INCREMENTAL LEARNING IN DEEP CONVOLUTIONAL NEURAL NETWORKS USING PARTIAL NETWORK SHARING

Syed Shakib Sarwar, Aayush Ankit & Kaushik Roy *
Department of Electrical Engineering and Computer Science
Purdue University
West Lafayette, IN 47906, USA
{sarwar,aankit,kaushik}@purdue.edu

ABSTRACT

Deep convolutional neural network (DCNN) based supervised learning is a widely practiced approach for large-scale image classification. However, retraining these large networks to accommodate new, previously unseen data demands high computational time and energy requirements. Also, previously seen training samples may not be available at the time of retraining. We propose an efficient training methodology and incrementally growing a DCNN to allow new classes to be learned while sharing part of the base network. Our proposed methodology is inspired by transfer learning techniques, although it does not forget previously learned classes. An updated network for learning new set of classes is formed using previously learned convolutional layers (shared from initial part of base network) with addition of few newly added convolutional kernels included in the later layers of the network. We evaluated the proposed scheme on several recognition applications. The classification accuracy achieved by our approach is comparable to the regular incremental learning approach (where networks are updated with new training samples only, without any network sharing).

1 INTRODUCTION

Deep Convolutional Neural Networks (DCNNs) have achieved remarkable success in various cognitive applications, particularly in computer vision (Rosenberg, 2013). They have shown human like performance on a variety of recognition, classification and inference tasks, albeit at a much higher energy consumption. One of the major challenges for convolutional networks is the computational complexity and the time needed to train large networks. Since training of DCNNs requires state-of-the-art accelerators like GPUs (Chetlur et al., 2014), large training overhead has restricted the usage of DCNNs to clouds and servers. It is common to pre-train a DCNN on a large dataset (e.g. ImageNet, which contains 1.2 million images with 1000 categories), and then use the trained network either as an initialization or a fixed feature extractor for the specific application (Sharif Razavian et al., 2014). A major downside of such DCNNs is the inability to learn new information since the learning process is static and only done once before it is exposed to practical applications. In real-world scenarios, classes and their associated labeled data are always collected in an incremental manner. To ensure applicability of DCNNs in such cases, the learning process needs to be continuous. However, retraining these large networks using both previously seen and unseen data to accommodate new data, is not feasible most of the time. Incremental learning plays a critical role in alleviating this issue by ensuring continuity in the learning process through regular model update based only on the new available batch of data. A system that can learn incrementally is beneficial in practical situations, since it can gradually expand its capacity to accommodate increasing number of classes. Nevertheless, incremental learning can be computationally expensive and time consuming, if the network is large enough.

*Use footnote for providing further information about author (webpage, alternative address)—*not* for acknowledging funding agencies. Funding acknowledgements go at the end of the paper.

This paper focuses on incremental learning of deep convolutional neural network (DCNN) for image classification task. In doing so, we attempt to address the more fundamental issue: an efficient learning system must deal with new knowledge that it is exposed to, as humans do. To achieve this goal, there are two major challenges. First, as new data becomes available, we should not start learning from scratch. Rather, we leverage what we have already learned and combine them with new knowledge in a continuous manner. Second, to accommodate new data, if there is a need to increase the capacity of our network, we will have to do it in an efficient way.

There have been several prior works on incremental learning of neural networks. Many of them focus on learning new classes from fewer samples (Fei-Fei et al., 2006; Lampert et al., 2009) utilizing transfer learning techniques. To avoid learning new categories from scratch, Fei-Fei et al. (2006) proposed a Bayesian transfer learning method using very few training samples. By introducing attribute-based classification the authors (Lampert et al., 2009) achieved zero-shot learning (learning a new class from zero samples). These works rely on shallow models instead of DCNN, and the category size is small in comparison. The challenge of applying incremental learning (transfer learning as well) on DCNN lies in the fact that it consists of both feature extractor and classifier in one architecture. Polikar et al. (2001) utilized ensemble of classifiers by generating multiple hypotheses using training data sampled according to carefully tailored distributions. The outputs of the resulting classifiers are combined using a weighted majority voting procedure. Inspired form Polikar et al. (2001), Medera & Babinec (2009) utilized ensemble of modified convolutional neural networks as classifiers by generating multiple hypotheses. The existing classifiers are improved in (Polikar et al., 2001; Medera & Babinec, 2009) by combining new hypothesis generated from newly available examples without compromising classification performance on old data. The new data in (Polikar et al., 2001; Medera & Babinec, 2009) may or may not contain new classes. Xiao et al. (2014) proposed a training algorithm that grows a network not only incrementally but also hierarchically. Classes are grouped according to similarities, and self-organized into different levels of the hierarchy. All new networks are cloned from existing ones and therefore inherit learned features. These new networks are fully retrained and connected to base network. The problem with this method is the increase of hierarchical levels as new set of classes are added over time.

