Efficient Ensemble Sparse Convolutional Neural Networks with Dynamic Batch Size

Shen Zheng, Liwei Wang, and Gaurav Gupta(✉)

College of Science and Technology, Wenzhou-Kean University, Wenzhou, China
{zhengsh,wangli,ggupta}@kean.edu

Abstract. In this paper, an efficient ensemble sparse Convolutional Neural Networks (CNNs) with dynamic batch size is proposed. We addressed two issues at the heart of deep learning—speed and accuracy. Firstly, we presented ensemble CNNs with weighted average stacking which significantly increases the testing accuracy. Secondly, we combine network pruning and Winograd-ReLU convolution to accelerate computational speed. Motivated by electron movement in electrical fields, we finally propose a novel, dynamic batch size algorithm. We repeatedly increase the learning rate and the momentum coefficient until validation accuracy falls, while scaling the batch size. With no data augmentation and little hyperparameter tuning, our method speeds up models on FASHION-MINST, CIFAR-10, and CIFAR-100 to 1.55x, 2.86x, and 4.15x with a testing accuracy improvement of 2.66%, 1.37%, and 4.48%, respectively. We also visually demonstrate that our approach retains the most distinct image classification features during exhaustive pruning.

1 Introduction

In the past few decades, Convolutional Neural Networks (CNN) have achieved state-of-art results on a variety of machine learning applications such as visual recognition, speech recognition, and natural language processing. LeCun et al. [1] proposed a simple CNN model with only two convolutional neural networks and three fully-connected layers. Later, Krizhevsky et al. [2] introduced AlexNet, being the first one to adopt ReLU as the activation function and to use drop-out technique to reduce overfitting. After that, Simonyan et al. [3] designed VGG, which improves AlexNet by decreasing kernel filter sizes to 3×3. Most recently, He et al. [4] developed ResNet, which popularized connections skipping to design deeper CNN without compromising its generalization ability.

Despite excellent classification accuracy, CNN is slow for computation. In deep learning, two accessible research directions have been explored to address this problem. One is to combine network pruning and convolutional accelerator for matrix compression and computations. The other is to investigate various activation functions. Details of them are in Sect. 2. Unfortunately, prior efforts in CNN fails to consider ensemble methods, which leads to diminishing margin return on

S. K. Singh et al. (Eds.): CVIP 2020, CCIS 1378, pp. 262–277, 2021.
https://doi.org/10.1007/978-981-16-1103-2_23

the testing accuracy. Besides, previous studies tend to use small, fixed batch size. However, our analysis show that small batch size result in noisy gradient descent steps and low optimal learning rates. Therefore, the convergence rate is slow. To overcome these challenges, we design an efficient stacked convolutional neural network structure. First, we use a weighted average stacking algorithm with three base CNNs. Next, we conduct network pruning to exploit model sparsity. Simultaneously, we perform the Winograd convolution with ReLU as the activation function. To address the accuracy loss from sparsity and to further increase computational speed, we finally propose a novel framework with dynamic batch size.

2 Related Work

2.1 Convolutional Accelerator

Various convolution algorithms have been introduced to reduce arithmetic complexity and thus to enhance the computational speed. Mathieu et al. [5] has incorporated Fast Fourier Transform convolution algorithms to accelerate the computation process of convolution neural network. However, it fails to work well with small filters. To address this issue, Lavin & Gray [6] have applied Winograd convolution [7], which has fast computation on minimal complexity convolution over small filters and batch sizes. Ioffe & Szegedy [8] presented a novel mechanism for dramatically accelerating the training of deep networks which is based on the premise that covariate shift, which is known to complicate the training of machine learning systems, applies to sub-networks and layers, and removing it from internal activations of the network may aid in training. Liu & Liang [9] further speeds up Winograd Convolution by rearranging the filter layout and by implementing dynamic programming algorithms to balance the computation among different processing components. However, Winograd Convolution is inconsistent with network pruning [8], which removes weak predictors to lower complexity by the reduction of overfitting. To address this problem, Liu & Turakhia [10] uses pruning in the first epoch to create sparse data, with subsequent retraining to help retain accuracy. Sheng et al. [11]) demonstrates that introducing Winograd layer in place of a standard convolution layer could achieve minimal accuracy loss with high sparsity. Liu et al. [12] provides a novel approach that moves ReLU activation into the Winograd domain to exploit weight sparsity and to reduce the computational complexity.

2.2 Activation Function

Activation Function helps the network learn complex patterns in the data. An activation function must be computationally efficient because it is calculated across thousands or even millions of neurons for each data sample. The need for speed has led to the development of new activation functions. Nair and Hinton [13] introduced rectified linear units (ReLU), a non-linear activation function that prevents gradients from being progressively small by having a gradient

slope of 1. ReLU is also computationally efficient. However, ReLU can trigger the dying neuron problem, where small inputs value mute the backpropagation. Recent researches have therefore introduced variants of ReLU such as Leaky ReLU [14] and Parametric ReLU [15]. They address the dying neurons by having a slight positive slope in the negative zones but do not demonstrate a robust noise control. Clevert et al. [16] later propose exponential linear unit (ELU). ELU uses mean shifts toward zero to reduce bias and variance in the forward propagation, leading to faster learning and generalization performance.

