Scalable Bio-Inspired Training of Deep Neural Networks with FastHebb

Gabriele Lagani[a,*], Fabrizio Falchi[a], Claudio Gennaro[a], Hannes Fassold[b], Giuseppe Amato[a]

[a] ISTI-CNR, Pisa, 56124, Italy
[b] Joanneum Research, Graz, 8010, Austria

Abstract

Recent work on sample efficient training of Deep Neural Networks (DNNs) proposed a semi-supervised methodology based on biologically inspired Hebbian learning, combined with traditional backprop-based training. Promising results were achieved on various computer vision benchmarks, in scenarios of scarce labeled data availability. However, current Hebbian learning solutions can hardly address large-scale scenarios due to their demanding computational cost. In order to tackle this limitation, this contribution develops a novel solution by reformulating Hebbian learning rules in terms of matrix multiplications, which can be executed more efficiently on GPU. We experimentally show that the proposed approach, named FastHebb, accelerates training speed up to 70 times, allowing us to gracefully scale Hebbian learning expriments on large datasets and network architectures such as ImageNet and VGG.

Keywords: Hebbian Learning, Deep Learning, Neural Networks, Biologically Inspired

1. Introduction

Recent efforts from the research community focused on the development of biologically plausible alternative to the backpropagation algorithms for Deep Neural Network (DNN) training. Biological constraints require neurons to use only locally available information to compute the weight updates, and neuroscientific observations suggest that synaptic plasticity follows the *Hebbian* model [1, 2]. In simple terms, the weight update should be proportional to the input on the respective synapse and the neuron output at a given point in time. The study of biologically realistic learning models is interesting both because they are well suited for neuromorphic applications [3, 4], and for the perspective to better understand the mechanisms behind biological intelligence and use them to enhance current Artificial Intelligence (AI) technologies.

Among the recently proposed bio-inspired learning approaches, Contrastive Hebbian Learning (CHL) [5] and Equilibrium Propagation (EP) [6] leverage recurrent architectures with Hebbian and anti-Hebbian phases, showing that the resulting update steps approximate backprop. More recently, the Forward-Forward (FF) approach has been proposed [7] for feedforward networks, which is also based on an alternation between two phases. While the approaches mentioned above focus on supervised learning solutions, a lot of attention on bio-inspired methods has converged on unsupervised learning. For example, the Similarity Matching criterion [8, 9, 10, 11] or the Hebbian PCA rule [12, 13] allow neurons to learn to extract the principal components from data. Similarly, Hebbian learning with Winner-Takes-All (WTA) competition allows neurons to find clusters in the data space [14, 15, 16, 17, 18, 19, 20]. This reveals interesting connections between the Hebbian theory of learning and data science.

In this work, we focus on a hybrid solution of unsupervised Hebbian learning and supervised backprop training, which are combined together in a semi-supervised fashion. In fact, supervised training alone has the disadvantage of requiring numerous training samples to achieve high performances, but the latter are often expensive to gather, requiring a consistent manual effort. To circumvent this issue, a possible direction is to pre-train the model on a large amount of unlabeled data, with an unsupervised algorithm, and then fine-tune with supervision on a small labeled dataset [21, 22]. In this scenario, recent work has shown superior performance of Hebbian-based semi-supervised

*Corresponding author
Email addresses: gabriele.lagani@isti.cnr.it (Gabriele Lagani), fabrizio.falchi@isti.cnr.it (Fabrizio Falchi), claudio.gennaro@isti.cnr.it (Claudio Gennaro), hannes.fassold@joanneum.at (Hannes Fassold), giuseppe.amato@isti.cnr.it (Giuseppe Amato)

training, compared to other unsupervised methods for pre-training, such as Variational Auto-Encoders (VAE) [23], especially in scenarios where the available labeled data is scarce [13, 17]. Due to the difficulty of collecting labeled data, these scenarios are of strong practical interest.

Despite their promising results, current Hebbian learning solutions can hardly be used to address large-scale problems, due to their demanding computational cost. In this perspective, the goal of our contribution is to address the performance limitations of Hebbian algorithms. For this purpose, we developed a new Hebbian learning solution, named *FastHebb*, which is designed to better take advantage of GPU acceleration. This is done in two steps. First, we notice that Hebbian learning with mini-batch processing evolves in two stages, one is the weight update computation for each sample in the mini-batch, and the other is the aggregation of updates over all the minibatch elements. These two phases can be merged together with a significant speedup. Second, the resulting Hebbian equations of synaptic updates can translated in terms of matrix multiplications, which can be executed very efficiently on GPU.

In order to provide an experimental evaluation of the proposed method, we used established computer vision benchmarks such as CIFAR10/100 [24], Tiny ImageNet [25] and ImageNet [26]. Besides the image classification evaluation, we also studied the performance of Hebbian neural features for Content-Based Image Retrieval (CBIR). We considered sample efficient learning scenarios, where label information is assumed to be available only for a certain percentage of the data used for training. Results confirm previous observation about the superior performance of Hebbian-based semi-supervised approaches, compared to alternative solutions, especially in label-scarce learning regimes. Moreover, the FastHebb solution exhibits a significant acceleration of training times, both compared to previous Hebbian learning solutions, and compared to backprop-based alternatives. In particular, FastHebb achieves a peak improvement in training up to 70 times faster than corresponding Hebbian approaches not leveraging FastHebb. This allowed to gracefully scale Hebbian algorithms to large datasets, on the scale of ImageNet, and architectures, on the scale of VGG [27]. Extending Hebbian learning to other types of architectures, such as residual networks [28] and transformers [29], in non-trivial and deserves to be explored in a separate work.

Some of the results on FastHebb were already presented in our recent conference publication [30]. However, those results were just preliminary and the aim of this paper is to significantly extend previous work. Compared to [30], this paper performs a more comprehensive evaluation of the method by considering two types of test scenarios, image classification and CBIR, over four different computer vision benchmarks, including ImageNet. Moreover, we also extend the range of backbone architectures on which the approach is applied, pushing Hebbian learning to VGG-scale architectures for the first time, as far as we know.

In summary, our contribution is twofold:

1. A scalable solution for Hebbian synaptic updates is proposed;
2. Extensive evaluation of Hebbian algorithms is presented, including new experiments on large-scale datasets (ImageNet) and architectures (VGG) which (to the best of our knowledge) have been out of reach for Hebbian algorithms so far.

Here is the structure of the following Sections: Section 2 illustrates related contributions; Section 3 describes the proposed FastHebb method more in detail; Section 4 delves into the details of our experimental scenarios in sample efficient and large-scale settings; in Section 5, the results of our experiments are described; conclusive remarks are presented in Section 6.