Our work differs in goal, as we want to grow a DCNN to accommodate new set of classes by network sharing, without forgetting the old classes. For learning new set of classes, we form a new network by reusing previously learned convolutional layers (shared from initial part of the base network) followed by new (added) trainable convolutional layers towards the later layers of the network. The shared convolutional layers work as fixed feature extractors (learning parameters are frozen) and minimize learning overhead for new set of classes. The error resilience property of neural network ensures that even if we freeze some of the learning parameters, the network will be able to adapt to it and learn. In fact, utilizing this attribute, in recent work Sarwar et al. (2017) showed the use of fixed Gabor filters as convolutional kernels in conjunction with regular trainable convolutional kernels. Since the frozen parameters are from an already learned network of similar (not same) classes, it further guarantees that the new classes can be learned smoothly without having convergence issues. By sharing initial convolutional layers of the base network, a significant fraction of the power-hungry components of the backpropagation training was eliminated, thereby achieving considerable reduction in training energy while maintaining competitive output accuracy.

The novelty of this work lies in the fact that we developed an empirical mechanism to identify how much of the network can be shared as new classes are added. In this work, we also quantified the energy consumption, training time and memory storage savings associated with models trained with different amounts of sharing to emphasize the importance of network sharing from hardware point of view. In summary, the key contributions of our work are as follows:

- We propose sharing of convolutional layers to reduce computational complexity while training a network to accommodate new set of classes.
- We developed a methodology to identify optimal sharing of convolutional layers in order to get the best trade-off between accuracy and other parameters of interest, especially computation energy consumption, training time and memory access.
- We developed an energy model for quantifying energy consumption of the network during training, based on the Multiplication and Accumulation (MAC) operations in the training algorithm.

- We substantiate the scalability and robustness of the proposed methodology by applying the proposed method to different network structures trained for different benchmark datasets.

We show that our proposed methodology leads to energy efficiency, reduction in storage requirements, memory access and training time, while maintaining classification accuracy.

2 Incremental Learning

A crude definition of incremental learning is that it is a continuous learning process as batches of labeled data of new classes are made available gradually. In literature, the term incremental learning is also referred to incremental network growing and pruning or on-line learning. Moreover, various other terms, such as lifelong learning, constructive learning and evolutionary learning have also been used to denote learning new information. Development of a pure incremental learning model is important in mimicking real, biological brains. Owing to superiority of biological brain, humans and other animals can learn new events without forgetting old events. However, exact sequential learning does not work flawlessly in artificial neural networks. The reasons can be the use of a fixed architecture and/or a training algorithm based on minimizing an objective function which results in *catastrophic interference*. It is due to the fact that the minima of the objective function for one example set may be different from the minima for subsequent example sets. Hence each successive training set causes the network to partially or completely forget previous training sets. This problem is called the *stability-plasticity dilemma* (Mermillod et al., 2013). To address these issues, we define an incremental learning algorithm that meets the following criteria:
i. It should be able to grow the network and accommodate new classes that are introduced with new examples.
ii. Training for new classes should have minimal overhead.
iii. It should not require access to the previously seen examples used to train the existing classifier.
iv. It should preserve previously acquired knowledge, *i.e.* it should not suffer from catastrophic forgetting.

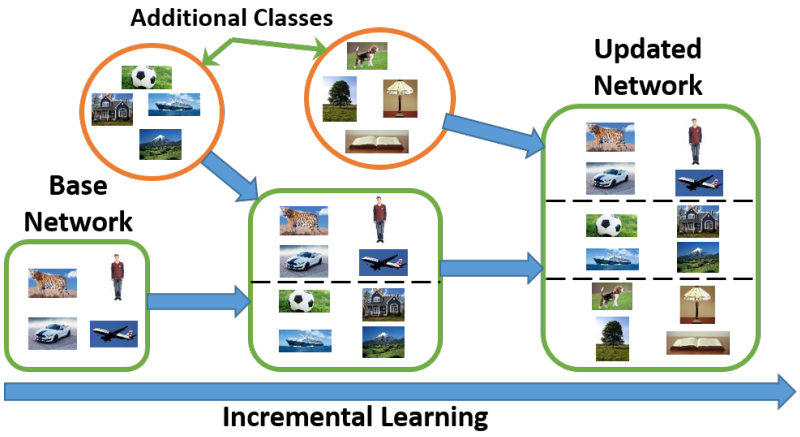


Figure 1: Incremental learning model: the network needs to grow its capacity with arrival of data of new classes.

