Supplementary Material for: Weight Conditioning for Smooth Optimization of Neural Networks

Hemanth Saratchandran[1], Thomas X Wang[2], and Simon Lucey[1]

[1] Australian Institute for Machine Learning, University of Adelaide
[2] Sorbonne Université, CNRS, ISIR
hemanth.saratchandran@adelaide.edu.au

In this supplementary material accompanying our paper, we give details into the concepts and methods outlined in the main paper. Initially, we present a detailed proof of Thm. 2, as introduced in Sec. 5.1, offering a thorough mathematical foundation. Following this, we elucidate the application of our novel weight conditioning technique to convolutional layers, providing a bridge between theory and practice. The subsequent section is dedicated to an expanded experimental analysis, extending the work discussed in Sec. 6 of the paper. Here, we elaborate on the experimental frameworks and methodologies employed in our study. Concluding these materials, we explore additional applications of our method to 3D shape modeling and modern Convolutional Neural Network (CNN) architectures. Additionally, we present a cost-benefit analysis of our approach, complemented by a series of ablation studies, thereby underscoring the efficacy and efficiency of our proposed weight conditioning method.

1 Theoretical Analysis

1.1 Feed Forward Layers

In this section, we give the proof of Thm. 2 from Sec. 5.1 of the main paper.

We fix a loss function $\mathcal{L}$ that we use as an objective function to train a neural network F. The loss functions we deal with will be of the following form: We start by fixing a L-layer neural network F as defined in Sec. 3 of the main paper. Recall that the neural network F has layers F_k for $1 \leq k \leq L$ and each F_k has parameters given by the weights and biases: $\{(W_1, b_1), \cdots, (W_k, b_k)\}$. Let us assume that each pair (W_l, b_l) for $1 \leq l \leq k$ defines a vector in $\mathbb{R}^{n_l}$ so that each F_k has $n_1 + \cdots + n_l$ parameters. Let us further assume that the output dimension of the network is $\mathbb{R}^o$ and the input dimension is $\mathbb{R}^{n_0}$ so that the network can be veiwed as a map:

$$F : \mathbb{R}^{n_0} \times \mathbb{R}^{n_1} \times \cdots \times \mathbb{R}^{n_L} \to \mathbb{R}^o. \tag{1}$$

Assume we have N data points $X \in \mathbb{R}^{N \times n_0}$ and we fix our labels as a target matrix $Y \in \mathbb{R}^{N \times o}$.

We fix a convex cost function $c : \mathbb{R}^o \times \mathbb{R}^o \to \mathbb{R}$. Examples of convex cost functions that are used in machine learning are the quadratic cost function used in the MSE loss function and the cross entropy cost used for Binary Cross Entropy

(BCE). The loss functions we will be dealing with can be written as

$$\mathcal{L}(\theta) = \frac{1}{N}\sum_{i=1}^{N} c(F(X_i;\theta), Y_i). \tag{2}$$

Note that almost all loss function used to train neural networks are given by the general form (2). Examples are, the MSE loss function and the Binary Cross Entropy (BCE) loss function.

Given an L-layer neural network F and a loss function $\mathcal{L}$, as defined by Eqn. (2), we can write the Hessian of $\mathcal{L}$, $H(\mathcal{L}$, as

$$H(\mathcal{L}) = (DF)^T H(c) DF + (Dc)H(F) \tag{3}$$

where D denotes the derivative with respect to parameters θ.

We thus see that the Hessian $H(\mathcal{L})$ is determined by two terms $(DF)^T H(c)DF$ and $(Dc)H(F)$. The term $(DF)^T H(c)DF$ is known as the Gauss-Newton matrix associated to $H(\mathcal{L})$. In [11] it was shown that the singular values of the Hessian can be approximated by the singular values of the Gauss-Newton matrix. Therefore, we will approximate the Hessian via the Gauss-Newton matrix $(DF)^T H(c)DF$. Note that in our assumption of Thm. 2 from Sec. 5.1 of the main paper, we assume $H(\mathcal{L})$ has full rank which implies DF has full rank.

Note that the condition number satisfies

$$\kappa(AB) \leq \kappa(A)\kappa(B). \tag{4}$$

Therefore, we have that $\kappa((DF)^T H(c)DF) \leq \kappa((DF)^T)\kappa(H(c))\kappa(DF)$. Furthermore, since DF and $(DF)^T$ have the same singular value set in the SVD decomposition we have that $\kappa(DF) = \kappa((DF)^T)$ implying that

$$\kappa((DF)^T H(c)DF) \leq \kappa(H(c))\kappa(DF)^2. \tag{5}$$

Using the chain rule we have

$$DF = \sum_{k=1}^{L} JF_L \cdots JF_{k+1} DF_k \tag{6}$$

where JF_i denotes the Jacobian of the ith layer F_i w.r.t inputs, and by the chain rule again we have

$$JF_i = diag(\phi'(W_i^T F_{i-1} + b_i)) \cdot W_i \tag{7}$$

where $diag(\phi'(W_i^T F_{i-1} + b_i))$ denotes the diagonal matrix with diagonal entries $\phi'(W_i^T F_{i-1} + b_i)$. Observe that $\kappa(diag(\phi'(W_i^T F_{i-1} + b_i))) = 1$ and since $diag(\phi'(W_i^T F_{i-1}+b_i))$ is a diagonal matrix with the same entries on the diagonal we have that

$$\kappa(JF_i) = \kappa(diag(\phi'(W_i^T F_{i-1} + b_i)))\kappa(W_i) = \kappa(W_i). \tag{8}$$

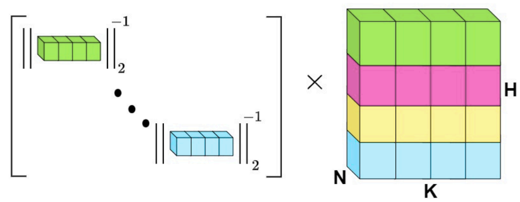


Fig. 1: Schematic diagram showing row equilibrated preconditioning of a kernel filter for a convolutional network. The filter is represented by a 3-tensor of dimensions (N, H, K). Each (H, K) matrix is row equilibrated giving N such row equilibrations (left) defining a 3-tensor with dimensions (N, H, H).

Proof (of Thm. 2 from Sec. 5.1 of main paper). Using Eqn. (6) we have

$$\kappa(DF) \leq \sum_{k=1}^{L} \kappa(W_L) \cdots \kappa(W_{k+1}) \kappa(DF_k). \tag{9}$$

We therefore see that in order to bring down $\kappa(DF)$ it suffices to bring down each of the $\kappa(W_i)$. By Van Der Sluis' theorem, see Thm. 1 in Sec. 5.1 of the main paper, we can bring down $\kappa(W_i)$ by applying an equilibrated preconditioner E_i to W_i. This is precisely what equilibrated weight conditioning does. If we let F^{eq} denote the equilibrated weight conditioned network. Then we have

$$\kappa(JF_i^{eq}) \leq \kappa(JF_i) \tag{10}$$

by definition of F^{eq}. It follows that

$$\kappa(DF^{eq}) \leq \kappa(DF) \tag{11}$$

from which the theorem follows.

1.2 Weight conditioning for convolutional layers

Weight conditioning can also be extended to the case of convolutional architectures. Convolutional neural networks have three main types of layers, feedforward layers, convolutional layers and max pooling layers. Sec. 5.1 from the main paper shows how to weight condition a feedforward layer. We therefore focus on the case of a convolutional. The case of a maxpooling layer follows similarly.

A convolutional layer has as its building block a collection of kernels that perform the convolutional operator. Such a component is given by a 3 tensor of the form (N, H, K) where N denotes the number of kernels and each kernel is $H \times K$. In this setting we apply weight conditioning to each of the N kernels treating each $H \times K$ kernel as matrix and applying row equilibration to it as we did for a feedforward layer. Fig. 1 gives a schematic representation of row equilibration for a kernel.

