Pattern Recognition Letters 170 (2023) 39–47

Contents lists available at ScienceDirect

Pattern Recognition Letters

journal homepage: www.elsevier.com/locate/patrec

Large-margin representation learning for texture classification

Jonathan de Matos[a,d,*], Luiz Eduardo Soares de Oliveira[c], Alceu de Souza Britto Junior[b,d], Alessandro Lameiras Koerich[a]

[a] École de Technologie Supérieure (ÉTS), Université du Québec, 1100 Notre-Dame Street West, Montreal, QC H3C 1K3, Canada
[b] Pontificia Universidade Catolica do Paraná (PUCPR), Rua Imaculada Conceição, 1155, Curitiba, PR 80215-901, Brazil
[c] Universidade Federal do Paraná (UFPR), Rua Coronel Francisco Heraclito dos Santos, 100, Curitiba, PR 81530-900, Brazil
[d] Universidade Estadual de Ponta Grossa (UEPG), Avenida General Carlos Cavalcanti, 4748, Ponta Grossa, PR 84030-900, Brazil

ARTICLE INFO

Article history:
Received 5 May 2022
Revised 3 March 2023
Accepted 17 April 2023
Available online 19 April 2023

Edited by: Jiwen Lu

Keywords:
Fully convolutional networks
Large-margin classifier
Feature extraction
Texture

ABSTRACT

This paper presents a novel approach combining convolutional layers (CLs) and large-margin metric learning for training supervised models on small datasets for texture classification. The core of such an approach is a loss function that computes the distances between instances of interest and support vectors. The objective is to update the weights of CLs iteratively to learn a representation with a large margin between classes. Each iteration results in a large-margin discriminant model represented by support vectors based on such a representation. The advantage of the proposed approach w.r.t. convolutional neural networks (CNNs) is two-fold. First, it allows representation learning with a small amount of data due to the reduced number of parameters compared to an equivalent CNN. Second, it has a low training cost since the backpropagation considers only support vectors. The experimental results on texture and histopathologic image datasets show that the proposed approach achieves competitive accuracy with lower computational cost and faster convergence compared to equivalent CNNs.

1. Introduction

Convolutional neural networks (CNNs) have been established as the state-of-the-art approach to computer vision. CNNs achieve high accuracy due to how the convolutional layers filter images, the number of parameters available to learn representations [1,2], and the large datasets [3] used in their training. They usually present high accuracy in object recognition problems, e.g., cars, people, animals, and digits. Networks pretrained with large datasets of objects can be reused in other contexts. There are two ways of transferring networks between contexts, fine-tuning or using them as feature extractors. The fine-tuning procedure allows fast training with less data because pretrained filters can already identify some patterns. Although it reduces training effort, data may be insufficient even for fine-tuning in small datasets.

Furthermore, when working with datasets where textural information prevails to the detriment of shape and spatial characteristics, the first layers of pretrained CNNs may not respond well to new patterns. There are two additional problems: (i) deeper layers also have more problem-specific knowledge; (ii) there is the need for image size adaptation, which can slow the training [4]. When used as feature extractors, CNN filters are not updated. Instead, the activation maps of a specific layer are used as a feature vector for images of a new context, and an alternative classifier is trained on them. They work similarly to a handcrafted feature extractor and neither adapt to the new patterns nor generate features targeted to facilitate the classification task. Small and simple models like these are more suitable for classifying non-object and small datasets since they do not require data for representation learning but only for training a discriminant.

This paper proposes a novel large-margin representation learning that overcomes most problems related to CNNs and handcrafted feature extractors. For this purpose, it addresses the following questions: a) is it possible to learn representations and discriminants on small-size texture datasets? b) can it speed up the training convergence while achieving high accuracy? The proposed approach uses a short sequence of convolutional layers (CLs) to learn representation for texture classification. The CLs feed a large-margin discriminant that, in turn, provides information to update the CLs' weights and increase the decision margin. A novel loss function calculates the distance between instances in the decision frontier and anchors. The backpropagation algorithm minimizes

* Corresponding author at: Universidade Estadual de Ponta Grossa (UEPG), Avenida General Carlos Cavalcanti, 4748, Ponta Grossa, PR 84030-900, Brazil.
E-mail addresses: jonathan@uepg.br (J. de Matos), luiz.oliveira@ufpr.br (L.E.S. de Oliveira), alceu@ppgia.pucpr.br (A. de Souza Britto Junior), alessandro.koerich@etsmtl.ca (A.L. Koerich).

https://doi.org/10.1016/j.patrec.2023.04.006
0167-8655/

the loss function while enlarging the margin between classes. The novelty of our approach is that it employs only instances that violate the decision margin to train the CLs and produce latent representations. That speeds up the convergence of the backpropagation algorithm to suitable latent representation and discriminant. In addition, it uses fewer parameters than a conventional CNN, allowing training in small datasets, and it performs well on non-object context recognition. Otherwise, the CNNs would still be more effective. Due to using a binary large-margin discriminant, the proposed approach is designed for two-class datasets or scenarios with few classes. However, when the number of classes grows, our method needs multiple binary models generating high computational costs and making other methods, like CNNs or deep metric learning, more suitable. The proposed approach was evaluated on a synthetic dataset based on Gaussian distributions, texture datasets, and three histopathological image (HI) datasets.

The main contributions of this paper are: (i) an approach that trains CLs from scratch with little data; (ii) A representation learning method that adapts itself to the characteristics of different texture datasets; (iii) a computationally efficient training technique that uses only support vectors (SVs) in each iteration instead of all training instances; (iv) A fast convergence method compared to methods used for training conventional CNNs; (v) resilience to imbalanced data; (vi) A detailed comparison of the performance achieved by the proposed approach with approaches using handcrafted features and CNNs.

2. Related works

The first CLs of CNNs are suitable for identifying general and straightforward patterns such as textures. Their training help to make them adaptable to motifs of each context, so it is worthwhile to train these first CLs and use only them in more simple image contexts. One may find different contributions in the literature to adapt pretrained CNN models to a new domain [5]. Cimpoi et al. [6] proposed the Fisher vector CNN (FV-CNN), which pools the last CL of a pretrained network, using it as a feature vector. The pooling allows using input images without resizing them to fit a fully connected CNN (FC-CNN). They used the FV-CNN features as input to an SVM and compared the FC-CNN and SIFT features. The FV-CNN has shown to be a good texture descriptor, performing well on several benchmarks. The simple fully convolutional neural network (SFCN) proposed by Gong et al. [7] uses a shallow, fully convolutional network to predict brain age in MRI images. It has a low computational cost due to weight sharing and performs well in terms of accuracy, as MRI images do not depend on spatial and shape characteristics.

The texture CNN (TCNN) [4] uses a sequence of fully connected (FC) layers instead of a support vector machine (SVM) to classify images. Unlike a traditional FC-CNN, it has a global average pooling (GAP) layer at the end, which acts as an energy layer to the FC layers. This architecture allows the joint training of CLs and the classifier, eliminating the need for a pretrained CNN. This approach can be useful in medical imaging applications, such as MRI, CT, radiography, or microscopy, where textural characteristics are prominent [8], but sample sizes are small. In these scenarios, transfer learning or using the first layers of a CNN as an FCN to extract texture-aware features can simplify the classification problem. The extracted feature vectors can be used with a simpler classifier with fewer parameters than the FC layers of a CNN.

