Neural Bag-of-Features Learning

Nikolaos Passalis, Anastasios Tefas

Department of Informatics, Aristotle University of Thessaloniki
Thessaloniki 54124, Greece Tel,Fax: +30-2310996304

Abstract

In this paper, a neural learning architecture for the well-known Bag-of-Features (BoF) model, called Neural Bag-of-Features, is proposed. The Neural BoF model is formulated in two neural layers: a Radial Basis Function (RBF) layer and an accumulation layer. The ability of the Neural BoF model to improve the classification performance is demonstrated using four datasets, including a large-scale dataset, and five different feature types. The gains are two-fold: the classification accuracy increases and, at the same time, smaller networks can be used, reducing the required training and testing time. Furthermore, the Neural BoF natively supports training and classifying from feature streams. This allows the proposed method to efficiently scale to large datasets. The streaming process can be also used to introduce noise and reduce the over-fitting of the network. Finally, the Neural BoF provides a framework that can model and extend the dictionary learning methodology.

Keywords: Bag-of-Features, RBF Neural Networks, Dictionary Learning

1. Introduction

Many machine learning problems involve dealing with objects that are represented as a collection of *feature vectors*. For example, the typical pipeline of an image recognition system is composed of the following steps:

1. First, multiple feature vectors, such as SIFT descriptors [1], HOG descriptors [2], or even raw pixel patches [3], are extracted from an image,
2. then, these feature vectors are used to create a constant length vector representation of the image, and
3. finally, this vector is fed to a classifier that recognizes the content of the image.

Usually, a large number of feature vectors are extracted from each image. Furthermore, the number of the extracted features vectors might vary, especially when interest point detectors are used [1], [4]. Therefore, it is critical to "aggregate" the extracted vectors into a meaningful constant length representation in order to efficiently measure the similarity of two images.

This problem can be formally defined as follows: Let $\mathcal{X} = \{x_i\}_{i=1}^{N}$ be a set of N objects. Each object x_i consists of N_i feature vectors: $\mathbf{x}_{ij} \in \mathbb{R}^D$ $(j = 1...N_i)$, where D is the dimensionality of the extracted features. Aggregate the

Email addresses: passalis@csd.auth.gr (Nikolaos Passalis), tefas@aiia.csd.auth.gr (Anastasios Tefas)

features of the i-th object into a compressed representation $\mathbf{s}_i \in \mathbb{R}^L$, where L is the dimensionality of the extracted representation. This representation should retain as much as possible information about the corresponding object, i.e., its feature vectors.

The most common and well-studied approach to create a constant-length representation of such objects is the *Bag-of-Features* (BoF) model [5], also known as Bag-of-Visual-Words when applied in the context of images. Although originally it was applied to problems related to image classification [6], it can be also used to create representations of other types of objects, including but not limited to video [7], music [8], and time-series data [9].

The BoF model, inspired by the text-oriented Bag-of-Words models [10], treats each object as a "document" and each feature vector as a "word". That way, each object/document is represented as a histogram over a predefined set of features/words, known as *codebook* or *dictionary*. More formally, the BoF model involves the following:

- *feature extraction*, in which multiple feature vectors are extracted from each object. These vectors define the *feature space*, where each object is represented as a set of features.

- *dictionary learning*, in which a set of representative features, known as *codewords* or simply *words*, are learned.

- *object encoding*, in which every feature vector of an object is represented using the codebook and a histogram is created. These histograms define the *histogram space*, where each object is represented as a constant-length vector.

Unsupervised clustering techniques, such as the k-means algorithm, are usually utilized for the dictionary learning step [5, 6]. The feature vectors are clustered, the centroids of the clusters are used to form the codebook and then each feature vector is represented by its nearest codeword. Note that the codewords (centroids) are chosen in such way to minimize the *reconstruction loss* of the feature vectors that belong to the corresponding cluster.

Although the previous approach has been successfully applied to a wide range of problems, it was quickly established that minimizing the reconstruction loss is not optimal with respect to the final classification task [11, 12, 13, 14]. This can be better understood by the following example. Let A be an image recognition problem where two object, cars and bicycles, are to be classified. Also, let B be a second image recognition problem where dogs and cats should be recognized. If the same training data are used (suppose that both animals and vehicles appear in the images), then an unsupervised dictionary learning method would generate the same dictionary for both problems. However, two different dictionaries that are specialized for the features that discriminate the cars from the bicycles and the dogs from the cats would be more appropriate for these two problems than a generic dictionary that simply compresses the feature vectors.

Indeed, supervised dictionary learning, which optimizes the dictionary for a specific classification problem, has been shown to perform better than the corresponding unsupervised methods (e.g., [12, 13, 15, 16]). Most of these supervised techniques either learn both the dictionary and the final classifier simultaneously or use a discrimination criterion in the histogram space and fine-tune the dictionary in order to increase the discriminative ability of the resulting histograms.

In this paper, the BoF model is generalized and formulated as a neural network, that is composed of a *Radial Basis Function* (RBF) layer [17], followed by a *recurrent* accumulation layer. Each RBF neuron acts like a codeword and the RBF layer is equipped with input weights that allow each individual RBF neuron to adjust the shape of its Gaussian function to better fit the input distribution. This also allows to indirectly alter the length of each codeword. The proposed neural layer, called *Neural Bag-of-Features* (Neural BoF), receives the feature vectors of an object and gradually compiles its representation at the accumulation layer. In order to classify an object the output of the Neural BoF layer must be connected to a classifier, such as a Multilayer Perceptron (MLP) [18]. Any shallow or deep classifier can be connected to the output of the proposed layer. This unified architecture can be trained using the standard backpropagation technique [18]. A number of issues that arose during the training, such as problems of exploding gradients or instabilities during the training process, were successfully resolved and they are discussed through this paper. Also, if trainable feature extractors are used, such as Convolutional Neural Networks (CNNs) [19, 20], the gradients can also backpropagate, through the Neural BoF, to them. That way, the resulting deep architecture can be further fined-tuned.

Most of the related supervised dictionary learning methods are hindered by their high computational complexity and, as a result, cannot scale to large datasets. To partially overcome this problem, an incremental technique for learning from feature streams, that can both decrease the training time and reduce the problem of over-fitting, is proposed. The proposed incremental learning technique works at feature level allowing to learn with a subset of the feature vectors of an object. This is in contrast with "regular" incremental learning techniques that require a fixed representation, i.e., all the extracted feature vectors, for each new training sample. A similar incremental approach, which can reduce the classification time, is also proposed for the classification: the feature vectors are continuously fed to the network and the streaming process stops when there is enough confidence for the output of the network. This process is somewhat similar to the way that humans recognize a visual object: we continuously examine the object until we are confident enough. The longer this process takes, the more details (features) are examined.

The proposed method is also capable of providing visual attention information [21]. The Neural BoF can "justify" its decision by highlighting the parts of the input image that are associated with each class/label. This provides an intrinsic way to perform object localization, image segmentation and/or further fine-tune the classification procedure.

The contributions of this paper are briefly summarized below. First, a neural network, inspired from the BoF model, is proposed as a generic neural layer that can be combined with various feature extractors and classifiers to

form very powerful representation and recognition machines. This allows to learn task-specific BoF representations using back-propagation. The model also supports the discriminative weighting of the input of each RBF neuron allowing for fine-grained segmentation of the feature space. The proposed neural model is evaluated using four datasets, including one large-scale dataset, and five different feature types. Second, an incremental method for optimizing the proposed model is introduced. Third, a fast incremental technique for classification is also proposed and its performance is evaluated. Finally, a method for tracking the visual attention of the proposed neural architecture is introduced and evaluated.

The rest of the paper is organized as follows. The related work is discussed in Section 2 and the Neural BoF model, the incremental algorithms and the method for tracking the attention of the proposed model are presented in Section 3. The experimental evaluation of the proposed methods is presented in Section 4. Finally, conclusions are drawn and possible extensions are discussed in Section 5.

2. Related Work

The proposed method uses concepts from both the RBF neural networks [17, 22, 23], and the recurrent neural networks [24, 25, 26]. It is also closely related to the dictionary learning approaches for the BoF representation since learning the Neural BoF parameters is similar to dictionary/codebook learning. Note that the term dictionary learning is also used for representation methods other than the BoF method, such as sparse coding [27, 28, 29].

The related supervised dictionary learning methods can be classified into two categories according to the used optimization criterion. The first category uses a feature-space objective (feature-space methods), while the second category uses a histogram-space objective (histogram-space methods).

Feature-space methods. These works assume that each feature carries the same label as the image from which it was extracted. This assumption is not always true and can, consequently, decrease the discriminative ability of the learned codebooks. In [12], the proposed optimization scheme tries to increase the mutual information between each codeword and the corresponding features labels, while in [30], and [31], to minimize the entropy of the cluster defined by each codeword.

Histogram-space methods. The histogram-space methods can be further divided into two subcategories.

The first subcategory ties the classifier and the codebook learning and rely on the classifier's decisions to optimize the codebook. In [13], multiple maximum margin hyperplanes are learned and at the same time the codebooks are adjusted to maximize the corresponding margins. However, this method requires a quadratic number of codebooks with respect to the number of the training labels. Later works, such as [15], and [16], utilized multi-class Support Vector Machine (SVM) formulations to address this problem. A simple supervised learning and codebook optimization for the Bag-of-Words model, incorporating both a traditional MLP layer and a codebook layer, is proposed in [32]. The resulting network can be trained either by using backpropagation or by using

a cluster reassignment technique. Also, in [33], the optimization aims to minimize the logistic regression loss. Another proposed approach is to learn multiple dictionaries with complementary discriminative information [14]. The dictionaries are learned by adjusting the weights used during the clustering process according to the predictions of a histogram-space classifier.

The second subcategory utilizes a discriminative criterion in the histogram space instead of relying on a classifier for the codebook optimization. These approaches focused on learning class-specific dictionaries [11], [34], or adjusting the dictionary in order to increase a specific objective such as the mutual information [35], or the ratio of intra/inter class variation [9], [36], [37]. In [38], a two-fold objective is used for the optimization: each feature vector should be close to its codeword, while the extracted histograms should separate the objects that belong to different classes.

The proposed Neural BoF model provides a new way of understanding the dictionary learning for the BoF model. Many of the aforementioned methods can be considered as special cases of the proposed Neural BoF. For example, in [33], logistic regression is used instead of an MLP, while in [37], the output layer is replaced by a loss function that measures the discrimination ability of the learned representation. In [32], a simplified neural formulation is also used, but lacks several features of the proposed method, such as the discriminant weighting of the feature vectors, which allows for variable-length codewords and finer segmentation of the input space.

The Convolutional Neural Networks (CNNs) are also well-known models that can be used for image classification [19, 20]. Similar to the CNNs, the proposed Neural BoF model ties the feature aggregation step with the classification layer. However, the Neural BoF model uses a modified vector quantization scheme for segmenting the feature space and extracting a compact representation instead of a simple max/mean polling layer that is utilized by the CNNs. Also, in contrast to the CNN model, where only convolutional features can be used, the proposed method can be combined with any kind of feature extractors, either handcrafted, e.g., SIFT, HOG, etc., or trainable, e.g., convolutional feature extractors.

To the best of our knowledge, this work presents the first dictionary learning method for the BoF representation that a) supports discriminant weighting of the feature-space individually for each codeword and b) natively supports learning and classifying from feature streams.

3. Proposed Method

In this section the Neural BoF model and the feature streaming algorithms that are used to train and test the model are presented. First, the standard BoF model is briefly described. Then, the Neural BoF model is introduced and the utilized classification layer is described. Also, a learning algorithm for the proposed architecture is derived and discussed. Then, the incremental algorithms for the learning and the testing of the network are described. Finally, a method that provides attention-based information using the Neural BoF model is introduced.

3.1. BoF Model

Let $\mathcal{X} = \{x_i\}_{i=1}^{N}$ be a set of N objects to be represented using the BoF model. As stated before, each object x_i consists of N_i feature vectors: $\mathbf{x}_{ij} \in \mathbb{R}^D$ $(j = 1...N_i)$, where D is the dimensionality of the extracted features. For example, in image classification each x_i would be an image and each $\mathbf{x}_{ij}$ a vector extracted from it, such as SIFT or HOG features. Then, a fixed-length histogram is compiled for each object by quantizing its feature vectors into a predefined number of histogram bins/codewords. In hard assignment each feature vector is quantized to its nearest codeword, while in soft assignment every feature contributes, by a different amount, to each histogram bin/codeword.

To learn a codebook, the set of all feature vectors, $\mathcal{S} = \{\mathbf{x}_{ij} | i = 1...N, j = 1...N_i\}$, is clustered into N_K clusters and the corresponding centroids (codewords) $\mathbf{v}_k \in \mathbb{R}^D (k = 1...N_K)$ are used to form the codebook $\mathbf{V} \in \mathbb{R}^{D \times N_K}$, where each column of $\mathbf{V}$ is a centroid. These centroids are used to quantize the feature vectors. It is common to cluster only a subset of $\mathcal{S}$, since this can reduce the training time with little effect on the learned representation. This process is done only once and the learned codebook can be used to represent any object.

