The supplement of "Sandwich Batch Normalization"

Xinyu Gong Wuyang Chen Tianlong Chen Zhangyang Wang
Department of Electrical and Computer Engineering, the University of Texas at Austin
{xinyu.gong, wuyang.chen, tianlong.chen, atlaswang}@utexas.edu

1. Implementation details

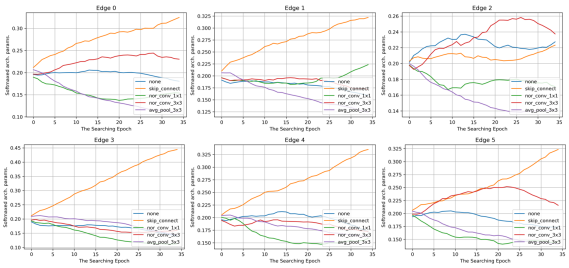
In Figure 1, we show the pseudo algorithms for Batch Normalization (BN) [3], Sandwich Batch Normalization (SaBN) and Sandwich Auxiliary Batch Normalization (SaAuxBN).

```
def BatchNorm(x, gamma, beta, running_mean, running_var, eps=1e-5,
    momentum=0.1):
    # x: input features with shape [N,C,H,W]
    # gamma, beta: scale factors with shape [1,C,1,1]
    mean, var = tf.nn.moments(x, [0, 2, 3], keep dims = True)

    running_mean = (1 - momentum) * running_mean + momentum * mean
    running_var = (1 - momentum) * running_var + momentum * var
    x = (x - running_mean) / tf.sqrt(running_var + eps)
    return x * gamma + beta

def SaBatchNorm(x, sa_gamma, sa_beta, gammas, betas, index,
    running_mean, running_var, eps=1e-5, momentum=0.1):
    # x: input features with shape [N,C,H,W]
    # sa_gamma, sa_beta: shared scale factor with shape [1,C,1,1]
    # gammas, betas: a list of scale factors with shape [1,C,1,1]
    mean, var = tf.nn.moments(x, [0, 2, 3], keep_dims = True)
    running_mean = (1 - momentum) * running_mean + momentum * mean
    running_var = (1 - momentum) * running_var + momentum * var

    x = (x - running_mean) / tf.sqrt(running_var + eps)
    x = x * sa_gamma + sa_beta
    return x * gammas[index] + betas[index]

def SaAuxBatchNorm(x, sa_gamma, sa_beta, gammas, betas, index,
    running_means, running_vars, eps=1e-5, momentum=0.1):
    # x: input features with shape [N,C,H,W]
    # sa_gamma, sa_beta: shared scale factors with shape [1,C,1,1]
    # gammas, betas: a list of scale factors with shape [1,C,1,1]
    mean, var = tf.nn.moments(x, [0, 2, 3], keep dims = True)
    running_means[index] = (1 - momentum) * running_means[index]
                         + momentum * mean
    running_vars[index] = (1 - momentum) * running_vars[index]
                        + momentum * var

    x = (x - running_means[index]) / tf.sqrt(running_vars[index]
        + eps)
    x = x * sa_gamma + sa_beta
    return x * gammas[index] + betas[index]
```

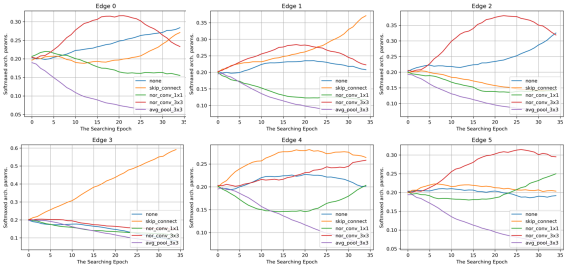
Figure 1: **Pseudo Python code of BN [3], SaBN and AuxSaBN with TensorFlow [1].** We highlight the main difference between our approaches with vanilla BN.

2. Additional Analysis in Neural Architecture Search

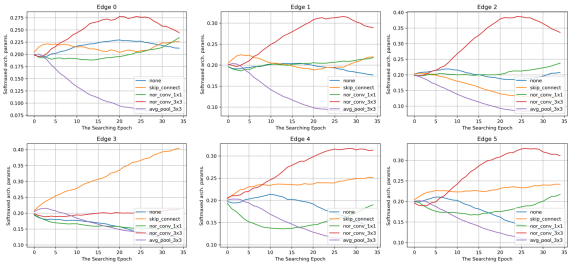
2.1. Architecture Evolving during Neural Architecture Search



(a) The softmaxed architecture parameters on each edge in DARTS during searching.



(b) The softmaxed architecture parameters on each edge in DARTS-CCBN during searching.



(c) The softmaxed architecture parameters on each edge in DARTS-SaBN during searching.

Figure 2: **The evolving trend of architecture parameters.**

The evolution of architecture parameters for each method are presented in Fig. 3.

2.2. Gradient Analysis

We further analyze the gradient magnitude in neural architecture search task. The gradients are taken on the convolution layer in the residual block of NAS-Bench-201 supernet [2]. We visualize the the standard deviation of gradient L2-norm across different architectures in Fig. 4, indicating that the model with SaBN has more balanced gradient magnitude.

