal is to learn a network whose weight at each layer l, $\boldsymbol{W}^{(l)}$ is a block-diagonal matrix, where each block $\boldsymbol{W}_g^{(l)}$ is associated with a class group $g \in \mathcal{G}$, where $\mathcal{G}$ is the set of all groups. Such a block-diagonal $\boldsymbol{W}^{(l)}$ ensures that each disjoint group of classes has exclusive features associated with it, such that no other groups use those features; this allows the network to be split across multiple class groups, for faster computation and parallelization.

In order to obtain such a block diagonal weight matrix $\boldsymbol{W}^{(l)}$, we propose a novel splitting algorithm that learns a feature-group and class-group assignment along with the network weights. We first illustrate our splitting method on the parameters for the softmax classifier in Section 3.2, and then describe how to extend this to other layers of DNNs in Section 3.3.

We assume that the number of groups G, is given. Let p_{gi} be a binary variable indicating whether feature i is assigned to group g ($1 \leq g \leq G$), and q_{gj} be a binary variable indicating whether class j is assigned to group g. We define $\boldsymbol{p}_g \in \mathbb{Z}_2^D$ as a feature group assignment vector for group g, where $\mathbb{Z}_2 = \{0, 1\}$ and D is the dimension of features. Similarly, $\boldsymbol{q}_g \in \mathbb{Z}_2^K$ denotes a class group assignment vector for group g. That is, $\boldsymbol{p}_g$ and $\boldsymbol{q}_g$ define a group g together, where $\boldsymbol{p}_g$ represents features associated with the group and $\boldsymbol{q}_g$ indicates a set of classes assigned to the group.

We assume that there is no overlap between groups, either in features or classes, i.e. $\sum_g \boldsymbol{p}_g = \mathbb{1}_D$ and $\sum_g \boldsymbol{q}_g = \mathbb{1}_K$, where $\mathbb{1}_D$ and $\mathbb{1}_K$ are the vectors with all-one elements. While this assumption imposes hard regularizations on group assignments, it enables the weight matrix $\boldsymbol{W} \in \mathbb{R}^{D \times K}$ to be sorted into a block-diagonal matrix, since each class is assigned to a group and each group depends on a disjoint subset of features. This greatly reduces the number of parameters, and at the same time, the multiplication $\boldsymbol{W}^\mathsf{T}\mathbf{x}$ can be decomposed into smaller and faster block matrix multiplications.

The objective for training our SplitNet is then defined as:

$$\min_{\omega, \boldsymbol{P}, \boldsymbol{Q}} \mathcal{L}(\omega, \boldsymbol{X}, \boldsymbol{y}) + \sum_{l=1}^{L} \lambda ||\boldsymbol{W}^{(l)}||_2^2 + \sum_{l=S}^{L} \Omega(\boldsymbol{W}^{(l)}, \boldsymbol{P}^{(l)}, \boldsymbol{Q}^{(l)}), \tag{1}$$

where $\mathcal{L}(\boldsymbol{W}, \boldsymbol{X}, \boldsymbol{y})$ is the cross entropy loss on the training data, $\omega = \{\boldsymbol{W}^{(1)}, \dots, \boldsymbol{W}^{(L)}\}$ is the set of network weights at all layers, $\|\boldsymbol{W}\|_2^2$ is the weight decay regularizer with a hyperparameter λ, S is the layer where splitting starts, $\Omega(\boldsymbol{W}, \boldsymbol{P}, \boldsymbol{Q})$ is the regularizer for splitting the network, and $\boldsymbol{P}^{(l)}$ and $\boldsymbol{Q}^{(l)}$ are the set of feature-to-group and class-

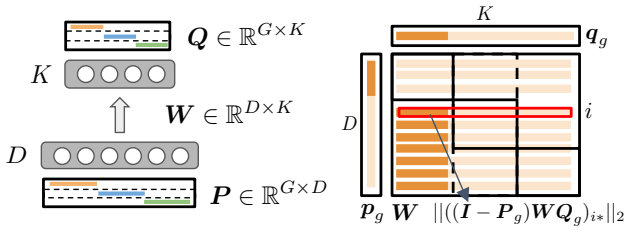


Figure 2. **Group Assignment and Group Weight Regularization**. (**Left**) An example of group assignment with $G = 3$. Colors indicate groups. Each row of matrix $\boldsymbol{P}, \boldsymbol{Q}$ is group assignment vectors for group g: $\boldsymbol{p}_g$, $\boldsymbol{q}_g$. (**Right**) Visualization of matrix $(\boldsymbol{I} - \boldsymbol{P}_g)\boldsymbol{W}\boldsymbol{Q}_g$. The group assignment vectors work as soft indicators for inter-group connections. As the groupings converge, $\ell_{2,1}$-norm is concentrated on inter-group connections.

to-group assignment vectors respectively, for each layer l.