To encode the i-th object, the similarity between each feature vector $\mathbf{x}_{ij}$ and each codeword $\mathbf{v}_k$ is computed as:

$$[\mathbf{d}_{ij}]_k = exp(\frac{-||\mathbf{v}_k - \mathbf{x}_{ij}||_2}{g}) \in \mathbb{R} \tag{1}$$

where the notation $[\mathbf{d}_{ij}]_k$ is used to denote the k-th element of the vector $\mathbf{d}_{ij}$. The parameter g controls the quantization process: for harder assignment $g << 1$ is used, while for softer assignment larger values are used. It is also common to use hard and non-continuous assignments [5, 6]:

$$[\mathbf{d}_{ij}]_k = \begin{cases} 1 & \text{if } k = \arg\min_{k'}(||\mathbf{v}_{k'} - \mathbf{x}_{ij}||_2) \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

Note that the first definition in (1) converges to the second in (2) as $g \to 0$. Also, the equation (1) is usually preferred over the equation (2) for codebook learning algorithms (including, but not limited to [9], [13], [31], [32], [37]), since it is continuous with respect to the codewords $\mathbf{v}_k$, which allows simple algorithms, e.g., gradient descent, to be used for the optimization. Then, the l^1 normalized membership vector of each feature vector $\mathbf{x}_{ij}$ is obtained:

$$\mathbf{u}_{ij} = \frac{\mathbf{d}_{ij}}{||\mathbf{d}_{ij}||_1} \in \mathbb{R}^{N_K} \tag{3}$$

This vector describes the similarity of the feature vector $\mathbf{x}_{ij}$ to each codeword. Finally, the histogram $\mathbf{s}_i$ is extracted for every object x_i:

$$\mathbf{s}_i = \frac{1}{N_i} \sum_{j=1}^{N_i} \mathbf{u}_{ij} \in \mathbb{R}^{N_K} \tag{4}$$

The histogram $\mathbf{s}_i$ has unit l^1 norm, since $||\mathbf{u}_{ij}||_1 = 1$ for every j. These histograms describe each object and they

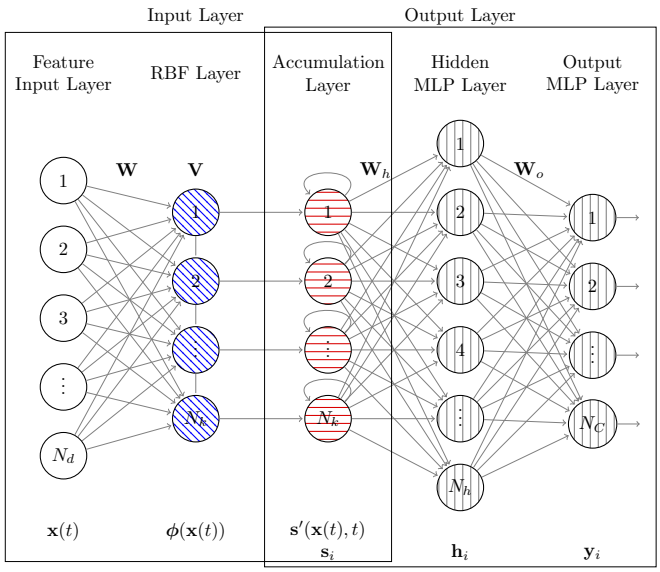


Figure 1: The proposed neural architecture. The nodes marked with the diagonal lines are RBF neurons, the nodes marked with the horizontal lines are recurrent accumulation neurons and the nodes marked with the vertical lines are sigmoid units. The RBF neurons are tied together due to the normalization of their output.

can be used for the subsequent classification. Finally, note that the training and the encoding process are fully unsupervised and no labeled data are required.

3.2. Neural BoF Model

Motivated by the BoF model, a neural generalization of this model is proposed in this subsection. The left part of Figure 1 depicts the proposed neural architecture. The network is composed of two layers: a RBF layer that measures the similarity of the input features to the RBF centers and a recurrent accumulation layer that builds a histogram of the input features. The proposed architecture can be thought as a unified input layer that feeds the extracted representation to a subsequent classifier. Note the "streaming" nature of the Neural BoF: it sequentially receives the feature vectors of an object and gradually compiles the object's histogram which is fed to the connected classifier. The classification can be either done only once after all the feature vectors are presented to the network, as in the standard BoF model, or it can be done continuously as the feature vectors arrive (the connected classifier continuously update its decision). The latter option allows to stop the streaming process early, if the classifier is confident enough for its output, reducing the time needed for the classification process (less feature vectors are extracted and fed to the network).

The output of the k-th RBF neuron is defined as:

$$[\boldsymbol{\phi}(\mathbf{x})]_k = \exp(-||(\mathbf{x} - \mathbf{v}_k) \odot \mathbf{w}_k||_2) \tag{5}$$

where $\mathbf{x}$ is a feature vector and $\odot$ is the element-wise multiplication operator. The RBF neurons behave somewhat like the codewords in the BoF model, i.e., they are used to measure the similarity of the input vectors to a set

of predefined vectors. Each RBF neuron is also equipped with a weight vector $\mathbf{w}_k \in \mathbb{R}^D$ that adjusts the width of its Gaussian function per each dimension. That allows for better modeling of the input distribution, since the distribution modeled by each RBF can be independently adjusted. Also, note that by zeroing some of the input weights the length of each codeword can be reduced. These were not possible with the previous similarity definition in (1), which only depends on the global scaling factor g. However, the BoF model can be re-obtained from the Neural BoF model by setting all the weights $[\mathbf{W}]_{ij} = \frac{1}{g}$, where the matrix $\mathbf{W} \in \mathbb{R}^{D \times N_K}$ is defined by stacking the vectors $\mathbf{w}_k$.

To ensure that the output of each RBF neuron is bounded, a normalized RBF architecture is used. This normalization is equivalent to the l^1 scaling that is utilized in (3). Thus, the output of the RBF neurons is re-defined as:

$$[\boldsymbol{\phi}(\mathbf{x})]_k = \frac{\exp(-||(\mathbf{x} - \mathbf{v}_k) \odot \mathbf{w}_k||_2)}{\sum_{m=1}^{N_K} \exp(-||(\mathbf{x} - \mathbf{v}_m) \odot \mathbf{w}_m||_2)} \tag{6}$$

The output of the RBF neurons is accumulated in the next layer. To this end, the concept of time is introduced and a recurrent self loop is used in the accumulation layer:

$$[\mathbf{s}'(\mathbf{x}(t), t)]_k = \frac{1}{t}[\boldsymbol{\phi}(\mathbf{x}(t))]_k + \frac{t-1}{t}[\mathbf{s}'(\mathbf{x}(t-1), t-1)]_k \tag{7}$$

where $\mathbf{x}(t)$ is the feature vector fed at time t in the neural network and $[\mathbf{s}'(\mathbf{x}(t), t)]_k$ the output of the accumulation layer at time t. Note that the output of the accumulation layer remains normalized, i.e., it has unit l^1 norm. Before feeding the features of a new object, the time is reset ($t = 0$) and the output of this layer is zeroed:

$$[\mathbf{s}'(\mathbf{x}, 0)]_k = 0 \tag{8}$$

Over the time, the histogram of the input object is gradually compiled at the output of the Neural BoF layer. It can be easily seen that the previous definition is equivalent to the following (when the first t features of the i-th object are fed to the layer):

$$\mathbf{s}_i = \frac{1}{t}\sum_{j=1}^{t} \boldsymbol{\phi}(\mathbf{x}_{ij}) \tag{9}$$

where $\boldsymbol{\phi}(\mathbf{x}, t) = ([\boldsymbol{\phi}(\mathbf{x}, t)]_1, ..., [\boldsymbol{\phi}(\mathbf{x}, t)]_{N_K})^T \in \mathbb{R}^{N_K}$ is the output vector of the RBF layer. The proof is provided in the Appendix A.

This definition a) simplifies the derivative calculations and b) removes the sequential dependencies on the time variable t (i.e., the features are fed one at a time), which allows for more efficient implementations, since the output/gradients of the Neural BoF layer can be computed in parallel for a batch of input features.

Hard quantization, which is described by equation (2), is also supported by the proposed formulation. In this case, only the RBF neuron with the maximum response is activated for each feature and the rest of the architecture remains unchanged. However, this modification introduces non-continuities that make the optimization of the input

layer intractable and, therefore, it is not used in this work.

3.3. Classification Layer

The previous layer receives the features of an object and compiles its histogram. Then, this histogram must be fed to a classifier that decides the class of the object. In this work a multilayer perceptron (MLP) with one hidden layer is used for this purpose. Figure 1 also depicts the used MLP (Output Layer).

Only the single-label classification problem is considered: each training object x_i is annotated by a label l_i and there are N_C different labels. This is without loss of generality since the proposed method can be readily applied to multi-label classification problems as well.

Let $\mathbf{W}_H \in \mathbb{R}^{N_H \times N_K}$ be the hidden layer weights and $\mathbf{W}_O \in \mathbb{R}^{N_C \times N_H}$ be the output layer weights, where N_H is the number of hidden neurons. Then, the hidden layer activations for the input histogram $\mathbf{s}_i$ of the i-th object are computed as:

$$\mathbf{h}_i = \phi^{(s)}(\mathbf{W}_H \mathbf{s}_i + \mathbf{b}_H) \in \mathbb{R}^{N_H} \tag{10}$$

where $\phi^{(s)}(x)$ is the sigmoid activation function:

$$\phi^{(s)}(x) = \frac{1}{1 + e^{-x}} \tag{11}$$

which is applied element-wise and $\mathbf{b}_H \in \mathbb{R}^{N_H}$ is the hidden layer bias vector.

Similarly, the output of the MLP is calculated as:

$$\mathbf{y}_i = \phi^{(s)}(\mathbf{W}_O \mathbf{h}_i + \mathbf{b}_O) \in \mathbb{R}^{N_C} \tag{12}$$

where each output neuron corresponds to a label (the one-vs-all strategy is used) and $\mathbf{b}_O \in \mathbb{R}^{N_C}$ is the output layer bias vector.

Furthermore, a loss function must be defined in order to train the network and backpropagate the gradients to the input layer. For this task, the cross entropy loss is chosen:

$$L = -\sum_{i=1}^{N} \sum_{j=1}^{N_C} [\mathbf{t}_i]_j \log([\mathbf{y}_i]_j) + (1 - [\mathbf{t}_i]_j) \log(1 - [\mathbf{y}_i]_j) \tag{13}$$

where $\mathbf{t}_i \in \mathbb{R}^{N_C}$ is the target output vector, which depends on the label (l_i) of the input object and it is defined as:

$$[\mathbf{t}_i]_j = \begin{cases} 1 & j == l_i \\ 0 & \text{othewise} \end{cases} \tag{14}$$

The output of the network ($[\mathbf{y_i}]_j$) is bounded to 0-1 since the sigmoid activation function is used. Also, note that the cross entropy loss is computed individually for each output neuron, by considering the probabilities $[\mathbf{y_i}]_j$ and

$1 - [\mathbf{y_i}]_j$, instead of tying (and normalizing) the output of all neurons of the network (which is usually done in single-label classification problems). This technique improves slightly the classification results.

After the Neural BoF layer has been trained using an MLP, the output layer can be replaced by another classifier, such as an SVM, without re-training the Neural BoF layer. Experiments done in Section 4 show that the supervised training of the Neural BoF layer (using an MLP) improves the classification accuracy for other types of classifier too.

3.4. Neural BoF Learning Algorithm

The complete network architecture is shown in Figure 1. If the centers of the RBF neurons are chosen using the k-means algorithm and the input weights are the same and fixed, then the input layer is equivalent to the standard BoF model. However, in this work supervised training is employed to optimize the input layer. More specifically, the backpropagation algorithm [18], is used to train both the output and the input layers. In order to learn the parameters of the network, the following derivatives must be calculated: $\frac{\partial L}{\partial \mathbf{W}_O}$, $\frac{\partial L}{\partial \mathbf{W}_H}$, $\frac{\partial L}{\partial \mathbf{b}_O}$, $\frac{\partial L}{\partial \mathbf{b}_H}$, $\frac{\partial L}{\partial \mathbf{V}}$, $\frac{\partial L}{\partial \mathbf{W}}$. The derivation of these derivatives is provided in the Appendix B.

