Variance-Aware Weight Initialization for Point Convolutional Neural Networks

Pedro Hermosilla[1], Michael Schelling[1], Tobias Ritschel[2], and Timo Ropinski[1]

[1] Ulm University, Germany
[2] University College London, UK

Abstract. Appropriate weight initialization has been of key importance to successfully train neural networks. Recently, batch normalization has diminished the role of weight initialization by simply normalizing each layer based on batch statistics. Unfortunately, batch normalization has several drawbacks when applied to small batch sizes, as they are required to cope with memory limitations when learning on point clouds. While well-founded weight initialization strategies can render batch normalization unnecessary and thus avoid these drawbacks, no such approaches have been proposed for point convolutional networks. To fill this gap, we propose a framework to unify the multitude of continuous convolutions. This enables our main contribution, variance-aware weight initialization. We show that this initialization can avoid batch normalization while achieving similar and, in some cases, better performance.

1 Introduction

Weight initialization schemes play a crucial role when training deep neural networks. By initializing weights appropriately, it can be avoided that layer activations diminish or explode during a forward pass through the network. Accordingly, researchers have dedicated great efforts in order to optimize the weight initialization process, such that these downsides are circumvented [7]. As modern weight initialization schemes have been developed for structured CNNs, they are based on a few assumptions, that do not hold (as well show in Sec. 4.2), when learning on unstructured point cloud data. Nevertheless, it is common practice, to overlook this discrepancy, and to apply these initialization schemes. These initialization schemes result in exploding or vanishing variance in layer activations, when learning on point cloud data. To overcome this shortcoming, usually batch normalization is applied, as it helps to rescale activations based on the current batch. While this approach is highly effective when learning on some moderate-resolution image data, where many training samples fit into memory and thus can be considered in one batch, it has severe downsides when learning on point clouds. Due to memory limitations, these data sets usually only allow for rather small batch sizes. This makes the sample mean and variance of each individual batch usually not representative of the entire training set. Therefore, in order to use batch normalization on point clouds, researchers are usually forced to reduce the point cloud resolution or to process each scene in chunks. However, as recently shown by Nekrasov et al. [15] and Choy et al. [3], the sampling resolution and the context size play crucial roles in the final prediction of the model. Whilst methods exist to increase the batch size virtually during

training, such as accumulating the gradients over several batches or using multiple GPUs, batch normalization has the same limitations in these setups since the mean and standard deviation are computed separately in each iteration/GPU.

Within this paper, we tackle the drawbacks which result from applying weight initialization schemes, originally developed for structured data, when learning on point clouds. Based on our observations of layer activations in point convolutional networks, we are able to derive a variance-aware initialization scheme, which avoids the aforementioned downsides. To this end, we make the following contributions:

- A unified mathematical framework for 3D point convolutional neural networks.
- We show that spatial autocorrelation increases with the depth of point convolutional networks, and show how to account for it with variance-aware weight initialization.
- We demonstrate how the proposed weight initialization scheme can be generalized across training data sets, and thus does not require additional preprocessing.

Since the proposed weight initialization scheme is variance-aware, we are able to omit batch normalization during the training process. Thus we do not only avoid the aforementioned issues which come with batch normalization, but we also are able to use larger point clouds during training and therefore arrive at en-par or sometimes even better results. To our knowledge, the proposed initialization scheme is the first one, which has been specifically developed for point convolutional neural networks.

2 Related Work

Training of deeper convolutional neural networks (CNNs) [18] is a challenging optimization and hence benefits from suitable initialization. Simonyan and Zisserman [18] simply used Gaussian noise with a small standard deviation. Improving upon this, He et al. [7] proposed an initialization that takes into account the effect variance of the activations. These directly affect the numeric quality of the gradients and can be important for convergence. Their design is specific to convolutions with certain non-linearities. Mishkin and Matas [14] and Krähenbühl et al. [11] have devised alternative inits for CNNs which initialize layer-by-layer such that that the variance of the activation affect each layer remains constant, e.g. close to one. Alternative to good initialization, batch normalization [10] can serve a similar purpose: instead of changing the tunable parameters (weights), additional normalization using the statistics of each batch is employed. These deteriorate if the sample statistics are not representative of the true statistics, which is pressing if models are complex and batches are small. Our work aims to enable the same benefits of good initialization found for CNNs on images on unstructured convolutions [1, 19, 12, 8, 21, 6] as used for 3D point clouds.

3 Formalizing Point Cloud Convolution

We propose a formalization to cover a large range of existing 3D point cloud convolutions. Our contribution is to derive an initialization based on that.

Convolution of output feature F^l at layer l is defined as:

$$F^l(\mathbf{x}) = \sum_{c=0}^{C} \int F_c^{l-1}(\mathbf{x}+\tau)\kappa_c(\tau)\mathrm{d}\tau \tag{1}$$

where C is the number of input features, κ is the convolution kernel and τ an offset. This formulation alone is well-known. What is not formalized so far, however, is the many different learnable point cloud convolutions.

