Estimate the Performance of Different Approaches for Different Deep Neural Network Compressing Methods on Embedded System

1st Ruzhuo Wang
University of California, Riverside
Riverside, United States
rwang085@ucr.edu

***Abstract*—Recent advances in deep learning motivate the use of deep neutral networks in many applications, but their excessive resource needs on constrained embedded devices remain an important impediment. A recently explored solution space lies in compressing (approximating or simplifying) deep neural networks in some manner before use on the device.**

I trained two Deep Neural Network, one is Alexnet [1], one is Pilotnet [2]. And I apply two model compression methods for each Deep Neural Network (DNN) model, one is Singular-value decomposition (SVD) decomposition [3], one is dropout [4]. Also, there is two approach for applying SVD decomposition methods, I estimate two approach of SVD decomposition for each network on Raspberry Pi 3, but I only estimate one approach of dropout for each network on Raspberry Pi 3.

***Index Terms*—Model Compression, Deep Neural Network, Embedded System**

I. Introduction

A. Target problem

In order to use the DNN on resource constrained devices, an useful solution is compress the original DNN architecture. Among the DNN model compression methods, SVD decomposition and dropout turn out to be efficient and widely-used methods. So I want to estimate the performance for these two compression methods when I modify the specific parameters for these two methods.

Meanwhile, I find there can be two approaches for SVD decomposition method, I am also interesting in estimating the performance of these two compressing methods.

Then, the difference of architecture for the DNN models should also be taken into consideration.

B. Main contribution

- I successfully apply two different model compression methods for different DNN models, and I can modify the parameters for those two methods to have different performance for estimation.
- All the experiments for estimating the execution time performance are on the Raspberry Pi 3, so I build a experiment environment on embedded system.
- I find two approaches to apply SVD decomposition methods, and do experiments to test the performance of these two approaches.
- I successfully trained two functional DNN models, and do experiments to estimate the influence of different compression methods on the DNN models' accuracy and execution time.

II. Background and Related work

A. Background

1) Deep Neural Network:

A deep neural network (DNN) is an artificial neural network (ANN) with multiple hidden layers between the input and output layers. DNNs can model complex non-linear relationships. DNN architectures generate compositional models where the object is expressed as a layered composition of primitives. The extra layers enable composition of features from lower layers, potentially modeling complex data with fewer units than a similarly performing shallow network. [6]

2) SVD decomposition:

In case of a fully-connected layer, updating states of all nodes requires evaluating the product: $W^L \cdot x^L$, where, $x^L \in \mathbb{R}^n$ is the state of nodes in the previous layer and $W^L \in \mathbb{R}^{m \times n}$ is the matrix representing all the connections between layer L and $L+1$. Now, the basic idea in decreasing the number of required computations is to replace the weight matrix W^L with a product of two different matrices, i.e.,

$$W^L = U \cdot V$$

Under SVD, the weight matrix can be efficiently factorized as:

$$W^L_{m \times n} = X_{m \times m} \cdot \Sigma_{m \times n} \cdot N^T_{n \times n}$$

where, $\Sigma_{m \times n}$ is a rectangular diagonal matrix containing L *singular values* of $W^L_{m \times n}$ as the diagonal elements. To gain computational efficiency the weight matrix can be approximated well by keeping k highest singular values, i.e.:

$$W^L_{m \times n} \approx X_{m \times m} \cdot \Sigma_{k \times k} \cdot N^T_{n \times n}$$

Now, the architecture of a fully-connected layer of a deep model can be modified by replacing W^L with $U = X_{m \times k}$ and $V = \Sigma_{k \times k} \cdot N^T_{k \times n}$, as is shown in Figure1.

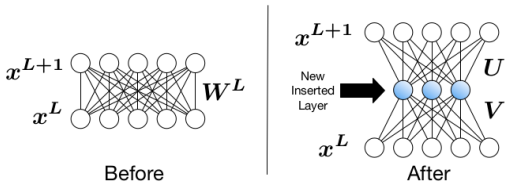


Fig. 1: Layer insertion for SVD decomposition

3) Dropout:

Dropout prevents overfitting and provides a way of approximately combining exponentially many different neural network architectures efficiently. The term dropout refers to dropping out units (hidden and visible) in a neural network. By dropping a unit out, it means temporarily removing it from the network, along with all its incoming and outgoing connections, as shown in Figure2.