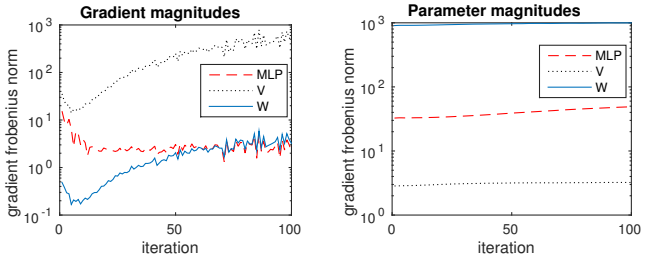
In the first attempt to optimize the network, the simple stochastic gradient descent (SGD) algorithm was used to update the parameters of the network:

$$\Delta(\mathbf{W}_O, \mathbf{W}_H, \mathbf{b}_O, \mathbf{b}_H, \mathbf{V}, \mathbf{W}) = -(\eta_{MLP}\frac{\partial L}{\partial \mathbf{W}_O}, \eta_{MLP}\frac{\partial L}{\partial \mathbf{W}_H}, \eta_{MLP}\frac{\partial L}{\partial \mathbf{b}_O}, \eta_{MLP}\frac{\partial L}{\partial \mathbf{b}_H}, \eta_V\frac{\partial L}{\partial \mathbf{V}}, \eta_W\frac{\partial L}{\partial \mathbf{W}})$$

where η_{MLP} is the learning rate for the output layer, η_V the learning rate for the centers of the RBF neurons and η_W the learning rate for the input layer weights. However, similarly to many deep learning architectures, it was not trivial to select the proper learning rates, since the learning process was very fragile. For larger learning rates the network oscillated and the convergence process was unstable, while for smaller learning rates the convergence process was smoother, but significantly slower.

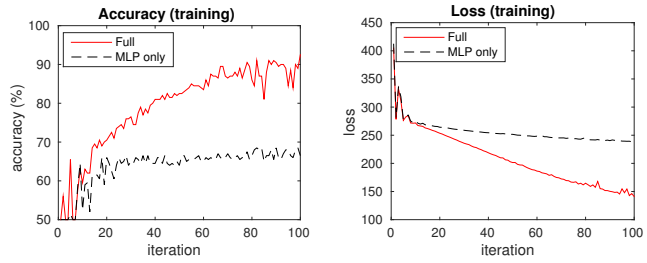
To better understand the dynamics behind this phenomenon is useful to examine the magnitude of the derivatives during the learning process, as well as the magnitude of the network's parameters. The Frobenius norm of the derivatives and the network's parameters are shown in Figure 2. The derivative $\frac{\partial L}{\partial \mathbf{V}}$ abruptly increases after the first 20 iterations, which, in turn, invalidates the previous choice of the learning rate for the corresponding parameters ($\mathbf{V}$). This behavior is aggravated by the magnitude of the input layer weights $\mathbf{W}$, since the derivative $\frac{\partial L}{\partial \mathbf{V}}$ depends on the value of $\mathbf{W}$. Therefore, the used learning rate should be gradually reduced to adapt to the changes in the magnitude of the derivatives and avoid instabilities during the training process.

To this end, a recently proposed method for stochastic optimization, the Adam (Adaptive Moment Estimation) algorithm [39], is utilized. The Adam algorithm is easy to implement and proved to be very stable during the conducted experiments. In order to match the great difference in the magnitude of the derivatives, three different learning rates are used (η_{MLP}, η_V and η_W). The derivative $\frac{\partial L}{\partial \mathbf{V}}$ is about two orders of magnitude greater than



(a) (b)

Figure 2: Gradient (a) and parameters (b) magnitude (Frobenius norm) during optimization. A binary problem (living room vs. bedroom) of the 15-scene dataset and 16 codewords were used to illustrate the process.



(a) (b)

Figure 3: Training accuracy (a) and loss (b) during the optimization of the output layer (MLP) and the whole network (Full)

$\frac{\partial L}{\partial \mathbf{W}}$. Therefore, the corresponding learning rates should be chosen to maintain the same rate of changes in the input layer. After some experimentation the following learning rates were selected: $\eta_{MLP} = 0.01$, $\eta_V = 0.001$ and $\eta_W = 0.2$. These learning rates ensure that the learning process would be both smooth and fast enough.

A toy optimization example using two classes of the most difficult binary problem of the 15-scene image classification dataset (bedroom vs living room) is also provided. Figure 3 plots the accuracy and the classification loss during the optimization of the output layer (MLP with 100 hidden neurons), while the 16-word codebook is learned using the k-means algorithm. In the same Figure the accuracy and the loss during the optimization of the whole network (both the Neural BoF and MLP layers) are also plotted. The same learning rate ($\eta_{MLP} = 0.01$) was used for both experiments. It is evident that the optimization of the input layer leads to a neural network with greater capacity, which also learns faster (in terms of iterations).

3.5. Incremental Learning and Classification with Feature Streaming

During each training epoch the feature vectors must be re-fed to the network, since the changes in the Neural BoF alters its output. Also, the gradients must be re-propagated to the input layer through every feature. This significantly reduces the ability of the proposed method to optimize objects with a large number of features. To overcome this limitation, an incremental stream-based learning algorithm is proposed. The rationale behind this incremental approach is that the features of each object should be gradually presented to the neural network in order to a) speedup the learning process and b) prevent over-fitting to the learned representation.

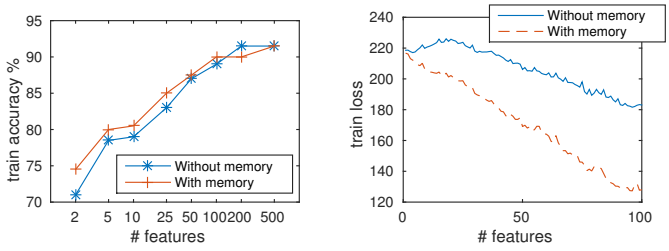


Figure 4: Effect of the number of sampled features on the training accuracy (a) and learning curve (training loss) when 10 sampled are used (b). Two different techniques were used: a) setting the RBF memory to zero before each iteration (without memory) and b) initializing the RBF memory to the previous histogram of each object (with memory). A binary problem (bedroom vs. living room) of the 15-scene dataset was used to illustrate the process.

Instead of feeding all the features for each object to the input layer, only the most recently extracted features are used. If the features are already extracted, a random sample of them is used. In both cases, after each iteration a different set of features is used for each object. The memory ($\mathbf{s}'(\mathbf{x}, 0)$) of the accumulation layer can be either set to zero or initialized to the histogram that was compiled in the previous epoch for each object. That way, the sampled histogram will gradually converge to the histogram of the object (if all the features were used).

Care should be given to ensure that the sampled feature distribution is representative enough. Experimentally it was established that the number of the required feature samples increases when more RBF neurons are used. Sampling about N_k features per object seems to provide an adequate estimation of the final histogram for the most cases. If an adequate number of features cannot be used, e.g., due to memory or time constraints, then initializing the memory of the accumulation layer to the previously extracted histogram increases both the convergence speed and the stability of the training process. However, if over-fitting issues arise, then resetting the memory to zero before each epoch, can reduce the over-fitting phenomena.

This feature sub-sampling technique introduces noise to the output of the Neural BoF. However, it significantly speedups the training process allowing more iterations to be done in less time and, at the same time, makes the network less prone to over-fitting. It can be also considered as a modified dropout technique [40], that works at feature level.

Figure 4 plots the training set accuracy after 50 iterations, using 64 RBF neurons for different numbers of sampled features. For less than 10 features per object the accuracy is severely harmed, while the increase is limited when 100 or more features are sampled. Also, when memory is used (by initializing the accumulation neurons to the previously extracted histogram) the learning process is faster, especially for small numbers of sampled features.

The proposed incremental learning algorithm is described in Algorithm 1. First, the weights of the network are initialized (lines 2-4). Several options exist for the initialization of the RBF neurons [41]. One of the most widely used techniques is to cluster a subset of the input features and use the resulting centroids as the centers of the RBF neurons. Usually, the k-means algorithm is used for the clustering. In the proposed method, 50,000

features are randomly sampled, the clustering process is repeated 5 times (using different initializations for the k-means algorithm) and the best clustering configuration, in terms of reconstruction error, is chosen. Note that this process is essentially the same with the process used to learn an unsupervised codebook for the BoF representation (Subsection 3.1).

Then, the output layer of the network is pretrained (line 5) for a small number of iterations (N_{piters}). This pretraining provides a better starting point for the optimization of the whole network and, as a result, reduces the subsequent training time.

Finally, the whole network is optimized (lines 6-12) using mini-batches of size N_{batch}. For each object either the most recently extracted features or a randomly selected subset of the (already extracted) features are used in each iteration of the algorithm. Also, the default parameters are used for the Adam algorithm ($\beta_1 = 0.9, \beta_2 = 0.999$ and $\epsilon = 10^{-8}$) [39]. A reference implementation of the proposed algorithm using the Theano library [42], that also allows to accelerate the learning process using the GPU instead of the CPU, is provided in the supplementary material.

Algorithm 1 Incremental learning algorithm

Input: A set $\mathcal{X} = \{x_1, ..., x_N\}$ of N training objects and their class labels $\mathcal{L} = \{d_1, ..., d_N\}$
Hyper-parameters: g, N_K, N_{iters}, N_{piters}, N_{batch}, $N_{samples}$, η_{MLP}, η_V, η_W
Output: The parameters of the optimized neural network

1: **procedure** OPTIMIZENETWORK
2: Randomly initialize the MLP weights ($\mathbf{W}_O, \mathbf{W}_H, \mathbf{b}_O, \mathbf{b}_H$) to $[-0.5, 0.5]$
3: Initialize $\mathbf{V}$ by running k-means on a subsample of the training features
4: Initialize $\mathbf{W}$ to $1/g$
5: Train only the output layer (MLP) for N_{piters} epochs using the ADAM algorithm (batch size: N_{batch}, learning rate: η_{MLP})
6: **for** $i \leftarrow 1; i \leq N_{iters}; i++$ **do**
7: **for** every batch $B \in \mathcal{X}$ **do**
8: Stream / randomly sample $N_{samples}$ features for each object in B
9: **if** memory is used **then** initialize the memory of the RBF layer using the previous histogram of each object
10: Feedforward the features for each object
11: Backpropagate the network and compute the corresponding gradients
12: Apply the Adam algorithm at each layer using the corresponding learning rates (η_{MLP}, η_V and η_W)

Likewise, for the classification phase an incremental algorithm is also proposed. The features are continuously fed to the network in mini-batches (N_{fbatch}), e.g., of 50 features, until the confidence of the output layer exceeds a certain threshold. For the used MLP classifier, the threshold can be set by measuring the mean activation per class/neuron using the training set. The feature streaming stops, when a) a neuron exceeds its assigned threshold, b) only one such neuron exists and c) at least $N_{fsamples}$ features have been streamed. The second and the third conditions ensure that the network has converged before stopping the feature streaming process. The value of $N_{fsamples}$ is set to 100 features. The proposed algorithm is shown in Algorithm 2.

Note that this process is somewhat similar to the way that humans work to recognize an object: first we take a brief look at the object (i.e., sample an initial number of features and create its representation) and then we

Algorithm 2 Incremental classification algorithm

Input: A feature extractor, an object x, the trained Neural BoF network and the mean activations of the output neurons per class (calculated using the training set)
Parameters: $N_{fsamples}$, N_{fbatch}
Output: the label of x

procedure ClassifyObject
    Initialize RBF memory to zero
    done $\leftarrow$ *false*
    while *done* == *false* **do**
        Extract (or sample) a batch of N_{fbatch} feature vectors
        Feed the feature vectors to the input layer and feed-forward the network
        if the activation of exactly one output neuron exceeds its mean activation **and** at least $N_{samples}$ features vectors have been fed **then** *done* $\leftarrow$ *true*
return the most probable label according to the MLP responses

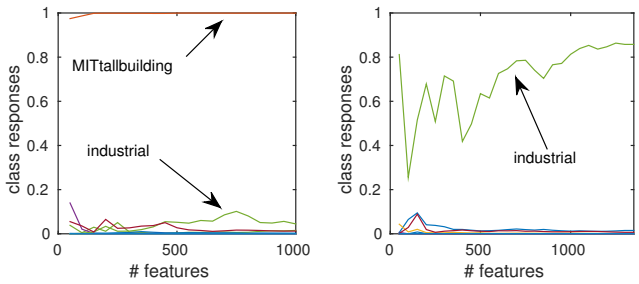


Figure 5: Incremental classification using feature streaming. The plots refer to two different images where each curve corresponds to the response of a different output neuron for the corresponding image.

examine it more carefully (i.e., request more features until the confidence has reached a certain threshold). The responses of the MLP during the feature streaming for two images of the 15-scene dataset are shown in Figure 5. For some images (usually when the classification task is relatively easy), the winning neuron stabilizes its output relatively soon. Therefore the decision can be taken without examining all the available features.

3.6. Neural BoF Attention Tracking

The proposed Neural BoF layer also allows to track the attention of the connected neural network on various regions of the input image associated with each class. The following method is used to provide visual attention information in the form of a binary mask that can be applied to the input image:

1. The contribution of each RBF neuron to each output neuron is calculated by multiplying the weights of the MLP layers: $\mathbf{W}_O\mathbf{W}_H$.
2. The features extracted from an image are fed to the network and the activation of each RBF neuron is calculated. Each feature vector is labeled by a number (1 to N_K) according to the RBF neuron that wins it (has the maximum response).
3. The patches that correspond to the features vectors that belong to the top-3 RBF neurons correlated with the correct class (as calculated by $\mathbf{W}_O\mathbf{W}_H$) are used to create a mask that increases the brightness of the

corresponding regions (the brightness is increased proportionally to the contribution of the winning RBF neuron). The regions of the mask that correspond to feature vectors that are won by the remaining RBF neurons remain black.

