Supplementary Materials for
SplineCNN: Fast Geometric Deep Learning with Continuous B-Spline Kernels

1. Content

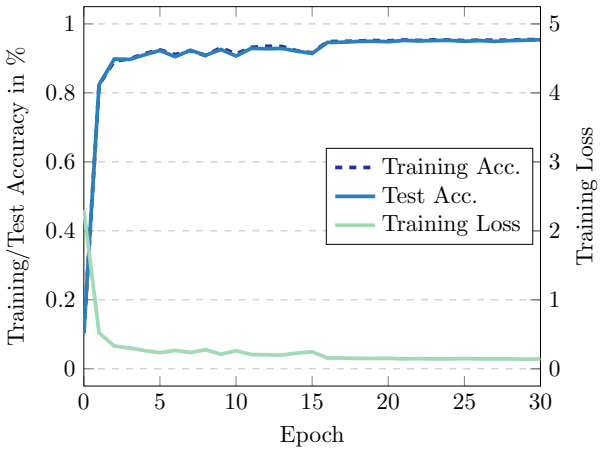
This document contains supplementary materials for the CVPR submission "*SplineCNN: Fast Geometric Deep Learning with Continuous B-Spline Kernels*" with paper id 3872. The materials consist of:

- Plots for training accuracy, test accuracy and loss over the training process for best performing experiment configurations of MNIST superpixels and FAUST experiments in Section 2,
- visualizations of the geodesic error of predictions for all test examples of the FAUST dataset in Section 3,
- more examples for the used MNIST superpixel dataset in Section 4, and
- the GPU algorithm for the backward step of our method in Section 5.

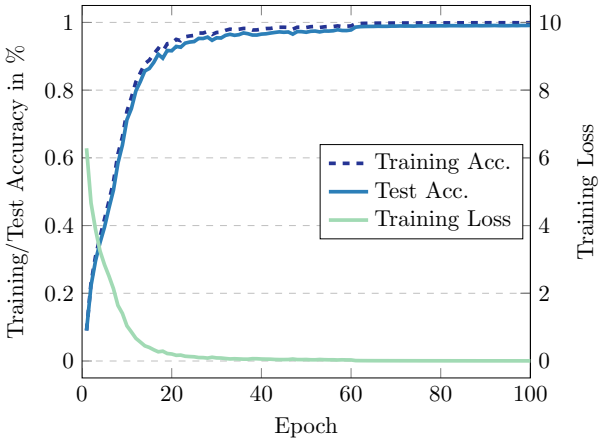
2. Accuracy and loss plots

Training plots for one experiment with the best performing configuration on MNIST superpixels are shown in Figure 1a. The results were obtained with the architecture SConv($(5, 5)$,1,32) $\rightarrow$ MaxP(4) $\rightarrow$ SConv($(5, 5)$,32,64) $\rightarrow$ MaxP(4) $\rightarrow$ AvgP $\rightarrow$ FC(128) $\rightarrow$ FC(10), using Exponential Linear Units (ELUs) as non-linearities after each SConv layer and the first FC layer, B-spline degree $m = 1$ and local Cartesian coordinates as input. The network was trained for 30 epochs using the Adam Optimizer, with batch size 64, an initial learning rate of 0.01 and learning rate decay of 0.001 after epoch 15. It should be noted that the used pooling approach finds node clusters non-deterministically, which leads to some kind of data augmentation in every epoch. This is also the reason why the training accuracy on this augmented training data is not significantly higher than the test accuracy and does not converge to one. However, generalization to the distinct test dataset works very well.

The exact correspondence accuracies (geodesic error of zero) and losses of one experiment with the best experiment configuration for the FAUST shape correspondence task are shown in Figure 1b. The SplineCNN architecture is SConv($(5, 5, 5)$,1,32) $\rightarrow$ SConv($(5, 5, 5)$,32,64) $\rightarrow$



(a) MNIST image classification on superpixels



(b) FAUST shape correspondence

Figure 1: Training accuracies, test accuracies and loss over the training procedure of the two SplineCNNs which achieved the best performance in the respective tasks. For the shape correspondence task, shown in Figure (b), the accuracy for a geodesic error of zero (exact correspondence) is shown. Training loss values are averaged over one epoch.