In the next section, we propose a novel regularization Ω that automatically finds appropriate disjoint group assignments with no external semantic information. The objective of Eq.(1) are jointly optimized using (stochastic) gradient descent, starting with a full weight matrix, and an unknown group assignment. As we jointly optimize the cross entropy loss and the group regularization, our method automatically obtains appropriate grouping and prune out inter-group connections. Once the grouping is learned, the weight matrix can be explicitly *split* into block diagonal matrices to reduce number of parameters, which in turn allows for much faster inference at test time.

3.2. Learning to Split Network Weights into Disjoint Groups

Our regularization assigns features and classes into disjoint groups; it consists of three objectives as follows:

$$\begin{aligned}\Omega(\boldsymbol{W}, \boldsymbol{P}, \boldsymbol{Q}) = \gamma_1 R_W(\boldsymbol{W}, \boldsymbol{P}, \boldsymbol{Q}) \\ + \gamma_2 R_D(\boldsymbol{P}, \boldsymbol{Q}) + \gamma_3 R_E(\boldsymbol{P}, \boldsymbol{Q})\end{aligned} \tag{2}$$

where $\gamma_1, \gamma_2, \gamma_3$ controls the strength of each regularization, which will be discussed in following subsections.

3.2.1. Group Weight Regularization

To make the numerical optimization tractable, we first relax the binary variables p_{gi} and q_{gj} to have real values in the interval of $[0, 1]$ with the constraints $\sum_g \boldsymbol{p}_g = \mathbb{1}_D$ and $\sum_g \boldsymbol{q}_g = \mathbb{1}_K$. These sum-to-one constraints can be directly optimized using reduced gradient algorithm (Rakotomamonjy et al., 2008), which yields sparse solutions. Or, we can perform soft assignments, by reparamterizing p_{gi} and q_{gj} with unconstrained variables α_{gi} and β_{gj} in the softmax form:

$$p_{gi} = e^{\alpha_{gi}} / \sum_g e^{\alpha_{gi}}, \quad q_{gj} = e^{\beta_{gj}} / \sum_g e^{\beta_{gj}}. \tag{3}$$

We empirically observe that the softmax form results in more semantically meaningful grouping. However, the direct optimization of sum-to-one constraint often leads to faster convergence than the softmax formulation.

Let $\boldsymbol{P}_g = diag(\boldsymbol{p}_g)$ and $\boldsymbol{Q}_g = diag(\boldsymbol{q}_g)$ be the feature and class group assignment matrix for group g respectively. Then $\boldsymbol{P}_g\boldsymbol{W}\boldsymbol{Q}_g$ represents the weight parameters associated with group g, *i.e.* intra-group connections between features and classes. Since our goal is to prune out inter-group connections to obtain block-diagonal weight matrices, we minimize off block-diagonal entries as follows:

$$\begin{aligned}R_W(\boldsymbol{W}, \boldsymbol{P}, \boldsymbol{Q}) = \sum_g \sum_i ||((\boldsymbol{I} - \boldsymbol{P}_g)\, \boldsymbol{W}\, \boldsymbol{Q}_g))_{i*}||_2 \\ + \sum_g \sum_j ||(\boldsymbol{P}_g\, \boldsymbol{W}\, (\boldsymbol{I} - \boldsymbol{Q}_g))_{*j}||_2\end{aligned} \tag{4}$$

where $(\boldsymbol{M})_{i*}$ and $(\boldsymbol{M})_{*j}$ denote i-th row and j-th column of $\boldsymbol{M}$. Eq.(4) imposes row/column-wise $\ell_{2,1}$-norm on the inter-group connections. Figure 2 illustrates this regularization, where the portions of the weights to which the regularization is applied are colored differently. [1]

We observe that this regularization yields groups that are fairly similar to semantic groups. One caution is to avoid uniform initialization on group assignments, *i.e.* $\boldsymbol{p}_i = \mathbb{1}/G$, in which case the objective reduces to row/column-wise $\ell_{2,1}$-norm and some row/column weight vectors may *die out* before appropriate group assignments are obtained.

3.2.2. Disjoint Group Assignment

For the group assignment vectors to be completely mutually exclusive, they should be orthogonal; *i.e.* they should satisfy the condition $\boldsymbol{p}_i \cdot \boldsymbol{p}_j = 0$ and $\boldsymbol{q}_i \cdot \boldsymbol{q}_j = 0, \forall\, i \neq j$. We introduce an additional orthogonal regularization term:

$$R_D(\boldsymbol{P}, \boldsymbol{Q}) = \sum_{i<j} \boldsymbol{p}_i \cdot \boldsymbol{p}_j + \sum_{i<j} \boldsymbol{q}_i \cdot \boldsymbol{q}_j. \tag{5}$$

where the inequalities avoid the duplicative dot products.

