Spectral Batch Normalization: Normalization in the Frequency Domain

Rinor Cakaj
Image Processing
Robert Bosch GmbH & University of Stuttgart
71229 Leonberg, Germany
Rinor.Cakaj@de.bosch.com

Jens Mehnert
Image Processing
Robert Bosch GmbH
71229 Leonberg, Germany
JensEricMarkus.Mehnert@de.bosch.com

Bin Yang
ISS
University of Stuttgart
70550 Stuttgart, Germany
bin.yang@iss.uni-stuttgart.de

Abstract—Regularization is a set of techniques that are used to improve the generalization ability of deep neural networks. In this paper, we introduce *spectral batch normalization* (SBN), a novel effective method to improve generalization by normalizing feature maps in the frequency (spectral) domain. The activations of residual networks without batch normalization (BN) tend to explode exponentially in the depth of the network at initialization. This leads to extremely large feature map norms even though the parameters are relatively small. These explosive dynamics can be very detrimental to learning. BN makes weight decay regularization on the scaling factors γ, β approximately equivalent to an additive penalty on the norm of the feature maps, which prevents extremely large feature map norms to a certain degree. It was previously shown that preventing explosive growth at the final layer at initialization and during training in ResNets can recover a large part of Batch Normalization's generalization boost. However, we show experimentally that, despite the approximate additive penalty of BN, feature maps in deep neural networks (DNNs) tend to explode at the beginning of the network and that feature maps of DNNs contain large values during the whole training. This phenomenon also occurs in a weakened form in non-residual networks. Intuitively, it is not preferred to have large values in feature maps since they have too much influence on the prediction in contrast to other parts of the feature map. SBN addresses large feature maps by normalizing them in the frequency domain. In our experiments, we empirically show that SBN prevents exploding feature maps at initialization and large feature map values during the training. Moreover, the normalization of feature maps in the frequency domain leads to more uniform distributed frequency components. This discourages the DNNs to rely on single frequency components of feature maps. These, together with other effects (e.g. noise injection, scaling and shifting of the feature map) of SBN, have a regularizing effect on the training of residual and non-residual networks. We show experimentally that using SBN in addition to standard regularization methods improves the performance of DNNs by a relevant margin, e.g. ResNet50 on CIFAR-100 by 2.31%, on ImageNet by 0.71% (from 76.80% to 77.51%) and VGG19 on CIFAR-100 by 0.66%.

I. Introduction

Deep neural networks contain multiple non-linear hidden layers which make them powerful machine learning systems [1]. However, such networks are prone to overfitting [2] due to the limited size of training data or the high capacity of the model. Overfitting describes the phenomenon where a neural network (NN) perfectly fits the training data while achieving poor performance on the test data. Regularization is a set of techniques used to reduce overfitting and is therefore a key element in deep learning [3]. It allows the model to generalize well to unseen data.

Many methods have been developed to regularize DNNs, weight penalties as L_1-regularization [4] and weight decay [5], soft weight sharing [6], dropout [1], data augmentation [7] and ensemble learning methods [8].

Normalization techniques [9]–[12] like batch normalization (BN) [9] normalize features by subtracting the mean and dividing by the standard deviation computed across different dimensions of a feature map. In some cases such normalization techniques act as regularizers, eliminating the need for dropout [9]. There are different explanations for the regularizing effects of BN: (i) the stochastic uncertainty of the batch statistics can benefit generalization [11], (ii) BN reduces the explosive growth of feature maps in deeper layers which acts as a regularizer [13] and (iii) BN discourages reliance on a single neuron and encourages different neurons to have equal magnitude in the sense that corrupting individual neurons does not harm generalization [14].

The feature maps (i.e. activations) of residual networks without BN tend to explode exponentially in the depth of the network at initialization [15]. This leads to extremely large feature map norms even though the parameters are relatively small [13]. Large and exploding feature maps also occur in a weakened form in non-residual networks. These explosive dynamics can be very detrimental to learning [15].

Intuitively, it is not preferred to have large values in the feature maps since they have too much influence on the prediction in contrast to other parts of the feature map. Therefore some sort of scaling is needed to restrict the influence of single outputs and to distribute the decision making process on a larger part of the feature map.

With ormalization techniques is weight decay regularization on the scaling and shifting factors γ, β approximately equivalent to an additive penalty on the norm of the feature maps [13]. This prevents exploding features during training to some degree. Dauphin et al. [13] showed experimentally that preventing explosive growth at the final layer at initialization and during training can recover a large part of BNs generalization boost.

However, in our analysis of feature maps for different networks with BN we show that feature maps tend to explode at the beginning of the training and that they contain large values during the whole training despite the additive penalties γ, β. Figure 1 compares the feature map norm over the course of training of a BN layer ("BN"), two subsequent BN layers ("2BN") and a *Spectral Batch Normalization* (SBN) layer following a BN layer ("BN + SBN") in a ResNet50 and VGG19 trained on CIFAR-100 for an arbitrary batch. The figure shows that SBN prevents exploding feature maps at initialization and large feature map norms during the whole training. Note that the Subfigure 1(a) has logarithmic scaling.

To prevent exploding feature maps at initialization and large feature maps during the whole training, SBN normalizes the feature maps in the frequency (spectral) domain.

Firstly, SBN computes the 2-dimensional discrete Fourier transform (DFT) of the feature maps. Secondly, it computes the channel-wise mean and standard deviation of the DFT coefficients across all frequencies and the mini-batch. Thirdly, the frequency components are normalized. After a re-scaling and re-shifting by learned parameters, the feature map is transformed back into the spatial domain using the inverse DFT.

SBN prevents exploding feature maps at initialization and large values in feature maps during the whole training which would destabilize the training and would lead to a more complex model compared to DNNs with small feature maps. Furthermore, SBN has more positive effects: (i) it inserts stochastic noise in the frequency domain which makes the network more robust to small perturbations in the frequency domain and (ii) re-scaling and re-shifting gives the network the opportunity to weight channels in the frequency domain and (iii) it leads to more uniform distributed frequency components in the feature maps which discourage the network to rely on single frequency components.

In Section IV we will show experimentally that using SBN in addition to BN prevents large values in the feature maps during the whole training and therefore reduces overfitting in DNNs. The feature maps are scaled down to smaller values in comparison to only using BN. Furthermore, we will show that SBN increases the accuracy of networks by a relevant margin compared to the original models.

A. Contributions of this Paper and Applications

In this work, we present *spectral batch normalization*, a novel effective method to improve generalization by normalizing feature maps in the frequency (spectral) domain. Our core contributions are:

- Introducing spectral batch normalization.
- Analyzing the impact of our method on the weights and feature maps (in the spatial and frequency domain) during the training process.
- Showing experimentally that using spectral batch normalization in addition to standard regularization methods increases the performance of various different network architectures on CIFAR-10/CIFAR-100 and on ImageNet.