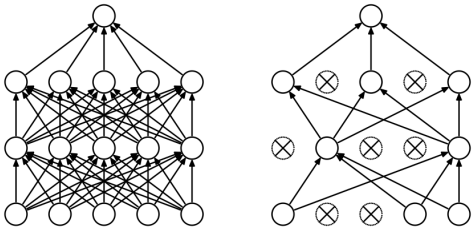


Fig. 2: Dropout Neural Net Model. $Left$: A standard neural net with 2 hidden layers. $Right$: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

B. Related work

There is a paper, in which they build a Compressor-Critic Framework called $DeepIoT$ [7], it obtain the optimal dropout probabilities for the neural network and exploits the network parameters themselves. In fully-connected neural networks, neurons are dropped in each layer; in convolutional neural networks, filters are dropped in each layer; and in recurrent neural networks, dimensions are reduced in each layer. This means that DeepIoT can be applied to all commonly-used neural network structures and their combinations.

Based on their job, I apply dropout method for fully-connected neural networks and convolutional neural networks. Beside that, I also apply two approaches of SVD decomposition for two different DNN models. Then I estimate the performance on the Raspberry Pi 3.

III. SYSTEM FRAMEWORK

A. Software

I use $Caffe$ [8] as deep-learning framework. Caffe is a deep learning framework made with expression, speed, and modularity in mind. It is developed by Berkeley AI Research (BAIR) and by community contributors. Yangqing Jia created the project during his PhD at UC Berkeley. Caffe is released under the BSD 2-Clause license.

B. Hardware

I use Raspberry Pi 3 Model B [5] for estimating the execution time performance of different DNN models. The Raspberry Pi 3 Model B is the earliest model of the third-generation Raspberry Pi, it has following basic parameters:

- Quad Core 1.2GHz Broadcom BCM2837 64bit CPU
- 1GB RAM

C. Two DNN architecture

1) Pilotnet: The original Pilotnet consists of 9 layers, including a normalization layer, 5 convolutional layers and 3 fully-connected layers. It uses strided convolutions in the first three convolutional layers with a 2×2 stride and a 5×5 kernel and a non-strided convolution with a 3×3 kernel size in the last two convolutional layers. Based on this architecture, I build a modified Pilotnet based on SullyChen's project [9], which deletes one normalization layer, add 7 relu layers, add two dropout layers. As is shown in the Figure3.

2) Alexnet: $imagenet - classification - with - deep - convolutional - nn.pdf$ The first convolutional layer filters the $224 \times 224 \times 3$ input image with 96 kernels of size $11 \times 11 \times 3$ with a stride of 4 pixels (this is the distance between the receptive field centers of neighboring neurons in a kernel map). The second convolutional layer takes as input the (response-normalized and pooled) output of the first convolutional layer and filters it with 256 kernels of size $5 \times 5 \times 48$. The third, fourth, and fifth convolutional layers are connected to one another without any intervening pooling or normalization layers. The third convolutional layer has 384 kernels of size $3 \times 3 \times 256$ connected to the (normalized, pooled) outputs of the second convolutional layer. The fourth convolutional layer has 384 kernels of size $3 \times 3 \times 192$, and the fifth convolutional layer has 256 kernels of size $3 \times 3 \times 192$. The fully-connected layers have 4096 neurons each. I modify the final fully-connected layer to be 2, because I trained it to classify between cats and dogs.

D. Main assumption

- Change the value of the weight matrix for a pre-trained DNN model will not influence the execution time performance. And the build in dropout layer for Caffe with randomly set the value in the nodes to be zero based on the given dropout parameters.
- For Alexnet, I apply the SVD decomposition method for the weight matrix between $pool5$ layer and $fc6$ layer, and apply dropout method for $conv3$ layer and $fc6$ layer, since they take relatively long time to be executed.
- For Pilotnet, I apply the SVD decomposition method for the weight matrix between $relu5$ layer and $fc6$ layer, and apply dropout method for $conv2$ layer and $fc6$ layer, since they take relatively long time to be executed.

