The 33rd Annual Conference of the Japanese Society for Artificial Intelligence, 2019

Final Sample Batch Normalization For Quantized Neural Networks

Joel Nicholls Atsunori Kanemura

LeapMind Inc.

We outline and conduct an empirical study into the effectiveness of a modified version of batch normalization, for combination with quantized neural networks. The proposed method uses only the statistics of the final batch for determining the batch normalization operation at the inference stage. This contrasts with the usual implementation, where population statistics are accumulated over many batches of training. The proposed and existing methods are compared over several models and datasets, which span both classification and object detection tasks. Overall, the proposed method exceeds the value and consistency of test performance compared to the usual batch normalization, in the case of quantized networks. For floating point precision networks, the usual method is best.

1. Introduction

Both batch normalization and quantization are widely used in the deep learning community for improving the training and inference speed of neural networks, respectively. However, it has been recently observed that the training of quantized neural networks can suffer from fluctuating performance, which is related to the way that batch normalization is implemented [8].

Batch normalization has for some time been an important technique in the deep learning toolkit. Primarily, batch normalization is used to accelerate the training of deep neural networks. Some argue that it eases training by reducing internal covariate shift [7], while others say that it smooths the loss landscape [13]. Either way, this technique has been empirically useful for deep learning practitioners to improve training.

Quantization is used for streamlining neural network models. Deep learning typically results in large models that are computationally expensive in the inference stage (e.g. to make predictions on new data). Quantized neural networks deal with this deficiency by greatly reducing the precision of the majority of parameters and computations involved, without significantly hurting the accuracy of the model.

To be explicit, here we are talking about very low precision, where the neural network must be trained in a quantization-aware way in order to give good performance. Weights and activations can be only a few bits [16], or even binary [3], to rapidly accelerate inference.

Prompted by the issue of fluctuating test performance of quantized neural networks and its relation to batch normalization, we have undertaken investigations on how to better combine these two techniques. In particular, we find that the quantized neural network has both higher test accuracy and is more consistent (has less jitter), when the statistics of only the most recent batch are used for determining the batch normalization used at inference.

This is in contrast to the standard method for batch normalization in floating point precision neural networks, which uses population statistics accumulated over many batches. We refer to our proposed method as *final sample batch normalization*.

Contact: Joel Nicholls, LeapMind Inc. Tokyo 150-0044, `joel@leapmind.io`

2. Method

For the neural network model and quantization functionality, we use the Blueoil framework [6]. Blueoil is based on TensorFlow [1] and provides convolutional neural networks in both floating point and quantized precision.

Within the Blueoil framework (and within Tensorflow also), there is an argument to the batch normalization operation called `decay`, which is a hyperparameter that controls how batch normalization statistics are accumulated.

The default batch normalization `decay` hyperparameter of Blueoil is 0.99. This value is close to 1, which means that population statistics are being used. Furthermore, both PyTorch [10] and Tensorflow [14] use population statistics by default. We will abbreviate this (standard) implementation of batch normalization as *BN-pop*. Throughout the experiments, BN-pop will refer to neural network models with `decay` of 0.99.

Our proposed modification to batch normalization can be easily implemented in Blueoil using a `decay` hyperparameter of 0.0. This uses the final batch statistics, so we will abbreviate the method as *BN-final*. Through empirical comparisons, we show that for quantized neural networks, our proposed BN-final performs equally well or better than BN-pop.

The quantized models of Blueoil use a combination of techniques from the literature on quantized networks [2, 11, 16]. For all experiments in this paper, the quantized neural networks have 2 bit activations and 1 bit weights, except for the first and last layers.

To evaluate the effectiveness of our proposed final sample batch normalization, we use classification and object detection models. There are more specific hyperparameter details in Appendix A. We compared BN-final and BN-pop over floating point precision and quantized models for 3 classification datasets (CIFAR-10, CIFAR-100 [9], and Caltech 101 [5]), and 1 object detection dataset (PASCAL VOC [4]). Making a range of comparisons is important to empirically establish the usefulness of our proposed change to the batch normalization method.