We develop an efficient training methodology that can cover aforementioned criteria. Figure 1 shows the incremental learning model we implemented. The updated network can classify all the classes learned by the system. The following section will describe the design approach of the proposed scheme.

3 Design Approach

The superiority of DCNNs comes from the fact that it contains both feature extractor and classifier in the same network with many layers. '*Sharing*' convolutional layers as fixed feature extractors is

the base of our proposed training methodology. '*Sharing*' means reusing already learned network parameters/layers to learn new set of classes. Note that in all cases, while learning new classes, only newly available data is used. This section outlines the key ideas behind the proposed methodology.

3.1 Increasing Convolutional Kernels in the Last Layer

To accommodate more classes, the network must increase its capacity. The simplest way to do that is widening the final *softmax* layer to output the extra probabilities for the new classes. One obvious drawback of this approach is that the increment of learning capacity is small (Xiao et al., 2014). For example, let us consider a small CNN with two convolutional layers each containing 4 (5×5) convolutional kernels and a fully connected layer to connect the feature vector output with the output neurons. If the initial number of classes is 10 and 5 new classes are to be accommodated, then the increase in trainable parameters is only 17.8% compared to 50% increase in the number of classes. Since we do not want to forget the already learned classes, we can only train the small percentage of trainable parameters with the new examples. However, this does not result in a good inference accuracy for the new classes. For a large network with many convolutional layers, this increment of learning capacity reduces further and goes as low as less than 1%.

Therefore, it is prudent to widen the network by having more feature maps in the convolutional layers. To investigate this idea, we trained the above-mentioned network denoted by [784 4c5 2s 4c5 2s 10o] (CNN containing 784 input neurons, 2 convolutional layers (4c5) each followed by a sub-sampling layer (2s), and finally a fully connected layer with 10 output neurons (10o)), for 10 classes (digits 0-9 from TiCH dataset (Van der Maaten, 2009)). Then we added rest of the 26 classes (alphabets) to the existing network in five installments. Each time we retrain the network for new classes, we add two feature maps in the last convolutional layer. For example, when we add first 5 classes (A-E) with the existing 10 classes, we retrain the network [784 4c5 2s **6**c5 2s **5o**]. We only increment the concluding convolutional layer since it has been shown by Yosinski et al. (2014) that initial layers in a DCNN is more generic while last layers are more specific. Therefore, we only focus on incrementing and retraining the last few layers. Note that we added specifically 2 feature maps in the last convolutional layer (for each addition of 5-6 classes) in order to increase the model capacity while maintaining the existing class/filter ratio (~2.5 classes/filter) and prevent over-fitting. The new parameters are initialized using random numbers which have distribution similar to the learned weights. Cloning weights from learned filters provide similar results.

In the retraining process, only the 8 kernels corresponding to the 2 new feature maps and connections to the 5 new output neurons are trained with the new examples. Rest of the parameters are shared with the base network of 10 classes and as they are fixed, we will not forget the previously learned 10 classes. The new network becomes base network for the next 5 classes (F-J) to be added. That means, for any new set of classes, the network will be [784 4c5 2s **8**c5 2s **5o**], where only the 8 kernels corresponding to the 2 new feature maps and connections to the 5 new output neurons will be trained with the new examples. The accuracy achieved by this approach is given in the Table 1.

Table 1: Accuracy results for approach 1

Classes	Network	Incremental Training Accuracy (%)	
		With partial network sharing	Without partial network sharing
0-9 (base)	[784 4c5 2s 4c5 2s 10o]	–	96.68
A-E	[784 4c5 2s **6**c5 2s **5**o]	98.50	98.82
F-J	[784 4c5 2s **8**c5 2s **5**o]	98.95	99.90
K-O	[784 4c5 2s **10**c5 2s **5**o]	98.03	98.14
P-T	[784 4c5 2s **12**c5 2s **5**o]	98.17	98.41
U-Z	[784 4c5 2s **14**c5 2s **5**o]	96.57	96.76

We can observe from the above table that the accuracy degradation due to *'partial network sharing'* is negligible compared to the network *'without partial sharing'*. Note that *'without partial network*

sharing' is the case when new classes are learned using all trainable parameters, none of which are shared with the already learned network parameters; In this case, the new layers are initialized using the model with data A (old), and then fine-tuned with data B (new) without fixing any parameters.

