Efficient Incremental Training for Deep Convolutional Neural Networks

Yudong Tao, Yuexuan Tu, Mei-Ling Shyu
Department of Electrical and Computer Engineering
University of Miami, Coral Gables, FL, USA
{yxt128, yxt120, shyu}@miami.edu

Abstract—While the deep convolutional neural networks (DCNNs) have shown excellent performance in various applications, such as image classification, training a DCNN model from scratch is computationally expensive and time consuming. In recent years, a lot of studies have been done to accelerate the training of DCNNs, but most of them were performed in a one-time manner. Considering the learning patterns of the human beings, people typically feel more comfortable to learn things in an incremental way and may be overwhelmed when absorbing a large amount of new information at once. Therefore, we demonstrate a new training schema that splits the whole training process into several sub-training steps. In this study, we propose an efficient DCNN training framework where we learn the new classes of concepts incrementally. The experiments are conducted on CIFAR-100 with VGG-19 as the backbone network. Our proposed framework demonstrates a comparable accuracy compared with the model trained from scratch and has shown 1.42x faster training speed.

Keywords—Deep Convolutional Neural Network (DCNN); Incremental Model Training; Efficient Model Training

I. INTRODUCTION

Since the first deep convolutional neural network (DCNN), AlexNet, was proposed in 2012 [1], DCNNs have proved to be one of the best techniques for image classification tasks and various advanced network structures have been proposed to further improve the performance of DCNNs [2]. In 2014, Oxfords renowned Visual Geometry Group proposed VGGNet [3], which increases the depth to 16-19 convolutional layers and replaces the kernels in AlexNet with the small (3x3) ones. In the same year, GoogleLeNet [4] implemented the Network-in-Network architecture and performed very well in the ILSVRC-2014 challenge [5]. After that, the ResNet architecture was proposed, which incorporates the skip connections to improve the efficiency in backpropagation and gains considerable accuracy improvements [6]. ResNet achieved 3.57% error rate on the ImageNet test dataset, which was the first time that a computer beats humans in image classification on ImageNet. With the success of DCNNs in image classification, DCNNs have also been extended and widely used in a variety of applications, including text classification [7], face recognition [8], speech recognition [9], etc.

Training a DCNN requires a lot of computational resources due to its increasing depth and complexity. For example, training a 50-layer ResNet on ImageNet [10] from scratch on an NVIDIA M40 Graphics Processing Unit (GPU) takes about 2 weeks. There have been attempts to develop new techniques to accelerate the training process of DCNNs, including upgrading the optimization algorithms, model compression, and highly-parallel training with a large number of GPUs. Several major approaches are to improve the performance of the stochastic gradient descent (SGD) optimization algorithms, and to develop new optimization algorithms including Adagrad [11], Adam [12], learning rate annealing [13], etc. Compared to SGD, these advanced optimization algorithms have better robustness and higher convergence speeds, and thus enabled the training process to take fewer steps to reach good performance. In addition, tremendous studies rely on model compression for training acceleration, which can be roughly divided into the following four categories [14]: 1) Parameter pruning [15] eliminates redundant parameters that are less crucial to the overall performance; 2) Low-rank factorization [16] estimates the value parameters by tensor decomposition; 3) Transferred convolutional filters [17] replace the over-parametric filters with simpler blocks to improve the speed; and 4) Knowledge distillation [18] transfers the knowledge learned from the original large CNN model to a more compressed compact model. However, these approaches either need extra efforts to compress the model after the original model is trained or the model performance can be degraded.

Other than algorithmic acceleration, the recent focus is on accelerating model training by distributed computing with extreme scalability. In [19], a large minibatch size was shown to greatly help highly-parallel training without accuracy loss, and training ResNet-50 on ImageNet for 90 epochs was achieved in 15 minutes with 1024 Tesla P100 GPUs. The batch size was further increased using LARS algorithm and a slightly better speed with less GPU computation powers was obtained [20]. In [21], a mixed-precision training method with a large batch size was proposed and the ResNet50 training was accelerated to 6.6 minutes with comparable GPUs. These approaches require a massive amount of computing resources and are not available in all the scenarios.

However, most of the current frameworks treat model training as a static processing, and few studies focus on training the model for multiple-class classification in a dynamic
process. This motivates us to investigate an incremental training framework for the DCNN model, which splits the whole training process into several sub-training steps and dynamically evolves the model for the training efficiency and model performance. The framework first divides the learning concepts into different groups, and trains the initial model with the first group of concepts. The model is adapted and further trained while new groups of concepts are fed into the model in an incremental learning scenario. In this proposed framework, the concepts are grouped and added into the model in a sequence and the overhead of the old training data and the convergence speed is balanced to obtain a more efficient training process.

