Pattern Recognition 141 (2023) 109649

Contents lists available at ScienceDirect

Pattern Recognition

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

Deep feature annotation by iterative meta-pseudo-labeling on 2D projections

Bárbara C. Benato[a,*], Alexandru C. Telea[b], Alexandre X. Falcão[a]

[a] Laboratory of Image Data Science, Institute of Computing, University of Campinas, Campinas, Brazil
[b] Department of Information and Computing Sciences, Faculty of Science, Utrecht University, Utrecht, the Netherlands

ARTICLE INFO

Article history:
Received 28 July 2022
Revised 9 March 2023
Accepted 27 April 2023
Available online 29 April 2023

Keywords:
Pseudo-labeling
Deep feature annotation
Semi-supervised learning
Feature space projection
Data annotation

ABSTRACT

The absence of large annotated datasets to train deep neural networks (DNNs) is an issue since manual annotation is time-consuming, expensive, and error-prone. Semi-supervised learning techniques can address the problem propagating pseudo labels from supervised to unsupervised samples. However, they still require training and validation sets with many supervised samples. This work proposes a methodology, namely *Deep Feature Annotation* (DeepFA), that dismisses the validation set and uses very few supervised samples (e.g., 1% of the dataset). DeepFA modifies the feature spaces of a DNN along with meta-pseudo-labeling iterations in a 2D non-linear projection space using the most confidently labeled samples of an optimum-path forest semi-supervised classifier. We present a comprehensive study on DeepFA and a new variant that detects the best DNN model for generalization during the pseudo-labeling iterations. We evaluate components of DeepFA on eight datasets, finding the best DeepFA approach and showing that it outperforms self-pseudo-labeling.

1. Introduction

The success of deep neural networks (DNNs) is evident in many applications. However, among the supervised models, the need for large annotated training sets is a well-known problem [1,2], being data augmentation and transfer learning common approaches that create synthetic samples and exploit pre-learned weights to amend the problem, respectively.

To train DNNs, the set with supervised samples is usually split to generate a validation set for hyperparameter search and model evaluation. When the validation set is representative, it provides a good idea of the model's performance on unseen test sets [3]. A critical problem appears when the training set is too small [2] – with only dozens of supervised samples per class. Assuming a set with many unsupervised samples is available, semi-supervised learning techniques can propagate pseudo labels from the supervised samples to the unsupervised ones, considerably increasing the number of labeled training samples. However, semi-supervised learning techniques still require hundreds to thousands of supervised samples for training and validation [4–7].

This work proposes a meta-pseudo-labeling methodology, namely *Deep Feature Annotation* (*DeepFA*), to train DNNs from very few supervised samples (e.g., 1% of the dataset) without a validation set. In *DeepFA*, the teacher (a connectivity-based semi-supervised classifier) exploits modifications of a given latent feature space of the student (a DNN) along with iterations of non-linear projection on a 2D space for pseudo-labeling. At each iteration, the most confidently labeled samples are used to retrain the DNN, modifying its latent feature space. The semi-supervised classifier does not require parameter optimization, dismissing a validation set. It then increases the number of labeled samples to improve the DNN with pseudo-labeled training and validation sets.

For pseudo-labeling in 2D, we use a semi-supervised Optimum Path Forest (OPFSemi) classifier [8], which has outperformed several techniques in different works [9–12]. In [9], for instance, OPFSemi outperformed LapSVM [13], achieving the highest performance when label propagation was done in the 2D embedded space created by t-SNE [14] from the intermediary feature space of an autoencoder, in contrast to [10] where label propagation was performed in the latent feature space.

Isolated aspects of *DeepFA* using OPFSemi on a 2D embedded space have been evaluated in conference papers. In [12], a few iterations of the training loop with truly-and-artificially-labeled samples was enough to improve the generalization performance of a supervised DNN. The study used only 1%–5% of supervised samples. We call this version as *orig-DeepFA*, while we use *DeepFA* to refer to the entire methodology. OPFSemi's confidence was also considered when selecting unsupervised samples to train the

* Corresponding author.
E-mail addresses: barbara.benato@ic.unicamp.br (B.C. Benato), a.c.telea@uu.nl (A.C. Telea), afalcao@ic.unicamp.br (A.X. Falcão).

https://doi.org/10.1016/j.patcog.2023.109649
0031-3203/

Table 1
Comparison of how earlier *vs* our work address questions Q1.Q6 for *DeepFA*.

Question	Earlier work (*orig-*& *conf-DeepFA*)	Our contribution (*ext-DeepFA*)
Q1	used an autoencoder to generate the initial feature space for interactive data annotation [9,11];	use a supervised, pretrained, deep architecture with few available supervised samples [12];
Q2	compared OPFSemi label propagation only with Laplacian SVM [9,11]. Similar comparisons exist in [10] but were not applied to *DeepFA*;	compare OPFSemi to two additional semi-supervised learning methods (L.Prop, L.Spread) in *DeepFA*;
Q3	OPFSemi's confidence was used to select samples for label propagation [11], but not in *DeepFA*;	use OPFSemi's confidence to select samples to propagate labels in *DeepFA* [15];
Q4	only VGG-16 was explored to learn the feature space [12];	compare VGG-16 with an additional architecture (MobileNet [17]);
Q5	only the output of the last convolutional layer was used to propagate labels [12]	use different layers of the deep network for the same purpose;
Q6	such methods either used no iterations [9,11] or an upfront fixed number of iterations [12];	propose a clustering metric to find the iteration delivering the optimal trained model.

supervised DNN [15], reducing label propagation errors. Let us call this last version *conf-DeepFA*.

However, *conf-DeepFA*'s performance on test sets can oscillate along with the pseudo-labeling iterations such that the model obtained at the last iteration is not guaranteed to be the best model. To circumvent this problem, we propose *ext-DeepFA*, which extends *conf-DeepFA* by computing a clustering-based metric from the pseudo-labeled samples to select the optimal model for generalization among the ones generated along with the pseudo-labeling iterations. While earlier *orig-DeepFA* and *conf-DeepFA* variants have shown promising results, the methods may differ in the deep architecture used for feature learning and classification, the semi-supervised classifier for label propagation, the projection technique, and the criterion to select the model for generalization. Thoroughly exploring this 'design space' is needed to gain confidence in the results' robustness and, where possible, to find hyperparameters that lead to higher performance. The present work is then a comprehensive study on DeepFA.

We next outline six question whose answers support the exploration of *DeepFA*'s design space.

- (Q1) Can a deep neural network with pre-trained weights improve performance by self-pseudo-labeling?
- (Q2) Can performance be improved by using other label propagation methods than OPFSemi?
- (Q3) Can OPFSemi's confidence improve *DeepFA*'s pseudo-labeling?
- (Q4) Does *DeepFA* work for other deep architectures than the currently used VGG-16 [16]?
- (Q5) Can we improve *DeepFA* by choosing other layers from the network to extract a feature space?
- (Q6) How can we identify the optimal model among those computed during the *DeepFA* iteration sequence?

Table 1 summarizes our contributions, described next, as follows. Section 3 details *ext-DeepFA*, our extension of the existing *orig-DeepFA* and *conf-DeepFA* methods, as well as the experiments that we propose to address questions Q1..Q6. Section 4 details the experimental procedure. Section 5 shows the experimental results. Section 6 summarizes our answers to Q1..Q6. Section 7 discusses limitations of our work. Finally, Section 8 concludes the paper.

2. Related works

Supervised DNNs require large annotated sets for training models with high classification measures in the test set [1,2]. Several strategies are concerned with tackling this problem. In the following, we concern ourselves with cases where we know all classes to infer in advance – the problem of discovering new classes in the data is out of our scope. Recently, few-shot learning techniques have shown the ability to deal with the absence of large supervised datasets in classification tasks using DNNs. Using very few supervised samples, few-shot learning guarantees generalization in test sets when training from one to few (one/few-shot learning) or even zero (zero-shot learning) examples per class [18]. Both few-shot and semi-supervised learning methods use few supervised samples to train their models. However, in addition to the (few) available supervised samples, semi-supervised learning still considers many unsupervised samples to capture additional information during its learning process [18]. (A) These unsupervised samples are not part of standard few-shot learning methods. Recent works have explored combining few-shot learning and semi-supervised learning [19–21] by considering a fixed amount of unsupervised samples along with the few supervised ones. (B) Few-shot learning can take advantage of pre-trained models only when those are trained without using labels in an *unsupervised* pre-training [18]. Summarizing the above, few-shot learning differs from our approach since we (i) intend to use large amounts of unsupervised samples (unlike A); and (ii) benefit from pre-trained models on large labeled datasets for medical applications (unlike B).