2. Background and related work

Some past contributions focused on addressing the biological plausibility problem of backpropagation by proposing solutions that can be shown to approximate backprop using Hebbian updates. Contrastive Hebbian Learning (CHL) [31] and Equilibrium Propagation (EP) [6] approaches do so by leveraging recurrent network architectures with two phases of activity. A free phase fixes the values of input neurons to represent a given sample, while output neurons and the remaining hidden units are left free. The recurrent dynamics lead the network to settle down into a steady state, where an anti-Hebbian update is performed. During the forced phase, the activations of output neurons are also fixed to a value closer to the desired target. Again, the recurrent dynamics will induce the network into another steady state, where a Hebbian update is performed. This combination of updates can be shown to approximate backprop at each neuron, using only local information. Another approach with strong biological support is Predictive Coding (PC) [32], in which a layer optimizes a local loss function that accounts for the error in predicting the next layer activations. Optimization is performed in a nested fashion. First, neuron activations are optimized to meet the objective, which leads to the emergence of recurrent interactions among neurons, followed by optimization of the weights. Again, the resulting updates can be shown to match backprop updates using only local information [33]. The PC approach has been successfully applied in different flavors to DNN training [34, 35, 36]. More recently, the Forward-Forward (FF) approach has been proposed [7]. This is based on standard feedforward architectures, but using an input vector composed of both sample and target. The approach alternates a positive phase, where sample and the correct target are provided to the network, which is required to maximize its activations, and a negative phase, where the sample is paired with a randomly generated target, and the network is required to minimize its activations. In a previous work [37], we have used a similar method for training biologically realistic models of *in vitro* cultured neural networks, where sample and target are provided simultaneously to the network, and Hebbian plasticity reinforces the connection between the two, so that at test time, when a sample with no target is provided, the network can recall the association. A weight normalization mechanism plays the role of the negative phase in this case.

In addition to these attempts to model supervised learning from a biologically plausible perspective, other efforts have been focused on modeling bio-inspired unsupervised learning mechanisms. Past works used Hebbian learning with WTA competition models to train feature extractors in feedforward and/or convolutional CNNs [14, 15, 16, 17, 18, 19, 20], showing impressive convergence speed. In particular, a recent work [38] showed that soft-WTA training of DNNs allows the network to extract increasingly abstract representations, in the same vein as backprop training, but at the cost of using very wide layers. The authors also provide experimental results with Hebbian learning on ImageNet, although only for a single training epoch. The method that we propose allows us to run a full training session (20 epochs or more) even on ImageNet scale. Miconi [39] proposed translations of some Hebbian synaptic update equations into optimizable objective functions, which are more relatable to common frameworks for DL. Another line of research explored the *Similarity Matching* objective as a possible direction to derive biologically plausible neural models for principal subspace extraction [8, 9, 10, 11], with extensions also to the supervised end-to-end recurrent training case [40, 41].

In our past contributions, we took a hybrid approach, and explored Hebbian WTA and PCA training of DNNs in semi-supervised scenarios, using unsupervised Hebbian algorithms as a tool for pre-training [12, 13, 42]. Experiments showed promising results, compared to backprop-based alternative methods, especially in scarce data learning scenarios. Due to the difficulty of gathering manually labeled data, these scenarios are of strong practical interest. Given the promising results obtained in previous works, in this contribution we further enhance previous solutions towards achieving higher efficiency and scalability to more complex scenarios.

Other works have explored semi-supervised approaches exploiting unsupervised pre-training with backprop-based auto-encoding architectures [43, 44, 21, 22]. A different direction towards semi-supervised learning is instead based on pseudo-labeling or consistency-based methods [45, 46, 47, 48]. Since our approach belongs to the unsupervised pre-training category, we will focus our comparisons in this setting. However, it is worth mentioning that the other approaches are not mutually exclusive with unsupervised pre-training, and, indeed, these can be integrated together, as also suggested in Sec. 6

3. Speeding Up Hebbian learning with FastHebb

In this Section we present the FastHebb method. We start by introducing a convenient notation, that will be used to translate Hebbian synaptic update equations into the FastHebb formulation. Then, we illustrate the learning rules that are analyzed in this work, and we derive their FastHebb-enhanced counterpart.

3.1. A convenient notation

When working with common packages for DL, such as Pytorch, data are typically represented as *tensors*. In this context, a tensor is simply a data array with multiple dimensions. We wish to introduce a notation for tensors that is better suited for relating mathematical formalism with the corresponding implementation in DL packages. For example, a tensor has a number of dimensions, whose interpretation lies in the mind of the programmer (e.g. batch, channel, height, and width dimensions for images). Given a tensor, packages such as Pytorch allow us to transpose or `permute` any of its dimensions, which corresponds to reordering indexes. We can also `unsqueeze` singleton dimensions or `squeeze` them out, which correspond to adding or removing *singleton* indexes, i.e. indexes of dimension 1. Therefore, in essence, the notation that we introduce is motivated by a more straightforward mapping to the programming formalism for working with tensors.

We consider tensors, denoted by capital letters, followed by one index per dimension. The symbol used to denote the index denotes its meaning. For example, we can use an index $p = 1...M$ to represent the *batch* dimension, an index $q = 1...Q$ to represent the *channel* dimension, and index $m = 1...M$ to represent the *weight vector* dimension. A 1 symbol used as an index represents a singleton. For example, a neural feature map at some layer can be denoted as $A_{p,q,1}$, which is a tensor with one element for each mini-batch element (dimension p) and for each neuron (dimension q). The last dimension is a singleton because the output of each neuron for each mini-batch element is a scalar. Similarly, a weight matrix can be denoted by $B_{1,q,m}$, because it does not extend along the mini-batch dimension, but it has a number of channels (one per neuron), and each corresponds to a weight vector. Concerning the input tensors, for example images, they have a mini-batch, channel, height, and width dimensions. However, due to the convolutional processing, a patch will be extracted from each horizontal and vertical location of each image, which is treated, for the Hebbian learning purposes, as a separate input. Therefore, our mini-batch is the collection of all patches extracted from all images. Each patch is flattened into a vector, whose size corresponds to the weight vector size of the next neural layer. On the other hand, this tensor does not extend along the neuron dimension. Overall, our input tensors can be represented as $C_{p,1,m}$.

This notation is convenient, because it allows us to easily swap indexes, or squeeze and unsqueeze singleton dimensions. If tensors have compatible dimensions, we can also perform element-wise operations (additions, multiplications, etc.). When a dimension is a singleton, it automatically undergoes *broadcasting*, i.e. the tensor is replicated along that direction until it matches the corresponding dimension of the other tensor involved in the operation.

Matrix multiplication plays an important role in DNN processing. We make the usage of matrix multiplications explicit in our notation, by writing $\mathrm{bmm}(\cdot,\cdot)$ (which stands for *batch matrix multiplication*):

$$\begin{aligned} Z_{i,j,l} &= \sum_k U_{i,j,k} V_{i,l,k} \\ &= \sum_k U_{i,j,k} V_{i,k,l} := \mathrm{bmm}(U_{i,j,k}, V_{i,k,l}) \end{aligned} \tag{1}$$

Notice that the matrix multiplication operation is equivalent to taking the element-wise product between tensors U and V, identifying the common index i and summing over (or *contracting*) index k. Specifically, index i represents a batch dimension, and the operation is a batch matrix multiplication between i matrix pairs. For each pair, the two matrices have indices (j,k) and (k,l), which are mapped to indices (j,l): $(j,k)\times(k,l) \rightarrow (j,l)$. The operation generalized to tensors with more that three dimensions as follows: all dimensions except the last two are considered as batch dimensions, while the last two dimensions represent rows and columns of the matrices being batch-multiplied. Batch dimensions must correspond between the two tensors, or be singleton (in which case, broadcasting takes place).

