Weight and Gradient Centralization in Deep Neural Networks

Wolfgang Fuhl
University of Tübingen
Sand 14, 72076 Tübingen, Germany
wolfgang.fuhl@uni-tuebingen.de

Enkelejda Kasneci
University of Tübingen
Sand 14, 72076 Tübingen, Germany
enkelejda.kasneci@uni-tuebingen.de

Abstract

Batch normalization is currently the most widely used variant of internal normalization for deep neural networks. Additional work has shown that the normalization of weights and additional conditioning as well as the normalization of gradients further improve the generalization. In this work, we combine several of these methods and thereby increase the generalization of the networks. The advantage of the newer methods compared to the batch normalization is not only increased generalization, but also that these methods only have to be applied during training and, therefore, do not influence the running time during use. Link to CUDA code `https://atreus.informatik.uni-tuebingen.de/seafile/d/8e2ab8c3fdd444e1a135/`

1. Introduction

Deep neural networks (DNN) [50] are currently the most successful machine learning method and owe their recent progress to the steadily growing data sets [48], improvements in massively parallel architectures [46], high-speed bus systems such as PCIe, optimization methods [57, 44], new training techniques [36, 45], validation [19, 26], and the regularly growing fields of application like eye tracking [6, 15, 14] for pupil [33, 22, 17, 15, 31, 13] or eyelid extraction [29, 28, 30], semantic segmentation [16, 27, 12] or gesture recognition [7]. These advances in technology make it possible to train deep neural networks on huge datasets like ImageNet [48], however, further techniques had to be introduced to prevent the gradients from becoming too small [39]. The normalization of the data [42] has a huge impact on the generalization of large networks. Generalization alone is not the only quality feature of a good learning process of neural networks. Another important point is the acceleration of the learning process and the resource-saving [20] use of the techniques. This is due to the fact that the most successful architectures already have an intrinsically high resource requirement and additional techniques to improve generalization can, therefore, only use a small number of supplementary resources. This can be seen very clearly when comparing the optimization techniques themselves. The most popular methods are Stochastic Gradient Decent (SGD) with momentum [57] and Adam [44] which introduces a second momentum. There are many other optimization algorithms [57, 44, 2, 4], but SGD and Adam are the most popular. Both methods allow batch based learning and require only a constant multiple of the gradient (for the momentum) as additional memory. Comparing this with the Levenberg-Marquardt algorithm (LM) [51, 53], which was the most popular method for training neural networks for quite some time, it is noticeable that the memory consumption in the case of LM grows quadratic to the weights. This is due to the fact that the LM algorithm calculates the exact derivatives for each weight over the whole network and not only local derivatives as is the case with backpropagation. Further procedures like the residual layers [39], weight initialization strategy [35, 38], activation functions [54], gradient clipping [55, 56], algorithms for adaptive learning rate optimization [57, 44], and many more have been introduced and are subject to the same conditions of generalization improvement, training stabilization, and resource conservation.

In neural networks themselves, statistics are also collected and used to balance the forward and backward flow of data and errors. The best known method used directly on the activation of neurons is Batch Normalization (BN) [42]. Other procedures that work on the activation functions are instance normalization (IN) [63, 41], layer normalization (LN) [1] and group normalization (GN) [64]. These procedures smooth the optimization landscape [61] and lead to an improvement of the generalization. The disadvantages of BN are that it continues to process the data in the neural network as an independent layer even after training and that it must be applied to a relatively large batch size. To avoid these disadvantages, weight normalization (WN) [60, 40] and weight standardization (WS) [58] were introduced. These must only be applied during training and are independent of the batch size. WN limits the weight

vectors via different standards whereas WS normalizes the weight vectors via the mean and standard deviation. A newer technique that works only on the gradient is Gradient Centralization (GC) [65], which subtracts the mean value from the gradient. All these advanced techniques smooth the error space and lead to a faster and, typically, better generalization of neural networks.

In this work, we deal with these extended techniques and seek to find a good combination of the methods. In our evaluation, it has been shown that the combination of the filter mean subtraction and the gradient mean subtraction in union is very effective for different networks. We have also tested other combinations and found that many also depend on batch normalization. Our main contributions are as follows:

1 The combination of mean gradient and mean filter subtraction
2 Publicly available CUDA implementations
3 Description of the integration into the back propagation algorithm
4 A comprehensive comparison with advanced techniques

2. Related Work

In this section, we describe the related work based on three groups. The first group is the manipulation of the data after the activation functions, which has the disadvantage that the activation functions have to be executed in the later application of the model. The second group is the manipulation of the weights during training. Here, the weights can be standardized or otherwise restricted. The last group is the manipulation of the gradients. In this instance, after each back propagation, normalizations and restrictions are applied to the gradients before they are being used to change the weights.

2.1. Manipulation of the output of the activations

This type of normalization is the most common use of internal manipulation in DNNs today. In batch normalization (BN) [42], the mean value and standard deviation are calculated over several batches and used for normalization. This gives the output of neurons after the activation layer a mean value of zero and uniform variance. With group normalization GN [64], groups are formed over which normalization is performed unlike BN where normalization occurs over the number of copies in a batch, this eliminates the need for large batches, which is the case with BN. Other alternatives are instance normalization IN [63, 41] and layer normalization LN [1]. For IN, each specimen is used individually for the calculation of the mean and standard deviation, and for LN, the individual layers. IN and LN have been successfully used for recurrent neural networks (RNN) [62]. However, all these methods have the disadvantage that normalization has to be applied even after training.