In this work, we are interested in using very few supervised samples and many unsupervised ones, in a semi-supervised setup. When some supervised samples and many unsupervised samples are available, semi-supervised learning can increase the number of labeled training samples and, consequently, improve the performance of a DNN [4] – the primary goal of our work.

In semi-supervised learning, deep learning [4–7,22] approaches have been exploited to propagate labels from a small set of supervised samples to a large set of unsupervised ones exploiting their feature space distribution. Pseudo-labeling (a particular case of self-training) was first proposed for more effectively fine-tuning a pre-trained model [4]. Nevertheless, label propagation errors can negatively affect the classification performance of models trained with pseudo labels [9,23]. The confidence of the apprentice model was included in the loss function to mitigate the problem [22,24]. Recently, pseudo-labeling approaches [4,6,7] essentially adopt the semi-supervised strategy with the apprentice model assigning uncertain (pseudo) labels to unsupervised samples. Those approaches has been also combined with different strategies (*e.g.*, self-supervised methods [25,26]). With the same purpose, meta-pseudo-labeling [7] uses an auxiliary model (teacher) to generate pseudo labels to train the primary model (student). Still, to guarantee reasonable label propagation accuracy, such deep-learning-based methods require a training set with hundreds of supervised samples per class and a validation set with additional supervised samples for parameter optimization [4–7]. When only a few supervised samples (e.g., dozens per class) are available, this requirement is a clear obstacle.

Recent works have investigated pseudo labels stemming from graph-based semi-supervised methods, which are typically faster than deep learning methods. By modeling training samples as nodes of a graph whose arcs connect adjacent samples in a given feature space, one can propagate labels from supervised samples to their most strongly connected unsupervised ones. Such methods combine the connectivity aspect of graph-based methods with the

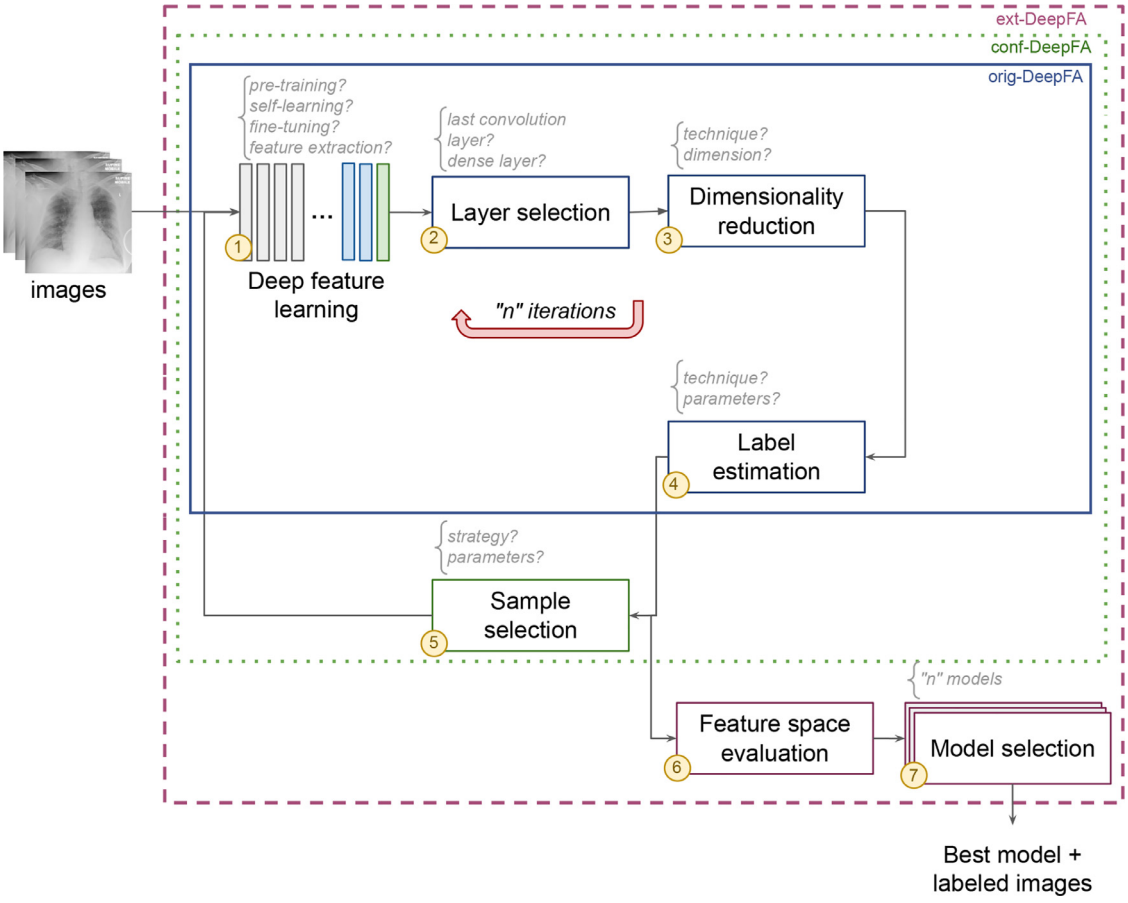


Fig. 1. Proposed *ext-DeepFA* pipeline. Using a few supervised images, a deep feature learning algorithm is trained (1) and features are extracted from unsupervised images from a selected layer (2). The features are projected to a 2D space (3) and a semi-supervised technique propagates labels from supervised to unsupervised samples in that space (4) – the *orig-DeepFA* pipeline (inner blue box). The most informative samples are selected (5) and their labels are used to retrain the network along with the supervised samples. Steps (1–5) are iteratively repeated – the *conf-DeepFA* pipeline (green dashed box). In each iteration, the labeled projected feature space is evaluated (6) using a quality measure to select the best model (7). The process outputs at end the optimal model and labeled samples – the *ext-DeepFA* pipeline (outer maroon dashed box). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

learning ability of DNNs in a co-training strategy [10,22,23]. However, most graph-based label propagation methods need parameter tuning for distinct datasets and also need a validation set with extra supervised samples. Again, the requirement of extra supervised samples is an obstacle since only very few supervised samples are available for training a DNN.

Among the studies that focus on labeling unsupervised data from a few supervised samples are those based on cluster-then-label methods for classification [27] and few-shot learning [28]. Specifically, such works do not use extra (data) supervision for parameter optimizing and searching [27,28]. They deal with consistency bias by combining (i) a similarity loss and (ii) an ensemble from different Gaussian Mixture Models. However, exploring challenging datasets can be an issue when using unsupervised learning [11] or similarity-based loss functions [29]. For such situations, strategies such as co-training, graph-consistency, and uncertainty information combined with limited supervised information can leverage pseudo-labeling approaches to avoid consistency biases [23].

3. Proposed pipeline

We next detail our extension *ext-DeepFA* of the original (*orig-DeepFA* [12]) and confidence-based (*conf-DeepFA* [15]) methods with the contributions listed in Table 1. Fig. 1 shows our extended method called *ext-DeepFA*. We next discuss each method step and outline how the questions Q1..Q6 relate to design decisions about these steps.

3.1. Deep feature learning

We start our pipeline by training a deep feature learning algorithm by using the few available supervised samples (Fig 1, step 1). Obviously, since supervised samples are few, one cannot expect good results using a too deep network trained from scratch. As such, an interesting question is whether a deep neural network with pre-trained weights can improve performance by self-pseudo-labeling (Q1). A negative answer implies the need of a separate machine learning method for pseudo labeling, such as OPFSemi proposed by *DeepFA* and its extensions. A separate question (Q4) related to this step is which are pre-trained models that one can use to obtain high performance in the context of our application, besides the VGG-16 one proposed in earlier work [12].

3.2. Layer selection

The feature spaces learned in the different deep layers of the network capture the structure of the data space the network is exposed to. Hence, we can use such a feature space as input for the label propagation (see Section 3.4 next). In earlier works, the last convolutional layer was used for this purpose [12,15]. An open question (Q5) is whether using deeper layers would improve performance.