The additional gains in performance of ResNet50 is on CIFAR-10 by 1.40%, on CIFAR-100 by 2.32%, on TinyImageNet by 2.36% and on ImageNet by 0.71% are worth noting. Moreover, the performance of non-residual networks is also improved, e.g. VGG19 on CIFAR-100 by 0.66%.

II. RELATED WORK

A. Regularization

Regularization is one of the key elements of deep learning [3], allowing the model to generalize well to unseen data even when trained on a finite training set or with an imperfect optimization procedure [16]. There are several techniques to regularize NNs which can be categorized into groups. Data augmentation methods like cropping, flipping, and adjusting brightness or sharpness [7] and cutout [17] transform the training dataset to avoid overfitting. Regularization techniques like dropout [1], dropblock [18], dropconnect [2] drop neurons or weights from the NN during training to prevent units from co-adapting too much [1]. Furthermore, NNs can be regularized using penalty terms in the loss function. Weight decay [5], [19] encourages the weights of the NN to be small in magnitude. The L_1-regularization [4] forces the weights of non-relevant features to zero.

B. Normalization

Normalization methods are an essential building block of the most successful deep learning architectures and have enabled training very deep residual networks [15].

Batch normalization (BN) [9] normalizes features by subtracting the mean and dividing by the standard deviation computed within a mini-batch. The normalization is done to reduce the internal covariate shift, i.e. the change in the distribution of network activations due to the change in network parameters during training. The features are then scaled and shifted by learned parameters to maintain the expressive power of the network [3]. During the training, a moving average of the means and variances is computed which are then used during inference. Batch normalization enables faster and more stable training of deep NNs. Moreover, it also acts as a regularizer, in some cases eliminating the need for dropout.

The effect of batch normalization is dependent on the mini-batch size and it is not obvious how to apply it to recurrent neural networks. Layer normalization [10] computes the mean and variance used for normalization from all the features in a layer on a single training case. It improves the training time and generalization performance of several sequential models (RNN [20]).

Instance normalization [12] normalizes across the width and height of a single feature map of a single example, i.e. not across the channels like layer normalization. It works well for generative models.

Layer normalization and instance normalization have limited success in visual recognition. Therefore, Wu et al. [11] present group normalization which avoids normalizing along the batch dimension but divides the channels into groups
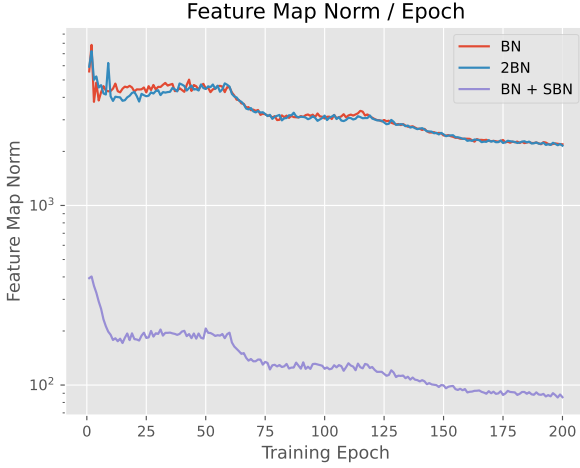
(a) Feature map norm of BN, 2BN and BN + SBN layer per epoch for an arbitrary batch in ResNet50 trained on CIFAR100.
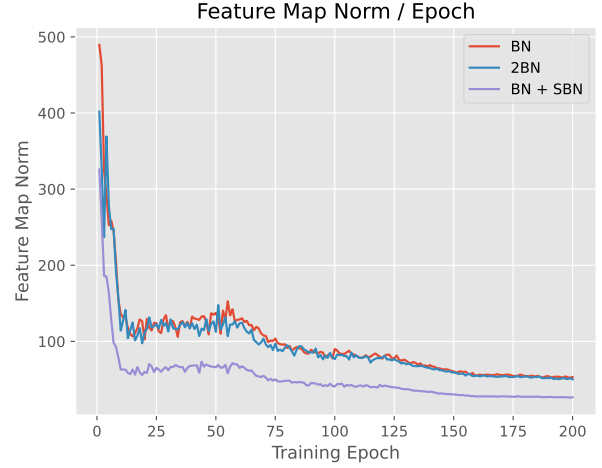
(b) Feature map norm of BN, 2BN and BN + SBN layer per epoch for an arbitrary batch in VGG19 trained on CIFAR100.

Fig. 1: Feature map norms per epoch. The subfigures show that using SBN leads to smaller feature map norms. Note that the Subfigure 1(a) has logarithmic scaling which makes the explosion not so visible.

and computes the mean and variance within each group for normalization.

Weight normalization [21] is a weight reparameterization approach that accelerates the convergence of SGD optimization. It reparameterizes the weight vectors of each layer such that the length of those weight vectors is decoupled from their direction. In detail, they express the weight vector $w \in \mathbb{R}^d$ by $w = \Psi(v, g) = \frac{g}{\|v\|} v$ where $v \in \mathbb{R}^d$ is the new weight vector and $g \in \mathbb{R}$ the scalar parameter.

There is a number of works that analyzed the reasons for the success of normalization methods. In the case of BN, Ioffe et al. [9] stated that it reduces the internal covariate shift, i.e. the change in the distribution of each layer's input during training. However, Santurkar et al. [22] demonstrate that the distributional stability of layer inputs has little to do with the success of BN. Instead, they point out that BN makes the optimization landscape significantly smoother. This would induce a more predictive and stable behavior of the gradients, allowing faster training. It has been shown that the activations of residual networks without BN tend to explode exponentially in the depth of the network initialization [15]. Preventing the explosive growth at the final layer at initialization and during the training can recover a large part of BNs generalization effect [13].

C. Frequency

Due to the dual of the convolution theorem, stating that multiplication in the time/spatial domain is equivalent to convolution in the frequency domain, the DFT is used in deep learning to provide a significant speedup in the computation of convolutions. Moreover, Rippel et al. [23] showed that the frequency domain also provides a powerful representation to model and train CNNs. They introduced spectral pooling

which performs a dimensionality reduction by truncating the representation in the frequency domain.

The DFT is also used to regularize DNNs. Spectral dropout [24] prevents overfitting by eliminating weak Fourier domain coefficients below a fixed threshold and randomly dropping a fixed percentage of the remaining Fourier domain coefficients of the neural network activations.

Furthermore, the frequency domain can be used to prune NNs [25].

III. METHOD

In order to counteract exploding or large feature maps, we want to use a frequency decomposition. It allows manipulating an input across its various length-scales of variation, and as such provides a natural framework for the manipulation of data with spatial coherence [23].

Inspired by batch normalization [9], we introduce *spectral batch normalization*, which has a regularizing effect on DNNs, by normalizing feature maps in the frequency domain. SBN prevents explosive dynamics during the initialization and leads to smaller feature maps during the whole training. To introduce spectral batch normalization, we need the discrete Fourier transformation.