However, such an approach has scalability issue. If we keep on adding more classes and continue increasing feature maps in the last convolutional layer, the network will become inflated towards the end. And hence, there can be overfitting and convergence issues while retraining for the new set of classes. We take care of this problem by retraining the final convolutional layer completely, the details of which are described in the following section.

3.2 RETRAINING FINAL CONVOLUTIONAL LAYER

The approach presented in section 3.1 is a straight forward one and shown in earlier related works. However, the limitations of such approach has motivated our search for a robust scalable method. We build upon the learnings of this approach to develop our proposed methodology.

To learn new set of classes, we clone and retrain the final convolutional layer. A new network is formed every time we add new set of classes, which shares the initial convolutional layers with the base network, and has a separate final convolutional layer and layers following it to the output. The cloned and retrained layers of the new network thus become a branch of the existing network. The advantage of cloning the final layers is that we do not have to worry about the initialization of the new trainable parameters. Otherwise, new kernels initialized with too big or too small a random value will either ruin the existing model or make training tediously long. Another advantage of cloning is that it maximizes the transfer of learned features. To investigate this approach, we implemented a deep CNN: [1024×3 (5×5)128c 100fc 64fc (3×3)mp (5×5)128c 128fc 128fc (3×3)mp (3×3)128c 128fc 10o] (c: convolutional layer kernels, fc: fully connected layer neurons, mp: max pooling layer kernel) for CIFAR10 (Krizhevsky & Hinton, 2009) dataset using MatConvNet (Vedaldi & Lenc, 2015). The network contained three MlpConv (Lin et al., 2013) blocks, each of which contained one convolutional layer, two fully connected layers and one max-pooling layer. We trained the base network for 4 classes. Then we added rest of the 6 classes to the existing network in two installments of 3 and 3. Each time we retrain the network for new classes, we clone the last MlpConv (Lin et al., 2013) block (highlighted in table 2) and retrain it using new examples for the new set of classes. During this retraining, the initial two MlpConv blocks are shared from the base network which work as fixed feature extractors (learning parameters are frozen) and minimize learning overhead for new set of classes. The cloned and retrained MlpConv block is then added to the existing network as a new branch. We can observe from the Table 2 that the accuracy degradation due to partial network sharing is negligible when we train for additional class sets, while accuracy for updated network of 10 classes suffers ~1.8% degradation. Note that ~89% classification accuracy can be achieved for CIFAR10 using NIN (Lin et al., 2013) architecture, if the training is done with all training samples applied together. However, for incremental learning, all training samples are not available together, hence it is not possible to get that high accuracy even without any network sharing.

Note that freezing a set of parameters in a pre-trained convolutional network is a standard practice for many applications involving knowledge transfer. But previous works used this method to learn a different dataset using the frozen parameters as fixed feature extractors. In that case, the new network can only classify the new dataset, not previously learned dataset. On the other hand, in our proposed methodology, both old and new learned classes can be classified together. This has not

Table 2: Accuracy results for approach 2

Classes	Network	Incremental Training Accuracy (%)	
		With partial network sharing	Without partial network sharing
10 (all classes)	[1024×3 (5×5)128c 100fc	–	88.90
4 (base)	64fc (3×3)mp (5×5)128c	–	91.82
3	128fc 128fc (3×3)mp	89.60	90.53
3	(3×3) **128c 128fc 4/3/3/10o**]	96.07	96.40
10 (updated)		58.72	60.49

been explored previously. However, one question is still unanswered: in a large DCNN with many convolutional layers, is retraining the final convolutional layer enough? To answer this question, we move to our third and final approach which will be explained in the next sub-section.

3.3 Replacing Part of the Base Network with New Convolutional Layers

A large DCNN usually has many convolutional layers followed by a fully connected final classifier. To apply our approach in a large DCNN, we implemented ResNet (He et al., 2016a;b) for a real-world object recognition application. CIFAR100 (Krizhevsky & Hinton, 2009) was used as the benchmark dataset. We trained a base network (ResNet101) that contained 100 convolutional layers with 50 classes out the 100 in CIFAR100. Then we added rest of the 50 classes to the existing network in three installments of 10, 20 and 20. Each time we retrain the network for new classes, we clone the last convolutional layer and retrain it using new examples for the new set of classes. That means we shared 99 convolutional layers and retrained only the last convolutional layer. We compared the accuracies achieved by this method with the accuracy of a network of same depth, trained without sharing any learning parameter, and observed that there is an 8-12% accuracy degradation for the former method. We also assessed the updated network accuracy for the all 100 classes by generating prediction probabilities from each of the separately trained networks and selecting the maximum probability. Even for the updated network, we saw about 10% accuracy degradation. To mitigate this accuracy loss, we reduced network sharing and allowed more freedom for retraining the networks. By gradually reducing sharing we observed improvement in the inference accuracy for both separate networks and the updated network. Figure2a shows the network architecture and figure 2b shows the *Accuracy vs Sharing* curve.