3.3. Dimensionality reduction

The feature space from the selected network layer (Section 3.2) can be reduced before being used for label propagation. Earlier work has shown that using a 2D t-SNE projection for this often yields labels of higher accuracy than when propagation is done directly in the feature space [9]. A potential question is whether one could use for this step other existing projection methods than t-SNE. We believe there is evidence to the contrary: An extensive study [30] showed that t-SNE has one of the highest quality, measured in terms of combined trustworthiness, neighborhood hit, continuity, Shepard correlation, and normalized stress metrics (all common quality metrics in projection literature), among 45 studied projection algorithms. As such, from an application-agnostic perspective, one can say that t-SNE preserves the high-dimensional data structure better than its competitors, so it is the candidate of choice to be used.

Separately, the chosen label estimation algorithm (see Section 3.4) uses Euclidean distances *in the 2D projection space*. As such, having a compact projection where similar data points are close to each other (*i.e.*, with high neighborhood preservation), like t-SNE, favors our label propagation as opposed to projections which spread the data points more in 2D, *e.g.*, MDS variants. However, to fully prove our above point, *i.e.*, the suitability of 2D t-SNE projections, more studies are needed, *e.g.*, replacing t-SNE in our pipeline by other top-ranking projections in terms of quality from [30] such as UMAP, IDMAP, or PBC.

We also argue that using t-SNE to project features in spaces higher than 2D, *e.g.*, using a 3D projection, is overall not an attractive option since (a) 3D projections score only slightly quality metrics than their 2D counterparts [31]; (b) OPFSemi's use of Euclidean distances can be affected by the higher space dimensionality (and by higher Euclidean distance values as well); (c) users experience significantly higher difficulty when visualizing 3D as opposed to 2D projections, which only gets worse if one wants to use the projection to also interact with the data, *e.g.*, for manually fine-tuning the labeling.

3.4. Label estimation

The few available labels are propagated in the 2D projection to create a rich set of labeled points which is next used to refine the network training. As stated earlier, *DeepFA* uses OPFSemi for this step. OPFSemi first was proposed in [8] and we use the same algorithm for propagating pseudo labels from the supervised samples to the unsupervised ones in a semi-supervised way. However, OPFSemi can make mistakes and its effectiveness depends on filtering out the most likely mislabeled samples by thresholding its confidence value [11]. An open question here (Q2) is whether performance can be improved by using counterparts pseudo-labeling techniques rather than OPFSemi.

3.5. Sample selection

Earlier work [9] used all pseudo labels constructed by OPFSemi which, as noted earlier, can lead to training based on wrongly propagated labels. Separately, the confidence of OPFSemi was used to select a subset of most confident pseudolabels to use next [11]. However, this strategy has not been used in *DeepFA*. This raises the question (Q3) on how can confidence-based pseudo-label selection improve the *DeepFA* end-to-end pipeline.

3.6. Evaluation of the projected labeled space

Earlier methods either applied the *DeepFA* idea for a single iteration [9,11] or used a predefined number of iterations [12,15]. Both approaches are suboptimal if one is after finding the best way to train a deep model. Rather, we need to (a) execute the *DeepFA* iterations and (b) select, from the trained models computed after each iteration, the one with the highest performance.

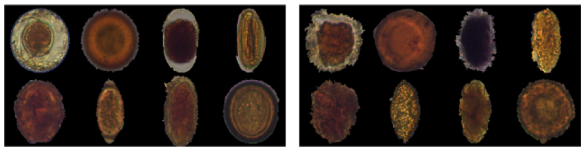


Fig. 2. Examples of *H.Eggs* species (left) and similar images of impurities (right).

For our scenario of few labeled samples, an inherent problem is how to gauge performance of such models. One approach is to use as a proxy the quality of the pseudo-labeled samples. Importantly, we cannot use true label information of the unsupervised samples for this purpose as this would defeat the very purpose of training with a very small supervised set (tens of samples in some cases). Rather, we need a quality metric that considers not only labels but also the separation (distance) between samples in the 2D projection space. A joint question (Q6) is which metric to use for this purpose and how to use it to determine the optimal model over all executed iterations.

4. Experimental setup

We next outline how we organized our study of the questions Q1.Q6 in Section 1 in terms of used datasets (Section 4.1, experimental setup (Section 4.2), and implementation (Section 4.3).

4.1. Datasets

We choose eight diverse datasets to perform our investigations, as follows. **MNIST:** We first chose the public MNIST [32] dataset, to explore a known and easy classification task. MNIST has 0 to 9 handwritten digits grayscale images (28×28 pixels). We use 5000 random samples from the original training dataset. **Parasites:** The next five datasets come from a Parasite medical image collection [33]. This collection has three main dataset types: (i) *Helminth larvae*, (ii) *Helminth eggs*, and (iii) *Protozoan cysts*. The datasets contain color microscopy images (200×200 pixels) of the most common species of human intestinal parasites in Brazil, responsible for public health problems in most tropical countries [33]. The datasets are challenging since they are unbalanced and contain a majority impurity class, with samples very similar to parasites, making classification hard (see Fig. 2). Table 2 shows the number, type, and sample count per class for the datasets (i-iii) listed above. To these datasets, we add the (iv) *Helminth eggs* and (v) *Protozoan cysts* without the impurity class datasets, yielding a total of 5 datasets. **Coconut:** We use a random subset of the Coconut trees dataset [34] with 7,827 regions (90×90 pixels) of aerial colored images from the Kingdom of Tonga, acquired by satellite imagery in October 2017, labeled by Humanitarian OpenStreetMap. The dataset has two classes: images with (6,139) or without coconut trees (1,688). **COVID:** A team of researchers from the Universities of Qatar and Dhaka have created a database [35,36] of chest X-ray images (299×299 pixels). We obtained the second update of the dataset with 21165 images split in four classes: COVID-19 positive (3616), lung opacity (non-COVID lung infections, 6012), viral pneumonia (1345), and normal (10192) cases. We use a randomly subset of this dataset with 10583 images.

4.2. Experimental setup

To reproduce the scenario of few supervised samples, we define a supervised training set S with only 1% of supervised samples

Table 2
MNIST, Coconut, COVID19 (left) and three Parasites datasets (right). For each dataset, we list its number of classes, class names, and sample count per class.

Dataset	Classes	# samples	Dataset	Classes	# samples
	zero	479		*S.stercoralis*	446
	one	563	(i) *H. larvae* (2 classes)	impurities	3068
	two	488		total	3514
	tree	493		*H.nana*	348
	four	535		*H.diminuta*	80
MNIST (10 classes)	five	434		*Ancilostomideo*	148
	six	501		*E.vermicularis*	122
	seven	550		*A.lumbricoides*	337
	eight	462	(ii) *H. eggs* (9 classes)	*T.trichiura*	375
	nine	495		*S.mansoni*	122
	total	5000		*Taenia*	236
	coconut tree	6139		impurities	3444
Coconut (2 classes)	none	1688		total	5112
	total	7827		*E.coli*	719
	COVID19	1808		*E.histolytica*	78
	lung opacity	3002		*E.nana*	724
COVID19 (4 classes)	pneumonia	672		*Giardia*	641
	normal	5096	(iii) *P. cysts* (7 classes)	*I.butschlii*	1501
	total	10,583		*B.hominis*	189
				impurities	5716
				total	9,568

Table 3
Number of supervised (S) and unsupervised (U) samples for each dataset.

	MNIST	H.eggs (w/o imp)	P. cysts (w/o imp)	H. larvae	H. eggs	P. cysts	Coconut	COVID19
S	50	17	38	35	51	95	78	105
U	3450	1220	2658	2424	3527	6602	5400	7302

from a given dataset D. The unsupervised U and test T sets have 69% and 30% of samples, respectively ($D = S \cup U \cup T$). The small S simulates the real-world scenario when one has a large D but manual effort is needed to label samples to create S. We randomly divide each dataset D into S, U, and T in a stratified manner and also generate three distinct splits for each experiment for statistical analysis. Table 3 shows the number of supervised samples in S for each of the eight datasets in Section 4.1.