2.2. Manipulation of the weights of the model

In weight normalization (WN) [60, 40], the weights of the neural network are multiplied by a constant divided by the Euclidean distance of the weight vector of a neuron. This decouples the weights with respect to their length, thereby accelerating the training. An extension of this method is weight standardization (WS) [58], which does not require a constant, but calculates the mean and standard deviation thus normalizing the weights. Like the previous manipulation methods, this method smooths the error landscape, which speeds up training and levels the generalization of the final model. An advantage of these methods is that they only have to be applied during training and not in the final model. These methods, however, have a limitation and that is the fine-tuning of neural networks. If the original model was not trained with a weight normalization, these methods cannot be used for fine-tuning without creating a high initial error on the model. This is due to the fact that the restrictions and norms for the original model's weights most likely do not apply.

2.3. Manipulation of the gradient after back propagation

Another very common technique is gradient manipulation over the first [57] and second moment [44]. This gradient impulse allows neural networks to be trained in a stable way without the gradients exploding, which is interpreted as a damped oscillation. The second momentum leads, in most cases, to a faster generalization, but the model's final performance is usually slightly worse when compared to training with only the first momentum. These moments are moving averages which are formed over the calculated gradients and represent a pre-determined portion of the next weight adjustment. An advanced method in this area is gradient clipping [55, 56] wherein randomly selected gradients are set to zero or a small random value is added to each gradient. Another technique is to project the gradients onto subspaces [37, 49, 3]. Here, for example, the Riemannian approach is used to map the gradients onto a Riemannian manifold. After the mapping, the gradients are used to adjust the weights. Finally, in [65] a very simple procedure was presented which subtracts the current mean value of the gradients in addition to the moments.

3. Method

Since our approach is a combination of several previously published approaches (Weight mean subtraction and gradient centralization), we proceed as follows in this section: we formally describe the already published methods and introduce a naming convention, which we use later in the evaluation. This should make it easier for the reader to evaluate the effectiveness of different methods. In the

following, we will refer to operations on the data in forward propagation as $F_{s,c,y,x}$ where s is the sample, c is the channel and y, x is the spatial position in the data. For the weights we use $W_{out,in,y,x}$. Where out is the output channel, in is the input channel and y, x is the spatial position for fully connected layers. In the case of convolution layers, y, x is the position in the two-dimensional convolution mask, which together with in defines the convolution tensor. To manipulate the gradient we use $\Delta W_{out,in,y,x}$ with the same indices as we used for the weights ($W_{out,in,y,x}$). Since a normalization can be applied not only to the data and the gradient, but also to the back propagated error, the error is denoted by $E_{s,c,y,x}$ where the indices are the same as the indices of the forward propagated data ($F_{s,c,y,x}$).

3.1. Weight normalization

In this section, the equations used for weight normalization are presented. In all equations, j represents the axis to which the normalization was performed orthogonally. This means that we have calculated a separate mean value, standard deviation or Euclidean distance for each index of j.

$$W_{j,in,y,x} = W_{j,in,y,x} * \frac{k}{||W_{j,in,y,x}||} \tag{1}$$

In Equation 1, the weight normalization [60, 40] is described (WN). This normalizes each weight in a tensor with the ratio of a constant k (in our experiments 1) divided by the Euclidean distance of the tensor.

$$W_{j,in,y,x} = W_{j,in,y,x} - \overline{W}_{j,in,y,x} \tag{2}$$

Since the pure normalization over the mean value of the tensor of the weights has no separate designation, we use WC in our work. WC is defined in Equation 2 and calculates a separate mean value for each weight tensor and subtracts it from each weight.

$$W_{j,in,y,x} = \frac{W_{j,in,y,x} - \overline{W}_{j,in,y,x}}{std(W_{j,in,y,x})} \tag{3}$$

The final normalization of the weights is the weight standardization [58] which is defined in Equation 3. Here, as in WC, the mean value is subtracted and each weight of a tensor is also divided by the standard deviation.

3.2. Gradient normalization

In this section, the gradient normalization is introduced. Modern optimizers already use moving averaging with momentum [57, 44]. We also think that the authors exploring gradient centralization [65] already tried different approaches like the standardization. We only present the recently published approach here. Also, as in the weight normalization section, j corresponds to j against the axis along which orthogonal normalization is performed.

$$\Delta W_{j,in,y,x} = \Delta W_{j,in,y,x} - \overline{\Delta W}_{j,in,y,x} \tag{4}$$

As can be seen in Equation 4, the mean value is subtracted from each gradient tensor. The mean value is recalculated for each output layer.

3.3. Data normalization

In this section, we briefly describe the different data normalizations. In our analysis, we only used batch normalization [42]. In this section, j as well as j_1 and j_2 (in case of instance normalization) stand for the axis or plane to which the normalization is orthogonal. Since scal and shift is learned in data manipulation, we denote them with γ and β respectively.

$$F_{s,j,y,x} = \gamma * (\frac{F_{s,j,y,x} - \overline{F}_{s,j,y,x}}{std(F_{s,j,y,x})}) + \beta \tag{5}$$

Equation 5 describes the batch normalization[42]. As mentioned above, γ and β are the scale and shift parameters which are learned during training. Since j is on the second index, each channel has its own average and standard deviation.

$$F_{j,c,y,x} = \gamma * (\frac{F_{j,c,y,x} - \overline{F}_{j,c,y,x}}{std(F_{j,c,y,x})}) + \beta \tag{6}$$

Equation 6 is the layer normalization [1]. Compared to batch normalization [42], layer normalization is the normalization of the samples in a batch. This means that each sample has its own average and standard deviation.

$$F_{j_1,j_2,y,x} = \gamma * (\frac{F_{j_1,j_2,y,x} - \overline{F}_{j_1,j_2,y,x}}{std(F_{j_1,j_2,y,x})}) + \beta \tag{7}$$

In the case of instance normalization [63, 41], each sample is normalized on its own. Equation 7 describes this procedure. It does not normalize along an axis like the other methods, but each sample and each channel separately.

The only approach still missing is group normalization [64]. Here groups are formed between the individual instances, which have their own mean values and standard deviations. Since we cannot simply describe this with our annotation, the equation for the group normalization [64] is not included in this paper.