3.2.3. Balanced Group Assignment

The disjoint group assignment objective in Eq.(5) alone may drive one group to dominate over all other groups; that is, one group includes all features and classes, while other groups do not. Therefore, we also constrain the group assignments to be balanced, by regularizing the squared sum of elements in each group assignment vector.

$$R_E(\boldsymbol{P}, \boldsymbol{Q}) = \sum_g \Big((\sum_i p_{gi})^2 + (\sum_j q_{gj})^2 \Big). \tag{6}$$

[1] When using group weight regularization followed by batch normalization, the weights tend to decrease their magnitudes while the scale parameters of BN layers increase. To prevent this effect, we use ℓ_2-normalized weights $\boldsymbol{W}/||\boldsymbol{W}||_2$, instead of $\boldsymbol{W}$ in R_W, or simply deactivate the scale parameters of BN layers.

Algorithm 1 Splitting Deep Neural Networks

Input: Number of groups G, layers to split $S \leq L$ and hyperparamaters $\gamma_1, \gamma_2, \gamma_3$
Initialize weights and group assignments
while groupings have not converged **do**
  Optimize the objective using SGD with a learning rate η
  $\mathcal{L}(\omega, \boldsymbol{X}, \boldsymbol{Y}) + \lambda \sum_{l=1}^{L} ||\mathbf{W}^{(l)}||_2^2 + \gamma_1 \sum_{l=S}^{L} R_W(\boldsymbol{W}^{(l)}, \boldsymbol{P}^{(l)}, \boldsymbol{Q}^{(l)})$
  $+\gamma_2 \sum_{l=S}^{L} R_D(\boldsymbol{P}^{(l)}, \boldsymbol{Q}^{(l)}) + \gamma_3 \sum_{l=S}^{L} R_E(\boldsymbol{P}^{(l)}, \boldsymbol{Q}^{(l)})$
end while
Split the network using the obtained group assignments and weight matrices
while validation accuracy improves **do**
  Optimize $\mathcal{L}(\omega, \boldsymbol{X}, \boldsymbol{Y}) + \lambda \sum_{l=1}^{L} ||\mathbf{W}^{(l)}||_2^2$ using SGD
end while

Due to the constraints $\sum_g \boldsymbol{p}_g = \mathbb{1}^D$ and $\sum_g \boldsymbol{q}_g = \mathbb{1}^K$, the objective of Eq.(6) is minimized when sums of elements in each group assignment vector are even; *i.e.* each group has identical number of elements. Since the dimensions of feature and class group assignment vectors may differ, we scale the two terms with appropriate weights. See Figure 4 to see the effect of balanced group regularization.

3.3. Splitting Deep Neural Networks

Our weight-splitting method in section 3.2 can be applied to deep neural networks (DNN), which has two types of layers: 1) the input and hidden layers that produce a feature vector for a given input, and 2) the output fully-connected (FC) layer on which the softmax classifier produces class probabilities. The output FC layer can be split by directly applying our method in section 3.2 on the output FC weight matrix $\boldsymbol{W}^{(L)}$. Our splitting framework can be further extended into deep splits, involving either multiple consecutive layers or recursive hierarchical group assignments. Algorithm 1 describes the deep splitting process.

3.3.1. Deep Split

The lower layers of a DNN learn low-level, generic representations, which are likely to be shared across all classes. The higher level representations, on the contrary, are more likely to be specific to the classes in a particular group. Therefore we do not split all layers but split layers down to S-th layer ($S \leq L$), while maintaining lower layers ($l < S$) to be shared across class groups.

Each layer consists of input and output nodes with the weights $\boldsymbol{W}^{(l)}$ that connects between them, and $\boldsymbol{P}_g^{(l)}$, $\boldsymbol{Q}_g^{(l)}$ for input-to-group and output-to-group assignments. Since the output nodes of each layer correspond to the input nodes of the next layer, the grouping assignment are shared as $\boldsymbol{q}_g^{(l)} = \boldsymbol{p}_g^{(l+1)}$. This enforces that no signal is passed across different groups of layers, so that forward and backward propagation in each group is independent from the processes in other groups. This allows the computations for each group to be parallelized, except for the softmax layer. The softmax layer includes a normalizing operation over all classes which requires aggregating logits over groups; however, during inference it suffices to identify the class with the maximum logit, which can be simply obtained by first identifying the class with maximum logit in each group, and then selecting the class with maximum logit among the identified group-specific maximums. This requires minimal communication overhead.