A. The Discrete Fourier Transform

The discrete Fourier transform converts an input array of real (or complex) numbers into another sequence of complex numbers. In our case, it is a frequency domain representation of the original spatial input sequence. The 2D DFT of a matrix $\mathbf{x} \in \mathbb{R}^{H \times W}$ is defined by

$$\hat{\mathbf{x}}_{k,l} := \mathscr{F}(\mathbf{x})_{k,l} := \sum_{h=0}^{H-1} \sum_{w=0}^{W-1} \mathbf{x}_{h,w} \cdot e^{-2\pi i \cdot \left(\frac{h \cdot k}{H} + \frac{w \cdot l}{W} \right)} \quad (1)$$

for $k = 0, \ldots, H-1$ and $l = 0, \ldots, W-1$. Its inverse transform is given by $\mathscr{F}^{-1}(\cdot) = \mathscr{F}^*(\cdot)/(HW)$, i.e. the conjugate of the transform normalized by $1/(HW)$.

Intuitively, the DFT decomposes an input array into the frequencies contained in the input sequence. Roughly speaking, it compares a basis of complex sinusoidal functions with the input sequence and computes the similarity between the complex sinusoidal functions and the input sequence.

The DFT of a real signal is Hermitian-symmetric, i.e. $\hat{\mathbf{x}}_{k,l} = \text{conj}(\hat{\mathbf{x}}_{H-i,W-j})$. Therefore, only $HW/2$ complex numbers are needed to represent the real signal in the frequency domain. This does not reduce the effective dimensionality of the transformed input since each DFT coefficient consists of a real and imaginary component.

Before we explain spectral batch normalization, we have to discuss how to propagate the gradient through a Fourier transform layer. This is well described by Rippel et al. [23]. Let $\mathbf{x} \in \mathbb{R}^{H \times W}$ be the input and $\mathscr{F}(x) = \mathbf{y} \in \mathbb{C}^{H \times W}$ be the output of a DFT. Moreover, let $Z: \mathbb{C}^{H \times W} \to \mathbb{R}$ be a real-valued loss function applied to $\mathbf{y}$ which can be considered as the remainder of the forward pass. The DFT is a linear operator, therefore its gradient is the transformation matrix itself. During back-propagation, the gradient is conjugated [26]. This corresponds to the application of the inverse transform

$$\frac{\partial Z}{\partial \mathbf{x}} = \mathscr{F}^{-1}\left(\frac{\partial Z}{\partial y}\right). \qquad (2)$$

We use the real 2D DFT implementation of PyTorch [27], which uses the FFT algorithm to compute the transformation. The implementation of the inverse real 2D DFT in PyTorch uses zero-padding to get the original array size $H \times W$ for signals with odd length in a transformed dimension.

B. Spectral Batch Normalization

Let $\mathbf{X} \in \mathbb{R}^{B \times C \times H \times W}$ be a feature map. The introduced spectral batch normalization block first computes the real 2-dimensional discrete Fourier transform $\hat{\mathbf{X}} \in \mathbb{C}^{B \times C \times H \times W/2}$ of the last two dimensions of the feature map $\mathbf{X}$. Secondly, it computes the channel-wise mean and standard deviation of the DFT coefficients across all frequencies and a mini-batch. Hence, the resulting mean and variance have size $1 \times C \times 1 \times 1$. During the training, a moving average of the means and standard deviations is computed which are then used during inference. Then, the transformed feature maps are normalized channel-wise using the computed mean and standard deviation (like in batch normalization [9]). To recover the representation power of the layer, the feature maps are scaled and shifted channel-wise by learnable parameters $\gamma \in \mathbb{R}^C$ and $\beta \in \mathbb{R}^C$. After the normalization and scale/shift step, the feature map is transformed back using the inverse real DFT. SBN is presented in Algorithm 1.

At this point, the question could arise why we compute the **channel-wise** mean and standard deviation of the feature maps across a mini-batch, i.e. the resulting mean and variance has size $1 \times C \times 1 \times 1$.

Algorithm 1: Spectral Batch Normalization Layer

Parameter: momentum=$\lambda \in (0,1)$, epsilon=$\epsilon \in \mathbb{R}$,
running_mean=$\mu_r \in \mathbb{R}^C = \mathbf{0}$,
running_variance=$\sigma_r^2 \in \mathbb{R}^C = \mathbf{1}$

Input: $\mathbf{x} \in \mathbb{R}^{B \times C \times H \times W}$

Output: $\mathbf{y} \in \mathbb{R}^{B \times C \times H \times W}$

```
/* Compute the 2D-DFT of x, i.e. on
   the last two dimensions.        */
```
1 $\hat{\mathbf{x}} = \mathscr{F}(x)$
```
// Compute mini-batch mean/variance
```
2 $\mu = \text{mean}(\hat{\mathbf{x}}, \dim = [0,2,3])$
3 $\sigma^2 = \text{var}(\hat{\mathbf{x}}, \dim = [0,2,3])$
```
// Compute running mean/variance
```
4 $\mu_r = \lambda \cdot \mu + (1-\lambda) \cdot \mu_r$
```
// Compute running unbiased variance,
   n=CHW/2
```
5 $\sigma_r^2 = \lambda \cdot \sigma^2 \cdot \frac{n}{n-1} + (1-\lambda) \cdot \sigma_r^2$
6 $\bar{\mathbf{x}} = \frac{\hat{\mathbf{x}} - \mu}{\sqrt{\sigma^2 + \epsilon}}$ `// Normalize`
7 $\hat{\mathbf{y}} = \gamma \cdot \bar{\mathbf{x}} + \beta$ `// Scale and shift`
```
      channel-wise
```
8 $\mathbf{y} = \mathscr{F}^{-1}(\hat{\mathbf{y}})$ `// Compute inverse 2D DFT`

Different channels in a feature map are often independent and represent different information depending on the input. Therefore, it is not beneficial to normalize across different channels. Moreover, computing the mean and standard deviation across a mini-batch and the channels and *not* over the frequency components leads for some datasets (e.g. ImageNet) to a huge number of additional scaling and shifting weights (i.e. $\#H \cdot W$).

We now have a short look at the effects of SBN on the frequency components in the feature map. The resulting values of a DFT are complex numbers. Let $z = a + ib$ be a complex number. Every nonzero complex number can be written in the form $re^{i\theta}$, where $r := \sqrt{a^2 + b^2}$ is the **magnitude** of z, and $\theta \in [0, 2\pi)$ is the **phase, angle** or **argument** of z. The magnitude of z represents the amplitude of the corresponding frequency. Hence, the magnitude gives us information about which frequency components are mainly represented in our feature map. The phase of z represents the phase of the corresponding frequency, i.e. the spatial delay for that frequency in the feature map. Due to the normalization and scaling/shifting, the magnitude and the phase of the frequency components are affected by SBN.