To represent κ, we use a $w_{c,i}$-weighted sum of the projection of the offset τ into a set of K basis functions b_i:

$$\kappa_c(\tau) = \sum_{i=0}^{K} b_i(\tau) w_{c,i}, \tag{2}$$

resulting in the following definition of convolution:

$$F^l(\mathbf{x}) = \sum_{c=0}^{C} \int F_c^{l-1}(\mathbf{x}+\tau) \sum_{i=0}^{K} b_i(\tau) w_{c,i} \mathrm{d}\tau. \tag{3}$$

This definition of convolution covers many existing state-of-the-art methods, as well as the common discrete convolution used for images. We will first show how discrete convolution can be represented by Eq. 3, before applying it to continuous convolutions.

3.1 Discrete convolution

Here, the bases b_i are Dirac delta functions on the positions of the kernel points $\mathbf{p}_i$:

$$b_i(\tau) = \delta(\tau - \mathbf{p}_i) \tag{4}$$

In images, these points are placed on a regular grid in each axis. Therefore, if pixels are laid out on a regular grid, the kernel can be reduced to a matrix indexed by pixel displacements, a summation over the neighboring pixels:

$$F^l(x) = \sum_{c=0}^{C} \sum_{i=-\tau}^{\tau} F_c^{l-1}(\mathbf{x}+i) w_{c,i}. \tag{5}$$

3.2 Continuous convolution

Several methods have been proposed to perform a convolution in the continuous domain, as it is beneficial when learning on point clouds. As stated above, many of these methods can be expressed through the mathematical framework introduced in Section 3. To illustrate this, we have selected the most commonly used and highly cited continuous convolutions to be expressed in our framework as summarized in Tbl. 1. This "zoo" can be structured along two axis: the basis (Sec. 3.3) and the integral estimation (Sec. 3.4).

Table 1: Taxonomy of continuous convolutions.

Method	Basis	Estim.	Init
Atzmon et al. [1]	Gauss	$\hat{A}_{\text{MC}}$	He et al. [7]
Lei et al. [12]	Box	$\hat{A}_{\text{Avg}}$	He et al. [7]
Thomas et al. [19]	Lin. Corr.	$\hat{A}_{\text{Sum}}$	He et al. [7]
Hermosilla et al. [8]	MLP	$\hat{A}_{\text{MC}}$	He et al. [7]
Wu et al. [21]	MLP	$\hat{A}_{\text{NN}}$	He et al. [7]
Groh et al. [6]	Dot	$\hat{A}_{\text{Sum}}$	He et al. [7]
Hua et al. [9]	Box	$\hat{A}_{\text{Avg}}$	He et al. [7]
Mao et al. [13]	Lin. Corr.	$\hat{A}_{\text{Avg}}$	He et al. [7]
Boulch [2]	MLP Corr.	$\hat{A}_{\text{Avg}}$	He et al. [7]

3.3 Zoo Axis 1: Basis

We will here show how most published work on deep point cloud learning (those that are convolutional, which would not include the acclaimed PoinNet architecture [16]) can be expressed in the framework of different basis functions (Fig. 1), allowing to derive a joint way of initialization.

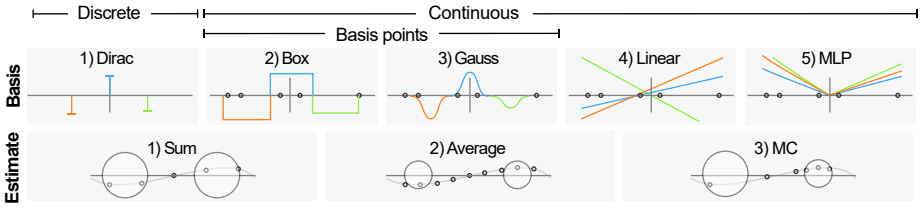

Fig. 1: Our framework organizes continuous convolution along the basis function (top) and convolution integral estimation (bottom) axis.

In **PCCNN**, Atzmon et al. [1] define the basis b_i as Gaussians of the distance to a set of kernel points $\mathbf{p}_i$:

$$b_i(\tau) = \exp\left(-\frac{\|\mathbf{p}_i - \tau\|^2}{s}\right), \tag{6}$$

where s is a bandwidth parameter.

For **PointWiseCNN**, Hua et al. [9] also used a set of kernel points as PCCNN. However, the authors used a box function as point correlation:

$$b_i(\tau) = \begin{cases} 1 & \underset{j}{\text{argmin}}(\|\mathbf{p}_j - \tau\|) = i \\ 0 & \underset{j}{\text{argmin}}(\|\mathbf{p}_j - \tau\|) \neq i \end{cases} \tag{7}$$

This approach was also adopted by Lei et al. [12] in **SPHConv**, but in spherical coordinates.

In **KPConv**, Thomas et al. [19] also used a set of kernel points as PCCNN. However, the authors used linear correlation instead of a Gaussian:

$$b_i(\tau) = \max\left(1 - \frac{\|\mathbf{p}_i - \tau\|}{s}, 0\right). \tag{8}$$

Here, kernel points are arranged as the vertices of platonic solids.

Linear correlation was also used by He et al. [7] in their convolution operation **InterpCNN** with kernel points arranged in a grid.

In **ConvPoint**, Boulch [2] also used kernel points. However, the correlation function was learned by an MLP instead.