3.4. Error normalization

Inspired by the data normalization, we have also done some small evaluations regarding error normalization as a separate normalization approach. For this purpose, we evaluated the standardization as well as the simple mean value subtraction. The simple mean subtraction is based on the fact that, in the case of weight normalization, the simple

Table 1. The used naming convention for our evaluation.

Name	WN	WC	WS	GC	BN	LN	IN	EBN	ELN	EB	EL
Eq.	1	2	3	4	5	6	7	8	9	10	11

mean has proven to be very effective. In our simple implementations we did not use the scale and shift (γ, β) parameters and applied the normalization directly.

$$E_{s,j,y,x} = \frac{E_{s,j,y,x} - \overline{E}_{s,j,y,x}}{std(E_{s,j,y,x})} \tag{8}$$

$$E_{j,c,y,x} = \frac{E_{j,c,y,x} - \overline{E}_{j,c,y,x}}{std(E_{j,c,y,x})} \tag{9}$$

The Equations 8 and 9 describe error normalization along the channels and samples. The procedure is the same as for batch normalization [42] and layer normalization [1]. As you can see in the equations, we have omitted the learned γ and β parameters and the remainder of the equations are the same. Thus, the standard deviation and the mean value are calculated in each iteration and normalization is performed by subtracting the mean value and dividing by the standard deviation.

$$E_{s,j,y,x} = E_{s,j,y,x} - \overline{E}_{s,j,y,x} \tag{10}$$

$$E_{j,c,y,x} = E_{j,c,y,x} - \overline{E}_{j,c,y,x} \tag{11}$$

For the two other Equations 10, 11, we calculated only the average value over the samples or the channels and subtracted it. This was recalculated accordingly in each iteration. An overview of the abbreviations used in the rest of the document is shown in Table 1.

One way to include the normalizations is to add them to the back propagation workflow. This is shown in algorithm 1 were each normalization is placed in either the forward, backward, or gradient computation flow. Since batch normalization [42] is a separate layer and learns the scaling and shifting, it was not inserted.

As you can see in Algorithm 1, filter normalization is applied before use in the forward path and gradient normalization is applied immediately after the gradient calculation. This is because the filters must be adjusted first, otherwise the gradient will not match the weights and the weights will have no influence on the forward pass. For gradient normalization, it is applied after the calculation so that the gradients are correctly available for the weight update in the optimizer. In the case of the back propagated error, the error is normalized after the calculation of the back propagation, in which case it would, of course, also be possible to normalize the input error. However, since this is normalized in the previous layer, it is already normalized.

In this paper, we present the combination of GC [65] and WC, whereas WC without the standard deviation (division with the standard deviation) has, to our knowledge, never been published independently. GC [65] and WC can also be integrated into the optimizer itself. In the following, we give two examples: One for SGD [2] with momentum [57] and the other for ADAM [44].

Data: Data,Weights
Result: Output
Function *Forward* **is**
  Weights=Normalize(Weights);
  Output=cuDNNFWD(Weights,Data);
end
Data: ErrorIn,Weights
Result: ErrorOut
Function *Backward* **is**
  ErrorOut=cuDNNBWD(Weights,ErrorIn);
  ErrorOut=Normalize(ErrorOut);
end
Data: ErrorIn,Data
Result: Grad
Function *CompGrad***is**
  Grad=cuDNNCompGrad(Data,ErrorIn);
  Grad=Normalize(Grad);
end

Algorithm 1: Algorithmic description of the function placement for the normalizations. Since batch normalization is implemented as its own layer with learned scaling and shifting, it is not considered in this illustration. It could be placed functionally in the forward propagation and would normalize the output.

Data: $W^t_{j,in,y,x}$,$\Delta W^t_{j,in,y,x}$,α,ψ,$M^t_{j,in,y,x}$
Result: $W^{t+1}_{j,in,y,x}$
Function *SGD* **is**
  $\Delta W^t_{j,in,y,x} = \Delta W^t_{j,in,y,x} - \overline{\Delta W}^t_{j,in,y,x}$;
  $M^t_{j,in,y,x} = \psi * M^t_{j,in,y,x} + (1-\psi) * \Delta W^t_{j,in,y,x}$;
  $W^{t+1}_{j,in,y,x} = W^t_{j,in,y,x} - \alpha * M^t_{j,in,y,x}$;
  $W^{t+1}_{j,in,y,x} = W^{t+1}_{j,in,y,x} - \overline{W}^{t+1}_{j,in,y,x}$;
end

Algorithm 2: Integration of the GC and WC normalization into the stochastic gradient decent optimization with momentum. The variables are weights $W^t_{j,in,y,x}$, gradients $\Delta W^t_{j,in,y,x}$, learning rate α, momentum factor ψ, and momentum $M^t_{j,in,y,x}$. j again is the index of the normalization.

In Algorithm 2, the integration of the GC and the WC normalization in combination with SGD is shown. Here the first line ($\Delta W^t_{j,in,y,x} = \Delta W^t_{j,in,y,x} - \overline{\Delta W}^t_{j,in,y,x}$) is the gradient centralization. Afterwards, the momentum is combined with the gradients by the factor ψ. In the next step,

the weights are adjusted using the learning rate α and the mean value is subtracted from the final weights. This last step is the weight centralization ($W^{t+1}_{j,in,y,x} = W^{t+1}_{j,in,y,x} - \overline{W}^{t+1}_{j,in,y,x}$).