We evaluate our method by the probability of the chosen deep architecture's last fully-connected layer, *i.e.*, just before the classification layer. From this, we compute accuracy. Since we have unbalanced datasets, we also use Cohen's κ. $\kappa \in [-1, 1]$ gives the agreement level between two distinct predictions, where $\kappa \leq 0$ means no possibility and $\kappa = 1$ means full possibility of agreement occurring by chance, respectively. We evaluate label propagation accuracy by computing the number of correctly assigned labels in U.

4.3. Implementation details

As stated, we want to use *DeepFA* without a validation set whose creation would require extra user supervision (data annotation effort). For this, we fix all pipeline's parameters without any optimization step. For t-SNE, we use the default parameters in *scikit-learn*. Note that OPFSemi has no parameters.

All our neural networks were implemented in Python using Keras [37], replacing the original fully-connected layers by two fully-connected layers with 4096 neurons and rectified linear activation followed by a decision layer with c neurons, where c is the number of classes of each dataset, and softmax activation. Models are trained by error backpropagation for a categorical cross-entropy function, using stochastic gradient descent with a linearly decaying learning rate initialized at 0.1 and momentum of 0.9, respectively. We loaded ImageNet pretrained weights and used a linear decay of 1×10^{-6} over 15 epochs. The pretrained weights for convolutional layers were fixed for the feature extraction experiments and unfrozen for fine-tuning, respectively.

5. Experimental results

We next address questions Q1.Q6 listed in Section 1 by presenting experiments, results, and discussion for each of them.

5.1. Q1: Can a pretrained VGG-16 improve performance by self pseudo labeling?

We evaluate VGG-16 pre-trained on ImageNet, with and without fine-tuning its convolutional layers (Section 4.3). We also evaluate VGG-16 performing self pseudo-labeling. We did four experiments (below, *ft* stands for fine-tuning and *fe* stands for feature extraction, respectively):

- *VGG-16*$_{ft}$: VGG-16 with fine-tuning, trained on S and tested on T;
- *self-VGG-16*$_{ft}$: VGG-16$_{ft}$ trained on $S \cup U$, being the samples in U pseudo-labeled by VGG-16$_{ft}$, and tested on T. The self training loop uses five iterations;
- *VGG-16*$_{fe}$: VGG-16 with only the last four convolutional layers used as *unfrozen* for feature extraction, trained on S and tested on T;
- *self-VGG-16*$_{fe}$: VGG-16$_{fe}$ trained on $S \cup U$, being the samples in U pseudo-labeled by VGG-16$_{fe}$, and tested on T. The self training loop uses five iterations.

5.1.1. Q1: Results and discussion

Table 4 shows the mean values of accuracy in label propagation and classification, κ, and their standard deviations over three splits, using each VGG-16-based model. We can see that feature extraction and fine-tuning do not show relevant gains in self pseudo-labeling along with the iterations. The results of the models based

Table 4
(Q1) Results for four VGG-16 models considering feature extraction and fine-tuning. Best values per metric and dataset in bold.

		VGG-16 variants			
dataset	metric	*VGG-16$_{ft}$*	*self-VGG-16$_{ft}$*	*VGG-16$_{fe}$*	*self-VGG-16$_{fe}$*
MNIST	prop. acc	-	0.447238 ± 0.146	-	**0.586000 ±0.007**
	acc	**0.629555 ±0.037**	0.441334 ± 0.149	0.614444 ± 0.015	0.592222 ± 0.020
	kappa	**0.588195 ±0.041**	0.378648 ± 0.166	0.571176 ± 0.017	0.546162 ± 0.023
H.eggs (w/o imp)	prop. acc	-	**0.758825 ±0.088**	-	0.744004 ± 0.114
	acc	**0.790961 ±0.050**	0.779033 ± 0.095	0.738858 ± 0.054	0.774011 ± 0.131
	kappa	**0.752807 ±0.060**	0.735591 ± 0.113	0.693278 ± 0.060	0.734030 ± 0.153
P.cysts (w/o imp)	prop. acc	-	0.399481 ± 0.010	-	**0.648739 ±0.111**
	acc	0.561130 ± 0.093	0.400519 ± 0.011	**0.736159 ±0.027**	0.650230 ± 0.101
	kappa	0.324051 ± 0.175	0.020734 ± 0.021	**0.626632 ±0.039**	0.483706 ± 0.170
H.larvae	prop. acc	-	0.897384 ± 0.031	-	**0.912837 ±0.038**
	acc	0.874566 ± 0.001	0.886572 ± 0.017	0.893523 ± 0.017	**0.908689 ±0.040**
	kappa	0.021406 ± 0.019	0.174158 ± 0.208	0.256836 ± 0.203	**0.385892 ±0.402**
H.eggs	prop. acc	-	0.773803 ± 0.034	-	**0.847308 ±0.018**
	acc	**0.858323 ±0.013**	0.775750 ± 0.034	0.848327 ± 0.017	**0.850934 ±0.014**
	kappa	**0.734333 ±0.019**	0.519971 ± 0.114	0.713649 ± 0.030	0.714227 ± 0.038
P.cysts	prop. acc	-	0.730327 ± 0.022	-	**0.817978 ±0.004**
	acc	0.758853 ± 0.077	0.734239 ± 0.028	0.818182 ± 0.004	**0.824800 ±0.011**
	kappa	0.542967 ± 0.218	0.492070 ± 0.107	0.697633 ± 0.009	**0.705397 ±0.022**
Coconut	prop. acc	-	**0.826153 ±0.026**	-	0.817147 ± 0.016
	acc	0.821200 ± 0.027	0.828721 ± 0.026	**0.835249 ±0.027**	0.813822 ± 0.031
	kappa	0.304424 ± 0.182	0.324694 ± 0.153	**0.385120 ±0.147**	0.228646 ± 0.224
COVID19	prop. acc	-	0.659174 ± 0.022	-	**0.660816 ±0.016**
	acc	0.627612 ± 0.034	**0.677008 ±0.039**	0.589186 ± 0.094	0.675066 ± 0.028
	kappa	0.389689 ± 0.074	**0.480240 ±0.068**	0.274597 ± 0.239	0.476834 ± 0.049

on feature extraction only are usually better than those of the fine-tuned models. We notice considerable gains in accuracy and κ when using *VGG-16$_{fe}$* and *self-VGG-16$_{fe}$*, indicating that the work in [12] could have presented better results with feature extraction than using fine-tuning. As *self-VGG-16$_{fe}$* achieved the best results, we use this pipeline in all subsequent experiments.

5.2. Q2: *OPFSemi's pseudo-labeling* vs *other comparable methods*

As related work [12,15] only tested *orig-DeepFA* looping with OPFSemi, we evaluate other semi-supervised learning methods for label propagation over the learned (and next reduced) feature space over a few iterations. Specifically, we use the LabelPropagation (*L.Prop*) and LabelSpreading (*L.Spread*) methods, available in *scikit-learn*, with k-nearest neighbors (knn) and radial basis functions (rbf) kernels. As *L.Prop* and *L.Spread* have parameters and we want to avoid parameter searching (due the few supervised samples available), we set parameters to their default values in *scikit-learn*. The experiments done are listed below:

- *OPFSemi*: VGG-16 is trained on S. Deep features for $S \cup U$ from the last convolutional layer are projected in 2D with t-SNE and used for OPFSemi to propagate labels from S to all samples in U. VGG-16 is then retrained with $S \cup U$ and tested on T (at the last iteration of *orig-DeepFA* looping);
- *L.Prop$_{knn}$*: As above but replaces OPFSemi by *L.prop* (knn kernel);
- *L.Prop$_{rbf}$*: As above but replaces OPFSemi by *L.prop* (rbf kernel);
- *L.Spread$_{knn}$*: As above but replaces OPFSemi by *L.spread* (knn kernel);
- *L.Spread$_{rbf}$*: As above but replaces OPFSemi by *L.spread* (rbf kernel).