3.2. Hebbian synaptic updates: from computation to aggregation

We consider two types of Hebbian learning rules: Hebbian PCA (HPCA) and soft-Winner-Takes-All (SWTA). In this paper, we just give the definition of these learning rules, but the interested reader can find more details in [20, 19, 13, 12].

Given a layer of neurons whose activations are denoted by a_q (index q refers to the q-th neuron in the layer), whose weight vectors are denoted by $\mathbf{b}_q$, and whose input vector is denoted by $\mathbf{c}$, the SWTA synaptic update equation is the following:

$$\Delta \mathbf{b}_q = \alpha\, s_q\, (\mathbf{c} - \mathbf{b}_q) \tag{2}$$

Where α stands for the learning rate and s_q is the softmax of the activations with temperature T [49]:

$$s_q = \frac{a_q/T}{\sum_k a_k/T} \tag{3}$$

Essentially, this modulates the update steps so that neurons with higher activations will also 'win' larger updates.

The HPCA learning rule, instead, is the following:

$$\Delta \mathbf{b}_q = \alpha\, a_q\, (\mathbf{c} - \sum_{k=1}^{q} a_k \mathbf{b}_k) \tag{4}$$

This rule can be shown to induce neurons to extract the principal components from data [50, 51].

When working with images and convolutional layers, these weight updates need to be computed for each patch extracted from a given image. However, due to the constraints of convolutional layers, neurons at different offsets need to maintain shared weights, hence they are bound to follow the same synaptic modifications. This can be achieved by aggregating the different weight updates, obtained from patches at different locations, into a unique update. Aggregation needs to be performed for all the images in a mini-batch as well. The overall two-phases approach is depicted in Fig. 1.

Aggregation is performed simply by averaging, in the HPCA case, or by a weighted average, where the weights are s_q coefficients, for SWTA.

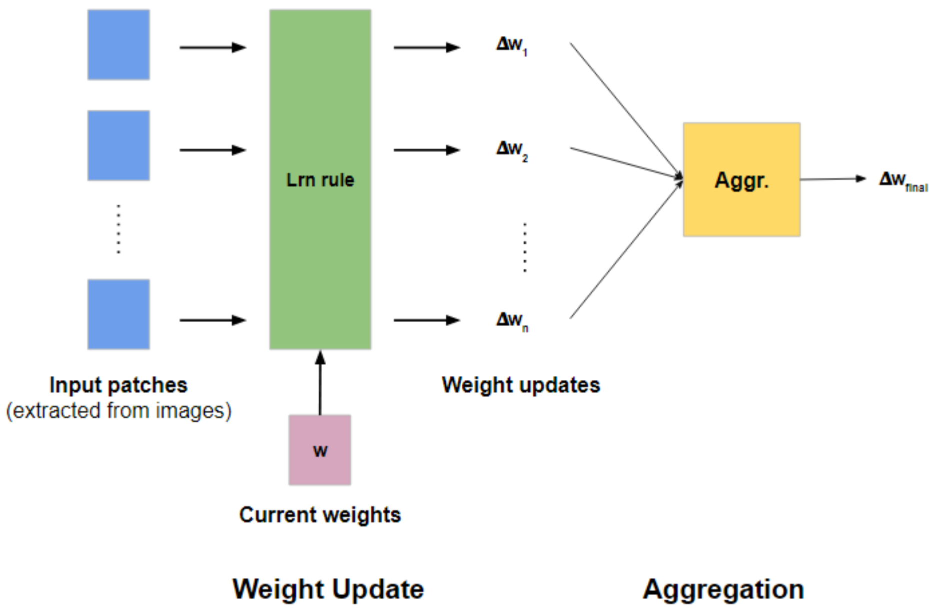

Figure 1: Two phases of weight update: update computation for each patch, followed by aggregation of several updates.

3.3. *From Hebbian synaptic updates to FastHebb*

The Hebbian rules presented above can be rewritten in matrix form, including the aggregation step, using the notation outlined at the beginning of this Section:

$$\Delta B_{1,q,m} = \sum_p D_{p,q,1}\, \Delta B_{p,q,m} = \mathrm{bmm}(D_{q,1,p}, \Delta B_{q,p,m}) \tag{5}$$

where $\Delta B_{p,q,m}$ is the collection of all weight updates that need to be aggregated, and $D_{p,q,1}$ is the tensor of coefficients for the aggregation.

Now that the two phases of weight computation and aggregation are merged together, we proceed differently depending on the specific learning rule.

FastHebb for SWTA. Let's rewrite the SWTA update rule as follows:

$$\begin{aligned}\Delta B_{1,q,m} &= \alpha \sum_p D_{p,q,1}\, S_{p,q,1} \left(C_{p,1,m} - B_{1,q,m}\right)\\ &= \alpha \sum_p (D\,S)_{p,q,1}\, (C - B)_{p,q,m}\\ &= \alpha\, \mathrm{bmm}\big((D\,S)_{q,1,p}, (C - B)_{q,p,m}\big)\end{aligned} \tag{6}$$

Where $D_{p,q,1} = \frac{S_{p,q,1}}{\sum_p S_{p,q,1}}$.

The computational complexity required by this algorithm is $O(P\,Q\,M)$ in time. Moreover, if we wish to exploit GPU parallelism, we need to keep a $P \times Q \times M$ tensor in memory, thus requiring $O(P\,Q\,M)$ space complexity as well, which can be prohibitive for large-scale scenarios.

However, it is possible to improve over these bounds by contracting the aggregation index p (which is typically the largest dimension) early:

$$\begin{aligned}\Delta B_{1,q,m} &= \alpha \sum_p D_{p,q,1}\, S_{p,q,1} \left(C_{p,1,m} - B_{1,q,m}\right)\\ &= \alpha \sum_p (D\,S)_{p,q,1}\, C_{p,1,m} - \alpha \sum_p (D\,S)_{p,q,1}\, B_{1,q,m}\\ &= \alpha\, \mathrm{bmm}\big((D\,S)_{1,q,m}, C_{1,q,m}\big) - \alpha \sum_p (D\,S)_{p,q,1}\, B_{1,q,m}\\ &= \alpha\, \mathrm{bmm}\big((D\,S)_{1,q,p}, C_{1,p,m}\big) - \alpha\, E_{1,q,1}\, B_{1,q,m}\end{aligned} \tag{7}$$

Where $E_{1,q,1} = \sum_p (D\,S)_{p,q,1}$.

This requires only $O(Q\,(P + M))$ space complexity. Concerning the time complexity, this depends on the specific matrix multiplication algorithm adopted, but this can be made lower than $O(P\,Q\,M)$. This is the FastHebb formulation for SWTA.