Deep metric learning (DML) can also be used to improve the training of FCNs for better filter and representation performance. DML is a useful alternative for high-dimensional or low-sample data in applications such as face recognition and signature verification [9]. DML seeks to learn a representation that can identify the similarity between samples, usually using a distance metric, such as Euclidean distance. DML combines metric and representation learning, the latter facilitated by deep learning and the training of parameters in its layers, particularly CLs. Chopra et al. [9] presented a method that uses siamese CNNs and an energy-based model. It consists of using raw energy values instead of probabilistic normalized ones. Their concept of energy is analogous to the one of Andrearczyk and Whelan [4]. The advantage of using CNNs is that they provide end-to-end training that learns low and high-level features, resulting in shift-invariant detectors. Their method aggregates the energy output of each siamese network into one neural network trained with contrastive loss. The loss allows training the system together (siamese networks and the single neural network), improving the sample representation on the energy layer for texture classification. DML algorithms can be split into pair-based or proxy-based. The former, like contrastive loss, aims to minimize intra-class distance and maximize inter-class distance. However, this approach has prohibitive computational complexity and pairs that do not contribute to the training. Therefore, several works have addressed these issues [10]. The ranked list loss method [11] relies on a threshold margin that gives more attention, using weighting, to the samples that maximize the margins between opposite classes. The margin determines the negative points too close to the query and violates most of the margin. These are the negative selected points. On the other hand, proxy-based methods [12,13] create an instance representing a set of instances of the same class. It reduces the number of training instances and avoids noise and outliers.

3. Large-margin fully convolutional network (LMFCN)

The proposed approach is pair-based, selecting the most effective instances for the training procedure, reducing the training complexity, and discarding irrelevant instances. It also selects specific examples as anchors to calculate a novel loss function, which speeds up training. Although the proposed approach shares ideas of DML, our application context is different, with more instances per class and fewer classes than the usual DML context.

The proposed method has three components: a fully convolutional network (FCN) with a global average pooling (GAP), a large-margin classifier, and a novel loss function. The LMFCN[1] uses a sequence of CLs acting as a filter bank to learn representation from data used as input to a large-margin classifier. It acts as an end-to-end image classifier, like a CNN but requires less training data to learn high discriminant representations. Its advantage is to enable filter training and make the latent representation more suitable for the classifier. In addition, the backpropagation algorithm trains the filters with a novel loss function that uses the distance between instances of interest and their anchors provided by a large-margin classifier to improve representation learning.

3.1. Training procedure and loss function

The learning algorithm uses the concept of anchors to guide the training. Anchors are instances from the training set used as references by the loss function to calculate the distance to instances of interest. The backpropagation algorithm minimizes such distances during training. Fig. 1 presents a latent space split into two regions by an RBF SVM classifier and the three types of anchors.

Type 1 anchors (red circles and crosses) are the correctly classified instances closest to the support vectors (SVs). The LMFCN attempts to maximize the margin by pushing the SVs toward such anchors. The learning algorithm minimizes the distance between them and the SVs. In Fig. 1(a), the dotted straight lines linking SVs

[1] https://github.com/jonathandematos/lmfcn

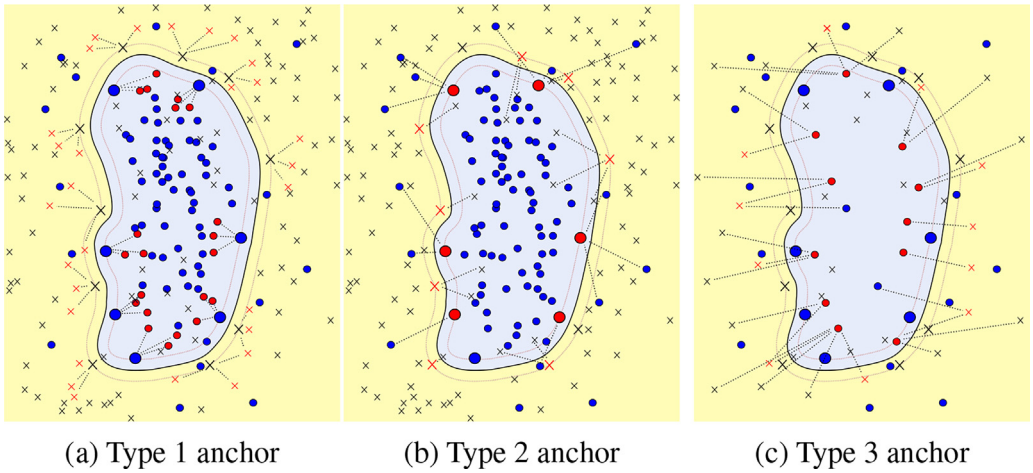

Fig. 1. A two-class latent space with a decision boundary computed by an RBF SVM. Circles are samples of class 0, and crosses, class 1. Bigger symbols mean the SVs for each class. In blue is the region of class 0, and in yellow is class 1. Dashed-straight black lines link the instances to their anchors. In red are the anchors for three different situations on the calculations. (a) Reference anchors to the SVs; (b) Anchors used to move the misclassified instances; (c) Anchors used to increase the separation of instances from opposite classes. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

identify the three anchors of each SV and help visualize the effect of the distance minimization. Type 2 anchors (Fig. 1(b)) are the SVs closest to misclassified examples used to move such instances in the direction of the right side of the decision boundary. Type 3 anchors are the closest correctly classified instances of the opposite class, as shown in Fig. 1(c). Type 3 anchors help to maximize the distance between samples of different classes. The number of anchors, regardless of their type, is a hyperparameter.

When using the three types of anchors, all instances from the training set participate in the backpropagation procedure. However, as further explained, only Type 1 and 2 anchors (two terms of our loss function) are generally used, so in one epoch, only the SVs and the misclassified instances of the training set usually participate in the backpropagation procedure. These instances are determined after obtaining the latent representation of all images by a random initialized FCN and training the large-margin discriminant. Usually, even with an untrained FCN, not all instances become SVs or are misclassified. Thus, some training instances are used at the backpropagation procedure in the first epoch without using the Type 3 anchors.

The training procedure starts with a dataset of size n denoted as $\mathcal{X} = \{\mathbf{X}^t, o^t\}_{t=1}^{n}$, where t indexes images $\mathbf{X}^t$ of width w, height h and c channels in $\mathcal{X}$, and $o^t \in \mathbb{N} : [0, 1]$ is its expected output. All images from $\mathcal{X}$ are fed to CLs denoted as $f_{\text{fcn}}(.)$, which produces a matrix $\mathbf{T}^{n\times\phi}$, with ϕ being the size of the latent representation (step 1 at Fig. 2). The algorithm uses matrix $\mathbf{T}$ to calculate matrix $\mathbf{P}$ (Eq. (1)), which in turn is used to calculate matrices $\mathbf{K}^{n\times n}$ and $\mathbf{D}^{n\times n}$ using Eqs. (2) and (3), respectively (step 2 at Fig. 2). $\mathbf{K}$ is an RBF kernel matrix used to train a large margin classifier $f_{sv}(.)$, which produces the output y^t for each element t of $\mathcal{X}$, and also provides the set of support vectors (SVs) indexes $\boldsymbol{S} = \{s^u\}_{u=1}^{v}$, where $s^u \subset \mathbb{N} : [0, n[$ and v is the number of SVs (step 3 at Fig. 2). $\mathbf{D}$ is essential to the rest of the algorithm as it contains the pairwise distance of all instances and is used to create the anchor