5.2.1. Q2: Results

Table 5 and Fig. 7 show the results of the experiments that compare *OPFSemi* against *L.Prop$_{rbf}$*, *L.Prop$_{knn}$*, *L.Spread$_{rbf}$*, and *L.Spread$_{knn}$* for label propagation in the *orig-DeepFA* looping. We show mean values of label propagation accuracy, classification accuracy, κ, and their standard deviation over three different splits. From the compared methods, OPFSemi yielded the best performance for *all* tested datasets. This is an important result since we want to reproduce a real scenario with a few supervised samples (Sec. 4.2) and OPFSemi shows that it is possible to handle diverse datasets with no parameter optimization. Interestingly, *L.Spread$_{rbf}$* showed the highest mean label propagation accuracy in the Coconut dataset, but having the highest standard deviation. While *L.Prop* shows very low results for its default parameters, *L.Spread$_{rbf}$* and *L.Spread$_{knn}$* show better results depending of the dataset. For datasets with significant confusion between distinct classes in the 2D projection (P.cysts, Coconut, COVID19), *L.Spread$_{rbf}$* surpasses *L.Spread$_{knn}$*.

5.2.2. Q2: Discussion

Using different pseudo-labeling methods within the *orig-DeepFA* looping means that the label propagation and the learned feature space can be mutually affected. To evaluate how, Fig. 3 shows the resulting feature space and label estimation of the two best methods found in Sec. 5.2, *i.e., OPFSemi* and *L.Spread$_{rbf}$*. In this figure, we use datasets with similar classification values (Coconut tree, $\kappa = 0.53$) and distinct classification values (H.larvae, $\kappa \in \{0.80, 0.06\}$).

For the Coconut dataset, we see that *L.Spread$_{rbf}$* was more conservative in propagating labels (green points concentrated in the bottom part of the projection), while *OPFSemi* was more sensitive to outliers (green points in other projection regions). This is confirmed by the f1-score metric for classes 1 (red) and 2 (green), where *OPFSemi* got higher f1 values for class 2 (0.64) compared to *L.Spread$_{rbf}$* (0.60). Also, the two methods influenced the 2D projection space, which is more circular for *L.Spread$_{rbf}$* and more more elongated for *OPFSemi*. Still, neither method could separate the feature space into distinct per-class clusters.

For H.larvae, *OPFSemi* was able to propagate labels only for regions with supervised samples of class 1 (red) and also provide a feature space (2D projection) in which class 1 is separated from other samples in the projection. The f1-score confirmed that as both classes have f1 values up to 0.8. In contrast, *L.Spread$_{rbf}$* was more conservative in propagating labels for class 2 (green) *vs* class 1 (red) so that only the closest samples of class 2 were labeled with that class. The f1-scores for both classes were lower than

Table 5
(Q2) Results from the last iteration for experiments using five label propagation methods over five iterations. Best values per dataset in bold.

		semi-supervised learning methods				
dataset	metric	$L.Prop_{rbf}$	$L.Prop_{knn}$	$L.Spread_{rbf}$	$L.Spread_{knn}$	*OPFSemi*
MNIST	prop. acc	0.095714 ± 0.000	0.095714 ± 0.000	0.416857 ± 0.005	0.460953 ± 0.016	**0.790000 ±0.047**
	acc	0.096000 ± 0.000	0.096000 ± 0.000	0.402889 ± 0.013	0.451555 ± 0.015	**0.797778 ±0.049**
	kappa	0.000000 ± 0.000	0.000000 ± 0.000	0.337752 ± 0.015	0.391369 ± 0.017	**0.775103 ±0.054**
H.eggs (w/o imp)	prop. acc	0.146591 ± 0.088	0.197251 ± 0.000	0.872811 ± 0.036	0.602263 ± 0.108	**0.983293 ±0.004**
	acc	0.145637 ± 0.087	0.195857 ± 0.000	0.898933 ± 0.028	0.622097 ± 0.111	**0.970496 ±0.003**
	kappa	0.000000 ± 0.000	0.000000 ± 0.000	0.879848 ± 0.033	0.547943 ± 0.132	**0.965085 ±0.003**
P.cysts (w/o imp)	prop. acc	0.186573 ± 0.000	0.186573 ± 0.000	0.264960 ± 0.008	0.472676 ± 0.051	**0.800569 ±0.035**
	acc	0.186851 ± 0.000	0.186851 ± 0.000	0.256055 ± 0.023	0.472030 ± 0.039	**0.819493 ±0.041**
	kappa	0.000000 ± 0.000	0.000000 ± 0.000	0.078200 ± 0.015	0.319221 ± 0.041	**0.756949 ±0.054**
H.larvae	prop. acc	0.127288 ± 0.001	0.126881 ± 0.000	0.306222 ± 0.044	0.609597 ± 0.047	**0.954182 ±0.008**
	acc	0.127014 ± 0.000	0.127014 ± 0.000	0.272986 ± 0.041	0.634123 ± 0.062	**0.955450 ±0.002**
	kappa	0.000000 ± 0.000	0.000000 ± 0.000	0.032267 ± 0.027	0.187448 ± 0.043	**0.789743 ±0.010**
H.eggs	prop. acc	0.069499 ± 0.002	0.052357 ± 0.030	0.482299 ± 0.049	0.621297 ± 0.062	**0.936743 ±0.011**
	acc	0.067797 ± 0.000	0.050413 ± 0.030	0.454803 ± 0.051	0.636679 ± 0.084	**0.942634 ±0.016**
	kappa	0.000000 ± 0.000	0.000000 ± 0.000	0.303932 ± 0.073	0.468250 ± 0.094	**0.899307 ±0.027**
P.cysts	prop. acc	0.079389 ± 0.007	0.075307 ± 0.000	0.464935 ± 0.059	0.421283 ± 0.034	**0.732716 ±0.056**
	acc	0.075235 ± 0.000	0.075235 ± 0.000	0.471381 ± 0.052	0.425404 ± 0.052	**0.740973 ±0.056**
	kappa	0.000000 ± 0.000	0.000000 ± 0.000	0.302018 ± 0.061	0.264922 ± 0.050	**0.580626 ±0.092**
Coconut	prop. acc	0.785262 ± 0.001	0.788061 ± 0.006	**0.821468 ±0.011**	0.783741 ± 0.022	0.815930 ± 0.036
	acc	0.784163 ± 0.000	0.790691 ± 0.011	0.835107 ± 0.016	0.804030 ± 0.026	**0.839364 ±0.017**
	kappa	0.000000 ± 0.000	0.068275 ± 0.118	0.402744 ± 0.096	0.275569 ± 0.228	**0.489274 ±0.092**
COVID19	prop. acc	0.170919 ± 0.000	0.171729 ± 0.001	0.540390 ± 0.041	0.512263 ± 0.066	**0.589487 ±0.039**
	acc	0.170709 ± 0.000	0.170709 ± 0.000	0.569869 ± 0.051	0.532283 ± 0.051	**0.614173 ±0.040**
	kappa	0.000000 ± 0.000	0.000000 ± 0.000	0.363335 ± 0.057	0.328030 ± 0.054	**0.407068 ±0.066**

0.35, and the learned feature space presented more (and more spread) groups. At a higher level, Fig. 3 illustrates how distinct ways of propagating labels intervene in the learned feature space produced by the proposed *orig-DeepFA* looping.

5.3. Q3: Effects of adding OPFSemi's confidence

To analyze the impact of OPFSemi's confidence in the *orig-DeepFA* looping, we compared VGG-16 using different settings of OPFSemi's label propagation, with and without confidence, after five iterations, by the following experiments:

- *orig-DeepFA*: VGG-16 is trained on S. Deep features for $S \cup U$ from the last convolutional layer are projected in 2D with t-SNE and used by OPFSemi to propagate labels from S to *all* samples in U. VGG-16 is retrained from $S \cup U$ and tested on T;
- *conf-DeepFA*$_{\tau=x}$: As above but selecting pseudo-labeled samples $U_x \subset U$ with confidence $\tau \geq x$ to retrain VGG-16 from $S \cup U_x$, with $x \in \{0.7, 0.8, 0.9\}$.
- *conf-DeepFA*$_{\tau=\alpha}$: As above but starting with $\alpha = 0.8$ and increasing it by 0.04 at each iteration until a final value $\alpha = 0.96$.

5.3.1. Q3: Results

Table 6 shows the mean label propagation accuracy, classification accuracy, κ, and their standard deviation over three splits of our experiments. For all datasets, we see that selecting the most confident samples by OPFSemi during the *orig-DeepFA* looping improves the results. For *H.eggs* with and without impurities, the best results were obtained for $\tau = 0.7$. For MNIST, *H.larvae*, *P.cysts* and *COVID19*, the best results occurred for $\tau = 0.8$. For *P.cysts* without impurities, $\tau = 0.9$ and $\tau = \alpha$ led to the best (similar) results. Finally, $\tau = 0.9$ was the best choice for Coconut.