The objective function for deep split is the same with the weight-splitting method in section 3.2 (*i.e.* Eq.(1)–(2)), except for the previously explained alignment constraints.

When applied to CNNs, the proposed group splitting process can be performed in the same manner on the convolutional filters. Suppose that a weight of a convolutional layer is a 4-D tensor $\boldsymbol{W}_c \in \mathbb{R}^{M \times N \times D \times K}$, where M, N denote height and width of receptive fields and D, K denote the number of input and output convolutional filters. We reduce the 4-D weight tensor $\boldsymbol{W}_c$ into a 2-D matrix $\boldsymbol{W}'_c \in \mathbb{R}^{D \times K}$ by taking the root sum squared of elements over height and width dimensions of $\boldsymbol{W}_c$, i.e. $\boldsymbol{W}'_c = \{w'_{dk}\} = \{\sqrt{\sum_{m,n} w_{mndk}^2}\}$. Then the weight regularization objective for the convolutional weight is obtained by Eq.(4), using $\boldsymbol{W}'_c$ instead.

3.3.2. Hierarchical Grouping

There often exist natural semantic hierarchies of classes: for example, The group *dog* and the group *cat* are subgroups of *mammal*. We can easily extend our deep split method to obtain multi-level hierarchies of categories. Figure 1 shows an example of such a hierachical split.

Assume that the grouping branches at the l-th layer and the output nodes of the l-th layer are grouped by G supergroup assignment vectors $\boldsymbol{q}_g^{(l)}$ with $\sum_g \boldsymbol{q}_g^{(l)} = \mathbb{1}_D$. Suppose that in the next layer, for each supergroup $g \in \{1, ..., G\}$ there are corresponding S subgroup assignment vectors $\boldsymbol{p}_{gs}^{(l+1)}$ with $\sum_{s,g} \boldsymbol{p}_{gs}^{(l+1)} = \mathbb{1}_D$. As aforementioned, the input nodes to the l+1-th layer corresponds to the output nodes of l-th layer. By defining $\boldsymbol{p}_g^{(l+1)} = \sum_s \boldsymbol{p}_{gs}^{(l+1)}$, we can map subgroup assignments into corresponding supergroup assignments. This allows us to impose the constraint $\boldsymbol{q}_g^{(l)} = \boldsymbol{p}_g^{(l+1)}$ as in Deep Split.

3.4. Parallelization of SplitNet

Our learning algorithm produces a tree-structured network whose subnetworks have no inter-group connections. This results in an embarrassingly model-parallel network where we can simply assign the obtained subnetworks to each

Table 1. **Comparison of Test Errors According to Depths of Splitting (row) and Splitting Methods (column) on CIFAR-100.** Postfix S, C and R denote SplitNet variants – Semantic, Clustering and Random, respectively.

WRN-16-8 (BASELINE)						24.28
WRN-16-8 (DROPOUT)						24.52
METHOD	SPLIT DEPTH	G	SPLITNET-S	SPLITNET-C	SPLITNET-R	SPLITNET
FC SPLIT	1	4	23.80	23.72	24.30	24.26
SHALLOW SPLIT	6	2	24.46	24.54	25.46	**23.96**
DEEP SPLIT (DROPOUT)	11	2	25.04	26.04	27.12	24.62
HIER. SPLIT (DROPOUT)	11	2-4	24.92	25.98	26.78	24.80

Table 2. **Comparison of Parameter/Computation Reduction and Test Errors on CIFAR-100.**

NETWORK	PARAMS(10^6)	% REDUCED	FLOPS(10^9)	% REDUCED	TEST ERROR(%)
WRN-16-8 (BASELINE)	11.0	0.0	3.10	0.0	24.28
FC SPLIT	11.0	0.35	3.10	0.0	24.26
SHALLOW SPLIT	7.42	32.54	2.64	14.63	**23.96**
DEEP SPLIT (DROPOUT)	5.90	46.39	2.11	31.97	24.66
HIER. SPLIT (DROPOUT)	4.12	62.58	1.88	39.29	24.80

processor, or a machine. In our implementation, we consider two approaches for model parallelization: 1) Assigning both the lower-layers and group-specific upper layers to each node. At the test time the lower layers are not changed; thus this approach is acceptable, although it causes unnecessary redundant computations across the processors. 2) Assigning the lower layer to a separate processor. This eliminates redundancies in the lower layer but incurs communication overhead between the lower layer and the upper layers.

Training-time parallelization is currently done only at the finetuning step, after group assignments have been decided. We leave the parallelization from the initial network training stage as future work.