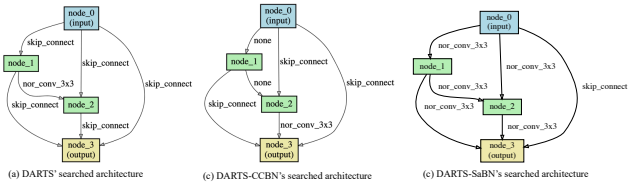


Figure 3: **The derived architectures.** The architectures searched by DARTS are dominated by "skip_connect" and the architecture of DARTS-CCBN is full of both "skip_connect" and "none". In contrast, DARTS-SaBN highly prefers "nor_conv_3x3".

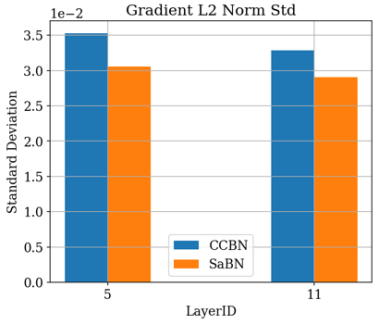


Figure 4: **The standard deviation of gradient magnitude among different architectures.** X-axis denotes the layer depth in the supernet.

3. Additional Analysis on Adversarial Robustness

We analyze the gradient magnitude of the network in adversarial robustness task for SaBN and CCBN. The gradient is taken from the last convolution layer in each network stage. We compare the gradient magnitude difference between clean and adversarial branch via standard deviation of gradient L2-norm. For gradient of clean branch, the clean examples are used as input, while adversarial examples are used as input for the calculation of gradient on adversarial branch. The results are shown in Fig. 5. We can see that model with SaBN has more balanced gradient magnitude between adversarial branch and clean branch.

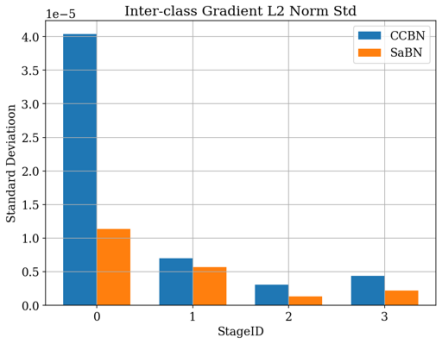


Figure 5: **The gradient L2-norm standard deviation between clean branch and adversarial branch.** X-axis denotes the depth in the supernet.

4. Visualization of Style Transfer

The visual results of style transfer are shown in Fig. 6. Compared with AdaIN and ILM-IN, SaBN generates more visual-appealing images.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Xuanyi Dong and Yi Yang. Nas-bench-102: Extending the scope of reproducible neural architecture search. *arXiv preprint arXiv:2001.00326*, 2020.

[3] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

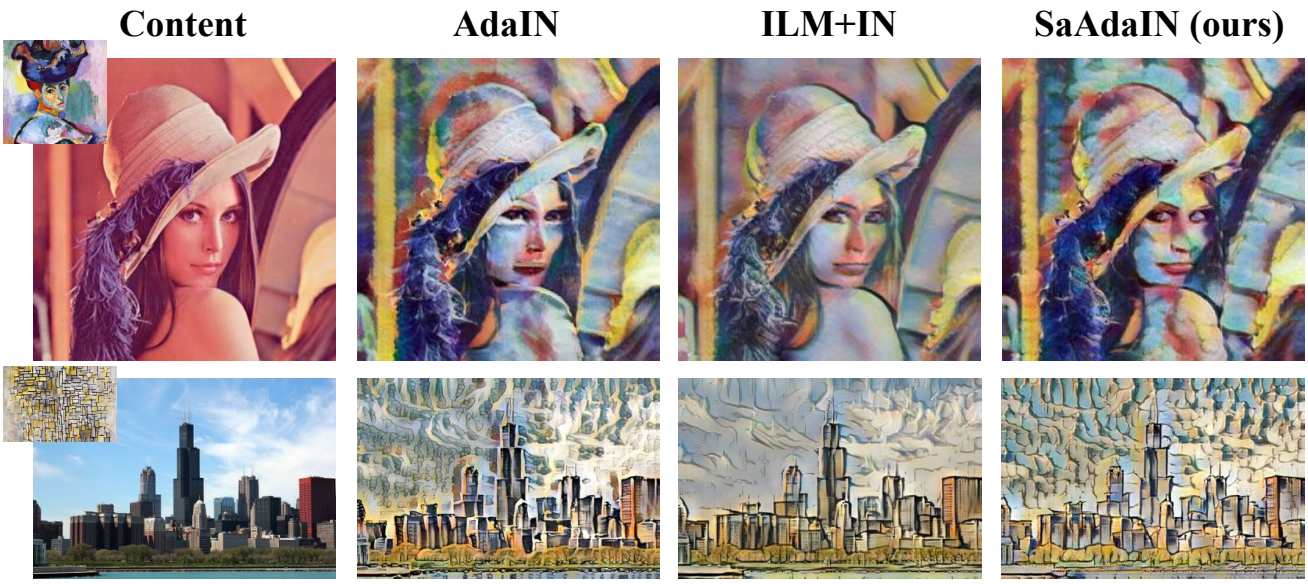


Figure 6: **The visual results of style transfer.** An ideally stylized output should be semantically similar to the content image, while naturally incorporate the style information from the referenced style image.