Data: $W^t_{j,in,y,x}$,$\Delta W^t_{j,in,y,x}$,α,ψ_1,ψ_2,$M^t_{j,in,y,x}$,$V^t_{j,in,y,x}$
Result: $W^{t+1}_{j,in,y,x}$
Function *ADAM* **is**

$\Delta W^t_{j,in,y,x} = \Delta W^t_{j,in,y,x} - \overline{\Delta W}^t_{j,in,y,x}$;
$M^t_{j,in,y,x} = \psi_1 * M^t_{j,in,y,x}$;
$M^t_{j,in,y,x} += (1-\psi_1) * \Delta W^t_{j,in,y,x}$;
$V^t_{j,in,y,x} = \psi_2 * V^t_{j,in,y,x}$;
$V^t_{j,in,y,x} += (1-\psi_2) * \Delta W^t_{j,in,y,x} \odot \Delta W^t_{j,in,y,x}$;
$\hat{M}^t_{j,in,y,x} = \frac{M^t_{j,in,y,x}}{1-\psi_1^t}$;
$\hat{V}^t_{j,in,y,x} = \frac{\psi_2 * V^t_{j,in,y,x}}{1-\psi_2^t}$;
$W^{t+1}_{j,in,y,x} = W^t_{j,in,y,x} - \alpha * \frac{\hat{M}^t_{j,in,y,x}}{\sqrt{\hat{V}^t_{j,in,y,x}}+\epsilon}$;
$W^{t+1}_{j,in,y,x} = W^{t+1}_{j,in,y,x} - \overline{W}^{t+1}_{j,in,y,x}$;

end

Algorithm 3: Integration of the GC and WC normalization into the ADAM optimization. The variables are Weights $W^t_{j,in,y,x}$, gradients $\Delta W^t_{j,in,y,x}$, learning rate α, first order momentum factor ψ_1, second order momentum factor ψ_2, first order momentum $M^t_{j,in,y,x}$, and second order momentum $V^t_{j,in,y,x}$. j again is the index of the normalization.

In Algorithm 3 shows the integration of GC and WC in the ADAM optimization. For this, as with SGD, GC is applied first ($\Delta W^t_{j,in,y,x} = \Delta W^t_{j,in,y,x} - \overline{\Delta W}^t_{j,in,y,x}$). Then the new first order momentum is calculated in the following two lines using the factor ψ_1 ($M^t_{j,in,y,x} = \psi_1 * M^t_{j,in,y,x} + (1-\psi_1) * \Delta W^t_{j,in,y,x}$). Subsequently, the second order momentum is calculated with the factor ψ_2 ($V^t_{j,in,y,x} = \psi_2 * V^t_{j,in,y,x} + (1-\psi_2) * \Delta W^t_{j,in,y,x}) \odot \Delta W^t_{j,in,y,x}$). In the penultimate step, the weights are adjusted with the learning rate α as well as the two momentums ($W^{t+1}_{j,in,y,x} = W^t_{j,in,y,x} - \alpha * \frac{\hat{M}^t_{j,in,y,x}}{\sqrt{\hat{V}^t_{j,in,y,x}}+\epsilon}$). The last step is then, again, the weight centralization WC ($W^{t+1}_{j,in,y,x} = W^{t+1}_{j,in,y,x} - \overline{W}^{t+1}_{j,in,y,x}$).

4. Neural Network Models

Figure 1 shows the architectures used in our experimental evaluation. The first model (Figure 1 a)) is a small model with batch normalization. We used this model to show the impact of the different normalization approaches on small models with and without batch normalization. The second model (Figure 1 b)) is a ResNet-34 and a commonly used larger deep neural network. We used it with and without batch normalization during our experiments to show the impact of the normalization approaches on residual networks. The third model (Figure 1 c)) is a classical architecture for neural networks without batch normalization. This model was used to show the impact of the normalization to classical neural network architectures. The last model (Figure 1 d) is a fully convolutional neural network [52]. It uses the U-connections [59] to improve the result for semantic segmentation. We used this network, together with the VOC2012 [5] data set, in the semantic segmentation task to show the impact of the normalizations. For training and evaluation, we used the DLIB [43] library for deep neural networks. In this library we have also integrated our normalization and the state of the art approaches against which we compare our work.

5. Data sets

In this section, we present all used data sets, describe the used training parameters as well as the optimization techniques and the data augmentation. For a simplified reproductibility, we have limited ourselves to a minimum of data manipulation and only used public data sets. The batch size and input resolution, as well as the random weight initialization, are given too.

CIFAR10 [47] has 60,000 colour images each with a resolution of 32×32. The public data set has ten different classes. For training, 50,000 images are provided with 5,000 examples per class. The validation set consists of 10,000 images with 1,000 examples for each class. The task in this data set is to classify a given image to one of the ten categories.

Training: *We used a batch size of 50 and an initial learning rate of* 10^{-3}*. As optimizer, we used ADAM [44] with weight decay of* $5 * 10^{-5}$*, momentum one with* 0.9 *and momentum two with* 0.999*. As random weight initialization we used formula 16 from [35] and all bias terms are set to 0. For data augmentation, we cropped a* 32×32 *region from a* 40×40 *image with zero padding of the original image at the borders. In addition, we used a constant mean subtraction (mean-red* 122.782*, mean-green* 117.001*, mean-blue* 104.298*) and division by* 256.0 *for the input image. The training itself was conducted for 300 epochs whereby the learning rate was decreased by* 10^{-1} *after each 50 epochs.*

CIFAR100 [47] is a more difficult but similar public data set like CIFAR10 and consists of color images each with a resolution of 32×32. The task here, as in CIFAR10, is to classify the given image to one of the one hundred classes provided. The training set consists of 500 examples per class and the validation set has 100 examples per class. This means that CIFAR100 has the same number of images as CIFAR10 for training and validation, but one hundred instead of ten classes.