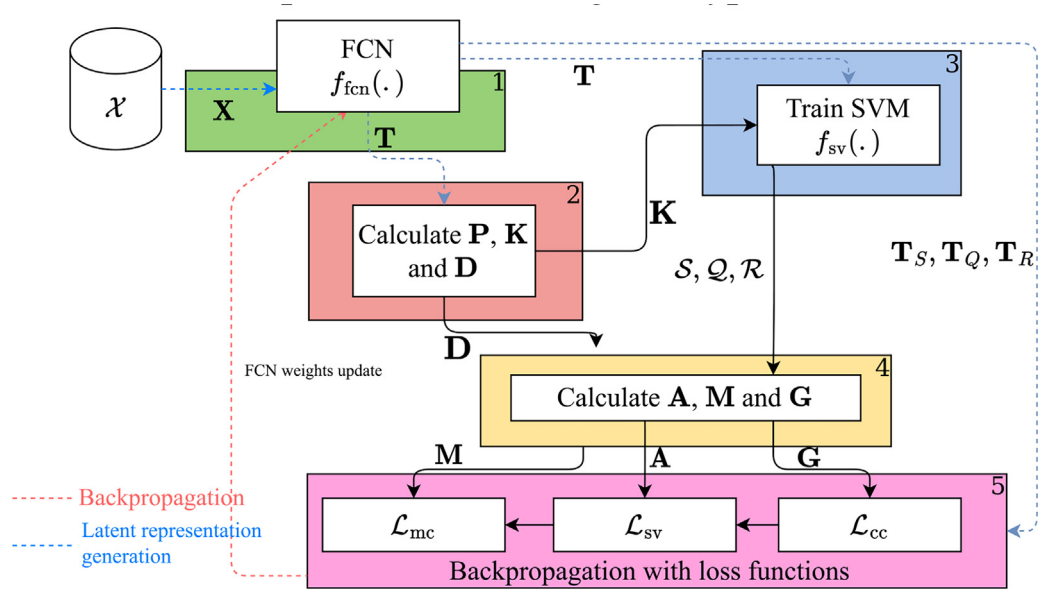

Fig. 2. An overview of the training scheme of the LMFCN.

matrices.

$$p_{ij} = \sum_{b=0}^{\phi} (T_{ib} - T_{jb})^2 \tag{1}$$

$$k_{ij} = \exp(-\gamma\, p_{ij}) \tag{2}$$

$$d_{ij} = \sqrt{p_{ij}} \tag{3}$$

Type 1 anchors (step 4 at Fig. 2) are obtained from matrix **D**, the set $\boldsymbol{S}$, the expected output o^t, and the output from $f_{\text{sv}}(f_{\text{fcn}}(\mathbf{X}^t))$, as shown in Eq. (4). In the first step of the first epoch, the LMFCN generates the matrix **T**, forwarding all training images through a randomly initialized FCN. **T** contains the latent representation of all images that are then used to compute the distance matrices and the large-margin discriminant. Finally, the SVs discovered during the optimization process and the correct and misclassified instances predicted by the discriminant are used to define the anchors of the three types.

$$e_{ij} = \begin{cases} \tau & \text{if } i = j \textbf{ or } j \in \boldsymbol{S} \text{ or } o^i \neq o^j \text{ or } o^j \neq f_{\text{sv}}(f_{\text{fcn}}(\mathbf{X}^j)) \\ d_{ij} & \text{otherwise} \end{cases} \tag{4}$$

where e_{ij} is either the distance between SVs and their anchors (d_{ij}) or a large constant τ that indicates that there is no SV-anchor relation (uninteresting instances), $f_{\text{fcn}}(\mathbf{X}^t)$ denotes the latent representation generated by the FCN for an input $\mathbf{X}^t$, $f_{\text{sv}}(.)$ is the output predicted by the large-margin classifier. The constant τ is defined as the largest distance in **D** multiplied by a large integer (1000), simulating an infinite distance.

Furthermore, we also define the sorting function $f_{\text{argsort}}(.)$ that takes a vector as input and returns a vector of indexes sorted in increasing order. Therefore, the anchor matrix $\mathbf{A}^{s\times n}$ is calculated using Eq. (5). During the calculation of **A**, all values of **D** corresponding to distances of possible anchors of each SV are copied to an auxiliary matrix **E**. Values that are not anchors are replaced by τ. **D** and **E** have the same dimensions with part of values identical and τ replacing uninteresting distances. The return of the $f_{\text{argsort}}(.)$ function over each line of **E** is assigned to each line of **A**, so at the end, it has a list of indexes pointing to elements in **T** (latent representations). For example, taking only the five first values of each line of **A** is the same as taking the indexes of the five closest anchors to an SV. The same procedure applies in calculating **Q** and **R**, but each of their lines relates to indexes in **T** pointing to anchors of Type 2 and 3.

$$\mathbf{a}_u = f_{\text{argsort}}(\mathbf{e}_{s^u}) \text{ with } s^u \in \boldsymbol{S} \text{ and } u \in [0, |S|[\tag{5}$$

Likewise **A**, we calculate with the Eqs. (7) and (9) the matrices $\mathbf{M}^{m\times n}$ and $\mathbf{G}^{g\times n}$ for type 2 and 3 anchors, respectively (step 4 at Fig. 2). We define a set of indexes to the misclassified and correct classified instances from $\mathcal{X}$ as $\boldsymbol{Q} = \{q \mid q \subset \mathbb{N} : [0, n[\wedge f_{\text{sv}}(f_{\text{fcn}}(\mathbf{X}^q)) \neq o^q\}$, and $\boldsymbol{R} = \{r \mid r \subset \mathbb{N} : [0, n[\wedge r \notin \boldsymbol{S} \cup \boldsymbol{Q}\}$, respectively.

$$z_{ij} = \begin{cases} d_{ij} & \text{if } i \in \boldsymbol{Q} \text{ and } j \in \boldsymbol{S} \text{ and } i \neq j \\ \tau & \text{otherwise} \end{cases} \tag{6}$$

$$\mathbf{m}_i = f_{\text{argsort}}(\mathbf{z}_{q^i}) \text{ with } q^i \in \boldsymbol{Q} \text{ and } i \in [0, |\boldsymbol{Q}|[\tag{7}$$

$$h_{ij} = \begin{cases} d_{ij} & \text{if } i \in \boldsymbol{R} \text{ and } j \in \boldsymbol{R} \text{ and } o^i \neq o^j \\ \tau & \text{otherwise} \end{cases} \tag{8}$$

$$\mathbf{g}_i = f_{\text{argsort}}(\mathbf{h}_{r^i}) \text{ with } r^i \in \boldsymbol{R} \text{ and } i \in [0, |\boldsymbol{R}|[\tag{9}$$

where z_{ij} and h_{ij} in Eqs. (6) and (8) are the distance between misclassified instances and the SVs (d_{ij}) and the distance between correct classified instances from opposite classes (d_{ij}), respectively. Again, τ is a large constant that indicates uninteresting instances.

Step 5 of the algorithm is the weight updating of the FCN by the backpropagation algorithm using the loss function with the anchors and the images from $\mathcal{X}^{\mathbf{S}}$, $\mathcal{X}^{\mathbf{Q}}$, and $\mathcal{X}^{\mathbf{R}}$, which contain the images corresponding to the SVs, the misclassified, and the correct classified images. They are fed to the FCN to calculate the gradients and calculate the loss functions to update the CL weights. Steps 1 to 5 define one epoch. After step 5, the process restarts from step 1, so the latent representation is computed again, as though the matrices **P**, **K** and **D** and the large-margin discriminant is retrained on such an updated latent representation, producing another set of SVs allowing recalculation of matrices **A**, **M** and **G**. Therefore, the elements of matrix **T** differ from the previous epoch because of the updated FCN weights.

