TRAINING WIDE RESIDUAL NETWORKS FOR DEPLOYMENT USING A SINGLE BIT FOR EACH WEIGHT

Anonymous authors
Paper under double-blind review

ABSTRACT

For fast and energy-efficient deployment of trained deep neural networks on resource-constrained embedded hardware, each learnt weight parameter should ideally be represented and stored using a single bit. Error-rates usually increase when this requirement is imposed. Here, we report methodological innovations that result in large reductions in error rates for deep convolutional neural networks deployed using a single bit for each weight. For 1-bit weight models with just 20 weight layers, requiring only $\sim$ 4 MB of parameter memory, for CIFAR-10, CIFAR-100, SVHN and Imagenet32 (Top-5) we achieve error rates of 3.74%, 18.41%, 1.93%, and 21.89% respectively. These error rates are about half those of previous reported error rates for CIFAR-10/100, and are within 1–3% of our error-rates for the same network with full-precision weights. Using a warm-restart learning-rate schedule, we found training for single-bit weights just as fast as full-precision networks, and achieved about 98%-99% of peak performance in just 62 training epochs for CIFAR 10/100.

1 INTRODUCTION

Fast parallel computing resources, namely GPUs, have been integral to the resurgence of deep neural networks, and their ascendancy to becoming state-of-the-art methodologies for many computer vision tasks. However, GPUs are expensive in terms of their energy requirements. They typically compute using single-precision floating point (32 bits), which has now been recognised as providing far more precision than needed for deep neural networks. Moreover, training and deployment can require the availability of large amounts of memory, both for storage of trained models, and for operational RAM. If deep-learning methods are to become embedded in resource-constrained sensors, devices and intelligent systems, ranging from robotics to the internet-of-things to self-driving cars, reliance on high-end computing resources will need to be reduced.

To this end, there has been increasing interest in finding methods to drive down the resource burden of modern deep neural networks. Existing methods typically exhibit good performance but for the ideal case of single-bit parameters and/or processing, still fall well-short of state-of-the-art error rates on known benchmarks.

In this paper, we detail methods that resulted in a significant reduction in the gap between Convolutional Neural Networks (CNNs) deployed using weights stored and applied using standard precision (32-bit floating point) and networks deployed using weights represented by a single-bit each[1]. In the process of developing our methods, we also obtained a small improvement in error-rates obtained by standard full-precision versions of the CNNs we used.

1.1 RELATED WORK

1.1.1 RESNETS

In 2015, a new form of CNN called a "deep residual network," or "ResNet" (He et al., 2015b) was developed and used to set many new accuracy records on computer-vision benchmarks. In

[1]After completion of double-blind review, if this paper is accepted, we will provide full training code and trained models on GitHub. An example trained model and matlab code for testing it is available at `https://github.com/SingleBitResNets/SingleBitResNets`

comparison with older CNNs such as Alexnet (Krizhevsky et al., 2012) and VGGnet (Simonyan & Zisserman, 2014), ResNets achieve higher accuracy with far fewer learnt parameters and FLOPs (FLoating-point OPerations) per image processed. The key to reducing the number of parameters in ResNets was to replace "all-to-all layers" in VGG-like nets with "global-average-pooling" layers that have no learnt parameters (Lin et al., 2013; Springenberg et al., 2014), while simultaneously training a much deeper network than previously. The key new idea that enabled a deeper network to be trained effectively was the introduction of so-called "skip-connections" (He et al., 2015b). Many variations of ResNets have since been proposed. ResNets offer the virtue of simplicity, and given the motivation for deployment in custom hardware, we have chosen them as our primary focus.

Despite the increased efficiency in parameter usage, as with other CNNs, accuracy in ResNets still tends to increase with the total number of parameters; unlike other CNNs, increased accuracy can result either from deeper (He et al., 2016) or wider networks (Zagoruyko & Komodakis, 2016).

In this paper, we use Wide Residual Networks (Zagoruyko & Komodakis, 2016), as they have been demonstrated to produce better accuracy in less training time than deeper networks.

1.1.2 Reducing the memory burden of trained neural networks

Achieving the best accuracy and speed possible when deploying ResNets or similar networks on hardware-constrained mobile devices will require minimising the total number of bits transferred between memory and processors for a given number of parameters. Motivated by such considerations, a lot of recent attention has been directed towards compressing the learnt parameters (*model compression* and quantizing other representations in trained neural networks—see Hubara et al. (2016) for a more detailed literature review.

Recently published strategies for *model compression* include reducing the precision (number of bits used for numerical representation) of weights in deployed networks by doing the same during training (Courbariaux et al., 2015; Hubara et al., 2016; Merolla et al., 2016; Rastegari et al., 2016), reducing the number of weights in trained neural networks by pruning (Iandola et al., 2016), quantizing weights following training (Zhou et al., 2017), reducing the precision of computations performed in forward-propagation during inference (Courbariaux et al., 2015; Hubara et al., 2016; Merolla et al., 2016; Rastegari et al., 2016), and modifying neural network architectures (Howard et al., 2017). A theoretical analysis of various methods proved results on the convergence of a variety of weight-binarization methods (Li et al., 2017).

From this range of strategies, we are focused on approaches that enable deployment of trained models immediately following training, for the simplicity this enables.

