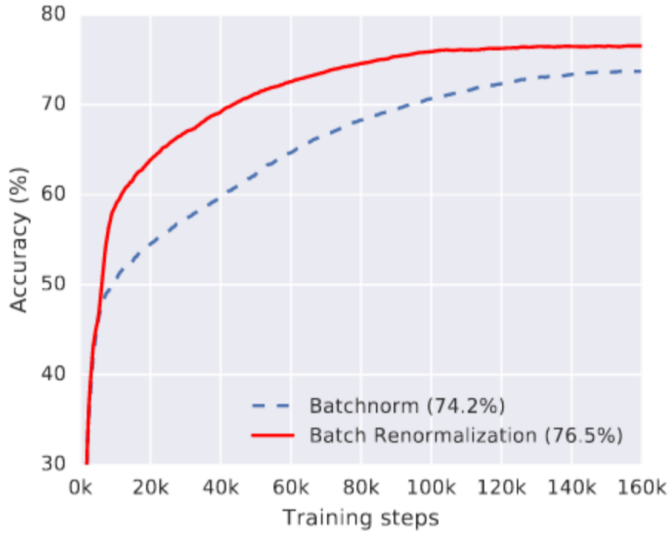
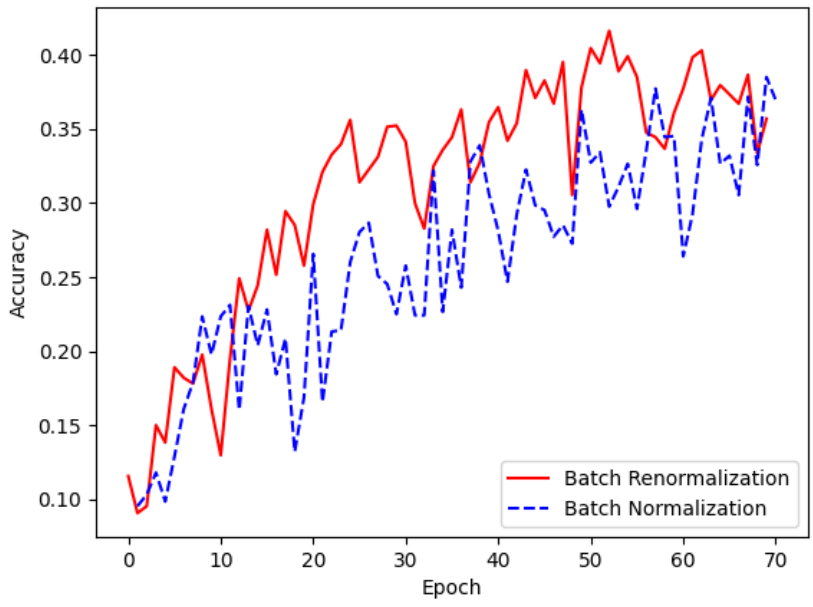
Steven Cho, Simon Yoon
Midterm Project Writeup

ECE472- Deep Learning
Professor Curro

Batch Renormalization: Towards Reducing Minibatch Dependence in Batch-Normalized Models
Author: Sergey Ioffe (arxiv.org/abs/1702.03275)
The goal of this project was to reproduce Figure 2 from the paper:



The figure below is our reproduction:



This figure displays the key behavior in that batch renormalization performs better than batch normalization earlier in training, but it converges later in training.

We implemented a custom Batch Renormalization layer CustomBatchNorm() and the plotted figures below were produced using Tensorflow's BatchNormalization layer with default configuration. Our key pieces in replicating the results of Sergey Ioffe's renormalization were to implement his algorithm by introducing additional arguments to regular BN particularly renorm_clipping to add the epoch-wise rmax, dmax customization and changing the normalization scheme from batch-wise independent scaling to one inclusive of prior normalization parameters. Rmax was set as a constant until 20 epochs, and gradually increased to rmax = 3. Likewise, for dmax, dmax was set as a constant for the first 20 epochs, and gradually increased to dmax = 5. The relaxation begins at 50 epochs. The paper ran classification on ImageNet with InceptionV3 for upwards of 160k training steps. This scale of compute was not feasible, so our variant was CIFAR-10 classification with a CNN for 100 epochs, which equates to 2.5k training steps (25 steps/gradient updates per epoch). And as a result, we scaled the parameters for rmax, dmax relaxation proportionately. The paper does not explicitly state their relaxation scheme, so ours increases linearly after a proportional threshold epoch.

Model

The model we used consisted of three convolutional blocks. Each convolutional block consisted of two convolution layers, two normalization layers, and a dropout layer with increased dropout rate deeper into the model. Each convolution layer was followed by a normalization layer, and the last layer of each block was the dropout layer. The model we used did not perform well on image classification on CIFAR-10, but we are only interested in the behavior of the validation accuracy curves of Batch Normalization and Batch Renormalization. Our original model was larger however, the relevant calculations lead to OOM issues. For simplicity and resource efficiency, we did not change this model.
CustomBatchNorm Layer
The custom BatchNorm layer updates the moving averages mu, sigma using the equation given in the paper.

$$\mu := \mu + \alpha(\mu_{\mathcal{B}} - \mu) \qquad \text{// Update moving averages}$$
$$\sigma := \sigma + \alpha(\sigma_{\mathcal{B}} - \sigma)$$

Like the paper, $\alpha = 0.01$. The dimensions of the inputs are (B, H, W, C), which are reshaped to (N, M, H, W, C), where B = # of samples in batch, N = # of microbatches, M = microbatch size. The moments were computed across N & C, with fixed M, H, W. In other words, the dimensions are [N, 1, 1, 1, C], and the moving average preserves the mean value across channels. The rmax, rmin, dmax values are fed in through renorm_clipping explained above.

Final Result

The final plot compares the validation accuracy of Batch Normalization and Batch Renormalization over epochs. This is slightly different from the paper, which plots validation accuracy over training steps. Batch Renormalization achieves higher validation accuracy initially, but the two plots converge past 70 epochs.

Relevant Work/Attempts

Some alterations to the project were informed by their unique difficulties. We originally intended to utilize InceptionV3 on CIFAR rather than ImageNet as the dataset was too large. However, InceptionV3 training had issues beyond training time increases. The model requires a minimum input_shape [75,75,3], which was resolved by up-sampling CIFAR from [32,32,3] and taking into consideration pixel intensity. And to train on a reasonable scale, we utilized Cooper's Kahan, but had issues allocating large tensors perhaps as GPUs ran out of VRAM. We attempted further alterations to make the project work on Kahan without compromising too much of the experiment's integrity while simultaneously trying to maintain the graph's quality to little success. We could not find a happy medium. We adjusted for a simpler CNN. In prior attempts, we wrote helper functions to replace layers for nonsequential models as manual replacement for Inception was not feasible. Implementing this functionality took up a large bulk of our time, but with more complicated issues coming up, we were advised to implement a layer and utilize a simpler model instead for the time being. Another aspect to replicate the paper included the plotting of accuracy over training steps rather than epochs as stated prior. This required a callback to output the validation accuracy at every training step rather than epoch. The evaluation of the validation accuracy at each training step was slower than the batch output, so the epoch would prematurely finish. This would mess up the data collection and attempts to suppress this behavior were unsuccessful. As a result, our graph is by enough epochs to represent the fast-rising renorm behavior. We used our discretion in removing certain graph improvementπ attempts such as changing batch-size, regularization, model complexity, learning rate, and even computing a moving average across multiple runs as it strayed disingenuously far from the experiment's procedure for the sake of mild smoothening and behavior clarity.