SHAKE-SHAKE REGULARIZATION OF 3-BRANCH RESIDUAL NETWORKS

Xavier Gastaldi
xgastaldi.mba2011@london.edu

ABSTRACT

The method introduced in this paper aims at helping computer vision practitioners faced with an overfit problem. The idea is to replace, in a 3-branch ResNet, the standard summation of residual branches by a stochastic affine combination. The largest tested model improves on the best single shot published result on CIFAR-10 by reaching 2.86% test error. Code is available at https://github.com/xgastaldi/shake-shake

1 INTRODUCTION

Deep residual nets (He et al., 2016a) were first introduced in the ILSVRC & COCO 2015 competitions (Russakovsky et al., 2015; Lin et al., 2014), where they won the 1st places on the tasks of ImageNet detection, ImageNet localization, COCO detection, and COCO segmentation. Since then, significant effort has been put into trying to improve their performance. Scientists have investigated the impact of pushing depth (He et al., 2016b; Huang et al., 2016a), width (Zagoruyko & Komodakis, 2016) and cardinality (Xie et al., 2016; Szegedy et al., 2016; Abdi & Nahavandi, 2016).

On the regularization side, the introduction of Batch Normalization (Ioffe & Szegedy, 2015) reduced the effectiveness of Dropout (Srivastava et al., 2014) on computer vision datasets (see Ioffe & Szegedy (2015); Zagoruyko & Komodakis (2016); Huang et al. (2016b)). Searching for alternatives, researchers started to look at the possibilities specifically offered by multi-branch networks. Some of them noticed that, given the right conditions, it was possible to randomly drop some of the information paths during training (Huang et al., 2016b; Larsson et al., 2016).

Finally, this paper is also related to Shakeout (Kang et al., 2016) (see appendix for details on similarities and differences) and to work which has shown that adding noise to activations or weights reduces overfit (An, 1996; Blundell et al., 2015; Neelakantan et al., 2015).

The method proposed in this document builds on this previous research and aims at improving the generalization ability of 3-branch residual networks by replacing, for the residual branches, the standard summation by a stochastic affine combination.

1.1 MODEL DESCRIPTION

Let x_i denote the tensor of inputs into residual block i. $\mathcal{W}_i^{(1)}$ and $\mathcal{W}_i^{(2)}$ are sets of weights associated with the 2 residual units. $\mathcal{F}$ denotes the residual function, e.g. a stack of two 3x3 convolutional layers. x_{i+1} denotes the tensor of outputs from residual block i.

A typical pre-activation ResNet with 2 residual branches would follow this equation:

$$x_{i+1} = x_i + \mathcal{F}(x_i, \mathcal{W}_i^{(1)}) + \mathcal{F}(x_i, \mathcal{W}_i^{(2)}) \tag{1}$$

Proposed modification - If α_i is a random variable following a uniform distribution between 0 and 1, then during training:

$$x_{i+1} = x_i + \alpha_i \mathcal{F}(x_i, \mathcal{W}_i^{(1)}) + (1 - \alpha_i)\mathcal{F}(x_i, \mathcal{W}_i^{(2)}) \tag{2}$$

Following the same logic as for Dropout, all α_i are set to the expected value of 0.5 at test time.