We conclude that confidence-based sampling shows clear added value in nearly all situations, as it increased κ up to 0.6 for all datasets (except *COVID19*). Setting τ is a dataset-dependent task. The adaptive ($\tau = \alpha$) confidence strategy does not seem to improve results on the test set compared to *orig-DeepFA* with no confidence sampling. One explanation can be our use of more samples in early iterations ($\tau = 0.8$) than in the later ones ($\tau = 0.96$). Testing whether the opposite strategy improves results is left for future study.

5.3.2. Q3: Discussion

Confidence-based sampling in the *orig-DeepFA* looping: Fig. 4 shows the average κ and propagation accuracy for *DeepFA* looping with full pseudolabeling of all samples (*orig-DeepFA*), our proposed *conf-DeepFA* using OPFSemi's confidence sampling for pseudolabeling with different ways to select the confidence threshold τ, and the best result for the VGG-16 experiments (*self-VGG*$_{fe}$, see Section 5.1), for all six studied datasets. For datasets yielding higher κ values, we see that *orig-DeepFA* obtained similar results to our proposed *conf-DeepFA* method. Yet, we see κ and propagation accuracy gains of almost 5% for the most challenging datasets. For *P.cysts* with impurities, κ gains actually over 10% and propagation accuracy gains over 17% – for which *orig-DeepFA* obtained worse results than VGG-16. In short, combining *DeepFA* with OPFSemi's confidence sampling (*conf-DeepFA* in Fig. 4) got the best results for most tested datasets. **Confidence-based sampling in *orig-DeepFA* along iterations:** Fig. 5 shows κ and propagation accuracy for one split of MNIST along five iterations of our experiments. We see that all compared approaches yielded an increase from the first to the second iteration, except *self-VGG-16*$_{fe}$. Also, we see that both κ and propagation accuracy slightly decrease after the third iteration. This may suggest that the proposed method saturates, mainly by the higher decrease in κ despite of propagation accuracy. The learned pseudolabels and the original images can be used as input for a better (known) deep architecture. Fig. 6 shows the plot for train and validation loss and accuracy considering 20% (from S) as validation set during one split of MNIST training. The initial learning curve and the learning curves for each iteration are also shown. The learning curves show that the labeled samples can improve the network convergence along iterations. As future work, a different deep network can be tested at the final stage. Also, an unsupervised quality measure can be proposed to define the best feature space found at certain iterations and, hence, the best iteration of the method. **Choosing OPFSemi's confidence threshold:** Adaptively selecting the confidence

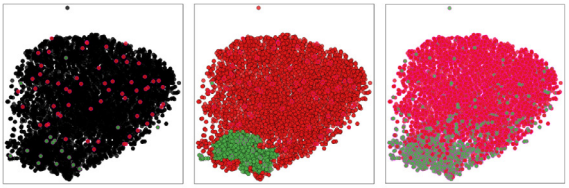
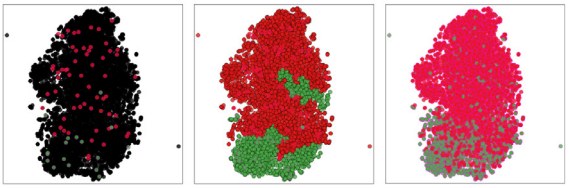
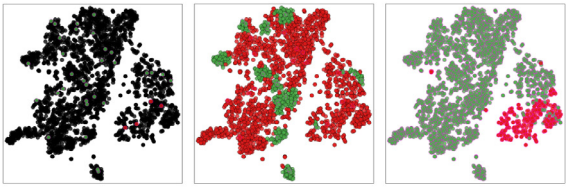
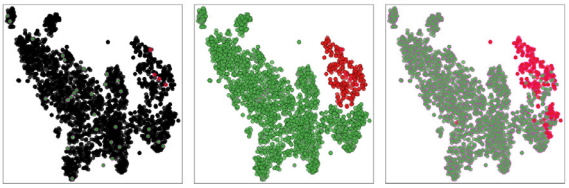
dataset	method	2D projections	metric	class 1	class 2	total
Coconut	$LSpread_{rbf}$		precision	0.87	0.79	
			recall	0.96	0.49	
			f1-score	0.92	0.60	
			accuracy			0.86
			kappa			0.53
	OPFSemi		precision	0.91	0.60	
			recall	0.88	0.67	
			f1-score	0.89	0.64	
			accuracy			0.83
			kappa			0.53
H.larvae	$LSpread_{rbf}$		precision	0.15	0.99	
			recall	0.99	0.21	
			f1-score	0.27	0.35	
			accuracy			0.31
			kappa			0.06
	OPFSemi		precision	0.82	0.97	
			recall	0.83	0.97	
			f1-score	0.82	0.97	
			accuracy			0.95
			kappa			0.80

Fig. 3. (Q2) Comparison of *DeepFA* using $LSpread_{rbf}$ and *OPFSemi* pseudo labeling for Coconut and H.Larvae datasets, with 1% supervised samples and last iteration out of five. 2D feature-space projections of training samples ($S \cup U$) in columns per dataset (from left to right): supervised samples colored by true labels (red=1, green=2), unsupervised ones are black; samples colored by assigned pseudo labels; and samples colored by their true labels. Classification results (per class and total) are shown on the right. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

threshold ($conf\text{-}DeepFA_{\tau=\alpha}$) looks promising only for one of the tested datasets. It shows a higher decreasing in κ when compared with the experiments without changing the confidence threshold τ along the iterations. As Section 5.2 outlined, choosing OPFSemi's confidence value may depend on the dataset, its difficulty, number of samples, number of classes, and class imbalance. Fig. 4 also shows this: It is not possible to define a *single* threshold τ for *all* chosen datasets. While this fact has been already noted in [11], it was not studied within a looping of data annotation as we did here. Rather, in [11], user interaction was employed to define the best confidence value based on the 2D projection guided by the data distribution and OPFSemi's confidence values (mapped to colors). We next intend to follow the same strategy to find the best confidence value for *conf-DeepFA* looping.

5.4. Q4: Choice of the deep architecture

As *COVID19* dataset's results showed the lower κ values in the realized experiments (Sec. 5.1, 5.2, and 5.3), we also aim to investigate the impact of VGG-16 architecture as feature learning strategy using labeled samples in the *DeepFA* looping. For this, we replace VGG-16 [16] by MobileNetV2 [17] (which has a reduced training time and good performance) and use ImageNet's pre-trained weights. We compared the two architectures by the

Table 6
(Q3) Results from the last iteration for proposed experiments with full label propagation (*orig-DeepFA*), and confidence-based label propagation (*conf-DeepFA*) with confidence $\tau \in \{0.7, 0.8, 0.9\}$ and adaptive confidence ($alpha \in [0.80, 0.96]$ over 5 iterations). Best values per dataset in bold.