In the **MCConv** work of Hermosilla et al. [8], the basis b_i is defined as the output of an MLP $\alpha(\tau)$ with a structure too complex to be written as an equation here. According to our taxonomie's axes, **PointConv** by Wu et al. [21] is the same as MCConv and only differs in the implementation of the MLP and along the integration design axis.

The **FlexConv** approach by Groh et al. [6] uses a single basis $\mathbf{v}_i$ for each input feature and it is defined as the affine projection $\cdot_1$ of the point to the learned vector:

$$b_i(\tau) = \mathbf{v}_i^T \cdot_1 \tau \tag{9}$$

These basis functions can be interpreted as a learned unit vector scaled by the convolution weight.

3.4 Zoo Axis 2: Integral Estimation

Orthogonal to the choice of basis just discussed, different methods also use different ways of estimating the inner convolution integral in Eq. 3. To see those differences, consider writing the integral as

$$A(\mathbf{x}) = \int a(\mathbf{x}, \tau)\mathrm{d}\tau \quad \text{where} \tag{10}$$

$$a(\mathbf{x}, \tau) = F_c^{l-1}(x + \tau) \sum_{i=0}^{K} b_i(\tau)\mathbf{w}_{c,i}.$$

In point cloud processing, several ways have been proposed to estimate this integral, based on summing, averaging, MC estimation and MC with learned density:

$$\hat{A}_{\mathrm{Sum}}(\mathbf{x}) = \sum_{y \in \mathcal{N}(\mathbf{x})} a(y) \qquad [6, 19], \tag{11}$$

$$\hat{A}_{\mathrm{Avg}}(\mathbf{x}) = \sum_{y \in \mathcal{N}(\mathbf{x})} a(y)/|\mathcal{N}(\mathbf{x})| \qquad [9, 12], \tag{12}$$

$$\hat{A}_{\mathrm{MC}}(\mathbf{x}) = \sum_{y \in \mathcal{N}(\mathbf{x})} a(y)/(p(y)|\mathcal{N}(\mathbf{x})|) \quad [1, 8], \tag{13}$$

$$\hat{A}_{\mathrm{NN}}(\mathbf{x}) = \sum_{y \in \mathcal{N}(\mathbf{x})} a(y)/\pi(p(y)) \qquad [21]. \tag{14}$$

The most similar approach to the discrete convolution case is to use a sum (Eq. 11) over the neighboring samples [6, 19]. Although easy to implement, it is sensitive to neighborhoods with a variable number of samples, as well as to non-uniformly sampled point clouds (second row in Fig. 1). To consider this shortcoming, other approaches normalize over the average (Eq. 12) of the neighboring contributions [9, 12, 13, 2]. This is robust to neighborhoods with a different number of samples. However, they are not robust under non-uniformly sampled point clouds. To be able to learn robustly on non-uniformly sampled point clouds, other methods have used a weighted sum Eq. 13, where weights depend on the density of the points, following the principles of Monte Carlo integration [1, 8]. Wu et al. [21] additionally propose an MPL π to map the estimated density to a corrected density as per Eq. 14. These methods are robust under different numbers of neighboring samples, as well as non-uniformly sampled point clouds.

4 Weight Initialization

We will now derive weights w that are *optimal in terms of feature variance* for the general form of convolution we describe in Eq. 18, for any form of convolution integral estimation and any basis.

Weights are optimal, if the variance of the features does not increase/decrease for increasing layer depth [7]. This is best understood from plots where the horizontal axis is network layer and the vertical axis is variance (Fig. 2). A method, such as uniform initialization will have increasing/decreasing variance for increasing depth. Previous work [7] has enabled keeping the variance constant as shown by the pink curve. All the continuous methods use similar initializations where variance decreases (solid red curves). Our contribution is the blue curve: variance remains constant for the continuous case.

Based on the convolution framework introduced in Section 3, we will in this section, first describe the weight initialization commonly used for discrete convolutions, before detailing the currently used weight initialization for point convolutional neural networks. Based on the shortcomings of these initialization schemes, we will further introduce our new initialization scheme, which we have developed for point convolutional neural networks.

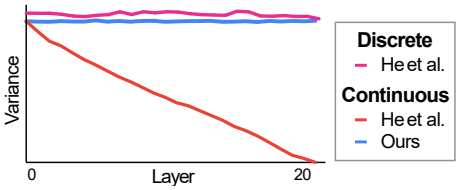

Fig. 2: Good weight initialization prevents decreasing variance (vertical axis) with increasing layer depth (horizontal axis).

4.1 Discrete convolutions

The weight parameters of a discrete convolution are usually initialized independently with respect to the input data. The underlying idea is to initialize the weights in such a way, that the variance of the output features is the same as the variance of the input. The weights are therefore initialized using a normal distribution with a carefully selected variance. This variance is computed, by relying on several assumptions. First, that

the weights are independent of each other, and second, that the features of each pixel are independent of each other. Following these assumptions, He et al. [7] derived the appropriate variance of the weights for convolutional neural networks with ReLUs:

$$\mathbb{V}[w] = \frac{2}{NC}, \tag{15}$$

where N is the number of pixels in the kernel, and C is the number of input features.

