# CSNNs: Unsupervised, Backpropagation-free Convolutional Neural Networks for Representation Learning

Bonifaz Stuhr
*University of Applied Sciences Kempten*
bonifaz.stuhr@hs-kempten.de

Jürgen Brauer
*University of Applied Sciences Kempten*
juergen.brauer@hs-kempten.de

Author's version of the paper. Final version presented at
18th IEEE International Conference on Machine Learning and Applications (ICMLA).Boca Raton, Florida / USA. 2019.

***Abstract*—This work combines Convolutional Neural Networks (CNNs), clustering via Self-Organizing Maps (SOMs) and Hebbian Learning to propose the building blocks of Convolutional Self-Organizing Neural Networks (CSNNs), which learn representations in an unsupervised and Backpropagation-free manner. Our approach replaces the learning of traditional convolutional layers from CNNs with the competitive learning procedure of SOMs and simultaneously learns local masks between those layers with separate Hebbian-like learning rules to overcome the problem of disentangling factors of variation when filters are learned through clustering. We investigate the learned representation by designing two simple models with our building blocks, achieving comparable performance to many methods which use Backpropagation, while we reach comparable performance on Cifar10 and give baseline performances on Cifar100, Tiny ImageNet and a small subset of ImageNet for Backpropagation-free methods.**

***Index Terms*—convolutional, self-organizing, neural networks, CSNNs, CNNs, maps, SOMs, hebbian, local, deep, unsupervised, representation, learning**

## I. Introduction

A well-known downside of many successful Deep Learning approaches like Convolutional Neural Networks (CNNs) is their need for big, labeled datasets. Obtaining these datasets can be very costly and time-consuming or the required quantity is even not available due to restrictive conditions. In fact the availability of big datasets and the computational power to process this information are two of the main reasons for the success of Deep Learning techniques [1]. This raises the question of why models like CNNs need that much data: Is it the huge amount of model parameters, are training techniques like Backpropagation [2], [3] just too inefficient or is the cause a combination of both?

The fact that neural networks have become both large and deep in recent years, and often consist of millions of parameters, suggests that the cause is a combination of both. On the one hand, large amounts of data are needed, because there are so many model parameters to set. On the other hand, the data may not yet be optimally utilized. The suboptimal utilization of data to date could be related to the following problem, which we call the *Scalar Bow Tie Problem*. Most training strategies of neural networks rely just on one single scalar value (the loss value) which is computed for each training step and then used as a feedback signal for optimizing millions of parameters. In contrast, humans use multiple feedback types to learn, e.g., when they learn representations of objects, different sensor modalities are used. With the Scalar Bow Tie Problem we refer to the current situation for training neural networks. Similar to a bow tie, both ends of the learning setting (the learning agent and the task to be learned) are complex and the only connection between these complex parts is one point (the scalar loss or feedback value).

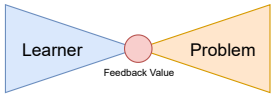


Fig. 1. The Scalar Bow tie Problem. With this term we want to describe the current problematic situation in Deep Learning that today most training approaches for neural networks and agents just rely on a single scalar loss value which is used to subsume the performance of a neural network or agent even for very complex problems.

There exists a current effort of Backpropagation-based methods to overcome the Scalar Bow Tie Problem, e.g., by using multiple loss functions at different output locations of the network (a variant of multi-task learning, i.a. [4]). Furthermore, there exists a current debate to which degree Backpropagation is biologically plausible on a network level (e.g., [5]–[7]), which often highlights that layer-dependent weight updates in the whole model are less likely than more independent updates, like layer-wise ones, and beyond that would require an exact symmetric copy $W^T$ of the upstream weight $W$. In [8] Grossberg describes this problem in detail and refers to it as the *weight transport problem*.

In this work we do not aim to solve all these problems, but want to make a small step away from common Deep Learning methods while keeping some benefits of these models, like modularity and hierarchical structure. We propose a CNN variant with building blocks that learn in an unsupervised, self-organizing and Backpropagation-free manner (CSNN). Within these blocks we combine methods from CNNs [9], Self-Organizing Maps (SOMs) [10] and Hebbian Learning [11].

Our main contributions are as follows:

- We propose an unsupervised, Backpropagation-free learning algorithm, which uses two learning rules to update weights layer-wise without an explicitly defined loss

Author's version of the paper. Final version presented at
18th IEEE International Conference on Machine Learning and Applications (ICMLA).Boca Raton, Florida / USA. 2019.

function and thereby reduces the Scalar Bow tie Problem.
- This learning algorithm is used to train CSNN models, with which we achieve comparable performance to many models trained in an unsupervised manner.
- We overcome a fundamental problem of SOMs trained on image patches, by presenting two types of weight masks, to mask input and neuron activities.
- We propose a multi-headed version of our building blocks, to further improve performance.
- To the best of our knowledge, our approach is the first to combine the successful principles of CNNs, SOMs and Hebbian Learning into a single Deep Learning model.

## II. Related Work

Our work relates to methods of unsupervised representation learning. Due to the usage of SOMs, it further relates to clustering methods [12]; in fact one could change the SOM learning algorithm with another clustering algorithm. Moreover, we can relate our proposed local learning rules in Section III-C to other local learning rules (e.g., [11], [13], [14]). To stay on track, we focus on methods for unsupervised, Backpropagation(-free) representation learning as follows:

### A. Unsupervised Representation Learning via Backpropagation

Recently unsupervised representation learning through auto-encoders [15]–[18], and in general self-supervised approaches [19], showed promising results. Zhang et al. [15] improved the approach of Gidaris et al. [20] by not only predicting rotations, but by auto-encoding a variety of transformations between images. In [18] a hierarchical, disentangled object representation and a dynamics model for object parts is learned from unlabeled videos by a novel formulation, which can be seen as a conditional, variational auto-encoder with additional motion encoder. Hjelm et al. [21] proposed *Deep InfoMax*, which defines an objective function that maximizes mutual information between global features created from smaller, random crops of local features and the full sets of local features, while matching a prior distribution to capture desired characteristics like independence. In [22] an objective is defined to search neighborhoods with high class consistency anchored to individual training samples, considering the high appearance similarity among the samples via a softmax function. The results in [23] support our view of the learning problem - the Scalar Bow Tie Problem - since a concept is introduced which uses an evolutionary algorithm (AutoML) to obtain a better weighting between individual losses in a multi-modal, multi-task loss setting for self-supervised learning. Yang et al. [24] learns patch features by a discriminative loss and an image-level feature learning loss. The discriminative loss is designed to pull similar patches together and vice versa. The image-level loss pulls the real sample and the surrogate positive samples (random transformations) together while pushing hard negative samples away and serves as an image-level guidance for patch feature generation. To learn representations with minimal description lengths Bizopoulos et al. [25] used i.a. sparse activation functions and minimizes the reconstruction loss dependent on the input and the summed up reconstructions of each individual weight used to reconstruct the entire input. In [26] a Siamese Network with a novel instance feature-based softmax embedding method is used to learn data augmentation invariant and instance spread-out features, by considering this problem as a binary classification problem via maximum likelihood estimation. Thereby only the augmented sample and no other instances should be classified as the unaugmented instance. Donahue et al. [27] showed that GANs can reach state-of-the-art performance by training a *BigBiGAN* on unlabeled ImageNet, and afterwards learning a linear classifier with labels on the latent variable predicted by the encoder of the BigBiGAN, given the input data.

### B. Unsupervised, Backpropagation-Free Representation Learning

While there are many advances in unsupervised methods using Backpropagation, in comparison there are only a few efforts to propose Backpropagation-free alternatives. Chan et al. [28] used Principal Component Analysis (PCA) in a convolutional manner to learn two-stage filter banks, which are followed by binary hashing and block histograms to compress the representation. In a similar way Hankins et al. [29] used SOMs and additionally introduced pooling layers to shrink SOM-activations along the feature dimension. Coates et al. [30] studied hyperparameter choices and preprocessing strategies on several single-layer networks learned on convolutional patches, including sparse auto-encoders, sparse restricted Boltzmann machines, K-means clustering, and Gaussian mixture models and showed, e.g., general performance gains when the representation size is increased. In [31] an adaptive-subspace SOM is used for saliency-based, invariant representation learning for image classification. Krotova et al. [32] introduced a learning algorithm motivated by Hebbian Learning to learn early feature detectors by utilizing global inhibition in the hidden layer. In [33] a Nonnegativity Orthogonal Matching Pursuit (NOMP) encodes representations by selecting a small number of atoms from the feature dictionary, such that their linear combination approximates the input data vector best under an nonnegativity constraint. This constraint mitigates OMPs stability issue and adds resistance to overfitting. Furthermore, there exists work on unsupervised spiking neural network models using spike-timing dependent plasticity learning rules (e.g., [34]).

## III. CSNN

This section describes the key methods of CSNNs. The proposed blocks for each method and their interactions are summarized in Fig. 2.

### A. Convolutional, self-organizing Layer

The convolutional, self-organzing layer (sconv) follows closely the convolutional layer of standard CNNs. The entire input is convolved patchwise with each learned filter:

$$y_{m,n,i} = \mathbf{p}_{m,n} \cdot \mathbf{w}_i \tag{1}$$

Author's version of the paper. Final version presented at
18th IEEE International Conference on Machine Learning and Applications (ICMLA).Boca Raton, Florida / USA. 2019.

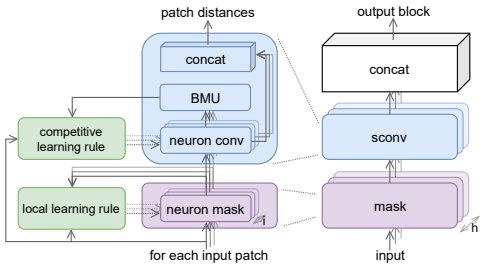


Fig. 2. (left) mask and sconv layer computations for a single patch: First each patch is masked for each neuron (i). The masked outputs and the input patch will be used in the local learning process. Then the sconv layer computes the convolution distances and the BMU. The BMU and the input patch will be used in competitive learning. The distances are concatenated to the patchs spatial activation vector in the convolution output block. (right) Multi-Head masked sconv: Multiple mask and sconv layers (h) are applied on the entire input and the resulting output blocks are concatenated in the feature dimension.

where $\mathbf{p}_{m,n}$ is the image patch centered at position $m, n$ of the input, $\mathbf{w}_i$ is the $i$-th filter of the layer and $y_{m,n,i}$ is the activation at position $m, n, i$ of the output. The only difference to a standard convolutional layer is that the weights are learned by a competitive, self-organizing optimization procedure. The output of the layer for one image patch are the distances for each neuron in a SOM. Each neuron of the SOM corresponds to one filter in a CNN, and each distance output of a SOM-neuron corresponds to a specific feature $y_{m,n,i}$ of the feature map $i$ at the center of the patch $m, n$ after the convolution operation was applied. The central difference to a standard CNN is that the CSNN only uses local learning rules. It is trained unsupervised and bottom-up fashion in contrast to the top-down supervised backpropagation approach. Analog to CNNs the proposed modular layers can be combined with other layers in Deep Learning architectures.

### B. Competitive Learning Rule

Most learning rules for SOMs require a best matching unit (BMU) in order to compute the weight change $\Delta w$. Since we use a convolution as distance metric the index of the BMU for one patch is defined as:

$$c_{m,n} = \underset{i}{\operatorname{argmax}}\{y_{m,n,i}\} \tag{2}$$

where $c_{m,n}$ is the index of the BMU in the (2D) SOM-grid for patch $m, n$.

To allow self-organization, learning rules for SOMs require a neighborhood function to compute the neighborhood coefficients from all the other neurons in the grid to the BMU. A common type of neighborhood function is the Gaussian:

$$h_{m,n,i}(t) = \exp\left(\frac{-d_A(k_{m,n,c_{m,n}}, k_{m,n,i})^2}{2\delta(t)^2}\right) \tag{3}$$

where $k_{m,n,c_{m,n}}$ and $k_{m,n,i}$ are the coordinates of the BMU and the $i$-th neuron in the SOM-grid. These coordinates depend on the dimensionality of the grid. For the distance function $d_A$ we use the Euclidean distance and $\delta(t)$ is a hyperparameter to control the radius of the Gaussian which could change with the training step ($t$). Then $h_{m,n,i}(t)$ is the neighborhood coefficient of neuron $i$ to the center (the BMU) $c_{m,n}$ for the patch $m, n$. In our experiments we did not investigate other neighborhood functions.