FastHebb for HPCA. Similarly to the SWTA case, we can rewrite the HPCA equation, together with the aggregation phase (in this case, the coefficient $D_{p,q,1}$ is just $\frac{1}{P}$), with the proposed notation:

$$\begin{aligned}\Delta B_{1,q,m} &= \alpha \frac{1}{P} \sum_{p} A_{p,q,1} \left(C_{p,1,m} - \sum_{q'=1}^{q} A_{p,q',1} \, B_{1,q',m} \right) \\ &= \alpha \frac{1}{P} \sum_{p} A_{p,q,1} \left(C_{p,1,m} - \sum_{q'=1}^{Q} T_{q,q'} \, A_{p,q',1} \, B_{1,q',m} \right) \\ &= \alpha \frac{1}{P} \sum_{p} A_{p,q,1} \, F_{p,q,m} \\ &= \alpha \frac{1}{P} \mathrm{bmm}\Big(A_{q,1,p}, F_{q,p,m} \Big)\end{aligned} \tag{8}$$

Where $F_{p,q,m} = \left(C_{p,1,m} - \sum_{q'=1}^{Q} T_{q,q'} \, A_{p,q',1} \, B_{1,q',m} \right)$, and $T_{q,q'}$ is simply a lower-triangular matrix with all ones on and below the main diagonal and all zeros above.

The computation of the HPCA equation is slightly more complex, requiring $O(PQ^2M)$ space and time, but this can be improved by reordering the sums:

$$\begin{aligned}\Delta B_{1,q,m} &= \alpha \frac{1}{P} \sum_{p} A_{p,q,1} \left(C_{p,1,m} - \sum_{q'=1}^{Q} T_{q,q'} \, A_{p,q',1} \, B_{1,q',m} \right) \\ &= \alpha \frac{1}{P} \sum_{p} A_{p,q,1} \, C_{p,1,m} \\ &- \alpha \frac{1}{P} \sum_{p} A_{p,q,1} \sum_{q'=1}^{Q} T_{q,q'} \, A_{p,q',1} \, B_{1,q',m} \\ &= \alpha \frac{1}{P} \mathrm{bmm}\Big(A_{1,q,p}, C_{1,p,m} \Big) \\ &- \alpha \frac{1}{P} \sum_{q'=1}^{Q} \sum_{p} A_{p,q,1} \, A_{p,q',1} \, T_{q,q'} \, B_{1,q',m} \\ &= \alpha \frac{1}{P} \mathrm{bmm}\Big(A_{1,q,p}, C_{1,p,m} \Big) \\ &- \alpha \frac{1}{P} \sum_{q'=1}^{Q} \mathrm{bmm}\Big(A_{1,q,p}, A_{1,p,q'} \Big) \, T_{q,q'} \, B_{1,q',m} \\ &= \alpha \frac{1}{P} \mathrm{bmm}\Big(A_{1,q,p}, C_{1,p,m} \Big) \\ &- \alpha \frac{1}{P} \sum_{q'=1}^{Q} G_{1,q,q'} \, B_{1,q',m} \\ &= \alpha \frac{1}{P} \mathrm{bmm}\Big(A_{1,q,p}, C_{1,p,m} \Big) \\ &- \alpha \frac{1}{P} \mathrm{bmm}\Big(G_{1,q,q'}, B_{1,q',m} \Big)\end{aligned} \tag{9}$$

Where, $G_{1,q,q'} = \mathrm{bmm}\Big(A_{1,q,p}, A_{1,p,q'} \Big) \, T_{q,q'}$.

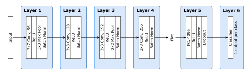

Figure 2: Backbone neural network model used for our experiments.

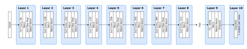

Figure 3: Bigger neural network model for ImageNet experiments.

The overall computation now has $O(Q^2 + QM)$ complexity in space, and up to $O(PQM + PQ^2 + Q^2M)$ in time. This is the FastHebb formulation for HPCA.

4. Evaluation scenario

We evaluated the proposed methodology on a number of established computer vision benchmarks: CIFAR10/100 [24], Tiny ImageNet [25], and ImageNet [26]. We performed an evaluation of Hebbian-based approaches in semi-supervised learning settings, on a backbone network model described in the following, compared to a Variational Auto-Encoder (VAE) [23, 21] baseline. In addition, we provide a FastHebb evaluation on VGG [27], to show the scalability of the proposed approach to large architectures. We evaluated the performance both in terms of classification accuracy, and in terms of training speedup achieved with FastHebb. We also provide an evaluation of Hebbian neural features for large-scale image retrieval tasks.

4.1. Neural network backbone for evaluation

In order to provide an evaluation for the proposed approach, we need to define a suitable backbone network architecture for our experiments. For this purpose, we need a network that presents the common architectural features of Convolutional Neural Networks (pooling and convolutional layers [52], batch normalization [53], etc.). On the other hand, we need to exclude more recent features such as residual connections [28] or attention layers [29], for Hebbian algorithms are not trivial to generalize to these cases, which deserve to be analyzed in a separate work. For a first experimentation stage, we do not need to consider a very large model; it is instead preferable to consider a more compact architecture, which enables faster experimentation, and easier analysis of deep features, also on a layer-by-layer basis. It also makes reproducibility by other researchers more accessible. Therefore, we opted for an AlexNet-inspired [52] architecture shown in Fig. 2, with 6 layers, which is also consistent with previous works [20, 30]. For larger-scale experiments on ImageNet we used an extended version of the previous model with 10 layers, shown in Fig. 3, as well as a VGG model [27].

4.2. *Semi-supervised training protocol for sample-efficient learning*

We evaluated the proposed approach assuming a condition of scarcity of available labeled training data. We define a *sample efficiency* regime as the percentage of available labeled samples, over the total number of training samples. For each of the considered datasets, we performed experiments in eight different sample efficiency regimes: 1%, 2%, 3%, 4%, 5%, 10%, 25%, and 100%.

In order to take advantage of both labeled and unlabeled training samples, for each sample efficiency regime, we followed a semi-supervised training protocol in two phases: first, the network is pre-trained using one of the proposed unsupervised Hebbian algorithms, exploiting all the available training samples; second, end-to-end fine-tuning is performed, using supervised backprop training on a cross-entropy loss, and exploiting the labeled samples only. Finally, both the resulting classification accuracy and the training time (in terms of epoch duration, number of epochs, and total duration) were recorded.

As a baseline for comparison, we used unsupervised pre-training based on the Variational Auto-Encoder (VAE) approach [54]. In this case, pre-training was performed by using the deep layers (excluding the final classifier) of the proposed architectures as encoder, mapping their output to 256 gaussian latent variables. This was augmented with a another network branch, acting as decoder, with a specular structure w.r.t. the encoder (i.e. pooling layers replaced with unpooling, and convolutions with transpose convolutions), mapping the latent variables to a decoded sample. The overall models were trained in the encoding-decoding task, optimizing the β-VAE Variational Lower Bound [55], in an end to end fashion, using all the available training samples. At this point, the decoder was dropped, a linear classifier was placed on top of the latent features, and supervised backprop-based end-to-end fine tuning was performed, using only the available labeled samples for the given sample efficiency regime. Essentially, this is the standard semi-supervised training approach based on state-of-the-art VAE architectures [21]. Notice that in this case, however, the pre-training phase, even if unsupervised, is still backprop-based, while Hebbian algorithms enable pre-training without requiring backprop.

4.3. *Retrieval with neural features*

Deep features extracted from pre-trained networks were also used as vector descriptors for multimedia content indexing and retrieval [56, 57, 58]. The performance of the resulting feature representation was evaluated in Content-Based Image Retrieval (CBIR) tasks.