4.2 Continuous convolutions

In this section, we discuss the implications that arise, when applying classical weight initialization approaches in the context of point convolutional neural networks, and further propose our weight initialization approach, specifically tailored to point convolutional neural networks.

Common practices A naïve approach would be to use the same initialization scheme for continuous convolution as the one used for discrete convolutions. However, the number of neighboring points highly depends on the data to process, e.g. convolution on samples on a plane will have fewer neighbors than convolutions on molecular data. Therefore, a common approach is to rely on the standard weight initialization schemes provided by the software packages, which in the best case results in the following variance:

$$\mathbb{V}[w] = \frac{2}{BC} \tag{16}$$

where B is the number of basis functions and C is the number of features. Other methods such as presented by Hermosilla et al. [8] and Thomas et al. [19], in their implementation, simply divide by C which produces an even more biased variance and, therefore, worse results. We will consider Eq. 16 as the standard initialization.

Spatial autocorrelation Even though approximating the number of neighbors by the number of basis functions is a crude approximation, this initialization scheme is designed using some assumptions that do not hold, when considering the continuous case. The derivations for the discrete case assumed, that features from neighboring pixels are independent of each other. In the continuous case, however, they are correlated.

Fig. 3 shows the empirical correlogram of the features of different layers of a point convolutional neural network. To obtain this data, we used the simplest 1D point cloud and the simplest continuous convolution conceivable (Fig. 3,a): As a point cloud we sample positions from the uniform random distribution on $(0, 1)$, and the initial features from a normal distribution with variance 0.1. As a basis we use three boxes separated by $r = .05$ and Eq. 11 to estimate the integral. Fig. 3,b depicts the correlation as a function of the distance between points for different layer depth. We can see that no clear pattern for the spatial autocorrelation in the initial features (Input) emerges. However, after the initial convolution, the correlation increases for close points, and this correlation slightly increases and widens with the depth of the network. This is empirical evidence, how the assumption of independence of features between neighbors does not hold in the continuous case, and thus an initialization scheme rooted on this might be suboptimal.

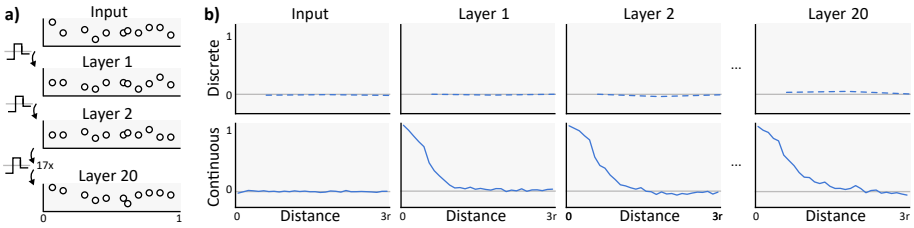

Fig. 3: **a**: Minimal setting for continuous convolution: Starting from the top 1D point cloud we convolve with a 1D 3-tap box kernel 20 times. **b**: Spatial autocorrelation of features in each layer as a function of the distance between points this setting. The top row shows that there is no spatial autocorrelation for the discrete case. However, for continuous convolutions, we can see high spatial autocorrelation for close points and an increase with the depth of the network.

Variance-aware weight initialization To obtain a more suitable weight initialization scheme for point convolutional neural networks, we start our derivation with the definition of the variance of layer l:

$$\mathbb{V}[F^l(\mathbf{x})] = \mathbb{E}[F^l(\mathbf{x})^2] - \mathbb{E}[F^l(\mathbf{x})]^2 \tag{17}$$

We will perform the derivations for a single input channel and use the assumption that each feature channel is independent of each other to scale the resulting variance by the number of channels, C.

Starting with Eq. 17, we compute the expectation of the output features of layer l. Therefore, we assume that each weight w_i is independent of each other, as well as from the basis functions and features, and that they are further initialized from a normal distribution centered at 0. Accordingly, we can reformulate the expectation of the output features of layer l as follows:

$$\begin{aligned}
\mathbb{E}[F^l(\mathbf{x})] &= \mathbb{E}\left[\int F^{l-1}(x+\tau)\sum_{i=0}^{K} b_i(\tau) w_{c,i} \mathrm{d}\tau\right] \\
&= \int \sum_{i=0}^{K} \mathbb{E}\left[F^{l-1}(\mathbf{x}+\tau) b_i(\tau)\right] \mathbb{E}\left[w_{c,i}\right] \mathrm{d}\tau \\
&= \int \sum_{i=0}^{K} \mathbb{E}\left[F^{l-1}(x+\tau) b_i(\tau)\right] 0 \mathrm{d}\tau \\
&= 0
\end{aligned}$$