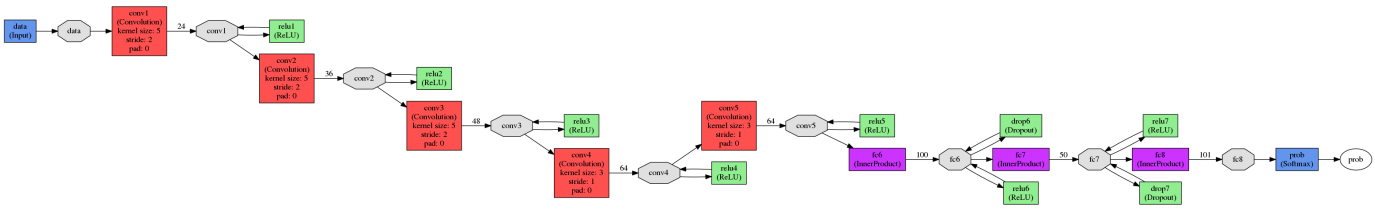


Fig. 3: Modified Pilotnet architecture

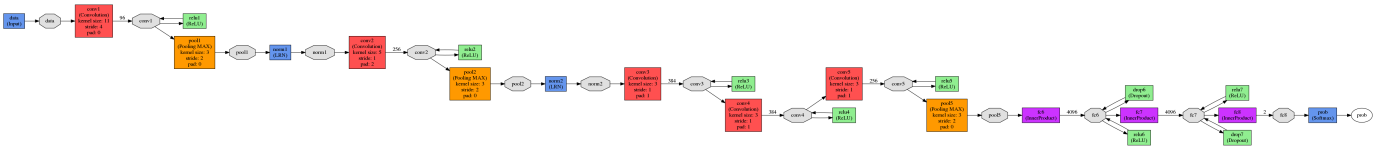


Fig. 4: Modified Alexnet architecture

- For the pre-trained DNN model, I assume their test accuracy cannot increase anymore even for more training iterations.

IV. Proposed Work

The key idea behind my work is that, since we know the theoretical formula for SVD decomposition and dropout, but how about implement them for specific DNN models using different approaches? If we can predict the accuracy and execution time given the compression parameters, it is helpful to determine the compression parameters for some specific budget on resource constrained devices.

V. Evaluation

A. Experiment setup

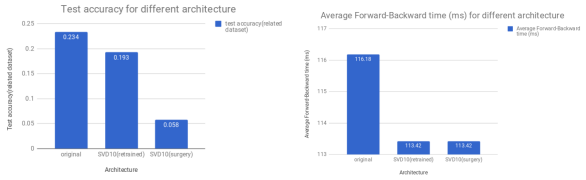
1) Two different approaches for SVD decomposition:

- Retrained approach: Based on the original DNN architecture, I insert a new fully-connected layer between two adjacent fully-connected layers. Then I train and test this new DNN model using the same training and validation data.
- Surgery approach: For a pre-trained DNN model, I write a python script(SVD_surgery.py) to generate a new DNN model without retraining the DNN model. First, the SVD_surgery.py storage all the parameters for the layers except the target two adjacent fully-connected layers. Then it use SVD decomposition for the weight matrix W that corresponding to those two adjacent fully-connected layers to generate three matrix X, Σ, V, keep k highest singular values of matrix Σ, and replace W^L with $U = X_{m\times k}$ and $V = \Sigma_{k\times k} \cdot N^T_{k\times n}$.

2) Application of dropout method:

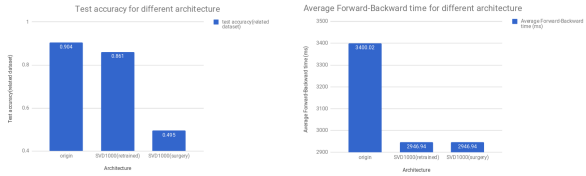
For both Alexnet and Pilotnet, based on the original DNN architecture, I decrease the number of output feature map for convolutional layer and decrease the number of nodes for fully-connected layer. Then, I retrain and test the modified DNN model using the same training and validation data as the original DNN model use.

B. Experimental results



(a) Test accuracy for different architectures (b) Test execution time for different architectures

Fig. 5: The accuracy and execution time estimation of Pilotnet for two approaches



(a) Test accuracy for different architectures (b) Test execution time for different architectures

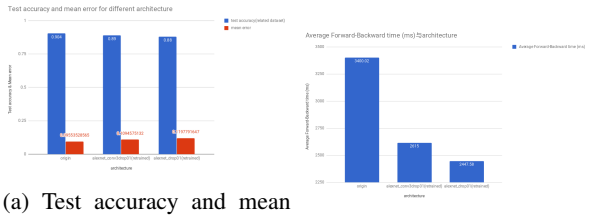
Fig. 6: The accuracy and execution time estimation of Alexnet for two approaches

1) The accuracy and execution time performance for two different approach:

The $SVD10$ in the Figure5 means I insert a fully-connected layer with 10 nodes between $relu5$ layer and $fc6$ layer. The $SVD1000$ in the Figure6 means I insert a fully-connected layer with 1000 nodes between $pool5$ layer and $fc6$ layer.

As for the Pilotnet, for the test accuracy result, I can tell that SVD decomposition method will decrease the prediction accuracy, and the pre-trained approach has much higher test accuracy than Surgery approach. For the test executing time result, I can tell that both approaches can decrease the execution time.

As for the Alexnet, for the test accuracy result, I can tell that SVD decomposition method will decrease the prediction accuracy, and the pre-trained approach has much higher test accuracy than Surgery approach. For the test executing time result, I can tell that both approaches can decrease the execution time.