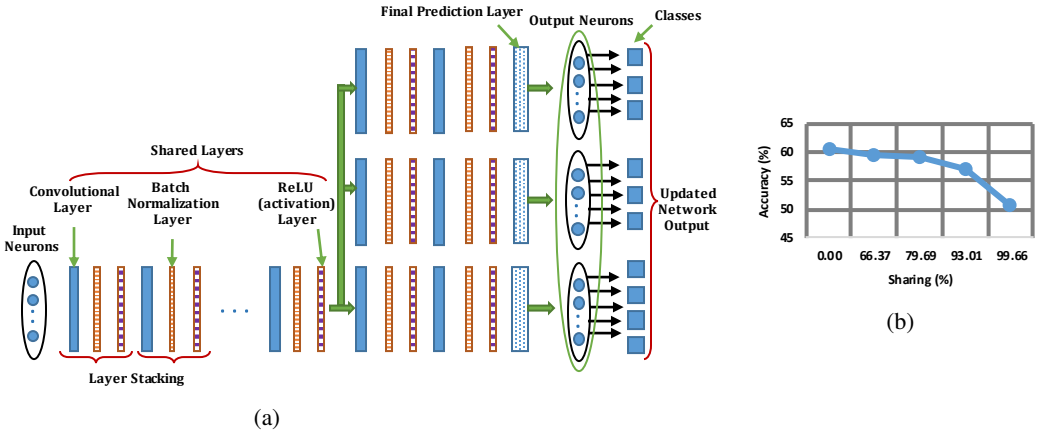


Figure 2: (a) Network architecture for proposed training methodology. For simplicity, the input bypass connections of ResNet (He et al., 2016a;b) is not shown here. (b) Combined network accuracy for 100 classes varied with different amount of network sharing. '%' Sharing in figure 2b is the portion of trainable parameters which are frozen and shared between the base and the new network.

Table 3: Accuracy results for approach 3

Classes	Network	Incremental Training Accuracy (%)	
		With partial network sharing	Without partial network sharing
100 (all classes)	ResNet101 (He et al., 2016a):	–	74.23
50 (base)	100 Convolution,	–	78.16
10	100 Batch Normalization,	88.8	87.2
20	100 ReLU, 1 average pooling,	78.3	81.4
20	1 Output Prediction layer	85.25	86.9
100 (updated)		59.08	60.65

From figure 2b we can see that when we share ~80% of the learning parameters in the convolutional layers (and corresponding batch normalization and ReLU layers), we can achieve accuracy within ~1% of baseline (incrementally trained network without network sharing). The accuracy results for this network configuration is listed in Table 3. Note that ~74% classification accuracy can be achieved for CIFAR100 using ResNet101, if the training is done with all training samples applied together. But for incremental learning, all training samples are not available together, hence it is not possible to get that high accuracy even without any network sharing. If we go beyond 80%, classification accuracy degrades drastically. Therefore, we can conclude that for this network structure and application, this is the optimal configuration. Based on this observation we developed the incremental training methodology for maximum benefits, which will be explained in the next sub-section.

3.4 Training Methodology

The proposed training methodology is depicted in figure 3. A base network is trained with initially available set of classes. Then a clone of the base network is trained with new, previously unseen data for different sharing configurations. From the training results, an *Accuracy vs Sharing* curve is generated, from which the optimal sharing configuration for this application and this network architecture is selected. This optimal configuration is then used for learning any new set of classes.

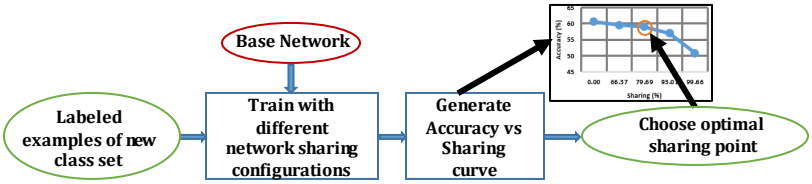


Figure 3: Overview of the DCNN incremental training methodology with partial network sharing.

Note that in the higher layers, there are branches for old and new set of classes. While retraining, and updating the network for new set of classes, only the branch of high-layers corresponding to the new set of classes are retrained. Thus, the high-layer filters keep information of their respective set of classes and the network do not suffer from catastrophic forgetting.

