Towards Accurate Binary Convolutional Neural Network

E4040 2019Fall ZXCV Report

Qichen Hu (qh2199), Xuechun Zhang (xz2795), Yingtong Han (yh3067), Zoran Kostic zk2172
Columbia University

Abstract

This report presents the results of an attempt to implement the ideas of binarizing convolutional neural network from the paper of Xiaofan Lin, Cong Zhao and Wei Pan [2017], and to replicate the results of their experiments. The paper introduces ABC-Net to approximate the full-precision weights with linear combinations of binary bases and activating the layers with multiple binary activations, which reduces the accuracy degradation compared to Binary neural networks (BNNs) and enhances the computational speed. The main technical challenge is to design a new complex convolution layer with binary weights and train the hyperparameters using TensorFlow. Although we implemented the model mostly as what the paper described, we could not observe the advantages brought by this new structure.

1. Introduction

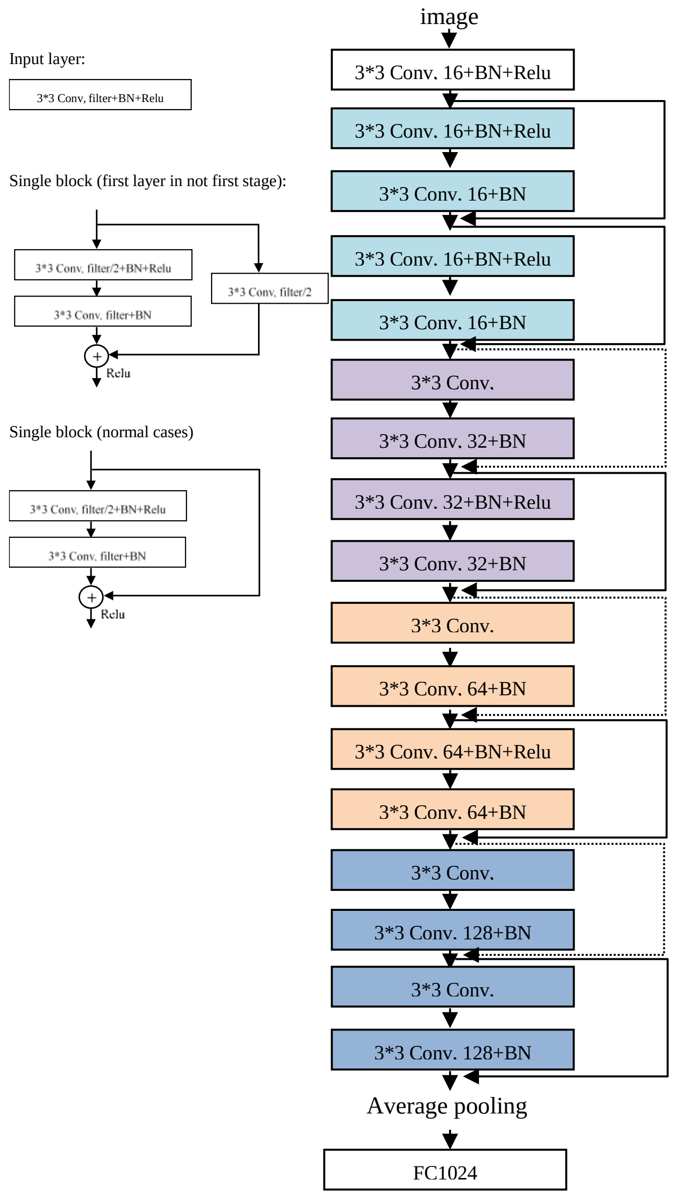
Convolutional Neural Networks (CNNs) has gained increasing attention in the field of mobile applications, due to its advantages in image classification. However, the employment of CNNs on mobile applications faces a big challenge that the test process costs too much computing power and incurs large hardware costs. Therefore, they cannot meet the commercial requirements of mobile apps. Binary neural networks (BNNs) are introduced by Courbariaux et al. [2016] and Rastergari et al. [2016] to tackle the cost problems, because BNN uses binary weights and activations which could reduce memory storage and speed up the test process, while at the cost of severe prediction accuracy degradation. To balance the test cost and the accuracy, this paper introduces Accurate-Binary-Convolutional (ABC-Net) [Lin et al., 2016]. Firstly, this model does not train the full-precision weights of CNNs. Instead, it utilizes a combination of binary bases to approximate the full-precision weights. Secondly, the authors use multiple binary activations to reduce the loss caused by binarization. As more binary weight bases and activations are combined in the model, the prediction accuracy ideally will be improved to a level close enough to the full-precision networks. The breakthrough of this paper lies in the fact that it is the first time that a binary neural network obtains a comparable result compared to the full-precision model.

