Convolutional SNNs with Filter Competition for Unsupervised Image Classification

Ivan Vegner

December 11, 2020

1 Introduction

Convolutional artificial neural networks have become the gold standard of machine learning on images. They excel at this task for one main reason – CNNs account for translational and spatial invariance, that is, a translated input yields a translated output. By the shared nature of all ANNs, CNNs have no concept of time – for a given input image, the entire convolutional map is built at once by convolving a "window" over the entire input image in a single processing step. This requires CNNs to rely on a large number of convolutional filters, so that for every given window into the input ("patch"), the network can compute a large number of features at once. The learning of so many filters is a heavy computational investment, and contributes to the data-hungry nature of the algorithm. Furthermore, for a given input a CNN will often rely on only some of the filters computed for it, meaning that the filter computation of the filters not relevant for the current input is effectively wasted[1]. What if, instead of computing many filters for every patch and ending up later relying on only a few, the CNN was able to automatically compute only the most salient filters for a given patch?

Unlike ANNs, spiking neural networks (SNNs) are time-dependent, which allows the convolution of an input to be processed patch-by-patch, instead of the entire image at once. Combined with the natural applicability of SNNs to "winner-take-all" modes of processing, this may allow SNNs to automatically compute the most active "filter" for a given patch.

Furthermore, if for every patch the network is not only able to compute the most salient filter(s) but also has access to the spatial position of the patch within the image, it is possible to induce the proposed architecture to learn a sparse, symbolic representation of the input image as a conjunctive set of certain features at certain image locations. In contrast to the highly-connected nature of CNNs, which process every filter at every location of the image simultaneously, such a symbolic approach reduces the number of connections necessary, promoting connection reuse which in turn makes the network more interpretable and may lead to less data being required for training.

[1] If for some reason the layer relies on a large number of filters equally, this contributes to the uninterpretable black box nature of the algorithm and often signals that the CNN is undertrained.

2 Prior Work

2.1 SNNs on images

The application of spiking neural networks to images is not a novel idea. The seminal work in unsupervised training of SNNs on images is Diehl and Cook 2015 [1], which trained one layer of laterally-inhibitory spiking neurons, each of which was connected to every pixel of the input image. This approach suffered from the lack of a convolutional mechanism, which greatly reduces its scalability to datasets larger than MNIST. This line of research is taken further by Saunders et al. 2019 [8], which trains a locally-connected (localist fully-connected) network with multiple output filters, in which each output neuron has a receptive field reminiscent of a CNN without convolutional weight sharing.

However, convolutions with weight-sharing filters are possible in spiking architectures. While several studies [4, 10] attempt to apply supervised gradient descent and hybrid gradient-STDP methods, these methods are biologically implausible and less generalizable than unsupervised methods due to their reliance on a large amount of labeled data.

Among the unsupervised explicit convolution efforts, notable prior work includes Kheradpisheh et al. 2016 [3]. This architecture processes the input using Difference of Gaussians and then iteratively trains deep pseudo-convolutional[2] spiking layers followed by pooling layers to condense the input into a low-dimensional sparse representation later used for an external classifier. Notably, this work reports single-shot category learning in the classifier trained on the outputs of the network, a testament to the quality of the input representation, and a favorable few-shot learning comparison is made with conventional CNNs. However, this model effectively uses one or more filters *per class* in each of its layers, which incurs a significant computational cost.

The present work is most similar to the architecture of Tavanaei and Maida 2017 [11]. Drawing heavily from conventional CNNs, this work utilizes the SAILnet model [12] to train a set of convolutional filters that encode the input patch as a set of D distributed sparse features similar to those found in the V1 cortical area, in a biologically-plausible way. For each $(P \times P)$-pixel patch of the $(C_1 \times C_2)$-pixel input, the features of the patch are coded for by the laterally-inhibited firing of D LIF spiking neurons (filter neurons). The $P \times P$ weights of the filter neurons represent different low-level visual features, learned by a Hebbian rule. The spike trains of the filter neurons for each patch are aggregated to form a convolutional feature map of the input image, of dimensions $(C_1 - P + 1, C_2 - P + 1, D, T)$, where T is the number of timesteps simulated for the filter neurons. The convolutional layer is followed by a max-pooling layer, which for each filter $d \in range(D)$ simply picks the highest-activity neuron from each $l_p \times l_p$ neighborhood of the d-th filter convolutional feature map. The stride value of the pooling layer is also l_p, thus the pooling layer produces a $((C_1 - P + 1)/l_p, (C_2 - P + 1)/l_p, D, T)$ feature map. Finally, there is a fully-connected feedforward spiking "feature discovery" layer, which takes input spike trains from the pooling layer. The LIF neurons of the feature discovery layer are laterally-inhibited, and spike in a "winner-take-all" (WTA) manner to achieve a low-dimensional sparse representation of the input.

[2] No actual convolution operation takes place, but weights of a given filter are manually shared among all its neurons.

All feature discovery layer synapses are learned by STDP[3]. The discovered features can then be fed to an outside classifier.