The CBIR systems architecture works as follows: in a first phase, feature representations are computed for all images in a given database, by extracting the deep representations from the convolutional part of the network. These feature representations are then mapped to a binary 256-dimensional descriptor which is then used for indexing the database images. This is done as in [57] by training another piece of network, with a 256 units hidden layer with tanh activations and a final classifier. This is trained in the classification task, so that the feature representation is mapped to the correct class, but passing through a compression stage into the desired 256 dimensional vector. The tanh activation is a "soft" proxy for the binarization operation, which doesn't block gradients from flowing backward during training. The 256 dimensional representation is then binarized by a thresholding operation: positive values are mapped to 1 and negative values are mapped to 0.

Test set images are used as sample queries: at test time, their 256-dimensional binary feature representation is computed as well, and the database images are ranked against the query based on the Hamming distance between feature representations. Retrieved images are considered to be a correct match if they belong to the same class as the query.

The evaluation measure used for the CBIR task is the Average Precision Score (APS) :

$$APS = \sum_{i=1}^{K} P_i \, (R_i - R_{i-1}) \tag{10}$$

where P_i is the precision at the i^{th} retrieved item, R_i is the corresponding recall. This score is renormalized (so that its maximum value is always 1) and averaged over all the queries, thus obtaining the mean Average Precision (mAP).

4.4. *Implementation details*

The experiments, implemented in Pytorch, depend on a number of hyperparameters, whose search was pursued by Coordinate Descent (CD) [59], optimizing, for each dataset, the accuracy results of the trained models on the respective validation set. In the following, the resulting parameters and implementation details are illustrated.

All training sessions were performed over 20 epochs (which were enough for the models to converge). Data were processed in mini-batches of 64 samples each, and each sample was an RGB image of 32 pixels in height and width for the 6-layer CIFAR10/100 and Tiny ImageNet network, 210 pixels for the 10-layer ImageNet network, and 224 pixles for VGG (specifically, the VGG-11 model was used), pre-normalized to zero mean and unit variance.

Concerning Hebbian pre-training, the learning rate parameter was set to 10^{-3}. For ImageNet training, we also introduced an adaptive learning rate mechanism to cope with the high variance of weight updates due to the high dimensionality of the feature maps (causing instability during training), which divides the learning rate by the square root of the input size (this corresponds to normalizing the output variance, assuming the inputs are normalized). For SWTA training only, whitening pre-processing was also necessary, as in [60, 24], although this step did not show any benefit on the other approaches. SWTA uses 0.02 as inverse temperature parameter $1/T$.

Batch-norm layers used momentum 0.9.

Backprop-based training (i.e. both fine-tuning and VAE pre-training) leveraged Stochastic Gradient Descent (SGD) optimization with learning rate 10^{-3}, and momentum 0.9, with Nesterov acceleration [61]. After 10 training epochs, learning rate was reduced by half every 2 epochs until the end of the training session. The best training epoch in terms of validation results was then selected as final model (early stopping).

β-VAE training used coefficient $\beta = 0.5$.

Supervised fine-tuning was regularized by dropout with 0.5 rate, and L_2 weight decay with penalty equal to $5 \cdot 10^{-2}$, 10^{-2}, $5 \cdot 10^{-3}$, $1 \cdot 10^{-3}$, for CIFAR10, CIFAR100, Tiny ImageNet, ImageNet, respectively.

The implementation used Pytorch version 1.8.1 and Python 3.7, with an Ubuntu 20.4 system running on an I7 series 10700K Intel Processor, 32GB RAM, and 12GB NVidia Geforce 3060 GPU.

5. Results

The results of the experiments described in the previous Section are illustrated hereafter. First, we report the recorded training speed, in terms of epoch duration, number of epochs for convergence, and total duration, on CIFAR10/100, Tiny ImageNet, and ImageNet datasets, comparing VAE pre-training, ordinary Hebbian learning, and FastHebb. Second, we report the classification and retrieval results of the various approaches in the label-scarcity scenarios described earlier. Finally, we report the results on the VGG architecture as well. The results were obtained from averaging five independent experiment iterations, and t-testing confirmed the observed differences to be statistically significant with p-values below 0.05.

5.1. Training speed analysis

Table 1 shows a comparison between the considered approaches in terms of computational performance of training, on the 10-layer (for ImageNet) and 6-layer (for the other datasets) architectures. The Table shows the single epoch duration, the number of epochs until convergence (measured as the point after which validation results stop improving), and the total training duration [1]. These results are specifically focused on the pre-training duration, while we observed no statistically significant difference in the duration of the successive fine-tuning phase for different pre-training approaches.

We can observe that FastHebb methods are significanly faster (up to 50 times for HPCA and HPCA-FH on ImageNet) than the traditional Hebbian counterparts, with an epoch duration becoming comparable to backprop-based VAE training. This enables Hebbian approaches to scale gracefully to complex datasets such as ImageNet, where the best speed-up by a factor of 50, in terms of epoch duration, is observed for HPCA. Moreover, the overall training duration of Hebbian approaches becomes faster (up to 5 times on ImageNet) than VAE, thanks to the lower number of epochs required to convergence. Among the Hebbian approaches, soft-WTA has lower time complexity, and it is in fact faster.

5.2. Label scarcity results

Table 2 illustrates the classification results, in terms of accuracy (top-1 for CIFAR10, and top-5 for the other datasets, since they contain many more classes), in various sample efficiency regimes, comparing the alternative approaches. Notice that in the results for HPCA and SWTA there is no difference between using FastHebb or not. In fact, despite the computational speedup, from the algebraic point of view FastHebb is equivalent to ordinary Hebbian learning, leading to the same results. Therefore, we show these results just once.

[1] For Hebbian approaches not using FastHebb, the training duration would be unfeasible to measure explicitly; instead, it was estimated by multiplying the single epoch duration by the number of epochs

Table 1: Analysis of algorithm performance on each dataset, for VAE, Hebbian PCA (HPCA), Hebbian PCA with FastHebb (HPCA-FH), soft-WTA (SWTA), and soft-WTA with FastHebb (SWTA-FH) methods, on the 10-layer (for ImageNet) and 6-layer (for the other datasets) networks.

Dataset	Method	Epoch Duration	Num. Epochs	Total Duration
CIFAR10	VAE	14s	17	3m 58s
	SWTA	4m 14s	1	4m 14s
	SWTA-FH	18s	1	**18s**
	HPCA	6m 23s	12	1h 16m 36s
	HPCA-FH	19s	12	3m 48s
CIFAR100	VAE	15s	15	3m 45s
	SWTA	4m 16s	1	4m 16s
	SWTA-FH	18s	1	**18s**
	HPCA	6m 25s	7	44m 55s
	HPCA-FH	19s	7	2m 13s
Tiny ImageNet	VAE	33s	20	11m
	SWTA	9m 41s	1	9m 41s
	SWTA-FH	41s	1	**41s**
	HPCA	14m 20s	14	3h 20m 40s
	HPCA-FH	43s	14	10m 2s
ImageNet	VAE	2h 59m 19s	16	47h 49m 4s
	SWTA	105h 13m 24s	3	315h 40m 12s
	SWTA-FH	3h 38m 6s	3	**10h 54m 18s**
	HPCA	155h 41m 39s	3	467h 4m 57s
	HPCA-FH	3h 39m 18s	3	10h 57m 54s

The results show that, in conditions of label scarcity (sample efficiency regimes below 4-5%), Hebbian approaches perform significantly better than VAE. On the other hand, it is only when far more labels are available for the supervised fine-tuning phase that VAE-based pre-training really kicks in. In these scenarios, however, the performance of Hebbian approaches is comparable or only slightly lower, but this is compensated by the speedup in training time observed before. Comparing HPCA and SWTA, it appears that the former performs typically better.