(a) Test accuracy and mean error for different architectures

(b) Test execution time for different architectures

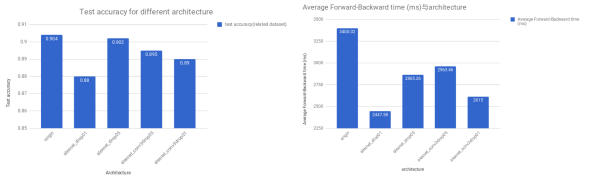
Fig. 7: The accuracy, mean error and execution time estimation of Alexnet when applying dropout method on different layer

2) The accuracy and execution time performance for dropout method:

The $conv3drop01$ in Figure7 means drop 90% of the feature map for $conv3$ layer. The $drop01$ means drop 90% of the nodes for $fc6$ layer.

When I apply dropout method for the fully-connected layer, the prediction accuracy slightly decrease and the execution time decrease. When I apply dropout method for the convolutional layer, the prediction accuracy slightly decrease and the execution time decrease.

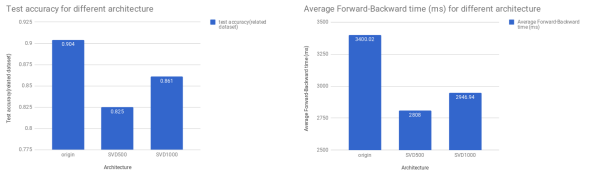
3) Change the parameters:



(a) Test accuracy for different architectures

(b) Test execution time for different architectures

Fig. 8: The accuracy, and execution time estimation of Alexnet when changing the dropout compression parameters



(a) Test accuracy for different architectures

(b) Test execution time for different architectures

Fig. 9: The accuracy, and execution time estimation of Alexnet when changing the SVD decomposition parameters

In Figure8, I can tell that the more feature map I drop for $conv3$ layer, the higher execution time benefit I can achieve, meanwhile, the lower accuracy I will get. Also, the more nodes I drop for $fc6$ layer, the higher execution time performance I can achieve, meanwhile, the lower accuracy I will get.

In Figure9, I can tell that the more nodes I insert between $pool5$ layer and $fc6$ layer, the worse execution time benefit I will achieve, the lower accuracy decrease I will get.

VI. Conclusion

- Both retrained approach and surgery approach can have execution time benefit, but as for accuracy, surgery approach has much more accuracy decrease than retrained approach.
- By applying dropout method for both convolutional layer and fully-connected layer, the execution benefit can always achieve at the cost of the accuracy decrease.
- The more nodes I insert between the adjacent fully-connected layer, the worse execution time benefit I will achieve, the lower accuracy decrease I will get.
- The more feature map I drop for convolutional layer, the higher execution time benefit I can achieve, meanwhile, the lower accuracy I will get. Also, the more nodes I drop for fully-connected layer, the higher execution time performance I can achieve, meanwhile, the lower accuracy I will get.

VII. Future work

- I do have an idea about surgery approach for dropout method, but the prediction result is bad for Pilotnet, I should find a way to fix it.
- The reason why most of my experiment result is about Alexnet is that, when I try to apply different approach for SVD decomposition for Pilotnet, the prediction result is bad, also when I try to dropout the Pilotnet, the prediction result is bad, so I should find way to fix it.
- Rather than two DNN models, I should apply these compression methods on more DNN models.
- I can only get a approximate prediction about the accuracy decrease and execution time benefit given the compression parameters, I need to change the compression parameters in a small step size, try to find a formula to relate the accuracy decrease and execution time benefit with the compression parameters.

References

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton, *ImageNet Classification with Deep Convolutional Neural Networks* .

[2] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao and Karol Zieba, *End to End Learning for Self-Driving Cars*, 2016.

[3] Sourav Bhattacharya and Nicholas D. Lane, *Sparsification and Separation of Deep Learning Layers for Constrained Resource Inference on Wearables*.

[4] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever and Ruslan Salakhutdinov, *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*, 2014.

[5] Raspberry Pi 3 Model B, `https://www.raspberrypi.org/products/raspberry-pi-3-model-b/`.

[6] Deep Neutral Networks, `https://en.wikipedia.org/wiki/Deep_learning`.

[7] Shuochao Yao, Yiran Zhao, Aston Zhang, Lu Su and Tarek Abdelzaher, *DeepIoT: Compressing Deep Neural Network Structures for Sensing Systems with a Compressor-Critic Framework*, 2017.
[8] Caffe, `http://caffe.berkeleyvision.org/`.
[9] Autopilot-TensorFlow, `https://github.com/SullyChen/Autopilot-TensorFlow`.