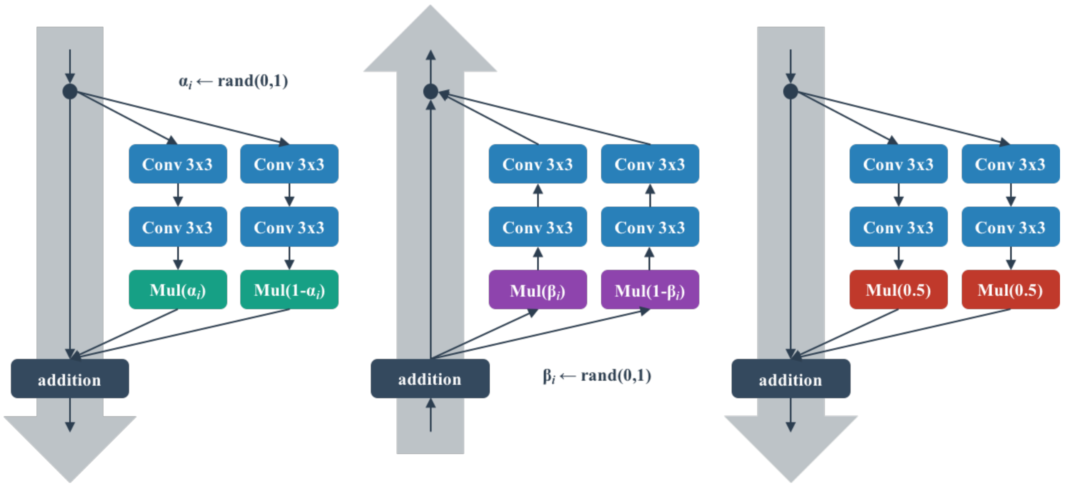

Figure 1: **Left:** Forward training pass. **Center:** Backward training pass. **Right:** At test time.

1.2 Training procedure

As shown in Figure 1, all scaling coefficients are overwritten with new random numbers before each forward pass. The key to making this work is to repeat this coefficient update operation before each backward pass. This results in a stochastic blend of forward and backward flows during training.

1.3 Motivation

This work started as an investigation into the possibility to perform internal tensor augmentation. The core idea behind this attempt was the belief that, for a computer, there is no real difference between an input image and an intermediate representation, both are tensors, the only difference is their shape. We treat them differently because we can't interpret a 128x8x8 tensor but can interpret a 3x32x32 image. A computer doesn't suffer from this prejudice, it sees them all as images. Continuing with this line of thought, a computer probably sees gradients as images too, which means that gradient augmentation might also be a possibility. Assuming this can be done, it is reasonable to assume that this might produce stronger results than noise injection since Batch Normalization reduces the effectiveness of Dropout (at the input image level or inside the network) but does not seem to affect the effectiveness of data augmentation (at the input image level). This architecture was created as an attempt to produce a softer augmentation effect than random crops or flips. The reasoning was that, for a computer, the affine combination would be somewhat similar to blending the pictures of a labrador at 70% opacity and of a chihuahua at 30% opacity. This blended picture would be more difficult to classify as a dog but it wouldn't be impossible.

2 Influence of Forward and Backward training procedures

This architecture was tested on CIFAR-10 (Krizhevsky, 2009). The Shake-Shake code is based on `fb.resnet.torch`[1] and is available at `https://github.com/xgastaldi/shake-shake`. All the implementation details can be found in appendix. The base network is a 26 2x32d ResNet (i.e. the network has a depth of 26, 2 residual branches and the first residual block has a width of 32). "Shake" means that all scaling coefficients are overwritten with new random numbers before the pass. "Even" means that all scaling coefficients are set to 0.5 before the pass. "Keep" means that we keep, for the backward pass, the scaling coefficients used during the forward pass. "Batch" means that, for each residual block i, we apply the same scaling coefficient for all the images in the mini-batch. "Image" means that, for each residual block i, we apply a *different* scaling coefficient for *each* image in the mini-batch. The numbers in Table 1 represent the average of 3 runs except for the 96d models which were run 5 times.

