Modified Branchynet

2018-2nd Semester

Interdisciplinary Project

Student ID	20131504
Name	임정택
School	자연과학부
1 track	MTH
2 track	CSE
Advisor 1	최진혁
Advisor 2	이경한

I. Introduction

1) Topic and Purpose of Research

Deep neural networks are methods for many learning tasks but the performance of additional layers in the networks incurs serious latency problem and energy consumption. With appearances of bigger and deeper network, BranchyNet[1] is one of the solution for these problems. If the samples already can be confirmed with high confidence, the branchyNet will have fast inference with early exit branches. With the growing of sizes of networks, performance improvement became important issue. another solution for this big size networks is compression of connections[2]. It reduces the number of Network connections. The purpose of this project was to make advanced improvement by using these 'BranchyNet' and 'compression of connections'. During the progress of the project, the final goal was adjusted. Now the goal of this project is to make improvement of the 'BranchyNet'. Two ways to improve BranchyNet will be suggested. Removing consecutive layers and replacing with a new layer is the first way to improve BranchyNet. Second way is substitution of exactly same computations that was already calculated. They will be discussed in latter part of this report.

II. Main Subject

1) Research Planning

First step was study of BranchyNet[1] and Neurosurgeon[3] because the first goal was mobile application combined with deep learning technologies. The mobile applications require higher reductions of latencies and energy costs. After read these two, the plan was changed to more focus on Branchynet. With a computer with NVIDIA GeForce GTX 1070, we tested BranchyNet codes on github(https://github.com/kunglab/branchynet). The Neurosurgeon gives an idea of improvement. The neurosurgeon splits network and computes some part on mobile device, and other part on cloud to get effective reductions of latency or energy cost. The part that should be done on cloud would be effective and it requires usually more computations. ″What if we compress it before we send the part of network into the cloud?″ was the first idea. From that idea, the idea was changed into compression of neural network. After the research, I tried to test implemented BranchyNet on github with Ubuntu18.04 on Virtual Box, but it doesn't work because 'Cuda' doesn't support VBox's own virtualized graphic card[5]. Further, Ubuntu 18.04 on host OS also had a problem with NVIDIA GeForce[4]. So to fix this problem, we remove graphic card from the computer and reinstall Ubuntu 18.04 and then, re-equip the graphic card. After the installing Ubuntu, by using git clone, we downloaded the implemented Branchynet codes, and make some changes with the ideas.

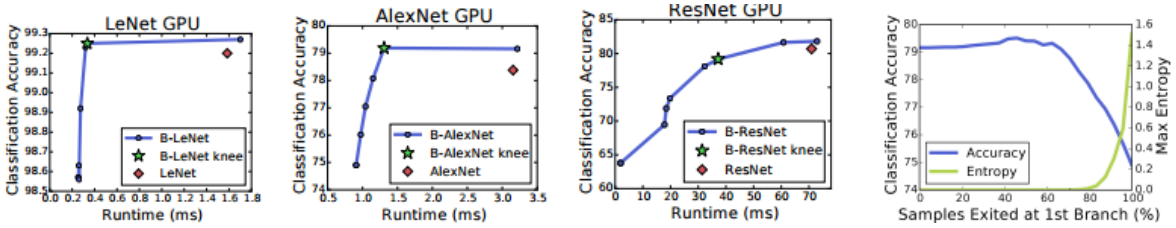
(Actually, original BranchyNet code couldn't be run under Ubuntu 18.04 because of some compatibility problems of versions of chainer and Cuda with NVIDIA driver. So, testing environment was changed to Ubuntu 16.04.)

III. Conclusion and Discussion

1) Research Results

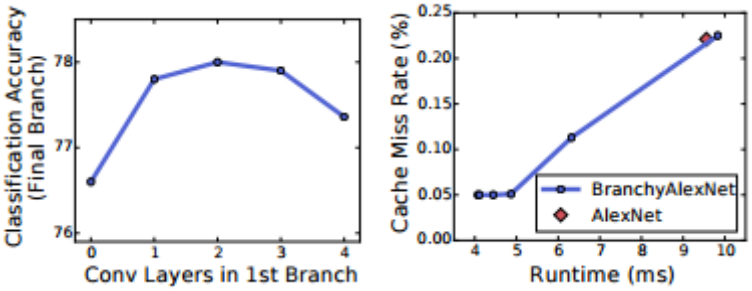
1)-1.BranchyNet from previous researches. (already done in BranchyNet[1])