4. The mask is normalized to $0-1$, an exponential function $f(x) = exp(a * x) - 1$ is applied at each pixel ($a = 1.8$) and then a disk filter with radius of 20 pixels is applied to smooth the mask. Any values greater than 1 are clipped to 1.
5. The created mask is applied to the image.

The regions of the image where the mask has values close to 1 are strongly correlated with the activated output label/class. Conversely, the regions of the image where the mask is close to 0 either provide no information or they are negatively correlated with the activated class/label. The parameters of the proposed technique (number of activated RBF neurons, type and radius of the smoothing filter, a) can be tuned according to the specific application by visually examining the resulting images. Examples of attention-based annotation of images using two image datasets, the 15-scene dataset and the MNIST dataset, are provided in the following Section.

4. Experiments

In this section the proposed method is evaluated using four datasets and five different types of features. First, the used datasets and the feature extraction methods are described. Then, the results for each dataset are presented and discussed. Finally, the statistical significant analysis, the parameter selection procedure and the used parameters for the conducted experiments are presented.

4.1. Datasets

Four different datasets were used to evaluate the proposed method: the fifteen natural scene (15-scene) dataset [6], the MNIST database of handwritten digits (MNIST) [43], the polarity dataset v2.0 (RT-2k) [44], and the scene understanding (SUN397) dataset for large-scale scene recognition [45].

4.1.1. 15-scene Dataset

The 15-scene dataset [6], consists of images belonging to fifteen different natural scene categories, such as industrial, forest, city, bedroom and living room scenes. The dataset has a total of 4485 images, with an average size of 300 $\times$ 250, and the number of images in each category ranges from 200 to 400 images. Since there is no predefined train/test split, the standard evaluation protocol is used [6]:

- The training split is composed of 1500 randomly chosen images (100 from each category).
- The testing split is composed of the remaining 2985 images.

- The evaluation process is repeated 5 times and the mean and the standard deviation of the evaluation metric (precision) is reported.

Three different types of features were extracted from the images:

1. **SIFT features**: As proposed in [6], SIFT features [1], of 16 × 16 patches were extracted from a grid with spacing of 8 pixes. The dimensionality of each feature vector is 128.
2. **HOG+LBP features**: HOG [2], and LBP features [46], of 8 × 8 non-overlapping patches were also densely extracted from each image. The VLFeat library [47], was used to extract the features. The two feature vectors extracted from each patch are fused together. The dimensionality of the fused vector is 89.
3. **Convolutional Features**: The VGG-16 neural network trained on the Places365 dataset was used to extract the convolutional features [19]. Each image was resized to 224x224 pixels and 196 (14x14) feature vectors were extracted from the last convolutional layer (conv5_3). The extracted feature vectors were normalized to have unit l^2 norm. The dimensionality of each feature vector is 512.

4.1.2. MNIST Dataset

The MNIST database [43], is a well-known dataset that contains 60,000 training and 10,000 testing images of handwritten digits. There are 10 different classes, one for each digit (0 to 9), and the size of each image is 28 × 28. The method proposed in [3], is used to extract features from the images. The features are extracted by scanning the image (from left to right and from top to bottom) using a sliding window of $c \times c$ with horizontal and vertical step of 1 pixel. Then, each $c \times c$ patch is transformed to a vector that contains $c * c$ values. The proposed patch normalization techniques [3], were not applied as it was difficult to establish the right parameters for different patch sizes and the improvement in the classification accuracy was small. Finally, the mostly black patches, i.e., patches with l^2 norm under 10^{-5}, are discarded.

4.1.3. Polarity Dataset v2.0 (RT-2k)

The polarity dataset [44], is a sentiment dataset that contains 1000 positive and 1000 negative movie reviews. Due to the small number of reviews, 10-fold cross-validation (the splits are predefined) is used to evaluate the performance of a method. The mean number of the sentences of each review is 34.1 ± 15.5. To extract feature vectors from the movie reviews the RNTN model is used [48]. More specifically, a 25-value sentiment vector is extracted from every sentence of each review. A pre-trained RNTN model, which is included in the Stanford CoreNLP library [49], was used for the feature extraction. This technique was proposed in [50], and used to extract the sentiment from movie reviews with moderate success. The small number of features per document (one feature vector is extracted from each sentence) makes this problem especially challenging, since it is easy to overfit the training data.

Table 1: 15-scene: Classification results using the SIFT features

	Number of RBF neurons / codewords				
Type	8	16	32	64	128
BoF + MLP	48.23 ± 0.73	57.85 ± 2.11	67.01 ± 0.98	70.63 ± 0.71	71.56 ± 0.85
Neural BoF	63.21 ± 1.07	69.42 ± 2.19	73.51 ± 0.59	75.35 ± 0.75	76.09 ± 1.02
Neural BoF (fine-tuned)	$\mathbf{67.12 \pm 0.93}$	$\mathbf{73.09 \pm 0.59}$	$\mathbf{75.34 \pm 0.83}$	$\mathbf{76.61 \pm 0.82}$	$\mathbf{77.65 \pm 0.43}$
BoF + SVM	47.64 ± 0.71	58.56 ± 1.69	67.54 ± 0.53	71.03 ± 0.40	73.61 ± 0.32
Neural BoF + SVM	$\mathbf{67.28 \pm 1.14}$	$\mathbf{74.02 \pm 0.87}$	$\mathbf{77.41 \pm 0.93}$	$\mathbf{79.00 \pm 0.71}$	$\mathbf{80.25 \pm 0.53}$

4.1.4. SUN397 Dataset

The SUN397 dataset [45], is a large-scale scene recognition dataset that contains 397 different scene categories. The total number of images is 108,754, while there are at least 100 images per category (the exact number varies since the dataset is unbalanced). To evaluate the performance of the proposed method in a large-scale setting the dataset is randomly split in half and each split is used as the training/testing set respectively. The classification accuracy is reported. From each image 196 convolutional feature vectors were extracted using the same procedure as in the 15-scene dataset.

4.2. 15-scene Dataset Evaluation

The classification results on the 15-scene dataset using the SIFT feature vectors are shown in Table 1. The Neural BoF network consists of a Neural BoF layer and a 2-layer MLP, as described before (Figure 1). The (unsupervised) BoF model is also connected to a 2-layer MLP. The parameters used for the training of the models are summarized in Section 4.7. Also, after the training of the Neural BoF network, the MLP layer can be further fine-tuned using backpropagation. For both the training of the MLP of the BoF model and the fine-tuning of the Neural BoF network, 500 iterations were used.

The Neural BoF significantly outperforms the BoF + MLP, when the same number of RBF neurons (codewords) are used. The precision increase is larger when a small number of RBF neurons is used. For example, for 8 RBF neurons the precision increases by more than 18%, while for 128 RBF neurons the precision increase is 6%. Then, to confirm that the expressive power of the Neural BoF layer is responsible for the better classification accuracy, the MLP layer was replaced by an SVM with χ^2 kernel (C=10) [51, 52], after the Neural BoF layer has been trained. Other kernels were also evaluated, but the χ^2 kernel consistently lead to better classification accuracy. Again, a large increase in the precision is observed when the supervised Neural BoF layer is used. Also, it should be noted that even if the number of the RBF neurons are decreased by a factor of 4-8, the Neural BoF network still performs better than larger BoF networks. This allows to use smaller Neural BoF networks and thus to reduce the time needed for the classification. For example, a trained Neural BoF network with only 16 RBF neurons (codewords) performs better than the BoF scheme with 128 codewords (73.09% vs. 71.56% when an MLP is used and 74.02% vs. 73.61% when an SVM is used).

A popular technique, which is used to increase the accuracy of the BoF model, is the Spatial Pyramid Matching

Table 2: 15-scene: Classification results using the SIFT features and the SPM scheme

		Number of RBF neurons / codewords				
Type	**Level**	8	16	32	64	128
BoF + MLP	1	60.41 ± 1.60	67.55 ± 0.92	71.81 ± 0.66	73.43 ± 0.25	74.66 ± 1.13
Neural BoF	1	$\mathbf{70.52 \pm 0.57}$	$\mathbf{75.29 \pm 0.61}$	$\mathbf{77.57 \pm 0.59}$	$\mathbf{78.56 \pm 0.69}$	$\mathbf{79.50 \pm 0.21}$
BoF + SVM	1	62.84 ± 0.86	69.99 ± 0.46	74.39 ± 0.50	77.02 ± 0.47	78.42 ± 0.34
Neural BoF + SVM	1	$\mathbf{72.30 \pm 0.53}$	$\mathbf{77.32 \pm 0.69}$	$\mathbf{80.49 \pm 0.23}$	$\mathbf{82.00 \pm 0.51}$	$\mathbf{82.63 \pm 0.49}$
BoF + MLP	2	64.79 ± 1.13	70.93 ± 0.30	73.50 ± 0.68	74.96 ± 0.30	76.33 ± 0.75
Neural BoF	2	$\mathbf{73.26 \pm 0.34}$	$\mathbf{77.34 \pm 0.79}$	$\mathbf{79.31 \pm 0.47}$	$\mathbf{80.49 \pm 0.52}$	$\mathbf{81.41 \pm 0.51}$
BoF + SVM	2	68.66 ± 0.40	74.00 ± 0.56	76.82 ± 0.79	78.90 ± 0.49	80.60 ± 0.20
Neural BoF + SVM	2	$\mathbf{75.98 \pm 0.74}$	$\mathbf{79.79 \pm 0.40}$	$\mathbf{81.95 \pm 0.57}$	$\mathbf{83.34 \pm 0.34}$	$\mathbf{83.83 \pm 0.43}$

Table 3: 15-scene: Classification results using the HOG+LBP features

	Number of RBF neurons / codewords				
Type	8	16	32	64	128
BoF + MLP	56.88 ± 0.42	64.33 ± 1.22	68.21 ± 0.58	71.59 ± 0.57	73.05 ± 1.01
Neural BoF	71.05 ± 0.94	74.60 ± 1.57	77.01 ± 2.02	78.11 ± 1.04	77.46 ± 1.46
Neural BoF (fine-tuned)	$\mathbf{74.29 \pm 1.00}$	$\mathbf{76.67 \pm 0.91}$	$\mathbf{77.90 \pm 0.65}$	$\mathbf{78.23 \pm 0.41}$	$\mathbf{79.44 \pm 0.28}$
BoF + SVM	57.81 ± 0.61	65.29 ± 1.02	71.24 ± 1.04	74.68 ± 1.06	77.01 ± 0.87
Neural BoF + SVM	$\mathbf{74.70 \pm 1.45}$	$\mathbf{77.94 \pm 1.09}$	$\mathbf{80.46 \pm 0.84}$	$\mathbf{81.35 \pm 0.82}$	$\mathbf{82.44 \pm 0.65}$

(SPM) [6]. The Neural BoF layer can be also used to generate histograms using the SPM scheme. Each part of the input image is treated as a separated image and the corresponding histogram is extracted. Then, these histograms are fused together, normalized and fed to a classifier.

Optimizing the Neural BoF layer for each pyramid level is also possible but increases the computational cost. Therefore, the Neural BoF is trained using the whole image (SPM level 0) as before and then repeatedly applied to extract the higher level representations during the testing. Note that the learned output layer (MLP) cannot be used since the length of the extracted histograms increases when the SPM is used. Therefore, a new MLP layer (300 iterations for the SPM level 1 and 200 iterations for the SPM level 2) and a new SVM (C=10 for both levels) are trained. The results are shown in Table 2. Even though the Neural BoF layer is not optimized for the extracted representation, it increases the recognition precision up to 10%.

The experiments were repeated using the fused HOG+LBP feature vectors. The results are shown in Tables 3 and 4. Again, a large increase in the precision is observed when the Neural BoF layer is trained using the proposed method (both with and without the use of the SPM technique).

Table 5 compares the Neural BoF to the state-of-the-art methods. The best two learned Neural BoF layers, the Neural BoF (128 neurons) for the SIFT features and the Neural BoF (128 neurons) for the HOG+LBP features, can be also "vertically" stacked, leading to a "wider" layer (with 256 RBF neurons) that receives both feature vectors and produces a unified histogram (which is renormalized to sum to 1). The stacked Neural BoF layers led to better recognition precision (87.30 ± 0.99), than any other single Neural BoF layer alone. SPM at level 1 and an SVM with the χ^2 kernel (C=10) were used. To the best of our knowledge, this is the best reported result on the 15-scene dataset using the BoF model and without using any external data.