5.3. Retrieval experiments

Table 3 shows the retrieval mAP results obtained on the various dataset, for each of the considered approaches, on the 10-layer (for ImageNet) and 6-layer (for the other datasets) architectures.

This second scenario confirms the previous observations that, in conditions of extreme label scarcity (below 10%), Hebbian-based neural features achieve better results than VAE counterparts. Again, VAE-based pre-training improves in higher regimes, but, as observed before, this is fairly compensated by the training time advantage of Hebbian approaches. Comparing HPCA and SWTA, also in this case it appears that the former performs typically better.

5.4. Experiments on VGG

In Tab. 5, we report the training times required for the pre-training phase of VGG models using the different approaches considered so far. We do not consider VAE-type training of the VGG model, because that requires a large decoder, making the overall model very deep, which we found to be untrainable due to vanishing gradients [62, 63]. On the other hand, Hebbian pre-training was straightforward to apply in this case, as it requires no gradient backpropagation. Instead, as a baseline for comparison, we used Xavier initialization [64] (note that, since this is not properly a training method, it is not included in Tab. 5). In fact, it is known that appropriate initialization methods can achieve competitive results compared to end-to-end pre-training [64, 65].

Training times show once more the effectiveness of FastHebb methods in training large scale architectures, while using ordinary Hebbian learning would be unfeasible. In the best case, a speedup of almost 70 times is reached, comparing HPCA-FH with HPCA.

Finally, in Tab. 5, we report the results, both in terms of classification accuracy and retrieval mAP, achieved by training the VGG model in the semi-supervised task. We show the results achieved with Xavier initialization, HPCA pre-training, and SWTA pre-training.

When the scale of the architecture increases, it ap-

Table 2: Accuracy results on each dataset (top-1 for CIFAR10, and top-5 for the other datasets, since they have many more classes), for the various approaches explored, on the 10-layer (for ImageNet) and 6-layer (for the other datasets) networks.

Regime	Method	CIFAR10	CIFAR100	Tiny ImageNet	ImageNet
1%	VAE	22.54	12.28	5.55	2.72
	SWTA	30.23	15.30	6.20	6.69
	HPCA	**39.75**	**22.63**	**11.38**	**8.65**
2%	VAE	26.78	15.25	6.74	6.14
	SWTA	36.59	20.76	8.56	11.52
	HPCA	**45.51**	**30.83**	**15.71**	**13.64**
3%	VAE	29.00	16.44	7.74	15.35
	SWTA	41.54	23.69	10.26	15.67
	HPCA	**48.80**	**35.04**	**18.23**	**17.28**
4%	VAE	31.15	17.89	8.45	**23.97**
	SWTA	45.31	26.91	11.52	19.95
	HPCA	**51.28**	**38.89**	**20.55**	20.39
5%	VAE	32.75	18.48	9.29	**29.04**
	SWTA	48.35	29.57	12.55	24.87
	HPCA	**52.20**	**41.42**	**22.46**	23.28
10%	VAE	45.67	23.80	13.51	**43.73**
	SWTA	**58.00**	38.26	16.70	41.54
	HPCA	57.35	**48.93**	**28.13**	34.27
25%	VAE	68.70	52.59	**37.89**	**61.33**
	SWTA	**69.85**	56.26	24.96	59.34
	HPCA	64.77	**58.70**	37.10	56.92
100%	VAE	85.23	**79.97**	**60.23**	76.84
	SWTA	**85.37**	79.80	54.94	76.10
	HPCA	84.38	74.42	53.96	**77.28**

pears that SWTA approach improves over HPCA. Previous observations about Hebbian methods performing better in low sample efficiency regimes (5% and below) are confirmed. In particular, SWTA outperforms the network with no pre-training by a margin up to 5 percent points in accuracy, in the 1-2% sample efficiency regimes. In terms of mAP, Hebbian pre-training is still slightly superior, although the difference is not statistically significant.

6. Conclusions and future work

In this article, we have illustrated the FastHebb approach for Hebbian learning, which leverages a matrix multiplication formulation of Hebbian synaptic updates to achieve higher efficiency. This makes Hebbian learning more scalable, enabling teh use of Hebbian neural features also on large datasets (ImageNet) and architectures (VGG), which (to the best of our knowledge) have been computationally prohibitive for Hebbian learning so far. Experimental scenarios of label scarcity show promising results of Hebbian pre-training compared to backprop-based alternatives such as VAE, considering classification accuracy, retrieval mAP, and training time.

Even though, in this paper, we have shown that it is possible to scale Hebbian training to large models such as VGG, further work needs to be done to adapt Hebbian approaches to more recent architectures, such as residual networks [28] or Transformers [29]. Moreover, additional performance improvements might come from the combination of Hebbian-based pre-training with pseudo-labeling and consistency-based semi-supervised methods [66, 48]. Finally, in line with recent efforts towards backprop-free learning (such as the Forward-Forward algorithm [7]), we plan to explore strategies to combine Hebbian approaches with local supervision signals.

Acknowledgment

This work was partially supported by:
- Tuscany Health Ecosystem (THE) Project (CUP I53C22000780001), funded by the National Recovery and Resilience Plan (NRPP), within the NextGeneration

Table 3: mAP results on each dataset, for the various approaches explored, on the 10-layer (for ImageNet) and 6-layer (for the other datasets) networks.

Regime	Method	CIFAR10	CIFAR100	Tiny ImageNet	ImageNet
1%	VAE	21.90	6.10	3.18	0.95
	SWTA	28.29	8.16	3.80	1.94
	HPCA	**37.39**	**15.39**	**6.61**	**4.88**
2%	VAE	23.13	6.40	3.20	1.39
	SWTA	31.71	8.82	3.95	2.54
	HPCA	**39.80**	**15.98**	**7.39**	**5.72**
3%	VAE	24.27	6.36	3.52	2.02
	SWTA	34.54	9.13	4.16	3.21
	HPCA	**41.64**	**16.40**	**7.57**	**6.56**
4%	VAE	24.36	6.36	3.57	3.83
	SWTA	36.54	9.41	4.32	3.77
	HPCA	**43.22**	**16.72**	**7.61**	**7.19**
5%	VAE	24.65	6.49	3.57	5.49
	SWTA	38.69	9.66	4.39	4.19
	HPCA	**44.92**	**17.10**	**7.79**	**7.75**
10%	VAE	28.26	7.09	3.76	**13.27**
	SWTA	48.55	11.27	5.01	9.14
	HPCA	**49.99**	**18.42**	**8.45**	10.56
25%	VAE	62.30	13.69	7.59	**24.50**
	SWTA	**63.95**	16.37	6.98	20.75
	HPCA	58.81	**21.49**	**10.13**	21.24
100%	VAE	**84.67**	42.83	**22.64**	**44.21**
	SWTA	84.54	**43.95**	20.86	39.60
	HPCA	81.80	36.19	17.99	43.81

Europe (NGEU) Program;
- AI4Media project, funded by the EC (H2020 - Contract n. 951911);
- INAROS (INtelligenza ARtificiale per il mOnitoraggio e Supporto agli anziani) project co-funded by Tuscany Region POR FSE CUP B53D21008060008.