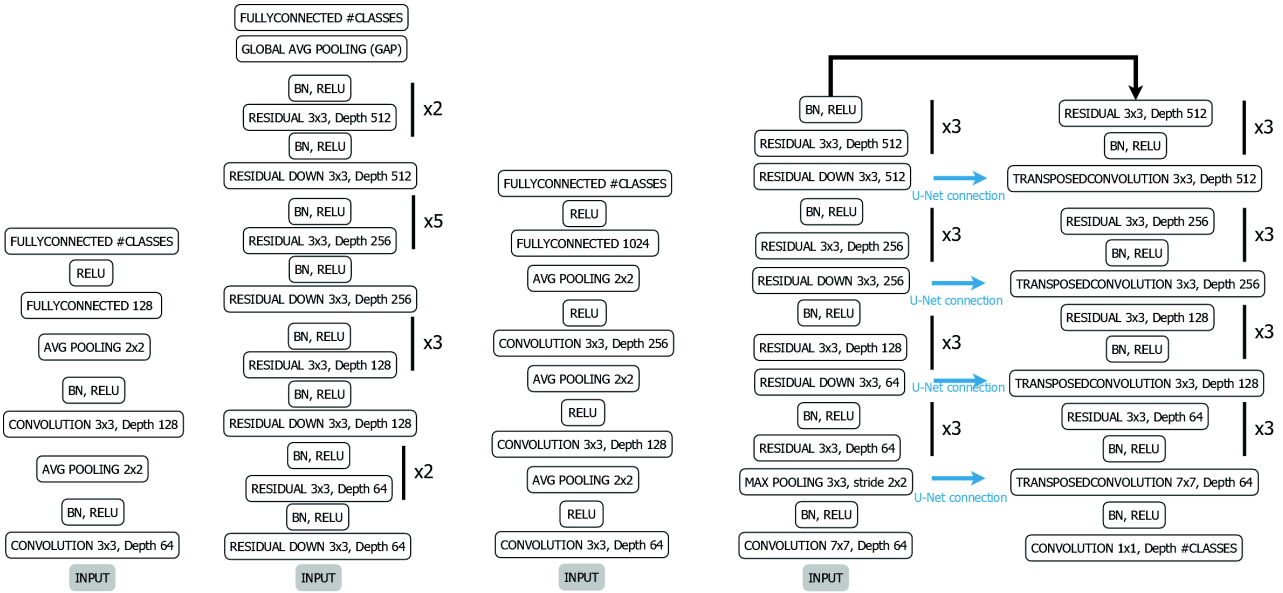

Figure 1. All used architectures in our experimental evaluation. (a) is a small neural network model with batch normalization. (b) is a ResNet-34 architecture. (c) is a small model without batch normalization. (d) is a residual network using the interconnections from U-Net [59] for semantic image segmentation.

Training: *We used a batch size of 50 and an initial learning rate of* 10^{-1}*. As optimizer we used SGD with momentum [57]* 0.9 *and a weight decay of* $(5*10^{-4})$*. For data augmentation, we cropped a* 32×32 *region from a* 40×40 *image with a zero padding of the original image at the borders. In addition, we used a constant mean subtraction (mean-red* 122.782*, mean-green* 117.001*, mean-blue* 104.298*) and division by* 256.0 *for the input image. The training itself was conducted for 300 epochs whereby the learning rate was decreased by* 10^{-1} *after each 50 epochs. For weight initialization we used formula 16 from [35] and all bias terms are set to 0.*

VOC2012 [5] is a publicly available data set which can be used for detection, classification and semantic segmentation. In our experiments we only used the semantic segmentation annotations as well as the semantic segmentation task. For the semantic segmentation task, a class is assigned for each output pixel. This data set has twenty different classes and each image can contain different object classes and different amounts of the same object. This also means that not every object is present in every image. The training set consists of 1,464 images with 3,507 segmented objects and the validation set has 1,449 images with a total of 3,422 segmented objects on it. The number of objects in this data set is not balanced, which increases the challenge. There is also a third data set which does not contain any annotations and can be used initially in an unsupervised fashion to have a good weight initialization. We did not use the third data set in our training nor in our evaluation.

Training: *The initial learning rate was set to* 10^{-1} *with a constant batch size of ten. As optimizer we used SGD with momentum [57] set to* 0.9 *and additionally weight decay of* $1 * 10^{-4}$*. For weight initialization, we used formula 16 from [35] and all bias terms were set to 0. The data was augmented by cropping* 227×227 *regions out of the input image. In addition, we used random color offset and left right flipping of the image. Before the image was processed we subtracted a constant mean (mean-red* 122.782*, mean-green* 117.001*, mean-blue* 104.298*) and divided each value by* 256.0*. We trained each model for 800 epochs and reduced the learning rate by* 10^{-1} *after each 200 epochs.*

6. Evaluation

In this section, we show the results on CIFAR10, CIFAR100 and VOC2012. We use the models from Figure 1 and apply the training parameters and procedures from Section 5. In the first experiment, we show over which areas in the data the mean value subtraction can be used most effectively. In the following three experiments we compare the combination of GC and WC with the state of the art.

Table 2 shows the evaluation of mean subtraction on different areas of weights, gradients and back propagated errors. As can be seen, the mean subtraction on the error propagated back is not very effective because it significantly worsens the generalization of the deep neural network. This shows that error normalization without data normalization,

Table 2. The results for the mean subtraction normalization on CIFAR10 on different target areas for mean computation. The used model was c) from Figure 1.

Reference area	Target	Accuracy
Baseline	non	84.14%
Global	Weight	84.91%
Tensor	Weight (WC)	**85.95%**
Channel	Weight	85.63%
Instance	Weight	84.37%
Global	Gradient	84.01%
Tensor	Gradient (GC [65])	84.89%
Channel	Gradient	84.38%
Instance	Gradient	83.36%
Global	ERROR	79.03%
Sample	ERROR (EL)	72.15%
Channel	ERROR (EB)	75.40%
Instance	ERROR	69.73%

as it happens in batch normalization [42], only brings disadvantages. For the weights and gradients, a convolutional tensor seems to be most effective for normalization. This means that each tensor is used for the mean calculation and this mean is subtracted only from this tensor. It is also clearly seen that weight normalization provides better results independent of gradient normalization with the exception of instance based normalization. In instance based normalization, an average value is calculated for every two dimensional mask and this average value is subtracted from the mask. Based on these results, we decided to define WC on the tensor and to discard the normalization of the back propagated error for further evaluations.