2.3 Batch Size

Early works in neural networks have shown that larger batch size results in degradation of the model accuracy. Krizhevsky [17] demonstrates that large batch size has resulted in "Generalization Gap", namely worse generalization abilities. Keskar et al. [18] shows that this is because large batch methods tend to converge to sharp minima. In contrast, Hoffer et al. [19] provide numerical evidence at the initial high learning rate region to demonstrate that "Generalization Gap" arise from the insufficient number of updates. Hoffer [19] is supported by Balles et al. [20] and McCandlish et al. [21], who both show that increasing batch size to reduce the gradient variance can also remove the 'Generalization Gap' increasing batch size. Later, Smith and Le [22] offers a Bayesian perspective on the generalization behaviour of neural nets. Smith et al. [22] further his claim by maintaining that one can encounter the accuracy loss of large batch methods with no additional updates by increasing the learning rate and the momentum coefficient while scaling the batch size. However, a large learning rate after scaling can result in divergence behaviour when the loss surface around the starting points is not smooth. To address this problem, Goyal et al. [23] propose a procedure for learning rate warmup, which ensures model stability at the initial phases.

3 Proposed Method

3.1 Stacking

Stacking is an ensemble learning technique that combines the predictions of multiple models. The reason we use stacking is that stacking combines the advantages of various heterogeneous models. By exploiting the predictive power of different models, stacking usually yields better accuracy than any single one of the trained models. Here we propose a weighted average stacking model which train the base models on the entire training set and then takes the best-weighted average of them to form an ensemble CNN. Following is the architecture for our weighted average stacking neural networks, where W1, W2 and W3 are weights for CNN1, CNN2 and CNN3, respectively.

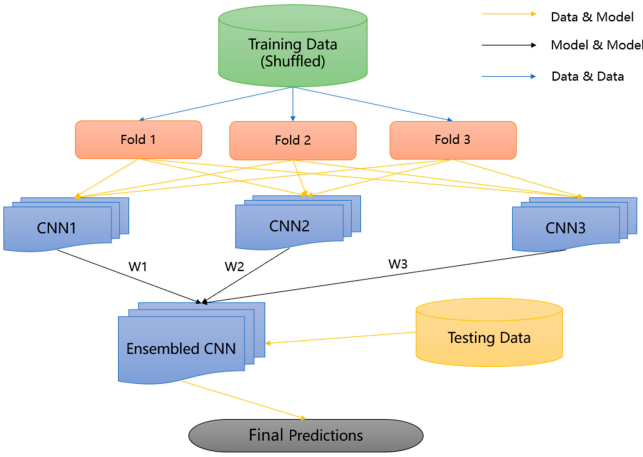

Fig. 1. Architecture for our weighted average stacking neural networks

3.2 Winograd-ReLU CNN

In deep learning, pruning is a model compression and acceleration technique to reduce the dimensions of the neural network by removing unnecessary weights connections. Lavin and Gray [6] has implemented the Winograd Minimal Filtering Algorithm [7].

$$Y = A^T[(GgG^T) \odot (B^T dB)]\, A \tag{1}$$

where G, B^T, A^T are filter, data and inverse transform, respectively, d, g represent input tile and filter, and Y represents output tile. However, Liu et al.'s [11]) study show that Winograd minimal filtering algorithm will occupy the sparse space and eliminate the gain from pruning. Liu et al. [11] then introduce the Winograd-ReLU CNN, which move ReLU into the Winograd Domain to eliminate the spatial-domain kernel. We will be implementing their framework, which obeys the following pruning schedule for creating model sparsity.

$$Y = A^T\left[\left[\text{Prune}\left(\text{GgG}^\top\right)\right] \odot \left[\text{ReLU}\left(\text{B}^\top\text{dB}\right)\right]\right]\text{A} \tag{2}$$

Where ReLU is an activation function and Prune means network pruning action.

3.3 Strategies for Better Convergence

Firstly, we examine the convergence performance of stochastic gradient descent. Suppose θ is a deterministic variable and $f(\theta)$ is the corresponding loss function. To ensure convergence for gradient descent algorithms, we must satisfy the condition that the gradient $\nabla_\theta f(\theta)$ is 'stable':

$$\nabla_\theta f(\theta) \approx \nabla_\theta f(\theta + \Delta\theta), \text{ where } \|\Delta\theta\| < \epsilon \cong 0 \tag{3}$$

From the perspective of Lipschitz Continuous Gradient hypothesis:

$$\|\nabla_\theta f(\theta + \Delta\theta) - \nabla_\theta f(\theta)\| \leq L\|\Delta\theta\|_2 \tag{4}$$

Where L is the Lipschitz constant. The above inequality has an important conclusion:

$$f(\theta + \Delta\theta) - f(\theta) \leq \frac{1}{2}L||\Delta\theta||_2^2 + \nabla_\theta f(\theta)^T \Delta\theta \tag{5}$$