1.2 Overall approach and summary of contributions

Our strategy for improving methods that enable inference with single-bit weights was threefold:

1. **State-of-the-art baseline.** We sought to begin with a baseline full-precision deep CNN variant with close to state-of-the-art error rates. At the time of commencement in 2016, the state-of-the-art on CIFAR-10 and CIFAR-100 was held by Wide Residual Networks (Zagoruyko & Komodakis, 2016), so this was our starting point. While subsequent approaches have exceeded slightly the error rates of wide ResNets (see Table 3), for hardware ResNets offer superior simplicity, which conforms with our third strategy in this list.
2. **Make minimal changes when training for 1-bit weights.** We aimed to ensure that training for 1-bit weights could be achieved with minimal changes in comparison to training our baseline.
3. **Simplicity is desirable in custom hardware.** In addition to training to enable single-bit weights at inference time, with custom hardware implementations in mind we sought to simplify the design of the baseline network as much as possible without sacrificing accuracy.

1.2.1 Contributions to Full-Precision Wide ResNets

Although this paper is chiefly about single-bit weights, we exceeded our objectives for the full-precision baseline network, and surpassed reported error rates reported for CIFAR-10 and CIFAR-

100 using full-precision Wide ResNets (Zagoruyko & Komodakis, 2016), while not requiring the use of dropout regularization (see Table 3). This was achieved using just 20 convolutional layers; most prior work has demonstrated best wide ResNet performance using 28 convolutional layers.

All but one of our modifications to standard Wide ResNets are mainly to achieve simplicity, as in the second and third prongs of our strategy. Collectively these minor modifications (see the Appendix) provide error-rate reductions of $\sim$1% on CIFAR-10/100 compared to not using them.

However, one innovation we introduce achieves a significant error-rate advantage for CIFAR-10 and CIFAR-100 in Wide ResNets. This is to simply not learn the per-channel scale and offset factors in the batch-normalization layers, while retaining the remaining attributes of these layers. This is done in conjunction with exchanging the ordering of the final weight layer and the global average pooling layer. We present a case for why these changes should be advantageous, in the Discussion section.

We observed this effect to be more pronounced for CIFAR-100 than CIFAR-10, gaining over 1% in test-error rate. But the method is advantageous only when training tended to cause overfitting; when overfitting is not an issue, such as for Imagenet, removing learning of batch-norm parameters is only detrimental.

1.2.2 Contributions to deep CNNs with single-bit weights for inference

Our approach surpasses by a large margin all previously reported error rates for CIFAR-10/100 (error rates halved) and SVHN for networks constrained to run with single-bit weights at inference time. We found no previous results for single-bit weights for Imagenet32 (Chrabaszcz et al., 2017), but our results on this dataset are also indicative of the general applicability of our methods (see Section 5.1).

One reason we have achieved lower error rates for the 1-bit case than previously is to start with a superior baseline network than in previous studies, namely wide ResNets. However, as we show in Section 5.1, our approach results in smaller error rate increases relative to full-precision error rates than previously. Moreover, previous modifications of training to use 1-bit weights has tended to result in slow convergence requiring training for many 100s of epochs.

Our contribution is to introduce a simple fixed scaling method for each convolutional layer, that permits activations and gradients to flow through the network with minimum change in standard deviation, in accordance with the principle underlying popular initialization methods (He et al., 2015a). We combine this with the use of a warm-restart learning-rate method (Loshchilov & Hutter, 2016) that enables us to report close-to-baseline results for the 1-bit case in far fewer epochs of training than reported previously.

2 Learning a model with convolution weights ± 1

2.1 The sign of weights propagate, but full-precision weights are updated

We follow the approach of Courbariaux et al. (2015); Rastegari et al. (2016); Merolla et al. (2016), in that we find good results when using single-bit weights at inference time if during training we apply the sign function to real-valued weights for the purpose of forward and backward propagation, but update full-precision weights using SGD with gradients calculated using full-precision.

However, previously reported methods for training using the sign of weights either need to train for many hundreds of epochs (Courbariaux et al., 2015; Merolla et al., 2016), or use computationally-costly normalization scaling for each channel in each layer that changes for each minibatch during training (Rastegari et al., 2016). We find it is possible to dramatically accelerate convergence and execution time, while reducing error rates, using a simple alternative approach, as we now describe.

2.1.1 We scale the output of conv layers using a constant for each layer

We begin by noting that the standard deviation of the sign of the weights in a convolutional layer with kernels of size $F \times F$ will be close to 1, assuming a mean of zero. In contrast, the standard deviation of layer i in full-precision networks is initialised in the method of He et al. (2015a) to

$\sqrt{2/(F^2 C_{i-1})}$, where C_{i-1} is the number of input channels to convolutional layer i, where $i = 1, .., L$, L is the number of convolutional layers and $C_0 = 3$ for RGB inputs.

Our strategy for a network trained using the sign of weights is to seek to make the propagation of weights and gradients statistically equivalent to those in the full precision network, so as to ensure the standard-deviation as information flows through the network is close to the ideal situation that motivated the method of He et al. (2015a).

When applying the sign function alone, there is a mismatch with the principled approach to controlling gradient and activation scaling through a deep network He et al. (2015a). Although the use of batch-normalization can still enable learning, convergence is empirically slow and less effective.

To address this problem, for training using the sign of weights, we use the initialisation method of He et al. (2015a) for the full-precision weights that are updated, but also introduce a scaling for the single-bit weights of each convolutional layer. This scaling has a constant un-learnt value equal to the initial standard deviation of $\sqrt{2/(F^2 C_{i-1})}$ of the method of He et al. (2015a). This enables the standard deviation of forward-propagating information to be equal to the value it would have initially in full-precision networks.