Previous researches was tested on three well-known convolutional neural networks: LeNet[6], AlexNet[8], and ResNet[7],



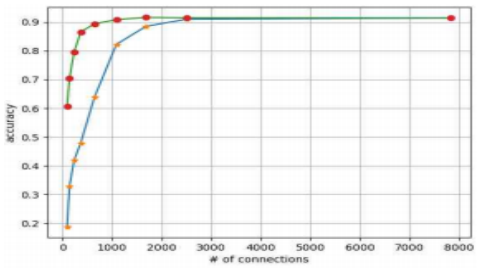
GPU performance results for BranchyNet when to LeNet, AlexNet and ResNet

The original network accuracy and runtime are the red diamonds. The star denotes a knee point in the curve. The CPU performance results have similar characteristics. Runtime is measured in milliseconds(ms) of inference per sample. For this evaluation, this previous research used batch size 1 as evaluated in [9, 10].

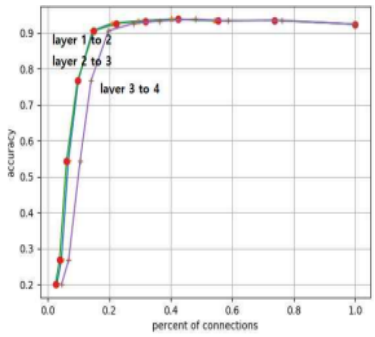


The impact of the number of convolutional layers. (left)
the runtime and CPU cache miss rate for the B-AlexNet model when the entropy threshold is varied. (right)

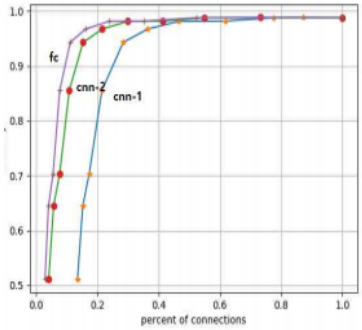
1)-2. Compression of Connection from previous researches. (already done in Compression of Connection[2])



The left figure shows degradation of accuracy and retraining performance varying the pruning ratio of FCN-1. Neural network has huge number of multi-layer network structure. Each layer has different structure of connection, so same weight doesn't means same effect. Therefore, considering of properties for each layer is important.

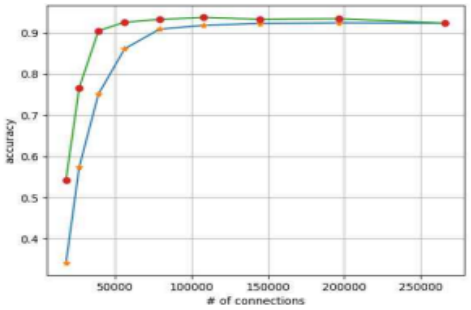


(a) FCN-2

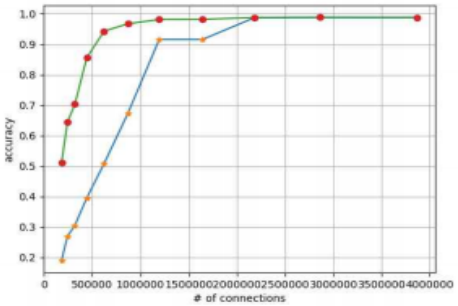


(b) CNN-3

For each layer, maintaining all states of all other layer and adjust only one layer and compute accuracy and error rate for the layer. This was done for FCN-2 and CNN-3, each layer pruning performances in FCN-2 and CNN-3 system.