The proposed loss function relies on the similarity between examples, and it has three terms, as shown in Eq. (10). It aims at finding a latent representation that maximizes the margin ($\mathcal{L}_{\text{sv}}$) while pushing misclassified examples towards the right side of the decision boundary ($\mathcal{L}_{\text{mc}}$) and moving well-classified examples farther away from the decision boundary ($\mathcal{L}_{\text{cc}}$). Although the sum of the loss function terms is not explicitly weighted in Eq. (10), the number of SVs and the number of misclassified and correctly classified instances work as an implicit weighting mechanism in Eqs. (11)–(13).

$$\mathcal{L} = \mathcal{L}_{\text{sv}} + \mathcal{L}_{\text{mc}} - \mathcal{L}_{\text{cc}} \tag{10}$$

$\mathcal{L}_{\text{sv}}$ calculates the sum of distances between SVs and their anchors using **A**, as shown in Eq. (11). Consequently, the gradients are affected only by the SV instances, not by the Type 1 anchors, as they were already generated and stored in **T**. The backpropagation procedure updates the weights in a way that the latent representation generated by $f_{\text{fcn}}(.)$ has the smallest possible distance to the fixed values of Type 1 anchors at the current epoch. Therefore, the weights are updated to move the SVs and not the anchors.

$$\mathcal{L}_{\text{sv}} = \frac{1}{|\boldsymbol{S}|} \sum_{i=0}^{|\mathcal{S}|} \sum_{j=1}^{sv_{\text{close}}} [f_{\text{fcn}}(\mathbf{X}^{s^i}) - \mathbf{t}_{a_{ij}}]^2 \tag{11}$$

where sv_{close} is the number of anchors to use for each SV, $f_{\text{fcn}}(.)$ is the FCN updated by the backpropagation procedure, $\mathbf{X}^{s^i}$ is the matrix that represents the s^i image from $\mathcal{X}$, a_{ij} is an index pointing to an anchor instance in the input set $\mathcal{X}$, $\mathbf{t}_{a_{ij}}$ is the latent representation of an image $\mathbf{X}^{a_{ij}}$, and $|\boldsymbol{S}|$ is the number of SVs.

The training algorithm computes the loss over the entire set of SVs as a single batch. It is also possible to use mini-batches, but considering small-size datasets and an FCN with compact architecture that yields a low-dimensional latent representation, this is unnecessary. In the next epoch, the updated weights will affect the generation of the latent representation of the entire dataset. Therefore, the latent representation of anchors also changes, and the training algorithm builds a new set of anchors.

$\mathcal{L}_{\text{mc}}$ calculates the summation of distances between misclassified instances and their anchors using matrix **M**, as shown in Eq. (12). We also use all misclassified instances as a single batch, although there is no limitation to performing it in mini-batches. The influence of $\mathcal{L}_{\text{mc}}$ in the training algorithm is the weight updating that minimizes the distance between the misclassified instances and their closest SVs. Consequently, the misclassified instances are pushed towards the right side of the decision boundary.

$$\mathcal{L}_{\text{mc}} = \frac{1}{|\boldsymbol{Q}|} \sum_{i=0}^{|\boldsymbol{Q}|} \sum_{j=0}^{wr_{\text{close}}} [f_{\text{fcn}}(\mathbf{X}^{q^i}) - \mathbf{t}_{m_{ij}}]^2 \tag{12}$$

Table 1
The architecture of the FCN (w: width, h: height, c: number of channels, ϕ: dimension of the latent representation).

Layer	Input	Output
Convolutional Layer	$w \times h \times c$	$w \times h \times 64$
Max Pooling	$w \times h \times 64$	$w/2 \times h/2 \times 64$
Convolutional Layer	$w/2 \times h/2 \times 64$	$w/2 \times h/2 \times 128$
Max Pooling	$w/2 \times h/2 \times 128$	$w/4 \times h/4 \times 128$
Convolutional Layer	$w/4 \times h/4 \times 128$	$w/4 \times h/4 \times \phi$
Global Average Pooling	$w/4 \times h/4 \times \phi$	$1 \times 1 \times \phi$

Batch Normalization and ReLU after each CL.

where wr_{close} is the number of anchors for each misclassified instance, $f_{\text{fcn}}(.)$ is the FCN updated by the backpropagation, $\mathbf{X}^{q^i}$ is the matrix that represents the q^i image from $\mathcal{X}$, m^{ij} is an index pointing to an anchor instance in the input set $\mathcal{X}$, $\mathbf{t}_{m_{ij}}$ is the latent representation of an image $\mathbf{X}^{m_{ij}}$, and $|\mathbf{Q}|$ is the number of misclassified instances.

When looking at only a single misclassified instance, the weight updating may not be enough to move such an instance to the right side of the decision boundary because its anchors are SVs right at the margin of the decision boundary. Despite that, the misclassified example can become an SV in the following training epoch.

$\mathcal{L}_{\text{cc}}$ represents the distance between well-classified instances and their anchors. Since we want to maximize such a distance, computed by Eq. (13), it is incorporated to the loss function as a negative term, which is minimized during training.

$$\mathcal{L}_{\text{cc}} = \frac{1}{|\boldsymbol{R}|} \sum_{i=0}^{|\boldsymbol{R}|} \sum_{j=0}^{sh_{\text{close}}} [f_{\text{fcn}}(\mathbf{X}^{r^i}) - \mathbf{t}_{g_{ij}}]^2 \tag{13}$$

where sh_{close} is the number of anchors for each correctly classified instance, $f_{\text{fcn}}(.)$ is the FCN updated by the backpropagation, $\mathbf{X}^{r^i}$ is the matrix that represents the r^i image from $\mathcal{X}$, g^{ij} is an index pointing to an anchor instance in the input set $\mathcal{X}$, $\mathbf{t}_{g_{ij}}$ is the latent representation of an image $\mathbf{X}^{g_{ij}}$, and $|\boldsymbol{R}|$ is the number of correct classified instances.

The number of anchors used for each training instance is controlled by sv_{close}, wr_{close}, and sh_{close} in Eqs. (11)–(13), respectively. Usually, complex classification problems require a higher number of anchors. Furthermore, using all three terms of the proposed loss function in the training process may not always be necessary. $\mathcal{L}_{\text{sv}}$ leads to good representations as such a loss is directly related to SVs and margin maximization. The computational cost for computing this loss term is not high and decreases as the number of instances used by the backpropagation algorithm reduce at each training epoch. Computing $\mathcal{L}_{\text{mc}}$ is also not expensive because the number of incorrectly classified examples tends to decrease over the training epochs. On the other hand, computing $\mathcal{L}_{\text{cc}}$ can become very expensive because the number of well-classified instances tends to increase over the training epochs. Therefore, the term $\mathcal{L}_{\text{cc}}$ should be used wisely, preferably only on challenging problems where just the other two terms of the proposed loss function may not lead to a low training error.

After the loss calculation and weight updating, a model is retained if the validation accuracy shows improvement. The entire process from step one to step five in Fig. 2 repeats for a user-defined number of epochs, and after it, the final model is the one with the best validation accuracy.

3.2. FCN architecture

The FCN is a sequence of CLs similar to ones used in CNNs, used as filter banks to learn representation related to textures, as presented in Table 1. Pooling layers follow these layers to reduce the input size progressively. The deeper the layers, the narrower the latent representation, but with an increasing number of channels, which allows more filters combination, increasing the complexity of the representation. At the end of the CLs, a GAP layer builds a latent representation where each element represents the response of a filter combination to a texture, measuring how much it happened and not its position on the image. The GAP layer also makes the output dimension independent of the input size, and the latent representation will always have the same number of channels at the end of the FCN. Such a latent representation feeds a large-margin discriminant to learn classification tasks. We compared our approach with two CNNs with identical architecture (Table 1), but with a sequence of FC layers as discriminant after the GAP layer. We employed binary cross-entropy (BCE) loss and hinge loss for binary problems and cross-entropy loss in multiclass problems.