2 Experiments

2.1 Convolutional Neural Networks (CNNs)

In this section we go through the experimental setup we used for the experiments in Sec. 6.1 of the main paper. We also give the results for training convnets on ImageNet1k with with weight conditioning.

For the experiments, we tested four differential normalization schemes:

1. Batch Normalization (BN) [8].
2. Batch Normalization with Weight Standardization (BN + WS) [12] .
3. Batch Normalization with Weight Normalization (BN + W) [13].
4. Batch Normalization with Weight Conditioning using an equilibrated preconditioner (BN + E).

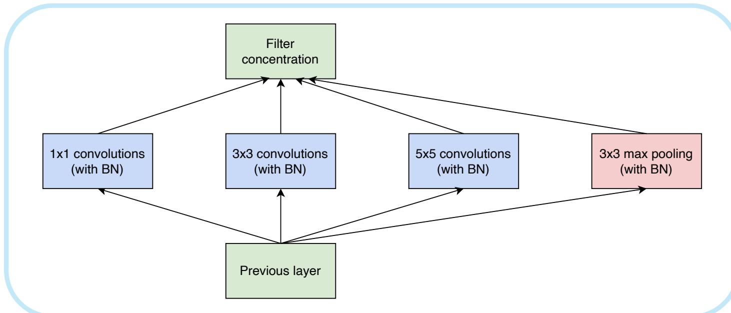
In our study, we chose to evaluate our weight conditioning technique against three predominant normalization methods cited extensively in existing literature. Notably, we did not incorporate Group Normalization, an important normalization introduced by [18], in our analysis. Despite its occasional preference over Batch Normalization (BN) and Layer Normalization (LN) in certain contexts, our empirical findings indicated superior performance of BN for Convolutional Neural Networks (CNNs) and LN for Transformers. This informed our decision to benchmark our method against these two normalization techniques.

Moreover, the large depth of CNN architectures necessitated the use of BN to ensure effective backpropagation of gradients. We found that if we removed BN and tried to compare our technique with other normalizations, training would consistently diverge. Consequently, our evaluation framework was designed to assess all normalization strategies in conjunction with BN, aiming to explore their impact comprehensively and ascertain the most conducive normalization approach for enhancing model performance.

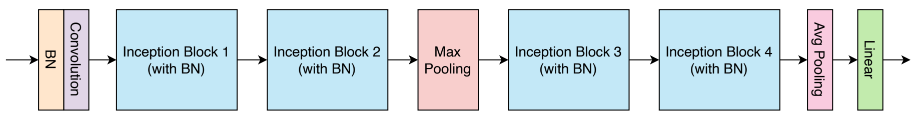
Inception: We tested all four normalizations on the Inception model [15], see Sec. 6.1 of the main paper.

Batch Normalized Inception has as its building block the Inception block shown in Fig. 2a The block consists of four different convolution filters each having batch norm applied to it. These blocks are then stacked together with a final a linear layer. Fig. 2b gives an example network applying inception blocks. The first convolution layer and all the inception blocks have batch norm applied. More details about the architecture can be found in the original paper [15].

Inception with weight standardization uses the same building block as the original inception block but now has weight standardization applied to each convolutional layer. We apply weights standardization to each convolutional layer making up the network. We found that adding weight standardization to the linear layer did not add any value to the network in terms of accuracy or train loss so for the experiments Fig. 3 gives the details of where weight standardization is applied.



(a) Standard Inception block with batch normalization (BN) as used in [15].



(b) Example of an inception network using four inception blocks. A max pooling layer is used between every pair of Inception blocks.

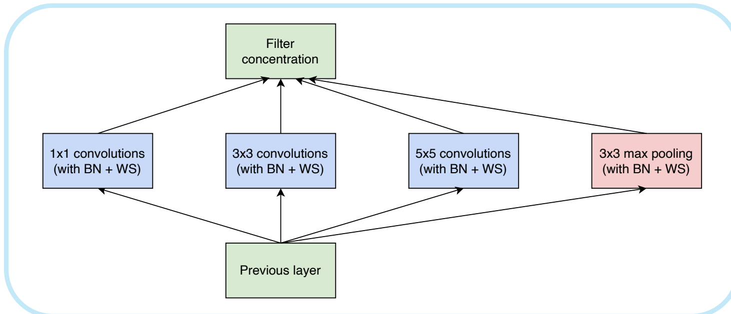
Fig. 2: Inception network using Batch Normalization (BN).

Inception with weight normalization uses the same building block as the original inception block but now has weight normalization applied to each convolutional layer and the final linear layer. Fig. 4 gives the details of where weight normalization is applied. With weight normalization we found that one could also apply weight normalization to the linear layer only and this yielded very similar results often doing better than applying weight normalization to every layer. Fig. 5 gives a diagrammatic picture of this case.

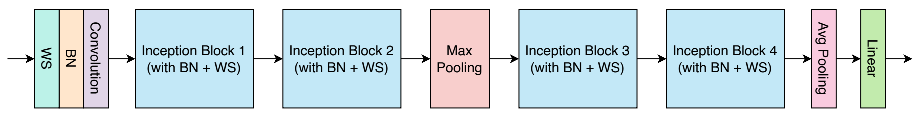
Inception with equilibrated weight conditioning uses the same building block as the original inception block but now has weight conditioning applied to each convolutional layer and the final linear layer. Fig. 6 gives the details of where weight conditioning is applied. We found that applying weight conditioning to the first convolutional layer and the last linear layer led to better results for the experiments. Given the cost of adding equilibrated weight conditioning to a layer, see Sec. 3.3, for the experiments we used the setup outlined in Fig. 7.

Inception on CIFAR10 and CIFAR100: The results for the CIFAR10 and CIFAR100 datasets were shown in Sec. 6.1 of the main paper. For these two datasets we used an Inception network with 8 inception blocks in total. The hyperparameters used for training on these datasets is showin Tab. 1. We also tested GoogLeNet from [15] with a large learning rate of 2 trained under SGD. The results of this were shown in Fig. 1 in the main paper.

Inception on ImageNet1k: For the ImageNet1k dataset we trained the GoogLeNet model of Inception from [15]. Fig. 8 shows the train loss and Top-1% accuracy.



(a) Standard Inception block with weight standardization (WS) in each convolution.



(b) Example network using weight standardization (WS). The first convolution layer also has weight standardization applied.

Fig. 3: Inception network using weight standardization (WS).

Table 1: Inception model hyperparameters for CIFAR10 AND CIFAR100

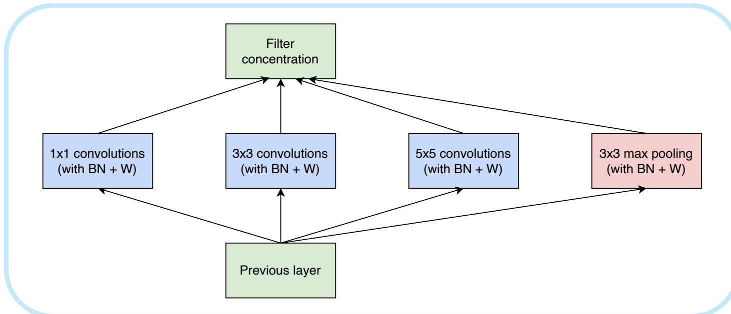
Hyperparameters	**Value**
Optimizer	SGD
Batch Size	128
Learning Rate	1e-3
Epochs	200

As can be seen from the figure, the equilibrated weight conditioned network achieves the lowest loss and highest accuracy.