(a) FCN-2 retraining result

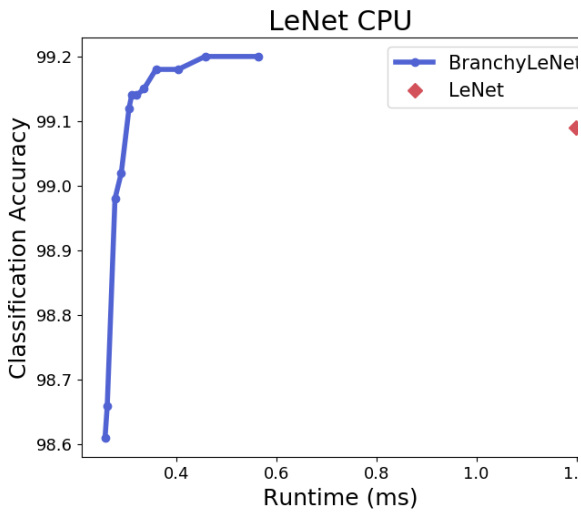


(b) CNN-3 retraining result

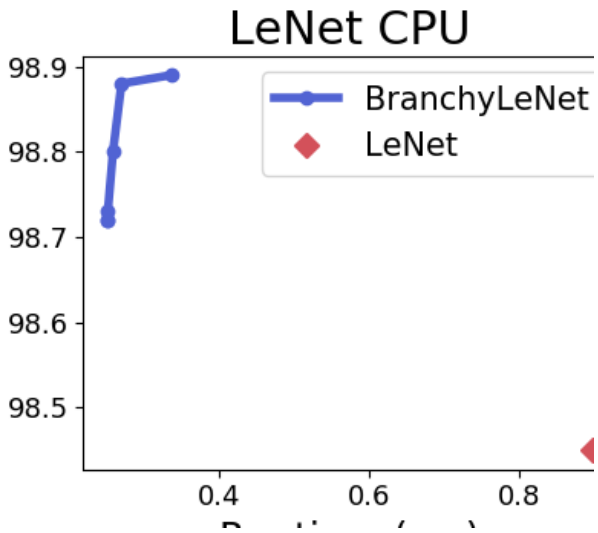
Both shows that reduction of lower layer is more effective with considering of accuracy and computational benefits, but the rate is not constant.

The figure right is Pruning (proposed Algorithm in [2]) ratio vs classification accuracies in FCN-2 and CNN-3 system.

1)-3. Modified Branchynet.



LeNet and B-LeNet



Modified LeNet and modified B-LeNet

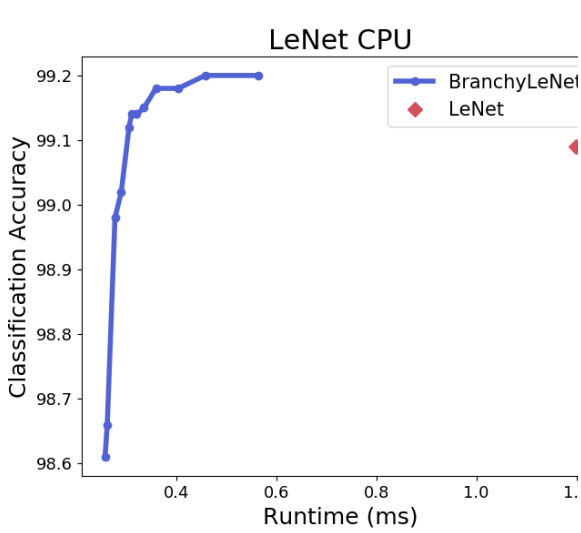
Modification is done in branchynet/network/lenet_mnist.py

By combining two consecutive layers into one layer, reduction of time consuming is done.

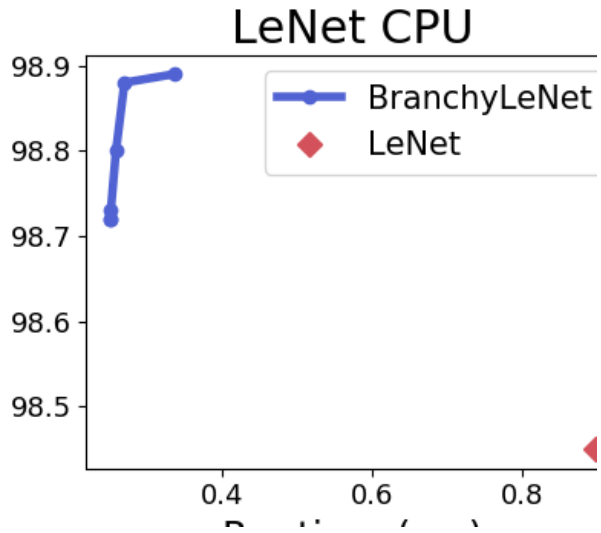
2) Discussion

In this discussion section, there are two improvement suggestion for BranchyNet. First suggestion is removing consecutive two layers and put new(combined) layer. Second suggestion is finding exactly same computations. The first suggestion was partly implemented.