Our project aims to implement ABC-Net proposed in this paper, and to replicate the results of their experiments. The main technical difficulty is to replace the convolution layer with ABC layers with or without binary activations, which could speed up the test process. We have tried based on TensorFlow 1.13, TensorFlow.Keras and Keras to build the layers from the bottom and would like to present our results in this paper.

2. Summary of the Original Paper

2.1 Methodology of the Original Paper

First, the original paper [Lin et al., 2016] approximates the full-precision weights of a CNN with binary weights and processes the input of each convolution layer with binary activations. It proposes the block structure of the approximated convolution layer (see below)



To show the advantages that ABC-Net could bring, they evaluated the neural network mainly on ILSVRC12 ImageNet classification dataset with ResNet as network topology [Deng et al., 2009].

First, they assessed the Top-1 and Top-5 prediction accuracy of ABC-Net on ImageNet with different numbers of binary weight base M and full precision activation and then compared the results with BNNs and the full precision network.

Then they changed the full precision activations to binary activations and combined different M and N, the number of binary activations, and then compared the performance in different network topology from ResNet18, ResNet34 to ResNet50 to check whether an obvious difference exists in these cases compared to the full-precision model.

In the third experiment, to further prove the cutting-edge property, the performance of ABC-Net is compared with the state-of-the-art models such as DoReFa-Net, XNOR-Net and BNN.

2.2 Key Results of the Original Paper

The first key result of their paper is that the experiments show that ABC-Net could approximate the accuracy of the full-precision model provided that the number of binary weight base M and the number of binary activation base N are properly chosen. When M and N increase, the prediction accuracy of ABC-Net elevates and the gap between the full-precision model and ABC-Net narrows. This effect is obvious for all ResNet models they have tested, including ResNet18, ResNet34 and ResNet50.

Secondly, for ABC-Net with and without binary activations, they could well approximate the full-precision ResNet18 model, which is also the best-performed model among these models tested, which means that the accuracy degradation is effectively mitigated.

3. Methodology

In this section, we would present how we replicate the paper. We would also show the main technical challenge in this project, which is to implement the whole algorithm of the highly customized ABC layer with TensorFlow basic language. The second difficulty lies in how to incorporate the ABC layer into the ResNet model. Then we would introduce how we overcome these difficulties.

3.1. Objectives and Technical Challenges

The original paper shows 3 series of experiments as we mentioned before. Our goal is to implement ABC-Net introduced by the paper and to verify whether or not these 3 experiments are convincing.

The first technical challenge is to build the framework of ResNet with bottom language. Although we have learnt to code networks such as LeNet, ResNet has a more complex and flexible structure.

The second challenge is how to incorporate the ABC layer into the original full-precision model. ResNet itself has a relatively complicated structure and it took us some time to figure out how to replace the convolution layer in ResNet with ABC layer.

The third challenge is that we need to save and use the pretrained weights of ResNet models as the initial values in ABC layers. How to use them is another problem.