SBN computes the mean and variance of the complex numbers using

$$\hat{\mu} = \frac{1}{N} \sum_{j=1}^{N} z_j = \frac{1}{N} \sum_{j=1}^{N} a_j + i \cdot \frac{1}{N} \sum_{j=1}^{N} b_j \qquad (3)$$

$$\hat{\sigma}^2 = \frac{1}{N} \sum_{j=1}^{N} |\hat{z}_j - \hat{\mu}|^2. \qquad (4)$$

Then the mean is substracted from the DFT coefficients. The

real and complex part of the DFT coefficients are changed separately to have zero mean (see Equation (3)). This leads to DFT coefficients which are distributed around $z_0 = 0 + i \cdot 0$. Subsequently, the DFT coefficients are divided by the standard deviation.

These steps make the magnitudes of the frequency components smaller which then leads to smaller values in the feature maps in the spatial domain. Moreover, the frequency components become more uniformly distributed, i.e. the importance of frequency components with large magnitudes is reduced. On the other hand, the frequency components with low magnitudes get a higher influence on the feature map. The effects on the phase are difficult and cannot be stated clearly. The effects of SBN on the the frequency components are discussed empirically in Section IV-D2.

Lastly, we want to point out where to insert a SBN layer. Empirically, it is preferred to insert the SBN layer in deeper layers after the BN layer. More details are discussed in Section V-C.

C. Regularizing effects

There are several components of SBN which act as a regularizer during training.

1) Preventing large feature map norm: As stated in the introduction, the activations of residual networks without BN tend to explode exponentially in the depth of the network at initialization [15] which can be detrimental to learning.

Batch normalization prevents exploding feature maps during the training to a certain degree by normalizing the feature maps. Dauphin et al. [13] showed that most of the regularizing effect of BN comes from the prevention of explosive growth from cascading to the output during training.

Our analysis of feature maps for different networks (with and without residual connections) trained with BN on different datasets showed that large values in feature maps occur during the whole training despite the usage of BN.

Using SBN in addition to BN reduces the explosive growth of feature maps further due to the normalization process in the frequency domain. The feature maps are scaled down to smaller values compared to only using BN (see Section IV).

2) Noise Injection: SBN subtracts a random value (the mean of the mini-batch across each feature map and mini-batch) from each DFT transformed feature map. Moreover, SBN divides each DFT transformed feature map by a random value (the standard deviation of the mini-batch across each feature map and mini-batch) at each step of training. Because different examples are randomly chosen for inclusion in the mini-batch at each step, the standard deviation and mean randomly fluctuate. Both of the sources of noise manipulate the feature maps in the frequency domain. Hence every layer has to learn to be more robust to a lot of variation in its inputs [3]. Therefore the stochastic uncertainty of the batch statistics acts as a regularizer during training [10], [11].

3) Re-scaling and Re-shifting: As in the standard BN, we introduce weights γ and biases β to recover the representation power of feature maps in the frequency domain by multiplying the weights and biases with the DFT coefficient. Since weights and biases are learned through SGD, the network learns to prioritize the channels of the feature maps in the frequency domain.

4) More uniform distribution of frequency components: As stated in Section III-B, SBN leads to more uniformly distributed frequency components. Hence, the influence of strong frequency components characterized via high magnitudes is reduced. On the other hand, the importance of weak frequency components is relatively increased. The effect has a regularizing effect on the training, since it discourages the DNN to rely on single frequency components. This is the key difference between SBN and BN, because BN normalizes feature maps only in the spatial domain.

IV. EXPERIMENTS

We experimentally validate the usefulness of our method in supervised image recognition. We compare DNNs using SBN in addition to standard regularization methods against DNNs using only standard regularization methods.

ResNets consist of four modules made up of basic and bottleneck building blocks, a convolutional layer at the beginning of the network and a linear layer at the end of the network. VGGs have a similar module structure. Through our experiments, we figured out that SBN should preferably be inserted in deeper layers of DNNs. The additions to the names of the models describe in which modules SBN is applied. For example "ResNet50 + SBN34" means that, a ResNet50 network is used where in the third and fourth module a SBN layer is inserted after each BN layer.

In order to achieve a fair comparison, we also added results of ResNets with two BN layers in the same modules where SBN is inserted, e.g. "ResNet50 + 2BN34" is the abbreviation for a ResNet50 with two BN layers, one after the other, in the third and fourth module.

A. Image Classification on CIFAR-10/100

We evaluate the performance of SBN on the CIFAR-10/100 classification datasets [28]. The CIFAR-10/100 datasets consist of 50,000 training and 10,000 test 32×32 color images. We used various different network architectures, the CNNs ResNet18, ResNet34, ResNet50 [29] and VGG16/19-BN [30].

The experiments were run five times with different random seeds for 200 epochs, resulting in different network initializations, data orders and additionally in different data augmentations. For every case we report the mean test accuracy and standard deviation. We used a $9/1$-split between training examples and validation examples and saved the best model on the validation set. This model was then used for evaluation on the test dataset.

The baseline networks ResNet18, ResNet34 and ResNet50 were trained with data augmentation, weight decay and early stopping. The baseline network VGG16-BN and VGG19-BN were trained with data augmentation, weight decay, early stopping and dropout in the fully connected layers.

TABLE I: Accuracy of experiments on CIFAR-10.

Model	Accuracy
ResNet18	$94.04\% \pm 0.08\%$
ResNet18 + 2BN34	$93.90\% \pm 0.26\%$
ResNet18 + SBN34	$\mathbf{94.43}\% \pm \mathbf{0.11}\%$
ResNet34	$93.69\% \pm 0.30\%$
ResNet34 + 2BN34	$94.01\% \pm 0.22\%$
ResNet34 + SBN34	$\mathbf{94.67}\% \pm \mathbf{0.18}\%$
ResNet50	$93.31\% \pm 0.36\%$
ResNet50 + 2BN34	$93.75\% \pm 0.19\%$
ResNet50 + SBN34	$\mathbf{94.72}\% \pm \mathbf{0.18}\%$
VGG16-BN	$93.27\% \pm 0.12\%$
VGG16-BN + 2BN4	$93.29\% \pm 0.15\%$
VGG16-BN + SBN4	$\mathbf{93.48}\% \pm \mathbf{0.19}\%$
VGG19-BN	$93.21\% \pm 0.08\%$
VGG19-BN + 2BN34	$93.15\% \pm 0.15\%$
VGG19-BN + SBN34	$\mathbf{93.34}\% \pm \mathbf{0.10}\%$

TABLE II: Accuracy of experiments on CIFAR-100.