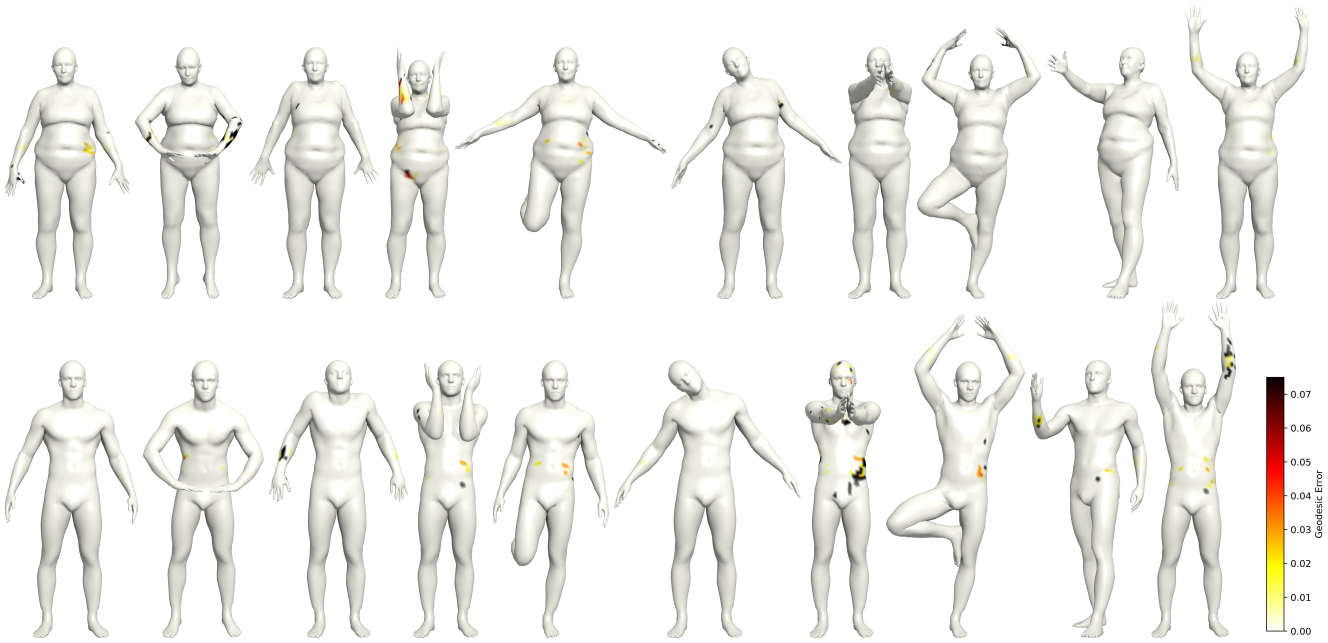


Figure 2: Vertex-wise geodesic errors of SplineCNN predictions on all 20 meshes of the FAUST test dataset.

$4\times$ SConv($(5,5,5)$,64,64) $\rightarrow$ Lin(256) $\rightarrow$ Lin(6890), with ELU activation after each SConv layer, $m = 1$ and three-dimensional Cartesian coordinates as input. Again, the Adam Optimizer was used to train the network for 100 epochs, using a batch size of 1, an initial learning rate of 0.01 and learning rate decay of 0.001 after epoch 60.

3. FAUST shape correspondence results

The vertex-wise geodesic error of SplineCNN predictions for all FAUST test examples is shown in Figure 2. The observation from the paper can be verified on all test examples: Most of the vertices have a geodesic error of *zero*, while those which are falsely classified, often show a high error.

4. Examples of the MNIST superpixels dataset

Figure 3 shows examples for each class and each graph representation of the used MNIST datasets. The first and fourth row show grid graphs with neighborhoods of 5×5, the second and fifth row show the graphs from 75 superpixels, which are used as input for our superpixel experiment, and the third and sixth row show results of one pooling step.