The large-margin classifier of the LMFCN is a support vector machine (SVM) with an RBF kernel, which Gram matrix $\mathbf{K}$ is obtained by Eq. (2) calculated jointly with the distance matrix $\mathbf{D}$. Part of the classifier calculation is reused and performed in GPU. We chose the precomputed RBF kernel due to its ease of computation using the GPU resources and space separation capacity. Although a linear kernel has reduced computational cost, it would require more training of the FCN weights to provide features with more class separation. In the preliminary studies, we compared the two kernel approaches and verified the advantage of the RBF.

A large-margin discriminant is inherently binary, and to deal with multiclass problems, we adopted the one-vs-all (OVA) approach, which reduces multiclass problems into multiple binary classification problems. In the training stage, one provides all instances to all n_{c} pairs of LMFCNs. After training all the n_{c} LMFCNs, we discard the discriminants, keeping only the FCNs. Then, we train a new multiclass SVM. The new classifier is trained using a latent representation with $\phi \times n_{\text{c}}$, which is the concatenation of all FCN latent representations. At the end of the training process, the model comprises n_{c} FCNs with an ϕ-dimensional output and a multiclass SVM with a latent representation of $\phi \times n_{\text{c}}$ dimensions. The multiclass SVM holds internally multiple binary SVMs with RBF kernel on OVA configuration. Although this approach seems similar to using multiple SVMs trained with the FCNs, the new classifiers have access to a latent representation that is better fitted for them.

The computational cost for training the LMFCN and equivalent CNNs up to the GAP layer is proportional to the number and resolution of input images ($n \times w \times h \times c$) and the number weights (α_{fcn}). The LMFCN replaces the FC layers (computational cost of $\alpha_{\text{fc}} \times n$) with an SVM, which requires kernel calculation (n^2), and sequential minimal optimization (SMO) algorithm, which requires n^3 in the worst case. Assuming a small training set, the SMO cost is lower than $n \times \alpha_{\text{fcn}}$. However, for large datasets, where n^2 is greater than α_{fcn}, the problem becomes more suitable for conventional CNN architectures.

The main advantage of the LMFCN is using only the SVs in the backpropagation, which reduces its computational cost from n to $|\boldsymbol{S}|$. The CLs used on both LMFCN and conventional CNNs have several trainable parameters. Therefore, having only SVs as training instances reduces the number of training instances and speeds up the training process, making the loss function converge faster within a few epochs.

4. Experimental results and discussion

The proposed LMFCN is evaluated on images with texture characteristics and low training data availability. We used a synthetic dataset of images generated from two Gaussian distributions with striped patterns, the Salzburg Texture Image Database [14] (misc

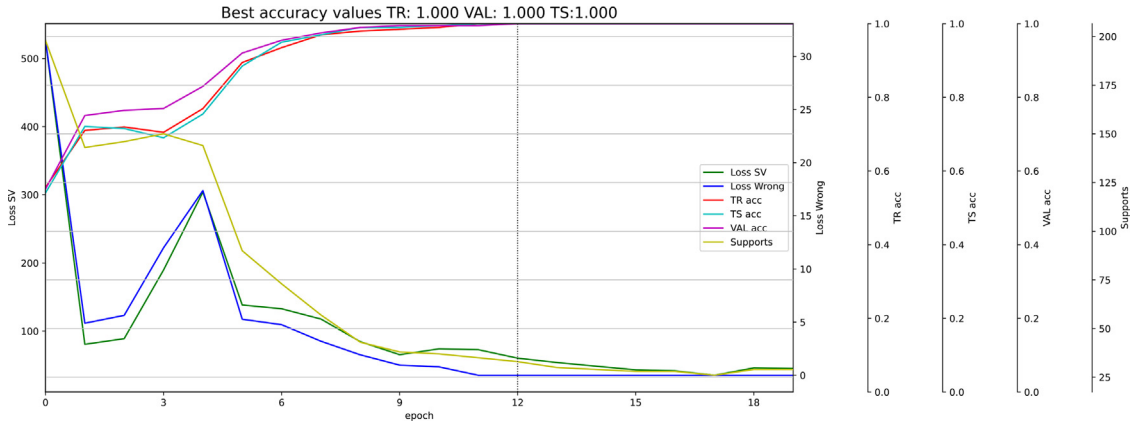

Fig. 3. Learning curves of the LMFCN with the Gaussian image dataset and a 2-dimensional latent representation. Parameters: $sv_{\text{close}} = 5$, $wr_{\text{close}} = 1$, and $sh_{\text{close}} = 0$.

and fabric categories), and three datasets of histopathological images (HIs): BreaKHis [15], BACH [16], and CRC [17].

Figure 3 shows the learning curves of the LMFCN with the Gaussian image dataset over 20 epochs. The best-balanced accuracy values for training, validation, and test sets correspond to the peak validation accuracy. It is possible to observe that the value of losses and the number of SVs decrease over the epochs, and the accuracy rises. That shows that the SVs are getting closer to their instances due to the loss reduction. In addition, the decrease in the number of SVs indicates that better separability is being achieved.

Table 2 compares the results of the LMFCN with other CNN architectures on five datasets. First, we compared with two additional LMFCNs, one with residual blocks (LMFCN-res) and one with inception blocks (LMFCN-inc) [18], and two CNNs with architectures similar to the LMFCN, one trained with hinge loss (CNN-H) and one trained with binary cross-entropy loss (CNN-CE). These two CNNs are equivalent to TCNNs [4] since they use a short sequence of CLs and a GAP to produce energy values. We limited the number of training epochs to 100 and 20 for the CNNs and LMFCN, respectively. We also compared the LMFCN with pretrained ResNet18 and InceptionV3, used as feature extractors, and an SVM with RBF kernel as a discriminant. Furthermore, we used PCA to reduce the feature vectors generated by these two CNNs from 512 and 2048 to 16, the same size used by the other architectures. Finally, we compared our approach with pretrained CNNs fine-tuned on the target datasets. The ResNet18 FT, DenseNet FT, and SqueezeNet FT had their first layers frozen and training only two additional FC layers at their end to adapt their output to the target datasets. We also allowed them to train for 100 epochs. Overall, the LMFCN converges to a latent representation that generalizes well much faster (less than ten epochs for all datasets) than different CNN architectures. Furthermore, in 2 out of 5 datasets, the accuracy achieved by the LMFCN on the test sets is higher than that achieved by other CNNs. For the three other datasets, the difference in accuracy is almost negligible.

Table 3 shows the average results of balanced accuracy of five repetitions comparing the results achieved by the LMFCN with shallow approaches that employ an SVM and three handcrafted feature extractors: Local Binary Pattern (LBP), Gray Level Co-occurrence Matrix (GLCM), and Parameter Free Threshold Adjacency Statistics (PFTAS) [15]. The LMFCN uses a 59-dimensional latent representation, the same dimension as the smallest feature vector, obtained with the LBP with uniform patterns for a fair comparison. PFTAS and GLCM produce 162- and 169-dimensional feature vectors, respectively. PFTAS achieved the best accuracy among the handcrafted feature extractors on three datasets (BACH, BreaKHis, and CRC). LBP achieved the best accuracy on Salzburg and GLCM on the Gaussian dataset. However, the LMFCN with a 59-dimensional latent representation achieves higher accuracy than all shallow methods on all datasets, indicating its adaptability to different problems and datasets.

The multiclass experiments were carried out on three HI datasets and included comparisons against handcrafted feature extractors GLCM, LBP, and PFTAS, as though as ResNet18 and InceptionV3 as feature extractors. We also used a CNN with cross-entropy loss (CE) and similar architecture to the LMFCN but using more filters. For a fair comparison, the number of filters is proportional to the total number of parameters summing all OVA models of the LMFCN. Finally, we used a ResNet18, a DenseNet, and a SqueezeNet, adding two more FC layers as in the CNN-CE and freezing the first layers during the fine-tuning.