Now the weight update for one patch can be defined as:

$$\Delta\mathbf{w}_{m,n,i}(t) = a(t)h_{m,n,i}(t)\mathbf{p}_{m,n} \tag{4}$$

$$\mathbf{w}_{m,n,i}(t) = \frac{\mathbf{w}_{m,n,i}(t) + \Delta\mathbf{w}_{m,n,i}(t)}{\|\mathbf{w}_{m,n,i}(t) + \Delta\mathbf{w}_{m,n,i}(t)\|} \tag{5}$$

where $\|...\|$ is the Euclidean norm used to get the positive effects of normalization such as preventing the weights from growing too much. $a(t)$ is the learning rate which could change over the course of the training.

For an entire image, the final weight change for the weight vector of a SOM-neuron is computed by taking the mean over all patch weight updates:

$$\Delta\mathbf{w}_i(t) = \frac{a(t)}{mn}\sum_{m,n} h_{m,n,i}(t)\mathbf{p}_{m,n} \tag{6}$$

In batch training the mean of all patches in a batch is taken.

The use of other distance metrics (like the L1- or L2-norm) may make it necessary to switch the computation of the BMU (e.g., to argmin) and the learning rule.

### C. Mask Layers

We argue that given the learning rule (6) a SOM-neuron is not able to disentangle factors of variation, which is a key property of Deep Learning architectures [35]. In other words, it is not able to choose which part of the input it wants to give more attention and which it wants to ignore. The SOM just tries to push its neurons around to fit the dataset best. This may lead to poor performance, because a neuron is not able to learn abstract features which can be seen as concepts. Even in higher layers a SOM-neuron just processes the input from a collection of SOM-neurons of the previous layer in its receptive field. Therefore, the network forms a hierarchy of collections rather than a hierarchy of concepts. Furthermore, a neuron in a deeper layer with a small receptive field of e.g., $3\times3$ needs to compute the distance between its $3\times3\times256$ vector and the input patch, if there are 256 SOM-neurons in the previous layer. Because the weight update shifts all the values of the weights from the BMU and its neighbors in one direction to the input at once, it seems unlikely that the SOM-neurons learn distinct enough features for data with a high amount of information such as images. This argument goes hand in hand with Dundar et al. [36], where similar observations are made for k-means clustering.

To allow the network to learn some kind of concept, we propose two types of separately learned mask vectors shown in Fig. 3. The first mask (a) enables each neuron to mask its input values with a vector of the same dimension as the input patch and the SOM-neuron's weight vector. Each CSNN-neuron multiplies the image patch with its mask and convolves the masked patch with the weight vector of the SOM neuron:

$$\hat{\mathbf{y}}_{m,n,i} = \mathbf{p}_{m,n} \circ \mathbf{m}_i \tag{7}$$

Author's version of the paper. Final version presented at
18th IEEE International Conference on Machine Learning and Applications (ICMLA).Boca Raton, Florida / USA. 2019.

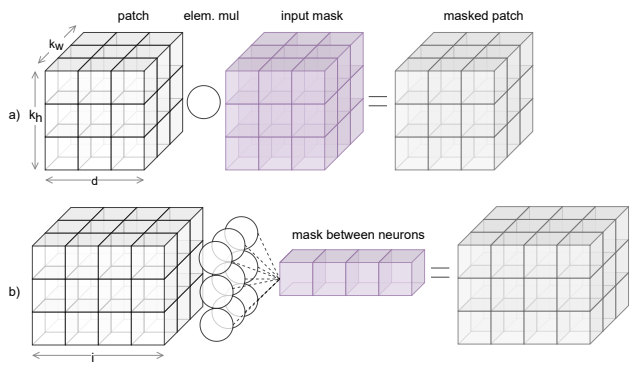


Fig. 3. a) Input mask: The Hadamard product between an input patch and a mask of same size. b) Mask between layers: The mask, with its length equal to the number of filters in the previous layer, is multiplied element-wise with every spatial activation vector from the patch.

$$y_{m,n,i} = \hat{\mathbf{y}}_{m,n,i} \cdot \mathbf{w}_i \tag{8}$$

where $\mathbf{m}_i$ is the mask of neuron $i$, $\hat{\mathbf{y}}_{m,n,i}$ is the masked patch for neuron $i$ and $\circ$ denotes the Hadamard product.

The second mask (b) lies between the neurons of two layers. Therefore, it requires $k_w \times k_h$ fewer parameters then the first (input) mask, when $k_w$ and $k_h$ are the kernel's sizes. The mask is defined as:

$$\hat{\mathbf{y}}_{m_h,n_w,i} = \mathbf{p}_{m_h,n_w} \circ \mathbf{n}_i \tag{9}$$

where $\mathbf{n}_i$ is the mask element-wise multiplied in depth $k_w \times k_h$ times. With this mask we want to enable each neuron to choose from which neuron in the previous layer it wants to get information. Mask (a) is used for the input, and mask (b) between each sconv layer.

We note that we still use the unmasked input for the training of the SOM-weight vectors the mask is only used to compute the convolution distances and to determine the BMU. Moving the BMU and its neighbors into unmasked input direction leads to a more stable learning process. This is due to the fact that the self-organization does not rely too heavily on the higher mask coefficients. Such a situation could drive the weight vector of a single SOM-neuron into a direction from which it can hardly adapt to other new, but similar input patches. Thereby the neuron could get stuck in a small dead end. Furthermore, this gives the mask the opportunity to learn new regions typical for a particular input type even when the mask tends to ignore these regions of the input. This is because the SOM moves in the direction of the unmasked input, which opens the opportunity for the mask to increase its corresponding coefficients.

### *D. Local Learning Rules*

To learn the input mask of each neuron, we propose two simple local learning rules which are inspired by the Generalized Hebbian Algorithm [13] (GHA). These procedures show good results in our experiments, but there are certainly other local learning rule alternatives that could guide the training process of the masks as well.