As the expectation equals 0, the variance defined in Eq. 17 has to be equal to the expectation of the squared features:

$$\mathbb{V}[F^l(\mathbf{x})] = \mathbb{E}[F^l(\mathbf{x})^2]$$

If we expand $\mathbb{E}[F^l(\mathbf{x})^2]$ with our definition of convolution we obtain that it is equal to

$$\mathbb{E}\left[\int G_1 \sum_{i=0}^{K} H_{i,1} w_{k,i} \mathrm{d}\tau_1 \int G_2 \sum_{j=0}^{K} H_{j2} w_{l,j} \mathrm{d}\tau_2\right],$$

where $G_{1/2} = F^{l-1}(x + \tau_{1/2})$ and $H_{i1/2} = b_i(\tau_{1/2})$. Since w is independent of the features and basis functions, we can re-arrange the terms to obtain

$$\int\int \sum_{i=0}^{K}\sum_{j=0}^{K} \mathbb{E}\left[G_1 G_2 H_{i1} H_{j2}\right] \mathbb{E}\left[w_{ki} w_{lj}\right] \mathrm{d}\tau_1 \mathrm{d}\tau_2.$$

This equation can be simplified using the assumption that each weight is also independent of each other, and that the weights are drawn from a distribution centered at 0:

$$\sum_{i=0}^{K}\sum_{j=0}^{K} \mathbb{E}\left[w_i w_j\right] = \sum_{i=0}^{K}\sum_{j=0}^{K} \mathbb{C}[w_i, w_j] + \mathbb{E}[w_i]\mathbb{E}[w_j] = \sum_{i=0}^{K} \mathbb{V}[w] + 0$$

Thus, we can reformulate $\mathbb{E}[F^l(\mathbf{x})^2]$ as:

$$\int\int \sum_{i=0}^{K} \mathbb{E}\left[G_1 G_2 H_{i,1} H_{i,2}\right] \mathbb{V}\left[w\right] \mathrm{d}\tau_1 \mathrm{d}\tau_2$$

Finally, we can arrange the terms to isolate the variance of the weights w on one side of the equality, and thus obtain a closed form to determine the optimal variance for our weights. When additionally including the constant C, to account for the number of input features, we obtain:

$$\mathbb{V}[w] = \frac{\mathbb{V}[F^l(\mathbf{x})]}{C z_l} \qquad \text{where} \tag{18}$$

$$z_l = \mathbb{E}\left[\sum_{i=0}^{K} \int G_{k2} H_{i2} \int G_{k1} H_{i1} \mathrm{d}\tau_1 \mathrm{d}\tau_2\right]. \tag{19}$$

This accounts for the spatial autocorrelation of the input features ($G_{k1} G_{k2}$) as well as the basis functions ($H_{i1} H_{i2}$).

4.3 Variance computation

In order to apply this initialization, we have to compute Eq. 18. First, we specify the desired variance in the output of each layer, $\mathbb{V}[F^l(\mathbf{x})]$, which can be chosen to be a constant such as 1. This leaves us with computing Eq. 19, a statement for layer l given the features in the previous layer $l-1$. Hence, the computation proceeds by-layers, starting at the first layer, adjusting variance and proceeding to the next layer as seen in the outer loop in Alg. 1.

Algorithm 1: Weight initialization.

```
for l ∈ Layers do
    z_l ← 0
    for 1 to N do                          ▷ Eq. 19
        P ← samplePointCloud()
        z̄_l ← estimate(P, l)
        z_l ← z_l + z̄_l
    end for
    z_l ← z_l / N
end for
```

The outer expectation in Eq. 19 is estimated by sampling N random point clouds $\mathcal{P}$. The inner double-integral for each point cloud is a double convolution with the same structure as the convolution Eq. 10. Hence, it can be estimated using one of the techniques defined by Eq. 11, Eq. 12 or Eq. 13. Depending on which technique is used, this might or might not be an unbiased estimator of the true convolution integral, but at any rate, for the initialization to work, the same estimation has to be used that will later be used for actual learning. The function $\texttt{estimate}(\mathcal{P}, l)$ executes the estimation with the weights already initialized up to level $l-1$ on the point cloud $\mathcal{P}$.

As we will show later, this algorithm can be used to estimate z_l from the training data for a specific architecture or can be pre-computed from a set of representative point clouds. In the latter case, the z_l is estimated for a set of consecutive layers where each will capture the spatial autocorrelation introduced by previous layers. These z_l values can be later queried for another network architecture based on the depth of each layer, scaling them by the number of input features, and use the result to initialize the layer.

5 Experiments

5.1 Operators

In order to evaluate the proposed initialization scheme, we have selected widely used point convolution approaches, that can be expressed in our framework introduced in Section 3. During the selection process, we have ensured, that we cover a variety of different basis functions as well as integral approximations and have selected `PCCNN` [1], `KPConv` [19], `SPHConv` [12], `MCConv` [8], and `PointConv` [21].

5.2 Variance evaluation

To empirically validate our derivations, we compare the variance of each layer in a 25-layer network initialized with the standard initialization (Eq. 16) and initialized with ours (Eq. 18). Fig. 4 shows the results obtained from this experiment. As we can see, while the variance of each layer exponentially approaches zero for the standard initialization, ours maintains a constant variance over all layers for all tested convolution operators.

5.3 Classification

We validate our algorithm on the task of shape classification on the SCANOBJECTNN data set [20]. Since all shapes contain similar numbers of points, this is an ideal setup to observe the effects of our initialization scheme under varying batch sizes.