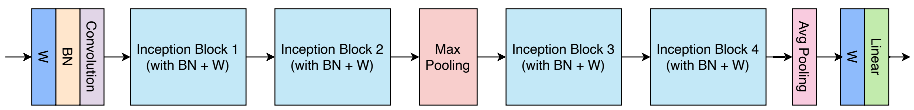
DenseNet: We tested all four normalizations on the DenseNet architecture from [7], see Sec. 6.1 of the paper. We applied the four normalizations in the following way:

Batch Normalized DenseNet (vanilla DenseNet): has as its building block a Dense block that is connected by a transition layer. Each dense block applies stacks of dense layers consisting of two 3x3 convolutions with differing input and output channels with batch normalization. Each such block is then connected to another DenseBlock by a transition layer which consists of another convolutional layer with an average pooling layer. Full details can be found in [7]. Fig. 9 gives a schematic representation of basic DenseNet consisting of two DenseBlocks.

Weight Standardized DenseNet: applies weight standardization to the first convolutional layer in the network, each DenseBlock making up the network,



(a) Inception block with weight normalization (W) in each convolution.



(b) Example network using weight normalization (W). The first convolution layer as well as the last linear layer also has weight normalization applied.

Fig. 4: Inception network using weight normalization (W).

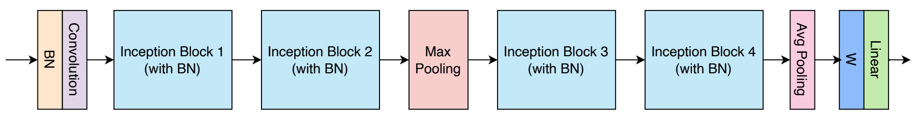


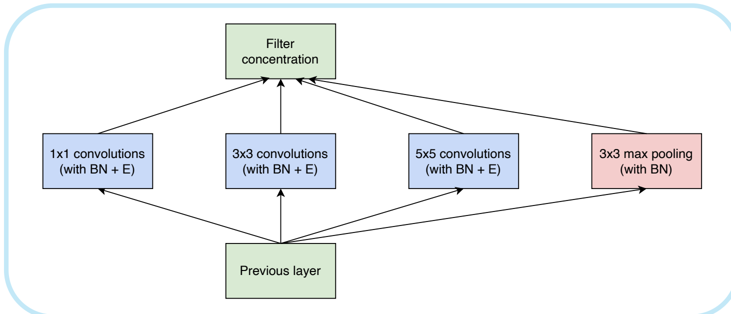
Fig. 5: An Inception network that uses weight normalization (W) on only the last linear layer.

to the convolutional layer in the transition layer, and to the final average pooling layer. Fig. 10 gives a diagram of how weight standardization is applied.

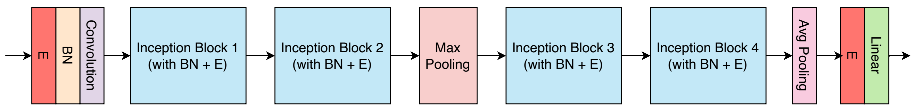
Weight Normalized DenseNet: applies weight normalization to the first convolutional layer in the network, each DenseBlock making up the network, to the convolutional layer in the transition layer, and to the final linear layer. We found that it was enough to apply weight normalization to the last linear layer as this often yielded better results. Fig. 11 gives a diagram of how weight normalization is applied.

Equilibrated weight conditioned DenseNet: applies weight conditioning to the first convolutional layer in the network, each DenseBlock making up the network, to the convolutional layer in the transition layer, and to the final linear layer. We found that it was enough to apply weight conditioning to the first convolutional layer and the last linear layer as this often yielded better results. Fig. 12 gives a diagram of how weight conditioning is applied.

DenseNet on CIFAR10 and CIFAR100: The results for CIFAR10 and CIFAR100 datasets were shown in Sec. 6.1 of the main paper. For these datasets we followed the architecture used in [7]. We used a total of three dense blocks,



(a) Inception block with weight conditioning (E) in each convolution. We don't apply weight conditioning to the max pooling layer.



(b) Example network using weight conditioning (E). The first convolution layer as well as the last linear layer also has weight conditioning applied.

Fig. 6: Inception network using weight conditioning (E).

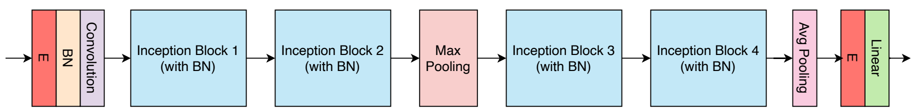


Fig. 7: An Inception network with weight conditioning (E) applied to the first convolutional layer and the last linear layer only.

with a growth rate of 20 and total number of layers $= 40$, see [7] for a detailed definition for growth rate. Tab. 2 shows the hyperparamters we used for training.

Table 2: DenseNet model hyperparameters for CIFAR10 AND CIFAR100

Hyperparameters	**Value**
Optimizer	SGD
Batch Size	128
Learning Rate	1e-3
Epochs	200

DenseNet on ImageNet1k: We trained each normalized DenseNet on the ImageNet1k dataset. For this we used the DenseNet-BC structure from the original paper [7] which consisted of four dense blocks. We employed the training regime used in [7]. Fig. 13 shows the results of the experiment. Equilibrated performs slightly better than all other normalizations.

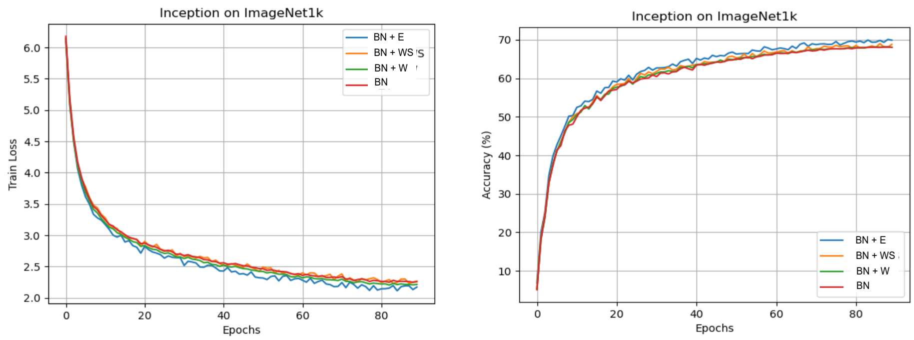


Fig. 8: Left: Train loss for the GoogLeNet Inception model trained on ImageNet1k. The equilibrated weight conditioned network achieves the lowest loss. Right: Top-1% accuracy on ImageNet1k dataset. Equilibrated weight conditioned network achieves the highest accuracy.

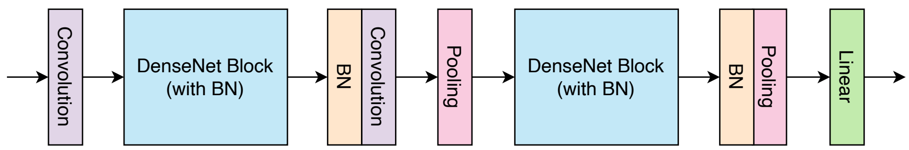


Fig. 9: DenseNet with Batch Normalization (BN): basic structure of a DenseNet with two DenseBlocks connected by a transition layer consisting of a convolutional layer with an average pooling layer.

ResNet: We tested all four normalizations on the ResNet architecture [6]. **ResNet with batch normalization:** has as its building block a ResNet block with two convolutional layers. Batch norm is applied to each of the convolutional layers. A ResNet then consists of stacking such blocks together, with a first convolutional layer wtih batch norm as the first part of the network and the final part of the network being an average pooling layer together with a linear layer. Fig. 14 gives a breakdown of the ResNet block and a ResNet architecture with two ResNet blocks.