In Table 3, the results on CIFAR10 show different normalizations and the baseline, which is CNN without normalization. As you can see, the combination WC and GC can be effectively applied to all convolutions and also to the penultimate fully connected layer (indicated by the keyword *fully*). This can be seen in model a) and c) from Figure 1. In model b), there is only one fully connected layer in which normalization is not effective because it generates the output. For model a) and b), we have also performed the evaluations with and without batch normalization. As you can see, the combination WC and GC works even better without batch normalization for model a). This is the best result for the model, especially together with normalization in the penultimate fully connected layer. In case of model b), the additional use of batch normalization is much better, because of the residual blocks. Therefore, for the additional evaluations, all residual blocks were evaluated with batch normalization only. Since model c) does not have an integrated batch normalization, we only evaluated without batch normalization.

The combination of WS and GC has not proven to be advantageous for all models, which is why we will not use it in further evaluations. In general, the best normalization across all evaluations on CIFAR10 is the combination of WC and GC. For residual blocks, batch normalization is added. Considering normalizations individually without batch normalization, WC is clearly the best, with GC a close

Table 3. Classification accuracy on the CIFAR10 data set. The first column specifies the methods, the second column the used model from Figure 1, and the third column is the classification accuracy. Models a) and b) where evaluated with and without batch normalization. For the models a) and c) we also used the normalization in the penultimate fully connected layer which is specified with the keyword *fully*.

Method	Model	Accuracy
Baseline	a	81.87%
WN [60, 40] $k = 1$	a	71.74%
WC	a	85.01%
WS [58]	a	73.00%
GC [65]	a	81.42%
WS [58], GC [65]	a	74.51%
WC, GC [65]	a	85.75%
WC, GC [65], fully	a	**87.07%**
Baseline, BN [42]	a	84.67%
WN [60, 40] $k = 1$, BN	a	82.95%
WC, BN [42]	a	85.38%
WS [58], BN [42]	a	79.65%
GC [65], BN [42]	a	84.01%
WS [58], GC [65], BN [42]	a	81.01%
WC, GC [65], BN [42]	a	85.48%
WC, GC [65], BN [42], fully	a	**85.95%**
Baseline	b	88.35%
WN [60, 40] $k = 1$	b	58.77%
WC	b	80.15%
WS [58]	b	nan
GC [65]	b	69.85%
WC, GC [65]	b	**89.61%**
Baseline, BN [42]	b	91.00%
WN [60, 40] $k = 1$, BN [42]	b	61.02%
WC, BN [42]	b	92.50%
WS [58], BN [42]	b	79.83%
GC [65], BN [42]	b	92.01%
WS [58], GC [65], BN [42]	b	79.71%
WC, GC [65], BN [42]	b	**92.68%**
Baseline	c	84.14%
WN [60, 40] $k = 1$	c	83.73%
WC	c	85.95%
WC, fully	c	86.64%
WS [58]	c	10.05%
GC [65]	c	84.89%
GC [65], fully	c	85.37%
WS [58], GC [65]	c	10.72%
WC, GC [65]	c	87.46%
WC, GC [65], fully	c	**87.62%**

Table 4. Classification accuracy on the CIFAR100 data set. The first column specifies the methods, the second column the used model from Figure 1 and the third column is the classification accuracy. Model a) was evaluated with and without batch normalization. For the models a) and c) we also used normalization in the penultimate fully connected layer which is specified with the keyword *fully*.

Method	Model	Accuracy
Baseline	a	46.31%
WN [60, 40] $k = 1$	a	45.93%
WC	a	50.19%
WS [58]	a	41.19%
GC [65]	a	45.55%
WC, GC [65]	a	50.99%
WC, GC [65], fully	a	**52.03%**
Baseline, BN [42]	a	54.04%
WN [60, 40] $k = 1$, BN	a	44.26%
WC, BN [42]	a	55.01%
WS [58], BN [42]	a	48.99%
GC [65], BN [42]	a	53.59%
WC, GC [65], BN [42]	a	56.45%
WC, GC [65], BN [42], fully	a	**56.78%**
Baseline, BN [42]	b	68.99%
WN [60, 40] $k = 1$, BN [42]	b	52.97%
WC, BN [42]	b	69.89%
WS [58], BN [42]	b	63.52%
GC [65], BN [42]	b	69.34%
WC, GC [65], BN [42]	b	**70.24%**
Baseline	c	46.31%
WN [60, 40] $k = 1$	c	10.31%
WC	c	52.05%
WS [58]	c	34.65%
GC [65]	c	47.95%
WC, GC [65]	c	53.16%
WC, GC [65],fully	c	**53.90%**

second.

Table 4 shows the results of models a), b), and c) of Figure 1 on the CIFAR100 data set. As you can see, again the combination WC and GC is the most effective. As with CIFAR10 (Table 3), this applies in particular to the additional use of normalization in the last fully connected layer (Indicated by the keyword *fully*). Like CIFAR10 (Table **??**), the normalization WC always delivers better results in comparison to GC, if both normalizations are evaluated alone. However, there is a difference in the batch normalization for model a). The additional batch normalization is much more effective than model a) is without batch normalization. In all evaluations in Tables 3 and 4 one also sees that the normalizations WS and WN have worsened the generalization of the model. In one case, WS even led to a NaN result.

Table 5 shows the evaluation of different normalization methods on the VOC2012 data set with model d) from Figure 1. Normalization was used in all layers except the final convolution before output. As you can see, both GC and WC improve the result significantly. In combination with the batch normalization, the result is improved by more than 3%. This clearly shows that the combination of WC and GC can be used very effectively together with batch normalization for residual blocks. For the methods WS and WN, however, the generalization of the deep neural network is even worse.

