Algorithm 3 Approximating Neural Persistence of Convolutions per filter

Require: filter $H \in \mathbb{R}^{p \times q}$; number of input and output neurons as m, n

$\mathcal{T} \leftarrow \emptyset$ ▷ Initialize set of tuples for persistence diagram
$\tau \leftarrow m + n, \;\; t \leftarrow 0, \;\; i \leftarrow 0$ ▷ Initialize number of tuples, tuple counter, weight index
$h_{\max} \leftarrow \max_{h \in H} |h|$ ▷ Determine largest absolute weight
$H' \leftarrow \{|h|/h_{\max} \mid h \in H\}$ ▷ Transform weights for filtration
$s \leftarrow \text{sort}(\text{vec}(H'))$ ▷ Sort weights in descending order
$H'_c \leftarrow \{h'_{0,0}, h'_{0,q-1}, h'_{p-1,0}, h'_{p-1,q-1}\}$ ▷ Determine the set of all corner weights of filter H'
$\mathcal{T} \leftarrow (1, 0), \;\; t \leftarrow t + 1$ ▷ Add tuple for surviving component
for $h'_c \in H'_c$ **do** ▷ Each corner of H' merges components
    $\mathcal{T} \leftarrow (1, h'_c), \;\; t \leftarrow t + 1$
end for
while 1 **do** ▷ Create the remaining tuples (Approximation step)
    $n' = n - \text{Ind}(s[i] \in H'_c)$ ▷ if current weight is a corner weight, write one less tuple
    if $t + n' \leq \tau$ **then** ▷ if there are at least n' more tuples, set their merge value to $s[i]$
        repeat n' times
            $\mathcal{T} \leftarrow (1, s[i])$ ▷ approximative as $s[i]$ does not always add n' merges due to loops
        $t \leftarrow t + n', \;\; i \leftarrow i + 1$
    else ▷ otherwise, process the remaining tuples similarly
        repeat $(\tau - t)$ times
            $\mathcal{T} \leftarrow (1, s[i])$
        break
    end if
end while
return $\|\mathcal{T}\|_p$ ▷ Compute norm of approximated persistence diagram

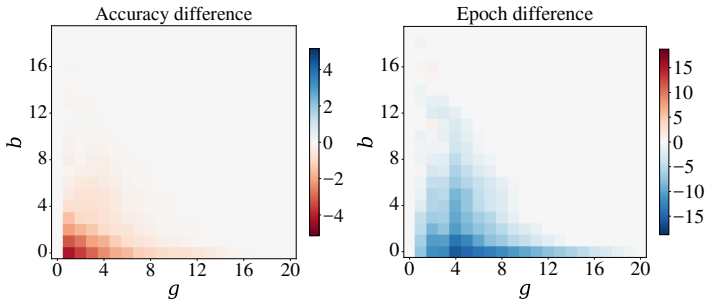
is no configuration that results in an increased accuracy. This empirically confirms our theoretical scepticism towards naively applying our edge-focused filtration scheme to CNNs.

A.5 Relationship between neural persistence and validation accuracy

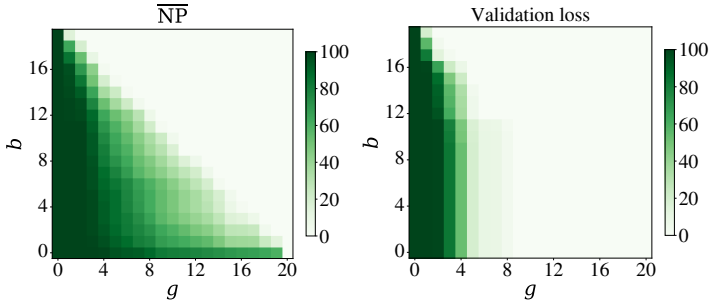
Motivated by Figure 2, which shows the different 'regimes' of neural persistence for a perceptron network, we investigate a possible correlation of (high) neural persistence with (high) predictive accuracy. For deeper networks, we find that neural persistence measures structural properties that arise from different parameters (such as training procedures or initializations), and *no* correlation can be observed.

For our experiments, we constructed neural networks with a *high* neural persistence prior to training. More precisely, following the theorems in this paper, we initialized most weights of each layer with very low values and reserved high values for very few weights. This was achieved by sampling the weights from a *beta distribution* with $\alpha = 0.005$ and $\beta = 0.5$. Using this procedure, we are able to initialize [20,20,20] networks with $\overline{\text{NP}} \approx 0.90 \pm 0.003$ compared to the same networks that have $\overline{\text{NP}} \approx 0.38 \pm 0.004$ when initialized by Xavier initialization. The mean validation accuracy of these untrained networks on the 'Fashion-MNIST' data set is 0.10 ± 0.01 and 0.09 ± 0.03, respectively.

Figure A.10 depicts how both types of networks converge to similar regimes of validation accuracy, while the mean normalized neural persistence achieved at the end of the training varies. For networks initialized with high $\overline{\text{NP}}$ (Figure A.10, left) the validation accuracy of networks with final $0.9 \leq \overline{\text{NP}} \leq 0.95$ ranges from 0.098 (not shown) to 0.863. For Xavier initialization (Figure A.10, right), the lack of correlation can also be observed. Furthermore, comparing the two plots, there are no clear advantages in initializing networks with high $\overline{\text{NP}}$. This observation further motivates the proposed *early stopping criterion*, which checks for *changes* in the $\overline{\text{NP}}$ value, and considers stagnating values to be indicative of a trained network.