3. Results

The main thrust of the results is the empirical comparsion between the existing method BN-pop, and our proposed modification BN-final. Therefore, multiple training runs over various datasets were made and the test performance was measured. For classification, top-1 accuracy was used. For object detection, mean average

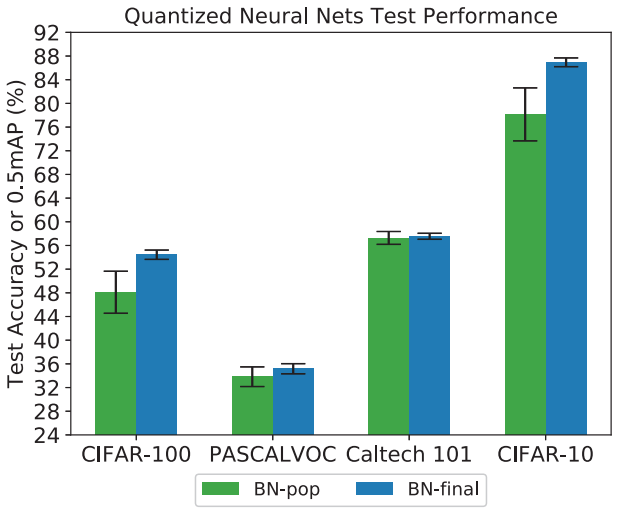

Figure 1: Test accuracy and 0.5mAP for existing batch normalization (BN-pop) and proposed batch normalization (BN-final) on quantized neural network models.

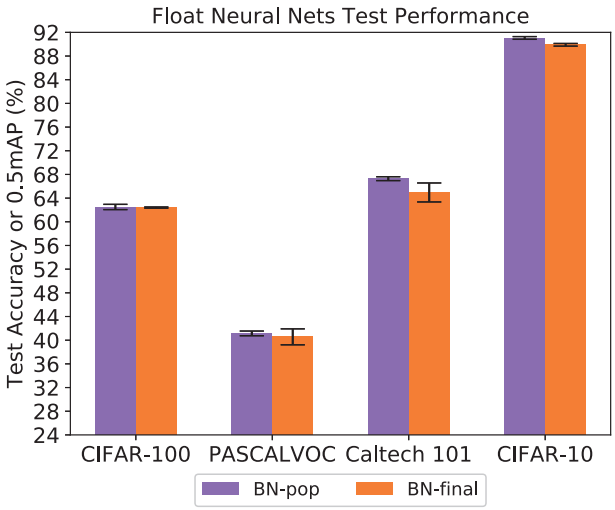

Figure 2: Test performance compared between existing BN-pop and proposed BN-final, for floating point precision neural networks.

precision (mAP) with 0.5 overlap threshold was used.

3.1 Main comparisons

The comparisons for quantized neural networks are given in Figure 1. In all cases, BN-final gave higher test performance than BN-pop. The CIFAR dataset shows the greatest difference.

The same comparisons are shown for floating point precision neural networks in Figure 2. For this set, BN-pop gives higher test performance. The order of preference of batch normalization method is reversed.

This suggests that the proposed BN-final should be used for quantized networks, but the usual BN-pop is better for floating point precision networks. Many of the investigations on batch normalization in the literature have focused on floating point precision networks. BN-final is not effective on floating point precision networks, which is a possible reason that it had not been previously explored.

3.2 Calculation of error bars

Here, we give more details about the values and error bars shown in Figure 1, Figure 2, and Figure 4. Two independent training runs were conducted for each experiment. However, two data points are not enough to get meaningful error bars.

Therefore, for each experiment, we took the last 3 test accuracies from the two training runs. These 6 data points were used to calculate the mean and standard deviation for the experiment.

The test steps are separated by only 1 000 train steps. Therefore, the last 3 test accuracies are not totally independent. But, they do give a rough idea of the fluctuations that can be expected. The error bars shown in the Figures are therefore generally an underestimate of the true standard deviation that would be obtained if all data points were completely uncorrelated.

Another point to note is that by taking the last 3 test steps, the model is still being trained during that time. There is a possibility that the performance is still increasing due to the training. However, by eyeballing the plots of TensorFlow training, the test accuracy has already levelled off at that late stage of training.