In image classification, transfer learning [22] has been widely applied, which initializes the DCNN with the weights trained from an existing dataset and adaptively trains the model with new data. Transfer learning enables the domain adaption from one dataset to another dataset and utilize the knowledge and patterns learned from previous dataset to help the learning process. Meanwhile, it is ubiquitous that one might want to accommodate several additional customized classes of concepts to a pre-trained DCNN and enhance the classification capability for the new concepts. In this case, it is wasteful to train the model from scratch again, giving that the pre-trained model has been optimized for all the original classes of concepts.

The contributions of this study include: (1) it presents a framework that incrementally trains a DCNN and achieves comparable performance of training from scratch; (2) it illustrates the relation between the convergence speed and the number of classes of concepts in the old and new dataset; and (3) it introduces a novel efficient incremental model training framework for the DCNNs.

The rest of the paper is organized as follows. We introduce some related work in Section II and present the proposed framework in Section III. The experiment results are then shown in Section IV for performance evaluation. Finally, Section V concludes the contribution of this study.

II. RELATED WORK

Inspired by the idea of transfer learning [23], the features learned from the previous model can assist the task of learning a new related concept (or class). It has been shown that the features extracted in the lower layers of CNNs are general features, similar to Gabor filters and color blobs; while the features eventually become specific at the last layer [24]. The general features can be also applied to extract low-level features of the new concepts. Thus, transferring the network parameters from a pre-trained model will greatly increase the training speed of a new model, especially when the concepts of the new model are related to those of the pre-trained model. It has been observed that a model initialized with transferred features will not lose its generalization ability even after it is fine tuned with the target dataset. Such an observation makes us confident to use the idea of transfer learning incrementally, since the initial generalization ability will linger after fine-tuning.

We also borrowed some ideas from the context of incremental learning, which is also very popular in image classifications. A partial sharing method between a new network and the base network was proposed in [25], which allows new classes to be trained incrementally and efficiently. In [5], a hierarchical DCNN which grows like a tree to incrementally learn new classes was proposed. An incremental learning technique that splits the base network into various sub-networks was proposed, which are then gradually incorporated in the training process [26]. Another approach trains the new model with minimum supervision to enhance the training efficiency in the incremental learning. In [27], the authors proposed an one-shot learning algorithm which learns the knowledge of a category from only one or very small number of images. Their proposed Bayesian transfer learning algorithm avoid retraining the whole model from scratch and achieves good performance in learning new classes with very few training data. However, the one-shot learning approach cannot achieve very good performance. However, the incremental learning approach cannot achieve very good performance. Although both our proposed diagram and the diagram of incremental learning add new concepts to the model during training, the proposed method works for offline dataset instead of training the model in an online manner. The training dataset is deliberately split into several parts to accelerate the overall training speed.

III. PROPOSED FRAMEWORK

A. Problem Formulation

In this paper, we propose to train the DCNN model for image classification in an incremental manner so the training process can be accelerated without using additional computing resources and without losing model performance. Assume that the model is expected to distinguish \(N^*\) classes of concepts, and this concept set is denoted as \(C^* = \{c_i\}_{i=1}^{N^*}\). The dataset \(\mathcal{D}^* = \bigcup_{i=1}^{N^*} I_i\) is given to train the model, where \(I_i\) refers to the set of training images of the concept \(c_i\).

Instead of training the model with \(\mathcal{D}^*\) directly, we propose to separate \(\mathcal{D}^*\) into \(T\) subsets \(d_1, d_2, \ldots, d_T\), where each subset \(d_j\) contains all the training images of \(n_j\) concepts and each concept belongs to only one subset. Without loss of generality, we split the training dataset as the given order of concepts, i.e., \(d_j = \bigcup_{k=N_{j-1}+1}^{N_j} I_k\), where \(N_j = \sum_{k=1}^{j} n_k\) is the number of concepts in \(d_j\) and all the preceding subsets. In particular, \(N_0 = 0\).

In our proposed framework, the training process is accomplished in \(T\) stages. In the \(j\)-th stage, all the training images of \(n_j\) concepts will be added into the original training set to form a new dataset \(\mathcal{D}_j\) using Equation (1). \(^{(1)}\)

\[
\mathcal{D}_j = \mathcal{D}_{j-1} \cup d_j,
\]
where $D_0 = \emptyset$. The new model $M_j$ will be trained based on the model from the previous stage $M_{j-1}$ and the new dataset $D_j$. Since each image sample will be added to the dataset once and only once, we have $D_T = D^*$, $N_T = N^*$, and $d_{j_1} \cap d_{j_2} = \emptyset, \forall j_1 \neq j_2$. An example of the proposed incremental training framework is depicted in Figure 1, where the training process is separated into $N^*$ stage and one class of concept is added in each stage ($n_j = 1, j = 1, 2, \ldots, N^*$). Our goal is then to find an appropriate method to train $M_j$ capable of classifying both the original and new classes of concepts and find an appropriate way to determine the number of stages $T$ and $n_j$ for each stage.