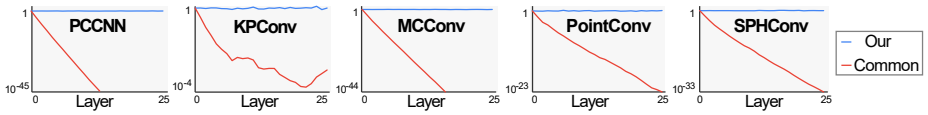

Fig. 4: Comparison of the variance of each layer when applying standard weight initialization (*red*), and when applying our weight initialization (*blue*). The plots show the results for different point convolution operators. Note the vertical axis to be in log-scale.

Data set: The data set is composed of $2,902$ real objects from 15 different categories, obtained from real 3D scans of the SCANNET data set [4]. For this experiment, we use the data set in which only points from the object are used as input to the network, i.e., the background points have been removed. We use the official training/test split which uses $80\,\%$ objects for training and $20\,\%$ for testing. We sample $1,024$ random points from each object from the $2,048$ points provided in the data set. As initial point features, we use a single float with the value of 1.0, while we use random anisotropic scaling on the point coordinates (random scaling in each axis independently between 0.9 and 1.1) and random rotation along the vertical axis for data augmentation. Performance is measured as overall accuracy accumulating the predicted probabilities for each model with different data augmentations.

Network architecture: We used an encoder network with three resolution levels, and two convolution layers per level (see Fig. 5). In order to compute the point cloud levels, we used Poisson disk sampling with radii $[0.1, 0.2, 0.4]$, and three times this radius as the convolution's receptive field. This results in a total of six convolution layers. We also used an increasing number of features per level, $[128, 256, 512]$, and a global average pooling layer on the resulting features to create the global shape descriptor. This descriptor was then processed by a single layer MLP with 512 hidden units, which generates the resulting probabilities. In order to have the same number of parameters for all different convolution operations, we used 16 basis functions for all of them, which enables analyzing the effect of our initialization scheme, while avoiding overfitting issues.

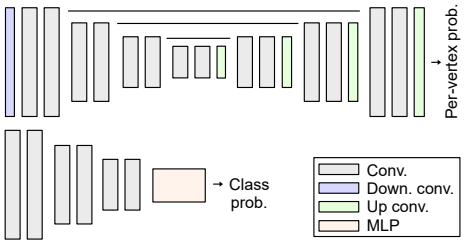

Fig. 5: Network architectures used for the SCANNET data set on the task of semantic segmentation (top) and the SCANOBJECTNN data set for the task of classification (bottom).

Training: We trained the models using SGD with momentum for 650 epochs, batch size between 2 and 16, and an initial learning rate of 0.005. To enable convergence for all methods, we scaled the learning rate by 0.1 after 500 epochs. In order to prevent overfitting, we used a dropout value of 0.2 before each convolution and 0.5 on the final MLP, and we used weight decay loss scaled by 0.0001.

Table 2: Comparison of ours and standard initialization, Group Normalization, and batch normalization on the SCANOBJECTNN data set for different batch sizes.

Batch size →	2					4			8			16		
	Ours	He	Xav.	GN	BN	Ours	GN	BN	Ours	GN	BN	Ours	GN	BN
PCCNN [1]	85.7	13.5	13.5	**86.1**	23.3	85.5	86.7	**87.3**	85.2	86.7	**87.3**	83.5	86.6	**87.9**
KPConv [19]	**84.8**	78.9	13.5	83.8	17.7	**84.5**	83.8	83.4	**85.1**	84.6	84.8	84.1	84.1	**85.3**
MCConv [8]	**86.3**	13.5	13.5	85.5	21.3	**85.9**	85.7	83.5	85.7	85.6	**85.9**	84.4	85.0	**85.6**
PointConv [21]	**85.7**	85.1	85.3	85.0	20.1	**85.2**	84.8	83.4	85.4	84.6	**85.9**	**85.4**	84.9	**85.4**
SPHConv [12]	**82.2**	13.5	13.5	79.8	26.2	**81.9**	80.8	81.6	81.0	80.3	**82.8**	81.0	80.8	**83.4**

Results: The resulting accuracy for different methods and different batch sizes is shown in Table 2. This table shows that our initialization allows to eliminate batch normalization from the network without a significant decrease in performance for most convolution operators. Moreover, we can see that for low batch sizes, our initialization obtains better performance than batch normalization, whose performance reduces with the batch size. Lastly, we can see that our initialization scheme outperforms batch normalization for some convolution operators (`MCConv`), where a small batch size acts as regularization during training. When compared to standard initialization schemes, such as He [7] or Xavier [5], our method always obtains better performance while these methods are not able to converge for most of the convolution operators. Moreover, we compare our initialization scheme with Group Normalization [22], a common normalization technique used for small batch sizes. Tbl. 2 shows that, although Group Normalization enables network training with small batch sizes, with most convolution operators and batch sizes, our initialization scheme obtains better results.