Model	Accuracy
ResNet18	$76.47\% \pm 0.20\%$
ResNet18 + 2BN34	$76.41\% \pm 0.15\%$
ResNet18 + SBN34	$\mathbf{77.11}\% \pm \mathbf{0.30}\%$
ResNet34	$77.07\% \pm 0.45\%$
ResNet34 + 2BN34	$77.05\% \pm 0.25\%$
ResNet34 + SBN34	$\mathbf{78.01}\% \pm \mathbf{0.45}\%$
ResNet50	$76.17\% \pm 0.69\%$
ResNet50 + 2BN34	$76.53\% \pm 1.03\%$
ResNet50 + SBN34	$\mathbf{78.49}\% \pm \mathbf{0.24}\%$
VGG16-BN	$72.48\% \pm 0.35\%$
VGG16-BN + 2BN4	$72.45\% \pm 0.17\%$
VGG16-BN + SN4	$\mathbf{72.79}\% \pm \mathbf{0.27}\%$
VGG19-BN	$71.34\% \pm 0.14\%$
VGG19-BN + 2BN4	$71.30\% \pm 0.30\%$
VGG19-BN + SBN4	$\mathbf{72.00}\% \pm \mathbf{0.26}\%$

TABLE III: Top-1 Accuracy of experiments on TinyImageNet.

Model	Accuracy
ResNet18	$63.52\% \pm 0.25\%$
ResNet18 + 2BN34	$63.60\% \pm 0.07\%$
ResNet18 + SBN34	$\mathbf{64.69}\% \pm \mathbf{0.23}\%$
ResNet50	$65.42\% \pm 0.60\%$
ResNet50 + 2BN4	$65.35\% \pm 0.26\%$
ResNet50 + SBN4	$\mathbf{67.79}\% \pm \mathbf{0.22}\%$

1) Implementation Details: For our experiments we used PyTorch 1.10.1 [27] and one Nvidia GeForce 1080Ti GPU.

All networks were trained with a batch size of 128. We used the SGD optimizer with momentum 0.9 and initial learning rate 0.1. For ResNet18, ResNet34 and ResNet50 trained on CIFAR-10 we decayed the learning rate by 0.1 at epoch 90 and 136 and used weight decay with the factor $1e-4$ (as in [29]). For all networks trained on CIFAR-100 and for VGG16/19 with BN trained on CIFAR-10, we decayed the learning rate by 0.2 at epochs 60, 120 and 160 and used weight decay with the factor $5e-4$ as in [31]. Dropout is used in the fully connected layers in VGG16-BN and VGG19-BN with a dropout rate of 0.5. We want to point out that we have not done a hyperparameter search for the learning rate, weight decay, etc. We used the same setting as in [29], [31].

The training data is augmented by using random crop with size 32 and padding 4, random horizontal flip and normalization [7]. The test set is only normalized. The ResNet networks are initialized Kaiming-uniform [32]. In the VGG networks the linear layers are initialized Kaiming-uniform. The convolutional layers are initialized with a Gaussian normal distribution with mean 0 and standard deviation $2/n$ where n is the size of the kernel multiplied with the number of output channels in the same layer.

2) Results: Table I presents the results of our experiments on CIFAR-10. Using SBN improves the accuracy for all ResNets and VGGs. The additional gain in performance by using SBN in the ResNet50 network ($+1.40\%$) is worth noting. Using two BN layers increases the performance in some cases. However, the performance gains are small compared to those of SBN.

Table II shows the results of our experiments on CIFAR-100. Similar to the experiments on CIFAR-10, ResNets generalize worse if more parameters are trainable. Additionally, the gain in performance by using SBN in the ResNet50 network ($+2.31\%$) and VGG19 ($+0.66\%$) are worth noting. The use of two BN layers slightly increases the performance in some cases. However, the performance gains are small compared to those of SBN.

B. Image Classification on TinyImageNet

TinyImageNet [33] contains 100,000 images of 200 classes downsized to 64×64 colored images. Each class has 500 training images and 50 validation images.

The experiments were run five times with different random seeds for 300 epochs using ResNet18 and ResNet50. Following the common practice, we report the top-1 classification accuracy on the validation set.

1) Implementation Details: For our experiments we used PyTorch 1.10.1 [27] and one Nvidia GeForce 1080Ti GPU.

We trained all the models using the SGD optimizer with momentum 0.9 and initial learning rate 0.1 decayed by factor 0.1 at epochs 75, 150, and 225. Moreover, we used weight decay with the factor $1e-4$. The batch size is set to 128.

The training data is augmented by using random crop with size 224, random horizontal flip and normalization. The validation data is augmented by resizing to 256×256, using center crop with size 224 and normalization.

2) Results: Table III presents the top-1 accuracy of our experiments on TinyImageNet. ResNet18 using SBN in the third and fourth block outperforms the standard ResNet18 by 1.16%. Using two BN layers in the third and fourth block slightly improves ResNet18 but slightly worsens the performance of ResNet50. The additional gain in performance for ResNet50 is worth noting since using SBN improved the baseline by 2.36%.

TABLE IV: Top-1 Accuracy of experiments on ImageNet.

Model	Accuracy
ResNet50	$76.80\% \pm 0.11\%$
ResNet50 + 2BN4	$76.90\% \pm 0.08\%$
ResNet50 + SBN4	$\mathbf{77.51\% \pm 0.07\%}$

TABLE V: Results of learning rate experiments for ResNet50 on CIFAR-10/100 .

Model	LR	Dataset	Accuracy
ResNet50	0.2	CIFAR-10/CIFAR-100	$93.69\%/77.19\%$
ResNet50 + SBN34	0.2	CIFAR-10/CIFAR-100	$\mathbf{95.08\%}/\mathbf{77.91\%}$
ResNet50	0.5	CIFAR-10/CIFAR-100	$93.58\%/\mathbf{66.67\%}$
ResNet50 + SBN34	0.5	CIFAR-10/CIFAR-100	$\mathbf{94.59\%}/65.82\%$

C. ImageNet

The ILSVRC 2012 classification dataset [34] contains 1.2 million training images, 50,000 validation images and 150,000 test images. They are labeled with 1,000 categories.

Regularization methods on ImageNet require a greater number of training epochs to converge [18]. The experiments were run three times with different random seeds for 400 epochs using ResNet50. Following the common practice, we report the top-1 classification accuracy on the validation set.

1) Implementation Details: For our experiments we used PyTorch 1.10.1 [27] and 4 Nvidia GeForce 1080Ti GPU.

We used the same hyperparameter setting as [35]. We trained all the models using the SGD optimizer with momentum 0.9 and initial learning rate 0.1 decayed by factor 0.1 at epochs 75, 150, 225, 300 and 375. Moreover, we used weight decay with the factor $1e-4$. The batch size is set to 256.

The training data and validation data is augmented as described for TinyImageNet.

2) Results: Table IV presents the top-1 accuracy of our experiments on ImageNet. Using SBN in the last block of ResNet50 increases the accuracy by 0.71% on ImageNet. The usage of two BN layers increases the performance only by 0.10%. This shows that it is advantageous to use SBN.

D. Effects on the Training

We used different measures to analyze the impact of SBN on the training of the networks: (i) we tracked the weight/gradient norm and weight/gradient distribution for every layer, (ii) we measured the feature map norm and distribution after every convolution, BN and SBN layer, (iii) we analyzed the effects of SBN on the magnitude and phase of the frequency components of feature maps.