References

[1] S. Haykin, Neural networks and learning machines, 3rd Edition, Pearson, 2009.

[2] W. Gerstner, W. M. Kistler, Spiking neuron models: Single neurons, populations, plasticity, Cambridge university press, 2002.

[3] C. Gamrat, O. Bichler, D. Roclin, Memristive based device arrays combined with spike based coding can enable efficient implementations of embedded neuromorphic circuits, 2015 IEEE International Electron Devices Meeting (IEDM) 2016 (2015) 4–5.

[4] X. Wu, V. Saxena, K. Zhu, S. Balagopal, A cmos spiking neuron for brain-inspired neural networks with resistive synapses andin situlearning, IEEE Transactions on Circuits and Systems II: Express Briefs 62 (11) (2015) 1088–1092.

[5] J. R. Movellan, Contrastive hebbian learning in the continuous hopfield model, in: Connectionist models, Elsevier, 1991, pp. 10–17.

[6] B. Scellier, Y. Bengio, Equilibrium propagation: Bridging the gap between energy-based models and backpropagation, Frontiers in computational neuroscience 11 (2017) 24.

[7] G. Hinton, The forward-forward algorithm: Some preliminary investigations, arXiv preprint arXiv:2212.13345 (2022).

[8] C. Pehlevan, T. Hu, D. B. Chklovskii, A hebbian/anti-hebbian neural network for linear subspace learning: A derivation from multidimensional scaling of streaming data, Neural computation 27 (7) (2015) 1461–1495.

[9] C. Pehlevan, D. Chklovskii, A normative theory of adaptive dimensionality reduction in neural networks, in: Advances in neural information processing systems, 2015, pp. 2269–2277.

[10] C. Pehlevan, D. B. Chklovskii, Optimization theory of hebbian/anti-hebbian networks for pca and whitening, in: 2015 53rd Annual Allerton Conference on Communication, Control, and Computing (Allerton), IEEE, 2015, pp. 1458–1465.

[11] Y. Bahroun, A. Soltoggio, Online representation learning with single and multi-layer hebbian networks for image classification, in: International Conference on Artificial Neural Networks, Springer, 2017, pp. 354–363.

[12] G. Lagani, F. Falchi, C. Gennaro, G. Amato, Hebbian semi-supervised learning in a sample efficiency setting, Neural Networks 143 (2021) 719–731.

[13] G. Lagani, F. Falchi, C. Gennaro, G. Amato, Evaluating hebbian learning in a semi-supervised setting, in: International Conference on Machine Learning, Optimization, and Data Science, Springer, 2021, pp. 365–379.

[14] A. Wadhwa, U. Madhow, Bottom-up deep learning using the

Table 4: Comparison of ImageNet training times, for Hebbian PCA (HPCA), Hebbian PCA with FastHebb (HPCA-FH), soft-WTA (SWTA), and soft-WTA with FastHebb (SWTA-FH) methods, on VGG-11.

Dataset	Method	Epoch Duration	Num. Epochs	Total Duration
ImageNet	SWTA	290h 44m	5	1454h
	SWTA-FH	7h 7m	5	**35h 35m**
	HPCA	453h 32m	13	5896h
	HPCA-FH	6h 25m	13	83h 25m

hebbian principle (2016).

[15] D. Krotov, J. J. Hopfield, Unsupervised learning by competing hidden units, Proceedings of the National Academy of Sciences 116 (16) (2019) 7723–7731.

[16] T. Moraitis, D. Toichkin, Y. Chua, Q. Guo, Softhebb: Bayesian inference in unsupervised hebbian soft winner-take-all networks, arXiv preprint arXiv:2107.05747 (2021).

[17] M. Gupta, S. K. Modi, H. Zhang, J. H. Lee, J. H. Lim, Is bio-inspired learning better than backprop? benchmarking bio learning vs. backprop, arXiv preprint arXiv:2212.04614 (2022).

[18] G. Amato, F. Carrara, F. Falchi, C. Gennaro, G. Lagani, Hebbian learning meets deep convolutional neural networks, in: International Conference on Image Analysis and Processing, Springer, 2019, pp. 324–334.

[19] G. Lagani, F. Falchi, C. Gennaro, G. Amato, Training convolutional neural networks with competitive hebbian learning approaches, in: International Conference on Machine Learning, Optimization, and Data Science, Springer, 2021, pp. 25–40.

[20] G. Lagani, F. Falchi, C. Gennaro, G. Amato, Comparing the performance of hebbian against backpropagation learning using convolutional neural networks, Neural Computing and Applications 34 (8) (2022) 6503–6519.

[21] D. P. Kingma, S. Mohamed, D. Jimenez Rezende, M. Welling, Semi-supervised learning with deep generative models, Advances in neural information processing systems 27 (2014) 3581–3589.

[22] Y. Zhang, K. Lee, H. Lee, Augmenting supervised neural networks with unsupervised objectives for large-scale image classification, in: International conference on machine learning, 2016, pp. 612–621.

[23] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980 (2014).

[24] A. Krizhevsky, G. Hinton, Learning multiple layers of features from tiny images (2009).

[25] J. Wu, Q. Zhang, G. Xu, Tiny imagenet challenge, Tech. rep., Stanford University (2017).

[26] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, L. Fei-Fei, Imagenet: A large-scale hierarchical image database, in: 2009 IEEE conference on computer vision and pattern recognition, Ieee, 2009, pp. 248–255.

[27] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, arXiv preprint arXiv:1409.1556 (2014).

[28] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.

[29] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, I. Polosukhin, Attention is all you need, in: Advances in neural information processing systems, 2017, pp. 5998–6008.

[30] G. Lagani, C. Gennaro, H. Fassold, G. Amato, Fasthebb: Scaling hebbian training of deep neural networks to imagenet level, in: International Conference on Similarity Search and Applications, Springer, 2022, pp. 251–264.

[31] X. Xie, H. S. Seung, Equivalence of backpropagation and contrastive hebbian learning in a layered network, Neural computation 15 (2) (2003) 441–454.

[32] R. P. Rao, D. H. Ballard, Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects, Nature neuroscience 2 (1) (1999) 79–87.

[33] J. C. Whittington, R. Bogacz, An approximation of the error backpropagation algorithm in a predictive coding network with local hebbian synaptic plasticity, Neural computation 29 (5) (2017) 1229–1262.

[34] H. Wen, K. Han, J. Shi, Y. Zhang, E. Culurciello, Z. Liu, Deep predictive coding network for object recognition, arXiv preprint arXiv:1802.04762 (2018).

[35] K. Han, H. Wen, Y. Zhang, D. Fu, E. Culurciello, Z. Liu, Deep predictive coding network with local recurrent processing for object recognition, in: Advances in Neural Information Processing Systems, 2018, pp. 9201–9213.