In stochastic gradient descent, we define another random variable φ and rewrite the loss function as $f(\theta, \varphi)$. We then follow Bottous et al. [24]'s footstep, replacing $f(\theta, \varphi)$ with the empirical risk function $R_N(\theta)$, which is measured as

$$R_N(\theta) = \frac{1}{N}\sum_{i=1}^{N} f(\theta, \varphi_i) \tag{6}$$

Where N is the total numbers of data points. We keep $R_N(\theta)$ out of summation form for readability. When optimizing $R_N(\theta)$, the updates at the t^{th} step is $\Delta\theta_t = -\eta * g(\theta_t, \varphi_t)$, where η is the learning rate. Substituting it into (5):

$$R_N(\theta_{t+1}) - R_N(\theta_t) \leq \frac{1}{2}\eta^2 L|\,|g(\theta_t, \varphi_t)|\,|_2^2 - \eta\nabla_\theta R_N(\theta_t)^T g(\theta_t, \varphi_t) \tag{7}$$

Where $g(\theta_t, \varphi_t)$ is the gradient estimate at t. To eliminate the randomness for each update, we apply mathematical expectations to both sides of the inequality and get:

$$E[R_N(\theta_{t+1})] - E[R_N(\theta_t)] \leq \frac{1}{2}\eta^2 LE\left[||g(\theta_t, \varphi_t)||_2^2\right] - \eta\nabla_\theta R_N(\theta_t)^T E[g(\theta_t, \varphi_t)] \tag{8}$$

To make sure that $R_N(\theta)$ keeps decreasing, that is, the left side of (8) always less than 0, we try to let the right side less than 0:

$$\frac{1}{2}\eta^2 LE\left[||g(\theta_t, \varphi_t)||_2^2\right] - \eta\nabla_\theta R_N(\theta_t)^T E[g(\theta_t, \varphi_t)] \leq 0 \tag{9}$$

Since $\eta > 0$, the above inequality can be written as:

$$\frac{1}{2}\eta LE\left[||g(\theta_t, \varphi_t)||_2^2\right] - \nabla_\theta R_N(\theta_t)^T E[g(\theta_t, \varphi_t)] \leq 0 \tag{10}$$

Now we have two options:

- to minimize $\frac{1}{2}\eta LE\left[||g(\theta_t, \varphi_t)||_2^2\right]$
- to maximize $\nabla_\theta R_N(\theta_t)^T E[g(\theta_t, \varphi_t)]$

To minimize $\frac{1}{2}\eta LE\left[||g(\theta_t, \varphi_t)||_2^2\right]$, we first repeatedly reduce the learning rate when the validation accuracy stops improving. Since the variance of the gradient estimate is calculates as

$$\mathrm{Var}[g(\theta_t, \varphi_t)] = E\left[|\,|g(\theta_t, \varphi_t)|\,|_2^2\right] - \|E[g(\theta_t, \varphi_t)]\|_2^2 \tag{11}$$

According to the central limit theorem, when the sample size (in this case, batch size m) is large, the distribution is approximate normal. Then we can compute variance from sample variance$\mathrm{Var}_s\left[g\left(\theta_t, \varphi_t\right)\right]$:

$$\mathrm{Var}\left[g\left(\theta_t \varphi_t\right)\right] = \frac{\mathrm{Var}_s\left[g\left(\theta_t, \varphi_t\right)\right]}{\sqrt{m}} \tag{12}$$

Here we can increase the batch size m to reduce variance and thereby minimizing $\frac{1}{2}\eta LE\left[||g\left(\theta_t, \varphi_t\right)||_2^2\right]$. To maximize $\nabla_\theta R_N\left(\theta_t\right)^T E\left[g\left(\theta_t, \varphi_t\right)\right]$, we shuffle the training data to ensure independent and identical distribution of the sample data. This makes sure $E\left[g\left(\theta_t, \varphi_t\right)\right] \cong \nabla_\theta R_N\left(\theta_t\right)$. In this case, we obtain the minimum angle between the estimated gradient from SGD and the true gradient with full batch.

Secondly, we decrease the Lipschitz constant by batch normalization and skip connection (for residual network). This reduce 'sharp minima' and allow larger learning rate to operate. Consider the following equation:

$$\|\nabla_\theta f(\theta + \Delta\theta) - \nabla_\theta f(\theta)\| \leq L\|\Delta\theta\|_2 \tag{13}$$

This is equivalent to

$$\nabla_\theta\left(\nabla_\theta f(\theta)\right) = \frac{\|\nabla_\theta f(\theta + \Delta\theta) - \nabla_\theta f(\theta)\|}{\|\Delta\theta\|_2} \leq L \tag{14}$$

When the Lipschitz constant is large, the upper bound for the rate of change of the gradient $\nabla_\theta f(\theta)$ is large. Therefore, the loss surface is less smooth and the effective gradient estimation range is narrow, where small input change can result in huge output alternations. Prior studies have attempted to improve the smoothness of the loss surface. Li et al. [25] visually display that skip-connection can smooth the loss surface. Santurkar et al. [26] demonstrates that batch normalization achieves the similar effect with reparameterization, making model training less sensitive to parameter initialization.

Finally, we adopt the learning rate warmup technique by Goyal et al. [23], who demonstrate that linear scaling rule will not hold when the neural network is changing rapidly (i.e. during the initial training phase). Goyal then proposes the gradual warmup strategy, in which he accumulate the learning rate by a constant amount such that it reaches reasonable condition after 5 epochs. This learning rate warmup strategy avoid model to stuck at local minimum at the start of training so that it is less sensitive to initial points selection on the loss surface.