5.4 Semantic Segmentation

We also evaluated our method on the task of semantic segmentation of the SCANNET data set [4]. In this task, each scene is too big to be processed as a whole and it has to be processed in blocks, allowing us to validate our initialization for different scene sizes.
Data set: The SCANNET data set [4] is composed of real 3D scans from $1,513$ different rooms, where the network has to predict the class of the object to which each point belongs to. We use the official splits, corresponding to $1,045$ rooms for training, 156 rooms for validation, and 312 rooms for testing. Since the ground truth annotation for the test set is not publicly available, we evaluated the methods on the validation set. We sample each scan with a sphere of radius r_1 around a random point in the scene and using another sphere of bigger radius, r_2, to select the context, i.e., the points of the bigger sphere are input to the network but we only perform predictions for the points inside the small sphere. We use two sets of radii in our experiments. First, we use $r_1 = 2\,\text{m}$ and $r_2 = 4\,\text{m}$, resulting in point clouds of around $120\,\text{k}$ points for the bigger sphere and around $45\,\text{k}$ for the smaller one. In this setup, we fill the available memory in our system and use only a point cloud from a single room in each training step. Then, we also

use $r_1 = 1\,\text{m}$ and $r_2 = 2\,\text{m}$, resulting in smaller point clouds of around $12\,\text{k}$ and $45\,\text{k}$ points. These radii provide a smaller context for the prediction but allow us to process four rooms in each batch. We use random rotation along the up vector and anisotropic scaling to augment the data during training. Performance is measured as Intersection Over Union (IoU), by accumulating the predicted probabilities of each point sampled from different spheres and with different random rotations and scalings.

Network architecture: We used a U-Net architecture [17] with four different levels and with two convolution layers in each level (see Fig. 5). The different levels are computed using Poisson Disk sampling with different radii, $[0.03, 0.06, 0.12, 0.24]$, and the receptive field of the convolution layers uses three times the Poisson radius of the level. The number of features increases per level in the encoder and decreases per level in the decoder, resulting in $[64, 128, 256, 512, 256, 128, 64]$. The up-sampling operations in the decoder are also continuous convolutions, which results in 19 convolution layers in total. As before, we used 16 basis functions for all tests and all methods.

Training: We trained the models using SGD with momentum for 500 epochs, and, for each epoch, we sample 3000 point clouds. We used an initial learning rate of 0.005, which was scaled by 0.25 after 300 and again by 0.25 after 400, allowing all methods to converge. In order to prevent overfitting, we used a dropout value of 0.2 before each convolution and weight decay loss scaled by 0.0001.

Results: When we train the models with a single scene per batch, Table 3 shows that batch normalization is not able to successfully learn the task. Our initialization scheme, on the other hand, enables us to eliminate batch normalization obtaining competitive IoUs. Moreover, we can see that for some methods, such as `KPConv`, `MCConv`, and `SPHConv`, using standard initialization without batch norm can result in an improvement on performance wrt. batch norm. However, these results do not match the IoU obtained with our initialization scheme. This is true for both variants of ours, the "direct" and the "transfer" one, as shown in the table. The direct one will perform the initialization based on access to the data to choose weights optimally. For the "transfer" one, we computed Eq. 19 on synthetic shapes from the MODELNET40 data set [23] on a network with 25 consecutive layers without any pooling operation or skip connections. Then, we used this initialization on the network for the semantic segmentation task. The only additional step is to select the appropriate w_l value based on the layer depth and

Table 3: Comparison of our initialization on different convolution operations, for Semantic Segmentation on SCANNET, on different batch sizes.

Batch size →	1				4	
	BN	NoBN			BN	NoBN
			Ours			Ours
	Std	Std	Dir.	Tran.	Std	Dir.
PCCNN [1]	32.4	13.3	**65.1**	63.6	**65.7**	62.6
KPConv [19]	33.9	52.5	**66.1**	64.9	66.0	**67.7**
MCConv [8]	40.3	58.8	67.4	**67.9**	**66.1**	65.1
PointConv [21]	41.4	1.6	**62.3**	61.7	**64.2**	62.5
SPHConv [12]	28.1	46.4	**60.7**	60.6	59.5	**60.1**

scale this value by the number of input channels C of the layer. We see that this transfer of weight initialization can work, as performance remains similar while retaining all the benefits and avoiding the need to do anything when deploying our initialization for a specific architecture. Table 3 also shows the results obtained when the networks are trained with four small point clouds per batch. We can see that in this setup, batch normalization achieves high IoU. However, for some methods, `MCConv` and `SPHConv`, using larger point clouds with our initialization significantly increases the obtained IoU.

6 Limitations

Our method is not exempt from limitations. The main limitation of our method is that, contrary to most initialization schemes used for CNN in images, it is data-dependent and requires processing the data first. This pre-processing overhead is negligible to the overall training time: for the SCANNET task results in 47.5 additional seconds from a total of 3 days of training, and for the SCANOBJECTNN task results in 6.1 additional seconds from a total of 4 hours of training. However, we also propose a method to transfer weight initialization's between tasks that does not require pre-processing the data, i.e. our "transfer" setup. However, a large difference between the shapes used to transfer the weight initialization and different statistics between the input features of the data sets could lead to low performance for some operators.