All of the above methods (except Diehl and Cook 2015 [1]) share the multi-filtered nature of conventional CNNs (sometimes with more than 1000 filters), and thus waste computation if a given filter's output does not end up being used, while decreasing interpretability and increasing redundancy and training time.

2.2 Spiking symbolic learning

There is evidence suggesting that the brain represents high-level concepts using sparse, distributed neural coding [6]. While this sparse coding is unlikely to be on the level of "grandmother cells" [7] – single neurons coding for single concepts – it may be useful to sparsify the representation of concepts as much as possible to increase reuse of synapses and concept-coding neurons.

The DORA network from Doumas, Puebla, and Martin 2018 [2] builds upon spiking neural network principles to incrementally encode concepts and relations in a sparse hierarchical way, without supervision of any kind. This is accomplished by incrementally augmenting the spiking network with new neurons that encode conjunctions of the firing neurons of the previous layer, while altering existing connections with a Hebbian mechanism. Combined with an explicit memory mapping mechanism and several phase binding rules, this allows DORA to learn a *symbolic role-filling calculus* upon its inputs and perform few-shot transfer learning to new tasks. The paper evaluates transfer between the Atari games of Pong and Breakout, and reports few-shot transfer of superhuman performance without supervision.

While transfer learning is out of scope for this proposal, the architecture attempts to learn a "feature discovery" layer similar to the aforementioned Tavanaei and Maida 2017 [11] in a way conceptually similar to that of DORA's [2] incremental conjunctive encoding of novel inputs.

2.3 Window coordinates as CNN inputs

The idea of including the coordinates of the current patch in the CNN input from said patch was explored in Liu et al. 2018 [5]. They found that simply giving the convolution access to the input coordinates as extra channels improved generalization in an R-CNN trained on MNIST, reduced mode collapse in GANs, and provided increased performance in reinforcement learning tasks. Increased performance on ImageNet classification is not reported, as would be expected because image classification requires translation invariance. However, the approach is still valid for the proposed architecture because the patches are processed one at a time and thus the coordinates are used to learn the position of the feature.

[3]Spike timing dependent plasticity.

3 Aim

The goal of this work is to build upon the architecture of Tavanaei and Maida 2017 [11], making it simpler, smaller, more efficient in data and compute, and even symbolically intepretable, by exploiting the affinity of SNNs for time-distributed processing.

The above paper relies on pre-computing the firing of D neurons for each patch of the image, which are then further aggregated across T timesteps of simulation. This aggregation operation is biologically implausible, computationally expensive, and being linear in T does not play to the strengths of SNNs in real-time processing. It should not be necessary to pre-compute and save the T-dimensional output of an SNN before feeding it to another SNN – the spike train can likely be fed directly to other spiking neurons as time-distributed inputs.

Furthermore, the output of the convolutional layer is max-pooled in a biologically-dubious way. The pooling window and stride used in the paper was $l_p = 2$, so for every 2×2-sized patch of the convolutional feature map, only one filter neuron's spike train is kept and the rest are discarded. This wastes 75% of the computation required to build the feature map in the first place. As the pooling criterion is simply "the activity of the most-active neuron is propagated", it seems that the pooling operation can be replaced with a WTA scheme performed directly on the filter neurons themselves.

Finally, the feature discovery layer is fully-connected to all of the outputs of the pooling layer. That is, each neuron of the feature discovery layer ("FD neuron") is effectively connected to all of the most active neurons in each filter of each patch of the input. With $D = 32$ filter neurons, a common number of filters in the source paper, and pooling size $l_p = 2$, a convolutional feature map of size 24×24 will require each FD neuron to have $(24/2) * (24/2) * D = 4,608$ excitatory connections. This entire-image connectivity is unnecessary, because it once again does not exploit the time-distributed nature of SNNs.

By getting rid of the pooling layer, connecting filter neurons with laterally-inhibitory connections, incorporating patch position as input and processing the convolutional windows one-by-one instead of all-at-once, the feature discovery layer can learn the same features as time- and space-distributed connections to the most salient filters for each patch. In this way, the required number of excitatory synapses per FD neuron can be reduced to just $D + 2$, which should lead to dramatic gains in computational and data efficiency. Additionally, as a result of the connective sparsity and reuse across filters in different spatial positions, the FD synapses start to take on more symbolically-interpretable meaning as explored in Doumas, Puebla, and Martin 2018 [2], which promises fewer-shot learning and opens possibilities for later integration of similar relational mechanisms.

4 Architecture

The overall structure of the network is conceptually similar to that of [11], and is fairly simple overall.

Let the input image be a $c_1 \times c_2 \times c_3$ array of real-value pixels, scaled in $[0, 1]$.

4.1 Patch Representation

The convolution operation slides a $p \times p \times c_3$ window over the input image with stride 1, producing a patch of real values corresponding to the window. Thus, create a set A of $p \times p \times c_3$ spiking neurons, which will later serve as the spiking representation of the patch currently in the window.