3.2. Problem Formulation and Design

We built the ResNet graph by building basis functions with TensorFlow 1.13. We tried Keras and submitted the Keras version of ABC-Net, but found that Keras, as an advanced API, provides less flexible customized layer and may not be a good choice in this case.

We then figured out the block diagram of ResNet20 and ResNet34. The authors use ResNet18, but we changed it to ResNet20, because the loop structure is easier to realize, and it is comparable to ResNet18 in terms of model capacity. ResNet models are composed of ResNet blocks and the difference from the traditional CNN is that at the end of each block, we need to add the input layer and the output of the last layer of this block. The property holds for all the ResNet model. For 20-layer, the convolution layer structure is as follows:

Conv1	1 layer, output filters number is a
Conv2_ResNet stage 1	$\begin{bmatrix} 3*3, & a \\ 3*3, & a \end{bmatrix} * 3$
Conv3_ResNet stage 2	$\begin{bmatrix} 3*3, & 2a \\ 3*3, & 2a \end{bmatrix} * 3$
Conv4_ResNet stage 3	$\begin{bmatrix} 3*3, & 4a \\ 3*3, & 4a \end{bmatrix} * 3$
Dense layer	1 layer, input filters 4a, flatten before putting into this layer

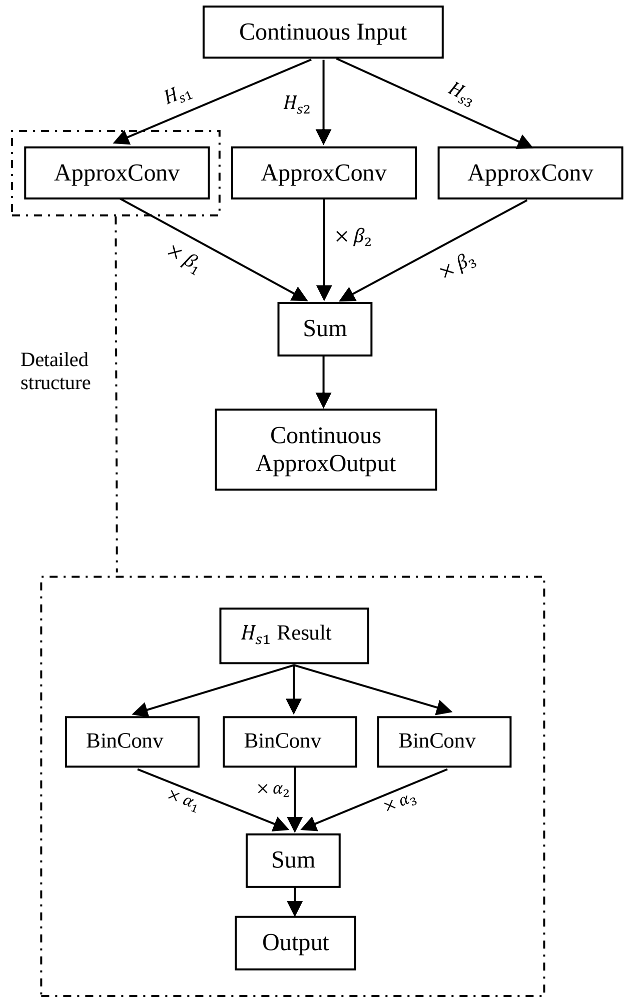
For 34-layer, the convolution layer structure is as follows:

Conv1	1 layer, output filters number is a
Conv2_ResNet stage 1	$\begin{bmatrix} 3*3, & a \\ 3*3, & a \end{bmatrix} * 3$
Conv3_ResNet stage 2	$\begin{bmatrix} 3*3, & 2a \\ 3*3, & 2a \end{bmatrix} * 4$
Conv4_ResNet stage 3	$\begin{bmatrix} 3*3, & 4a \\ 3*3, & 4a \end{bmatrix} * 6$
Conv5_ResNet stage 4	$\begin{bmatrix} 3*3, & 8a \\ 3*3, & 8a \end{bmatrix} * 3$
Dense layer	1 layer, input filters 8a, flatten before putting into this layer

Note: For the above two tables, in each ResNet block, it indicates how many ResNet blocks each stage has. For example, $\begin{bmatrix} 3*3, & a \\ 3*3, & a \end{bmatrix} * 3$ means that for this stage, there are 3 blocks, each block having 2 convolution layers, each layer having a kernel size of [3,3], the number of filters being a.