Table 4: 15-scene: Classification results using the HOG+LBP features and the SPM scheme

		Number of RBF neurons / codewords				
Type	**Level**	8	16	32	64	128
BoF + MLP	1	62.98 ± 0.62	69.20 ± 0.60	72.08 ± 0.92	74.80 ± 0.48	75.93 ± 0.90
Neural BoF	1	$\mathbf{75.82 \pm 0.75}$	$\mathbf{78.26 \pm 0.79}$	$\mathbf{79.91 \pm 0.96}$	$\mathbf{80.00 \pm 0.26}$	$\mathbf{80.86 \pm 0.83}$
BoF + SVM	1	65.90 ± 0.41	72.33 ± 0.51	76.31 ± 0.67	78.80 ± 0.63	80.29 ± 0.82
Neural BoF + SVM	1	$\mathbf{77.12 \pm 1.10}$	$\mathbf{80.07 \pm 0.82}$	$\mathbf{82.33 \pm 0.82}$	$\mathbf{83.41 \pm 0.76}$	$\mathbf{84.68 \pm 0.78}$
BoF + MLP	2	66.86 ± 0.59	72.02 ± 0.56	74.13 ± 0.80	76.18 ± 0.72	77.71 ± 0.46
Neural BoF	2	$\mathbf{77.03 \pm 0.50}$	$\mathbf{79.73 \pm 0.76}$	$\mathbf{81.35 \pm 0.80}$	$\mathbf{82.30 \pm 0.73}$	$\mathbf{82.52 \pm 0.61}$
BoF + SVM	2	70.01 ± 0.87	75.27 ± 0.79	78.42 ± 0.70	80.54 ± 0.80	81.79 ± 0.97
Neural BoF + SVM	2	$\mathbf{79.20 \pm 1.11}$	$\mathbf{82.00 \pm 0.62}$	$\mathbf{83.96 \pm 0.74}$	$\mathbf{85.03 \pm 1.00}$	$\mathbf{85.50 \pm 0.77}$

Table 5: 15-scene: Comparison of the Neural BoF to the state-of-the-art methods

Method	**# codewords**	**Precision**
Object Bank [53]	2400	80.90
SPM [6]	200	81.30 ± 0.30
RBM Dictionary Learning [27]	1024	86.00 ± 0.50
MMDL [13]	5250	86.43 ± 0.41
MMDL (Multiple Instance) [15]	165	86.35 ± 0.45
Kernel Descriptors [54]	1000	86.70 ± 0.40
Manifold Deep Learning [55]	-	86.90
Neural BoF+SVM (SPM 2, S)	128	83.83 ± 0.43
Neural BoF+SVM (SPM 2, H+L)	128	85.50 ± 0.77
Neural BoF+SVM (SPM 0, C)	2×128	85.59 ± 0.83
Neural BoF+SVM (SPM 1, C)	2×128	$\mathbf{87.30 \pm 0.85}$

S stands for the SIFT features, H+L for the HOG+LBP features and C for the combined SIFT and HOG+LBP neural BoF layers.

Table 6: 15-scene: Evaluation of the incremental classification algorithm

Method	**Precision**	**# features**
All Features (S)	76.09 ± 1.02	960.64 ± 69.19
Streaming (S)	75.41 ± 1.12	493.09 ± 384.35
All Features (H+L)	77.46 ± 1.46	1023.98 ± 64.27
Streaming (H+L)	77.23 ± 1.29	489.00 ± 411.06

S stands for the SIFT features and H+L for the HOG+LBP features.

Table 7: 15-scene: Classification results using convolutional features

	Number of RBF neurons / codewords				
Type	8	16	32	64	128
BoF + MLP	47.27 ± 1.39	67.26 ± 0.65	76.50 ± 4.22	85.96 ± 2.40	88.53 ± 2.02
Neural BoF	$\mathbf{80.90 \pm 3.01}$	$\mathbf{87.40 \pm 2.56}$	$\mathbf{86.94 \pm 4.40}$	$\mathbf{91.81 \pm 2.66}$	$\mathbf{91.78 \pm 2.00}$

The incremental classification algorithm is also evaluated in Table 6 using a Neural BoF layer with 128 neurons followed by an MLP. The number of the extracted features is greatly reduced leading to faster classification without significantly harming the recognition precision. Note that the number of the extracted features during the streaming process varies greatly between different images, with the "easier" images requiring less features and "harder" images requiring almost all the available features.

In Figure 6 the patches of each image that activate the top-3 RBF neurons that are associated with the correct class are highlighted using the method proposed in Subsection 3.6. The Neural BoF network better identifies the part of the image which is related to the correct label. For example, for the MIThighway class, the unsupervised Neural BoF focuses its attention on the sky (which is not a discriminative feature), while the supervised Neural BoF shifts its attention on the road lanes (which indeed discriminates this class). Similar observations can be made for the other images as well.

Finally, the Neural BoF method is combined with the pretrained VGG-16 neural network using the feature vectors extracted from its last convolutional layer [19]. The results are shown in Table 7. Both the BoF + MLP and the Neural BoF were trained for 100 iterations. As before, the Neural BoF leads to significant precision improvements over the regular BoF model (over 30% when 8 codewords are used), although the use of convolutional features led to greater variance. This also allows to use less RBF neurons / codewords without reducing the recognition precision. Using convolutional feature vectors leads to significantly better precision than using SIFT/HOG+LBP feature vectors (Table 5). Note that these results are not directly comparable since external data were used to train the CNN.

4.3. MNIST Dataset Evaluation

Similar experiments were conducted using the MNIST dataset. The results are shown in Table 8. Again, $C = 10$ was used for the SVM and 50 iterations were used for the training of the MLP layer and the fine-tunning. The classification accuracy increases by more than 20% (when 8 RBF neurons are used). If more codewords (larger

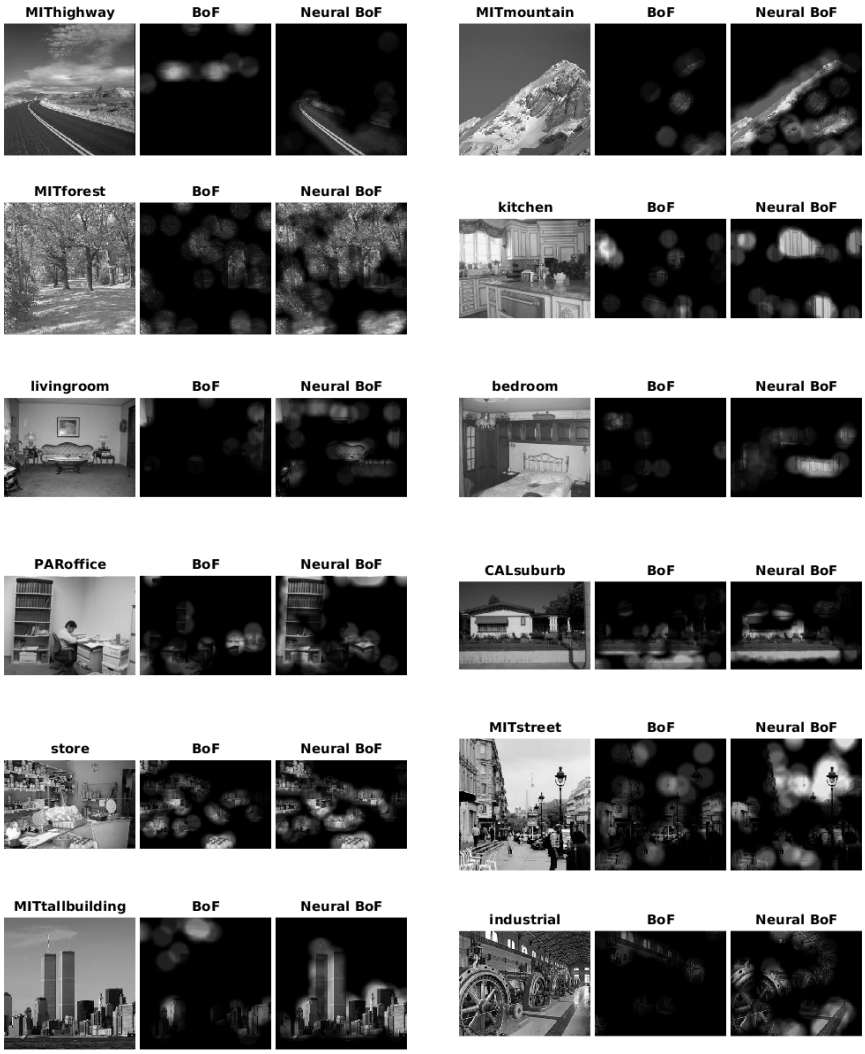


Figure 6: 15-scene: Comparison between the BoF and the Neural BoF methods. The patches that activate the top-3 RBF neurons for each class are highlighted.

Table 8: MNIST: Classification results using 9x9 raw pixel patches

	# neurons / codewords				
Type	8	16	32	64	128
BoF + MLP	71.80	92.49	96.40	97.59	98.15
Neural BoF	92.40	95.39	97.56	97.97	98.61
Neural BoF (finet.)	**94.85**	**97.46**	**98.43**	**98.79**	**98.90**
BoF + SVM	68.52	89.48	94.63	96.70	97.85
Neural BoF + SVM	**94.43**	**97.13**	**98.17**	**98.88**	**99.07**

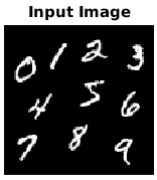


Figure 7: MNIST: 10 sample digits placed in various positions.

Neural BoF layers) are used, then the (unsupervised) recognition rate is significantly better and the accuracy increase after the training is smaller. The accuracy of the Neural BoF + SVM (99.07%) is comparable to that of a convolutional network (99.05%, LeNet-5 [56]).

Figure 7 shows an image that contains 10 digits (0 to 9). A similar procedure, as in the 15-scene, was used to illustrate the visual attention of the neural network during the classification. Instead of the top-3 RBF neurons, every neuron contributes in the process of creating the mask. Also, instead of using a disk filter, a square 20x20 average filter is applied to the mask, the same exponential function ($a = 5$) is utilized and then the mask is re-filterted and normalized to 0-1. The visual attention of each digit is shown in Figure 8. The Neural BoF with 32 RBF neurons was used for the visualization. It can be observed that the Neural BoF identifies and shifts its attention to the correct digits, while its attention to the wrong digits is greatly reduced. This is clearly illustrated for the digits 3, 4, 8 and 9.

The incremental classification algorithm is evaluated in Table 9 using 128 RBF neurons. Again, the feature

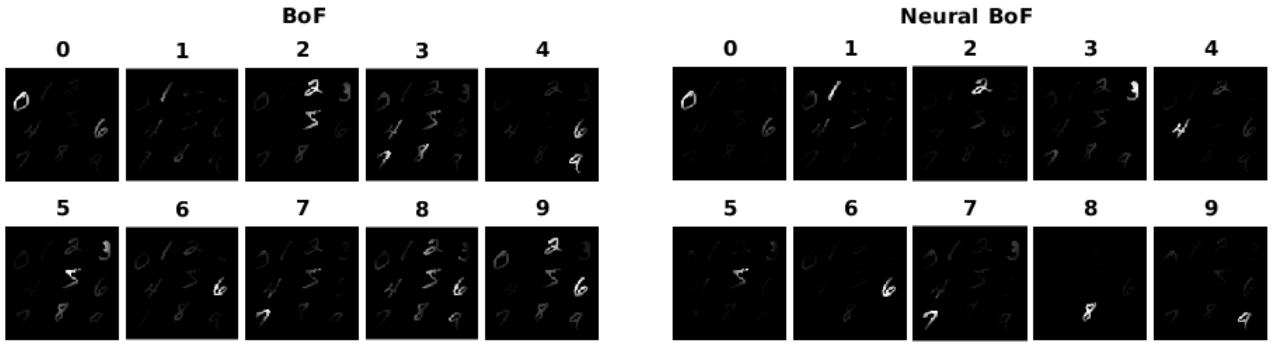


Figure 8: Highlighting visual attention of the BoF and the Neural BoF for the input image in Figure 7. The regions that are mostly activated for each class (digits 0-9) are highlighted.

Table 9: MNIST: Evaluation of the incremental classification algorithm

Method	N_k	**Precision**	**# features**
All Features	128	98.90	367.5 ± 37.5
Streaming	128	98.66	158.7 ± 36.9
All Features	64	98.79	367.5 ± 37.5
Streaming	64	98.52	161.1 ± 40.0
All Features	32	98.43	367.5 ± 37.5
Streaming	32	97.67	165.1 ± 46.8
All Features	16	97.46	367.5 ± 37.5
Streaming	16	96.60	176.6 ± 62.3
All Features	8	94.85	367.5 ± 37.5
Streaming	8	92.76	191.7 ± 74.9

Table 10: RT-2k: Classification results using the RNTN sentiment vectors

	Number of RBF neurons / codewords				
Method	2	4	8	16	32
BoF + MLP	80.05 ± 3.55	82.75 ± 2.85	83.30 ± 3.19	83.90 ± 2.69	83.80 ± 3.12
Neural BoF	**82.70 ± 2.43**	**83.40 ± 3.29**	**84.30 ± 3.63**	**84.45 ± 3.44**	**84.25 ± 3.62**
BoF + SVM	79.85 ± 3.36	82.60 ± 2.67	83.85 ± 3.31	83.60 ± 3.06	83.65 ± 3.19
Neural BoF + SVM	**82.85 ± 2.59**	**83.70 ± 3.34**	**84.30 ± 3.31**	**84.20 ± 3.24**	**83.90 ± 2.98**

streaming technique allows to reduce the classification time, with only a minor effect on the accuracy.