		DeepFA variants				
dataset	metric	*orig-DeepFA*	*conf-DeepFA*$_{\tau=0.7}$	*conf-DeepFA* $_{\tau=0.8}$	*conf-DeepFA* $_{\tau=0.9}$	*conf-DeepFA* $_{\tau=\alpha}$
MNIST	prop. acc	0.790000 ± 0.047	0.782286 ± 0.029	**0.821714 ±0.018**	0.750000 ± 0.028	0.795429 ± 0.007
	acc	0.797778 ± 0.049	0.788000 ± 0.030	**0.822666 ±0.022**	0.740222 ± 0.032	0.651778 ± 0.062
	kappa	0.775103 ± 0.054	0.764348 ± 0.034	**0.802863 ±0.024**	0.710961 ± 0.036	0.612766 ± 0.069
H.eggs (w/o imp)	prop. acc	**0.983293 ±0.004**	0.983832 ± 0.002	0.974401 ± 0.020	0.981945 ± 0.003	0.983832 ± 0.004
	acc	0.790961 ± 0.050	**0.973007 ±0.006**	0.971123 ± 0.013	0.938481 ± 0.056	0.806654 ± 0.126
	kappa	0.752807 ± 0.060	**0.968042 ±0.007**	0.965848 ± 0.015	0.927708 ± 0.066	0.771216 ± 0.148
P.cysts (w/o imp)	prop. acc	0.800569 ± 0.035	0.805143 ± 0.049	0.793274 ± 0.069	0.824060 ± 0.019	**0.828141 ±0.012**
	acc	0.819493 ± 0.041	0.826413 ± 0.039	0.814590 ± 0.060	**0.842561 ±0.004**	0.824394 ± 0.033
	kappa	0.756949 ± 0.054	0.764035 ± 0.052	0.747127 ± 0.086	**0.785441 ±0.006**	0.762919 ± 0.041
H.larvae	prop. acc	0.954182 ± 0.008	0.964213 ± 0.017	**0.964349 ±0.012**	0.941846 ± 0.039	0.951471 ± 0.014
	acc	0.955450 ± 0.002	0.959558 ± 0.015	**0.965561 ±0.004**	0.958926 ± 0.014	0.943128 ± 0.010
	kappa	0.789743 ± 0.010	0.800052 ± 0.099	**0.837948 ±0.029**	0.804689 ± 0.082	0.705475 ± 0.069
H.eggs	prop. acc	0.936743 ± 0.011	0.936091 ± 0.005	**0.937209 ±0.008**	0.931806 ± 0.007	0.930967 ± 0.006
	acc	0.942634 ± 0.016	**0.943938 ±0.003**	0.942634 ± 0.009	0.908518 ± 0.022	0.853107 ± 0.025
	kappa	0.899307 ± 0.027	**0.901604 ±0.006**	0.898922 ± 0.015	0.831488 ± 0.043	0.719695 ± 0.054
P.cysts	prop. acc	0.732716 ± 0.056	0.769748 ± 0.026	**0.780300 ±0.018**	0.748843 ± 0.048	0.744811 ± 0.068
	acc	0.740973 ± 0.056	0.792755 ± 0.027	**0.816905 ±0.027**	0.818066 ± 0.022	0.731104 ± 0.082
	kappa	0.580626 ± 0.092	0.652254 ± 0.051	**0.699603 ±0.054**	0.689325 ± 0.039	0.450283 ± 0.243
Coconut	prop. acc	0.815930 ± 0.036	0.834003 ± 0.038	0.837349 ± 0.022	**0.856760 ±0.035**	0.791165 ± 0.034
	acc	0.839364 ± 0.017	0.853696 ± 0.019	0.876827 ± 0.012	**0.880232 ±0.006**	0.820633 ± 0.012
	kappa	0.489274 ± 0.092	0.481834 ± 0.075	0.603094 ± 0.076	**0.621104 ±0.026**	0.346671 ± 0.058
COVID19	prop. acc	0.589487 ± 0.039	0.579767 ± 0.048	0.613249 ± 0.018	0.586112 ± 0.049	**0.624770 ±0.114**
	acc	0.614173 ± 0.040	0.609869 ± 0.059	**0.662257 ±0.002**	0.647454 ± 0.020	0.647874 ± 0.090
	kappa	0.407068 ± 0.066	0.416806 ± 0.076	**0.478667 ±0.013**	0.420928 ± 0.026	0.433786 ± 0.087

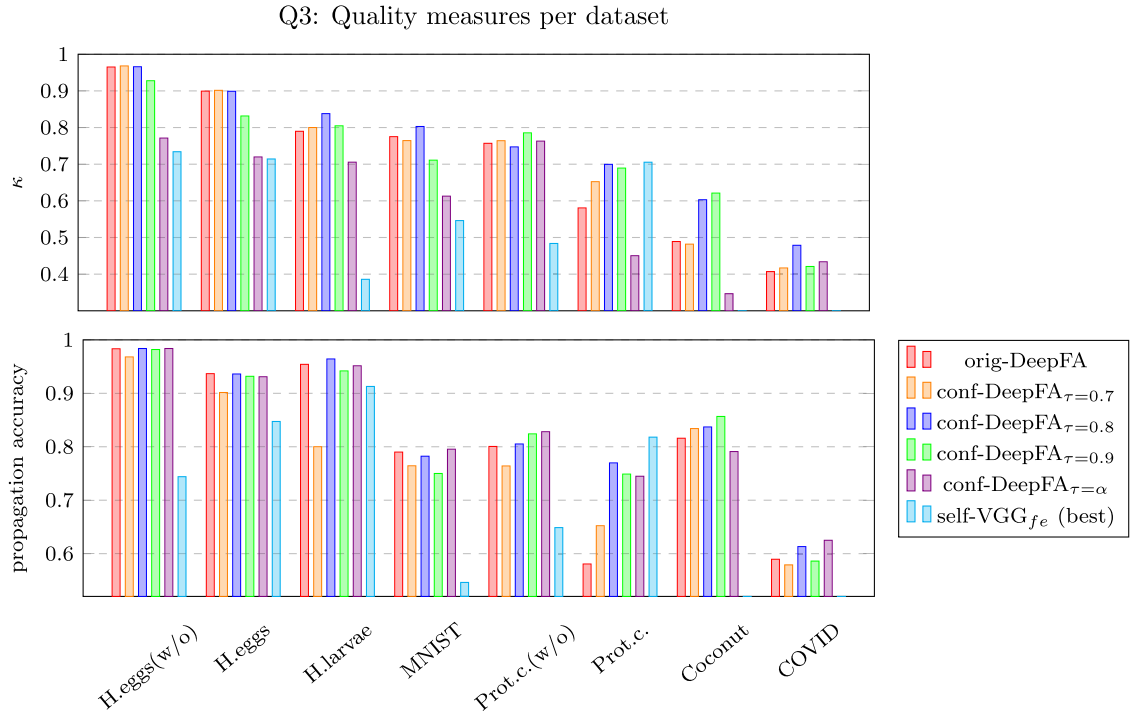


Fig. 4. (Q3) Results of κ (top) and propagation accuracy (bottom) for the studied datasets, considering self-VGG-16$_{fe}$ (best result), *orig-DeepFA*, and *conf-DeepFA*$_{\tau}$ experiments. The datasets are ordered by higher κ values in x axis (from left to right).

following experiments, each of them executed on both architectures $A \in \{VGG-16, MobileNetV2\}$:

- *orig-DeepFA*: The architecture A is trained on S. Deep features for $S \cup U$ from the last convolutional layer are projected in 2D with t-SNE and used by OPFSemi to pseudo label from S to all U samples. VGG-16 is retrained on these pseudo labels and tested on T (this is one iteration of *DeepFA* looping out of five);
- *conf-DeepFA*$_{\tau=0.8}$: As above, but OPFSemi pseudo labels from S to U_τ, for samples with confidence above $\tau = 0.8$.

5.4.1. Q4: Results

Table 7 shows the means of label propagation accuracy, classification accuracy, κ, and their standard deviation over three splits after five iterations. For COVID19, the pattern so far observed using VGG-16 was not reproduced by MobileNetV2 – i.e., a higher label propagation accuracy does not lead to the highest accuracy and κ values on the test set. Also, *conf-DeepFA* with $\tau = 0.8$ could not outperform *orig-DeepFA* using *MobileNetV2*. Different values of τ can be tested to validate that pattern. As such, for this dataset, we may conclude that MobileNetV2 was not able to learn