4. Experiments

Datasets. We validate our method for image classification tasks on two benchmark datasets.

1) CIFAR-100. The CIFAR-100 dataset contains 32×32 pixel images from 100 generic object classes. For each class, there are 500 images for training and 100 images for test. We set aside 50 images for each class from the training dataset as a validation set for cross-validation. Note that the test errors are not directly comparable to (Zagoruyko & Komodakis, 2016), as we use only 45,000 training images.

2) ImageNet-1K. The ImageNet 1K dataset (Deng et al., 2009) that consists of 1.2 million images from $1,000$ generic object classes. For each class, there are $1K - 1.3K$ images for training and 50 images for validation, which we use for test, following the standard procedure.

Baselines. To compare different ways to obtain grouping, we test multiple variants of our SplitNet and baselines.

1) Base Network. Base networks with full network weights. For experiments on the CIFAR-100, we use Wide Residual Network (WRN) (Zagoruyko & Komodakis, 2016), which is one of the state-of-the-art networks of the dataset. We use AlexNet (Krizhevsky et al., 2012) and ResNet-18 (He et al., 2016) variants as the base network for the ILSVRC2012.

2) SplitNet-Semantic. A variant of our SplitNet that obtains class grouping from a provided semantic taxonomy. Before training, we split the networks according to the taxonomy, evenly splitting layers and assigning subnetworks to each group, and train it from scratch. We use the same approach for SplitNet-Clustering and SplitNet-Random.

3) SplitNet-Clustering. A variant of our SplitNet, where classes are split (hierarchical) performing spectral clustering of the pre-trained base network weights.

4) SplitNet-Random. SplitNet using random class splits.

5) SplitNet. Our proposed SplitNet, that is trained using the proposed automatic splitting of weight matrices.

All SplitNet variants and other baselines are implemented using TensorFlow (Abadi et al., 2016).

4.1. Parameter Reduction and Accuracies

Experimental results below validate the two key benefits of our SplitNet: 1) Reducing the number of parameters without losing the prediction accuracy, and 2) obtaining a better structure for model-parallelization.

CIFAR-100. Table 1 summarizes split structures and test errors of different SplitNet variants and baselines. See the supplementary material for more details of used models. SplitNet variants using semantic taxonomy provided by the dataset (–S) and spectral clustering (–C) are better than random grouping (–R), showing that the appropriate grouping is critical for splitting DNNs. The SplitNet with learned splits outperforms all other variants, although it does not require any additional semantic information or pretrained network weights as with semantic or clustering split.

Table 3. **Comparison of Parameter/Computation Reduction and Test Errors of AlexNet variants on ILSVRC2012.** The number of splits indicates the split in *fc6*, *fc7* and *fc8* layer, respectively. In 2×5 split, we split from *conv4* to *fc8* with $G = 2$.

NETWORK	SPLITS	PARAMS(10^6)	% REDUCED	FLOPS(10^9)	% REDUCED	TEST ERROR(%)
ALEXNET(BASELINE)	0	62.37	0	2.278	0	41.72
SPLITNET	1-1-3	59.64	4.38	2.273	0.21	42.07
	1-2-5	50.69	18.72	2.256	1.00	42.21
	2-4-8	27.34	56.17	2.209	3.05	43.02
	2×5	31.96	48.76	1.847	18.95	44.60
SPLITNET-R	1-1-3	59.64	4.38	2.273	0.21	42.20
	1-2-5	50.70	18.70	2.256	0.99	43.20
	2-4-8	27.34	56.17	2.209	3.05	43.35
	2×5	31.94	48.79	1.846	18.93	44.99

Table 4. **Comparison of Parameter/Computation Reduction and Test Errors of ResNet-18 variants(ResNet-18x2) on ILSVRC2012.** The number of splits indicates the split in *conv4-1&2*, *conv5-1&2* and the last fc layer, respectively.

NETWORK	SPLITS	SPLIT DEPTH	PARAMS(10^6)	% REDUCED	FLOPS(10^9)	% REDUCED	TEST ERROR(%)
RESNET-18X2	0	0	45.67	0	14.04	0	25.58
SPLITNET	1-1-3	1	44.99	1.49	14.04	0.01	**24.90**
	1-2-2	6	28.39	37.84	12.39	11.72	**25.48**
	2-2-2	11	24.21	47.00	10.75	23.42	26.45
SPLITNET-R	1-1-3	1	44.99	1.49	14.03	0.01	25.86
	1-2-2	6	28.38	37.86	12.39	11.72	26.41
	2-2-2	11	24.14	47.14	10.75	23.46	28.61

Table 2 compares test errors, parameter reduction, and computation (FLOPs) reduction of SplitNets against the base network WRN-16-8: a 16 layer residual network with widening factor $k = 8$. Most of parameters in WRNs exist in convolutional layers, especially in higher layers due to the large number of filters. Thus, FC Split yields minimal parameter reduction. On the other hand, Shallow Split of the last 5 convolutional layers significantly reduces the network parameters by 32.44% while even slightly improving the accuracy. Deep and Hierarchical Split further reduce parameters and FLOPs at the cost of minor accuracy drop.