4.4. Polarity Dataset v2.0 Evaluation

The performance of the proposed method was also evaluated using the RT-2k dataset. The results are shown in Table 10 ($C = 20$ was used for the SVM and 300 training iterations were used for the MLP). Optimizing the Neural BoF layer for the RT-2k was more difficult than for the other datasets, since the method was very prone to over-fitting due to the nature of the used (sentiment) feature vectors and the structure of the dataset. Even though, the Neural BoF always improves the classification performance. The largest increase is observed when smaller Neural BoF layers are used. The incremental classification algorithm was not evaluated for this dataset, due to the small number of extracted features from each document. The proposed method exceeds the best reported accuracy (83.15%) for sentence-level feature aggregation using the BoF model [50].

4.5. SUN397 Dataset Evaluation

The proposed method was also evaluated in a large-scale setting using the SUN397 dataset. To further accelerate the convergence of the network the ELU activation function is used in the hidden layer [57], and the softmax activation function in the output layer. Both the BoF+MLP and the Neural BoF were trained for 10 iterations. The results are reported in Table 11. The training time includes both the time needed to initialize the network using the k-means algorithm as well as the time needed to train the MLP / Neural BoF model (including the time needed for feeding the feature vectors). The Neural BoF significantly increases the testing accuracy over the BoF + MLP model when the same number (64) of codewords / RBF neurons are used. The Neural BoF model still

Table 11: SUN937: Large-scale evaluation

Method	# RBF neurons / codewords	Test accuracy	Training time	Testing time
BoF + MLP	64	33.28	123.9 mins (3.4 mins)	9.6 mins
BoF + MLP	1024	53.38	627.9 mins (30.3 mins)	57.2 mins
Neural BoF	64	**57.46**	236.9 mins (3.4 mins)	9.5 mins

The time consumed for initializing the model using k-means is reported in the parenthesis (training time).

performs better even when the BoF+MLP uses one order of magnitude more codewords (1024 codewords instead of 64 codewords). This allows to use only 64 RBF neurons, reducing both the training and the testing time without harming the recognition accuracy. The method was implemented using the Theano library [42], which allows to optimize the networks using the GPU, and a 4-core workstation with 16GB of RAM and a mid-range GPU with 2GB of RAM were used for the conducted experiments.

4.6. Statistical Significance Analysis

The Wilcoxon signed rank test is used to validate the statistical significance of the obtained results [58]. The BoF + MLP method is compared to the proposed Neural BoF method. The null hypothesis H_0 is defined as: "*There is no statistical significant difference between the BoF + MLP and the Neural BoF methods when X RBF neurons/codewords are used*", where $X = \{16,\ 32,\ 64,\ 128\}$. The null hypothesis H_0 is rejected at significance level $a = 0.05$ for any number of RBF neurons/codewords.

4.7. Parameter Selection

The proposed method typically requires little tuning of its hyper-parameters. The chosen learning rates ($\eta_{MLP} = 0.01$, $\eta_V = 0.001$ and $\eta_W = 0.2$) worked well for most of the conducted experiments. For the large-scale experiments slightly different values were used ($\eta_V = 10^{-4}$ and $\eta_W = 0.01$), since different activation functions were utilized. The number of iterations (N_{iters}) and the number of feature samples ($N_{samples}$) mostly depend on the available resources (increasing these values usually improves the classification accuracy, as long as the network does not over-fit the data, but also increases the training time). The number of the pre-train iterations (N_{piters}) of the output layer also depends on the available resources. If enough processing time is available, the pre-training can be skipped. However, if pre-training is used, care should be given to avoid initializing the network with an already over-trained output layer. The memory of the accumulation layer is initialized to zero before each iteration since enough features are sampled for each dataset. This also prevents, to some extent, over-fitting the Neural BoF layer.

The remaining two hyper-parameters, the value used to initialize the Neural BoF weights (g) and the number of hidden neurons (N_H) can be chosen using a validation set and simple line search in the parameter space. Usually, the values for the g range between 0.005 and 0.5 and about 100-200 hidden neurons are used in the hidden layer.

A set of experiments were conducted to evaluate the effect of the hyper-parameter g on the learning process. The training set was randomly split into a new training set and a validation set: 500 training and 1000 validation images were used for the 15-scene, 1000 training/validation images for the MNIST and 1800 training and 200 validation

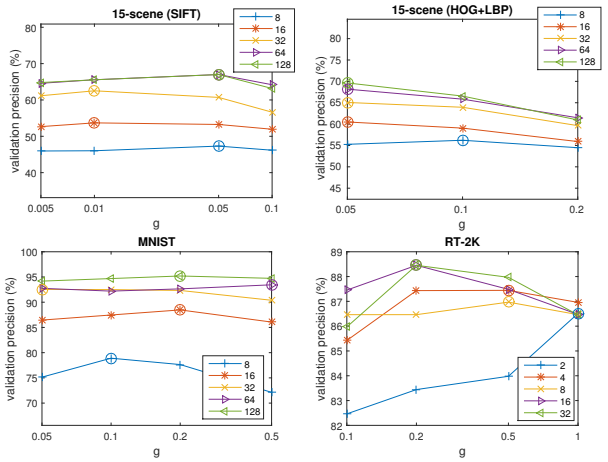


Figure 9: Validation set accuracy for different values of g and different number of RBF neurons (as shown in the legend).

documents for the RT-2k dataset. The output layer was pre-trained (50 iterations) and the network was trained for 20 iterations (10 iterations for the RT2-k). The results are shown in Figure 9. Using very small values (< 0.05) for the g might cause numerical stability issues. This happened for both the 15-scene dataset (HOG+LBP features) and the MNIST dataset.

Excluding the RT-2k dataset, the effect of g on the validation accuracy is consistent regardless the number of the RBF neurons. For the 15-scene $g = 0.05$ is selected (for both features). For the MNIST dataset the validation accuracy remains stable regardless the choice of the parameter and, therefore, $g = 0.15$ is chosen. However, for the RT-2k the optimal value of g depends on the number of the RBF neurons: $g = 1$ is chosen when 2 RBF neurons are used, $g = 0.5$ for 4 and 8 neurons and $g = 0.2$ for 16 and 32 neurons. When convolutional features are used g is set to 0.01 for the 15-scene dataset, while g is set to 0.1 for the SUN397 dataset.

A similar experiment was conducted to evaluate the effect of the hidden layer size. The results are shown in Figure 10. Using a hidden layer in the MLP leads to better validation accuracy in every case. For the 15-scene (SIFT, HOG+LBP) and the MNIST 200 hidden neurons are used, while for the RT-2k, which is more prone to overfitting, only 10 hidden neurons are used. When convolutional features are used, 100 hidden neurons are utilized for the 15-scene dataset and 200 hidden neurons for the SUN397 dataset

Table 12 summarizes the selected hyper-parameters for each dataset. To reduce the over-fitting for the RT-2k dataset, aggressive sub-sampling is used (30 sampled features) and the batch size is increased to 200 objects (instead of 50). For the incremental classification algorithm 50 features were initially sampled and then the features were fed to the network in mini-batches of 50.

5. Conclusions and Extensions

In this paper, a neural generalization of the well-known BoF model was introduced. The proposed model can be either trained in an unsupervised fashion, like the early BoF approaches, or supervised (when connected to

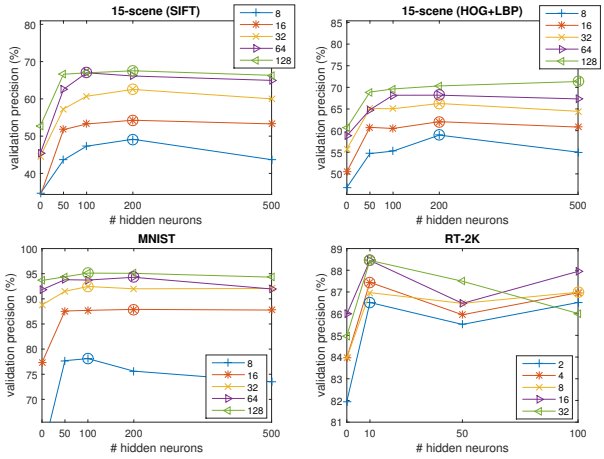


Figure 10: Validation set accuracy for different number of hidden neurons (0 means that no hidden layer is used) and different number of RBF neurons (as shown in the legend).

Table 12: Parameters used in the conducted experiments

Dataset	N_{iters}	N_{piters}	N_{batch}	$N_{samples}$	# hidden neurons	g
15-scene (SIFT/HOG+LBP)	100	50	50	200	200	0.05
15-scene (Convolutional)	100	0	50	196	100	0.01
MNIST	10	10	50	200	200	0.15
RT-2k	50	50	200	30	10	0.2-1
SUN397	10	0	50	150	200	0.1

a classification layer). It was demonstrated, using four different datasets, that the classification accuracy greatly increases when the Neural BoF layer is trained using the proposed supervised algorithm. This allows smaller Neural BoF layers to be used, reducing the time needed for training and testing.

Also, the Neural BoF network natively supports learning and classifying from feature streams. Two incremental algorithms (for both the training and the classification tasks) were proposed. The incremental learning with feature streaming can reduce the training and the testing time of the network, without significantly harming its classification ability. Therefore, it allows the proposed method to successfully scale to larger datasets. Also, it was found that the noise which is introduced during the feature streaming process can, sometimes, mitigate the effects of over-fitting.

Furthermore, the proposed neural architecture provides a framework that can model and extend the dictionary learning methodology. There are many possible ways to extend the proposed method: the output layer can be replaced by any other classifier with differentiable loss function, e.g., a max-margin classifier [13, 15], or a logistic regression classifier [33], or even by a discrimination criterion, e.g., [9, 37]. Also, the RBF neurons can be replaced by regular linear units [15]. The Neural BoF can be also combined with a retrieval-oriented loss function [59], to learn compact retrieval-oriented representations. That way, the Neural BoF can be easily extended and, possibly, improve some of the already proposed methods.

Finally, it was demonstrated that the Neural BoF layer can be successfully used with the features extracted from a CNN [19]. Furthermore, the Neural BoF gradients can also backpropagate to the CNN allowing the optimization of the whole neural network. That way, the Neural BoF approach can not only adjust the input weights and the

centers of the RBF neurons, but also optimize the used feature extractor, i.e., the CNN. Also, in [60], a simple spatial pooling layer is used after the convolutional layer (but before the fully connected layer) to allow images of arbitrary size to be fed to the network. The Neural BoF could be used after this spatial pooling layer to increase the scale and the location invariance of the network.

6. References

[1] D. G. Lowe, Object recognition from local scale-invariant features, in: Proceedings of the 7th IEEE international conference on Computer Vision, Vol. 2, 1999, pp. 1150–1157.

[2] N. Dalal, B. Triggs, Histograms of oriented gradients for human detection, in: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Vol. 1, 2005, pp. 886–893.

[3] A. Coates, A. Y. Ng, Learning feature representations with k-means, in: Neural Networks: Tricks of the Trade, 2012, pp. 561–580.

[4] K. Mikolajczyk, C. Schmid, Scale & affine invariant interest point detectors, International Journal of Computer Vision 60 (1) (2004) 63–86.

[5] J. Sivic, A. Zisserman, Video google: A text retrieval approach to object matching in videos, in: Proceedings of the 9th IEEE International Conference on Computer Vision, 2003, pp. 1470–1477.

[6] S. Lazebnik, C. Schmid, J. Ponce, Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories, in: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Vol. 2, 2006, pp. 2169–2178.

[7] Y.-G. Jiang, C.-W. Ngo, J. Yang, Towards optimal bag-of-features for object categorization and semantic video retrieval, in: Proceedings of the 6th ACM International Conference on Image and Video Retrieval, 2007, pp. 494–501.

[8] M. Riley, E. Heinen, J. Ghosh, A text retrieval approach to content-based audio retrieval, in: Proceedings of the 9th International Conference on Music Information, 2008, pp. 295–300.

[9] A. Iosifidis, A. Tefas, I. Pitas, Multidimensional sequence classification based on fuzzy distances and discriminant analysis, IEEE Transactions on Knowledge and Data Engineering 25 (11) (2013) 2564–2575.

[10] G. Salton, C. Buckley, Term-weighting approaches in automatic text retrieval, Information processing & management 24 (5) (1988) 513–523.

[11] F. Perronnin, C. Dance, G. Csurka, M. Bressan, Adapted vocabularies for generic visual categorization, in: Proceedings of the 9th European Conference on Computer Vision, 2006, pp. 464–475.

[12] S. Lazebnik, M. Raginsky, Supervised learning of quantizer codebooks by information loss minimization, IEEE Transactions on Pattern Analysis and Machine Intelligence 31 (7) (2009) 1294–1309.