ResNet with weight standardization: We apply weight standardization to the convolutional layers in the ResNet block as well as the first convolutional layer making up the first part of a ResNet architecture. We found that there was no need to add weight standardization to the final linear layer as it did not seem to help the network. Fig. 15 shows how this is done.

ResNet with weight normalization: We apply weight normalization to the convolutional layers in the ResNet block as well as the first convolutional layer making up the first part of a ResNet architecture and the final linear layer making up the last part of a ResNet architecture. Fig. 16 shows how this is done. For the experiments we found that if we just applied weight normalization to the last linear layer we could obtain equivalent or better results. Thus in the experiments we used weight normalization on only the final linear layer as this

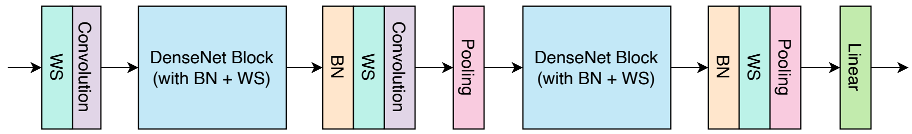
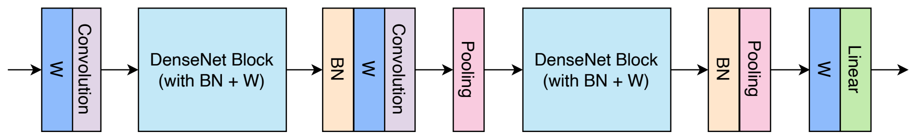
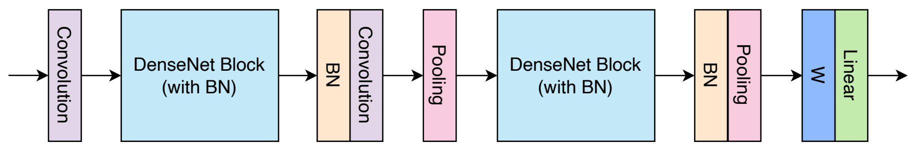


Fig. 10: Basic structure of a DenseNet with Weight Standardization (WS) with two DenseBlocks connected by a transition layer consisting of a convolutional layer with an average pooling layer. Weight standardization is applied to the convolutional layers and the pooling layers.



(a) Weight normalization is applied to all convolutional layers including the ones in the DenseBlock and the last linear layer.



(b) Weight normalization can also only be applied to the last linear layer.

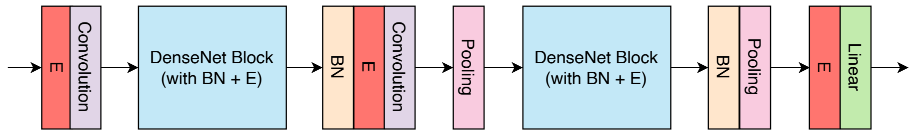
Fig. 11: Basic structure of a DenseNet using weight normalization (W) with two DenseBlocks.

was the most memory efficient for training. Fig. 17 gives the basic architecture that was used for the experiments.

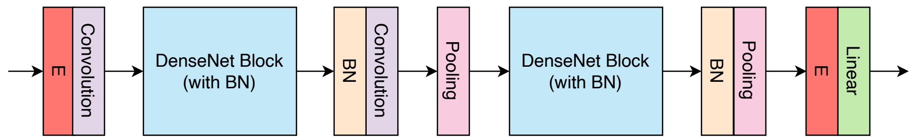
ResNet with equilibrated weight conditioning: We apply equilibrated weight conditioning to the convolutional layers in the ResNet block as well as the first convolutional layer making up the first part of a ResNet architecture and the final linear layer making up the last part of a ResNet architecture. Fig. 18 shows how this is done. For the experiments we found that if we just applied equilibrated weight conditioning to the first convolutional layer and the last linear layer we could obtain equivalent or better results. Thus in the experiments we did this. Fig. 19 gives the basic architecture that was used for the experiments.

ResNet on CIFAR100: We trained a ResNet18 and a ResNet50 [6] on the CIFAR100 dataset. The results for this were shown in Fig. 1 in the main paper. The ResNet18 used a total of 18 layers [6] and the ResNet50 a total of 50 layers [6]. Tab. 3 shows the hyperparameters used for training.

ResNet on ImageNet1k: We trained a ResNet50 on the ImageNet1k dataset following the recipe in the original paper [6]. Fig. 20 shows the train loss curves



(a) Equilibrated weight conditioning is applied to all convolutional layers including the ones in the DenseBlock and the last linear layer



(b) Equilibrated weight conditioning is only applied to the first convolutional layer and the last linear layer.

Fig. 12: Basic structure of a DenseNet using weight conditioning (E) with two DenseBlocks.

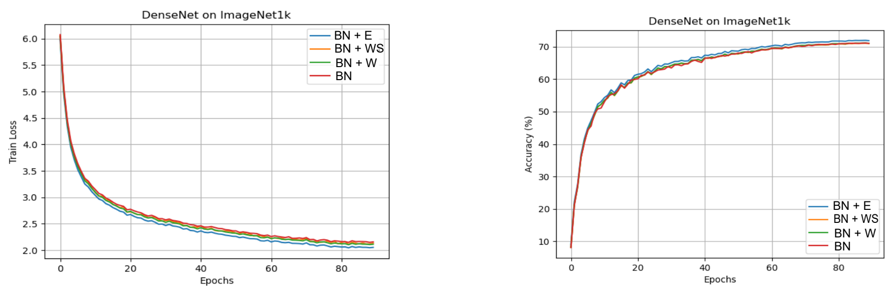


Fig. 13: Left: Train loss for the DenseNet model trained on ImageNet1k. The equilibrated weight conditioned network achieves the lowest loss. Right: Top-1% accuracy on ImageNet1k dataset. Equilibrated weight conditioned network achieves the highest accuracy.

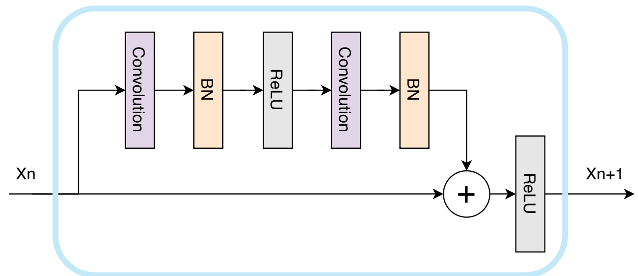
and the Top-1% accuracy. The equilibrated weight conditioned network does better than the others.

2.2 Vision Transformers (ViTs)

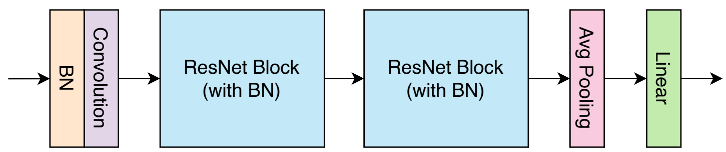
In this section we go through the experimental setup we used for the experiments in Sec. 6.2 of the main paper. We also give results for training smaller ViTs with weight conditioning and compare these against normalizations used in the literature.

For the experiments we tested three normalizations:

1. Layer Normalization (LN) [2]
2. Layer Normalization with Weight Normalization (LN + W).
3. Layer Normalization with Equilibrated weight conditioning (LN + E).



(a) ResNet block from [6]



(b) A ResNet with two ResNet blocks.

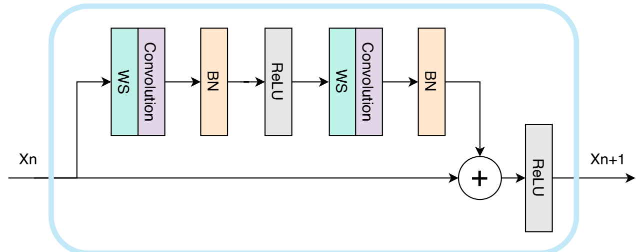
Fig. 14: Basic structure of a ResNet [6] using batch normalization (BN).