Below we derive our final learning rules by discussing the following equations within the context of the CSNN, where $e$ indicates that we present the element-wise form of the learning rules. For the original inviolate versions we refer to the literature.

$$\Delta h_e(\mathbf{p}_{m,n}, \hat{\mathbf{y}}_{m,n,i}) = \hat{\mathbf{y}}_{m,n,i} \circ \mathbf{p}_{m,n} \tag{10}$$

$$\Delta o_e(\mathbf{p}_{m,n}, \hat{\mathbf{y}}_{m,n,i}, \mathbf{m}_i) = \hat{\mathbf{y}}_{m,n,i} \circ (\mathbf{p}_{m,n} - \hat{\mathbf{y}}_{m,n,i} \circ \mathbf{m}_i) \tag{11}$$

$$\Delta m_{es}(\mathbf{p}_{m,n}, \hat{\mathbf{y}}_{m,n,i}, \mathbf{m}_i) = h_e([\mathbf{p}_{m,n} - \gamma \sum_k \hat{\mathbf{y}}_{m,n,k} \circ \mathbf{m}_k], \hat{\mathbf{y}}_{m,n,i}, \mathbf{m}_i) \tag{12}$$

$$\Delta m_{ec}(\mathbf{p}_{m,n}, \hat{\mathbf{y}}_{m,n,i}, \mathbf{m}_i) = o_e([\mathbf{p}_{m,n} - \gamma \sum_{k<i} \hat{\mathbf{y}}_{m,n,k} \circ \mathbf{m}_k], \hat{\mathbf{y}}_{m,n,i}, \mathbf{m}_i) \tag{13}$$

$$\Delta \mathbf{m}_i(t) = \frac{a(t)}{mn} \sum_{m,n} [\Delta m_{es}(\mathbf{p}_{m,n}, \hat{\mathbf{y}}_{m,n,i}, \mathbf{m}_i)] \tag{14}$$

$$\Delta \mathbf{n}_i(t) = \frac{a(t)}{mnhw} \sum_{m,n,h,w} [\Delta m_{es}(\mathbf{p}_{m_h,n_w}, \hat{\mathbf{y}}_{m_h,n_w,i}, \mathbf{n}_i)] \tag{15}$$

Eq. (10) describes simple Hebbian Learning in vector notation. It implements the principle "neurons that fire together, wire together", where $\mathbf{p}_{m,n}$ is the pre- and $\hat{\mathbf{y}}_{m,n,i}$ is the postsynaptic neuron activities vector. This principle per se is useful for mask learning, because the procedure could find mask values, which indicate the connection strengths between mask input and output. However, Hebb's rule is unstable. If there are any dominate signals in the network and in our case this is likely due to BMUs the mask values will approach numerical positive or negative infinity rapidly. In fact, it can be shown that the instability of Hebbian Learning accounts for every neuron model [37].

Eq. (11), known as Oja's rule [14], tries to prevent this problem through multiplicative normalization, where the resulting additional minus term can be seen as the forgetting term to control the weight growth. This restricts the magnitude of weights to lie between 0 and 1.0, where the squared sum of weights tends to 1.0 in a single-neuron fully-connected (fc) network [14]. For that reason, using Oja's rule in our case would lead to mask values approaching 1.0, because each mask value has its own input and output due to the element-wise multiplication and is therefore independent of all other mask value updates each mask value forms a one-weight, single-neuron fc network. In order to prevent this tendency to 1.0, we can make the mask value updates dependent on each other.

The GHA [13] makes its updates dependent trough input modification to approximate eigenvectors for networks with more than one output allowing to perform a neural PCA [13] in contrast to Oja's rule, which approximates the first eigenvector for networks with only one output. In [13] one can see that the algorithm is equivalent to performing Oja Learning using a modified version of the input, and a particular weight is dependent on the training of the other weights only through the modifications of the input. If Oja's algorithm is applied to the modified input, it causes the $i-$th output to learn the $i$-th eigenvector [13]. However, our mask values have their own single and independent inputs and outputs.

Author's version of the paper. Final version presented at
18th IEEE International Conference on Machine Learning and Applications (ICMLA).Boca Raton, Florida / USA. 2019.

Therefore, in (12) we use a modified input patch for mask $i$ by simply subtracting the input patch from the sum over mask outputs, which sums the filtered information of all masks when $k$ iterates over all masks. Similar to the GHA we additionally multiply the mask to the output before summing up, which leads to a normalization effect and drives the mask coefficients to the initialization interval $[-1, 1]$. Now each mask update tries to incorporate the missing input information in the output in a competitive manner, where the mask value growth is restricted. This can be seen as a self-organization process for the masks. In our experiments we found that due to the summation over all masks the updates are stable enough to use Hebbian Learning (10), which saves computation time.

In (13) we sum up all $k < i$ and each next mask $\mathbf{m}_i$ sees an output from which the filtered information of the previous masks $\{\mathbf{m}_1, ..., \mathbf{m}_{i-1}\}$ in the grid is subtracted. The hyperparameter $\gamma$ is used to control how much information we want to subtract from the input. A smaller $\gamma < 1.0$ value enables the masks to share more input information with each other.

Eq. (14) shows the final update formula for the input masks (7), where we compute the mean over every patch $\mathbf{p}_{m,n}$. For the second mask type (9) the final update formula is shown in (15). Instead we update here the mask $\mathbf{n}_i$ with the mean over each spatial activation vector $\mathbf{p}_{m_h,n_w}$ of every patch. Experiments show that the update rules (14) and (15) lead to barley better performance for smaller models when using the input modification of (13) and $\gamma = 0.5$. But this equation requires the Oja Rule and sometimes bigger neighborhood coefficients especially when $\gamma$ is near $1.0$ because of unequal opportunities of masks to learn which may lead to poorer efficiency. With the other input modification (12) and $\gamma = 1.0$ we achieve better performance for our deeper models.

Furthermore, we want to note that in our experiments the mask values are initialized uniformly in $[-1, 1]$ to encourage the model to learn negative mask values, which allow the masks to flip their input. For stability it is recommended that the input is normalized. It is also possible to multiply $\mathbf{m}_i$ by $h_{m,n,i}(t)$ to get similar masks for neighboring neurons, but experiments show that the resulting loss in diversity decreases performance. Therefore we only update the BMU mask.