(a) Accuracy and epoch differences



(b) Number of triggers

Figure A.9: Additional visualizations for the 'Fashion-MNIST' data set, following the preliminary examination of convolutional layers. Here, the approximated neural persistence calculation for the first convolutional layer was used. However, we also ran few runs of the same experiment using the exact method which showed the same results. Employing the second convolutional layer or both did not improve this result.

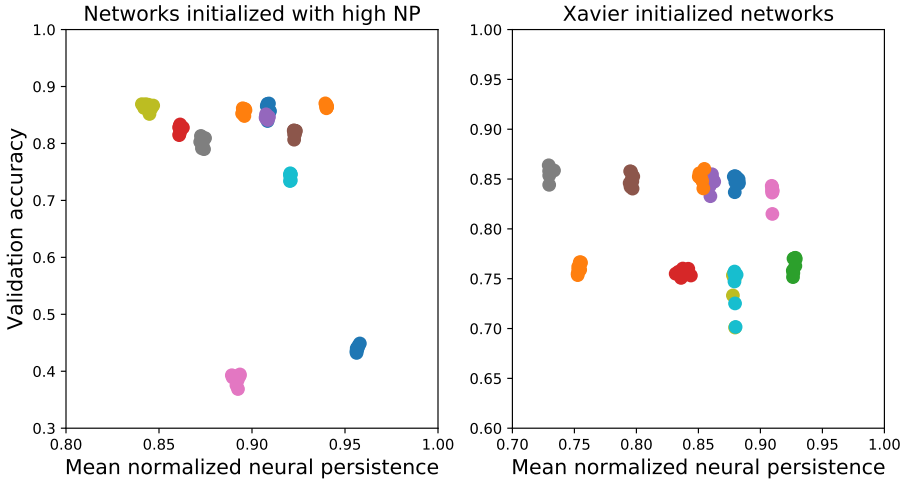


Figure A.10: Each cluster of points represent the last two training epochs (sampled every quarter epoch) of a [20,20,20] network trained on the 'Fashion-MNIST' data set. We observe no correlation between validation accuracy and normalized total persistence

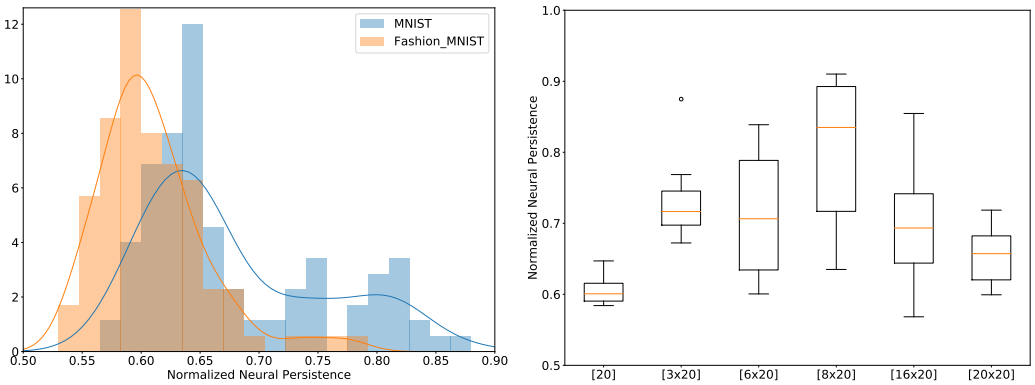


Figure A.11: (left) Histogram of the final normalized neural persistence of a $[50, 50, 20]$ network for 100 runs and 25 epochs of training. (right) Normalized neural persistence after 15 epochs of training on MNIST for different architectures with increasing depth. Deeper architectures are denoted as $[n \times 20]$ where n is the number of hidden layers.

A.6 Neural persistence for different data distributions and deeper FCN architectures

Neural persistence captures information about different data distributions during training. The weights tuned via backpropagation are directly influenced by the input data (as well as their labels) and neural persistence tracks those changes. To demonstrate this, we trained the same architecture , i.e. $[50, 50, 20]$, on two data sets with the same dimensions but different properties: MNIST and 'Fashion-MNIST'. Each data set has the same image size (28×28 pixels, one channel) but lay on different manifolds. Figure A.11 (left) shows a histogram of the mean normalized neural persistence ($\overline{\text{NP}}$) after 25 epochs of training over 100 different runs. The distributions have a similar shape but are shifted, indicating that the two datasets lead the network to different topological regimes.

We also investigated the effect of depth on neural persistence. We selected a fixed layer size (20 hidden units) and increased the number of hidden layers. Figure A.11 (right) depicts the boxplots of mean $\overline{\text{NP}}$ for multiple architectures after 15 epochs of training on MNIST. Adding layers initially increases the variability of $\overline{\text{NP}}$ by enabling the network to converge to different regimes (essentially, there are many more valid configurations in which a trained neural network might end up in). However, this effect is reduced after a certain depth: networks with deeper architectures exhibit less variability in $\overline{\text{NP}}$.