5. GPU algorithm of the backward step

In order to train our B-spline kernels, a backward step for the convolution operator has to be defined, which can be used by the backpropagation algorithm for gradient estimation. The algorithm of this backward step is outlined in Algorithm 1. The parallelization scheme of the backward step is equivalent to the scheme of the forward algorithm presented in the paper. We parallelize over E edges and M_{out} output features and compute gradients $\tilde{\mathbf{W}}$, $\tilde{\mathbf{F}}_{\text{in}}$ with respect to weights and input features, given gradients $\tilde{\mathbf{F}}_{\text{out}}$ with respect to the output features. Analogously to the forward step, we first gather the edgewise gradients with respect to $\tilde{\mathbf{F}}^{E}_{\text{out}}$ from $\tilde{\mathbf{F}}_{\text{out}}$, with gather index given by the origin node of the edge. Then, we compute partial gradients with respect to $\mathbf{W}$ and $\mathbf{F}_{\text{in}}$ and sum them up, using an atomic-add operation. Last, we scatter-add the edgewise gradients $\tilde{\mathbf{F}}^{E}_{\text{in}}$ to $\tilde{\mathbf{F}}_{\text{in}}$, given the origin node of each edge as index. We obtain gradients $\tilde{\mathbf{W}}$, which can be used for SGD weight update of $\mathbf{W}$, and $\tilde{\mathbf{F}}_{\text{in}}$ which will be backpropagated further to previous layers.

References

[1] F. Bogo, J. Romero, M. Loper, and M. J. Black. FAUST: Dataset and evaluation for 3D mesh registration. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2014.

[2] I. S. Dhillon, Y. Guan, and B. Kulis. Weighted graph cuts without eigenvectors: A multilevel approach. *IEEE Trans. Pattern Anal. Mach. Intell*, 29:1944–1957, 2007.

[3] F. Monti, D. Boscaini, J. Masci, E. Rodolà, J. Svoboda, and M. M. Bronstein. Geometric deep learning on graphs and manifolds using mixture model CNNs. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2017.