Shallow Split shows even better performance than the baseline with significantly fewer parameters. We attribute it to the fact that SplitNet starts from a full network and learns and cuts unnecessary connections between different groups for inner layers, imposing regularization effect on the layers. In addition, splitting layers can be regarded as a form of variable selection: each group in the layer parsimoniously selects only a needed group of input nodes.

ImageNet-1K. Table 3 and 4 summarize the results of ImageNet experiments with two base networks: AlexNet and ResNet-18x2, a variant of ResNet-18 where the numbers of filters are doubled, respectively. Refer to supplementary materials for model description. Using AlexNet as a base model, SplitNet greatly reduces the number of parameters concentrated in fc layers. However, most of the FLOPs come from lower conv layers, yielding only minor FLOPs reduction. We observe that SplitNet on AlexNet shows minor test accuracy drop with significant parameter reduction.

SplitNet based on ResNet-18x2 shows similar results as WRN-16-8 on CIFAR-100. With all split schemes, SplitNet outperforms SplitNet-Random. Further, with 1-2-2 split that splits the last FC layer and two residual blocks, the network parameters are significantly reduced by 37.86% while the test error is improved.

Table 5. **Model Parallelization Benchmark of SplitNet on Multiple GPUs.** We measure evaluation time performance of our SplitNet over 50,000 CIFAR-100 images with batch size 100 on TITAN X Pascal. Baseline implements layer-wise parallelization where sequential blocks of layers are distributed on GPUs.

NETWORK	GPUS	TIME(S)	SPEEDUP(×)
BASELINE	1	24.27 ± 0.35	1.00
BASELINE-HORZ.	2	46.70 ± 0.48	0.52
BASELINE-VERT.	2	20.15 ± 0.67	1.20
BASELINE-VERT.	3	22.45 ± 0.77	1.08
SHALLOW SPLIT	2	17.78 ± 0.23	1.37
DEEP SPLIT	2	14.03 ± 0.13	**1.73**
HIER. SPLIT	2	14.22 ± 0.05	1.71
DEEP SPLIT 3-WAY	3	10.92 ± 0.44	**2.22**

4.2. Test-time Model Parallelization

As illustrated in Figure 1, splitting DNNs yields speedup not only by reducing parameters, but also by utilizing the split structure for model parallelization. Thus we further validate parallelization performance of SplitNet at evaluation time, with multiple GPUs. Table 5 summarizes the run-time performance of SplitNet with model parallelization. We test two approaches: 1) Redundant assignment of lower layers (Deep and Hier. Split), and 2) assignment of lower layers to a separate GPU (Deep Split 3-way). With redundant assignment, the speedup becomes larger with

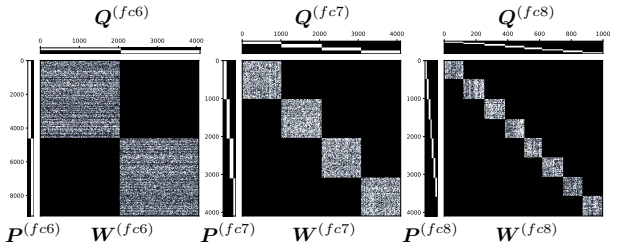


Figure 3. **Learned Groups and Block-diagonal Weight Matrices**. Visualization of the weight matrices along with corresponding group assignments learned in in AlexNet 2-4-8 split. Obtained block-diagonal weights are *split* for faster multiplication. Note the hierarchical grouping structure.

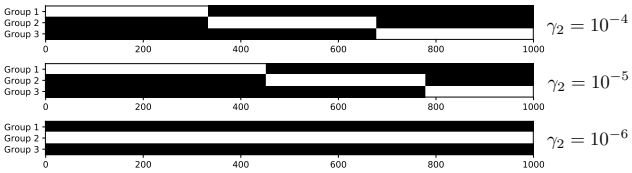


Figure 4. **Effect of Balanced Group Regularization** $R_E(\boldsymbol{P}, \boldsymbol{Q})$. The above figures show group-to-class assignment matrix $\boldsymbol{Q}^{(fc8)}$ for different values of γ_3 on ImageNet-1K with $G = 3$

deeper splits, up to $1.73\times$. Assigning lower layers to a third GPU eliminates redundant computation and achieves $2.22\times$ speedup. We compare against model-parallel approaches with both horizontal split (Baseline-Horz.) that splits the network horizontally as our split method but without pruning connections between two subnetworks, and vertical split (Baseline-Vert.) that splits the network into blocks of sequential layers. With vertical split, computations on GPUs are done asynchronously using queues to maintain intermediate activations. Horizontal split obtains much worse performance to original network due to excessive communication overhead. Vertical splitting scheme makes more sense with full networks, but the communication overhead is still large and it yields only $1.20\times$ and $1.08\times$ speedup with two and three GPUs respectively.