3.4 Dynamic Batch Size

Smith and Le [21] analysis the noise scale for SGD with momentum

$$\text{noise } = \frac{\eta}{1-m}\left(\frac{N}{B} - 1\right) \tag{15}$$

where m is the momentum coefficient, η is the learning rate, N is the data size, and B is the batch size. If $B \ll N$, the noise is approximated as

$$\text{noise} \approx \frac{\eta N}{B(1-m)} \tag{16}$$

Smith & Le [21] propose that we should increase the batch size by a factor of 3 and then scale the learning rate and the momentum coefficient to keep the noise scale. However, their empirical result shows that increasing momentum coefficient leads to significantly worse testing accuracy. To address this issue, we increase the learning rate and the momentum coefficient and then scale the batch size. We borrow the idea from the motion of electrons in the electric field to stabilize the growing process. In Electromagnetic, a positive electron receives repulsive force when it approaches an electrode plate with evenly distributed positive electrical charges. The magnitude of that force is proportional to the reciprocal of the distance between the electron and the plate. We can transfer this idea for increasing the momentum coefficients:

$$y''(n) = \frac{k}{y(n)} \tag{17}$$

Simplification gets us:

$$y''(n) * y(n) - k = 0 \tag{18}$$

Where y is the distance between the data points and the line $m = 1, n$ is the numbers of epochs and k is a constant that decides how violently momentum coefficient slow down when m approaches 1. We apply numerical methods and we will list the best value selection in Sect. 5.

Based on the previous discussion, we finally propose our algorithm for dynamic batch size.

- Warm Up the Learning Rate gradually from 0.01 to 0.02, for the beginning 10% of the total epochs.
- Increase the learning rate by a multiplier of 2 every n epoch until validation accuracy falls, keeping momentum coefficient fixed. Linearly Scale the batch size to the learning rate.
- Increase the momentum coefficient, keeping learning rate fixed. Scale the batch size to momentum coefficient.
- Stop the above action until reaching maximum batch size, which is determined by three restrictions: GPU memory limits, non-decreasing validation accuracy and linear scaling rule constraints ($B \ll N/10$)
- If validation accuracy does not improve for five consecutive epochs, decrease the learning rate by a multiplier of 0.1.

Following is the pseudo-code for in-epoch learning during the scaling period. The Round and the Clip function ensure the batch size be an integer between the minimum and the maximum batch size. The Stepwise function ensures the batch size will be a multiple of 32.

Algorithm 1. Mini-Batch SGDM with Dynamic Batch Size.

Require: Learning rate η, batch size B, momentum coefficient m, numbers of steps T, number of data points N, loss function $f(\theta)$.

for $t \in [1, T]$ **do**
  $B_{min} = B_0$
  $B = \text{Round\_\&\_Clip}\left(\frac{\eta(1-m_0)}{\eta_0(1-m)} B_0, B_{\min}, B_{\max}\right)$
  $B = Stepwise(B)$
  $g_t = \frac{1}{B}\sum_{i=1}^{B} \nabla f(\theta_i)$
  $v_t = m v_{t-1} + \eta g_t$
  $\theta_t = \theta_{t-1} - v_t$
end for
return B, θ_t

4 Experiments

For this experiment, we will use different convolutional neural networks on different datasets. All models will use ReLU as the activation function. One is that ReLU perform fast derivative operations on the gradient. The other is that ReLU mutes all negative values, resulting in higher network sparsity. We select image classification datasets including FASHION MNIST [27], CIFAR-10 [28] and CIFAR-100 [28]. For network architectures, we choose AlexNet [2], VGG-16 ch233 and ResNet-32 [4] respectively on the three datasets above. Using the TensorFlow framework and Nvidia RTX 2080 TI GPU, we train the Conventional CNN (C-CNN), Stacked Conventional CNN (SC-CNN), Stacked Winograd-ReLU CNN (SWR-CNN) and Stacked Winograd-ReLU CNN + Dynamic Batch Size (SWR-CNN + DBS). The stacked model takes a weighted average of the original model, ConvPool and AllCNN [29]. All models are iteratively pruned and retrained. Besides, batch normalization is applied. For a specific dataset, we will list the computational speed and visualize the testing accuracy. If not else specified, 'Time' in the table refers to training time per epoch in the unit of second.

4.1 FASHION MNIST (AlexNet)

We first examine the performance of AlexNet on Fashion MNIST. A baseline AlexNet without pruning takes 17 s per epoch to train on our GPU. We compute the relative computational speed to this number. A dropout [30] rate of 0.2 and an l2 regularization rate of 0.01 is used in convolutional layers for Conventional CNN. In contrast, the convolutional layers for all stacked models use no dropout and an l2 regularization rate of 0.001. Table 1 shows the computational speed for different AlexNet models at different sparsity. Pruning C-CNN speed up training to 1.49x, whereas SC-CNN slow down to 0.46x. Pruning SWR-CNN and SWR-CNN + DBS speed up training to 1.03x and 1.61x, respectively.