[13] X.-C. Lian, Z. Li, B.-L. Lu, L. Zhang, Max-margin dictionary learning for multiclass image categorization, in: Proceedings of the 11th European Conference on Computer Vision, 2010, pp. 157–170.

[14] W. Zhang, A. Surve, X. Fern, T. Dietterich, Learning non-redundant codebooks for classifying complex objects, in: Proceedings of the 26th Annual International Conference on Machine Learning, 2009, pp. 1241–1248.

[15] X. Wang, B. Wang, X. Bai, W. Liu, Z. Tu, Max-margin multiple-instance dictionary learning, in: Proceedings of the 30th International Conference on Machine Learning, 2013, pp. 846–854.

[16] H. Lobel, R. Vidal, D. Mery, A. Soto, Joint dictionary and classifier learning for categorization of images using a max-margin framework, in: Image and Video Technology, 2014, pp. 87–98.

[17] J. Park, I. W. Sandberg, Universal approximation using radial-basis-function networks, Neural Computation 3 (2) (1991) 246–257.

[18] S. S. Haykin, Neural networks and learning machines, Vol. 3, Pearson Education Upper Saddle River, 2009.

[19] B. Zhou, A. Lapedriza, J. Xiao, A. Torralba, A. Oliva, Learning deep features for scene recognition using places database, in: Advances in neural information processing systems, 2014, pp. 487–495.

[20] N. van Noord, E. Postma, Learning scale-variant and scale-invariant features for deep image classification, Pattern Recognition (in press).

[21] M. Denil, L. Bazzani, H. Larochelle, N. de Freitas, Learning where to attend with deep architectures for image tracking, Neural computation 24 (8) (2012) 2151–2184.

[22] Q. Meng, B. Li, H. Holstein, Y. Liu, Parameterization of point-cloud freeform surfaces using adaptive sequential learning RBF networks, Pattern Recognition 46 (8) (2013) 2361 – 2375.

[23] M. Carozza, S. Rampone, Function approximation from noisy data by an incremental RBF network, Pattern Recognition 32 (12) (1999) 2081 – 2083.

[24] Z. C. Lipton, A critical review of recurrent neural networks for sequence learning, arXiv:1506.00019.

[25] B. Pearlmutter, Gradient calculations for dynamic recurrent neural networks: A survey, IEEE Transactions on Neural Networks 6 (5) (1995) 1212–1228.

[26] M. Egmont-Petersen, D. de Ridder, H. Handels, Image processing with neural networksa review, Pattern Recognition 35 (10) (2002) 2279 – 2301.

[27] H. Goh, N. Thome, M. Cord, J.-H. Lim, Unsupervised and supervised visual codes with restricted boltzmann machines, in: Proceedings of the 12th European Conference on Computer Vision, 2012, pp. 298–311.

[28] H. Zhang, Y. Zhang, T. S. Huang, Simultaneous discriminative projection and dictionary learning for sparse representation based classification, Pattern Recognition 46 (1) (2013) 346 – 354.

[29] D. Zhang, P. Liu, K. Zhang, H. Zhang, Q. Wang, X. Jing, Class relatedness oriented-discriminative dictionary learning for multiclass image classification, Pattern Recognition (in press).

[30] Y. Kuang, K. Åström, L. Kopp, M. Oskarsson, M. Byröd, Optimizing visual vocabularies using soft assignment entropies, in: Proceedings of the 10th Asian Conference on Computer Vision, 2011, pp. 255–268.

[31] Y. Kuang, M. Byröd, K. Åström, Supervised feature quantization with entropy optimization, in: IEEE International Conference on Computer Vision Workshops (ICCV Workshops), 2011, pp. 1386–1393.

[32] M. Jiu, C. Wolf, C. Garcia, A. Baskurt, Supervised learning and codebook optimization for bag-of-words models, Cognitive Computation 4 (4) (2012) 409–419.

[33] Y.-L. Boureau, F. Bach, Y. LeCun, J. Ponce, Learning mid-level features for recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2010, pp. 2559–2566.

[34] F. Perronnin, Universal and adapted vocabularies for generic visual categorization, IEEE Transactions on Pattern Analysis and Machine Intelligence 30 (7) (2008) 1243–1256.

[35] B. Fulkerson, A. Vedaldi, S. Soatto, Localizing objects with smart dictionaries, in: Proceedings of the 10th European Conference on Computer Vision, 2008, pp. 179–192.

[36] J. Winn, A. Criminisi, T. Minka, Object categorization by learned universal visual dictionary, in: Proceedings of the 10th IEEE International Conference on Computer Vision, Vol. 2, 2005, pp. 1800–1807.

[37] A. Iosifidis, A. Tefas, I. Pitas, Discriminant bag of words based representation for human action recognition, Pattern Recognition Letters 49 (2014) 185–192.

[38] C. Lang, S. Feng, B. Cheng, B. Ni, S. Yan, A unified supervised codebook learning framework for classification, Neurocomputing 77 (1) (2012) 281–288.

[39] D. Kingma, J. Ba, Adam: A method for stochastic optimization, arXiv:1412.6980.

[40] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: A simple way to prevent neural networks from overfitting, The Journal of Machine Learning Research 15 (1) (2014) 1929–1958.

[41] F. Schwenker, H. A. Kestler, G. Palm, Three learning phases for radial-basis-function networks, Neural Networks 14 (4) (2001) 439–458.

[42] Theano Development Team, Theano: A Python framework for fast computation of mathematical expressions, arXiv e-prints abs/1605.02688.
URL http://arxiv.org/abs/1605.02688

[43] Y. LeCun, C. Cortes, C. J. Burges, The mnist database of handwritten digits (1998).

[44] B. Pang, L. Lee, A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts, in: Proceedings of the 42nd annual meeting on Association for Computational Linguistics, 2004, p. 271.

[45] J. Xiao, J. Hays, K. A. Ehinger, A. Oliva, A. Torralba, Sun database: Large-scale scene recognition from abbey to zoo, in: IEEE Conference on Computer Vision and Pattern Recognition, 2010, pp. 3485–3492.

[46] T. Ojala, M. Pietikäinen, T. Mäenpää, Multiresolution gray-scale and rotation invariant texture classification with local binary patterns, IEEE Transactions on Pattern Analysis and Machine Intelligence 24 (7) (2002) 971–987.

[47] A. Vedaldi, B. Fulkerson, VLFeat: An open and portable library of computer vision algorithms, http://www.vlfeat.org/ (2008).

[48] R. Socher, A. Perelygin, J. Y. Wu, J. Chuang, C. D. Manning, A. Y. Ng, C. Potts, Recursive deep models for semantic compositionality over a sentiment treebank, in: Proceedings of the Conference on Empirical Methods in Natural Language Processing, Vol. 1631, 2013, p. 1642.

[49] C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, D. McClosky, The Stanford CoreNLP natural language processing toolkit, in: Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations, 2014, pp. 55–60.

[50] D. Chatzakou, N. Passalis, A. Vakali, Multispot: Spotting sentiments with semantic aware multilevel cascaded analysis, in: Big Data Analytics and Knowledge Discovery, 2015, pp. 337–350.

[51] J. Zhang, M. Marszałek, S. Lazebnik, C. Schmid, Local features and kernels for classification of texture and object categories: A comprehensive study, International Journal of Computer Vision 73 (2) (2007) 213–238.

[52] C.-C. Chang, C.-J. Lin, LIBSVM: A library for support vector machines, ACM Transactions on Intelligent Systems and Technology 2 (2011) 1–27.

[53] L.-J. Li, H. Su, L. Fei-Fei, E. P. Xing, Object bank: A high-level image representation for scene classification & semantic feature sparsification, in: Proceedings of the Advances in Neural Information Processing Systems, 2010, pp. 1378–1386.

[54] L. Bo, X. Ren, D. Fox, Kernel descriptors for visual recognition, in: Proceedings of the Advances in Neural Information Processing Systems, 2010, pp. 244–252.

[55] Y. Yuan, L. Mou, X. Lu, Scene recognition by manifold regularized deep learning architecture, IEEE Transactions on Neural Networks and Learning Systems 26 (10) (2015) 2222–2233.

[56] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proceedings of the IEEE 86 (11) (1998) 2278–2324.

[57] D.-A. Clevert, T. Unterthiner, S. Hochreiter, Fast and accurate deep network learning by exponential linear units (elus), arXiv preprint arXiv:1511.07289.

[58] J. D. Gibbons, S. Chakraborti, Nonparametric statistical inference, Springer, 2011.

[59] N. Passalis, A. Tefas, Entropy optimized feature-based bag-of-words representation for information retrieval, IEEE Transactions on Knowledge and Data Engineering 28 (7) (2016) 1664–1677.

[60] K. He, X. Zhang, S. Ren, J. Sun, Spatial pyramid pooling in deep convolutional networks for visual recognition, IEEE Transactions on Pattern Analysis and Machine Intelligence 37 (9) (2015) 1904–1916.

Appendix A. Equivalence Proof of the Recurrent Definition

The following theorem is proved:

Theorem 1. *The output of the Neural BoF layer, which is calculated as* $\mathbf{s'}_i(\boldsymbol{x}(t), t) = \frac{1}{t}\boldsymbol{\phi}(\boldsymbol{x}(t)) + \frac{t-1}{t}\mathbf{s'}_i(\boldsymbol{x}(t-1), t-1)$*, equals to the histogram defined by* $\mathbf{s}_i = \frac{1}{t}\sum_{j=1}^{t}\boldsymbol{\phi}(\mathbf{x}_{ij})$ *when the first* t *features of the* i*-th object are fed to the network.*

Proof. The output of the Neural BoF layer through time is defined as:

The output of the Neural BoF layer through time is defined as:

$$\mathbf{s'}_i(x, 0) = 0 \tag{A.1}$$

$$\mathbf{s'}_i(\mathbf{x}_{i1}, 1) = \frac{1}{1}\boldsymbol{\phi}(\mathbf{x}_{i1}) \tag{A.2}$$

$$\mathbf{s'}_i(\mathbf{x}_{i2}, 2) = \frac{1}{2}\boldsymbol{\phi}(\mathbf{x}_{i2})) + \frac{1}{2}\mathbf{s'}_i(\mathbf{x}_{i1}, 1) \tag{A.3}$$

$$\mathbf{s'}_i(\mathbf{x}_{i3}, 3) = \frac{1}{3}\boldsymbol{\phi}(\mathbf{x}_{i3})) + \frac{2}{3}\mathbf{s'}_i(\mathbf{x}_{i2}, 2) \tag{A.4}$$

$$\vdots$$

$$\mathbf{s'}_i(\mathbf{x}_{i(t-1)}, t-1) = \frac{1}{t-1}\boldsymbol{\phi}(\mathbf{x}_{i(t-1)})) + \frac{t-2}{t-1}\mathbf{s'}_i(\mathbf{x}_{i(t-2)}, t-2) \tag{A.5}$$

$$\begin{aligned}\mathbf{s'}_i(\mathbf{x}_{it},t) &= \frac{1}{t}\boldsymbol{\phi}(\mathbf{x}_{it})) \\ &+ \frac{t-1}{t}\mathbf{s'}_i(\mathbf{x}_{i(t-1)},t-1)\end{aligned} \tag{A.6}$$

By substituting (A.5) into (A.6) the following formula is obtained:

$$\begin{aligned}\mathbf{s'}_i(\mathbf{x}_{it},t) = \frac{1}{t}\boldsymbol{\phi}(\mathbf{x}_{it}) + \frac{t-1}{t}\frac{1}{t-1}\boldsymbol{\phi}(\mathbf{x}_{i(t-1)}))\\ + \frac{t-1}{t}\frac{t-2}{t-1}\mathbf{s'}_i(\mathbf{x}_{i(t-2)},t-2)\end{aligned} \tag{A.7}$$

Then, by substituting every $\mathbf{s'}_i(\mathbf{x}_{i(t)},t)$:

$$\begin{aligned}\mathbf{s'}_i(\mathbf{x}_{it},t) = \frac{1}{t}\boldsymbol{\phi}(\mathbf{x}_{it}) + \frac{t-1}{t}\frac{1}{t-1}\boldsymbol{\phi}(\mathbf{x}_{i(t-1)}))\\ + \cdots \\ + \frac{t-1}{t}\frac{t-2}{t-1}\cdots\frac{3}{4}\frac{1}{3}\boldsymbol{\phi}(\mathbf{x}_{i3}) \\ + \frac{t-1}{t}\frac{t-2}{t-1}\cdots\frac{3}{4}\frac{2}{3}\frac{1}{2}\boldsymbol{\phi}(\mathbf{x}_{i2}) \\ + \frac{t-1}{t}\frac{t-2}{t-1}\cdots\frac{3}{4}\frac{2}{3}\frac{1}{2}\boldsymbol{\phi}(\mathbf{x}_{i1})\end{aligned} \tag{A.8}$$

which equals to:

$$\begin{aligned}\mathbf{s'}_i(\mathbf{x}_{it},t) &= \frac{1}{t}\boldsymbol{\phi}(\mathbf{x}_{it}) + \frac{1}{t}\boldsymbol{\phi}(\mathbf{x}_{i(t-1)}) + \cdots \\ &+ \frac{1}{t}\boldsymbol{\phi}(\mathbf{x}_{i2}) + \frac{1}{t}\boldsymbol{\phi}(\mathbf{x}_{i1}) \\ &= \frac{1}{t}\sum_{j=1}^{t}\boldsymbol{\phi}(\mathbf{x}_{ij}) = \mathbf{s}_i\end{aligned} \tag{A.9}$$