To make sure that we can add them without the shape problem, we passed the input layer through a convolution layer with strides of 2 and then added it to the last layer in this block.

As for the second challenge, the original paper provides a simple example explaining how to incorporate the ABC layer. We elaborated it into the larger ResNet model. We replaced each convolutional layer with the combination of binary filters so that we could use it in our ResNet model. When we implemented it, we wrote the ABC layer as a function so that we could call it every time we built a convolutional layer, which saved much time.



For the third problem, considering that our ResNet model may be a little bit different from the pretrained model, we trained the full-precision model at first to extract the weights. When we trained the model, we created a dictionary for storing values which could be reused in ABC-Net training.

4. Implementation

In this part, we would describe our implementation in detail, including the block diagram, the illustration of our training algorithms, the flow charts and the dataset we used.

4.1. Deep Learning Network

4.1.1 Architectural block diagram

ResNet20 and ResNet34 are our network topologies. We have described both the full-precision ResNet20 and ResNet34 model in the previous session and also introduced how to replace each convolution layer with ABC layer in a graph.

We built the ABC-Net with binary activations. We experimented this on both ResNet20 and ResNet34 with different (M, N) sets.

- We used Resnet20 and Resnet 34 as our network topologies and tested different (M, N),
- For Resnet20, we trained the full-precision model first and then we tried different sets of (M, N) including (3, 3), (5, 3) and (5, 5) on the basis of the pre-trained full-precision ResNet20.
- For Resnet34, we trained the full-precision model first and then we chose different sets of (M, N) including (3, 3) and (5, 5) based on the pre-trained full-precision ResNet34.
- We compared the performances among ResNet models and also with other state-of-art models.
- We also tried a simple ABC-Net based 2-layer CNN for further research purpose.

4.1.2 Training algorithm details

The detailed implementation of the training algorithm has been shown in the next section 4.2 with both step-by-step algorithm and pseudo code.

4.1.3 Flowchart(s)

Please see the figure on the next page for the flowcharts of ABC-Net based on ResNet20.

4.1.4 Data Used

We implemented ABC-Net on Cifar-10 dataset rather than ImageNet, because Cifar-10 is more accessible while ImageNet picture URLs are invalid now. Since Cifar-10 only has 10 classes, we did not present Top-5 accuracy in our study.