Table 3: ResNet18/ResNet50 model hyperparameters for CIFAR100

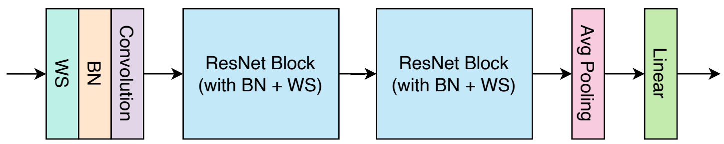
Hyperparameters	**Value**
Optimizer	SGD
Batch Size	128
Learning Rate	1e-3
Epochs	200

For the transformer experiments, we found that layer normalization did much better than batch normalization and also noticed that in many transformer papers, layer normalization was more often used than batch normalization [3,4,19]. Furthermore, we were unable to make weight standardization work with a transformer architecture and this seems to correlate with the literature as no transformer papers use weight standardization. This is the reason why we chose to test against layer normalization and weight normalization as these were the two most commonly used normalizations in the literature.

We apply weight normalization or equilibrated weight conditioning to the MLP head and the transformer encoder. Fig. 21 shows the part of the multi-head attention that we apply normalization to. As can be seen by that figure we only apply it to the linear part. In general, we found that it was enough to apply to just the final linear layer in the multi-head attention as opposed to all linear layers. Applying to all linear layers in the multi-head attention yielded no extra benefit in terms of train loss and accuracy.



(a) ResNet block with weight standardization (WS) applied to each convolutional layer.



(b) We further add weight standardization (WS) to the first convolutional layer.

Fig. 15: Basic structure of a ResNet using weight standardization (WS).

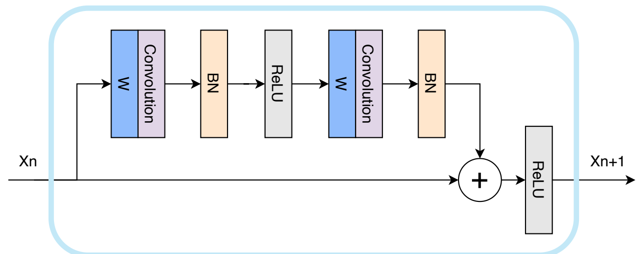
ViT-Small (ViT-S) on CIFAR100: We tested the normalizations on the ViT-S architecture from [16]. Tab. 4 shows the details of this architecture. Tab. 5 shows the hyperparamters we used for training.

Fig. 22 shows the results of applying the above three normalizations and training from scratch on the CIFAR100 dataset. As can be seen from that figure, LN + E is superior over the other 2 normalizations. Tab. 6 shows the final Top-1% and Top-5% accuracy. As can be seen from the table, LN + E performs the best. We note that the accuracies obtained might seem rather low, this is because ViT-S has 22.2 million parameters which is generally too much for a dataset such as CIFAR100.

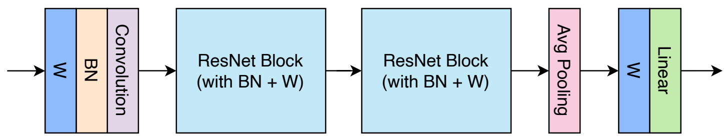
Table 4: Summary of ViT models used in experiments

Model	**Layers**	**Width**	**MLP**	**Heads**	**Params**
ViT-Ti	12	192	768	3	5.8M
ViT-S	12	384	1536	6	22.2M
ViT-B	12	768	3072	12	86M

ViT-Base (ViT-B) on ImageNet1k: In Sec. 6.2 of the main paper we compared the three initializations on the ViT-B architecture from [4]. We trained this on the ImageNet1k dataset using the hyperparameters from [4].



(a) ResNet block with weight normalization (W) applied to each convolutional layer.



(b) A ResNet with two ResNet blocks.

Fig. 16: ResNet with weight normalization (W).

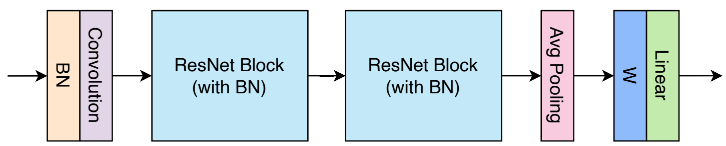


Fig. 17: For the experiments, we applied weight normalization to only the last linear layer as this performed equivalent to or better than applying it on other layers.

ViT-Tiny (ViT-T) on ImageNet1k: We also trained a ViT-T architecture from [16] with the three normalizations. We used the training regime presented in [16]. Fig. 23 shows the results of the experiment.

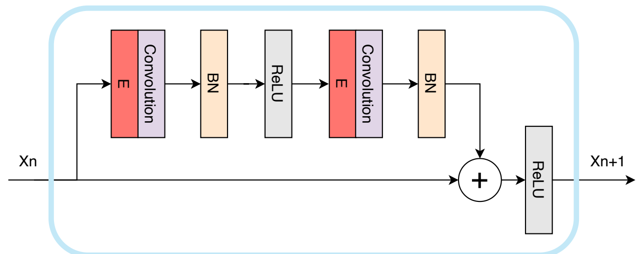
2.3 Neural Radiance Fields (NeRF)

In Sec. 6.3 of the main paper we gave the results for testing equilibrated weight conditioning on NeRF using the LLFF dataset [10]. Fig. 24 shows the standard NeRF architecture from [10] that we used to train the NeRF on the LLFF dataset. Fig. 25 shows the equilibrated weight conditioned NeRF (E-NeRF) architecture that we used in Sec. 6.3. The training hyperparamters that we used were all the same as that used in [10].

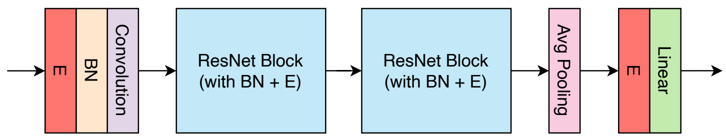
3 Further Experiments

3.1 3D Shape Modelling

We refine the representation of a 3D shape by optimizing a binary occupancy field, which is defined as the decision boundary within a feedforward neural



(a) ResNet block with equilibrated weight conditioning (E) applied to each convolutional layer.



(b) A ResNet architecture with equilibrated weight conditioning (E) applied to the convolutional layer and the last linear layer.

Fig. 18: ResNet with equilibrated weight conditioning (E).

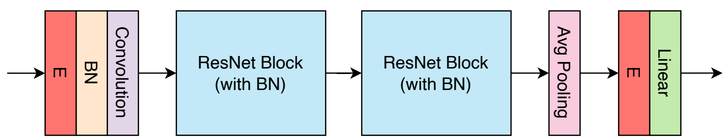


Fig. 19: For the experiments we applied equilibrated weight conditioning (E) to the first convolutional layer and the last linear layer as this performed equivalent too or better than applying it on all layers.

network [5,17]. Specifically, our work utilizes the *thai statue* instance, as sourced from [1]. Drawing on the methodology described in [14], our approach involves sampling across a $512 \times 512 \times 512$ grid, where each voxel inside the designated volume is marked with a 1, while those outside are marked with a 0, to define the occupancy volumes. To assess the accuracy of these volumes, we will use the intersection over union (IoU) metric as is common in the literature for 3D shape modelling, with the higher IOU being the better.

We will utilize a feedforward network with two hidden layers, each comprising 256 neurons, and we adopt the Gaussian activation function as the non-linearity. With this architecture, we trained three types of model:

1. A standard feedforward network with two hidden layers with 256 neurons each and no normalization (vanilla).
2. The feedfoward network from 1 with weight normalization (W).
3. The feedforward network from 1 with equilibrated weight conditioning (E).