Table 4 shows the average balanced accuracy for all evaluated methods. It is noticeable the superior performance of the LMFCN over all others. In the case of the CNNs, we allowed the training procedure to extend over 400 epochs. Although the LMFCN uses multiple models, we let each one only train for ten epochs. The balanced accuracy helps identify problems in imbalanced datasets. Our approach presented a promising performance on the BreaKHis dataset, which significantly differs between some classes, e.g., ductal carcinoma and phyllodes tumor. This result shows that the LMFCN performed well in imbalanced scenarios. We also noticed that the LMFCN performed well on the imbalanced OVA subproblems.

4.1. Computation complexity

Table 5 compares the running time[2] of a CNN and the LMFCN to complete ten training epochs for BreaKHis and BACH datasets, the two most challenging ones with different characteristics. The BreaKHis dataset has more but smaller images, while the BACH dataset has fewer images but bigger ones. Table 6 shows that the CNN took more epochs to achieve the best validation accuracy than the LMFCN. There was more difference for the BACH dataset due to the fewer images than BreaKHis. Fewer images reduce the SMO and matrices calculation time, as shown in Table 7. The LMFCN complexity is related to the number of instances and takes advantage of its fast convergence.

The LMFCN reduces the number of SVs over the training epochs. Fewer SVs imply that the computational effort reduces, as the first

[2] Intel Xeon E5-2620 processor, 64 GB RAM, Tesla P100 GPU with 12 GB of VRAM, Ubuntu 20.04.4 LTS, Python 3.6.13, and Pytorch 1.2.0+cu92.

Table 2
Average balanced accuracy and standard deviation for the LMFCN and equivalent CNN architectures. Epoch refers to the training epoch where the best validation accuracy was achieved. NA: Not applicable for models used as feature extractors.

Dataset	Architecture	Training	Validation	Test	Epoch
BACH	LMFCN	0.8811 ± 0.0426	0.7857 ± 0.0221	0.8056 ± 0.0203	6
	LMFCN-res	0.8845 ± 0.0264	0.7323 ± 0.0388	0.7570 ± 0.0417	18
	LMFCN-inc	0.8750 ± 0.0451	0.7759 ± 0.0531	0.7773 ± 0.0392	3
	CNN-CE	0.8926 ± 0.0135	0.8200 ± 0.0291	0.7864 ± 0.0212	88
	CNN-H	0.8534 ± 0.0410	0.8038 ± 0.0261	0.7838 ± 0.0242	68
	ResNet18	0.9981 ± 0.0026	0.7596 ± 0.0122	0.7584 ± 0.0270	NA
	InceptionV3	0.6295 ± 0.0746	0.5924 ± 0.0625	0.6049 ± 0.0630	NA
	ResNet18 FT	0.9184 ± 0.0158	0.8198 ± 0.0281	0.7812 ± 0.0251	10
	DenseNet FT	0.9740 ± 0.0243	0.7782 ± 0.0334	0.7906 ± 0.0302	17
	SqueezeNet FT	0.6690 ± 0.0677	0.7058 ± 0.0538	0.6512 ± 0.0125	37
BreaKHis	LMFCN	0.9882 ± 0.0064	0.9442 ± 0.0137	0.8942 ± 0.0372	5
	LMFCN-res	0.9175 ± 0.0089	0.9040 ± 0.0185	0.8814 ± 0.0071	9
	LMFCN-inc	0.9252 ± 0.0091	0.8953 ± 0.0349	0.8876 ± 0.0272	2
	CNN-CE	0.8926 ± 0.0092	0.9122 ± 0.0226	0.8926 ± 0.0140	91
	CNN-H	0.8560 ± 0.0041	0.8856 ± 0.0166	0.8638 ± 0.0042	91
	ResNet18	0.9777 ± 0.0062	0.8034 ± 0.0120	0.8262 ± 0.0122	NA
	InceptionV3	0.6058 ± 0.0207	0.6064 ± 0.0209	0.5949 ± 0.0207	NA
	Resnet18 FT	0.9154 ± 0.0342	0.8890 ± 0.0190	0.8640 ± 0.0143	37
	DenseNet FT	0.8776 ± 0.0738	0.8900 ± 0.0171	0.8640 ± 0.0150	25
	SqueezeNet FT	0.4996 ± 0.0089	0.5174 ± 0.0149	0.5108 ± 0.0175	23
CRC	LMFCN	0.9924 ± 0.0046	0.9914 ± 0.0024	0.9914 ± 0.0060	6
	LMFCN-res	0.9912 ± 0.0029	0.9862 ± 0.0088	0.9873 ± 0.0086	19
	LMFCN-inc	0.9906 ± 0.0041	0.9903 ± 0.0045	0.9900 ± 0.0068	17
	CNN-CE	0.9828 ± 0.0070	0.9934 ± 0.0025	0.9880 ± 0.0111	31
	CNN-H	0.9928 ± 0.0038	0.9924 ± 0.0027	0.9928 ± 0.0070	30
	ResNet18	0.9991 ± 0.0013	0.9683 ± 0.0081	0.9680 ± 0.0150	NA
	InceptionV3	0.7718 ± 0.0352	0.7670 ± 0.0375	0.7842 ± 0.0355	NA
	ResNet18 FT	0.9936 ± 0.0094	0.9788 ± 0.0165	0.9814 ± 0.0112	42
	DenseNet FT	0.9906 ± 0.0094	0.9884 ± 0.0086	0.9862 ± 0.0081	34
	SqueezeNet FT	0.9104 ± 0.0123	0.9258 ± 0.0162	0.9306 ± 0.0098	41
Salzburg	LMFCN	0.9842 ± 0.0049	0.9446 ± 0.0083	0.9230 ± 0.0081	8
	LMFCN-res	0.8729 ± 0.0187	0.8411 ± 0.0184	0.8362 ± 0.0123	9
	LMFCN-inc	0.8877 ± 0.0174	0.8666 ± 0.0301	0.8450 ± 0.0214	3
	CNN-CE	0.8966 ± 0.0103	0.8762 ± 0.0100	0.8602 ± 0.0115	91
	CNN-H	0.8258 ± 0.0257	0.8292 ± 0.0277	0.8046 ± 0.0314	84
	ResNet18	0.9946 ± 0.0033	0.9046 ± 0.0221	0.8932 ± 0.0088	NA
	InceptionV3	0.6680 ± 0.0332	0.6607 ± 0.0323	0.6644 ± 0.0473	NA
	ResNet18 FT	0.9744 ± 0.0104	0.9494 ± 0.0188	0.9470 ± 0.0130	45
	DenseNet FT	0.9792 ± 0.0170	0.9474 ± 0.0173	0.9460 ± 0.0132	38
	SqueezeNet FT	0.5712 ± 0.0808	0.7332 ± 0.0169	0.7242 ± 0.0266	28
Gaussian	LMFCN	1.0000 ± 0.0000	1.0000 ± 0.0000	0.9990 ± 0.0022	2
	LMFCN-res	1.0000 ± 0.0000	0.9970 ± 0.0067	0.9708 ± 0.0304	15
	LMFCN-inc	1.0000 ± 0.0000	1.0000 ± 0.0000	0.9586 ± 0.0472	9
	CNN-CE	0.9950 ± 0.0061	1.0000 ± 0.0000	1.0000 ± 0.0000	72
	CNN-H	1.0000 ± 0.0000	1.0000 ± 0.0000	1.0000 ± 0.0000	56
	ResNet18	0.9740 ± 0.0109	0.6126 ± 0.0198	0.6176 ± 0.0257	NA
	InceptionV3	0.5102 ± 0.0235	0.5311 ± 0.0185	0.5021 ± 0.0258	NA
	ResNet18 FT	0.9700 ± 0.0050	0.9310 ± 0.0129	0.9210 ± 0.0156	19
	DenseNet FT	0.9960 ± 0.0089	0.8958 ± 0.0175	0.8802 ± 0.0167	33
	SqueezeNet FT	0.5046 ± 0.0071	0.5236 ± 0.0370	0.5188 ± 0.0310	38

term of the loss function depends on the number of SVs. Compared to equivalent CNNs, the LMFCN has an extra cost related to the computation of kernel **K**, distance matrix **D**, and the quadratic optimization problem solved by the SMO algorithm, which has computational complexities $\mathcal{O}(n^2)$, $\mathcal{O}(n^2)$, and $\mathcal{O}(n^3)$, respectively. CNNs have a training complexity proportional to the number of instances and weights. Therefore, compared to the LMFCN, CNNs have more parameters, use more instances in the backpropagation algorithm, and take more epochs to converge.