Table 2 shows the computational speed for different batch sizes at 30% sparsity where all models have the best or the second-best testing accuracy. Our

Table 1. Computational speed for different AlexNet models on FASHION-MNIST at Different sparsity

Sparsity	C-CNN		SC-CNN		SWR-CNN		SWR-CNN + DBS (ours)	
	Time	Speed	Time	Speed	Time	Speed	Time	Speed
20%	12	1.42x	37	0.46	17	1.00x	11	1.55x
30%	12	1.42x	37	0.46	17	1.00x	11	1.55x
40%	12	1.42x	36	0.47	16	1.06x	11	1.55x
50%	11	1.55x	37	0.46	17	1.00x	11	1.55x
60%	11	1.55x	37	0.46	16	1.06x	10	1.7x
70%	11	1.55x	36	0.47	16	1.06x	10	1.7x
80%	11	1.55x	37	0.46	16	1.06x	10	1.7x
Overall	11.42	1.49x	36.71	0.46x	16.43	1.03x	10.57	1.61x

dynamic batch size speeds up C-CNN, SC-CNN, SWR-CNN + DBS from 0.85x to 2.83x, 0.32x to 0.71x and 0.71x to 1.55x, respectively.

Table 2. Computational speed for different AlexNet models on FASHION-MNIST with different batch sizes

Batch size	C-CNN		SC-CNN		SWR-CNN	
	Time	Speed	Time	Speed	Time	Speed
64	20	0.85x	53	0.32x	24	0.71x
128	12	1.42x	37	0.46x	17	1.00x
256	7	2.43x	28	0.61x	13	1.31x
512	5	3.4x	22	0.77x	10	1.70x
1024	3	5.67x	21	0.81x	8	2.13x
DBS(Ours)	6	2.83x	24	0.71x	11	1.55x

Figure 2(a) shows the testing accuracy for different AlexNet models at different sparsity. Our SWR-CNN + DBS has the best testing accuracy at most sparsity. At 30% sparsity, SC-CNN, SWR-CNN and SWR-CNN + DBS has the testing accuracy of 92.29%, 92.45% and 93.49% respectively, whereas C-CNN only of 90.83%. When we prune the model to 60% sparsity, the C-CNN outperforms other models with an accuracy of 89.49%. If we prune the network to 80% sparsity, SWR-CNN + DBS outperforms others with a 90.94% accuracy. Figure 2(b) shows the testing accuracy for different models at 30% sparsity. Batch Size is the legend. Increasing batch size from 64 to 1024 reduce the testing accuracy for C-CNN, SC-CNN, SWR-CNN from 90.62% to 89.51%, 90.87% to 88.02%, 90.95% to 88.34%, respectively. Our dynamic batch size enhance the accuracy for three models, from left to right, to 90.97%, 93.35% and 93.49%.

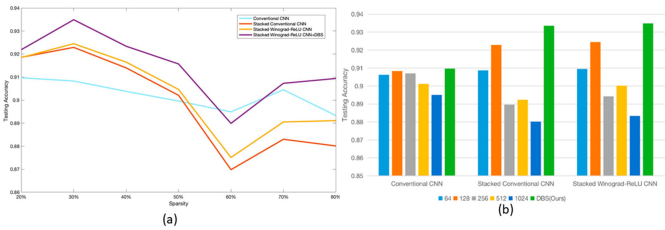

Fig. 2. (a) Testing accuracy for different AlexNet models on FASHION-MNIST with different batch sizes (b) Testing accuracy for different AlexNet models on FASHION-MNIST with different batch sizes

4.2 CIFAR-10 (VGG-16)

We then examine the performance of VGG-16 on CIFAR-10. A baseline VGG-16 without pruning takes 26 s per epoch to train on our GPU. The relative computational speed bases on this number. A dropout rate of 0.2 and an l2 regularization rate of 0.01 is applied to convolutional layers for Conventional CNN. In contrast, convolutional layers for all stacked models use no dropout and an l2 regularization rate of 0.001.

Table 3 shows the computational speed for different VGG-16 models at different sparsity. Pruning C-CNN speed up training to 1.42x, whereas SC-CNN slow down the model to 0.65x. Pruning SWR-CNN and SWR-CNN + DBS speed up training to 1.47x and 1.92x, respectively.

Table 3. Computational speed for different VGG models on CIFAR-10 at different sparsity

Sparsity	C-CNN		SC-CNN		SWR-CNN		SWR-CNN +DBS (ours)	
	Time	Speed	Time	Speed	Time	Speed	Time	Speed
20%	19	1.37x	41	0.63x	18	1.44x	15	1.73x
30%	19	1.37x	41	0.63x	18	1.44x	14	1.86x
40%	18	1.44x	40	0.65x	18	1.44x	14	1.86x
50%	18	1.44x	40	0.65x	18	1.44x	14	1.86x
60%	18	1.44x	40	0.65x	18	1.44x	13	2.00x
70%	18	1.44x	39	0.67x	17	1.53x	13	2.00x
80%	18	1.44x	39	0.67x	17	1.53x	12	2.17x
Overall	18.3	1.42x	40.0	0.65x	17.71	1.47x	13.57	1.92x

Table 4 shows the computational speed for different batch sizes at 40% sparsity where all models have the best or the third-best testing accuracy. Our dynamic batch size speeds up C-CNN, SC-CNN, SWR-CNN from 0.93x to 2.00x, 0.42x to 0.90x and 0.93x to 1.86x, respectively.