We found that batch/layer normalization and weight standardization did not train properly for such small networks and therefore focused on the above three

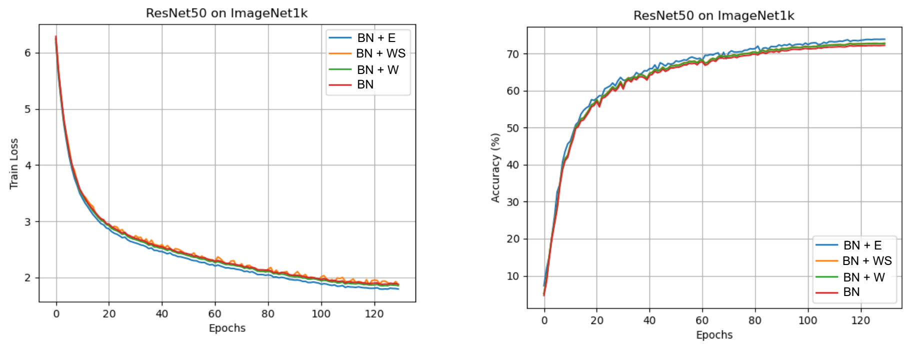


Fig. 20: Left: Train loss for the ResNet50 model from [6] trained on ImageNet1k. The equilibrated weight conditioned network achieves the lowest loss. Right: Top-1% accuracy on ImageNet1k dataset. Equilibrated weight conditioned network has slightly higher accuracy than the others.

Table 5: ViT-S model hyperparameters for training on CIFAR100

Hyperparameters	**Value**
Optimizer	AdamW
Batch Size	256
Learning Rate	1e-3
Epochs	200

normalizations. Furthermore, this fit with the literature where we found that none of the major works for occupancy fields were using any type of normalization. The training configuration for the networks is shown in Tab. 7.

The final train PSNR (dB) and the final IOU after convergence is given in Tab. 8. As can be seen from that table, the equilibrated weight conditioned network performs the best. Fig. 26 shows the reconstructions.

3.2 ConvNext

In Sec. 6.1 of the main paper and Sec. 2.1 of this supplementary material, we have trained various CNN architectures with weight conditioning showing that they get better accuracy and lower loss in each case. The CNN architectures we used were quite classical. In this section, we test weight conditioning on a more recent CNN architecture called ConvNeXt [9] from 2022.

We trained ConvNext on the ImageNet1k dataset using the regime outlined in [9]. It was found in [9] that layer normalization performs better than batch normalization and hence they used layer normalization for their experiments. We will follow their approach and use layer normalization on our networks. We consider four normalizations, layer normalizaton (LN), layer normalization with weight standardization (LN + WS), layer normalization with weight normalization (LN + W) and layer normalization with equilibrated weight conditioning (LN + E).

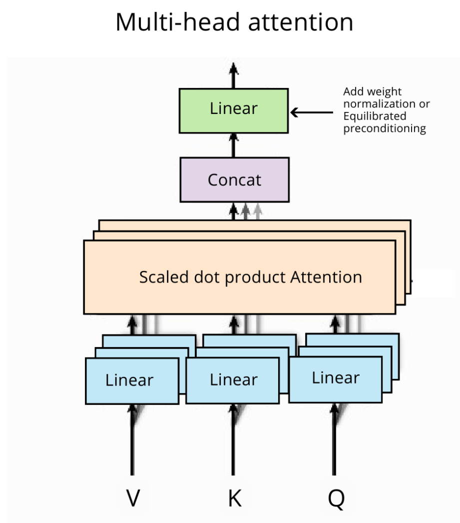


Fig. 21: Multi-head attention structure in a transformer network. Weight normalization and weight conditioning is applied only to the final linear layer.

Table 6: Final Top-1% and Top-5% accuracy for the three normalizations on a ViT-S trained on CIFAR100.

	CIFAR100	
	Top-1%	Top-5%
LN + E	**55.3**	**67.8**
LN + W	50.1	64.4
LN	49.9	64.1

Fig. 27 shows the results of the experiment. It is clear from that experiment that the equilibrated weight conditioned network reaches the highest Top-1% accuracy.

3.3 Computational Cost

The normalizations we have tested in this paper all involve some manipulation of the weight matrices within a neural architecture. In this section we go through the computational cost of one iteration of each normalization for a gradient descent algorithm.

We computed the wall time for one iteration of the gradient optimizer when training each of the networks. Fig. 28 shows the results for the case of Inception and DenseNet trained on CIFAR100, ViT-S trained on CIFAR100 and NeRF trained on the Fern instance from the LLFF dataset [10]. We see that in all cases training a network that has equilibrated weight conditioning applied will increase the training time.

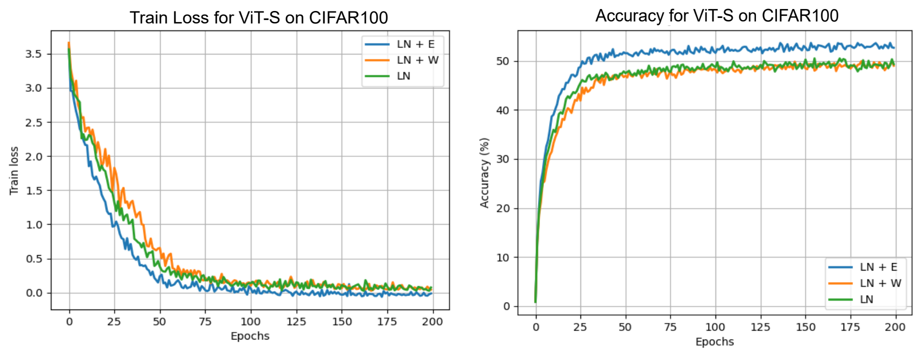


Fig. 22: Left: Train loss for all three normalized transformers on CIFAR100. Equilibrated weight conditioned converges the fastest. Right: Top-1% for all three networks. Equilibrated weight conditioned gives the highest accuracy.

Table 7: Binary Occupancy training hyperparameters.

Hyperparameters	**Value**
Optimizer	Adam
Batch Size	20000
Learning Rate	1e-3
Epochs	200

3.4 Ablations

Microbatching: In the experiments so far we have primarily stuck to batch sizes outlined in those papers whose architectures we are using. However, normalizations such as weight standardization work well for smaller micro batches. In this section we show an ablation on how these normalizations behave with respect to batch size.

We applied the normalizations from Sec. 6.1 of the main paper to the DenseNet architecture we used to train on CIFAR10 from that section. We then varied the batch size taking smaller and smaller batch sizes for training under SGD. The batch sizes we chose were, 64, 32, 16, 8, 4, 1. In this way we can see how weight conditioning is affected by microbatching. We trained each network for 50 epochs with each batch using SGD with a fixed learning rate of 1e-3.

Fig. 29 shows the results of the experiment. Overall, the architecture employing BN + WS performed the most stable which is not surprising as WS is particularly useful for microbatches, see [12]. The network employing BN + E did reasonably well performing better than those employing BN + W and just BN, showing that equlibrated weight conditioning is useful for microbatch training.