B. Model Growing and Incremental Training

Since the DCNN uses the fully connected (fc) layer as the last layer to perform classification, the fc layer in the model $M_{j-1}$ has $N_{j-1}$ nodes. Each of the node represents a class of concept and outputs the likelihood of the given image being the corresponding class of concept. Therefore, a straightforward way to grow the model is to add $n_j$ nodes in the last fc layer in the $j$-th stage, and the new model $M_j$ will have $N_j = N_{j-1} + n_j$ nodes in its last fc layer.

Since the model parameters in $M_{j-1}$ have already been optimized for $D_{j-1}$, all the parameters in the previous layers from $M_{j-1}$ can be transferred to the new model $M_j$ to initialize these layers. Given the success of transfer learning, the model $M_j$ initialized by this parameter transfer process should be able to learn the new classes of concepts with the constraint of the model learnability. Regarding the last fc layer, assume its input is a feature vector $x \in \mathcal{R}^a$, where $a$ is the dimension of the feature vector. Then, the parameters of the last fc layer in the new model $M_j$ can be represented as a matrix $W_j = [W_j^1, W_j^2]$, where $W_j^1 \in \mathcal{R}^{(a+1) \times N_{j-1}}$ are the parameters used to predict the classes of concepts in the previous model $M_{j-1}$, and $W_j^2 \in \mathcal{R}^{(a+1) \times N_j}$ are the parameters used to predict the new classes of concepts. The additional dimension is the bias in the fully connected layer. Since the parameters of all the previous layers remain the same and the model $M_{j-1}$ has been optimized to classify $D_{j-1}$, the parameters of the previous classes of concepts in the last fc layer should remain the same to ensure the best classification accuracy. That is,

$$W_j^1 = W_{j-1}.$$  \hspace{1cm} (2)

Without the prior knowledge, the best parameters for the new nodes are unable to determine. Hence, the standard procedure to initialize these parameters with random values is adopted, and the output with the dimension $N_j$ can be determined by Equation (3).

$$y_j = f([x^T, 1] \cdot W_j),$$  \hspace{1cm} (3)

where $x^T$ is the transpose of feature vector $x$, and $f$ is the activation function for classification and softmax is commonly used for image classification. Therefore, the output becomes the likelihood of being the classes of concepts in the training set. Since the activation function is calculated element-wise, the output can be written as follows.

$$y_j = [y_j^1, y_j^2] = [f([x^T, 1] \cdot W_j^1), f([x^T, 1] \cdot W_j^2)] .$$  \hspace{1cm} (4)

Therefore, it can be observed that all the parameters in $M_{j-1}$ are transferred to $M_j$. Since $y_j^1 = y_{j-1}$ before the training process in the $j$-th stage is performed, the model $M_j$ should keep the capability of identifying images from the previous $N_{j-1}$ classes of concepts.

C. Stage Separation

During the incremental training process, the total training time $t_{total}$ is the sum of training time of each stage, i.e., $t_{total} = \sum_{j=1}^{T} t_j$, where $t_j$ is the training time of the $j$-th stage. For each stage, the training time can be further decomposed into two parts: the training steps over $D_{j-1}$ and new concepts $d_j$. Since the computation times for both forward and backward propagation are roughly the same for various images, the training time $t_j$ is proportional to the number of images in the dataset and can be represented by
connected layers to 1 average pooling layer and 1 fully

The backbone network used in all the experiments is a variant of VGG-19 proposed in [29]. The network is designed for the CIFAR-100 dataset and shrinks the 3 fully connected layers to 1 average pooling layer and 1 fully

connected layer. During training, some common data augmentation methods are applied, including horizontal flipping, random cropping, and padding. Both the training and testing images are normalized by the mean and standard deviation along each channel. After each convolution layer, a Rectified Linear Units (ReLU) activation and batch normalization [30] are employed. SGD is adopted as the optimizer where the momentum is 0.9 and the weight decay is 5e-4. A softmax function is performed to the output of the last fully connected layer to generate the class probability.

The models trained from scratch including various numbers of concepts are used as the baseline. In each of the training process, 300 epochs training in total are performed, where the learning rate starts from 0.1 and is divided by 10 in every 100 epochs. The baseline model is used as the initial model \( M_0 \) in the experiments. The incrementally trained models are expected to have achieved comparable or slightly lower accuracy values.

**C. Results**

For the comparison, the performance results of incremental training in terms of accuracy and the convergence speed are presented. All the experiments are conducted with the CIFAR-100 dataset and the number of epoches for training is set based on the empirical study.