### E. Multi-Head Masked SConv-Layer

To further improve the performance of our models we do not use one big SOM per layer, but multiple smaller maps. For the output of a layer we concatenate the SOMs outputs in the feature dimension. We use multiple maps for multiple reasons: First, the neuron's mask may (and should) lead to specialization, and in combination with the self-organization process a single SOM is possibly still not able to learn distinct enough concepts. Second, multiple smaller maps seem to learn more distinct features more efficiently than scaling up a single map. This approach is very flexible, in fact, experiments show similar performance between SOMs using 3 big maps or 12 small maps per layer. To further increase diversity we update only the best BMU of all maps per patch. This could lead to maps that will never be updated or to maps where the update is stopped to early during training. We do not use a procedure to prevent these dead maps, because we have not had problems with these during our experiments. In fact, dead maps can often be prevented by scaling up the coefficient of the neighborhood function (3). It is a task for future research to examine dead maps and to investigate whether the use of multiple SOMs in the CSNN model leads to redundant information.

All the presented learning methods can be applied layer-wise. On the one hand, this brings benefits like the opportunity to learn big models by computing layer by layer on a GPU and more independent weight updates compared to gradient descent, which reduces the Scalar Bow tie problem. On the other hand, this independence could lead to fewer "paths" through the network, reducing the capability of the network to disentangle factors of variation. Examining this problem by creating and comparing layer-wise dependent and independent learning rules is an interesting direction for future research, and some work has already been done (e.g., [5]).

### F. Other Layers and Methods

Since we are in the regime of convolutional Deep Learning architectures, there are plenty of other methods we could test within the context of our modules. In our experiments we choose to investigate batch normalization [38] (with no trainable parameters) and max-pooling [1]. Furthermore, SOMs are pretty well studied and there are many improvements over the standard formulation of the SOM, which could be investigated in the future (e.g., the influence of different distance metrics).

## IV. Experiments

To evaluate our methods we design and follow two model architectures throughout the experiments. For training our CSNNs we do not use any fancy strategies like pre-training, decaying rates or regularization. We simply use the zero-mean and unit-variance of the datasets. Our implementation is in TensorFlow and Keras and available at https://github.com/BonifazStuhr/CSNN.

### A. Datasets

**Cifar10** [39] with its $32 \times 32$ images and 10 classes is used to evaluate the importance of the proposed building blocks. We split the validation set into $5000$ evaluation and test samples.

**Cifar100** [39] contains 100 classes with which we test the capability of our representation to capture multiple classes. We split the validation set into $5000$ evaluation and test samples.

**Tiny ImageNet** is used to investigate the capability of our models to learn on a dataset with larger images ($64 \times 64$) containing 200 classes with fewer examples.

**SOMe ImageNet**: We take a subset of ImageNet [40] containing $10100$ training, $1700$ evaluation and $1700$ test samples and 10 classes to test our performance on larger ($256 \times 256$) images. Please refer to our implementation for details.

Author's version of the paper. Final version presented at
18th IEEE International Conference on Machine Learning and Applications (ICMLA).Boca Raton, Florida / USA. 2019.

### B. Evaluation Metrics

To evaluate our models we mainly use the representations to train a **linear (l)**, a **nonlinear (nl)** and a **few-shot (fs) classifier**. To further evaluate the model we compute the percentage of used neurons per batch (**neuron utilization**), try to **reconstruct the input** from the representation and have a look at different **neuron activities** based on their input class. We prove the stability of our learning processes by reporting **10-fold cross-validation** results. Furthermore, we give **random (R) baseline** performances for our models. At this point we want to highlight the importance of such baselines, e.g., Saxe et al. [41] showed that models with random weights can achieve performances often just a few percent worse than sophisticated unsupervised training methods.

### C. Model Architectures and Details

**S-CSNN**: Our small model consists of two CSNN-layers, each is followed by a batch norm and a max-pooling layer to halve the spatial dimension. The first layer uses a $10 \times 10 \times 1$ SOM-grid (1 head with 100 SOM-neurons), stride $2 \times 2$ and input masks (14) the second layer a $16 \times 16 \times 1$ SOM-grid, stride $1 \times 1$ and masks between neurons (15). Both layers use a kernel size of $3 \times 3$ and the padding type "same".

**D-CSNN**: Our deeper model consists of three CSNN-layers, each is followed by a batch norm and a max-pooling layer to halve the spatial dimension. The first layer uses a $12 \times 12 \times 3$ SOM-grid and input masks (14), all remaining layers use masks between neurons (15). The second layer consist of a $14 \times 14 \times 3$ SOM-grid, and the third layer of a $16 \times 16 \times 3$ SOM-grid. All layers use a kernel size of $3 \times 3$, a stride of $1 \times 1$ and the padding type "same". For Tiny ImageNet the layer 1 stride is $2 \times 2$ and for SOMe ImageNet the layer 1 and 2 strides are $3 \times 3$ to keep the representation sizes the same.

**Training**: We simply set the learning rate of the SOM(s) to $0.1$ and the learning rate of the local mask weights to $0.005$ for all layers. The neighborhood coefficients for each layer are set to $(1.0, 1.25)$ for the S-CSNN and $(1.0, 1.5, 1.5)$ for the D-CSNN. The individual layers are learned bottom up, one after another, because its easier for deeper layers to learn, when the representations of the previous layers are fully learned. The steps for which each layer will be learned can be defined in a flexible training interval.

**Classifiers**: Our nl classifier is a three layer MLP (512, 256, num-classes) with batch norm and dropout $(0.5, 0.3)$ between its layers. Our l and fs classifier is simply a fc layer (num-classes). All classifiers use elu activation functions and are trained using the Adam optimizer with standard parameters and batch size 512. To infer the results on the test set we take the training step where the evaluation accuracy of the model was the best (pocket algorithm approach).