4.2. Software Design

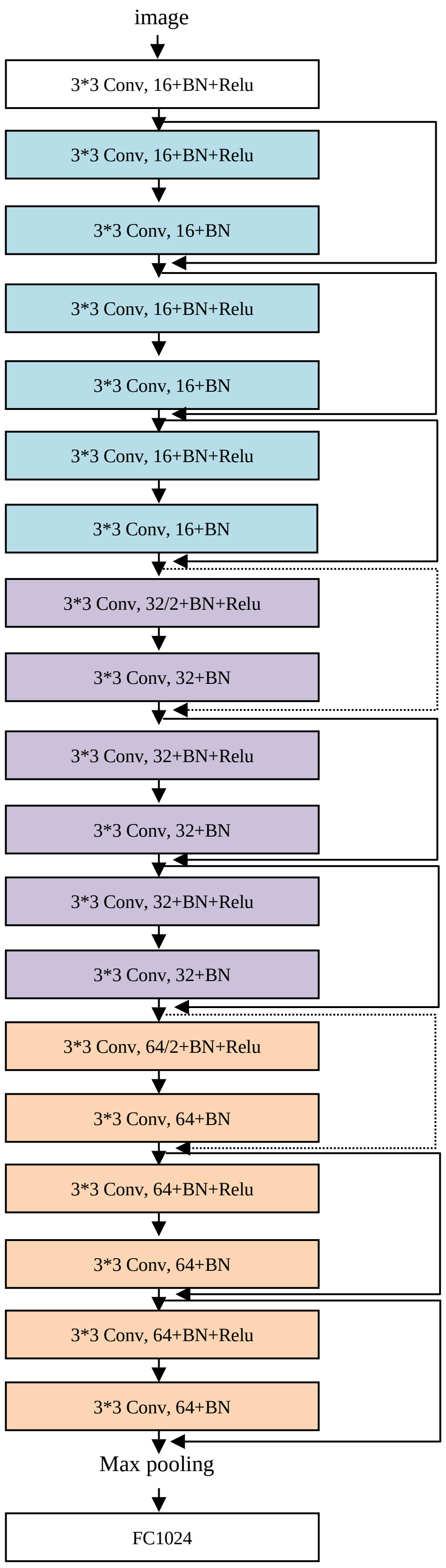
4.2.1 Flow chats

For ABC-Net based on ResNet, the top flow charts are as follows (please see next page). We only present the graph of ResNet20 here because the graph of ResNet34 is very much similar to it except that there are 4 stages and 2 blocks in each stage.

4.2.2 Algorithms

We would like to present the step-by-step algorithms. Let us take M = 3 and N =3 as an example. For ResNet20 based ABC-Net, the forward process can be described as following

The input goes through an ABC layer before it enters the ResNet stages. Here, ResNet20 has 3 stages and 3 basic blocks in each stage as shown before. For each block in the first stage, the input layer goes though 2 ABC layers consecutively and the input layer is added with the output



layer after the above operation. The sum is the input of the next block.

For the blocks in the second and third stage, if the block is the first block in that stage, the input layer goes through a down-sampling ABC layer with strides of 2 and the number of filters increases to 2 times. Then it enters the normal ABC layer again to get an output layer. After this layer, the input layer of this block goes through a down-sampling ABC layer with strides of 2 and the number of filters increase to 2 times so that the input layer could be added with the output layer and the sum is the input for the next block again.

But if the block is not the first block in that stage, the input layer just goes through the same process as the input layers in the first stage. After the last stage, the output layer should go through the pooling layer with strides of 2 and then go through a flatten layer which reshapes the output into a layer with dimension of 1. Then the flattened output goes through a dense layer, an output layer and a softmax activation layer.

In each ABC layer of the input data, there are 3 approximated convolution layers, weighted by β_j, which is the hyperparameters of the neural network and will be trained. Since the binary bases are fixed at first, given the pretrained weights for W, we have the expression below for approximation

$$W \approx \alpha_1 * B_1 + \alpha_2 * B_2 + \ldots + \alpha_M * B_M$$

where B_i is the binary filter i and M is the number of binary filters. The approximated convolution layer gets α_i by optimizing the following equation

$$min_{\alpha} J(\alpha) = || \omega - \alpha B||^2$$

The input is added to 3 shift parameters before it is clipped into 3 binary variables, each of which goes through the binary convolution layer B_i to get an output layer. Weighted by α, the 3 branches merge and compose one approximated convolution layer. The output of ABC layer is the weighted sum of the three approximated convolution layers with weights equal the hyperparameters β.

For ResNet34 based ABC-Net, the algorithm is quite similar. The only difference is that in ResNet34, there are 4 stages and 2 blocks in each stage.