We did not see any differences in the weight/gradient norm and weight/gradient distribution between networks with and without SBN. Moreover, there were no clear differences in the phases of the frequency components of feature maps before and after a SBN layer.

However, there were strong differences in the feature map norms and distributions between DNN with and without SBN. Furthermore, there were large differences in the magnitudes of the frequency components of feature maps before and after a SBN layer.

1) Differences in the Feature Map Norms and Distributions: We measured the feature map norm and distribution by looking at the feature map for an arbitrary batch during training. Figure 1(a) shows the feature map norm per epoch of three networks: (i) a standard ResNet50, (ii) a ResNet50 using two BN layers and (iii) a ResNet50 using BN and SBN. The output of a SBN layer has a much smaller feature map norm compared to the

output of one or of two BN layers. This can be seen also in 1(b) for the non-residual network VGG19.

Figure 2 shows the distribution of values in a feature map after a BN layer of a ResNet50 trained on CIFAR-100 without SBN and after the corresponding SBN layer of a ResNet50 trained on CIFAR-100 with SBN for an arbitrary batch. Without SBN feature maps have high values during the whole training. Using SBN prevents large and exploding values in feature maps.

2) Magnitude of the Frequency Components: We analyzed the effects of SBN on the magnitudes of the frequency components of feature maps. We looked at the feature maps of different images before and after the SBN layer in fully trained networks. The feature maps were transformed using the DFT and shifted such that the DC term is moved to the center of the tensor.

Figure 3 shows the magnitudes of frequency components of different channels in a feature map after a BN layer and after the SBN layer behind it for one arbitrary image from CIFAR-100. It shows how the magnitude are manipulated due to the SBN layer. SBN reduces the relevance of frequency components with high magnitudes, i.e. the importance of frequency components with low magnitude is increased. Hence, the magnitudes of the frequency components become more uniform distributed. This discourages reliance on a single frequency component and encourages the network to focus on more frequency components in the prediction process. Figure 3 generalizes to other images and is not an unique phenomenon.

V. ABLATION STUDY / FAQ

### A. Does SBN allow higher learning rates?

Similar to BN, SBN allows the usage of higher learning rates. We tested initial learning rates of 0.2 and 0.5 with ResNet50 on CIFAR-10/100 over three runs and could achieve better results using BN and SBN than only using BN (see Table V). The accuracy of ResNet50 with SBN on CIFAR-100 using a learning rate of 0.5 is lower than the baseline, however, the baseline algorithm did not perform well either.

B. Is there an acceleration of the training? How high is the computational overhead of SBN?

For easy tasks (e.g. training ResNet18/34/50 on CIFAR-10) we saw a huge acceleration of the training measured per epoch (see Figure 4). However, due to the Fourier transformations, the training time is longer compared to the baseline. The training time overhead of SBN depends on the network architecture. Clearly, if more SBN layers are used, the training time
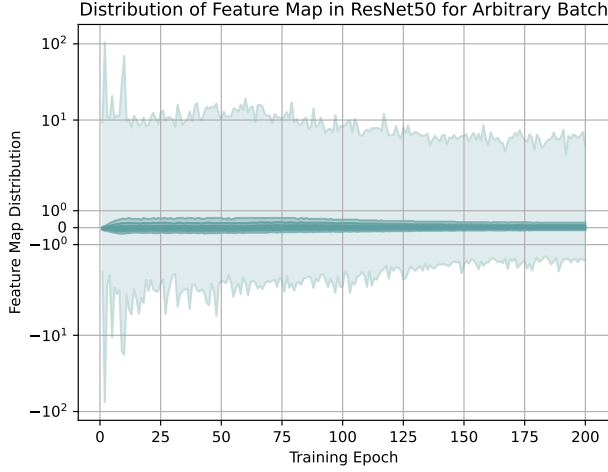
(a) Feature map distribution per epoch after the BN layer.
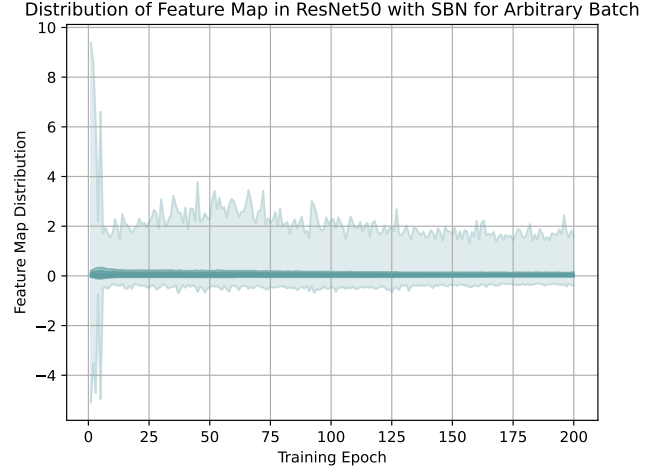
(b) Feature map distribution per epoch after the SBN layer.

Fig. 2: Feature map distribution per epoch after a BN layer 2(a) of a ResNet50 trained on CIFAR100 without SBN and after the corresponding SBN layer 2(b) of a ResNet50 trained on CIFAR-100 with SBN. From top to bottom, the lines represent percentiles with values: maximum; 93%; 84%; 69%; 50%; 31%; 16%; 7%; minimum. Please pay attention on the different scalings in Fig. 2(a) and 2(b).
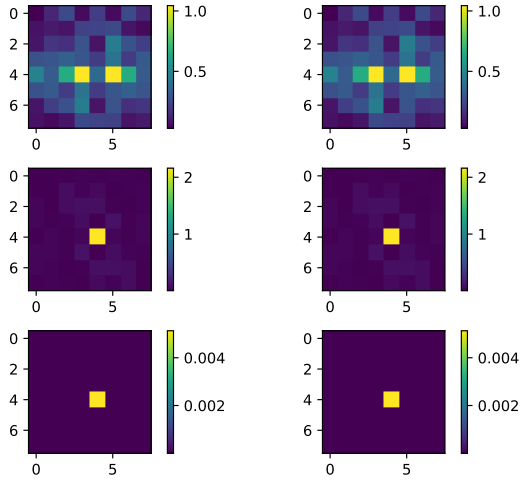
(a) Magnitudes of frequency components of a feature map after a BN layer.
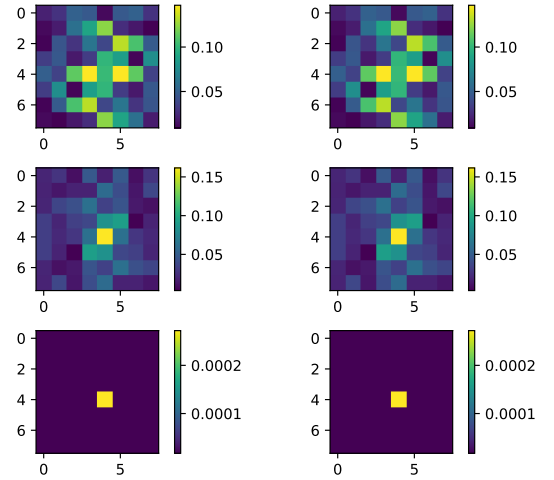
(b) Magnitudes of frequency components of a feature map after a SBN layer.