### D. Objective Function Mismatch

Fig. 4 shows that all models reach their accuracy peak very early in training. Because this peak is often reached before seeing the entire training set and the models show ongoing SOM-weight convergence, it seems likely that the decrease in performance is due to the objective function mismatch between the self-organization process and the ideal objective function to learn features for classification. This is a common problem in unsupervised learning techniques [42]. Taking another, positive view, one could argue that our models require fewer samples to learn representations (e.g., 2000 samples ($\approx 3\%$) of Cifar10 for the D-CSNN-B1 peak). Even more surprising is the huge jump in accuracy (by up to 62%) after the models have only seen 10 examples. This could be a positive effect of reducing the Scalar Bow tie Problem, but we also want to note that better initialization and not using dropout in the nl classifier leads to better initial performance. Furthermore, we can see that the mask learning rule (12) is superior in this setting for the deeper model and vice versa.

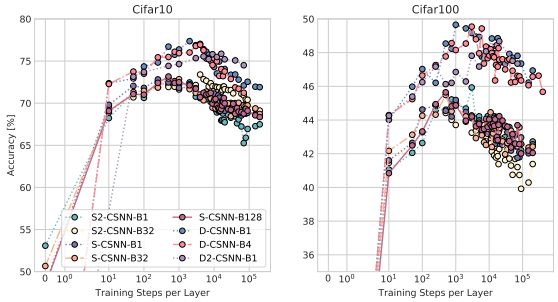


Fig. 4. Accuracies of our models trained with our local mask learning rules (S/D for (12) and S2/D2 for (13)) and different batch sizes B. Every point corresponds to the test dataset accuracy of an nl classifier which was trained on the representation of the CSNN for every sample in the training dataset. The layers of the CSNN where trained layer-wise for the steps shown on the x-axes. Best viewed in color.

### E. Influences of the Methods

The left part of Table I shows the influence of our proposed building blocks. We start with our small and deep model with random sconv weights and no masks (RS-NM). Then we incrementally add and train our methods showing that: 1) Training only the SOM (NM) decreases the performance. 2) Even static, random masks (RM) or noise masks (NO), which change every training step, lead to a huge accuracy increase. We generated these mask from a uniform distribution in $[-1, 1]$. 3) Training masks (no acronym) increases the performance again; even if we leave the SOM untrained (RS), accuracy increases through mask training. 4) Our overall learning increases accuracy by about 10% for the l classifier and 20-65% for the nl classifier compared to the uniform (in $[-1, 1]$) random baseline (R). The low accuracy of the nl classifier for random weights is due to the use of dropout.

In the right part of Table I we show that: 1) Increasing the amount of maps (single M) improves performance (1M, 2M, no acronym). 2) Similar performance can be reached if a higher number of smaller maps is used (M-12M: $(6 \times 6, 7 \times 7, 8 \times 8)$). 3) On the other hand, updating neighboring masks (NH), or updating all map BMUs per step (BMU), or not using batch normalization (NBN)

Author's version of the paper. Final version presented at
18th IEEE International Conference on Machine Learning and Applications (ICMLA).Boca Raton, Florida / USA. 2019.

TABLE I
INFLUENCES OF THE METHODS WITH D-CSNN-B1 AND S2-CSNN-B32

| Model | Cifar10 | | Model | Cifar10 | |
|---|---|---|---|---|---|
| | *l* | *nl* | | *l* | *nl* |
| S2-RSNM | 54.64 | 15.32 | D | $72.66^{+0.40}_{-1.07}$ | $\mathbf{77.21}^{+0.62}_{-0.56}$ |
| D-RSNM | 61.86 | 16.01 | S | $\mathbf{66.43}^{+0.96}_{-1.02}$ | $72.79^{+0.59}_{-0.84}$ |
| S2-NM | 40.37 | 22.32 | D2 | $71.74^{+0.99}_{-1.42}$ | $76.18^{+0.77}_{-1.61}$ |
| D-NM | 43.33 | 12.13 | S2 | $66.18^{+1.17}_{-0.95}$ | $\mathbf{72.82}^{+1.02}_{-1.22}$ |
| S2-RM | 62.89 | 70.42 | D-1M | 69.69 | 71.86 |
| D-RM | 69.86 | 73.69 | D-2M | $72.05^{+0.67}_{-0.99}$ | $76.11^{+0.49}_{-0.57}$ |
| S2-NO | 56.84 | 14.83 | M-12M | 71.28 | 75.77 |
| D-NO | 65.53 | 20.22 | D-BMU | $\mathbf{73.14}^{+0.73}_{-0.70}$ | $76.88^{+0.59}_{-0.90}$ |
| S2-RS | 65.20 | 67.75 | D-NH | 71.70 | 75.08 |
| D-RS | 69.69 | 69.74 | D-NBN | 71.67 | 76.16 |
| S2-R | $56.42^{+1.17}_{-1.46}$ | $52.23^{+2.16}_{-2.48}$ | S2-Aug | 66.32 | **76.90** |
| D-R | $63.36^{+1.28}_{-1.34}$ | $13.44^{+5.25}_{-3.22}$ | D-Aug | $\mathbf{73.73}^{+1.95}_{-1.17}$ | $\mathbf{80.03}^{+1.52}_{-1.05}$ |

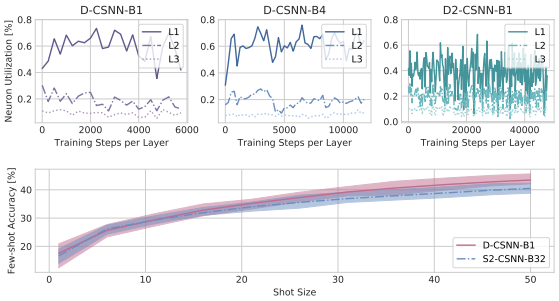


Fig. 5. (left) The neuron utilization of models trained with various batch sizes and mask learning rules on Cifar10. (right) The 10-fold few-shot classifier accuracy of these models with increasing shot sizes from 1 to 50 on Cifar10.

decreases performance. We note that for deeper models batch normalization is necessary to prevent too large values in deeper layers (exploding weights). 4) Using training data augmentation (rotations, shifts and horizontal flip) only to create a bigger representation dataset for the classifier learning increases performance (Aug). Analog to [19] we observe that increasing the representation size improves performance, but in contrast classifier training takes not long to converge.