In implementation, during training we multiply the sign of the weights in each layer by this value. For inference, we do this multiplication using a scaling layer following the weight layer, so that all weights in the network are stored using 0 and 1, and deployed using ± 1 (see, e.g. `https://github.com/SingleBitResNets/SingleBitResNets`). Hence, custom hardware implementations would be able to implement convolutions in our trained models without multipliers.

The fact that we scale the weights explicitly during training is important. Although for forward and backward propagation it is equivalent to scale the input or output feature maps of a convolutional layer, doing so also scales the calculated gradients with respect to filter weights, since these are calculated by convolving input and output feature maps. As a consequence, learning is dramatically slower unless an additional adjustment is made to introduce layer-dependent learning rates to cancel out the scaling,

Our approach is similar to the BWN method of Rastegari et al. (2016), but our constant scaling method differs in execution complexity, for two reasons. We also found we achieved better results. These aspects are discussed in Section 5.2.

In summary, the only differences we make in comparison with full-precision training are as follows (see also Figure 1). Let $\mathbf{W}_i$ be the tensor for the convolutional weights in the i–th convolutional layer. Then these weights are processed in the following way only for forward propagation and backward propagation, not for weight updates:

$$\hat{\mathbf{W}}_i = \sqrt{\frac{2}{F_i^2 C_{i-1}}} \operatorname{sign}(\mathbf{W}_i), \quad i = 1, \ldots, L, \tag{1}$$

where F_i is the spatial size of the convolutional kernel in layer i.

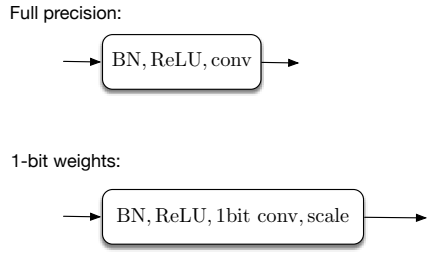

Figure 1: **Difference between our full-precision and 1-bit weights networks.** The "1-bit conv" and "scale" layers are equivalent to the operations shown in Eqn. (1).

3 METHODS COMMON TO BASELINE AND SINGLE-BIT NETWORKS

3.1 NETWORK ARCHITECTURE

Our ResNets use the 'post-activation' and identity mapping approach of He et al. (2016) for residual connections. Each residual block includes two convolutional layers, each preceeded by batch normalization (BN) and Rectified Linear Unit (ReLU) layers. Rather than train very deep ResNets, we use wide residual networks (wide ResNets) (Zagoruyko & Komodakis, 2016) with exactly 20 convolutional layers. Although Zagoruyko & Komodakis (2016) and others reported slightly better results on the CIFAR-10 dataset with 28 layers, after we made our other modifications we were able to match the performance of a 28 layer network with just 20 layers, and no additional width.

Our baseline ResNet design (see Figures 1 and 2) has several differences in comparison to those of He et al. (2016); Zagoruyko & Komodakis (2016). These details are articulated in the Appendix, and in combination enable us to obtain slightly improved error rates to comparison Wide ResNets.

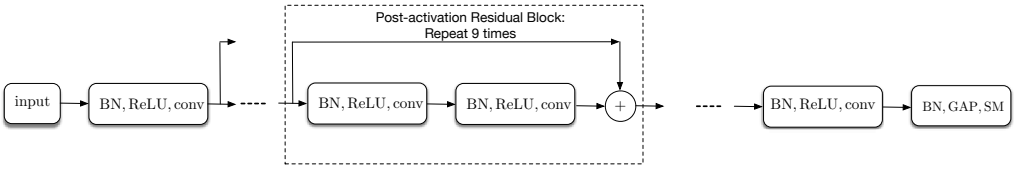

Figure 2: **Our 20-convolutional-layer ResNet architecture.** The design is mostly a standard pre-activation ResNet (He et al., 2016; Zagoruyko & Komodakis, 2016). The first (stand-alone) convolutional layer ("conv") and first two residual blocks have $16k$ output channels. The next three blocks have $32k$ output channels and the final four blocks have $64k$ output channels, where k is the widening parameter (we use $k = 4$ and $k = 10$ in this paper). The final (stand-alone) convolutional layer is a 1×1 convolutional layer that gives N output channels, where N is the number of classes. Blocks 3 and 6 are downsampling blocks (details are depicted in Figure 3) that reduce each spatial dimension in the feature map by a factor of 2, and simultaneously double the number of channels. "BN" denotes a batch-normalization layer, "ReLU" denotes a rectified-linear-unit layer, "GAP" denotes a global-average-pooling layer, and "SM" denotes a softmax layer. Our approach also works for deeper networks.

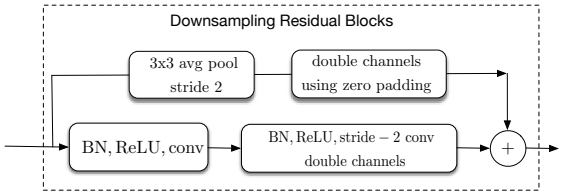

Figure 3: **The downsampling blocks in our 20-convolutional-layer ResNet architecture.** As in a standard pre-activation ResNet (He et al., 2016; Zagoruyko & Komodakis, 2016), downsampling (stride-2 convolution) is used in the convolutional layers where the number of output channels increases, i.e. here the second convolutional layer within the third and sixth residual blocks. The corresponding downsampling for skip connections is done in the same residual residual block, so that the residual summation for such blocks immediately follows downsampling and channel increase operations. Unlike standard pre-activation ResNets we use an average pooling layer ("Avg pool") when downsampling.