Over the course of the convolution operation, let $Q \in \mathbb{R}^{p \times p \times c_3}$ be the matrix of real pixel values for a given patch of input scaled in $[0, 1]$. Each value $q_{i,j,k} \in Q$ is Poisson-encoded to produce the firing rate for spiking neuron $a_{i,j,k} \in A$. That is, the average firing rate for $a_{i,j,k}$ is $(q_{i,j,k} * \alpha)$ Hz.[4] Thus, A is now a spiking pixelwise representation of the patch Q.

Finally, create a set B of two spiking neurons. B is similar to A, but it will code for the position of the patch in the input image. For a given patch with top-left pixel coordinates $[m, n]$, set the average firing rate of $b_1 \in B$ to $\frac{m}{c_1} * \alpha$ Hz, and of b_2 to $\frac{n}{c_2} * \alpha$ Hz. Alternatively, B could contain four neurons:

Neuron in B	b_1	b_2	b_3	b_4
Avg Firing Rate (Hz)	$\frac{m}{c_1}\alpha$	$\frac{c_1 - m}{c_1}\alpha$	$\frac{n}{c_2}\alpha$	$\frac{c_2 - n}{c_2}\alpha$

The latter coding fulfills the same function, but ensures the same average level of activity in B regardless of patch position. The latter 4-neuron position coding is the one used in the present work.

4.2 Convolutional Filters

Initialize a set F of d spiking neurons ("filter neurons"), with fully-connected randomly initialized excitatory synapses from each $a \in A$ to each $f \in F$, and fully-connected inhibitory synapses between neurons in F.

Each patch of the input is shown to the network for τ timesteps, i.e. milliseconds. For the next τ timesteps, A is spiking proportionately to the values in the patch. As filter neurons F are fully-connected to A with excitatory synapses but inhibitorily connected to each other, one neuron $\hat{f} \in F$ which randomly happens to have strong connections to strongly-spiking parts of the patch will spike first and temporarily inhibit all the other neurons. This neuron $\hat{f}$ is *the most salient filter* of the patch. Depending on τ and the neuron thresholds, there may be sufficient time to have another neuron spike for the same patch. This is the *second most salient filter* of the patch, and so forth. Thus, for a given patch of input, we automatically get the most salient filter neurons spiking in rank order. The number of filter neurons that spike per patch is a function of τ, the neuron spike thresholds, and the weight initializations. As a function of the STDP learning rule, the synapses of the filter neurons converge to the most frequent input patterns, and is similar to those learned by Zylberberg, Murphy, and DeWeese 2011 [12].

While this mechanism greatly simplifies the convolutional layer of the source paper, it does not yet incorporate pooling, which aggregates the most-active filters *across patches*. However, as will be explained in the section below, max-activity pooling is performed implicitly by the WTA feature discovery layers.

[4] $\alpha = 127$ in the current work.

Importantly, the voltages of all neurons in A and F are reset to their resting voltage after τ timesteps, to provide a clean slate for a new incoming patch.

4.3 Feature Discovery

As explained above, convolution proceeds one patch at a time, and for each patch, the neurons $\hat{f}_1, \hat{f}_2 \ldots \in F$ that code for the most salient filters in the patch spike in rank order. Let there be two feature discovery (FD) layers L_1, L_2 composed of h_1, h_2 laterally-inhibitory LIF neurons, with fully-connected excitatory connections from filter neurons F to L_1 and from L_1 to L_2. Additionally, let there be fully-connected excitatory connections from B to L_1.

By feeding the position of the current patch as input along with the filters, the FD neurons in L_1 learn to code for conjunctions of the most salient filters for the patch and their position in the image. This is crucial, because it is a highly sparse and symbolic representation of the noisy and high-dimensional data of the patch. For example, if an image contains a diagonal line in the top right corner, the same filter neuron in F fires as if the diagonal line was in the bottom left, but the two patterns are differentiated in L_1 by their conjunction with the position information from B. This behavior is reminiscent of the role-filling symbolic connections in Doumas, Puebla, and Martin 2018 [2], and stands in stark contrast to the source paper which learns separate FD features for each filter in each possible position. Such a position-independent formulation encourages connection reuse in the filters, and greatly reduces the number of connections necessary. As with A and F, the voltages of L_1 are also reset at the end of each patch exposure period.

Meanwhile, the voltages in L_2 are not reset between patches. Thus, while L_1 codes for combinations of filter+position for each patch, L_2 is able to form connections between patches across the entire image as the convolution operation examines different patches. Neurons in L_2 learn to code for conjunctions of the most salient filter-position patterns in L_1, and thus represent the input image symbolically as a combination of certain filters at certain positions. Such a representation can then be "walked back" down the connections to recover the constituent filters in the receptive field of a given L_2 neuron, and may be further extended with relational modeling as in Doumas, Puebla, and Martin 2018 [2].

Notably, L_2 implicitly performs a cross-patch max-pooling operation, in which the weaker-activity filter-position combinations in L_1 (that would have been max-pooled out in explicit pooling) fire less frequently and so provide less input to L_2. As the synaptic weights for every neuron are normalized, this difference in activity encourages the L_2 neurons to decrease synaptic weight to such weaker inputs, accomplishing a max-pool operation in a biologically-plausible way.