The upper part of Fig. 5 shows that for every model the neuron utilization the percentage of used neurons per input is getting smaller if we move towards deeper layers, indicating that the SOM-neurons of deeper layers specialize in certain inputs. Furthermore, we can see that measuring on a larger batch size leads to slightly higher neuron utilization, indicating that a fair amount of neurons serve a purpose for the dataset and are not dead. In comparison to learning rule (13) the rule (12) leads to a higher neuron utilization, which makes sense due to the fairer competition. We want to note that a higher neuron utilization does not necessarily lead to a higher performance, because a lower neuron utilization may lead to better distinguishable, sparser representations.

The lower part of Fig. 5 shows how the classification accuracy of the l classifier increases, when the number of representation vectors per class used to learn the l classifier increases, reaching over 41% when trained on 50 examples per class.

### F. Performances on the Datasets and Generalization

Table II shows the performance of our best models compared to other methods. For fs training 50 samples per class were used. The model we use for Cifar100, Tiny ImageNet and SOMe ImageNet is not tuned for these datasets. It uses the same hyperparameters as our Cifar10 model (except the training steps for Cifar100). We can see that D-CSNN-B1 is comparable to many recent methods, but seems to lack performance for Tiny ImageNet, probably because this amount of classes requires bigger models. To additionally test the generalization capability we train our model on one dataset and test it on the remaining datasets, achieving excellent generalization capability, where difference in accuracy lie in the low percentage range.

### G. Visual Interpretations

Fig. 6 row b) shows that the input images can be reconstructed from the representation showing a high degree of detail. Furthermore, one can see in rows c) and d) that "similar" classes lead to similar average representations, especially there exist similar spots with high average neuron activities. The classes are sorted according to their $L_1$ distance from the truck class and we can see that the car class is most similar, followed by the horse class. Row e) shows remaining distinguishable differences between average representations of the class and the neighboring class on the right hand side (e.g., the first entry equals the truck minus car average representation).

## V. DISCUSSION AND CONCLUSION

In this work we introduced the modular building blocks of CSNNs to learn representations in an unsupervised manner without Backpropagation. With the proposed CSNN modules

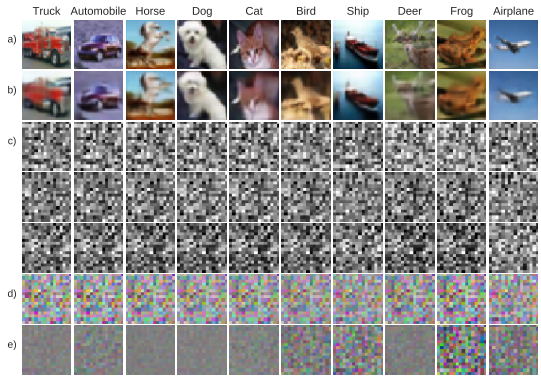


Fig. 6. a) The Cifar10 input image. b) The reconstruction using the representation of the D-CSNN-B1 (no augmentation). c) The average representation per map for each class, formed by taking the mean over the spatial dimension of each test sample's representation and reshaping the resulting vector to SOM-grid shape. d) RGB image of the average representation (three maps lead to an RGB image). e) Differences of several average representations achieved through element-wise subtracting the classes average representation from the classes average representation on the right hand side. Best viewed in color.

TABLE II
PERFORMANCES OF THE D-CSNN-B1 MODEL.

| Models (backpropagation, backpropagation-free, ours, ours) | Cifar10 | | | Cifar100 | | | Tiny ImageNet | | | SOMe ImageNet | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *l* | *nl* | *fs (50)* | *l* | *nl* | *fs (50)* | *l* | *nl* | *fs (50)* | *l* | *nl* | *fs (50)* |
| VAE [21] | 54.45 (SVM) | 60.71 | - | - | 37.21 | - | - | 18.63 | - | - | - | - |
| BiGAN [21] | 57.52 (SVM) | 62.74 | - | - | 37.59 | - | - | 24.38 | - | - | - | - |
| DIM(L) (maximum reported accuracy in the paper) [21] | 64.11 (SVM) | 80.95 | - | - | **49.74** | - | - | **38.09** | - | - | - | - |
| AET-project (best single model accuracy we know) [15] | **83.35** | **90.59** | - | - | - | - | - | - | - | - | - | - |
| SOMNet [29] | 78.57 (SVM) | - | - | - | - | - | - | - | - | - | - | - |
| K-means (Triangle) (best single model accuracy we know) [30] | **79.60** (SVM) | - | - | - | - | - | - | - | - | - | - | - |
| D-CSNN-B1 | 73.73 | 77.83 | **41.59** | 45.17 | 49.80 | 27.29 | 13.52 | 3.56 | 13.70 | 68.92 | 70.69 | **49.99** |
| D-CSNN-B1-Aug | **75.68** | **81.55** | - | **46.82** | **52.66** | - | 13.32 | **5.20** | - | **70.80** | **75.29** | - |
| D-CSNN-B1-Cifar10 | **73.73** | 77.83 | **41.59** | 45.68 | 49.70 | **27.96** | **14.36** | 2.03 | 13.46 | 65.34 | 65.59 | 47.65 |
| D-CSNN-B1-Cifar100 | 71.60 | 76.46 | 41.43 | 45.17 | 49.80 | 27.29 | 13.57 | 1.61 | **13.72** | 63.79 | 65.13 | 45.68 |

which combine CNNs, SOMs and Hebbian learning of masks a new, alternative way for learning unsupervised feature hierarchies has been explored. Along the way we discussed the Scalar Bow tie Problem and the Objective Function Mismatch: two problems in the field of Deep Learning, which we belief can be potentially solved together and are an interesting direction for future research.

## REFERENCES


[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*. Curran Associates, Inc., 2012.

[2] P. Werbos, "Beyond regression: New tools for prediction and analysis in the behavioral science," Ph.D. dissertation, Harvard University, 1974.

[3] Y. Chauvin and D. E. Rumelhart, Eds., *Backpropagation: Theory, Architectures, and Applications*. L. Erlbaum Associates Inc., 1995.

[4] C. Doersch and A. Zisserman, "Multi-task self-supervised visual learning," *CoRR*, vol. abs/1708.07860, 2017.

[5] T. P. Lillicrap, D. Cownden, D. B. Tweed, and C. J. Akerman, "Random synaptic feedback weights support error backpropagation for deep learning," *Nature Communications*, vol. 7, 2016.