4 Simulation Framework

Here, we discuss the circuit to system-level simulation framework used to analyze the effectiveness of the presented training scheme on DCNNs. We developed a computational energy model based on the number of MAC operations in the training algorithm. We implemented multiplier and adder units at the Register-Transfer Level (RTL) in Verilog and mapped them to the IBM 45nm technology in 1 GHz clock frequency using Synopsys Design Compiler. The power and delay numbers from the Design Compiler were fed to the energy computation model to get energy consumption statistics. We also computed storage requirements and memory access energy for the overall network based on input size, number of convolutional layers, number of kernels in each layer, size of fully connected layer, number of neurons in the fully connected layer and number of output neurons. The memory structure (SRAM) in our proposed system was modelled using CACTI (Muralimanohar et al., 2009), in 45nm technology library, to estimate the corresponding component of the energy consumption.

At the system-level, the deep learning toolbox (Palm, 2012), MatConvNet (Vedaldi & Lenc, 2015), and PyTorch (Paszke et al., 2017), which are open source neural network simulators in MATLAB, C++, and Python, are used to apply the algorithm modifications and evaluate the classification accuracy of the DCNNs under consideration. The DCNNs were trained and tested using 4 NVIDIA Titan XP GPUs. In all experiments, previously seen data were not used in subsequent stages of learning, and in each case the algorithm was tested on an independent validation dataset that was not used during training. Details of the benchmarks used in our experiments are listed in Table 4:

Table 4: Benchmarks

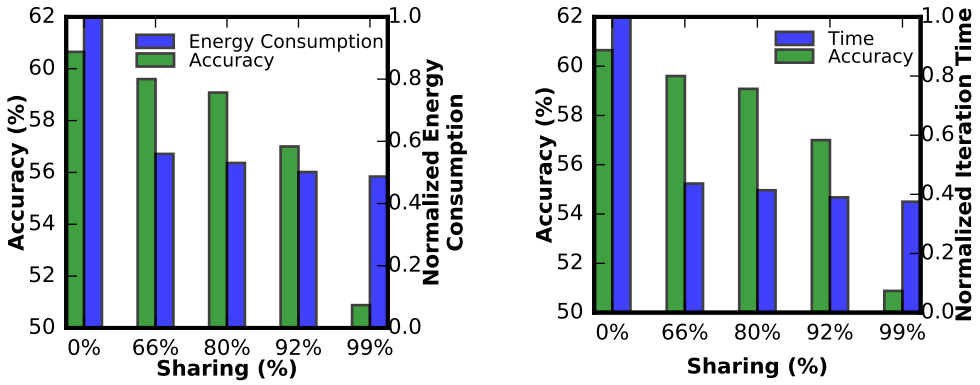
Application	Dataset	DCNN Structure
Character Recog.	TiCH	[784 4c5 2s 4c5 2s 10/5/6o]
Object Recog.	CIFAR10	[1024 × 3 (5 × 5)128c 100fc 64fc (3 × 3)mp (5 × 5)128c 128fc 128fc (3 × 3)mp (3 × 3)128c 128fc 4/3/3o]
Object Recog.	CIFAR100	ResNet101 100 Conv., 100 Batch Normalization, 100 ReLU, 1 average pooling, 1 Output Prediction layer
Object Recog.	ImageNet	ResNet34 33 Conv., 33 Batch Normalization, 33 ReLU, 1 average pooling, 1 Output Prediction layer

5 RESULTS

In this section, we present results that demonstrate the accuracy obtained, the energy efficiency and reduction in training time, storage requirements and memory access achieved by our proposed design.

5.1 ENERGY-ACCURACY TRADE-OFF

DCNNs are trained using the standard back-propagation rule with slight modification to account for the convolutional operators (Palm, 2012). The main power hungry steps of DCNN training (back-propagation) are gradient computation and weight update of the convolutional and fully connected layers (Sarwar et al., 2017). In our proposed training, we achieve energy efficiency by eliminating a large portion of the gradient computation and weight update operations, with minimal loss of accuracy or output quality. The normalized energy consumption per iteration for incremental training with and without sharing convolutional layers is shown in figure 4a. The accuracies reported in this work are obtained using test datasets, which are separate from the training datasets. Based on the accuracy requirement of a specific application the optimal sharing point can be chosen from the Accuracy vs Sharing curve mentioned in section 3.4. The optimal sharing configuration for CIFAR100 is 80% in ResNet101. By sharing 80% of the base network parameters, we can achieve 1.89x computation energy saving while learning new set of classes. The energy numbers slightly depend on number of classes in the new set of classes to be learned. However, it does not affect much since only the output layer connections vary with the number of new classes, which is insignificant compared to total connections in the network. Note that the energy mentioned in this comparison is computation energy. Memory access energy is discussed in a later section.