3.3 Test accuracy curve

Looking at the test accuracy of an example model over the training run, as in Figure 3, it can be seen that BN-final allows for much smoother generalization accuracy, consistently over the whole training process.

The accuracy on the train dataset was not shown because the two methods (BN-pop and BN-final), both act in the same way on the training dataset. It is worthwhile to note that the point of the final sample batch normalization method is to allow the trained model to transfer to an inference model in a better way than the existing batch normalization, for quantized networks.

In addition to the mean value of the test accuracy, the smoothness of test accuracy over the training of quantized neural networks is a positive point. It results in more consistent generalization accuracy, even if the deep learning practitioner decides to finish training early. The error bars shown in Figure 1 for various datasets also indicates more consistent test accuracy for BN-final, in comparison to BN-pop.

3.4 Additional quantizations

For classification on the CIFAR-100 dataset using quantized neural networks, two additional model versions were used. These other versions implement quantization in slightly different ways, giving a broader view on the compatibility between quantization and final sample batch normalization. They are shown in Figure 4.

Firstly, `usual_quant` makes use of channelwise quantization (this is the same as was shown for the CIFAR-100 category in Figure 1). Secondly, the network labelled `layerwise_quant` uses layerwise scaling factors. This can be useful to reduce the number of floating point precision computations in the neural network even further than channelwise quantization.

Thirdly, the network `divide255_quant` uses 8 bit inputs for the first convolutional layer. This differs with the usual kind of quantized neural network, which uses image standardization to provide floating point precision inputs to the first convolutional layer. The benefit of using `divide255_quant` is that the required precision of the first layer convolution is lowered, resulting in more efficient inference.

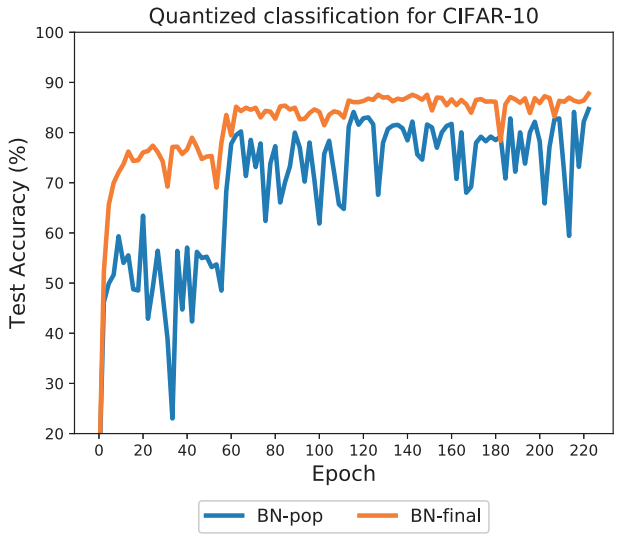

Figure 3: Time series plot for the test accuracy of the quantized neural network against epochs of training completed. Both BN-pop and BN-final are compared, for classification on the CIFAR-10 dataset. In this experiment, the batch size was 100, with 45 000 training images. The test accuracy was calculated once every 1 000 training steps, corresponding to once every $2.\overline{2}$ epochs, where the overline indicates a repeating decimal.

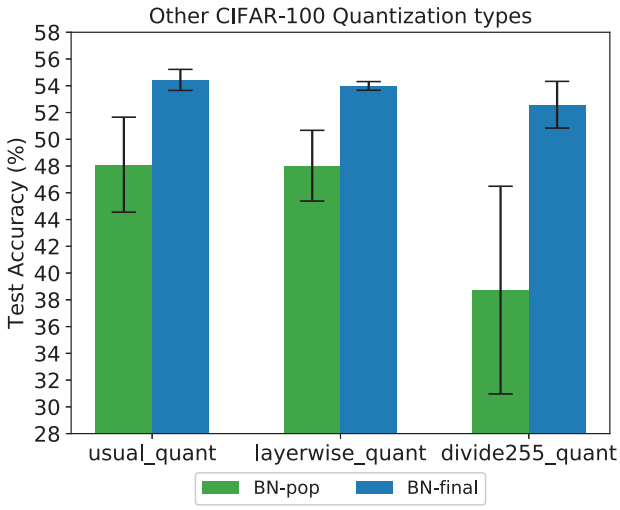

Figure 4: Test accuracy compared between BN-pop and BN-final, for different methods of quantization. In each case, the network performs classification on the CIFAR-100 dataset.