4.2.3 Pseudo

```
1. Preparation: calculate binary_weights[]
Inputs: W, M
    mean, std = W's mean and std
    W' = W – mean
    **for** i=1 to M **do**:
        u[i] = -1 +(i-1)*2/(M-1)
        binary_weight[i] = sign(W'+u[i]*std)
    **end for**
2. Preparation: define graph for alphas[] training
Inputs: W, binary_weights[]
    initialize alphas[] using random_normal()
    alpha_loss = || $W - \alpha B$||$^2$
    train alphas[] using AdamOptimizer
3. Core Function: ABC_conv  (to replace conv2d)
```

Inputs: input, shift parameters shift_para[], and multipliers beta[]:

```
    ABC_out = 0
    for n=1 to N do:
        shifted_inputs[n] = clip(input+ shift_para[n],0,1)
        binary_inputs[n] = sign(shifted_inputs[n]-0.5)
        compute result of approConv[n] (see 4.)
        ABC_out += beta[n]*approConv[n]
    end for
```

4. Function for approConv[n] calculation

Inputs: binary_inputs[n], binary_weights[], alphas[], bias, stride, padding

```
    for m=1 to M do:
        approConv[n] +=
            conv2d(binary_inputs[n], binary_weights[m], bias, stride, padding)*alphas[m]
    end for
```

5. Results

5.1. Project Results

We replicated the results mainly by exploring the ABC-Net with different combinations of hyperparameters, which are basically M, the number of binary filters that are used to approximate convolution layer, and N, the number of binary activations on the inputs. We ran 40 and 100 epochs to get the weights of the full-precision models of ResNet18 and ResNet20. To reduce the effects of other changeable variables, we fixed them when training ABC-Net. For instance, we trained all the ABC-Net for 20 epochs.

We have tested six configurations and the table below shows the summary of the results. Since we implemented ABC based on ResNet20 instead of ResNet18 while we believe that these two are similar in terms of model capacity, we group them together in the table.

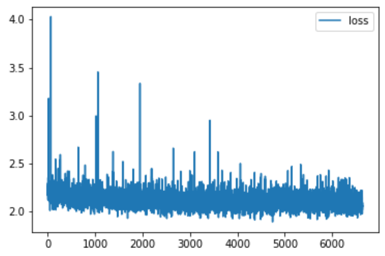
Network	M	N	Paper's Top1	Our Top1
res18/20	3	1	49.1%	27.3%
res18/20	3	3	61.0%	24.4%
res18/20	5	3	62.5%	22.8%
res18/20	5	5	65.0%	19.2%
res18/20	Full Precision		69.3%	65.7%
res34	3	3	66.7%	12.7%
res34	5	5	68.4%	15.7%
res34	Full Precision		73.3%	74.4%

5.2. Comparison of Results

We can see from the table that our results are quite different from that of the original paper. Our model with full precision achieved an accuracy at a similar level with what the authors present. But our ABC-Net performed poorly. The gap can be larger than 30%.

In addition, the author suggested that the more complex the ABC-Net is, which is to say the larger M and N are, the more accurate the ABC-Net is. However, we observed an almost reversed pattern. When we approximate the inputs and convolution weights with more binary objects, our model losses more accuracy.

Our group discussed the findings and came up with some explanations and thoughts. Firstly, although we get the accuracy for full-precision models close to what the authors suggest, we might not train the model completely and the weights we get for the initialization of ABC-Net may not be a stable one. This can be seen from the loss graph below which shows many peaks even at the end of the training period. ResNet is a relatively complex CNN model. More data accompanied with more training can yield a better model. While we did not have enough time, which may sacrifice the training and the quality of the weights we get.



Secondly, betas and shift parameters used in the binary activations are crucial to the ABC-Net. However, there are two issues regarding these two hyperparameters. The first one is that the authors did not provide their choice of initial values, which makes it more like a state-of-the-art. The second is that the clip operation is applied in the binary activation and there is no explicit solution to the gradient of clipping. The authors suggest that we adopt the Straight-Through Estimator (STE) to overcome this issue. While we relied on the numerical approximation by TensorFlow. This caused the betas and shift parameters barely trained. We checked our deduction by printing out the values of these parameters after each epoch and proved that the change is tiny (around 1e-3). This makes our model very sensitive to the initial value of betas and shift parameters. Since these parameters are used for the approximation of each convolution layer, the deeper the neural network is, the worse effect these parameters would have on training. This is the major reason for the underperformance of our ABC-Net.