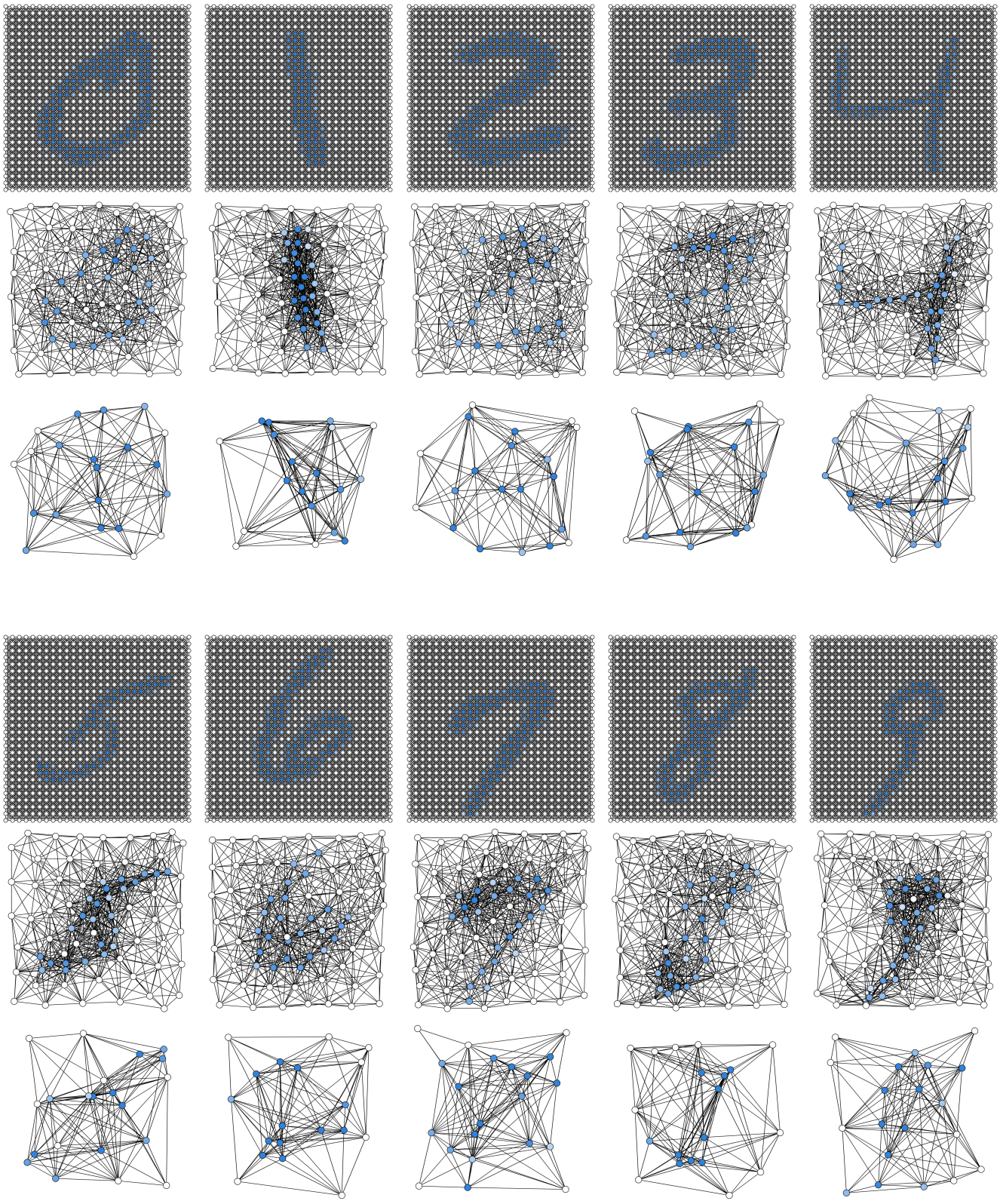


Figure 3: Examples for all classes of the used MNIST graphs: Shown are grid graphs with 5×5 neighborhoods, the graphs represented by 75 superpixel and the pooled versions of that graphs.

Algorithm 1 Backward step of geometric convolution with B-spline kernels

Input:
N: Number of nodes
M_{in}: Number of input features per node
M_{out}: Number of output features per node
$s = (m+1)^d$: Number of non-zero $B_{\mathbf{p}}$ for one edge
$\mathbf{W} \in \mathbb{R}^{K \times M_{\text{in}} \times M_{\text{out}}}$: Trainable weights
$\mathbf{B} \in \mathbb{R}^{E \times s}$: Basis products of s weights for each edge
$\mathbf{P} \in \mathbb{N}^{E \times s}$: Indices of s weights in $\mathbf{W}$ for each edge
$\mathbf{F}_{\text{in}}^{E} \in \mathbb{R}^{E_{\text{in}}}$: Edge-wise input features
$\tilde{\mathbf{F}}_{\text{out}} \in \mathbb{R}^{N \times M_{\text{out}}}$: Gradient w.r.t. $\mathbf{F}_{\text{out}}$
Output:
$\tilde{\mathbf{W}} \in \mathbb{R}^{K \times M_{\text{in}} \times M_{\text{out}}}$: Gradient w.r.t. $\mathbf{W}$
$\tilde{\mathbf{F}}_{\text{in}} \in \mathbb{R}^{N \times M_{\text{in}}}$: Gradient w.r.t. $\mathbf{F}_{\text{in}}$

Gather $\tilde{\mathbf{F}}_{\text{out}}^{E}$ from $\tilde{\mathbf{F}}_{\text{out}}$ based on origin nodes of edges
Parallelize over $e \in \{1, ..., E\}$ and $o \in \{1, ..., M_{\text{out}}\}$:
  for each $i \in \{1, ..., M_{\text{in}}\}$ **do**
    for each $p \in \{1, ..., s\}$ **do**
      $w \leftarrow \mathbf{W}[\mathbf{P}[e,p], i, o]$
      $g \leftarrow \tilde{\mathbf{F}}_{\text{out}}^{E}[e,o] \cdot w \cdot \mathbf{B}[e,p]$
      $\tilde{\mathbf{F}}_{\text{in}}^{E}[e,i] \leftarrow \tilde{\mathbf{F}}_{\text{in}}^{E}[e,i] + g$ (atomic add)
      $g \leftarrow \tilde{\mathbf{F}}_{\text{out}}^{E}[e,o] \cdot \mathbf{F}_{\text{in}}^{E}[e,i] \cdot \mathbf{B}[e,p]$
      $\tilde{\mathbf{W}}[\mathbf{P}[e,p], i, o] \leftarrow \tilde{\mathbf{W}}[\mathbf{P}[e,p], i, o] + g$ (atomic add)
    end for
  end for
Scatter-add $\tilde{\mathbf{F}}_{\text{in}}^{E}$ to $\tilde{\mathbf{F}}_{\text{in}}$ based on target nodes of edges
Return $\tilde{\mathbf{F}}_{\text{in}}$, $\tilde{\mathbf{W}}$

[4] P. Sen, G. Namata, M. Bilgic, and L. Getoor. Collective classification in network data. *AI Magazine*, 29(3):93–106, 2008.