4.3. Qualitative Analysis

Figure 3 visualizes the weight matrices and corresponding group assignments obtained in AlexNet 2-4-8 split on ImageNet-1K. Both group assignments of classes and features, $\boldsymbol{P}$ and $\boldsymbol{Q}$, are sparse, and the weight matrices $\boldsymbol{W}$ of all layers are block diagonal. It indicates that grouping has converged with inter-group connections zeroed out.

Figure 4 shows the effect of balanced group regularization on the group size. With large regularization, groups becomes almost uniform in size, which is desirable for parameter reduction and model parallelization. Relaxing this regularization grants some flexibility on the individual group sizes. Setting γ_3 to be too small causes all classes and features to collapse into a single group, which may more

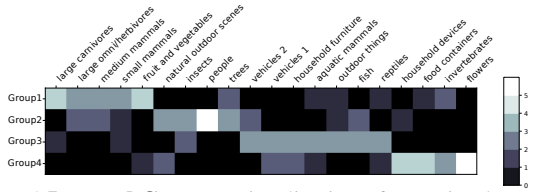


Figure 5. **Learned Groups**. Visualization of grouping learned in FC SplitNet on CIFAR-100. Rows denote learned groups, while columns denote semantic groups provided by CIFAR-100 dataset, each of which includes 5 classes. The brightness of a cell shows the agreement between learned groups and semantic groups.

closely resemble semantic taxonomies. In experiments, we enforced the models to be balanced, as our focus is more on efficiency and load balancing.

Figure 5 compares the learned group assignments in FC SplitNet ($G = 4$) with supercategories provided by the CIFAR-100. Each supercategory (column) includes five classes. For example, supercategory *people* includes *baby*, *boy*, *girl*, *man* and *woman*, which are grouped together by our algorithm into Group 2. Note that we have at least three classes from the same supercategory in each group. This shows that groupings learned by our method bear some resemblance to semantic categories even when no external semantic information is given.

The supplementary file presents the experimental results with varying G, the number of groups. Interestingly, with Shallow SplitNet on CIFAR-100, a higher G=4 achieves a better test accuracy with a parameter reduction of 48.66%.

5. Conclusion

We proposed a novel solution to split deep network into a tree of subnetworks, to not only reduce the number of parameters and computations, but to also enable straightforward model-parallelization. Specifically, we proposed an algorithm to cluster the classes into groups that fit to exclusive sets of features, which results in obtaining block-diagonal weight matrices. Our splitting algorithm is seamlessly integrated into the network training procedure as a regularization term, and thus allows the network weights and splitting to be trained at the same time. We validated our method on two classification tasks with two different CNN architectures, on which it greatly reduced the number of parameters over the base networks, while obtaining superior performance over networks with semantic or clustering-based groups. Moreover, our network obtained superior parallelization performance against the base networks at test time. As future work, we plan to explore ways to efficiently train SplitNet on multi-GPU or multi-processor environments from the initial training stage.

Acknowledgements This work was supported by Samsung Research Funding Center of Samsung Electronics under project number SRFC-IT150203.

References

Abadi, Martín, Agarwal, Ashish, Barham, Paul, Brevdo, Eugene, Chen, Zhifeng, Citro, Craig, Corrado, Greg S, Davis, Andy, Dean, Jeffrey, Devin, Matthieu, et al. Tensorflow: Large-scale Machine Learning on Heterogeneous Distributed Systems. *arXiv:1603.04467*, 2016.

Alvarez, Jose M and Salzmann, Mathieu. Learning the Number of Neurons in Deep Networks. In *NIPS*. 2016.

Ba, Jimmy and Caruana, Rich. Do Deep Nets Really Need to Be Deep? In *NIPS*, 2014.

Bengio, Yoshua, Ducharme, Rejean, Vincent, Pascal, and Jauvin, Christian. A Neural Probabilistic Language Model. *JMLR*, 3:1137–1155, 2003.