Learning Rates: In Sec. 4 and 5 of the main paper we showed how weight conditioning via an equilibrated preconditioner reduces the condition number of

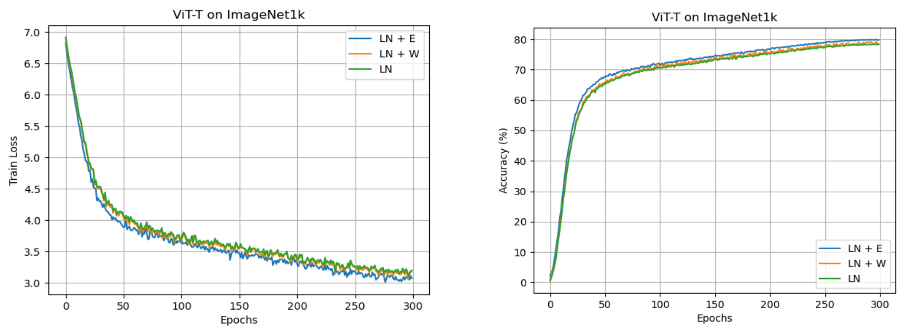


Fig. 23: Left: Train loss for ViT-T trained on ImageNet1k. The equilibrated weight conditioned network achieves the lowest loss amongst the three. Right: Top-1% accuracy on ImageNet1k dataset. Equilibrated weight conditioned network achieves the highest accuracy.

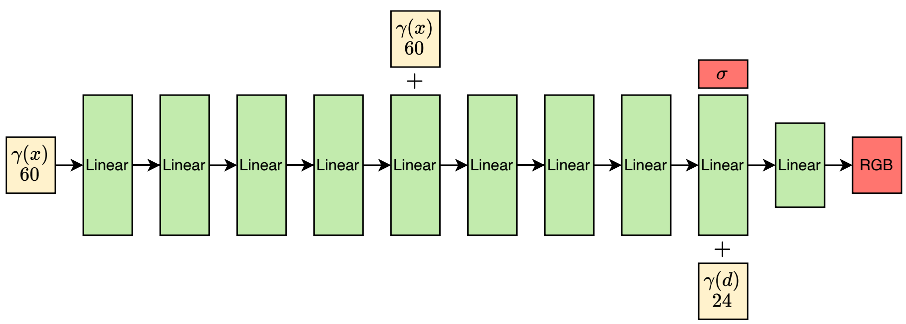


Fig. 24: Standard NeRF [10] architecture. Here γ denotes a positional embedding, see [10] for details.

the Hessian of the objective function. An effect of this should be that during training much larger learning rates should be able to be used leading to faster convergence. In this section we do an ablation study on larger learning rates and their effect on the various normalization schemes we have considered thus far.

The first experiment we ran was to train an Inception network with large learning rates till convergence. We applied each of the four normalizations, BN, BN + WS, BN + W and BN + E and trained each network for 50 epochs for a fixed learning rate starting from 3 and moving down to 0.5.

The second experiment we ran was similar to the above but using a ViT-S architecture with the normalizations LN, LN + W and LN + E. We trained these networks with an AdamW optimizer starting with a learning rate of 2e-3, training till convergence which was about 80 epochs, and then bringing down the learning rate and repeating.

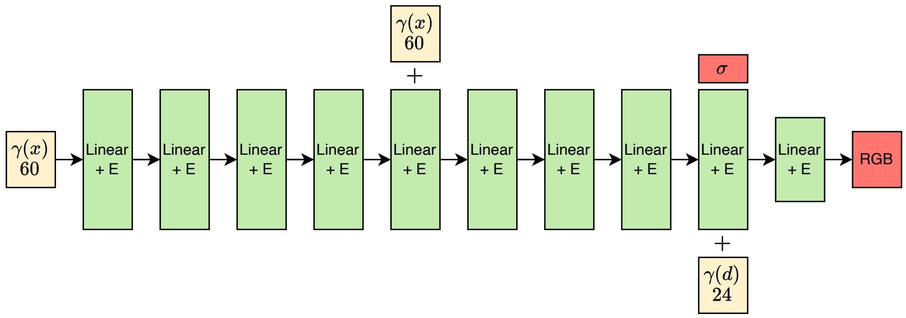


Fig. 25: Equilibrated weight conditioned NeRF architecture. The weight conditioning (E) is applied to each linear layer.

Table 8: Binary occupancy results for the three normalizations on the Thai statue instance.

	Train PSNR (dB)	IOU
Vanilla	21.6	89.4
W	20.8	88.9
E	23.2	92.3

The purpose of these two experiments is to see how each of those normalizations behave with respect to larger learning rates than are typically used in the literature. As predicted by our theory in Sec. 5.1 of the main paper, the networks applying equilibrated weight conditioning were much more stable to larger learning rates yielding decent Top-1& accuracy scores, while the others tends to perform exceedingly badly at the higher learning rates. Fig. 30 shows the results of the experiments.

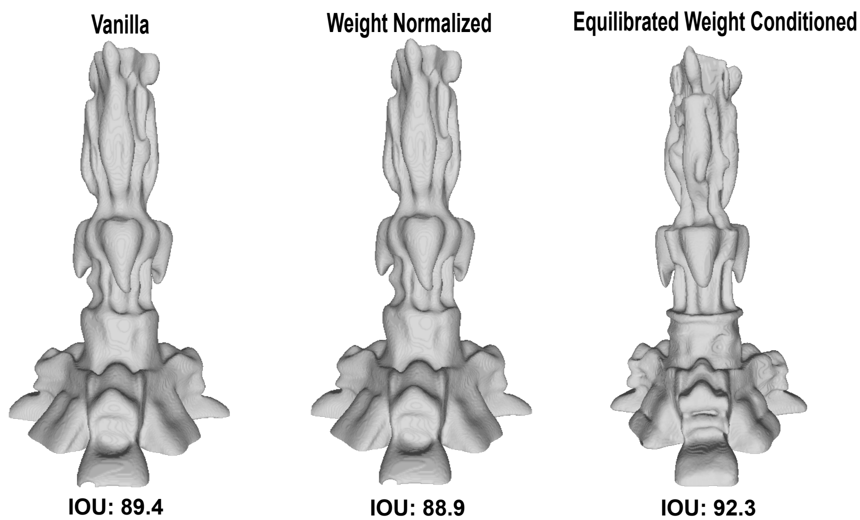


Fig. 26: Reconstructions from training a binary occupancy network with no normalization (vanilla), weight normalization and equilibrated weight conditioning. As can be seen from the reconstruction the equilibrated weight conditioned network has produced the highest IOU.

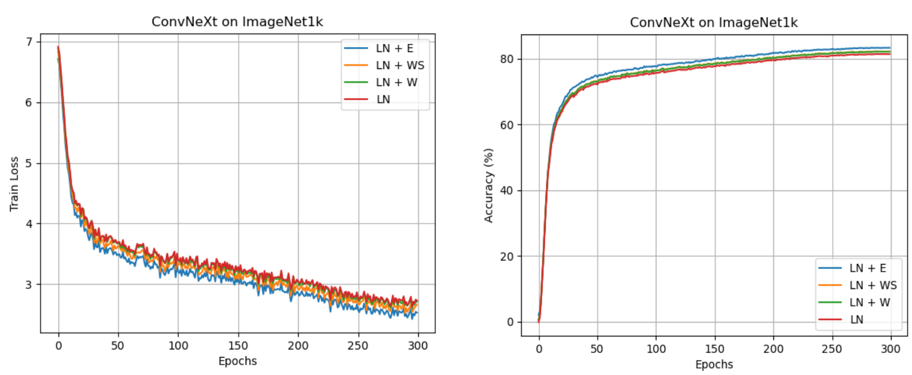


Fig. 27: Left: Train loss for the four ConvNeXt network. The equilibrated weight conditioned one reaches the lowest train loss. Right: The top-1% accuracy for the four networks. The equilibrated weight conditioned network achieves the highest accuracy.