5.3. Discussion of Insights Gained

Although we did not get our replicated results close to that in the paper, we gained much insights into the ABC-Net and would like to give our thoughts on a successful implementation of ABC-Net and further improvement.

Binarization is an important tool in forward propagation because it simplifies either the inputs or the neural network, which largely enhances the training speed. However, the negative side is that it loses much information as well, in both forward and backward propagation. Gradients are the core to update the neural network to get it close to the "real" network. We did not implement an obviously more accurate way to calculate gradients, which caused a large loss. Therefore, we believe that to apply STE or even better gradient approximation method would be a crucial step in binary neural network.

Secondly, we found that the binary neural network may perform differently on different dataset. At the beginning, we used MNIST as our dataset and tried two-layer CNN for testing purpose. The MNIST dataset is small enough for a two-layer CNN to be trained well. And when we implemented the ABC-Net, to our surprise, it also worked well. The accuracy is even much higher than what we get when we train ABC-based ResNet. We believe that for some small dataset, it is better to keep the model simple. Introducing too many parameters may incorporate much noise, especially in calculating gradients. In addition, contrary to what the authors suggest, we found empirically that the "bias" of the binary neural network may propagate, which suggest that a shallow model may outperform a deep model in some cases.

Thirdly, to train ABC-Net well, a good initialization and a longer training period may also be necessary. For instance, the use of betas and shift parameters in ABC-Net does make sense. The combination captures different patterns of the image and assigned different weights to them. At first, when we set betas to be one and shift parameters zero, which means we only extracted the image information in one way and gave a very large weight on it, we stuck at some point and the loss stopped to decrease. When we set them to be more disperse, we are more likely to get betas and shift parameters trained and get a higher accuracy by properly capturing information from the whole image.

Finally, we found that the forward propagation of a binary neural network is surely much faster than traditional CNNs. However, to train the binary model can take much longer time than to train common CNNs. This may result from the repeated steps to train alphas and the numerical difficulties in calculating gradients. It may also be a good idea to optimize the backpropagation in training binary neural network to improve the general usage of this powerful tool.

6. Conclusion

Provide summary of this project, briefly review the statements made in the abstract, in particular, if the enumerated objectives and goal are achieved. Emphasize and highlight the lessons learned, point out the direction for further improvement if needed.

7. References

[1] https://github.com/cu-zk-courses-org/e4040-2019Fall-Project-ZXCV-qh2199-xz2795-yh3067
[2] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. arXiv preprint arXiv:1602.02830, 2016.
[3] J. Deng,W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database.In Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on, pages 248–255. IEEE, 2009.
[4] X. Lin, C. Zhao, W. Pan. Towards Accurate Binary Convolutional Neural Network. arXiv:1711.11294, 2017.
[5] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In European Conference on Computer Vision, pages 525–542. Springer, 2016.
[6] Reference for TensorFlow Syntax https://github.com/layog/Accurate-Binary-Convolution-Network
[7] Keras reference, https://github.com/keras-team/keras

8. Appendix

8.1 Individual student contributions in fractions - table

	qh2199	xz2795	yh3067
Last Name	Hu	Zhang	Han
Fraction of (useful) total contribution	1/3	1/3	1/3
What I did 1	Build TensorFlow ABC layer functions	Build ABC layer in Keras	Build the ResNet model in Keras
What I did 2	Add the ABC layer to ResNet model	Build TensorFlow ResNet model	Refine the ResNet structure
What I did 3	Refine the report	Draft the report	Generate all graphs for the report