Fig. 3: Magnitudes of frequency components of different channels of a feature map after a BN layer 3(a) and the SBN layer behind it 3(b).

overhead increases. Table VI shows the training time overhead of ResNets on CIFAR-10/CIFAR-100 and ImageNet. Note that we use the real 2D DFT implementation of PyTorch [27], which uses the FFT algorithm to compute the transformation.

C. Where to insert SBN?

We performed an ablation study to find the perfect position for inserting a SBN layer. Firstly, it is preferred to insert a SBN after a BN layer. Inserting SBN prior to a BN layer does not improve generalization since the BN layer scales the

feature maps up again. Hence, the input feature map of the following convolutional layer again has a large feature map norm due to the normalization process of BN. Using a SBN without a BN layer does not improve the generalization since SBN does not replace BN. The normalization process in the spatial domain is needed for the training. However, using SBN in addition to BN increases the performance as shown in the previous experiments.

Beyond that, it is favorable to insert SBN in deeper layers, e.g. in ResNets for example in modules 3/4. The reason for that

TABLE VI: Approximate training time overhead of ResNets on CIFAR10/CIFAR100 and ImageNet when using SBN in %.

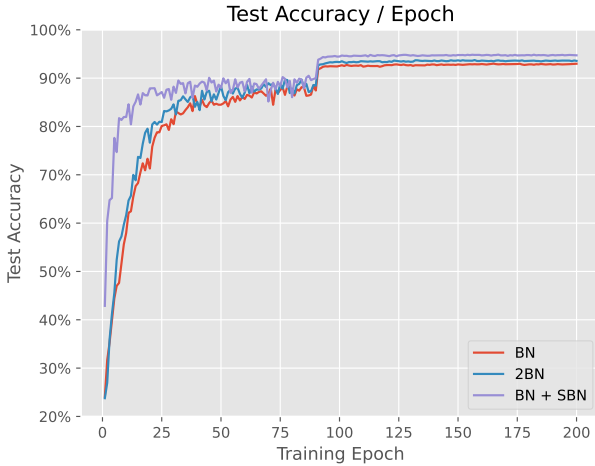| Model	Dataset	Training time overhead
ResNet18	CIFAR-10/CIFAR-100	16%/16%
ResNet34	CIFAR-10/CIFAR-100	23%/27%
ResNet50	CIFAR-10/CIFAR-100	50%/50%
ResNet50	ImageNet	12%

Fig. 4: Test Accuracy of ResNet50, ResNet50 with 2BN layer and ResNet50 with SBN on CIFAR-10.

is the phenomenon of large or exploding feature maps happening more often in deeper than in shallow layers [15]. However, inserting SBN after every BN layer in ResNet18/34/50 did also improve the accuracy on CIFAR-10 and CIFAR-100 but we got the best results applying SBN only in the modules 3/4. When the task gets even harder (e.g. ImageNet), it is preferred to insert it only in the fourth block. Otherwise, the regularizing effects are to strong.

D. Is it sufficient to only subtract the expectation?

The generalization boost can be partly reproduced by only subtracting the expectation. However, it does not work as well as applying the whole SBN algorithm.

E. Is it sufficient to only divide by the standard deviation?

Same answer as for the question above.

F. Is it sufficient to only normalize the feature map in the frequency space without re-scaling and re-shifting?

Same answer as for the question above.

G. Is it sufficient to only down-scale the feature map in the frequency representation?

The generalization boost can be partly reproduced by down-scaling the frequency representation. However, it does not work as well as applying the whole SBN algorithm. Moreover, this introduces one or more hyperparameters depending on

whether one decides to divide all feature maps by one single value or different feature maps by different values.

H. Is it sufficient to only-down scale the feature map?

Same answer as for the question above.

I. Is it sufficient to only use weighting of the feature maps in the frequency representation?

The generalization boost cannot be reproduced by only weighting the feature maps in the frequency domain. However, we need the weighting in the algorithm since only normalizing the feature map in the frequency domain does not reproduce the whole generalization boost.

J. Is it favorable to normalize real and imaginary part of the frequency components separately?

It is not favorable to normalize the real and imaginary part separately as it has negative effects on the performance.

VI. CONCLUSION AND FUTURE WORK

We presented spectral batch normalization (SBN), a novel effective method to improve generalization by normalizing feature maps in the frequency (spectral) domain. SBN prevents large feature maps, introduces stochastic noise in the frequency domain of the feature maps and leads to more uniform distributed frequency components. These effects act as regularizers during training.

Using SBN in addition to commonly used regularization methods (e.g. [1], [5], [7], [9], [19]) increases the performance of ResNets and VGGs on CIFAR-10, CIFAR-100 and ImageNet. The additional gain in performance of ResNet50 on CIFAR-10 by 1.40%, on CIFAR-100 by 2.32% and on ImageNet by 0.71% are worth noting.

We have not explored the full range of possibilities of SBN. Our future work will include applying SBN to other normalization techniques, e.g. Layer Normalization [10], Group Normalization [11], Instance Normalization [12]. Moreover, we will perform further investigations to analyze the impact on the loss landscape as in [22]. This could be extended to a theoretical analysis of SBN where we analyze the impact on the Lipschitz continuity of the loss. Furthermore, the weights γ and biases β could be learned in the complex domain which could be advantageous for training.

REFERENCES

[1] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014. [Online]. Available: http://dl.acm.org/citation.cfm?id=2670313

[2] L. Wan, M. D. Zeiler, S. Zhang, Y. LeCun, and R. Fergus, "Regularization of Neural Networks using DropConnect," in *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, ser. JMLR Workshop and Conference Proceedings, vol. 28. JMLR.org, 2013, pp. 1058–1066. [Online]. Available: http://proceedings.mlr.press/v28/wan13.html

[3] I. J. Goodfellow, Y. Bengio, and A. C. Courville, *Deep Learning*, ser. Adaptive computation and machine learning. MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org/

[4] R. Tibshirani, "Regression Shrinkage and Selection via the Lasso," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 58, no. 1, pp. 267–288, 1996. [Online]. Available: http://www.jstor.org/stable/2346178

[5] A. Krogh and J. A. Hertz, "A Simple Weight Decay Can Improve Generalization," in *Advances in Neural Information Processing Systems 4, [NIPS Conference, Denver, Colorado, USA, December 2-5, 1991]*, J. E. Moody, S. J. Hanson, and R. Lippmann, Eds. Morgan Kaufmann, 1991, pp. 950–957. [Online]. Available: http://papers.nips.cc/paper/563-a-simple-weight-decay-can-improve-generalization