4.4 Learning

All connections are learned by spike-timing dependent plasticity. STDP is both biologically-plausible and induces a favorable conjunctive coding that increases weights for inputs that correlate highly with the output and, crucially, decreases weights for inputs that do not. This makes sure that each neuron in L_1 and L_2 codes for a specific combination of patterns in the previous layer, and cannot be activated by other patterns.

For every neuron in F, L_1 and L_2, the sum of its incoming synaptic connections is normalized to a constant. The normalization constants are summarized in the following table. Note that L_1 has incoming synapses from both filter neurons F and position neurons B. As the position layer contains fewer neurons than filter layer, the $B \to L_1$ normalization constant is proportionally smaller than that for $F \to L_1$, but the total weight of connections for each L_1 neuron still sums to 15 like in L_2.

Layer	$I \to F$	$F \to L_1$	$B \to L_1$	$L_1 \to L_2$
Norm	5	13.3	1.7	15

Table 1: Normalization constants for incoming layer synapses.

4.5 External Classifier

After sufficient training, the spike trains from L_2 are aggregated across time and used to train an external classifier. A linear SVM was used in Tavanaei and Maida 2017 [11], and the present work uses a logistic regression for similar effect, chosen over an SVM due to the large number of classes in the dataset. This forms the only supervised component of the project, and is not directly part of the spiking network. The input to the classifier consists of the spike train from L_2 for a given image, summed across time and normalized to $[0, 1]$ via the softmax operation to transform the spike train into a vector of relative spike counts. Softmax is applied to reduce the effect of "whiter" images containing more active pixels generating more L2 spikes simply by having more filter activations, which without softmax normalization may adversely affect classification performance on less-active images.

5 Tasks

The presented architecture was applied to the 48-class "balanced" subset of EMNIST, a character recognition task on both letters and numbers. The "balanced" subset guarantees equal number of samples across classes, and removes several easily-confused characters such as lowercase and uppercase C. EMNIST is used as the task due to its single-channel nature, which avoids tripling the number of neurons in the filter layer to account for the RGB channels.

As this work is heavily based on Tavanaei and Maida 2017 [11] which tests on MNIST, the presented architecture is tested on the same for an apples-to-apples comparison with the source and many related papers.

Finally, extensive visualization is performed on the learned weights of the architecture. Filter weights are visualized, and the receptive fields of L_2 neurons are calculated by tracing its connections back down through L_1 and F. The similarity of visualized weights to the filters and reconstructions presented in Zylberberg, Murphy, and DeWeese 2011 [12] is assessed.

6 Results

6.1 Classification Accuracy

The best classification accuracy achieved using the network and the logistic classifier as described was 0.14 on EMNIST, and 0.45 on MNIST. While significantly better than chance, these results are not close to the state of the art.

The given performance was a result of extensive model iteration and parameter search, as well as applying all augmentations listed in section 7. As outlined in section 8, the compressed timeframe forced me to use high learning rates and a subset of the dataset, which likely impacted performance.

Layer	Neurons	Norm	Pre-syn. LR	Thresh	Init. inh.	Inh. LR
F	32	5	0.01	-63	0.01	0.1
L_1	256	15*	10	-63	0.01	0.1
L_2	64	15	10	-63	0.01	0.001

Table 2: Hyperparameters for best-performing model. *LR*=learning rate. *Thresh*=voltage threshold for spiking. *Init. inh.*= initial lateral inhibitory weight. * = see subsection 4.4 and Figure 5 for details of L_1 normalization constant.

6.2 Weight Analysis

The weight matrices of the best-performing network are presented in Figure 1. Note the significant sparsity achieved in all matrices. Furthermore, visualizing the $L_1 \rightarrow L_2$ weights over the course of training, it is evident that new L_2 units become active (symbolized by a "yellower" color) *without disturbing the weights and connectivity patterns of existing L_2 units*. This property is critical for online and few-shot learning, and its emergence in the present architecture is promising.

6.2.1 Filter Weights

A visualization of only the filter weights of the architecture is presented in Figure 2. Notably, almost every filter becomes active. Due to the inhibitory connections be-

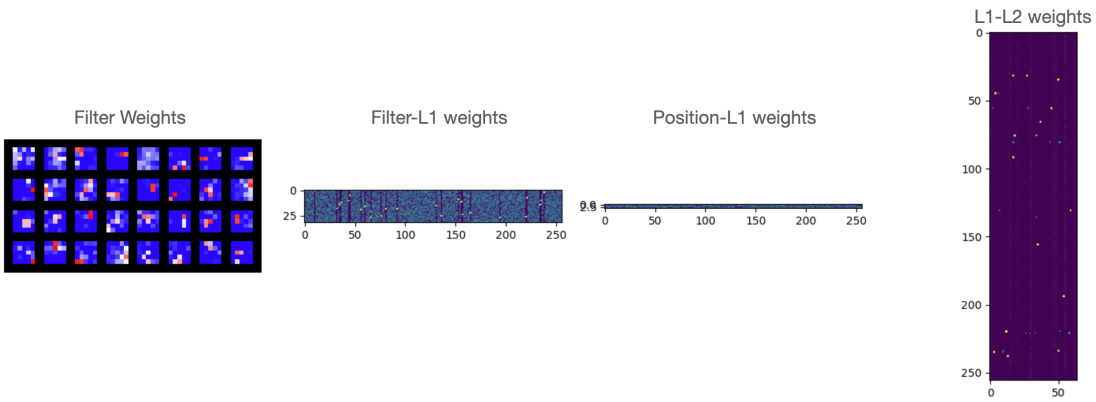

Figure 1: Visualization of the learned weights of the best-performing model. Incremental training described in subsection 7.1 was applied to train the filter weights.