2)-1. Removing consecutive two layers and put new layer.



LeNet and B-LeNet



Modified LeNet and Modified B-LeNet

Modified B-Net LeNet shows better performances in terms of time. The Modified LeNet is implemented by removing two consecutive layers, and a new(combined) layer will replace them.

Now, where to combine to get better network is an issue.

```
FindBestNetwork(Layers)
    BestLayers = Layers
        for i
            NewLayers[i] = Combine(Layers[i], Layers[i+1])
            for j < i
                NewLayers[j] = Layers[j]
            for j > i
                NewLayers[j] = Layers[j+1]

            if  IsbetterLayers(NewLayers, BestLayers)
                BestLayers = NewLayers
    if  BestLayers == Layers
        return  Layers
    return  FindBestNetwork(BestLayers)
```

Algorithm : Where to combine to get best network

For an network which has n-layers, if we combine consecutive two layers in the network, then there are n-1 cases of consecutive combining. Among these n-1 cases, choose one best network in terms of latency or energy and so on, and then repeat this recursively.

Notice that best network can be original network. This means this algorithm still need to be improve.

This algorithm does not guarantee to always gives us better network, but just give us possibilities of existence of better network. Fortunately, there were better modification for LeNet cases as shown in the graph above. It's needed to check that this algorithm works for other networks like AlexNet and ResNet.

2)-2. Finding exactly same computations.

Another suggestion to improve BranchyNet is finding same computations. Neural networks usually likely contain very big matrices and heavy computations in the networks. Matrix to matrix computations might be repeated several times in the very same way. Therefore, when we compute some part of matrix, the exactly same computation can appear again. If exactly same computations appear again and again, we can just save the result of the computation and substitute former result into latter computation. However, this idea have three issues, first, it's not guaranteed that the exactly same computations really exist and repeat several times. If every single computations in a network are different, or exactly same computations appear just few times, it will be not so useful. Second, cost of finding exactly same computation can be too high. Third, longest exactly same computations can be too short, so substitution can be heavier than the computation.

Classified collection of same computations will be needed when it saves exactly same computation. The classified collection can be done with hash function. Two different hash value guarantees they are different computations. We suggest to check how many times each hash values appear and sketch distribution of hashs. Most frequent hash value should be check first when you find same computations. The list of hashs in the order of frequency would be helpful to do that. To minimize total number of computation, we suggest you taking part of computations as a sample set. With the randomly chosen sample set, we can assume that the overall distribution of hash is same as distribution of the sample.

※References

[1] Surat Teerapittayanon, Bradley McDnel, H.T. kung , BranchyNet: Fast Inference via Early Exiting from Deep Neural Networks, arXiv:1709.01686, 2017

[2] Ahn Heejune, A Study on Compression of Connections in Deep Artificial Neural Networks, dbpia, 2017 http://www.dbpia.co.kr/Article/NODE07263082

[3] Y kang, J Hauswald, C HGao, A Rovinski, T Mudge, Neurosurgeon: Collabarative intelligence between the cloud and mobile edge, ACM SIGPLAN 2017.

[4] E Raggi, K Thomas, T Parsons A Channele, Beginning Ubuntu, (Solving Installation Problems) ,springer, 2010.

[5] https://developer.nvidia.com/cuda-gpus

[6] Y.LeCun, L.Bottou, Y.Bengio, and P.Haffner. Gradient-based learning applied to document

recognition. Proceedings of the IEEE, 86(11):2278-2324, 1998.

[7] K.He, X.Zhang, S.Ren, and J.Sun. Deep residual learning for image recognition. arXiv preprint arXiv:1512.03385, 2015.

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems, pages 1097-1105, 2012

[9] S. Han, H. Mao, and W.J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv:1510.00149, 2015

[10] Y-D. KIm, E. Park, Yoo, T, Choi. L. Yang, and D. shin. Compression of Deep convolutional neural networks for fast and low power mobile applications. arXiv:1511.06530, 2015

[11] C. Bucilu, R. Caruana, and A. Niculescu-Mizil. Model compression. In proceedings of the 12th ACM SIGKDD international conference on knowledge discovery and data mining, pages 535-541. ACM, 2006