□

Appendix B. Neural BoF Derivatives

Appendix B.1. Output Layer (MLP) Derivatives

In this subsection the derivatives of the MLP layer are calculated. Let $[\mathbf{W}_O]_i \in \mathbb{R}^{N_H}$ be the weight vector of the i-th output neuron and $\phi^{(s)\prime}(x) = \phi^{(s)}(x)(1-\phi^{(s)}(x))$ the derivative of the sigmoid function $\phi^{(s)}(x)$. First, the $\frac{\partial L}{\partial [\mathbf{W}_O]_i}$ is derived:

$$\frac{\partial L}{\partial [\mathbf{W}_O]_i} = \sum_{m=1}^{N}\frac{\partial L}{\partial [\mathbf{y}_m]_i}\frac{\partial [\mathbf{y}_m]_i}{[\mathbf{W}_O]_i} \tag{B.1}$$

where

$$\frac{\partial L}{\partial [\mathbf{y}_m]_i} = \frac{1-[\mathbf{t}_m]_i}{1-[\mathbf{y}_m]_i} - \frac{[\mathbf{t}_m]_i}{[\mathbf{y}_m]_i} \tag{B.2}$$

and

$$\begin{aligned}\frac{\partial [\mathbf{y}_m]_i}{\partial [\mathbf{W}_O]_i} &= \frac{\partial \phi^{(s)}([\mathbf{W}_O]_i^T\mathbf{h}_m + [\mathbf{b}_O]_i)}{\partial [\mathbf{W}_O]_i} \\ &= \phi^{(s)\prime}([\mathbf{W}_O]_i^T\mathbf{h}_m + [\mathbf{b}_O]_i)\frac{\partial([\mathbf{W}_O]_i^T\mathbf{h}_m + [\mathbf{b}_O]_i)}{\partial \mathbf{W}_{\mathbf{O}[i]}} \\ &= \phi^{(s)\prime}([\mathbf{W}_O]_i^T\mathbf{h}_m + [\mathbf{b}_O]_i)\mathbf{h}_m \in \mathbb{R}^{N_H}\end{aligned} \tag{B.3}$$

where the definitions (13) and (12) were used in (B.2) and (B.3) respectively.

Similarly, the derivative $\frac{\partial L}{\partial \mathbf{b}_O}$ is calculated:

$$\frac{\partial L}{\partial \mathbf{b_O}} = \sum_{m=1}^{N} \frac{\partial L}{\partial \mathbf{y}_m} \odot \frac{\partial \mathbf{y}_m}{\partial \mathbf{b_O}} \tag{B.4}$$

where

$$\frac{\partial L}{\partial \mathbf{y}_m} = \frac{1-\mathbf{t}_m}{1-\mathbf{y}_m} - \frac{\mathbf{t}_m}{\mathbf{y}_m} \in \mathbb{R}^{N_C} \tag{B.5}$$

and

$$\begin{aligned} \frac{\partial \mathbf{y}_m}{\partial \mathbf{b_O}} &= \frac{\partial sigm(\mathbf{W_O}\mathbf{h}_m + \mathbf{b_O})}{\partial \mathbf{b_O}} \\ &= \phi^{(s)\prime}(\mathbf{W_O}\mathbf{h}_m + \mathbf{b_O}) \frac{\partial \mathbf{W_O}\mathbf{h}_m + \mathbf{b_O}}{\partial \mathbf{b_O}} \\ &= \phi^{(s)\prime}(\mathbf{W_O}\mathbf{h}_m + \mathbf{b_O}) \in \mathbb{R}^{N_C} \end{aligned} \tag{B.6}$$

Again, the definitions (13) and (12) were used in (B.5) and (B.6).

The derivatives of the hidden layer can be now derived. As before, the $\frac{\partial L}{\partial [\mathbf{W}_H]_i}$ is calculated:

$$\frac{\partial L}{\partial [\mathbf{W}_H]_i} = \sum_{m=1}^{N} \sum_{l=1}^{N_C} \frac{\partial L}{\partial [\mathbf{y}_m]_l} \frac{\partial [\mathbf{y}_m]_l}{\partial [\mathbf{h}_m]_i} \frac{\partial [\mathbf{h}_m]_i}{\partial [\mathbf{W}_H]_i} \tag{B.7}$$

where

$$\begin{aligned} \frac{\partial [\mathbf{y}_m]_l}{\partial [\mathbf{h}_m]_i} &= \frac{\partial \phi^{(s)}([\mathbf{W}_O]_l^T \mathbf{h}_m + [\mathbf{b}_O]_l)}{\partial [\mathbf{h}_m]_i} \\ &= \phi^{(s)\prime}([\mathbf{W}_O]_l^T \mathbf{h}_m + [\mathbf{b}_O]_l)[\mathbf{W}_O]_{li} \end{aligned} \tag{B.8}$$

and

$$\begin{aligned} \frac{\partial h_{m[i]}}{\partial \mathbf{W_H}_{[i]}} &= \frac{\partial \phi^{(s)}([\mathbf{W}_H]_i^T \mathbf{s}_m + [\mathbf{b}_H]_i)}{\partial [\mathbf{W}_H]_i} \\ &= \phi^{(s)\prime}([\mathbf{W}_H]_i^T \mathbf{s}_m + [\mathbf{b}_H]_i)\mathbf{s}_m \in \mathbb{R}^{N_K} \end{aligned} \tag{B.9}$$

The definitions (12) and (10) where used in (B.8) and (B.9).

Similarly, the $\frac{\partial L}{\partial [\mathbf{b}_H]_i}$ is calculated as:

$$\frac{\partial L}{\partial [\mathbf{b}_H]_i} = \sum_{m=1}^{N} \sum_{l=1}^{N_C} \frac{\partial L}{\partial [\mathbf{y}_m]_l} \frac{\partial [\mathbf{y}_m]_l}{\partial [\mathbf{h}_m]_i} \frac{[\mathbf{h}_m]_i}{\partial [\mathbf{b}_H]_i} \tag{B.10}$$

where

$$\begin{aligned} \frac{\partial [\mathbf{h}_m]_i}{\partial [\mathbf{b}_H]_i} &= \frac{\partial \phi^{(s)}([\mathbf{W_H}]_i^T \mathbf{s}_m + [\mathbf{b}_H]_i}{\partial [\mathbf{b}_H]_i} \\ &= \phi^{(s)\prime}([\mathbf{W_H}]_i^T \mathbf{s}_m + [\mathbf{b}_H]_i) \end{aligned} \tag{B.11}$$

The definition (10) was used in the equation (B.11). The rest of the derivatives needed by (B.10) are defined in (B.2) and (B.6).

The previous gradients must be projected to the Neural BoF Layer. Therefore, the derivative $\frac{\partial L}{\partial [\mathbf{s}_m]_k}$ is also computed as:

$$\frac{\partial L}{\partial [\mathbf{s}_m]_k} = \sum_{l=1}^{N_C} \sum_{i=1}^{N_H} \frac{\partial L}{\partial [\mathbf{y}_m]_l} \frac{\partial [\mathbf{y}_m]_l}{\partial [\mathbf{h}_m]_i} \frac{\partial [\mathbf{h}_m]_i}{\partial [\mathbf{s}_m]_k} \tag{B.12}$$

where

$$\begin{aligned} \frac{\partial [\mathbf{h}_m]_i}{\partial [\mathbf{s}_m]_k} &= \frac{\partial \phi^{(s)}([\mathbf{W_H}]_i^T \mathbf{s}_m + [\mathbf{b}_H]_i)}{\partial [\mathbf{s}_m]_k} \\ &= \phi^{(s)\prime}([\mathbf{W_H}]_i^T \mathbf{s}_m + [\mathbf{b}_H]_i)[\mathbf{W}_H]_{ik} \end{aligned} \tag{B.13}$$

Appendix B.2. Input Layer (Neural BoF Layer) Derivatives

The two Neural BoF derivatives are calculated as follows:

$$\frac{\partial L}{\partial \mathbf{v}_m} = \sum_{i=1}^{N} \sum_{k=1}^{N_K} \frac{\partial L}{\partial [\mathbf{s}_i]_k} \frac{\partial [\mathbf{s}_i]_k}{\partial \mathbf{v}_m} \tag{B.14}$$

and

$$\frac{\partial L}{\partial \mathbf{w}_m} = \sum_{i=1}^{N} \sum_{k=1}^{N_K} \frac{\partial L}{\partial [\mathbf{s}_i]_k} \frac{\partial [\mathbf{s}_i]_k}{\partial \mathbf{w}_m} \tag{B.15}$$

where the iterative definition (9) was used instead of the recurrent definition (7).

To simplify the calculations the following quantity is defined:

$$[\mathbf{d}_{ij}]_k = exp(-||(\mathbf{x}_{ij} - \mathbf{v}_k) \odot \mathbf{w}_k||) \tag{B.16}$$

Now, the output of the RBF layer can be expressed as:

$$[\boldsymbol{\phi}(\mathbf{x}_{ij})]_k = \frac{[\mathbf{d}_{ij}]_k}{||\mathbf{d}_{ij}||_1} \tag{B.17}$$

The derivative that projects the outer layer gradients (MLP) into the RBF centers is computed as:

$$\begin{aligned} \frac{\partial [\mathbf{s}_i]_k}{\partial \mathbf{v}_m} &= \frac{1}{N_i} \sum_{j=1}^{N_i} \frac{\partial [\boldsymbol{\phi}(\mathbf{x}_{ij})]_k}{\partial \mathbf{v}_m} \\ &= \frac{1}{N_i} \sum_{j=1}^{N_i} \sum_{l=1}^{N_k} \frac{\partial [\boldsymbol{\phi}(\mathbf{x}_{ij})]_k}{\partial [\mathbf{d}_{ij}]_l} \frac{\partial [\mathbf{d}_{ij}]_l}{\partial \mathbf{v}_m} \\ &= \frac{1}{N_i} \sum_{j=1}^{N_i} \frac{\partial [\boldsymbol{\phi}(\mathbf{x}_{ij})]_k}{\partial [\mathbf{d}_{ij}]_m} \frac{\partial [\mathbf{d}_{ij}]_m}{\partial \mathbf{v}_m} \end{aligned} \tag{B.18}$$

where

$$\frac{\partial [\boldsymbol{\phi}(\mathbf{x}_{ij})]_k}{\partial [\mathbf{d}_{ij}]_m} = \frac{\delta_{km}}{||\mathbf{d}_{ij}||_1} - \frac{[\mathbf{d}_{ij}]_k}{||\mathbf{d}_{ij}||_1^2} \tag{B.19}$$

and

$$\frac{\partial [\mathbf{d}_{ij}]_m}{\partial \mathbf{v}_m} = [\mathbf{d}_{ij}]_m \frac{(\mathbf{x}_{ij} - \mathbf{v}_m) \odot \mathbf{w}_m}{||(\mathbf{x}_{ij} - \mathbf{v}_m) \odot \mathbf{w}_m||_2} \odot \mathbf{w}_m \tag{B.20}$$

The Kronecker delta function used in (B.19) is defined as:

$$\delta_{km} = \begin{cases} 1 & \text{if } m = k \\ 0 & \text{otherwise} \end{cases} \tag{B.21}$$

Similarly, for the derivative of the RBF input weights:

$$\frac{\partial[\mathbf{s}_i]_k}{\partial \mathbf{w}_m} = \frac{1}{N_i} \sum_{j=1}^{N_i} \frac{\partial[\boldsymbol{\phi}(\mathbf{x}_{ij})]_k}{\partial[\mathbf{d}_{ij}]_m} \frac{\partial[\mathbf{d}_{ij}]_m}{\partial \mathbf{w}_m} \tag{B.22}$$

where

$$\frac{\partial[\mathbf{d}_{ij}]_m}{\partial \mathbf{w}_m} = -[\mathbf{d}_{ij}]_m \frac{(\mathbf{x}_{ij} - \mathbf{v}_m) \odot \mathbf{w}_m}{||(\mathbf{x}_{ij} - \mathbf{v}_m) \odot \mathbf{w}_m||_2} \odot (\mathbf{x}_{ij} - \mathbf{v}_m) \tag{B.23}$$

The derivatives (B.20) and (B.23) do not exist when an RBF center and a feature vector coincide. When that happens, the corresponding derivatives are set to 0.

All the available feature vectors (N_i) were used in the calculation of the derivatives. If the incremental feature streaming approach is used, then the derivatives should be appropriately modified to use only the streamed feature vectors.