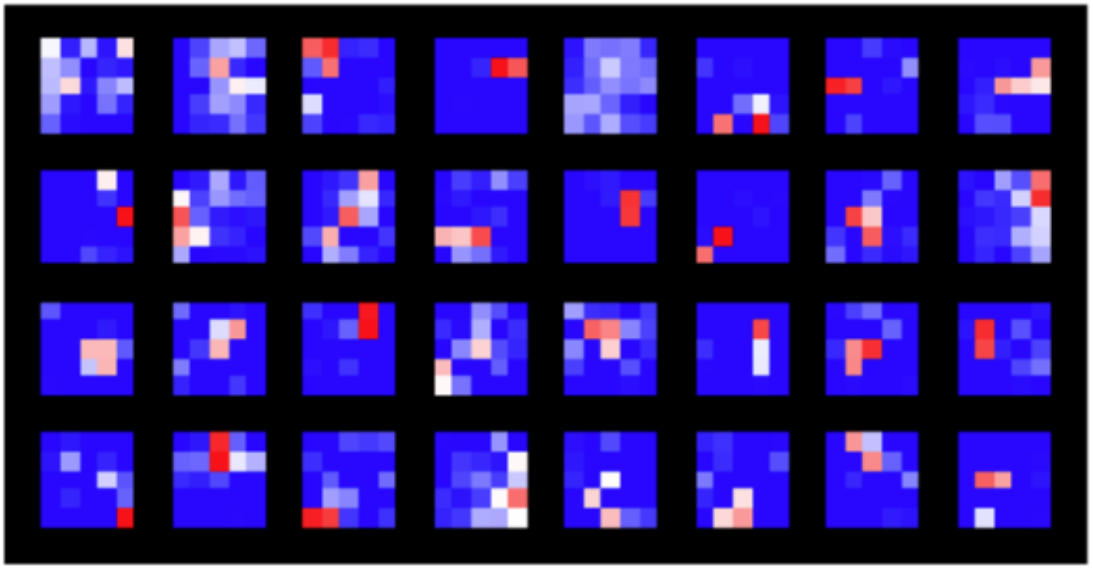

Figure 2: Visualization of learned filter weights

tween filters, the process of competition for activation between the filters induces them to code for different patterns, which is important for generalization performance. Some of the filters resemble the "tilted line" and "center dot" filters presented in SAILNet [12].

Intriguingly, with sufficiently high L_1 and L_2 learning rates it is possible to achieve high accuracy without training the filters at all. That is, even random filters can suffice to be combined by L_1 and achieve learning. This behavior breaks down at lower L_1 and L_2 learning rates, likely due to the fact that untrained filters do not provide consistent enough information to accurately group more than one sample. Whether this behavior is sustainable for achieving higher accuracy in the future remains to be explored.

6.2.2 L2 Weights

Recall that every L_2 neuron is connected to several L_1 neurons, which in turn are connected to filter neurons and position neurons. It is possible to recover the "receptive field" of an L_2 neuron by "walking down" its connections to sum the receptive fields of its constituent L_1 neurons, weighted by the strength of the synapse between the given L_1 and L_2 neuron. The receptive fields of L_1 neurons are a similar weighted sum of the filter neurons they are connected to, with the position of the summed filters given by the strength of the L_1 neuron's connections to the position neuron.

Thus, it is possible to reconstruct the constituent filters and positions that activate a given L_2 neuron the most. Such a reconstruction is presented in Figure 3.

As expected, the connectivity patterns of the L_2 neurons are distinct from one another, with each coding for a unique combination. Unfortunately, unlike in SAILNet they are not human-interpretable, which likely also causes the poor classification performance. Some reasons leading to this result are discussed in section 8.