4.2. Discussion

The experimental results have shown that the LMFCN outperforms all other methods in a scenario composed of textural images and small-size datasets. Besides achieving higher accuracy, the proposed method has several advantages over shallow and deep methods. The LMFCN approach requires little data to properly train a sequence of CLs and a large-margin discriminant. In the experiments comparing the LMFCN with equivalent CNNs, using a 16-dimensional latent representation, the LMFCN achieved training stability and high accuracy within 20 training epochs, while the CNNs needed 100 epochs to achieve comparable performance.

The latent representation learned by the LMFCN, constrained to a dimensionality similar to shallow methods (59- and 162-dimensional), is more discriminant than LBP, PFTAS, and GLCM. As a result, the LMFCN achieved higher balanced accuracy than the compared methods. Furthermore, the LMFCN obtained a competitive accuracy even with a 16-dimensional latent representation. Moreover, the improvement achieved by increasing the latent

Table 3
Average balanced accuracy and standard deviation for LMFCN, GLCM, LBP, and PFTAS. The best results on each dataset are underlined.

Dataset	Method	Training	Validation	Test
BACH	LMFCN-59	0.9664 ± 0.0751	0.7888 ± 0.0577	0.7684 ± 0.0435
	LMFCN-162	1.0000 ± 0.0000	0.8068 ± 0.0600	0.7934 ± 0.0536
	GLCM	0.7346 ± 0.0199	0.6309 ± 0.0346	0.6618 ± 0.0711
	LBP	0.9146 ± 0.0154	0.7455 ± 0.0442	0.7005 ± 0.0228
	PFTAS	0.9899 ± 0.0037	0.7372 ± 0.0634	0.7608 ± 0.0361
BreaKHis	LMFCN-59	1.0000 ± 0.0000	0.9639 ± 0.0247	0.9476 ± 0.0075
	LMFCN-162	1.0000 ± 0.0000	0.9666 ± 0.0183	0.9595 ± 0.0050
	GLCM	0.8128 ± 0.0086	0.8117 ± 0.0375	0.8076 ± 0.0071
	LBP	0.9292 ± 0.0059	0.7998 ± 0.0099	0.7925 ± 0.0098
	PFTAS	0.9801 ± 0.0034	0.9237 ± 0.0236	0.9145 ± 0.0168
CRC	LMFCN-59	0.9977 ± 0.0022	0.9937 ± 0.0023	0.9883 ± 0.0084
	GLCM	0.9864 ± 0.0036	0.9828 ± 0.0055	0.9853 ± 0.0101
	LBP	0.9974 ± 0.0012	0.9431 ± 0.0099	0.9479 ± 0.0061
	PFTAS	1.0000 ± 0.0000	0.9852 ± 0.0025	0.9872 ± 0.0097
Salzburg	LMFCN-59	0.9994 ± 0.0006	0.9672 ± 0.0112	0.9500 ± 0.0180
	GLCM	0.7706 ± 0.0163	0.7257 ± 0.0186	0.7338 ± 0.0142
	LBP	0.9987 ± 0.0012	0.9282 ± 0.0209	0.9116 ± 0.0208
	PFTAS	0.9650 ± 0.0051	0.9096 ± 0.0124	0.8973 ± 0.0137
Gaussian	LMFCN-59	1.0000 ± 0.0000	0.9354 ± 0.1254	0.9880 ± 0.0121
	GLCM	0.9583 ± 0.0028	0.9327 ± 0.0110	0.9527 ± 0.0191
	LBP	0.6303 ± 0.0091	0.6710 ± 0.0131	0.6109 ± 0.0075
	PFTAS	0.5578 ± 0.0080	0.5953 ± 0.0077	0.5608 ± 0.0076

Table 4
Average balanced accuracy and standard deviation on three HI datasets for seven methods considering a multiclass scenario.

Dataset	Method	Training	Validation	Test
BACH	LMFCN	0.9690 ± 0.0348	0.7170 ± 0.0210	0.6854 ± 0.0278
	CNN-CE	0.8390 ± 0.0184	0.6788 ± 0.0406	0.6444 ± 0.0187
	GLCM	0.4081 ± 0.0183	0.4003 ± 0.0753	0.3637 ± 0.0303
	LBP	0.4643 ± 0.0334	0.4871 ± 0.0729	0.3951 ± 0.0896
	PFTAS	0.6229 ± 0.0146	0.5690 ± 0.0464	0.6043 ± 0.0235
	ResNet18	1.0000 ± 0.0000	0.6056 ± 0.0578	0.5671 ± 0.0300
	InceptionV3	0.4628 ± 0.0807	0.4269 ± 0.0904	0.4057 ± 0.0261
	ResNet18 FT	0.6721 ± 0.0855	0.5513 ± 0.0078	0.4668 ± 0.0575
	DenseNet FT	0.9524 ± 0.0843	0.6220 ± 0.0453	0.4842 ± 0.0502
	SqueezeNet FT	0.5692 ± 0.1180	0.6238 ± 0.0676	0.5462 ± 0.0681
BreaKHis	LMFCN	0.9688 ± 0.0285	0.8125 ± 0.0321	0.7895 ± 0.0162
	CNN-CE	0.8032 ± 0.0319	0.7347 ± 0.0238	0.7040 ± 0.0307
	GLCM	0.4697 ± 0.0111	0.4218 ± 0.0212	0.4091 ± 0.0050
	LBP	0.9235 ± 0.0084	0.5373 ± 0.0174	0.5218 ± 0.0147
	PFTAS	0.9537 ± 0.0086	0.6643 ± 0.0109	0.6768 ± 0.0147
	ResNet18	1.0000 ± 0.0000	0.5933 ± 0.0341	0.5834 ± 0.0213
	InceptionV3	0.1848 ± 0.0182	0.1745 ± 0.0194	0.1767 ± 0.0123
	ResNet18 FT	0.3496 ± 0.0633	0.4055 ± 0.0343	0.3546 ± 0.0182
	DenseNet FT	0.8582 ± 0.1184	0.5561 ± 0.0475	0.5194 ± 0.0655
	SqueezeNet FT	0.3038 ± 0.0652	0.3689 ± 0.0108	0.3589 ± 0.0186
CRC	LMFCN	0.9834 ± 0.0127	0.9379 ± 0.0065	0.9338 ± 0.0073
	CNN-CE	0.7209 ± 0.2614	0.3946 ± 0.0453	0.3901 ± 0.0402
	GLCM	0.6062 ± 0.0057	0.5960 ± 0.0069	0.6049 ± 0.0084
	LBP	0.6148 ± 0.0045	0.6086 ± 0.0270	0.6144 ± 0.0123
	PFTAS	0.8359 ± 0.0058	0.8271 ± 0.0152	0.8297 ± 0.0097
	ResNet18	0.9506 ± 0.0033	0.5070 ± 0.0141	0.5066 ± 0.0138
	InceptionV3	0.1509 ± 0.0036	0.1546 ± 0.0075	0.1477 ± 0.0082
	ResNet18 FT	0.7285 ± 0.0189	0.7530 ± 0.0155	0.7460 ± 0.0115
	DenseNet FT	0.9955 ± 0.0063	0.9032 ± 0.0054	0.8933 ± 0.0037
	SqueezeNet FT	0.8233 ± 0.0163	0.8126 ± 0.0190	0.8041 ± 0.0112

Table 5
Time in seconds to complete ten training epochs .