[6] S. J. Nowlan and G. E. Hinton, "Simplifying Neural Networks by Soft Weight-Sharing," *Neural Comput.*, vol. 4, no. 4, pp. 473–493, 1992. [Online]. Available: https://doi.org/10.1162/neco.1992.4.4.473

[7] C. Shorten and T. M. Khoshgoftaar, "A survey on Image Data Augmentation for Deep Learning," *J. Big Data*, vol. 6, p. 60, 2019. [Online]. Available: https://doi.org/10.1186/s40537-019-0197-0

[8] D. W. Opitz and R. Maclin, "Popular Ensemble Methods: An Empirical Study," *J. Artif. Intell. Res.*, vol. 11, pp. 169–198, 1999. [Online]. Available: https://doi.org/10.1613/jair.614

[9] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," in *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, ser. JMLR Workshop and Conference Proceedings, F. R. Bach and D. M. Blei, Eds., vol. 37. JMLR.org, 2015, pp. 448–456. [Online]. Available: http://proceedings.mlr.press/v37/ioffe15.html

[10] L. J. Ba, J. R. Kiros, and G. E. Hinton, "Layer Normalization," *CoRR*, vol. abs/1607.06450, 2016. [Online]. Available: http://arxiv.org/abs/1607.06450

[11] Y. Wu and K. He, "Group Normalization," *Int. J. Comput. Vis.*, vol. 128, no. 3, pp. 742–755, 2020. [Online]. Available: https://doi.org/10.1007/s11263-019-01198-w

[12] D. Ulyanov, A. Vedaldi, and V. S. Lempitsky, "Instance Normalization: The Missing Ingredient for Fast Stylization," *CoRR*, vol. abs/1607.08022, 2016. [Online]. Available: http://arxiv.org/abs/1607.08022

[13] Y. Dauphin and E. D. Cubuk, "Deconstructing the Regularization of BatchNorm," in *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. [Online]. Available: https://openreview.net/forum?id=d-XzF81Wg1

[14] P. Luo, X. Wang, W. Shao, and Z. Peng, "Towards Understanding Regularization in Batch Normalization," in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. [Online]. Available: https://openreview.net/forum?id=HJlLKjR9FQ

[15] H. Zhang, Y. N. Dauphin, and T. Ma, "Fixup Initialization: Residual Learning Without Normalization," in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. [Online]. Available: https://openreview.net/forum?id=H1gsz30cKX

[16] J. Kukacka, V. Golkov, and D. Cremers, "Regularization for Deep Learning: A Taxonomy," *CoRR*, vol. abs/1710.10686, 2017. [Online]. Available: http://arxiv.org/abs/1710.10686

[17] T. Devries and G. W. Taylor, "Improved Regularization of Convolutional Neural Networks with Cutout," *CoRR*, vol. abs/1708.04552, 2017. [Online]. Available: http://arxiv.org/abs/1708.04552

[18] G. Ghiasi, T.-Y. Lin, and Q. V. Le, "DropBlock: A regularization method for convolutional networks," in *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., vol. 31. Curran Associates, Inc., 2018. [Online]. Available: https://proceedings.neurips.cc/paper/2018/file/7edcfb2d8f6a659ef4cd1e6c9b6d7079-Paper.pdf

[19] I. Loshchilov and F. Hutter, "Decoupled Weight Decay Regularization," in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. [Online]. Available: https://openreview.net/forum?id=Bkg6RiCqY7

[20] D. E. Rumelhart and J. L. McClelland, *Learning Internal Representations by Error Propagation*, 1987, pp. 318–362.

[21] T. Salimans and D. P. Kingma, "Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks," in *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, D. D. Lee, M. Sugiyama,

U. von Luxburg, I. Guyon, and R. Garnett, Eds., 2016, p. 901. [Online]. Available: https://proceedings.neurips.cc/paper/2016/hash/ed265bc903a5a097f61d3ec064d96d2e-Abstract.html

[22] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry, "How Does Batch Normalization Help Optimization?" in *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., 2018, pp. 2488–2498. [Online]. Available: https://proceedings.neurips.cc/paper/2018/hash/905056c1ac1dad141560467e0a99e1cf-Abstract.html

[23] O. Rippel, J. Snoek, and R. P. Adams, "Spectral Representations for Convolutional Neural Networks," in *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds., 2015, pp. 2449–2457. [Online]. Available: https://proceedings.neurips.cc/paper/2015/hash/536a76f94cf7535158f66cfbd4b113b6-Abstract.html

[24] S. H. Khan, M. Hayat, and F. Porikli, "Regularization of Deep Neural Networks with Spectral Dropout," *Neural Networks*, vol. 110, pp. 82–90, 2019. [Online]. Available: https://doi.org/10.1016/j.neunet.2018.09.009

[25] Z. Liu, J. Xu, X. Peng, and R. Xiong, "Frequency-Domain Dynamic Pruning for Convolutional Neural Networks," in *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., 2018, pp. 1051–1061. [Online]. Available: https://proceedings.neurips.cc/paper/2018/hash/a9a6653e48976138166de32772b1bf40-Abstract.html

[26] J. Bassey, L. Qian, and X. Li, "A Survey of Complex-Valued Neural networks," *CoRR*, vol. abs/2101.12249, 2021. [Online]. Available: https://arxiv.org/abs/2101.12249

[27] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Z. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, Eds., 2019, pp. 8024–8035. [Online]. Available: https://proceedings.neurips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html

[28] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," *Master's thesis, Department of Computer Science, University of Toronto*, 2009.

[29] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. IEEE Computer Society, 2016, pp. 770–778. [Online]. Available: https://doi.org/10.1109/CVPR.2016.90

[30] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1409.1556

[31] S. Zagoruyko and N. Komodakis, "Wide Residual Networks," in *Proceedings of the British Machine Vision Conference (BMVC)*, E. R. H. Richard C. Wilson and W. A. P. Smith, Eds. BMVA Press, September 2016, pp. 87.1–87.12. [Online]. Available: https://dx.doi.org/10.5244/C.30.87

[32] K. He, X. Zhang, S. Ren, and J. Sun, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification," in *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*. IEEE Computer Society, 2015, pp. 1026–1034. [Online]. Available: https://doi.org/10.1109/ICCV.2015.123

[33] Y. Le and X. S. Yang, "Tiny ImageNet Visual Recognition Challenge," 2015.

[34] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. S. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge,"

Int. J. Comput. Vis., vol. 115, no. 3, pp. 211–252, 2015. [Online]. Available: https://doi.org/10.1007/s11263-015-0816-y

[35] S. Yun, D. Han, S. Chun, S. J. Oh, Y. Yoo, and J. Choe, "CutMix: Regularization Strategy to Train Strong Classifiers With Localizable Features," in *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019.* IEEE, 2019, pp. 6022–6031. [Online]. Available: https://doi.org/10.1109/ICCV.2019.00612