The main point to note from Figure 4 is that the `divide255_quant` network does a lot worse than the other two networks, when using BN-pop. Instead, if using BN-final, all three networks have similar performance.

This suggests that `divide255_quant` is especially sensitive to the change between training model and inference model. One possible reason for why BN-final improves this network so significantly in the inference is that it allows for more stability with respect to the inputs. The results here are not enough to fully support this kind of hypothesis, which is an area for further work.

4. Related works

Several other works have tampered with the standard batch normalization. For example, instance normalization [15] uses the batch normalization behaviour of training for the inference stage, averaging over only the spatial dimensions of the batch in order to improve on other works. Their batch normalization at inference is not an affine transformation, so they have increased the computational cost of inference to improve style transfer. Our motivation and implementation are quite different, since we are interested in increasing the efficiency of deep learning.

Some works also consider batch normalization in the context of quantized neural networks. Courbariaux et al. [3] implement an approximate shift-based batch normalization to decrease the computational effort by using bit shifting. This technique saves computation in the training, and seems to be compatible with final sample batch normalization. We have not tested this combination because our main interest is in improving speed and accuracy of inference. In the inference stage, the affine transformation of batch normalization can be folded into scaling factors and layer bias, resulting in only a small computation cost relative to the convolutional layers.

Finally, Krishnamoorthi [8] implements a modification called batch normalization freezing. After a certain number of training steps, the updates of batch normalization statistics are frozen and the batch normalization behaviour during training switches to that of the inference stage. In this sense, the batch normalization becomes an affine transformation after the freeze step (which is a hyperparameter). This method reduces jitter by a large amount. However, the batch normalization is effectively switched off (frozen), at the later steps of training. It is hard to make a direct comparison with their work without more experiments because their whitepaper shows the effect of batch normalization on 8 bit weights and activations, whereas our experiments involve 1 bit weights and 2 bit activations.

5. Conclusion

For float point precision networks, there is no benefit in changing to BN-final. If anything, BN-pop is better. However, for quantized networks, BN-final gives higher and more consistent test accuracy. Moreover, using BN-final does not add any computational overhead compared to the usual method of batch normalization. This makes BN-final an excellent candidate for reducing the gap in performance between quantized and floating point precision neural networks.

It seems that both the CIFAR dataset and the `divide255_quant` network benefit especially well from BN-final. We hypothesise that this dataset and network have something in common that causes them to benefit most from BN-final: in both these cases the forward function of the neural network model is a sensitive (easily perturbed) function of its inputs. The CIFAR dataset provides very low resolution inputs. The `divide255_quant` network enables higher efficiency, but has a more sensitive first layer.

There is a difference between the forward functions in training and inference due to the change in batch normalization behaviour to an affine transformation. This increases the efficiency for inference, but it generally causes a decrease in test accuracy. Therefore, one possible reason for the success of BN-final is that it reduces the

difference between the forward function in training and inference. This gives the most benefit in cases (such as those mentioned in the previous paragraph), where the forward function is a sensitive function on its inputs. Here is an exciting direction for possible future research.

A major test case that we have missed out is comparison for the ImageNet [12] dataset. It is an important case because it is often used as pretraining for other datasets, and is used in the deep learning literature as a difficult and realistic problem. However, there was not sufficient time to gather results on this dataset for the current paper.

Another point to note about our proposed method is the case where some batches contain outlier data, are imbalanced, or especially small. In the case of a dataset where some batches contain poor quality outlier data, it is possible that the final batch is not representative of the overall dataset. This could result in low test accuracy.