3.2 TRAINING

We trained our models following, for most aspects, the standard stochastic gradient descent methods used by Zagoruyko & Komodakis (2016) for wide ResNets. Specifically, we use cross-entropy loss, minibatches of size 125, momentum of 0.9, and a weight decay of 0.0005. Apart from one set of experiments where we added a simple extra approach called cutout (detailed below), we use standard 'light' data augmentation, including randomly flipping each image horizontally with probability 0.5 for CIFAR 10/100 and Imagenet32. For the same 3 datasets plus SVHN, we pad by 4 pixels on all sides (using random values between 0 and 255) and crop a 32×32 patch from a random location in the resulting 40×40 image. We did not use any image-preprocessing, i.e. we did not subtract the mean, as the initial BN layer in the network learns an offset, and we did not use whitening or augment using color or brightness. We use the initialization method of He et al. (2015a).

We now describe differences in our training approach compared to those usually reported.

3.2.1 BATCH-NORMALIZATION SCALES AND OFFSETS ARE NOT LEARNT FOR CIFAR-10/100

For reasons explained in Section 5.3, when training on CIFAR-10/100 and SVHN, in all batch-norm layers except the first one at the input, we do not learn the scale and offset factor, instead initializating these to 1 and 0 in all channels, and keeping those values through training. Note that we also do not learn any biases for convolutional layers.

The usual approach to setting the moments for use in batch-norm layers in inference mode is to keep a running average through training. When not learning batch-normalization parameters, we found a small benefit in calculating the batch-normalisation moments used in inference only after training had finished. We simply form as many minibatches as possible from the full training-set, each with the same data augmentation as used during training applied, and pass these through the trained network, averaging the returned moments for each batch.

3.2.2 WE USE A WARM-RESTART LEARNING-RATE SCHEDULE

We use a 'warm restarts' learning rate schedule that has reported state-of-the-art wide residual network results (Loshchilov & Hutter, 2016) whilst also speeding up convergence. The method constantly reduces the learning rate from 0.1 to 1×10^{-4} according to a cosine decay, across a certain number of epochs, and then repeats across twice as many epochs. We restricted our attention to a maximum of 254 epochs (often just 62 epochs, and no more than 30 for Imagenet32) using this method, which is the total number of epochs after reducing the learning rate from maximum to minimum through 2 epochs, then 4, 8, 16, 32 and 64 and 128 epochs. For CIFAR-10/100, we typically found that we could achieve test error rates after 32 epochs within 1-2% of the error rates achievable after 126 or 254 epochs.

3.2.3 EXPERIMENTS WITH CUTOUT DATA AUGMENTATION FOR CIFAR 10/100

In the literature, most experiments with CIFAR-10 and CIFAR-100 use simple "standard" data augmentation, consisting of randomly flipping each training image left-right with probability 0.5, and padding each image on all sides by 4 pixels, and then cropping a 32×32 version of the image from a random location. We use this augmentation, although with the minor modification that we pad with uniform random integers between 0 and 255, rather than zero-padding.

Additionally, we experimented with "cutout" augmentation (Devries & Taylor, 2017) applied. This involves randomly selecting a patch of each raw training image to remove. The method was shown to combine with other state-of-the-art methods to set the latest state-of-the-art results on CIFAR-10/100 (see Table 1). Following Devries & Taylor (2017), for CIFAR-10 we choose patches of size 16×16 and for CIFAR-100 we choose patches of size 8×8, while if the chosen pixel is near the image border, the patch impacts on the image only for the part of the patch inside the image. As with padding, we use uniform random integers to replace the image pixel values in the location of the patches. We did not apply cutout to other datasets.

4 RESULTS

We conducted experiments with four databases consisting of 32×32 RGB images: CIFAR-10, CIFAR-100, SVHN and Imagenet32, as well as MNIST. Details of the first three datasets can be found in many papers, e.g. Zagoruyko & Komodakis (2016). Imagenet32 is a downsampled version of Imagenet, where the training and validation images are cropped using their annotated bounding boxes, and then downsampled to 32×32 (Chrabaszcz et al., 2017); see `http://image-net.org/download-images`. All experiments were carried out on either a single Titan X or Xp GPU using matlab with GPU acceleration from matconvnet and CuDNN. Using the Titan Xp, we were able to train our 27.4 million parameters 20-10 ResNet on Imagenet32 in just 35 hours total for 30 epochs.

We mostly report results for networks with between 25 and 40 million parameters, given state-of-the-art networks for these four datasets fall into this bracket. To this end, we mainly use wide ResNets that are $4\times$ and $10\times$ wider than baseline ResNets, to use the terminology of Zagoruyko & Komodakis (2016), where the baseline has 16 channels in the layers at the first spatial scale. We use notation of the form 20-10 to denote wide ResNets with 20 convolutional layers and width $4\times$.

Table 1 lists our top-1 error rates for CIFAR 10/100; C10 indicates CIFAR-10, C100 indicates CIFAR100. In Table 1, the superscript $^+$ indicates standard crop and flip augmentation, whereas $^{++}$ indicates additional cutout augmentation. Table 2 lists top-1 and top-5 error rates for SVHN and Imagenet32 (I32), as for both of these we trained for only 30 epochs.

Table 1: Test-set error-rates for our approach applied to CIFAR-10 and CIFAR-100.

