APPENDIX A
ARTIFACT DESCRIPTION APPENDIX: [PRECOMPUTING OUTPUTS OF HIDDEN LAYERS TO SPEED UP DEEP NEURAL NETWORK TRAINING]

A. Abstract

Deep learning has recently emerged as a powerful technique for many tasks including image classification. A key bottleneck of deep learning is that the training phase takes a lot of time, since state-of-the-art deep neural networks have millions of parameters and hundreds of hidden layers. The early layers of these deep neural networks have the fewest parameters but take up the most computation.

In this work, we reduce training time by progressively freezing hidden layers, pre-computing their output and excluding them from training in both forward and backward paths in subsequent iterations. We compare this technique to the most closely related approach for speeding up the training process of neural network.

Through experiments on two widely used datasets for image classification, we empirically demonstrate that our approach can yield savings of up to 25% wall-clock time during training with no loss in accuracy.

B. Description

1) Check-list (artifact meta information):

- **Algorithm: Stochastic Gradient Descent**
- **Experiment workflow: Empirical evaluation**
- **Publicly available?: Yes**

2) How software can be obtained : The source code can be obtained from Github[1]

3) Hardware dependencies: The experiment was carried out on the Texas Advanced Computing Center (TACC) Maverick cluster with the following configuration:

- CPU - Intel Xeon E5-2680 v2 Ivy Bridge, 2.80 GHz,
- 20 CPUs/node,
- NVIDIA Tesla K40 GPU
- Cuda 7.5.

The experiment ran on single node of Maverick cluster with the above configuration. CUDA-capable GPU is preferable for training. However the open sourced script can run on CPU as well removing all .cuda() calls from the code.

4) Software dependencies: The source code uses Pytorch library [2] version 0.3

5) Datasets: The experiment uses the MNIST and CIFAR (CIFAR-10 and CIFAR-100) dataset. The script automatically downloads the dataset.

C. Installation

To run the experiment with basic configuration, following command is sufficient:

python train.py

But the script supports different arguments as defined in the FreezeOut Github page[3]

[1]https://github.com/50417/PreFast

[2]https://pytorch.org/

[3]https://github.com/ajbrock/FreezeOut

D. Experiment workflow

We run experiments with the same dataset on both PreFast[1] and FreezeOut[3] using different values of the user defined parameter t_0. Following python script run the experiment with t_0=0.1:

python train.py -t_0 0.1

E. Evaluation

In our work, we used FreezeOut's open source implementation[3] with default hyper parameter settings and incorporated our technique to empirically compare the results. The original Freezeout work validates its claim using the CIFAR-10 and CIFAR-100 datasets on the state-of-the-art DenseNet model. We compared our technique using both of the datasets along with the MNIST dataset.

Output pre-computation of the frozen layers required large memory size. Due to resource constraints, we took multiple random samples of 2000 training images for the CIFAR datasets and 4000 training images for MNIST to train the DenseNet model and record the training time. We then tested the DenseNet model trained on both techniques on the respective test dataset to compute validation errors. The empirical results suggest that for a user-defined parameter t_0 of 0.6, we could save additional 20-25% training time without losing on accuracy in comparison to FreezeOut.