Table 4. Computational speed for different VGG models on CIFAR-10 with different batch sizes

Batch size	C-CNN		SC-CNN		SWR-CNN	
	Time	Speed	Time	Speed	Time	Speed
64	28	0.93x	62	0.42x	28	0.93x
128	18	1.44x	40	0.65x	18	1.44x
256	13	2.00x	32	0.81x	15	1.73x
512	11	2.36x	28	0.93x	13	2.00x
1024	9	2.89x	27	0.96x	12	2.17x
DBS(Ours)	13	2.00x	29	0.90x	14	1.86x

Figure 3(a) shows the testing accuracy for different VGG-16 models at different sparsity. Our SWR-CNN + DBS has the best testing accuracy when sparsity is at 20%, 40% and 50% At 40% sparsity, SC-CNN, SWR-CNN and SWR-CNN + DBS has testing accuracy of 88.34%, 89.02% and 89.73% respectively, whereas C-CNN of 88.36%. When we prune the model to 60% sparsity, C-CNN outperforms other models with an accuracy of 88.42%. If we prune the model to 80% sparsity, C-CNN CNN significantly outperforms other models with an accuracy of 83.65%.

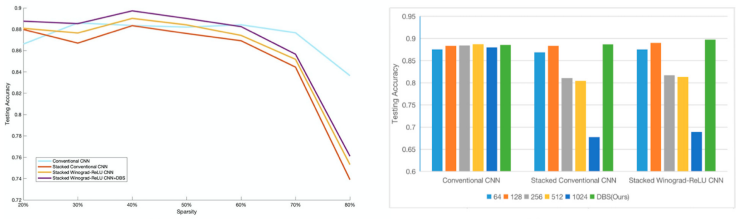

Fig. 3. (a) Testing accuracy for different VGG models on CIFAR-10 at different sparsity (b) Testing accuracy for different VGG models on CIFAR-10 with different batch sizes

Figure 3(b) shows the testing accuracy for different models at 40% sparsity. Batch Size is used as the legend. Increasing batch size from 64 to 1024 change the testing accuracy for C-CNN, SC-CNN, SWR-CNN from 87.54% to 87.99 86.87% to 67.73%, 87.50% to 68.91%, respectively. Our dynamic batch size enhance the accuracy for three models, from left to right, to 88.54%, 88.67% and 89.73%

4.3 CIFAR-100 (ResNet-32)

We last examine the performance of ResNet-32 on CIFAR-100. A baseline ResNet-32 without pruning takes 54 s per epoch to train on our GPU. Our

relative computational speed grounds on this number. A dropout rate of 0.1 and an l2 regularization rate of 0.001 is used at convolutional layers for Conventional CNN. In contrast, convolutional layers in all stacked models use the same dropout and an l2 regularization rate of 0.01.

Table 5. Computational speed for different ResNet models on CIFAR-100 at different sparsity

Sparsity	C-CNN		SC-CNN		SWR-CNN		SWR-CNN +DBS (ours)	
	Time	Speed	Time	Speed	Time	Speed	Time	Speed
20%	28	1.93x	51	1.06x	25	2.16x	21	2.57x
30%	28	1.93x	51	1.06x	25	2.16x	21	2.57x
40%	29	1.86x	49	1.10x	24	2.25x	20	2.7x
50%	28	1.93x	53	1.02x	26	2.08x	22	2.45x
60%	29	1.86x	53	1.02x	26	2.08x	22	2.45x
70%	30	1.8x	51	1.06x	25	2.16x	21	2.57x
80%	28	1.93x	49	1.1x	24	2.25x	20	2.7x
Overall	28.57	1.89x	51	1.06x	25	2.16x	21	2.57x

Table 5 shows the computational speed for different ResNet-32 models at different sparsity. Pruning C-CNN and the SC-CNN speed up training to 1.89x and 1.06x, respectively. Pruning SWR-CNN and SWR-CNN + DBS speed up training to 2.16x and 2.57x, respectively.

Table 6. Computational speed for different ResNet models on CIFAR-100 with different batch sizes

Batch size	C-CNN		SC-CNN		SWR-CNN	
	Time	Speed	Time	Speed	Time	Speed
64	49	1.10x	90	0.60x	53	1.02x
128	29	1.86x	49	1.10x	20	2.70x
256	20	2.70x	34	1.59x	14	3.86x
512	14	3.86x	25	2.16x	11	4.91x
1024	12	4.50x	22	2.45x	10	5.40x
DBS(Ours)	18	3.00x	29	1.86x	13	4.15x

Table 6 shows the computational speed for different batch sizes at 40% sparsity where all models have the best or the second-best testing accuracy. Our dynamic batch size speeds up C-CNN, SC-CNN, SWR-CNN from 1.10x to 3.30x, 0.60x to 1.86x and 1.02x to 4.15x, respectively.