[36] V. Boutin, A. Franciosini, F. Ruffier, L. Perrinet, Meaningful representations emerge from sparse deep predictive coding, arXiv preprint arXiv:1902.07651 (2019).

[37] G. Lagani, R. Mazziotti, F. Falchi, C. Gennaro, G. M. Cicchini, T. Pizzorusso, F. Cremisi, G. Amato, Assessing pattern recognition performance of neuronal cultures through accurate simulation, in: 2021 10th International IEEE/EMBS Conference on Neural Engineering (NER), IEEE, 2021, pp. 726–729.

[38] A. Journé, H. G. Rodriguez, Q. Guo, T. Moraitis, Hebbian deep learning without feedback, arXiv preprint arXiv:2209.11883 (2022).

[39] T. Miconi, Multi-layer hebbian networks with modern deep learning frameworks, arXiv preprint arXiv:2107.01729 (2021).

[40] D. Obeid, H. Ramambason, C. Pehlevan, Structured and deep similarity matching via structured and deep hebbian networks, in: Advances in Neural Information Processing Systems, 2019, pp. 15403–15412.

[41] S. Qin, N. Mudur, C. Pehlevan, Supervised deep similarity matching, arXiv preprint arXiv:2002.10378 (2020).

[42] G. Lagani, D. Bacciu, C. Gallicchio, F. Falchi, C. Gennaro, G. Amato, Deep features for cbir with scarce data using hebbian learning, arXiv preprint arXiv:2205.08935 (2022).

[43] Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle, Greedy layer-wise training of deep networks, in: Advances in neural information processing systems, 2007, pp. 153–160.

[44] H. Larochelle, Y. Bengio, J. Louradour, P. Lamblin, Exploring strategies for training deep neural networks., Journal of machine learning research 10 (1) (2009).

[45] Q. Xie, Z. Dai, E. Hovy, M.-T. Luong, Q. V. Le, Unsupervised data augmentation for consistency training, arXiv preprint arXiv:1904.12848 (2019).

[46] D. Berthelot, N. Carlini, I. Goodfellow, N. Papernot, A. Oliver, C. Raffel, Mixmatch: A holistic approach to semi-supervised learning, arXiv preprint arXiv:1905.02249 (2019).

[47] D. Berthelot, N. Carlini, E. D. Cubuk, A. Kurakin, K. Sohn,

Table 5: Accuracy results on ImageNet (top-5), and retrieval mean Average Precision (mAP) for the various approaches explored, on the VGG network.

Regime	Pre-train	Accuracy (%)	mAP (%)
1%	None	14.71	4.79
	SWTA	**19.40**	3.63
	HPCA	15.53	**5.16**
2%	None	26.88	6.42
	SWTA	**31.91**	5.18
	HPCA	27.24	**6.70**
3%	None	36.68	**8.05**
	SWTA	**40.24**	7.08
	HPCA	36.74	**8.05**
4%	None	44.01	9.34
	SWTA	**46.98**	8.95
	HPCA	43.69	**9.68**
5%	None	49,37	10.80
	SWTA	**51.36**	10.47
	HPCA	50.40	**10.92**
10%	None	**65.61**	17.44
	SWTA	65.09	17.53
	HPCA	65.49	**17.65**
25%	None	**78.71**	29.01
	SWTA	78.17	**29.52**
	HPCA	78.53	29.20
100%	None	**90.03**	**50.58**
	SWTA	88.00	49.93
	HPCA	88.54	50.06

H. Zhang, C. Raffel, Remixmatch: Semi-supervised learning with distribution alignment and augmentation anchoring, arXiv preprint arXiv:1911.09785 (2019).

[48] P. Sellars, A. I. Aviles-Rivero, C.-B. Schönlieb, Laplacenet: A hybrid energy-neural model for deep semi-supervised classification, arXiv preprint arXiv:2106.04527 (2021).

[49] B. Gao, L. Pavel, On the properties of the softmax function with application in game theory and reinforcement learning, arXiv preprint arXiv:1704.00805 (2017).

[50] J. Karhunen, J. Joutsensalo, Generalizations of principal component analysis, optimization problems, and neural networks, Neural Networks 8 (4) (1995) 549–562.

[51] S. Becker, M. Plumbley, Unsupervised neural network learning procedures for feature extraction and classification, Applied Intelligence 6 (3) (1996) 185–203.

[52] A. Krizhevsky, I. Sutskever, G. E. Hinton, Imagenet classification with deep convolutional neural networks, Advances in neural information processing systems 25 (2012) 1097–1105.

[53] S. Ioffe, C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, arXiv preprint arXiv:1502.03167 (2015).

[54] D. P. Kingma, M. Welling, Auto-encoding variational bayes, arXiv preprint arXiv:1312.6114 (2013).

[55] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, A. Lerchner, beta-vae: Learning basic visual concepts with a constrained variational framework (2016).

[56] J. Wan, D. Wang, S. C. H. Hoi, P. Wu, J. Zhu, Y. Zhang, J. Li, Deep learning for content-based image retrieval: A comprehensive study, in: Proceedings of the 22nd ACM international conference on Multimedia, 2014, pp. 157–166.

[57] C. Bai, L. Huang, X. Pan, J. Zheng, S. Chen, Optimization of deep convolutional neural network for large scale image retrieval, Neurocomputing 303 (2018) 60–67.

[58] G. Amato, F. Falchi, C. Gennaro, F. Rabitti, Yfcc100m-hnfc6: A large-scale deep features benchmark for similarity search, in: L. Amsaleg, M. E. Houle, E. Schubert (Eds.), Similarity Search and Applications, Springer International Publishing, Cham, 2016, pp. 196–209.

[59] T. G. Kolda, R. M. Lewis, V. Torczon, Optimization by direct search: New perspectives on some classical and modern methods, SIAM review 45 (3) (2003) 385–482.

[60] G. Lagani, Hebbian learning algorithms for training convolutional neural networks, Master's thesis, School of Engineering, University of Pisa, Italy (2019).
URL https://etd.adm.unipi.it/theses/available/etd-03292019-2208

[61] M. Assran, M. Rabbat, On the convergence of nesterov's accelerated gradient method in stochastic settings, arXiv preprint arXiv:2002.12414 (2020).

[62] Y. Bengio, P. Simard, P. Frasconi, Learning long-term dependencies with gradient descent is difficult, IEEE transactions on neural networks 5 (2) (1994) 157–166.

[63] R. Pascanu, T. Mikolov, Y. Bengio, On the difficulty of training recurrent neural networks, in: International conference on machine learning, PMLR, 2013, pp. 1310–1318.

[64] X. Glorot, Y. Bengio, Understanding the difficulty of train-

ing deep feedforward neural networks, in: Proceedings of the thirteenth international conference on artificial intelligence and statistics, JMLR Workshop and Conference Proceedings, 2010, pp. 249–256.

[65] K. He, X. Zhang, S. Ren, J. Sun, Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, in: Proceedings of the IEEE international conference on computer vision, 2015, pp. 1026–1034.

[66] A. Iscen, G. Tolias, Y. Avrithis, O. Chum, Label propagation for deep semi-supervised learning, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019, pp. 5070–5079.