As discussed in Section III-C, the number of incremental concepts is determined based on the \( \alpha \) value. In particular, to train the model for the whole CIFAR-100 dataset (with 100 concepts), the model is trained in five stages. In the initial stage, 10 concepts are applied to train the model and then in the three following stages, the number of new concepts added is the same as the number of concepts in the model at the current stage (i.e., \( n_j = N_{j-1} \)). In this study, the numbers of new concepts added in the first three stages are 10, 20, and 40, respectively. That is, the total numbers of concepts to train the model in the first three stages are 20, 40, and 80. In the last stage, since there are 20 concepts (i.e., 100 - 80) left to add, all of these 20 concepts are included. The comparison of accuracy between the proposed model and the baseline is shown in Figure 2, where the x-axis refers to the number of concepts in the model and y-axis refers to
the accuracy value. The final accuracy of the incrementally trained model is able to reach 70.83% which is comparable to the baseline result, and the total training takes 5.6 million steps, which is 1.42 times faster than training from scratch.

The determination of the stage partitioning is essentially the hyper-parameter tuning of the number of concepts in the initial stage $n_1$ and the number of incremental concepts $\Delta n$. In the following experiments, the same $n_j$ is used for all the following stages and thus $\Delta n$ is used to represent the number of incremental concepts). The performance of training depends on $n_1$ and $\Delta n$. Hence, an empirical study is conducted to analyze the relations of the convergence speed of the training process with $n_1$ and $\Delta n$. The learning rate is fixed as 0.01 and the number of steps for the training process to reach a sufficiently small gradient is recorded as the measure of the convergence speed. As shown in Table I, each result corresponds to the pair of the numbers of incremented concepts and initial concepts $(\Delta n, n_1)$. For example, when the numbers of initial concepts and incremented concepts are both 10, the training process takes 650 thousand steps to converge. The dash (“-”) in the table means that the data point $(\Delta n, n_1) = (50, 80)$ is not applicable since there are only 100 concepts in total in the dataset. From the results in Table I, it can be observed that the convergence speed slows down when $\Delta n$ and $n_1$ become larger, which verifies the property mentioned in Section III-C. Since training is a stochastic process, the cost to obtain a complete information about the convergence speed is unaffordable. This demonstrates that our proposed framework balances the number of stages and the convergence speed of each stage.

Furthermore, we executed nine runs with various numbers of initial concepts $n_1$ and incremented concepts $\Delta n$, and the same number of incremented concepts $(\Delta n)$ is adopted in all the stages. In each run, 300 epochs are executed as the same setting of the baseline. In these experiments, the top-1 error rates of the obtained models with 90 and 100 final concepts ($N^* = 90$ and 100) are calculated, where the results of $N^* = 90$ only account for the data of the 90 concepts included in the model.

The error rates on the testing dataset are shown in Table II. Each of the error rate corresponds to the tuple of the number of initial concepts, the number of incremented concepts, and the final number of concepts $(n_1, \Delta n, N^*)$. For example, when the numbers of initial concepts and incremented samples are both 10 and the number of final concepts is 90, the top-1 error rate is 26.66%. The dashes (“-“) in the table also mean that the data points $(n_1, \Delta n, N^*) = (10, 45, 90)$ and $(10, 90, 90)$ are not applicable.

As shown in Table II, Runs 1 to 7 show comparable performance results compared to the baseline model, and Run 7 has the best performance results (26.66% for $N^* = 90$ and 28.20% for $N^* = 100$) which are slightly better than those of the baseline (26.72% for $N^* = 90$ and 28.30% for $N^* = 100$). On the other hand, Runs 8 and 9 show significantly higher error rates than the baseline results. The reason might be that the visual patterns learned from the first 10 concepts are insufficient to be generalized to the whole dataset and misleading the model. Therefore, the models require more steps to converge to the original performance and show higher error rates with the same setting as the baseline model.

Based on the results of Runs 1 to 4, it can be observed that a model with a larger number of initial concepts has better performance than the lower ones. The initial model with a higher $n_1$ learns more generalizable visual patterns in the first place so that the error rate becomes lower when the initial model is trained with more concepts.

V. CONCLUSION

In this paper, a novel efficient incremental training framework for deep convolutional neural networks (DCNNs) is proposed. The experiments using the CIFAR-100 dataset to train the image classification model are conducted and the performance results in terms of the accuracy and the convergence speed are presented. It can be seen from the experimental results that the model trained with our proposed framework is able to achieve comparable accuracy results in comparison to the model trained from scratch and converges with 1.42 times faster speed. These results further demonstrate the effectiveness and efficiency of our proposed incremental training framework.

REFERENCES