Weights Precision	**ResNet**	**Epochs**	**Params**		**C10$^+$**	**C100$^+$**		**C10^{++}**	**C100^{++}**
32-bits	20-4	126	4.3M		5.25	22.03		4.54	21.04
32-bits	20-10	254	26.8M		4.32	18.67		3.51	17.81
32-bits	20-20	126	107.0M		-	17.74		-	-
1-bit	20-4	126	4.3M		5.59	24.06		5.46	23.60
1-bit	20-10	254	26.8M		4.66	19.84		3.74	18.41
1-bit	20-20	126	107.0M		-	19.02		-	-

Table 2: Top-1 error-rates for our approach on SVHN and Imagenet32, and Top-5 for Imagenet32.

Weights Precision	**ResNet**	**Epochs**	**Params**		**SVHN**		**I32 Top-1**	**I32 Top-5**
32-bits	20-4	30	∼4.5M		1.75		48.31	24.09
32-bits	20-10	30	27.4M		-		40.62	18.06
1-bit	20-4	30	∼4.5M		1.93		57.62	32.37
1-bit	20-10	30	27.4M		-		46.34	22.77
1-bit	20-15	14*	61.1M		-		45.35*	21.89*

Table 3 shows comparison results from the original work on wide ResNets, and subsequent papers that have reduced error rates on the CIFAR-10 and CIFAR-100 datasets. We also show the only results we know of for 32×32 Imagenet. The current state-of-the-art for SVHN without augmentation is 1.59% (Huang et al., 2016), and with cutout augmentation is 1.30% (Devries & Taylor, 2017). Our full-precision result for SVHN (1.75%) is only a little short of these even though we used only a $4\times$ ResNet, with less than 5 million parameters, and only 30 training epochs.

Table 4 shows comparison results for previous work that trains models by using the sign of weights during training. Additional results appear in Hubara et al. (2016), where activations are quantized, and so the error rates are much larger.

Inspection of Tables 1 to 4 reveals that our $10\times$ baseline full-precision networks, when trained with cutout, surpass the performance of much larger wide ResNets trained with dropout. Even without the use of cutout, our 20-10 network surpasses by over 2% the CIFAR-100 error rate reported for essentially the same network by Zagoruyko & Komodakis (2016) and is marginally better on CIFAR-10 and ImageNet32.

Table 3: Test error rates for full-precision CNN approaches with less than 40 million parameters.

Method	# params	C10	C100	I32 Top-1 / Top-5
1-bit WRN 20-10 (This Paper)	**26.8M**	**4.66**	**19.84**	**46.34 / 22.77**
WRN 22-10 (Zagoruyko & Komodakis, 2016)	26.8M	4.44	20.75	-
WRN 28-10 (Chrabaszcz et al., 2017)	37.1M	-	-	40.97 / 18.87
WRN 28-10 + dropout (Zagoruyko & Komodakis, 2016)	36.5M	3.8	18.3	-
1-bit WRN 20-10 + cutout (This Paper)	**26.8M**	**3.74**	**18.41**	*
ResNeXt-29, $8 \times 64d$ Xie et al. (2016)	34.4M	3.65	17.77	-
Wide ResNet 20-10 (This Paper)	**26.8M**	**3.51**	**17.81**	**40.62 / 18.06**
DenseNets (Huang et al., 2016)	25.6M	3.46	17.18	-
Shake-shake regularization (Gastaldi, 2017)	26.2M	2.86	15.97	-
Shake-shake + cutout (Devries & Taylor, 2017)	26.2M	2.56	15.20	-

Table 4: Test error rates using single-bit weights at test time and propagation during training.

Method	C10	C100	SVHN	Top-1 Imagenet
BC (Courbariaux et al., 2015)	8.27	-	2.30	-
Weight binarization (Merolla et al., 2016)	8.25	-	-	-
BWN (Rastegari et al., 2016)	9.88	-	-	34.5 (full Imagenet)
VGG+HWGQ (Cai et al., 2017)	7.49	-	-	-
BC with ResNet + ADAM (Li et al., 2017)	7.17	35.34	-	52.11 (full Imagenet)
This Paper	**3.74**	**18.41**	**1.93**	**45.35 (Imagenet32)**
BW with VGG (Cai et al., 2017)	-	-	-	34.5 (full Imagenet)

For our 1-bit networks, we observe that there is always an accuracy gap compared to full precision networks. As discussed in Section 5.1, this gap tends to grow as the error-rate in the full-precision network grows.

The use of cutout reduced error rates on CIFAR-10/100 in most cases. In comparison with training $20\times$ ResNets on CIFAR-100, as shown in Table 1, it is as effective to use cutout augmentation in the $10\times$ network to reduce the error rate, but only a quarter of the weights are required.

Figure 4 demonstrates convergence and overfitting trends for CIFAR-10/100 for 20-4 wide ResNets, and a comparison of the use of cutout in 20-10 wide ResNets. Clearly, even for width-4 ResNets, the gap in error rate between full precision weights and single-bit weights is small. Also noticeable is that the warm-restart method enables convergence to very good solutions after just 30 epochs; training longer to 126 epochs reduces error rates further by between 2% and 5%. It can also be observed that the 20-4 network is powerful enough to model the CIFAR-10/100 training sets to well over 99% accuracy, but the modelling power is reduced in the 1-bit version, particularly for CIFAR-100. The reduced modelling capacity for single-bit weights is consistent with the gap in test-error rate performance between the 32-bit and 1-bit cases. When using cutout, training for longer gives improved error rates, but when not using cutout, 126 epochs suffices for peak performance.

Finally, for MNIST and a $4\times$ wide ResNet without any data augmentation, our full-precision method achieved 0.71% after just 1 epoch of training, and 0.28% after 6 epochs, whereas our 1-bit method achieved 0.81% and 0.36% after 1 and 6 epochs of training. For the 1-bit version we also observed 0.27% after 14 epochs. In comparison, 1.29% was reported for the 1-bit case by Courbariaux et al. (2015), and 0.96% by Hubara et al. (2016).