7 Conclusions

In this paper, we have introduced a novel, variance-aware weight initialization scheme, developed for point convolutional neural networks. By exploiting spatial autocorrelation within the layers of a point convolutional neural network, we were able to derive the weight variance used for initialization. In contrast to standard weight initialization schemes, which have been developed for structured data, our proposed weight initialization scheme allows omitting batch normalization, which leads to several issues in the learning process. We have shown, that when using our weight initialization scheme, we are able to neglect batch normalization, and still obtain the same or sometimes even better learning performance. Moreover, we have shown that our weight initialization allows training with small batch sizes and, therefore, larger point clouds, the main limitation of using batch normalization on point clouds. We believe, that the proposed weight initialization scheme, is the first one developed for point convolutional neural networks, and we hope that it will establish itself as the standard method in this subfield.

In the future, we see several endeavors for future work. Same as He et al. [7], we assume features to be independent, which would not be true for positions and normals (one is approximately the derivative of the other) or position or orientation and color (due to shading). Further, we would like to investigate the impact of our weight initialization scheme in other domains, such as graphs and problems such as protein learning.

Acknowledgements. This work was partially funded by the Deutsche Forschungsgemeinschaft (DFG) under grant 391088465 (ProLint) and by the Federal Ministry of Health (BMG) under grant ZMVI1-2520DAT200 (AktiSmart-KI).

Bibliography

[1] Atzmon, M., Maron, H., Lipman, Y.: Point convolutional neural networks by extension operators. ACM Trans Graph (Proc. SIGGRAPH) (2018) 2, 4, 5, 6, 10, 12, 13

[2] Boulch, A.: Convpoint: Continuous convolutions for point cloud processing. Computers & Graphics (2020) 4, 5, 6

[3] Choy, C., Gwak, J., Savarese, S.: 4d spatio-temporal convnets: Minkowski convolutional neural networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 3075–3084 (2019) 1

[4] Dai, A., Chang, A.X., Savva, M., Halber, M., Funkhouser, T., Nießner, M.: Scannet: Richly-annotated 3d reconstructions of indoor scenes. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 5828–5839 (2017) 11, 12

[5] Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (2010) 12

[6] Groh, F., Wieschollek, P., Lensch, H.P.A.: Flex-convolution (million-scale point-cloud learning beyond grid-worlds). In: ACCV (2018) 2, 4, 5, 6

[7] He, K., Zhang, X., Ren, S., Sun, J.: Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In: ICCV, pp. 1026–1034 (2015) 1, 2, 4, 5, 6, 7, 12, 14

[8] Hermosilla, P., Ritschel, T., Vazquez, P.P., Vinacua, A., Ropinski, T.: Monte carlo convolution for learning on non-uniformly sampled point clouds. ACM Trans. Graph. (Pro. of SIGGRAPH Asia) (2018) 2, 4, 5, 6, 7, 10, 12, 13

[9] Hua, B.S., Tran, M.K., Yeung, S.K.: Pointwise convolutional neural networks. In: Computer Vision and Pattern Recognition (CVPR) (2018) 4, 5, 6

[10] 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) 2

[11] Krähenbühl, P., Doersch, C., Donahue, J., Darrell, T.: Data-dependent initializations of convolutional neural networks. arXiv:1511.06856 (2015) 2

[12] Lei, H., Akhtar, N., Mian, A.: Octree guided cnn with spherical kernels for 3d point clouds. CVPR (2019) 2, 4, 5, 6, 10, 12, 13

[13] Mao, J., Wang, X., Li, H.: Interpolated convolutional networks for 3d point cloud understanding. In: International Conference on Computer Vision (ICCV) (2019) 4, 6

[14] Mishkin, D., Matas, J.: All you need is a good init. arXiv:1511.06422 (2015) 2

[15] Nekrasov, A., Schult, J., Litany, O., Leibe, B., Engelmann, F.: Mix3D: Out-of-Context Data Augmentation for 3D Scenes. In: International Conference on 3D Vision (3DV) (2021) 1

[16] Qi, C.R., Su, H., Mo, K., Guibas, L.J.: Pointnet: Deep learning on point sets for 3d classification and segmentation. In: CVPR, pp. 652–660 (2017) 4

[17] Ronneberger, O., Fischer, P., Brox, T.: U-net: Convolutional networks for biomedical image segmentation. In: International Conference on Medical image computing and computer-assisted intervention, pp. 234–241, Springer (2015) 13

[18] Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv:1409.1556 (2014) 2

[19] Thomas, H., Qi, C.R., Deschaud, J.E., Marcotegui, B., Goulette, F., Guibas, L.J.: Kpconv: Flexible and deformable convolution for point clouds. ICCV (2019) 2, 4, 5, 6, 7, 10, 12, 13

[20] Uy, M.A., Pham, Q.H., Hua, B.S., Nguyen, D.T., Yeung, S.K.: Revisiting point cloud classification: A new benchmark dataset and classification model on real-world data. In: International Conference on Computer Vision (ICCV) (2019) 10
[21] Wu, W., Qi, Z., Fuxin, L.: Pointconv: Deep convolutional networks on 3d point clouds. CVPR (2019) 2, 4, 5, 6, 10, 12, 13
[22] Wu, Y., He, K.: Group normalization. In: Proceedings of the European Conference on Computer Vision (ECCV) (2018) 12
[23] Wu, Z., Song, S., Khosla, A., Yu, F., Zhang, L., Tang, X., Xiao, J.: 3d shapenets: A deep representation for volumetric shape modeling. In: Proceedings of 28th IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2015) 13