[1]`https://github.com/facebook/fb.resnet.torch`

Table 1: Error rates (%) on CIFAR-10. Results that surpass all competing methods by more than 0.1% are **bold** and the overall best result is **blue**

			Model		
Forward	Backward	Level	26 2x32d	26 2x64d	26 2x96d
Even	Even	n/a	4.27	3.76	3.58
Even	Shake	Batch	4.44	-	-
Shake	Keep	Batch	4.11	-	-
Shake	Even	Batch	3.47	**3.30**	-
Shake	Shake	Batch	3.67	**3.07**	-
Even	Shake	Image	4.11	-	-
Shake	Keep	Image	4.09	-	-
Shake	Even	Image	3.47	**3.20**	-
Shake	Shake	Image	3.55	**2.98**	**2.86**

What can be observed in Table 1 and Figure 2 is that ”Shake-Keep” or ”S-K” models (i.e. ”Shake” → Forward → ”Keep” → Backward) do not have a particularly strong effect on the error rate. The network seems to be able to see through the perturbations when the weight update is done with the same ratios as during the forward pass. ”Even-Shake” only works when applied at the ”Image” level. ”Shake-Even” and ”Shake-Shake” models all produce strong results at 32d but the better training curves of ”Shake-Shake” models start to make a difference when the number of filters of the first residual block is increased to 64d. Applying coefficients at the ”Image” level seems to improve regularization.

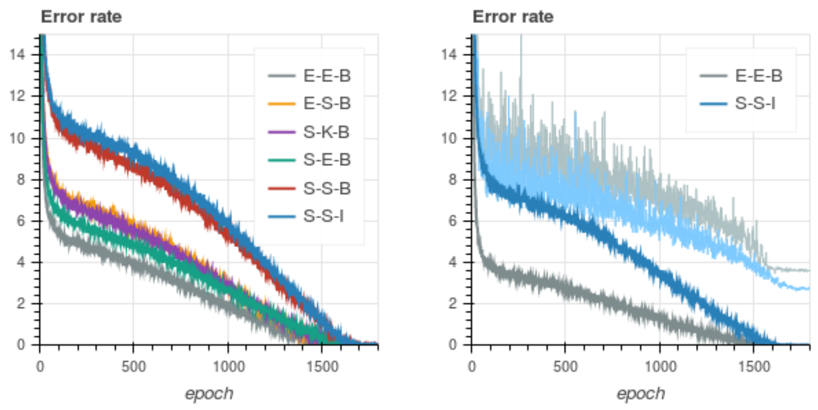

Figure 2: **Left:** Training curves of a selection of 32d models **Right:** Training curves (dark) and test curves (light) of the 96d models.

As of Feb. 16 2017, the best single shot model on CIFAR-10 is a DenseNet-BC k=40 (3.46% error rate) with 25.6M parameters. The second best model is a ResNeXt-29, 16x64d (3.58% error rate) with 68.1M parameters. A small 26 2x32d ”Shake-Even-Image” model with 2.9M parameters obtains approximately the same error rate. That’s roughly 9 times less parameters than the DenseNet model and 23 times less parameters than the ResNeXt model. A 26 2x96d ”Shake-Shake-Image” ResNet with 26.2M parameters, reaches a test error of 2.86%[2].

3 Conclusion

Early experiments seem to indicate an ability to combat overfit. While these results are encouraging, further tests need to be performed to explore the possibilities offered by this regularization method.

[2]Average of 5 runs. Median 2.87%, Min = 2.72%, Max = 2.95%.

REFERENCES

Masoud Abdi and Saeid Nahavandi. Multi-residual networks. *arXiv preprint arXiv:1609.05672*, 2016.

Guozhong An. The effects of adding noise during backpropagation training on a generalization performance. *Neural Comput.*, 1996.

Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural networks. In *ICML'15*. 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016a.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *ECCV*, 2016b.

Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. Densely connected convolutional networks. *arXiv preprint arXiv:1608.06993*, 2016a.

Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q. Weinberger. Deep networks with stochastic depth. In *ECCV*, 2016b.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

Guoliang Kang, Jun Li, and Dacheng Tao. Shakeout: A new regularized deep neural network training scheme. In *AAAI Conference on Artificial Intelligence*, 2016.

Alex Krizhevsky. Learning multiple layers of features from tiny images. *Tech Report*, 2009.

Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. Fractalnet: Ultra-deep neural networks without residuals. *arXiv preprint arXiv:1605.07648*, 2016.

Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: Common objects in context. In *ECCV*, 2014.

Ilya Loshchilov and Frank Hutter. Sgdr: stochastic gradient descent with restarts. *arXiv preprint arXiv:1608.03983*, 2016.

Arvind Neelakantan, Luke Vilnis, Quoc V Le, Ilya Sutskever, Lukasz Kaiser, Karol Kurach, and James Martens. Adding gradient noise improves learning for very deep networks. *arXiv preprint arXiv:1511.06807*, 2015.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *IJCV*, 2015.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alex A. Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *ICLR 2016 Workshop*, 2016.

Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. *arXiv preprint arXiv:1611.05431*, 2016.

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *BMVC*, 2016.

A Implementation details

The first layer is a 3x3 Conv with 16 filters, followed by 3 stages each having 4 residual blocks. The feature map size is 32, 16 and 8 for each stage. Width is doubled when downsampling. The network ends with a 8x8 average pooling and a fully connected layer (total 26 layers deep). Residual paths have the following structure: `ReLU-Conv3x3-BN-ReLU-Conv3x3-BN-Mul`. The skip connections represent the identity function except during downsampling where a slightly customized structure consisting of 2 concatenated flows is used. Each of the 2 flows has the following components: 1x1 average pooling with step 2 followed by a 1x1 convolution. The input of one of the two flows is shifted by 1 pixel right and 1 pixel down to make the average pooling sample from a different position. The concatenation of the two flows doubles the width. Models were trained on the CIFAR-10 50k training set and evaluated on the 10k test set. Standard translation and flipping data augmentation is applied on the 32x32 input image. Due to the introduced stochasticity, all models were trained for 1800 epochs. Training starts with a learning rate of 0.2 and is annealed using a Cosine function *without* restart (see Loshchilov & Hutter (2016)). All models were trained on 2 GPUs with a mini-batch size of 128. Other implementation details are as in `fb.resnet.torch`.

B Analysis of similarities and differences between Shake-Shake and Shakeout

B.1 Similarities

Both methods use the idea of replacing bernoulli variables by scaling coefficients.

B.2 Differences

Starting points: The starting point for Shakeout is Dropout while the starting point for Shake-Shake is a mix of FractalNet drop-path and Stochastic Depth (if we imagine applying drop-path to a 3 branch ResNet where the skip connection is never dropped). Dropping a path is equivalent to setting α_i to 0 or 1.

Multiplications: Both Dropout and Shakeout perform an element-wise multiplication between 2 tensors. In the case of Dropout, the usual steps are to:

1. Create a tensor of the same size as the input tensor
2. Fill this tensor with 0s and 1s taken from a Bernoulli distribution
3. Perform an element-wise multiplication between this noise tensor and the original input

In the case of Shakeout the Bernoulli distribution is replaced by eq (1) in the Shakeout paper.

Shake-Shake, on the other hand, multiplies the whole mini-batch tensor with just one scalar α_i (or $1 - \alpha_i$).

Applying Shake-Shake regularization at the "Image" level is slightly more complex but follows the same logic. Let x_0 denote the original input mini-batch tensor of dimensions 128x3x32x32. The first dimension "stacks" 128 images of dimensions 3x32x32. Inside the second stage of a 26 2x32d model, this tensor is transformed into a mini-batch tensor x_i of dimensions 128x64x16x16. Applying Shake-Shake regularization at the Image level means slicing this tensor along the first dimension and, for each of the 128 slices, multiplying the j^{th} slice (of dimensions 64x16x16) with a scalar $\alpha_{i.j}$ (or $1 - \alpha_{i.j}$).

Forward - Backward: Shakeout keeps the same coefficients between the Forward and Backward passes whereas Shake-Shake updates them before each pass (Forward and Backward).

Number of flows: Shake-Shake regularization works by summing up 2 residual flows plus a skip connection whereas Shakeout only needs one flow.