Table 5. The average pixel accuracy classification results for different normalization methods on the VOC2012 validation set using model d) from Figure 1. We applied the normalization specified in column one to all layers except for the last convolution.

Method	Average Pixel Accuracy
Baseline, BN [42]	85.15%
WS [58], BN [42]	81.23%
WN [60, 40] $k = 1$	75.76%
GC [65], BN [42]	85.91%
WC, BN [42]	86.92%
WC, GC [65], BN [42]	**88.98%**

7. Limitations

A disadvantage of WC and GC is that for residual blocks without batch normalization the results are also poor. This can be seen in Table 3 for model b) from Figure 1. Here you can see in the evaluations that for both, as a single normalization without batch normalization, the results are significantly worse. In combination, however, they work better than the model without normalization and without batch normalization. An advantage of the combination of WC and GC compared to batch normalization is that they only need to be used in training (see Algorithm 2 and 3). For batch normalization, however, it is necessary to apply the mean subtraction, division by the standard deviation, scaling, and shift at runtime. However, since this can be calculated with a complexity linear to the input, it hardly affects the runtime.

8. Conclusion

In this work, we have shown that weight centralization is a very effective normalization method. Together with gradient centralization and, for residual networks, batch normalization, this combination exceeds the state of the art. We have also shown over which area mean subtraction is most effective. Our results were generated with four different nets on three public data sets and clearly show that the additional use of weight centralization is effective and improves the generalization of deep neural networks. Further research will evaluate the applicability of the weight and gradient normalization in the fields of gaze behaviour analysis [21, 23, 34, 24] which includes eye movement seg-

mentation [25, 32, 9, 10, 18, 25] and scan path classification [8, 11].

References

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[2] Léon Bottou. Stochastic gradient learning in neural networks. *Proceedings of Neuro-Nımes*, 91(8):12, 1991.

[3] Minhyung Cho and Jaehyung Lee. Riemannian approach to batch normalization. In *Advances in Neural Information Processing Systems*, pages 5225–5235, 2017.

[4] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

[5] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

[6] W. Fuhl. *Image-based extraction of eye features for robust eye tracking*. PhD thesis, University of Tbingen, 04 2019.

[7] Wolfgang Fuhl. From perception to action using observed actions to learn gestures. *User Modeling and User-Adapted Interaction*, pages 1–18, 08 2020.

[8] Wolfgang Fuhl, Efe Bozkir, Benedikt Hosp, Nora Castner, David Geisler, Thiago C Santini, and Enkelejda Kasneci. Encodji: encoding gaze data into emoji space for an amusing scanpath classification approach. In *Proceedings of the 11th ACM Symposium on Eye Tracking Research & Applications*, pages 1–4, 2019.

[9] W. Fuhl, N. Castner, and E. Kasneci. Histogram of oriented velocities for eye movement detection. In *International Conference on Multimodal Interaction Workshops, ICMIW*, 2018.

[10] W. Fuhl, N. Castner, and E. Kasneci. Rule based learning for eye movement type detection. In *International Conference on Multimodal Interaction Workshops, ICMIW*, 2018.

[11] W. Fuhl, N. Castner, T. C. Kbler, A. Lotz, W. Rosenstiel, and E. Kasneci. Ferns for area of interest free scanpath classification. In *Proceedings of the 2019 ACM Symposium on Eye Tracking Research & Applications (ETRA)*, 06 2019.

[12] W. Fuhl, N. Castner, L. Zhuang, M. Holzer, W. Rosenstiel, and E. Kasneci. Mam: Transfer learning for fully automatic video annotation and specialized detector creation. In *International Conference on Computer Vision Workshops, ICCVW*, 2018.

[13] W. Fuhl, S. Eivazi, B. Hosp, A. Eivazi, W. Rosenstiel, and E. Kasneci. Bore: Boosted-oriented edge optimization for robust, real time remote pupil center detection. In *Eye Tracking Research and Applications, ETRA*, 2018.

[14] W. Fuhl, H. Gao, and E. Kasneci. Neural networks for optical vector and eye ball parameter estimation. In *ACM Symposium on Eye Tracking Research & Applications, ETRA 2020*. ACM, 01 2020.

[15] W. Fuhl, H. Gao, and E. Kasneci. Tiny convolution, decision tree, and binary neuronal networks for robust and real time pupil outline estimation. In *ACM Symposium on Eye Tracking Research & Applications, ETRA 2020*. ACM, 01 2020.

[16] W. Fuhl, D. Geisler, W. Rosenstiel, and E. Kasneci. The applicability of cycle gans for pupil and eyelid segmentation, data generation and image refinement. In *International Conference on Computer Vision Workshops, ICCVW*, 11 2019.

[17] W. Fuhl, D. Geisler, T. Santini, T. Appel, W. Rosenstiel, and E. Kasneci. Cbf:circular binary features for robust and real-time pupil center detection. In *ACM Symposium on Eye Tracking Research & Applications*, 06 2018.

[18] W. Fuhl and E. Kasneci. Eye movement velocity and gaze data generator for evaluation, robustness testing and assess of eye tracking software and visualization tools. In *Poster at Egocentric Perception, Interaction and Computing, EPIC*, 2018.

[19] W. Fuhl and E. Kasneci. Learning to validate the quality of detected landmarks. In *International Conference on Machine Vision, ICMV*, 11 2019.

[20] W. Fuhl, G. Kasneci, W. Rosenstiel, and E. Kasneci. Training decision trees as replacement for convolution layers. In *Conference on Artificial Intelligence, AAAI*, 02 2020.