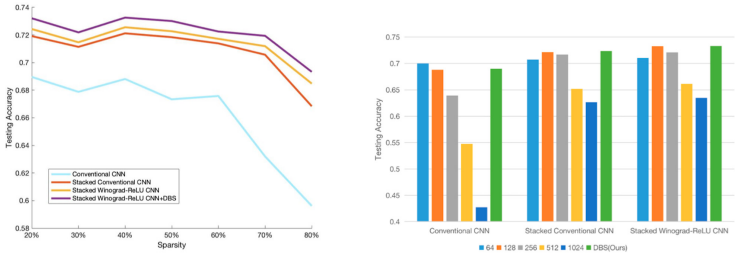

Fig. 4. (a) Testing accuracy for different ResNet models on CIFAR-100 at different sparsity (b) Testing accuracy for different ResNet models on CIFAR-100 with different batch sizes

Figure 4(a) shows the testing accuracy for different ResNet-32 models at different sparsity. Our SWR-CNN + DBS has the best testing accuracy for all sparsity. At 40% sparsity, the Stacked Conventional CNN, SWR-CNN and SWR-CNN + DBS have testing accuracy of 72.12%, 72.56% and 73.26% respectively, whereas C-CNN of 68.81%. If we prune the model to 80% sparsity, C-CNN significantly underperforms other models with an accuracy of 59.62%.

Figure 4(b) shows the testing accuracy for different models at 40% sparsity. Batch Size is the legend. Increasing batch size from 64 to 1024 reduce the testing accuracy for C-CNN, SC-CNN, SWR-CNN from 70.00% to 42.71%, 70.72% to 62.64%, 71.04% to 63.49%, respectively. Using our dynamic batch size change the accuracy for three models, from left to right, to 68.98%, 72.33% and 73.29%

5 Discussion

5.1 Result Interpretation

For Alexnet on Fashion-MNIST, we note that C-CNN has the least accuracy reduction when we increase sparsity. Recall that pruning reduces the model size by removing connections that have little classification power. In this way, pruning reduces the model complexity and prevent overfitting. In C-CNN architecture, a single model learns more from the data patterns and is more overfitted than stacked ones. Therefore, increasing sparsity save some accuracy loss by reducing C-CNN's overfitting, whereas less-overfitted stacked model loss more accuracy along the pruning process. This phenomenon is more evident for VGG on CIFAR-10, where we can prune the C-CNN from 20% to 80% sparsity with only 1.64% of accuracy loss.

Conversely, the Stacked VGG models experience massive accuracy loss of 13.98%, 12.78% and 12.66% respectively. This is reasonable since VGG-16 is much more complex and therefore suffers more from overfitting. Note that Stacked ResNet models do not exhibit this behaviour. This is because ResNet adopts multiple 'skip-connection' to reduce the model complexity. When sparsity

is less than 60%, both SC-CNN and C-CNN displays similar accuracy loss from pruning. However, C-CNN and SC-CNN drop accuracy massively from 60% and 70 % sparsity, respectively. We believe that they lost essential connections and begin to underfitting at that strong sparsity.

We also find that weighted stacking methods will generally increase the testing accuracy but will significantly reduce the computational speed. However, this negative effect can be mostly encountered by Winograd-ReLU methods, where we move the ReLU layer after Winograd Transformation. For VGG on CIFAR-10 and ResNet on CIFAR-100, SWR-CNN have faster speed then C-CNN. For AlexNet, however, the SWR-CNN is still significantly slower than the C-CNN. The reason is that we use ConvPool and AllCNN to stack the models. They are considerably less complicated than VGG and ResNet but have similar perplexity as AlexNet. Therefore, Stacked AlexNet with these two models will almost triple the computational time, whereas Stacked VGG and ResNet with them will not.

In contrast to Smith et al. [31]'s findings that increasing momentum coefficient will reduce the testing accuracy, we find that increasing momentum coefficient within our dynamic batch size framework will insteadincreasethe testing accuracy. For almost all experiments, our SWR-CNN + DBS outperforms other models, although in VGG and AlexNet it loses the match at high sparsity. Besides, our dynamic batch size method achieves similarly accurate result as the smaller batch size of 64 and 128 with faster training speed between that of a batch size of 256 and 512. In short, it allows the model to enjoy high accuracy and fast computational speed.

5.2 Kernel Sparsity Visualization

We visualize the kernels of the proposed Stacked Winograd-ReLU + DBS model. Figure 5(a) show we chose the first 6 inputs and output channels of layer 2 of ResNet-32 at three different pruning densities. During pruning, we find that the values tend to be kept in the shape of 2*2 blocks in each 4*4 patch. Note that 2*2 blocks are the only element that is linearly transformed with only addition, which is computational efficient than multiplication. Besides, our algorithm keeps distinct value and average the weak values (because of weighted average stacking) during aggressive pruning. It helps the model focus on the most crucial features and helps Winograd-ReLU operation retain more classification accuracy from pruning.

5.3 Optimizing Momentum Increase

We start with m = 0.90 and = 0.50 so that momentum coefficient should rise to 1.00 when proportion of total epochs reach 0.20, if k value for is zero. To choose the best k value for increasing momentum coefficient, we interpolate and plot the momentum coefficient to the proportion of total epochs with k values ranging from 1.30 to 1.75. We want the k value that achieves the faster increase subject to the best convergence when the portion of total epochs is 0.30. Figure 5(b) shows the optimal k value is approximate 1.60.