5 Discussion

5.1 The accuracy gap for 1-bit vs 32-bit weights

It is to be expected that a smaller error rate gap will result between the same network using full-precision and 1-bit weights when the test error rate on the full precision network gets smaller. How-

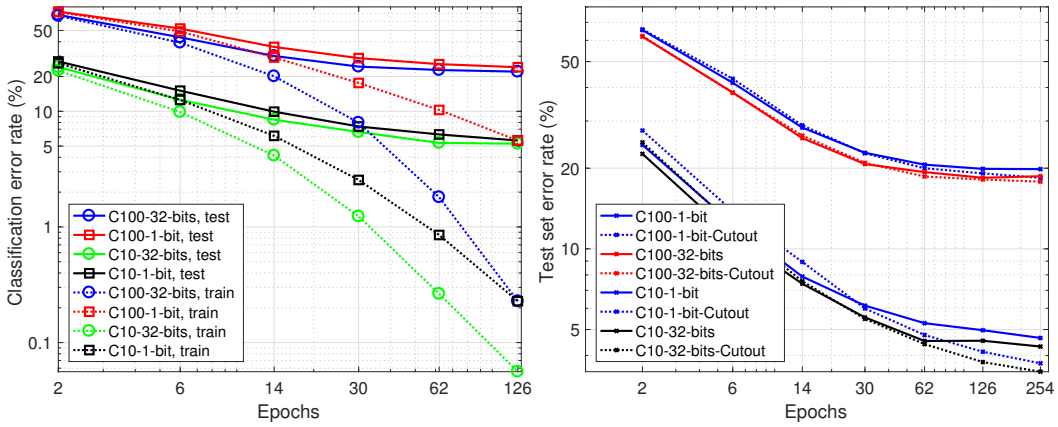

Figure 4: **Convergence through training.** Left: Each marker shows the error rates on the test set and the training set at the end of each cycle of the warm-restart training method, for 20-4 ResNets (less than 5 million parameters). Right: each marker shows the test error rate for 20-10 ResNets, with and without Cutout augmentation. C10 indicates CIFAR-10, and C100 indicates CIFAR-100.

ever, how the magnitude of the gap changes with full precision error rate is dependent on many factors, including the method used to generate models with 1-bit weights.

To illustrate the trend for our approach, we have plotted in Figure 5 the gap in Top-1 error rates vs the Top-1 error rate for the full precision case, for some of the best performing networks for the five datasets we used. Conclusions from this data can only be made relative to alternative methods. Hence, we have also plotted in Figure 5 the error rate and the gap reported by Rastegari et al. (2016) for two different networks using their BWN 1-bit weight method. It is clear that the gaps in those datapoints would not lie on a straight-line fit through the points from our experiments.

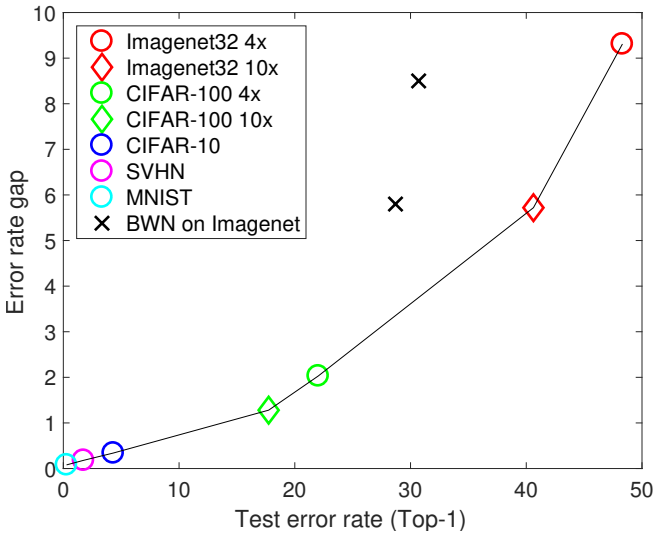

Figure 5: **The error-rate gap between full-precision and 1-bit weight ResNets.** Data in this figure is for our best results for each dataset in experiments without using cutout augmentation.

However, a fair comparison requires further experiments seeking confirmation of whether our method applied to the full Imagenet dataset produces a gap closer to the line connecting our results for the five datasets. These experiments are currently underway.

Two challenges for further work are (1) to devise methods that reduce the gap significantly more than shown here for datasets where error-rates are large; or (2) to predict theoretical bounds on what

the gap should be. For high-error rate cases, the loss function throughout training is much higher for the 1-bit case than the 32-bit case, and hence, the 1-bit network is not able to fit the training set as well as the 32-bit one. Whether this is because of the loss of precision in weights, or due to the mismatch in gradients inherent is propagating with 1-bit weights and updating full-precision weights during training is an open question. If it is the latter case, then it is possible that principled refinements to the weight update method we used will further reduce the gap.

5.2 Comparison with the BWN method

Our approach differs from the BWN method of Rastegari et al. (2016) for two reasons. First, we do not need to calculate mean absolute weight values of the underlying full precision weights for each output channel in each layer following each minibatch, and this enables faster training. Second, we do not need to adjust for a gradient term corresponding to the appearance of each weight in the mean absolute value.