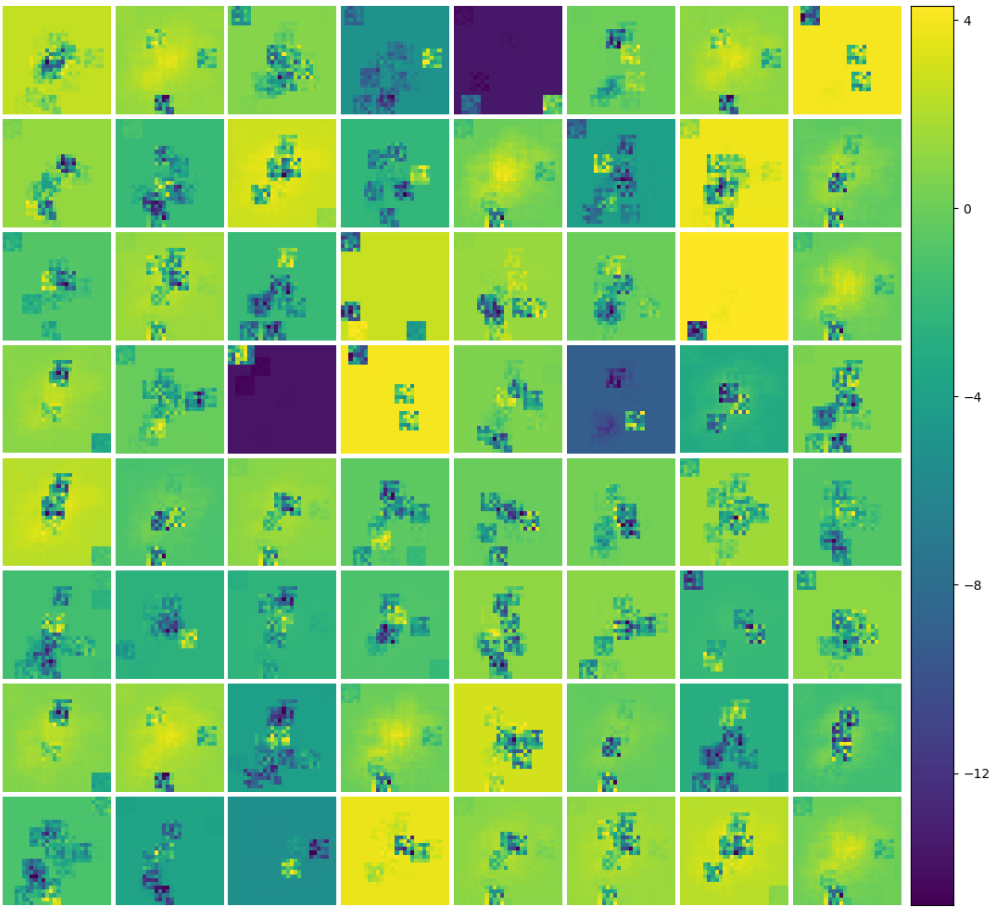

Figure 3: Visualization of the receptive fields of L2 neurons of the best-performing model.

7 Augmentations Attempted

7.1 Incremental Training

The network is structured in a hierarchical manner, with higher layers using information from the lower. Thus, it would make sense to train the filter layer first, to provide a base of invariant patterns for L_1 and L_2. A similar approach was used by Tavanaei and Maida [11], wherein the SAILNet features were trained first and separately.

Experiments were conducted using both incremental and joint training. Minor gains in performance resulted from incremental training, around 1 classification accuracy percentage point. The incremental training of filters first would likely benefit from a training run on the entire dataset using a small learning rate in order to extract the most representative low-level patterns. This option was not accessible to me due to computational cost.

7.2 Batching

The operation of SNNs is time-distributed and thus not very efficient to train. To mitigate this, batching was implemented, in which the network was trained on a batch of images simultaneously (activations from one image did not influence

activations on the other).

However, it came to light that batching does not make sense for the given architecture. The updates must be aggregated across the batch, which inhibits the hierarchical few-shot learning performed by L_1 and L_2.

For example, suppose an L_1 neuron a fires for a particular filter-position combination C. If the batch size is 1, the weights of a will immediately be adjusted to be more selective to the combination C. However, with batch sizes greater than 1, it is possible that in the other images in the batch, a will fire for a different filter-position combination C'. As weight updates are aggregated (summed) across the batch, this may cause a to become selective for neither C nor C' or a combination thereof, which in the context of the given architecture (with an abundance of L_1) is less beneficial than having a become more selective for either C or C'.

Furthermore, the weight updates across the batch can cancel each other out, preventing the neuron from learning at all. This behavior is desirable for deep learning, which seeks to interpolate between its inputs, but not for the proposed architecture, which encodes its inputs in a few-shot manner.

Unfortunately, this realization was made relatively late in the research process. Furthermore, the lack of batching made training slower by an order of magnitude, which further slowed progress.

7.3 Inhibitory learning

At first, only a few filters in F were getting learned over the course of training. This was due to inhibitory weights being set too high, allowing a few neurons to prevent the activation of the majority of other neurons. This also had high trickle-down effects — if only a few filters were being activated, only a few L_1 neurons would be getting activated in turn, which prevents broad learning in L_2.

This was corrected by setting initial inhibitory weights lower and allowing them to be learned by a Hebbian co-activation rule, which increased the inhibitory weight between two neurons only if they were frequently co-active (and thus needed to compete to differentiate themselves). This improvement allowed all neurons of a given layer to be learned, and was crucial for achieving performance above random chance.

7.4 Weight Normalization

Weight normalization turned out to be crucial for the learning process, especially for the filter layer. Un-normalized weights were overwhelmed by the occasional highly-active patch, and were unable to learn, leading to performance equal to random chance. A comparison of non-normalized and normalized weights is given in Figure 5.