[6] J. C.R. Whittington and R. Bogacz, "Theories of error back-propagation in the brain," *Trends in Cognitive Sciences*, vol. 23, 2019.

[7] Y. Bengio, D. Lee, J. Bornschein, and Z. Lin, "Towards biologically plausible deep learning," *CoRR*, vol. abs/1502.04156, 2015.

[8] S. Grossberg, "Competitive learning: From interactive activation to adaptive resonance," *Cognitive Science*, vol. 11, 1987.

[9] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, 1989.

[10] T. Kohonen, "The self-organizing map," *Proceedings of the IEEE*, vol. 78, 1990.

[11] G. L. Shaw, "Donald hebb: The organization of behavior," in *Brain Theory*. Springer Berlin Heidelberg, 1986.

[12] A. Saxena, M. Prasad, A. Gupta, N. Bharill, o. Patel, A. Tiwari, M. Er, W. Ding, and C.-T. Lin, "A review of clustering techniques and developments," *Neurocomputing*, vol. 267, 2017.

[13] T. D. Sanger, "Optimal unsupervised learning in a single-layer linear feedforward neural network," *Neural Networks*, vol. 2, 1989.

[14] E. Oja, "Simplified neuron model as a principal component analyzer," *Journal of Mathematical Biology*, vol. 15, 1982.

[15] L. Zhang, G. Qi, L. Wang, and J. Luo, "AET vs. AED: presentation learning by auto-encoding transformations rather than data," *CoRR*, vol. abs/1901.04596, 2019.

[16] M. Tschannen, O. Bachem, and M. Lucic, "Recent advances in autoencoder-based representation learning," *CoRR*, 2018.

[17] V. Fortuin, M. Hüser, F. Locatello, H. Strathmann, and G. Rätsch, "Deep self-organization: Interpretable discrete representation learning on time series," *CoRR*, vol. abs/1806.02199, 2018.

[18] Z. Xu, Z. Liu, C. Sun, K. Murphy, W. T. Freeman, J. B. Tenenbaum, and J. Wu, "Unsupervised discovery of parts, structure, and dynamics," *CoRR*, vol. abs/1903.05136, 2019.

[19] A. Kolesnikov, X. Zhai, and L. Beyer, "Revisiting self-supervised visual representation learning," *CoRR*, vol. abs/1901.09005, 2019.

[20] S. Gidaris, P. Singh, and N. Komodakis, "Unsupervised representation learning by predicting image rotations," *CoRR*, 2018.

[21] R. D. Hjelm, A. Fedorov, S. Lavoie-Marchildon, K. Grewal, P. Bachman, A. Trischler, and Y. Bengio, "Learning deep representations by mutual information estimation and maximization," in *International Conference on Learning Representations*, 2019.

[22] J. Huang, Q. Dong, S. Gong, and X. Zhu, "Unsupervised deep learning by neighbourhood discovery," *CoRR*, vol. abs/1904.11567, 2019.

[23] A. J. Piergiovanni, A. Angelova, and M. S. Ryoo, "Evolving losses for unlabeled video representation learning," *CoRR*, 2019.

[24] Q. Yang, H.-X. Yu, A. Wu, and W.-S. Zheng, "Patch based discriminative feature learning for unsupervised person re-identification," in *International Conference on Computer Vision and Pattern Recognition*, 2019.

[25] P. A. Bizopoulos and D. D. Koutsouris, "Sparsely activated networks," *CoRR*, vol. abs/1907.06592, 2019.

[26] M. Ye, X. Zhang, P. C. Yuen, and S. Chang, "Unsupervised embedding learning via invariant and spreading instance feature," *CoRR*, vol. abs/1904.03436, 2019.

[27] J. Donahue and K. Simonyan, "Large scale adversarial representation learning," *CoRR*, vol. abs/1907.02544, 2019.

[28] T. Chan, K. Jia, S. Gao, J. Lu, Z. Zeng, and Y. Ma, "Pcanet: A simple deep learning baseline for image classification?" *CoRR*, 2014.

[29] R. Hankins, Y. Peng, and H. Yin, "Towards complex features: Competitive receptive fields in unsupervised deep networks," in *Intelligent Data Engineering and Automated Learning – IDEAL 2018*. Springer International Publishing, 2018.

[30] A. Coates, A. Ng, and H. Lee, "An analysis of single-layer networks in unsupervised feature learning," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, 2011.

[31] H. Zheng, G. Lefebvre, and C. Laurent, "Fast-learning adaptive-subspace self-organizing map: An application to saliency-based invariant image feature construction," *IEEE Transactions on Neural Networks*, 2008.

[32] D. Krotov and J. J. Hopfield, "Unsupervised learning by competing hidden units," *Proceedings of the National Academy of Sciences*, 2019.

[33] T.-H. Lin and H. T. Kung, "Stable and efficient representation learning with nonnegativity constraints," in *Proceedings of the 31st International Conference on Machine Learning*, 2014.

[34] D. J. Saunders, D. Patel, H. Hazan, H. T. Siegelmann, and R. Kozma, "Locally connected spiking neural networks for unsupervised feature learning," *CoRR*, vol. abs/1904.06269, 2019.

[35] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[36] A. Dundar, J. Jin, and E. Culurciello, "Convolutional clustering for unsupervised learning," *CoRR*, vol. abs/1511.06241, 2015.

[37] J. C. Principe, N. R. Euliano, and W. C. Lefebvre, *Neural and Adaptive Systems: Fundamentals Through Simulations with CD-ROM*, 1st ed. John Wiley & Sons, Inc., 1999.

[38] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015.

[39] A. Krizhevsky, "Learning multiple layers of features from tiny images," *University of Toronto*, 2009.

[40] J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Li Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Conference on Computer Vision and Pattern Recognition*, 2009.

[41] A. M. Saxe, P. W. Koh, Z. Chen, M. Bhand, B. Suresh, and A. Y. Ng, "On random weights and unsupervised feature learning," in *In NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2010.

[42] L. Metz, N. Maheswaranathan, B. Cheung, and J. Sohl-Dickstein, "Learning unsupervised learning rules," in *International Conference on Learning Representations*, 2019.



Author's version of the paper. Final version presented at
18th IEEE International Conference on Machine Learning and Applications (ICMLA).Boca Raton, Florida / USA. 2019.