Moreover, although we observed that the method of Rastegari et al. (2016) converges slightly faster early in training, by the end of training our simpler method gave better results for CIFAR-10 and CIFAR-100, and significantly better results for Imagenet32. We speculate that this is because the scaling method of Rastegari et al. (2016) assumes that mean square error between full precision weights and scaled binary weights is the best measure for determining scaling; for practical purposes, minimising mean square error does not necessarily mean better classification error rates. It is also possible that combining the adaptive method with weight decay has undesirable consequences. In our method, since the scale factors never change, weight decay only influences the underlying full-precision weights that get updated, not the propagation of data.

5.3 The influence of not learning batch-normalisation parameters

The unusual design choice of not learning the batch normalization parameters was made for CIFAR-10/100, SVHN and MNIST because for wide ResNets, overfitting is very evident on these datasets (see Figure 4); by the end of training, typically the loss function becomes very close to zero, corresponding to severe overfitting. Inspired by label-smoothing regularization (Szegedy et al., 2015) that aims to reduce overconfidence following the softmax layer, we hypothesised that imposing more control over the standard deviation of inputs to the softmax might have a similar regularizing effect. This is why we removed the final all-to-all layer of most ResNets and replaced it with a 1×1 convolutional layer followed by a batch-normalization layer prior to the global average pooling layer. In turn, not learning a scale and bias for this batch-normalization layer ensures that batches flowing into the softmax layer have standard deviations that do not grow throughout training, which tends to increase the entropy of predictions following the softmax, which is equivalent to lower confidence (Guo et al., 2017).

After observing success with these methods in $10\times$ wide ResNets, we then observed that learning the batch-norm parameters in other layers also led to increased overfitting, and increased test error rates, and so removed that learning in all layers except the first one applied to the input.

As expected from the motivation, we found this method is not appropriate for datasets such as ImageNet where overfitting is not as evident, in which case learning the batch normalization parameters can significantly reduce test error rates.

5.4 Limitations and Further Work

In this paper we focus only on the memory required to store a trained model, i.e. *model compression*. It is also interesting and desirable to reduce the computational load of inference using a trained model, by carrying out layer computations in very few numbers of bits (Hubara et al., 2016; Rastegari et al., 2016; Cai et al., 2017). Facilitating this requires modifying non-linear activations in a network from ReLUs to quantized ReLUs, or in the extreme case, binary step functions. Here we use only full-precision calculations. It can be expected that combining our methods with reduced precision processing will inevitably increase error rates. We have addressed this extension in a forthcoming submission.

ACKNOWLEDGMENTS

Removed for double-blind peer-review.

REFERENCES

Z. Cai, X. He, J. Sun, and N. Vasconcelos. Deep learning with low precision by half-wave Gaussian quantization. *CoRR*, abs/1702.00953, 2017. URL http://arxiv.org/abs/1702.00953.

P. Chrabaszcz, I. Loshchilov, and F. Hutter. A downsampled variant of imagenet as an alternative to the CIFAR datasets. *CoRR*, abs/1707.08819, 2017. URL http://arxiv.org/abs/1707.08819.

M. Courbariaux, Y. Bengio, and J.-P. David. BinaryConnect: Training Deep Neural Networks with binary weights during propagations. *CoRR*, abs/1511.00363, 2015. URL http://arxiv.org/abs/1511.00363.

T. Devries and G. W. Taylor. Improved regularization of convolutional neural networks with cutout. *CoRR*, abs/1708.04552, 2017. URL http://arxiv.org/abs/1708.04552.

X. Gastaldi. Shake-shake regularization. *CoRR*, abs/1705.07485, 2017. URL http://arxiv.org/abs/1705.07485.

C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger. On calibration of modern neural networks. *CoRR*, abs/1706.04599, 2017. URL http://arxiv.org/abs/1706.04599.

K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In *Proc. IEEE International Conference on Computer Vision (ICCV) (see arXiv:1502.01852)*, 2015a.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. Technical report, Microsoft Research, 2015b. Arxiv.1512.03385.

K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. Technical report, Microsoft Research, 2016. Arxiv.1603.05027.

A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017. URL http://arxiv.org/abs/1704.04861.

G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten. Densely connected convolutional networks. *CoRR*, abs/1608.06993, 2016. URL http://arxiv.org/abs/1608.06993.

I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *CoRR*, abs/1609.07061, 2016. URL http://arxiv.org/abs/1609.07061.

F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1MB model size. *CoRR*, abs/1602.07360, 2016. URL http://arxiv.org/abs/1602.07360.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. In *NIPS 2012: Neural Information Processing Systems, Lake Tahoe, Nevada*, 2012.

H. Li, S. De, Z. Xu, C. Studer, H. Samet, and T. Goldstein. Training quantized nets: A deeper understanding. *CoRR*, abs/1706.02379, 2017. URL http://arxiv.org/abs/1706.02379.

M. Lin, Q. Chen, and S. Yan. Network in network. *CoRR*, abs/1312.4400, 2013. URL http://arxiv.org/abs/1312.4400.

I. Loshchilov and F. Hutter. SGDR: stochastic gradient descent with restarts. *CoRR*, abs/1608.03983, 2016. URL http://arxiv.org/abs/1608.03983.

P. Merolla, R. Appuswamy, J. V. Arthur, S. K. Esser, and D. S. Modha. Deep neural networks are robust to weight binarization and other non-linear distortions. *CoRR*, abs/1606.01981, 2016. URL http://arxiv.org/abs/1606.01981.

M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. XNOR-Net: Imagenet classification using binary convolutional neural networks. *CoRR*, abs/1603.05279, 2016. URL http://arxiv.org/abs/1603.05279.