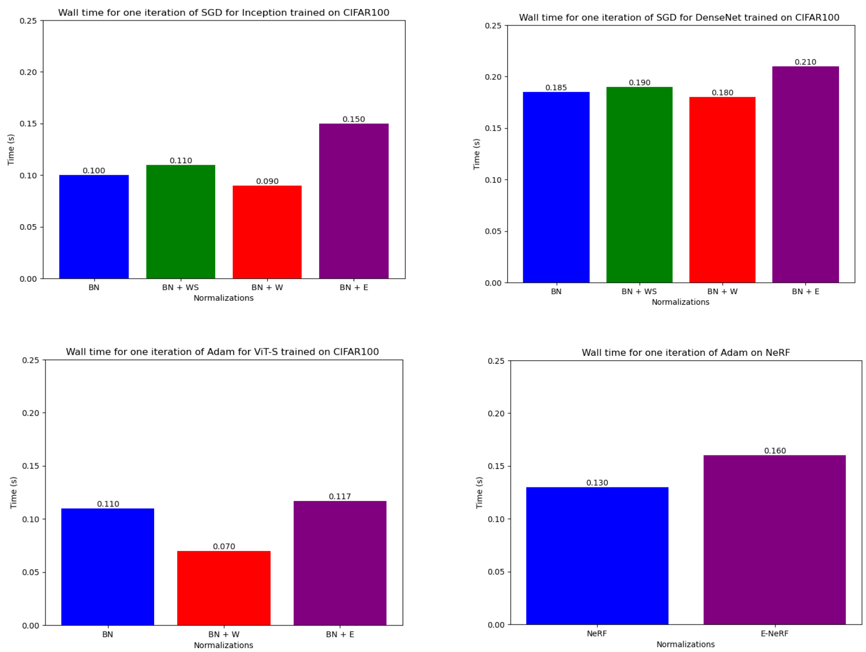


Fig. 28: Top left: The wall time for one iteration of SGD for the Inception model from Sec. 2.1. We see that equilibrated weight conditioning takes approximately 0.04s longer than the other methods. Top right: Wall time for one iteration of SGD for the DenseNet model from Sec. 2.1. Equilibrated weight conditioning takes approximately 0.02s longer than the others. Bottom left: Wall time for one iteration of AdamW for ViT-S trained on the CIFAR100 dataset. In this case equilibrated weight conditioning takes approximately 0.007s longer than the others. Bottom right: Wall time for one iteration of Adam for NeRF. Equilibrated weight conditioning takes approximately 0.03s longer.

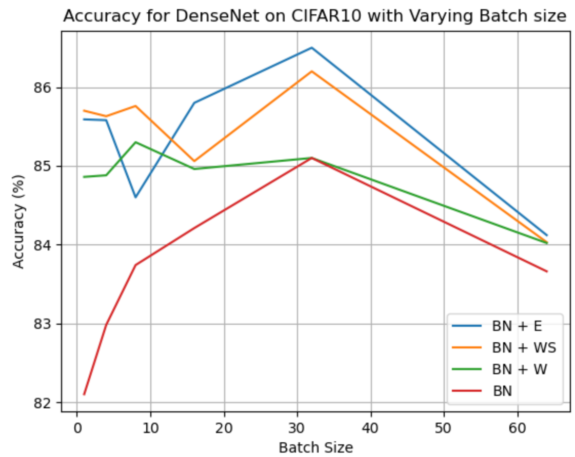


Fig. 29: Top-1% accuracy for a DenseNet on CIFAR10 with varying batch size. The network applying batch normalization together with weight standardization performs much more stably than the others as the batch size gets smaller.

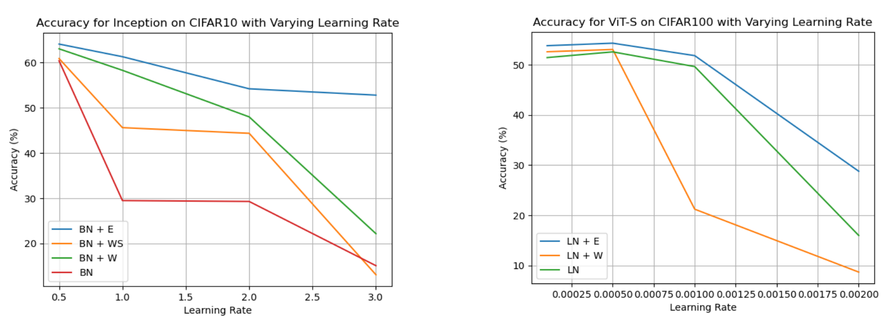


Fig. 30: Left: Training the four different normalizations with varied learning rates using SGD for Inception on CIFAR10 until convergence. The architecture employing BN + E yields good accuracy of 52% even at a large learning rate of 3 while all the others are below 25%. Right: The same experiment done for ViT-S and with the AdamW optimizer. In this case the architecture employing LN + E is much more stable with higher learning rates than the other two.

References

1. The Stanford 3D Scanning Repository, http://graphics.stanford.edu/data/3Dscanrep/ 15
2. Ba, J.L., Kiros, J.R., Hinton, G.E.: Layer normalization. arXiv preprint arXiv:1607.06450 (2016) 11
3. Brody, S., Alon, U., Yahav, E.: On the expressivity role of layernorm in transformers' attention. arXiv preprint arXiv:2305.02582 (2023) 12
4. Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al.: An image is worth 16x16 words: Transformers for image recognition at scale. arXiv preprint arXiv:2010.11929 (2020) 12, 13
5. Gropp, A., Yariv, L., Haim, N., Atzmon, M., Lipman, Y.: Implicit geometric regularization for learning shapes. ICML (2020) 15
6. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016) 9, 10, 12, 16
7. Huang, G., Liu, Z., Van Der Maaten, L., Weinberger, K.Q.: Densely connected convolutional networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 4700–4708 (2017) 6, 7, 8
8. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: International conference on machine learning. pp. 448–456. pmlr (2015) 4
9. Liu, Z., Mao, H., Wu, C.Y., Feichtenhofer, C., Darrell, T., Xie, S.: A convnet for the 2020s. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 11976–11986 (2022) 16
10. Mildenhall, B., Srinivasan, P.P., Tancik, M., Barron, J.T., Ramamoorthi, R., Ng, R.: Nerf: Representing scenes as neural radiance fields for view synthesis. Communications of the ACM **65**(1), 99–106 (2021) 14, 17, 19
11. Papyan, V.: Measurements of three-level hierarchical structure in the outliers in the spectrum of deepnet hessians. arXiv preprint arXiv:1901.08244 (2019) 2
12. Qiao, S., Wang, H., Liu, C., Shen, W., Yuille, A.: Micro-batch training with batch-channel normalization and weight standardization. arXiv preprint arXiv:1903.10520 (2019) 4, 18
13. Salimans, T., Kingma, D.P.: Weight normalization: A simple reparameterization to accelerate training of deep neural networks. Advances in neural information processing systems **29** (2016) 4
14. Saragadam, V., LeJeune, D., Tan, J., Balakrishnan, G., Veeraraghavan, A., Baraniuk, R.G.: Wire: Wavelet implicit neural representations. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 18507–18516 (June 2023) 15
15. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 1–9 (2015) 4, 5
16. Touvron, H., Cord, M., Douze, M., Massa, F., Sablayrolles, A., Jégou, H.: Training data-efficient image transformers & distillation through attention. In: International conference on machine learning. pp. 10347–10357. PMLR (2021) 13, 14
17. Wang, P.S., Liu, Y., Yang, Y.Q., Tong, X.: Spline positional encoding for learning 3d implicit signed distance fields. arXiv preprint arXiv:2106.01553 (2021) 15

18. Wu, Y., He, K.: Group normalization. In: Proceedings of the European conference on computer vision (ECCV). pp. 3–19 (2018) 4
19. Xiong, R., Yang, Y., He, D., Zheng, K., Zheng, S., Xing, C., Zhang, H., Lan, Y., Wang, L., Liu, T.: On layer normalization in the transformer architecture. In: International Conference on Machine Learning. pp. 10524–10533. PMLR (2020) 12