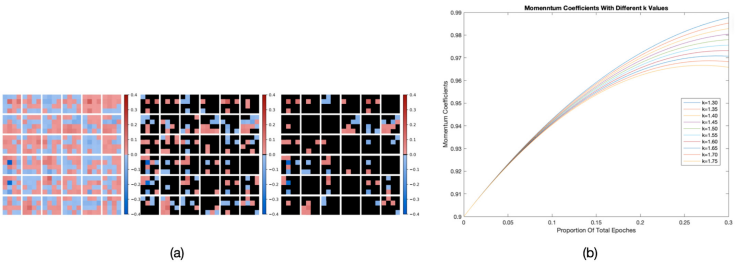

Fig. 5. (a) Kernels of Layer 2 from Winograd-ReLU ResNet-32 Model with dynamic batch size at different pruning sparsity (Left 0, Middle 60%, Right 80%) (b) Increase of momentum coefficient with different k values.

6 Conclusion

In this paper, we have shown that we can construct an efficient CNN with ensemble methods, convolutional accelerator and dynamic batch size. We first ensemble three CNNs and take a weighted average of their predictions. This stacking approach increases accuracy but slows down model training. Therefore, we iteratively prune and retrain the model, while performing Winograd convolution with ReLU as the activation function to accelerate computational speed. We also demonstrate that Winograd-ReLU operation will restore more accuracy loss when model sparsity is high. Finally, we further speed up model learning by sequentially increasing the learning rate and momentum coefficient and scale the batch size. With little hyperparameter tuning and no data augmentation, we speed up AlexNet on Fashion-MNIST, VGG on CIFAR-10 and ResNet on CIFAR-100 to 1.55x, 2.86x, and 4.15x with a testing accuracy increase of 2.66%, 1.37% and 4.48%, respectively.

References

1. Lecun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proc. IEEE **86**(11), 2278–2324 (1998)
2. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. Commun. ACM **60**(6), 84–90 (2017)
3. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition (2014)
4. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition (2015)
5. Mathieu, M., Henaff, M., LeCun, Y.: Fast training of convolutional networks through FFTs (2013)
6. Lavin, A., Gray, S.: Fast algorithms for convolutional neural networks (2015)
7. Winograd, S.: Arithmetic Complexity of Computations. Society for Industrial and Applied Mathematics (1980)
8. Ioffe, S., Szegedy, C.: Batch normalization: accelerating deep network training by reducing internal covariate shift (2015)

9. Lu, L., Liang, Y.: SpWA: an efficient sparse winograd convolutional neural networks accelerator on FPGAs. In: 2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC), pp. 1–6 (2018)
10. Liu, X.: Pruning of winograd and FFT based convolution algorithm (2016)
11. Li, S., Park, J., Tang, P.T.P.: Enabling sparse winograd convolution by native pruning (2017)
12. Liu, X., Pool, J., Han, S., Dally, W.J.: Efficient sparse-winograd convolutional neural networks (2018)
13. Nair, V., Hinton, G.E.: Rectified linear units improve restricted Boltzmann machines. In: Proceedings of the 27th International Conference on International Conference on Machine Learning, ICML 2010, Madison, WI, USA, pp. 807–814. Omnipress (2010)
14. Maas, A.L.: Rectifier nonlinearities improve neural network acoustic models (2013)
15. He, K., Zhang, X., Ren, S., Sun, J.: Delving deep into rectifiers: surpassing human-level performance on ImageNet classification (2015)
16. Clevert, D.A., Unterthiner, T., Hochreiter, S.: Fast and accurate deep network learning by exponential linear units (ELUs) (2015)
17. Krizhevsky, A.: One weird trick for parallelizing convolutional neural networks (2014)
18. Keskar, N.S., Mudigere, D., Nocedal, J., Smelyanskiy, M., Tang, P.T.P.: On large-batch training for deep learning: generalization gap and sharp minima (2016)
19. Hoffer, E., Hubara, I., Soudry, D.: Train longer, generalize better: closing the generalization gap in large batch training of neural networks (2017)
20. Balles, L., Romero, J., Hennig, P.: Coupling adaptive batch sizes with learning rates (2016)
21. McCandlish, S., Kaplan, J., Amodei, D., OpenAI Dota Team: An empirical model of large-batch training (2018)
22. Smith, S.L., Le, Q.V.: A Bayesian perspective on generalization and stochastic gradient descent (2017)
23. Goyal, P., et al.: Accurate, large minibatch SGD: training ImageNet in 1 hour (2017)
24. LeCun, Y., Cortes, C., Burges, C.J.: MNIST handwritten digit database. ATT Labs, 2 (2010). http://yann.lecun.com/exdb/mnist
25. Li, H., Xu, Z., Taylor, G., Studer, C., Goldstein, T.: Visualizing the loss landscape of neural nets (2017)
26. Santurkar, S., Tsipras, D., Ilyas, A., Madry, A.: How does batch normalization help optimization? (2018)
27. Xiao, H., Rasul, K., Vollgraf, R.: Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms (2017)
28. Krizhevsky, A., Hinton, G.: Learning multiple layers of features from tiny images (2009)
29. Springenberg, J.T., Dosovitskiy, A., Brox, T., Riedmiller, M.: Striving for simplicity: the all convolutional net (2014)
30. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. J. Mach. Learn. Res. **15**(1), 1929–1958 (2014)
31. Smith, S.L., Kindermans, P.J., Ying, C., Le, Q.V.: Don't decay the learning rate, increase the batch size (2017)