Chilimbi, Trishul, Suzue, Yutaka, Apacible, Johnson, and Kalyanaraman, Karthik. Project Adam: Building an Efficient and Scalable Deep Learning Training System. In *OSDI*, 2014.

Collins, Maxwell D. and Kohli, Pushmeet. Memory Bounded Deep Convolutional Networks. In *arXiv:1412.1442*, 2014.

Dean, Jeffrey, Corrado, Greg S., Monga, Rajat, Chen, Kai, Devin, Matthieu, Le, Quoc V., Mao, Mark Z., Ranzato, MarcAurelio, Senior, Andrew, Tucker, Paul, Yang, Ke, and Ng, Andrew Y. Large Scale Distributed Deep Networks. In *NIPS*, 2012.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and ei, L. Fei-F˙ Imagenet: A Large-Scale Hierarchical Image Database. In *CVPR*, 2009.

Goo, Wonjoon, Kim, Juyong, Kim, Gunhee, and Hwang, Sung Ju. Taxonomy-Regularized Semantic Deep Convolutional Neural Networks. In *ECCV*, 2016.

Han, Song, Pool, Jeff, Tran, John, and Dally, William. Learning Both Weights and Connections for Efficient Neural Network. In *NIPS*. 2015.

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Deep Residual Learning for Image Recognition. In *CVPR*, 2016.

Hinton, Geoffrey, Deng, Li, Yu, Dong, Dahl, George, rahman Mohamed, Abdel, Jaitly, Navdeep, Senior, Andrew, Vanhoucke, Vincent, Nguyen, Patrick, Kingsbury, Brian, and Sainath, Tara. Deep Neural Networks for Acoustic Modeling in Speech Recognition. *IEEE Signal Processing Magazine*, 29:82–97, 2012.

Hinton, Geoffrey, Vinyals, Oriol, and Dean, Jeff. Distilling the Knowledge in a Neural Network. *NIPS 2014 Deep Learning Workshop*, 2014.

Huth, Alexander H., Nishimoto, Shinji, Vu, An T., and Gallant, Jack L. A Continuous Semantic Space Describes the Representation of Thousands of Object and Action Categories across the Human Brain. *Neuron*, 76 (6):1210–1224, December 2012.

Krizhevsky, Alex. One Weird Trick for Parallelizing Convolutional Neural Networks. *arXiv:1404.5997*, 2014.

Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. ImageNet Classification with Deep Convolutional Neural Networks. In *NIPS*, 2012.

Murdock, Calvin, Li, Zhen, Zhou, Howard, and Duerig, Tom. Blockout: Dynamic Model Selection for Hierarchical Deep Networks. In *CVPR*, 2016.

Rakotomamonjy, Alain, Bach, Francis R, Canu, Stéphane, and Grandvalet, Yves. SimpleMKL. *JMLR*, 9:2491–2521, 2008.

Reed, Russell. Pruning Algorithms - A Survey. *IEEE Transactions on Neural Networks*, 4(5):740–747, 1993.

Shankar, Sukrit, Robertson, Duncan, Ioannou, Yani, Criminisi, Antonio, and Cipolla, Roberto. Refining Architectures of Deep Convolutional Neural Networks. In *CVPR*, 2016.

Sutskever, Ilya, Vinyals, Oriol, and Le, Quoc V. Sequence to Sequence Learning with Neural Networks. In *NIPS*, pp. 3104–3112, 2014.

Szegedy, Christian, Liu, Wei, Jia, Yangqing, Sermanet, Pierre, Reed, Scott, Anguelov, Dragomir, Erhan, Dumitru, Vanhoucke, Vincent, and Rabinovich, Andrew. Going Deeper with Convolutions. In *CVPR*, 2015.

Warde-Farley, D., Rabinovich, A., and Anguelov, D. Self-informed Neural Network Structure Learning. *arXiv:1412.6563*, 2014.

Wen, Wei, Wu, Chunpeng, Wang, Yandan, Chen, Yiran, and Li, Hai. Learning Structured Sparsity in Deep Neural Networks. In *NIPS*. 2016.

Yan, Zhicheng, Zhang, Hao, Jagadeesh, Vignesh, DeCoste, Dennis, Di, Wei, and Yu, Yizhou. HD-CNN: Hierarchical Deep Convolutional Neural Network for Image Classification. In *ICCV*, 2015.

Zagoruyko, Sergey and Komodakis, Nikos. Wide Residual Networks. In *BMVC*, 2016.

Zhang, Hao, Hu, Zhiting, Wei, Jinliang, Xie, Pengtao, Kim, Gunhee, Ho, Qirong, and Xing, Eric. Poseidon: A System Architecture for Efficient GPU-based Deep Learning on Multiple Machines. *arXiv:1512.06216*, 2015.