(a) (b)

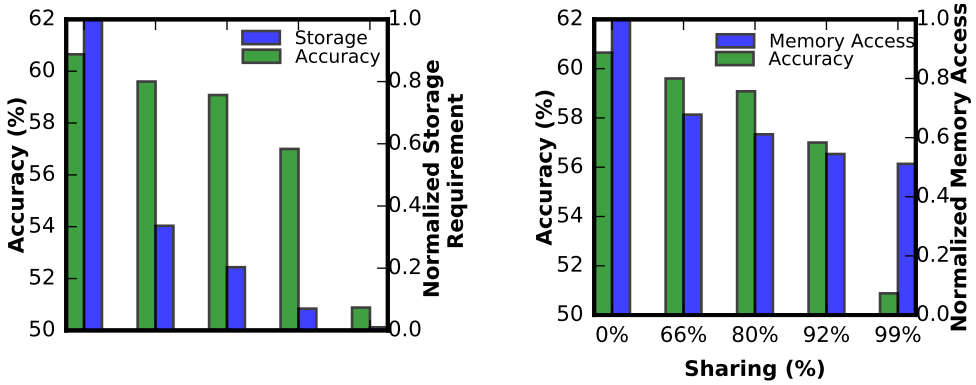
Figure 4: Comparison of (a) energy/accuracy trade-off and (b) training time requirements, between incremental training with and without sharing convolutional layers, is shown for different sharing configurations.

5.2 Training Time Reduction

Since gradient computation and weight update is not required for the shared convolutional layers, we achieve significant savings in computation time with our proposed approach. Figure 4b shows the normalized training time per iteration for learning a set of new classes. We observe 2.3-2.6x reduction in training time per iteration for CIFAR100 in ResNet101 (He et al., 2016a) for different sharing configurations. As a byproduct of the proposed scheme, convergence becomes faster due to inheriting features from base model. Note that the spare time cannot be used to improve accuracy of the networks by providing more epochs to the training. One way to improve accuracy is to retrain the networks with all the training samples (previously seen and unseen), which can be very time consuming.

5.3 Storage Requirement and Memory Access Reduction

Figure 5a shows the storage requirement reduction obtained using proposed scheme for CIFAR100 in ResNet101 (He et al., 2016a). We achieve 66-99% reduction of storage requirement. A large part of the training energy is spent on the memory read/write operations for the synapses. Proposed partial network sharing based training also provides 32-49% savings in memory access energy for CIFAR100 in ResNet101, since we do not need to write (update during backpropagation) the fixed kernel weights during training. Figure 5b shows the memory access requirement reduction obtained using proposed approach.



(a) (b)

Figure 5: Comparison of (a) storage and (b) memory access requirements, between incremental training with and without sharing convolutional layers, is shown for different sharing configurations.

5.4 Results on ImageNet

The ImageNet (Deng et al., 2009) (ILSVRC2012) is one of the most challenging benchmarks for object recognition/classification. The data set has a total of ~1.2 million labeled images from 1000 different categories in the training set. The validation and test set contains 50,000 and 100,000 labeled images, respectively. We implemented ResNet18 and ResNet34 (He et al., 2016a), and trained them on scaled (reduced resolution) ImageNet2012 dataset. We achieved 55.50% (top 1%), 80.54% (top 5%) and 59.91% (top 1%), 84.13% (top 5%) classification accuracy for ResNet18 and ResNet34, respectively, in regular training (all 1000 classes trained together). Then we divided the dataset into 3 sets of 500, 300 and 200 classes for the purpose of incremental learning. The set of 500 classes were used to train the base network. The other 2 sets were used for incremental learning. Utilizing our proposed method, we obtained the optimal sharing configuration. For ResNet18, we were able to share only ~1.5% of the learning parameters from the base network and achieve classification accuracy within ~1% of the baseline (network w/o sharing) accuracy. On the other hand, using ResNet34, we were able to share up to 38% of the learning parameters from the base network. The classification accuracy results for ResNet34 with ~38% sharing configuration are listed in Table 5. By sharing ~38% of learning parameters, we achieved 1.7x improvement in computation energy. We also reduced training time per iteration by 55% and memory access up to ~19%.