K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. URL http://arxiv.org/abs/1409.1556.

J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. A. Riedmiller. Striving for simplicity: The all convolutional net. *CoRR*, abs/1412.6806, 2014. URL http://arxiv.org/abs/1412.6806.

C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015. URL http://arxiv.org/abs/1512.00567.

S. Xie, R. B. Girshick, P. Dollár, Z. Tu, and K. He. Aggregated residual transformations for deep neural networks. *CoRR*, abs/1611.05431, 2016. URL http://arxiv.org/abs/1611.05431.

S. Zagoruyko and N. Komodakis. Wide residual networks. Arxiv.1605.07146, 2016.

A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen. Incremental network quantization: Towards lossless CNNs with low-precision weights. *CoRR*, abs/1702.03044, 2017. URL http://arxiv.org/abs/1702.03044.

A DIFFERENCES FROM STANDARD RESNETS IN OUR BASELINE

A.1 OUR NETWORK'S FINAL WEIGHT LAYER IS A 1×1 CONVOLUTIONAL LAYER

The most significant difference to He et al. (2016); Zagoruyko & Komodakis (2016) is we exchange the ordering of the global average pooling layer and the final weight layer; our final weight layer becomes a 1×1 convolutional layer with as many channels as there are classes in the training set. This design is not new, but it does seem to be new to ResNets: it corresponds to the architecture of Lin et al. (2013), which originated the global average pooling concept, and also to that used by Springenberg et al. (2014). Using this method means we entirely follow the "all-convolutional" concept of Springenberg et al. (2014).

A.2 OUR NETWORK DOWNSAMPLES THE RESIDUAL PATH USING AVERAGE POOLING

For skip connections between feature maps of different sizes, we use zero-padding for increasing the number of channels as per option 1 of He et al. (2015b). However, for the residual pathway, we use average pooling using a kernel of size 3×3 with stride 2 for downsampling, instead of typical lossy downsampling that discards all pixel values in between samples. We found in some cases this reduced error rates by around 1%.

A.3 WE USE BATCH NORMALIZATION APPLIED TO THE INPUT LAYER

The literature has reported various options for the optimal ordering, usage and placement of BN and ReLU layers in residual networks. Following He et al. (2016); Zagoruyko & Komodakis (2016), we precede convolutional layers with the combination of BN followed by ReLU.

However, different to He et al. (2016); Zagoruyko & Komodakis (2016), we also insert the BN-ReLU combination immediately after the input layer and before the first convolutional layer. Unlike all other batch-normalization layers, we enable learning of the scale and bias factors in this first batch-norm, which enables us to avoid doing any pre-processing on the inputs to the network, since

the BN layer learns how to scale and bias each of the RGB channels; we found that the bias learnt ensures the input to the first ReLU is never negative.

In accordance with our strategy of simplicity, without exception all weight layers can be thought of as a block of three operations in the same sequence, as indicated in Figure 1. Conceptually, batch-norm followed by ReLU can be thought of as a single layer consisting of a ReLU that adaptively changes its centre point and positive slope for each channel and relative to each mini-batch.

We also precede the global average pooling layer by a BN layer, but do not use a ReLU at this point, since nonlinear activation is provided by the softmax layer that follows immediately after global average pooling. We found including the ReLU leads to differences early in training but not by the completion of training.

A.4 Our first conv layer has as many channels as the first residual block

The wide ResNet of Zagoruyko & Komodakis (2016) is specified as always having a constant number of output channels, even when the number of output channels for other layers increases. We found no need to impose this constraint, and instead always allow the first layer to share the same number of output channels as all blocks at the first spatial scale. The increase in the total number of parameters from doing this is small relative to the total number of parameters, since the number of input channels to the first layer is just 3. The benefit of this change is increased simplicity in the network definition, and allow one fewer change in the dimensional of the residual skip pathway.

B Comparison of Residual Networks and Plain All-Convolutional Networks

We were interested in understanding whether the good results we achieved for single-bit weights were a consequence of the skip connections in residual networks. We therefore applied our method to plain all-convolutional networks identical to our $4\times$ residual networks, except with the skip connections removed. Initially, training indicated a much slower convergence, but we found that altering the initial weights standard deviations to be proportional to 2 instead of $\sqrt{2}$ helped, so this was the only other change made. The change was also applied in Equation (1).

As summarised in Figure 6, we found that convergence remained slower than our ResNets, but there was only a small accuracy penalty in comparison with ResNets after 126 epochs of training. This is consistent with the findings of He et al. (2015b) where only ResNets deeper than about 20 layers showed a significant advantage over plain all-convolutional networks. We conclude that our method is not particular to ResNets.

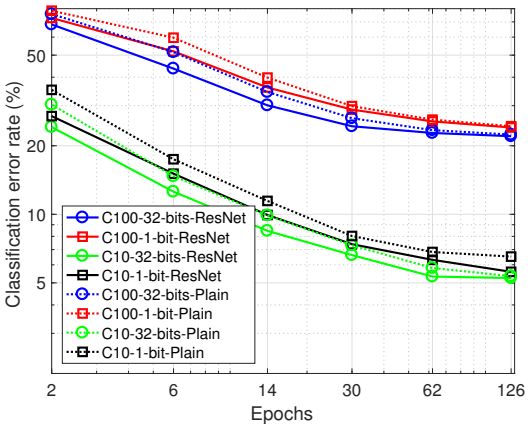

Figure 6: **Residual networks compared with all-convolutional networks.** The data in this figure is for networks with width $4\times$, i.e. with about 4.3 million learnt parameters.