Dataset	Method	Folds				
		1	2	3	4	5
BACH	CNN	30.74	32.96	31.57	32.94	33.14
	LMFCN	33.66	34.57	31.17	31.20	28.81
BreaKHis	CNN	65.74	64.55	63.22	64.42	68.39
	LMFCN	43.65	43.16	43.36	39.41	42.44

representation to 59 dimensions is meaningful, with a slight gain when increasing it to 169 dimensions.

Data imbalance is burdensome for training machine learning algorithms. However, the LMFCN deals well with imbalanced classes in two-class and multiclass scenarios, achieving competitive performance even on highly imbalanced OVA subproblems. Furthermore, the computational effort is not extremely high, given the reduced number of epochs to train the model at each subproblem.

Table 6
Time in seconds to achieve the best validation accuracy .

Dataset	Method	Folds				
		1	2	3	4	5
BACH	CNN	263.81	281.81	247.05	251.72	285.25
	LMFCN	11.28	21.91	16.97	22.58	28.81
BreaKHis	CNN	605.56	570.43	547.856	565.95	633.45
	LMFCN	31.20	18.50	22.31	32.15	18.07

Table 7
Epoch time breakdown of the LMFCN (in milliseconds) .

Dataset	Epoch	FCN	Distance and Matrices	SMO	Backpropagation
BreaKHis	1	1398.18	5.68	31.08	5530.60
	5	1464.87	5.61	100.19	2622.38
	10	1445.52	5.60	104.30	2490.75
BACH	1	609.54	0.37	2.40	4292.28
	5	604.34	0.28	3.05	2610.76
	10	605.97	0.28	4.10	2488.21

5. Conclusion

This paper proposed a large-margin representation learning approach comprising convolutional layers and a large-margin discriminant. As a result, the LMFCN achieved competitive accuracy compared to CNNs with similar architecture but with a reduced computational cost. Furthermore, the LMFCN achieved training stability in a few epochs thanks to using only the SVs in the backpropagation algorithm. These achievements were possible using a large-margin discriminant, which replaces the fully connected and softmax layers of conventional CNNs. As a result, an SVM with an RBF kernel can produce complex margins, avoiding expensive refinement of the latent representation. This way, the FCNs do not need to suffer drastic updates. Unlike the handcrafted feature extractors, the LMFCN has more adaptability, given its consistent results in different datasets. The most problematic scenario for the LMFCN is the multiclass classification task, but it showed promising results despite the overhead caused by the OVA approach. However, it requires a few epochs per subproblem to alleviate the computational cost of using several models, keeping the cost of our approach similar to the CNNs. In conclusion, the LMFCN has advantages in computational cost and classification performance over the compared methods for small datasets of textural images.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

This work was funded by the NSERC-Canada, Grant RGPIN 2016-04855, and Collaborations Intl de Recherche de l'ÉTS.

References

[1] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: IEEE/CVF Conf Comp Vis Patt Recog, 2016, pp. 770–778.
[2] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, Z. Wojna, Rethinking the inception architecture for computer vision, in: IEEE/CVF Conf Comp Vis Patt Recog, 2016, pp. 2818–2826.
[3] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A.C. Berg, L. Fei-Fei, ImageNet large scale visual recognition challenge, Int. J. Comput. Vis. 115 (3) (2015) 211–252, doi:10.1007/s11263-015-0816-y.
[4] V. Andrearczyk, P.F. Whelan, Using filter banks in convolutional neural networks for texture classification, Pattern Recognit. Lett. 84 (2016) 63–69, doi:10.1016/j.patrec.2016.08.016.
[5] L. Alzubaidi, J. Zhang, A.J. Humaidi, A. Al-Dujaili, Y. Duan, O. Al-Shamma, J. Santamaría, M.A. Fadhel, M. Al-Amidie, L. Farhan, Review of deep learning: concepts, CNN architectures, challenges, applications, future directions, J. Big Data 8 (2021) 1–74.
[6] M. Cimpoi, S. Maji, A. Vedaldi, Deep filter banks for texture recognition and segmentation, in: IEEE Conf Comp Vis Patt Recog, 2015, pp. 3828–3836.
[7] W. Gong, C.F. Beckmann, A. Vedaldi, S.M. Smith, H. Peng, Optimising a simple fully convolutional network for accurate brain age prediction in the PAC 2019 challenge, Front. Psychiatry 12 (2021) 627996.
[8] A. Humeau-Heurtier, Texture feature extraction methods: a survey, IEEE Access 7 (2019) 8975–9000.
[9] S. Chopra, R. Hadsell, Y. LeCun, Learning a SIMILARITY metric discriminatively, with application to face verification, in: IEEE CS Conf Comp Vis Patt Recog, 2005, pp. 539–546, doi:10.1109/CVPR.2005.202.
[10] S. Kim, D. Kim, M. Cho, S. Kwak, Proxy anchor loss for deep metric learning, in: IEEE/CVF Conf Comp Vis Patt Recog, 2020, pp. 3238–3247.
[11] X. Wang, Y. Hua, E. Kodirov, G. Hu, R. Garnier, N.M. Robertson, Ranked list loss for deep metric learning, in: IEEE/CVF Conf Comp Vis Patt Recog, 2019, pp. 5207–5216.
[12] Y. Movshovitz-Attias, A. Toshev, T.K. Leung, S. Ioffe, S. Singh, No fuss distance metric learning using proxies, in: IEEE Intl Conf Comp Vis, 2017, pp. 360–368.
[13] N. Aziere, S. Todorovic, Ensemble deep manifold similarity learning using hard proxies, in: IEEE/CVF Conf Comp Vis Patt Recog, 2019, pp. 7299–7307.
[14] P.M. Roland Kwitt, Stex, Salzburg texture image database (stex), 2019, https://wavelab.at/sources/STex/. Accessed: 2022-03-28.
[15] F.A. Spanhol, L.S. Oliveira, C. Petitjean, L. Heutte, Breast cancer histopathological image classification using convolutional neural networks, in: Intl Joint Conf Neural Networks, 2016, pp. 2560–2567.
[16] G. Aresta, T. Araújo, S. Kwok, et al., Bach: grand challenge on breast cancer histology images, Med. Image Anal. 56 (2019) 122–139, doi:10.1016/j.media.2019.05.010.
[17] J.N. Kather, C.-A. Weis, F. Bianconi, S.M. Melchers, L.R. Schad, T. Gaiser, A. Marx, F.G. Zöllner, Multi-class texture analysis in colorectal cancer histology, Sci. Rep. 6 (1) (2016) 27988, doi:10.1038/srep27988.
[18] J. de Matos, B.A. de Souza, L.E.S. de Oliveira, A.L. Koerich, Texture CNN for histopathological image classification, in: IEEE Intl Symp Comp-Based Med Syst, 2019, pp. 580–583, doi:10.1109/CBMS.2019.00120.