By normalizing each neuron's pre-synaptic weights to sum to a constant, the architecture instituted a "competition" process inside the weights of each neuron, such that to increase the weight of one synapse the other ones would necessarily have to be decreased, which acted as regularization.

Further performance gains were made for filters by also restricting synaptic weights to lie within the range $[-2, 2]$. Combined with weight normalization, these two

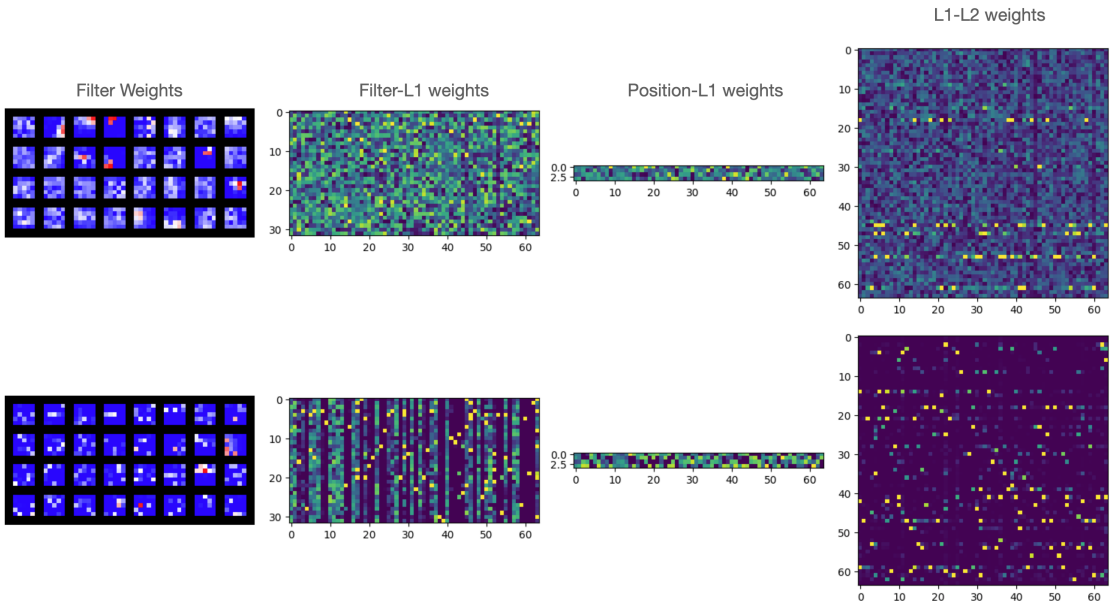

Figure 4: High inhibitory weights, no inhibitory learning (top) leads to activity concentrated in a few neurons per layer. Low initial inhibitory weights with Hebbian learning (bottom) produces broad learning across all neurons of a layer.

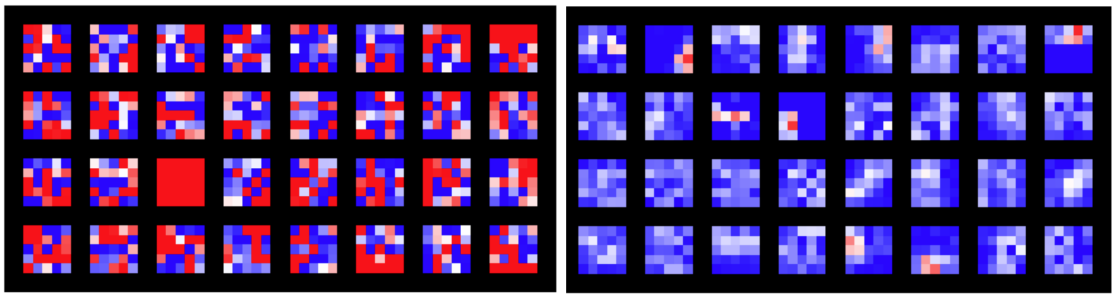

Figure 5: No weight normalization (left) produced learned patterns of random noise, and resulting classification performance was very poor. Note the completely useless all-red filter, which spikes for every input patch. Normalizing filter weights to a constant (right) allowed for concrete patterns to emerge in the weights.

measures approximate L2 regularization by preventing a single weight from becoming too high and using up all of the normalization constant "budget" by itself.

8 Roadblocks

Over the course of building the architecture, I had to iterate very rapidly on a large space of parameters and ideas. Due to extreme time constraints on this independent study, this restricted me to using small subsets of EMNIST (at most 1000 examples, usually around 300-400); otherwise, training the model took too long. This became doubly true after discovering the incompatibility of the architecture with mini-batching methods, described above. Thus, the problem was transformed to a few-shot learning model, which is a significantly more difficult.

Furthermore, to get any learning to occur in the weights on the small subset, I had to increase learning rates far above the usual range for spiking neural networks.

The dangers of doing so are well-studied in ANNs, and while I attempted to mitigate the overfitting and destructive updating resulting from high learning rates using the methods in the above section, it likely had a large impact on overall accuracy.

