Algorithm 1 Creation of parameters for one Eedn convolutional layer

Input: Input feature count f_i, filter height r rows, filter width c columns, output feature count f_o (number of filters), and number of groups G (where each group is a disjoint subset of the filters, applied to a disjoint subset of the input features),

Output: Permuted ordering of input features $\mathbf{P}$, continuous weight matrix $\mathbf{W}^h$, discretized weight matrix $\mathbf{W}$, weight mask matrix $\mathbf{M}$ to enforce groups, vector of biases $\mathbf{B}$

```
Assert(f_i rc/G = ⌈f_i rc/G⌉)  // Ensure total filter inputs are evenly divisible across groups
Assert(f_o/G = ⌈f_o/G⌉)  // Ensure output features are evenly divisible across groups
Assert(f_i rc/G ≤ 128)  // Ensure filter support region fits on one core
Assert(f_o/G ≤ 128)  // Ensure output features per group fits on one core
P ← Randperm(f_i)  // Create list of integers from 1 to f_i, randomly ordered
B = zeros(f_o)  // Vector of zeros of length f_o
W^h ← Rand(r, c, f_i, f_o)  // rcf_i × f_o matrix of random real numbers drawn from [−1, 1]
M ← Createmask(r, c, f_i, f_o, G)  // Create rcf_i × f_o block diagonal matrix with blocks of
    size rcf_i/G × f_o/G consisting of all 1s, and with 0s everywhere else; block size corresponds to
    input features per group × output features per group
W^h ← W^h ∘ M  // enforce groups by zeroing non-block diagonal values
W ← Round(W^h)  // Initialize weights to use in forward and backward pass
```

Algorithm 2 One iteration of Eedn training algorithm for a network with K layers. Blue text indicates steps not standard in conventional deep learning

Input: Network parameters $\{\mathbf{P}, \mathbf{W}, \mathbf{W}^h, \mathbf{M}, \mathbf{B}\}_{k=1}^{K}$ (set in Algorithm 1); network input $\mathbf{X}_0$; $\mathbf{F}$, a list of the class each output feature is assigned to predict (assignments are made randomly such that each class has an equal number of features); class labels $\hat{\mathbf{Y}}$

Output: Trained network parameters

Forward propagation:

for $k = 1$ to K **do**
    $\mathbf{X}_{k-1} \leftarrow$ Permute$(\mathbf{X}_{k-1}, \mathbf{P}_k)$ // Permute features of $\mathbf{X}_{k-1}$ according to $\mathbf{P}_k$
    $\mathbf{X}_k \leftarrow$ Forward$(\mathbf{X}_{k-1}, \mathbf{W}_k, \mathbf{B}_k)$ // Layer forward pass (see eqs. 1, 2, and 4)
end for
$\mathbf{Y} \leftarrow$ ComputeVotes$(\mathbf{X}_K, \mathbf{F})$ // Each output feature at each spatial location casts one vote for its assigned class (determined by $\mathbf{F}$) if it is spiking, creating the network's prediction $\mathbf{Y}$.

Loss:

$L, \frac{\partial \mathbf{L}}{\partial \mathbf{Y}} \leftarrow$ ComputeLoss$(\mathbf{Y}, \hat{\mathbf{Y}})$ // Compute loss, L, and loss gradient, $\frac{\partial \mathbf{L}}{\partial \mathbf{Y}}$. The softmax cross entropy loss function was used for this work.

Backward propagation:

$\frac{\partial \mathbf{L}}{\partial \mathbf{X}_K} \leftarrow$ BackwardComputeVotes$(\frac{\partial \mathbf{L}}{\partial \mathbf{Y}}, \mathbf{F})$ // Backward pass through voting step.
for $k = K$ to 1 **do**
    $\frac{\partial \mathbf{L}}{\partial \mathbf{X}_k}, \frac{\partial \mathbf{L}}{\partial \mathbf{W}_k}, \frac{\partial \mathbf{L}}{\partial \mathbf{B}_k} \leftarrow$ Backward$(\frac{\partial \mathbf{L}}{\partial \mathbf{X}_{k+1}}, \mathbf{W}_k, \mathbf{B}_k)$ // Layer backward pass (see eq. 5)
    $\frac{\partial \mathbf{L}}{\partial \mathbf{X}_k} \leftarrow$ Permute$(\frac{\partial \mathbf{L}}{\partial \mathbf{X}_k}$, Inverse$(\mathbf{P}_k))$ // Reverse feature permutation of forward pass
end for

Parameter update:

for $k = 1$ to K **do**
    $\frac{\partial \mathbf{L}}{\partial \mathbf{W}_k} \leftarrow \frac{\partial \mathbf{L}}{\partial \mathbf{W}_k} \circ \mathbf{M}$ // Enforce groups
    $\mathbf{B}_k \leftarrow$ Update$(\mathbf{B}_k, \frac{\partial \mathbf{L}}{\partial \mathbf{B}_k})$ // Update bias.
    $\mathbf{W}_k^h \leftarrow$ Update$(\mathbf{W}_k^h, \frac{\partial \mathbf{L}}{\partial \mathbf{W}_k})$ // Update continuous weights using gradient computed with respect to discretized weights.
    $\mathbf{W}_k^h \leftarrow$ Clip$(\mathbf{W}_k^h, -1, 1)$ // Snap values outside of range $[-1, 1]$ to nearest value in range.
    $\mathbf{W}_k \leftarrow$ Cast$(\mathbf{W}_k^h, \mathbf{W}_k)$ // Cast weights to $\{-1, 0, 1\}$ values using hysteresis (see eq. 6).
end for