One possible partial remedy is to save the model at multiple steps near the end of training, and choose the best model based on validation accuracy. When the batch size is imbalanced or small, the single batch statistics will naturally be less consistent between batches. This may exacerbate the issue of imbalanced datasets.

A Extra detail on experimental setup

Here we give further explanation on hyperparameters, as well as the method for constructing the train and test datasets used in the experiments.

CIFAR-10: 45 000 images were used for the train dataset and 10 000 images for the test dataset. The image size was 32×32 pixels. We used 100 000 train steps, `divide255_quant`, and channelwise scaling factors. The network in Blueoil is `LmnetV1Quantize`. For the floating point precision version, network `LmnetV1` and scaling `PerImageStandardization` were used.

CIFAR-100: 45 000 images were used for the train dataset and 10 000 images for the test dataset. The image size was 32 × 32 pixels. We used 100 000 train steps. We used `PerImageStandardization` and channelwise scaling except where stated. For the quantized version, we used the network `LmnetV1Quantize`, and for floating point precision version `LmnetV1`.

Caltech 101: 7 809 images were used for the train dataset and 868 images for the test dataset. For Caltech 101 experiments, we trained without augmentation. We used 10 000 steps, `PerImageStandardization`, `Image_Size` = 128, and `batch_size` of 64.

PASCALVOC: The test dataset consists of 4 952 images from PASCAL VOC 2007. The train dataset consists of 16 551 images from a mix of the remaining images of PASCAL VOC 2007 and 2012. We resize the images to 320 × 320 pixels. We used `divide255_quant`, channelwise scaling, and 100 000 train steps. In Blueoil, the network is called `LMFYoloQuantize`. The config file is `lm_fyolo_quantize_pascalvoc_2007_2012.py`.

Acknowledgements

We thank Hiroyuki Tokunaga and Takuya Wakisaka from LeapMind Inc. for their useful advice on this paper.

References

[1] Martín Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from `https://www.tensorflow.org/`.

[2] Zhaowei Cai, Xiaodong He, Jian Sun, and Nuno Vasconcelos. Deep learning with low precision by half-wave Gaussian quantization. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5406–5414, July 2017.

[3] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or −1. *arXiv:1602.02830 [cs.LG]*, 2016.

[4] Mark Everingham, Luc Van Gool, Christopher K. I. Williams, John Winn, and Andrew Zisserman. The Pascal Visual Object Classes (VOC) challenge. *International Journal of Computer Vision*, 88(2):303–338, September 2009.

[5] Li Fei-Fei, Rob Fergus, and Pietro Perona. Learning generative visual models from few training examples: An incremental Bayesian approach tested on 101 object categories. In *CVPR Workshop on Generative Model Based Vision*, 2004.

[6] LeapMind Inc.˙ Blueoil. `https://github.com/blue-oil/blueoil`, 2018. [Online; accessed 5-Feb-2019].

[7] Sergey Ioffe and Christian Szegedy. Batch normalization accelerating deep network training by reducing internal covariate shift. In *International Conference on International Conference on Machine Learning (ICML)*, 2015.

[8] Raghuraman Krishnamoorthi. Quantizing deep convolutional networks for efficient inference: A whitepaper. *arXiv:1806.08342 [cs.LG]*, June 2018.

[9] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

[10] PyTorch. Normalization layers, torch.nn, pytorch documentation. `https://pytorch.org/docs/stable/nn.html#normalization-layers`. [Online; accessed 5-Feb-2019].

[11] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. XNOR-Net: ImageNet classification using binary convolutional neural networks. In *European Conference on Computer Vision (ECCV)*, pages 525–542, 2016.

[12] Olga Russakovsky et al. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[13] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? *arXiv:1805.11604 [stat.ML]*, May 2018.

[14] TensorFlow. tf.contrib.layers.batch_norm. `https://www.tensorflow.org/api_docs/python/tf/contrib/layers/batch_norm`. [Online; accessed 5-Feb-2019].

[15] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance Normalization: The Missing Ingredient for Fast Stylization. *arXiv:1607.08022 [cs.CV]*, July 2016.

[16] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv:1606.06160 [cs.NE]*, June 2016.