9 Future Work

I believe the principles of the network are sound, and I would like to continue working on it. As was expected, SAILNet-like features are emergent in the filter layer, and L_2 does in fact grow to code for different conjunctions of L_1 neurons. The fact that the emergence of new active L_2 neurons does not impact the operation of the existing ones is an extremely promising development, is in line with the principles of Doumas et al. [2], and is crucial for a few-shot learning model.

The compressed nature of this semester and especially of this research period forced me to make several concessions which decreased the performance of the model. With sufficient compute and time, it is quite possible that high classification performance is simply just a matter of lowering the learning rate and training for a few epochs on the full dataset.

It may be possible to use batching to speed up training, while not compromising on performance. One such way would be to run all patches of the image through the network until L_1 at once, which is theoretically possible because the information from one patch does not interact with information from others until L_2. I attempted to develop this system, but was stymied by a bug in BINDSnet.

Inhibition may not be as necessary in L_2 as it is in the filter layer, because L_2 neurons are much less likely to be co-active. Perhaps a form of local delayed inhibition, as in Tal et al (2014) [9], can be applied to the L_2 layer to induce automatic localist clustering of related L_2.

Finally, the selection of a classifier can potentially be improved upon. While I believe that a logistic regression is a good fit for the problem, perhaps a perceptron classifier or a voting mechanism would be more suitable.

References

[1] Peter U. Diehl and Matthew Cook. "Unsupervised learning of digit recognition using spike-timing-dependent plasticity". In: *Frontiers in Computational Neuroscience* 9 (2015), p. 99. DOI: 10.3389/fncom.2015.00099.

[2] Leonidas A. A. Doumas, Guillermo Puebla, and Andrea E. Martin. "Human-like generalization in a machine through predicate learning". In: (2018). eprint: 1806.01709. URL: https://arxiv.org/abs/1806.01709.

[3] Saeed Reza Kheradpisheh et al. "STDP-based spiking deep convolutional neural networks for object recognition". In: *arXiv* (2016). Pseudo-convolutional SNN trained by STDP. One "neuronal map", aka filter, per class. No actual convolutions, but explicit weight sharing. Good few-shot performance. DOI: 10.1016/j.neunet.2017.12.005. eprint: 1611.01421.

[4] Chankyu Lee et al. “Training Deep Spiking Convolutional Neural Networks With STDP-Based Unsupervised Pre-training Followed by Supervised Fine-Tuning”. In: *Frontiers in Neuroscience* 12 (2018), p. 435. ISSN: 1662-453X. DOI: 10.3389/fnins.2018.00435. URL: https://www.frontiersin.org/article/10.3389/fnins.2018.00435.

[5] Rosanne Liu et al. *An Intriguing Failing of Convolutional Neural Networks and the CoordConv Solution*. 2018. arXiv: 1807.03247 [cs.CV].

[6] R. Quian Quiroga et al. “Invariant visual representation by single neurons in the human brain”. In: *Nature* 435.7045 (2005), pp. 1102–1107.

[7] R. Quian Quiroga et al. “Sparse but not ‘Grandmother-cell’ coding in the medial temporal lobe”. In: *Trends in Cognitive Sciences* 12.3 (2008), pp. 87–91. ISSN: 1364-6613. DOI: 10.1016/j.tics.2007.12.003.

[8] Daniel J Saunders et al. “Locally Connected Spiking Neural Networks for Unsupervised Feature Learning”. In: *arXiv* (2019). Pseudo-convolutional SNN for learning MNIST and EMNIST. Main contribution is n-gram and sum-of-spikes voting methods for classification. eprint: 1904.06269.

[9] Amir Tal, Noam Peled, and Hava T. Siegelmann. “Biologically inspired load balancing mechanism in neocortical competitive learning”. In: *Frontiers in Neural Circuits* 8 (2014). Runs 3D network of locally excitatory, laterally inhibitory pyramidal neurons on RANDOM input data. Including delayed local inhibition to any active neuron on top of more global immediate lateral inhibition greatly improves clustering behavior., p. 18. DOI: 10.3389/fncir.2014.00018.

[10] A. Tavanaei, Z. Kirby, and A. S. Maida. “Training Spiking ConvNets by STDP and Gradient Descent”. In: *2018 International Joint Conference on Neural Networks (IJCNN)*. 2018, pp. 1–8. DOI: 10.1109/IJCNN.2018.8489104.

[11] Amirhossein Tavanaei and Anthony S. Maida. “Multi-Layer Unsupervised Learning in a Spiking Convolutional Neural Network”. In: *2017 International Joint Conference on Neural Networks (IJCNN)* (2017), pp. 2023–2030. DOI: 10.1109/ijcnn.2017.7966099.

[12] Joel Zylberberg, Jason Timothy Murphy, and Michael Robert DeWeese. “A sparse coding model with synaptically local plasticity and spiking neurons can account for the diverse shapes of V1 simple cell receptive fields”. In: *PLoS Comput Biol* 7.10 (2011), e1002250.