Table 5: Accuracy results for ResNet34 trained on ImageNet

#Classes	Accuracy(%) w/o sharing		Accuracy(%) w/ sharing	
	Top 1%	Top 5%	Top 1%	Top 5%
1000 (all classes)	59.91	84.13	-	-
500	69.85	91.2	-	-
300	64.34	85.05	60.88	82.42
200	63.9	85	63.73	85
1000 (updated)	59.05	82.24	58.11	81.15

Efficacy of incremental learning largely depends on the quality of the base network. In table 5, we can observe that the updated network performance is very close to the regular network compared to the performance on CIFAR10 and CIFAR100, on table 2 and 3, respectively. This is due to the fact that in the case of ImageNet, the base network had learned sufficient features since it was trained with large number of classes and examples. Another important fact to be noted here is that the amount of network sharing is largely dependent on the network size. Larger networks allow more learning parameters to be shared without performance loss. We could share lot more of the learning parameters from the base network in ResNet34 compared to ResNet18. Also for CIFAR100 trained using ResNet101, we could share up to 80% of the learning parameters. We could conclude that if the network depth is increased, the sharing percentage will also increase while maintaining network performance.

6 Conclusion

The performance of DCNNs relies greatly on the availability of a representative set of the training examples. Generally, in practical applications, data acquisition and learning process is time consuming. Also, it is very likely that the data are available in small batches over a period of time. A competent classifier should be able to support an incremental method of accommodating new data without losing ground on old data inference capability. In this paper, we introduce an incremental training methodology for DCNNs, which employs partial network sharing. This method allows us to accommodate new, previously unseen data without the need of retraining the whole network with previously seen data. It can preserve existing knowledge, and can accommodate new information. Importantly, all new networks start from an existing base network and share learning parameters. The new updated network inherit features from the base network by sharing convolutional layers, leading to improved computational effort and energy consumption during training and thus, speed up the learning process. We applied the proposed method on different DCNNs trained on real-world recognition applications. Results confirm the scalability of the proposed approach with significant improvements.

Acknowledgments

REFERENCES

Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759*, 2014.

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

Li Fei-Fei, Rob Fergus, and Pietro Perona. One-shot learning of object categories. *IEEE transactions on pattern analysis and machine intelligence*, 28(4):594–611, 2006.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016a.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European Conference on Computer Vision*, pp. 630–645. Springer, 2016b.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

Christoph H Lampert, Hannes Nickisch, and Stefan Harmeling. Learning to detect unseen object classes by between-class attribute transfer. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 951–958. IEEE, 2009.

Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

Dusan Medera and Stefan Babinec. Incremental learning of convolutional neural networks. In *IJCCI*, pp. 547–550, 2009.

Martial Mermillod, Aurélia Bugaiska, and Patrick Bonin. The stability-plasticity dilemma: Investigating the continuum from catastrophic forgetting to age-limited learning effects. *Frontiers in psychology*, 4, 2013.

Naveen Muralimanohar, Rajeev Balasubramonian, and Norman P Jouppi. Cacti 6.0: A tool to model large caches. *HP Laboratories*, pp. 22–31, 2009.

Rasmus Berg Palm. Prediction as a candidate for learning deep hierarchical models of data. *Technical University of Denmark*, 5, 2012.

Adam Paszke, Sam Gross, and Soumith Chintala. Pytorch, 2017.

Robi Polikar, Lalita Upda, Satish S Upda, and Vasant Honavar. Learn++: An incremental learning algorithm for supervised neural networks. *IEEE transactions on systems, man, and cybernetics, part C (applications and reviews)*, 31(4):497–508, 2001.

Chuck Rosenberg. Improving Photo Search: A Step Across the Semantic Gap. `http://googleresearch.blogspot.com/2013/06/improving-photo-search-step-across.html`, 2013. [Online; accessed 26-October-2017].

Syed Shakib Sarwar, Priyadarshini Panda, and Kaushik Roy. Gabor filter assisted energy efficient fast learning convolutional neural networks. In *2017 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pp. 1–6, July 2017. doi: 10.1109/ISLPED.2017.8009202.

Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. Cnn features off-the-shelf: an astounding baseline for recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pp. 806–813, 2014.

Laurens Van der Maaten. A new benchmark dataset for handwritten character recognition. *Tilburg University*, pp. 2–5, 2009.

Andrea Vedaldi and Karel Lenc. Matconvnet: Convolutional neural networks for matlab. In *Proceedings of the 23rd ACM international conference on Multimedia*, pp. 689–692. ACM, 2015.

Tianjun Xiao, Jiaxing Zhang, Kuiyuan Yang, Yuxin Peng, and Zheng Zhang. Error-driven incremental learning in deep convolutional neural network for large-scale image classification. In *Proceedings of the 22nd ACM international conference on Multimedia*, pp. 177–186. ACM, 2014.

Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pp. 3320–3328, 2014.