[21] W. Fuhl, T. C. Kbler, H. Brinkmann, R. Rosenberg, W. Rosenstiel, and E. Kasneci. Region of interest generation algorithms for eye tracking data. In *Third Workshop on Eye Tracking and Visualization (ETVIS), in conjunction with ACM ETRA*, 06 2018.

[22] W. Fuhl, T. C. Kbler, D. Hospach, O. Bringmann, W. Rosenstiel, and E. Kasneci. Ways of improving the precision of eye tracking data: Controlling the influence of dirt and dust on pupil detection. *Journal of Eye Movement Research*, 10(3), 05 2017.

[23] W. Fuhl, T. C. Kbler, K. Sippel, W. Rosenstiel, and E. Kasneci. Arbitrarily shaped areas of interest based on gaze density gradient. In *European Conference on Eye Movements, ECEM 2015*, 08 2015.

[24] Wolfgang Fuhl, Thomas C Kübler, Thiago Santini, and Enkelejda Kasneci. Automatic generation of saliency-based areas of interest for the visualization and analysis of eye-tracking data. In *VMV*, pages 47–54, 2018.

[25] Wolfgang Fuhl, Yao Rong, and Kasneci Enkelejda. Fully convolutional neural networks for raw eye tracking data segmentation, generation, and reconstruction. In *Proceedings of the International Conference on Pattern Recognition*, pages 0–0, 2020.

[26] Wolfgang Fuhl, Yao Rong, Thomas Motz, Michael Scheidt, Andreas Hartel, Andreas Koch, and Enkelejda Kasneci. Explainable online validation of machine learning models for practical applications. In *Proceedings of the International Conference on Pattern Recognition*, pages 0–0, 2020.

[27] W. Fuhl, W. Rosenstiel, and E. Kasneci. 500,000 images closer to eyelid and pupil segmentation. In *Computer Analysis of Images and Patterns, CAIP*, 11 2019.

[28] W. Fuhl, T. Santini, D. Geisler, T. C. Kbler, and E. Kasneci. Eyelad: Remote eye tracking image labeling tool. In *12th Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP 2017)*, 02 2017.

[29] W. Fuhl, T. Santini, D. Geisler, T. C. Kbler, W. Rosenstiel, and E. Kasneci. Eyes wide open? eyelid location and eye aperture estimation for pervasive eye tracking in real-world scenarios. In *ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct publication – PETMEI 2016*, 09 2016.

[30] W. Fuhl, T. Santini, and E. Kasneci. Fast and robust eyelid outline and aperture detection in real-world scenarios. In *IEEE Winter Conference on Applications of Computer Vision (WACV 2017)*, 03 2017.

[31] W. Fuhl, T. Santini, and E. Kasneci. Fast camera focus estimation for gaze-based focus control. In *CoRR*, 2017.

[32] W. Fuhl, T. Santini, T. Kuebler, N. Castner, W. Rosenstiel, and E. Kasneci. Eye movement simulation and detector creation to reduce laborious parameter adjustments. *arXiv preprint arXiv:1804.00970*, 2018.

[33] W. Fuhl, T. Santini, C. Reichert, D. Claus, A. Herkommer, H. Bahmani, K. Rifai, S. Wahl, and E. Kasneci. Non-intrusive practitioner pupil detection for unmodified microscope oculars. *Elsevier Computers in Biology and Medicine*, 79:36–44, 12 2016.

[34] D. Geisler, W. Fuhl, T. Santini, and E. Kasneci. Saliency sandbox: Bottom-up saliency framework. In *12th Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP 2017)*, 02 2017.

[35] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[36] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[37] Harshit Gupta, Kyong Hwan Jin, Ha Q Nguyen, Michael T McCann, and Michael Unser. Cnn-based projected gradient descent for consistent ct image reconstruction. *IEEE transactions on medical imaging*, 37(6):1440–1453, 2018.

[38] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[39] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[40] Lei Huang, Xianglong Liu, Yang Liu, Bo Lang, and Dacheng Tao. Centered weight normalization in accelerating training of deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2803–2811, 2017.

[41] Xun Huang and Serge Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1501–1510, 2017.

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

[43] Davis E King. Dlib-ml: A machine learning toolkit. *Journal of Machine Learning Research*, 10(Jul):1755–1758, 2009.

[44] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[45] Durk P Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. In *Advances in neural information processing systems*, pages 2575–2583, 2015.

[46] David Kirk et al. Nvidia cuda software and gpu parallel computing architecture. In *ISMM*, volume 7, pages 103–104, 2007.

[47] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[48] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[49] Måns Larsson, Anurag Arnab, Fredrik Kahl, Shuai Zheng, and Philip Torr. A projected gradient descent method for crf inference allowing end-to-end training of arbitrary pairwise potentials. In *International Workshop on Energy Minimization Methods in Computer Vision and Pattern Recognition*, pages 564–579. Springer, 2017.

[50] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[51] K Levenberg. A method for the solution of certain problems in least squares, 1944.

[52] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[53] Donald W Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the society for Industrial and Applied Mathematics*, 11(2):431–441, 1963.

[54] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010.

[55] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. Understanding the exploding gradient problem. *CoRR, abs/1211.5063*, 2:417, 2012.

[56] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318, 2013.

[57] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural networks*, 12(1):145–151, 1999.

[58] Siyuan Qiao, Huiyu Wang, Chenxi Liu, Wei Shen, and Alan Yuille. Weight standardization. *arXiv preprint arXiv:1903.10520*, 2019.

[59] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[60] Tim Salimans and Durk P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in neural information processing systems*, pages 901–909, 2016.

[61] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? In *Advances in Neural Information Processing Systems*, pages 2483–2493, 2018.

[62] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*, 45(11):2673–2681, 1997.

[63] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016.

[64] Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19, 2018.

[65] Hongwei Yong, Jianqiang Huang, Xiansheng Hua, and Lei Zhang. Gradient centralization: A new optimization technique for deep neural networks. *arXiv preprint arXiv:2004.01461*, 2020.