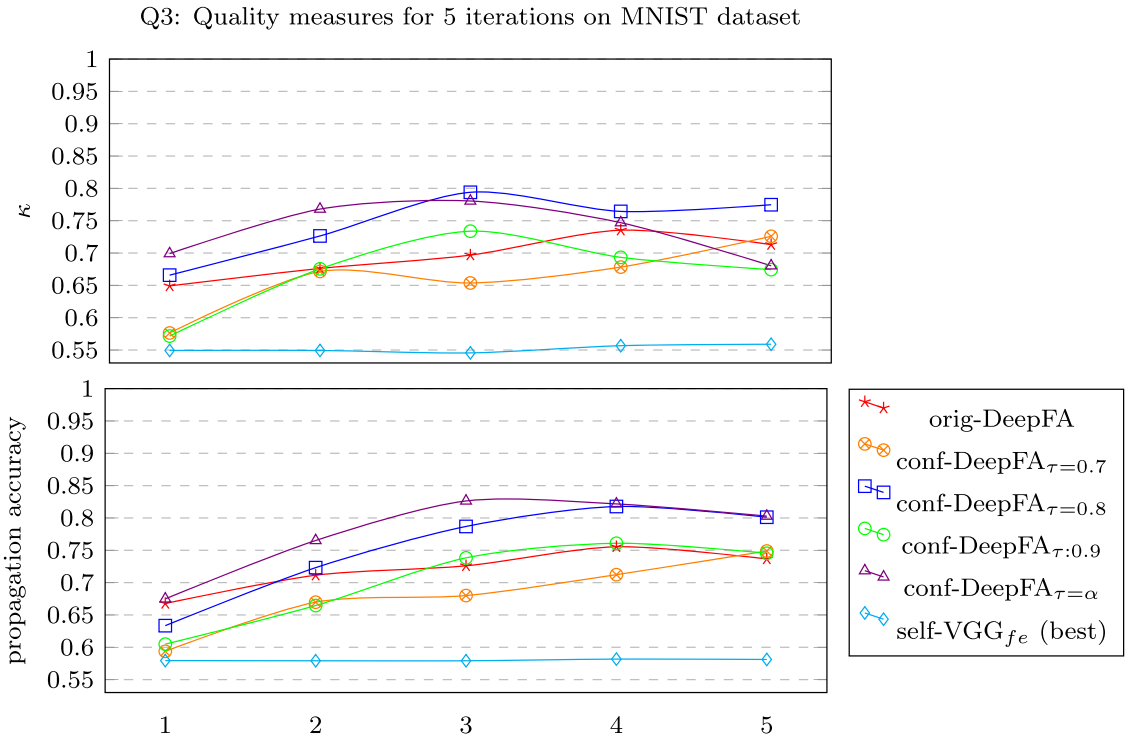


Fig. 5. (Q3) Results of κ (top) and propagation accuracies (bottom) for the MNIST dataset in one split over 5 iterations, considering self-VGG-16$_{fe}$ (best result), *orig-DeepFA*, and *conf-DeepFA* experiments.

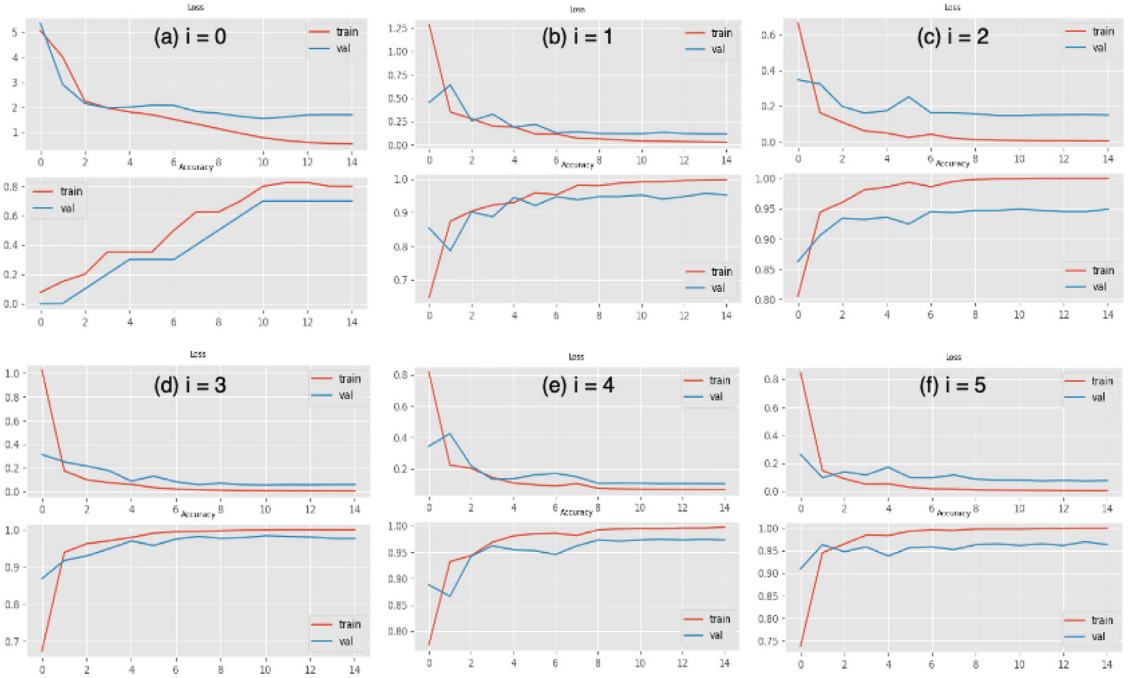


Fig. 6. Q3) Plots of loss and accuracy for one split of MNIST. The (a) initial learning curves and per-iteration curves (b-f) are shown.

Table 7
(Q4) Results from the last iteration for proposed experiments using VGG-16 and MobileNetV2 architectures with five learning iterations. Best values per dataset in bold.

dataset	metric	distinct architectures			
		VGG-16		MobileNetV2	
		orig-DeepFA	*conf-DeepFA*$_{\tau=0.8}$	*orig-DeepFA*	*conf-DeepFA*$_{\tau=0.8}$
COVID19	prop. acc	0.589487 ± 0.039	0.613249 ± 0.048	0.643580 ± 0.0385	**0.669839 ±0.0100**
	acc	0.614173 ± 0.040	**0.662257 ±0.002**	0.541732 ± 0.0528	0.539685 ± 0.0096
	kappa	0.407068 ± 0.066	**0.478667 ±0.013**	0.212528 ± 0.0628	0.147655 ± 0.0369

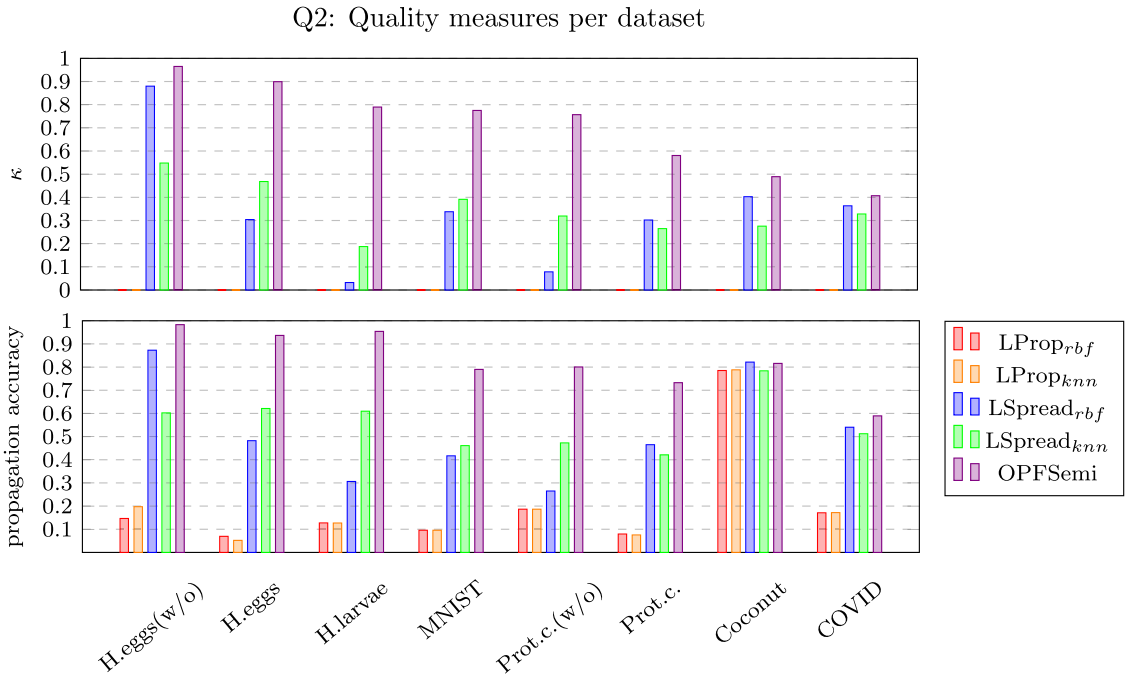


Fig. 7. (Q2) Results of κ (top) and propagation accuracies (bottom) for the studied datasets with orig-DeepFA using *L.Prop*$_{rbf}$, *L.Prop*$_{knn}$, *L.Spread*$_{rbf}$, *L.Spread*$_{knn}$, and OPFSemi. Datasets are ordered by higher κ values on the x axis.

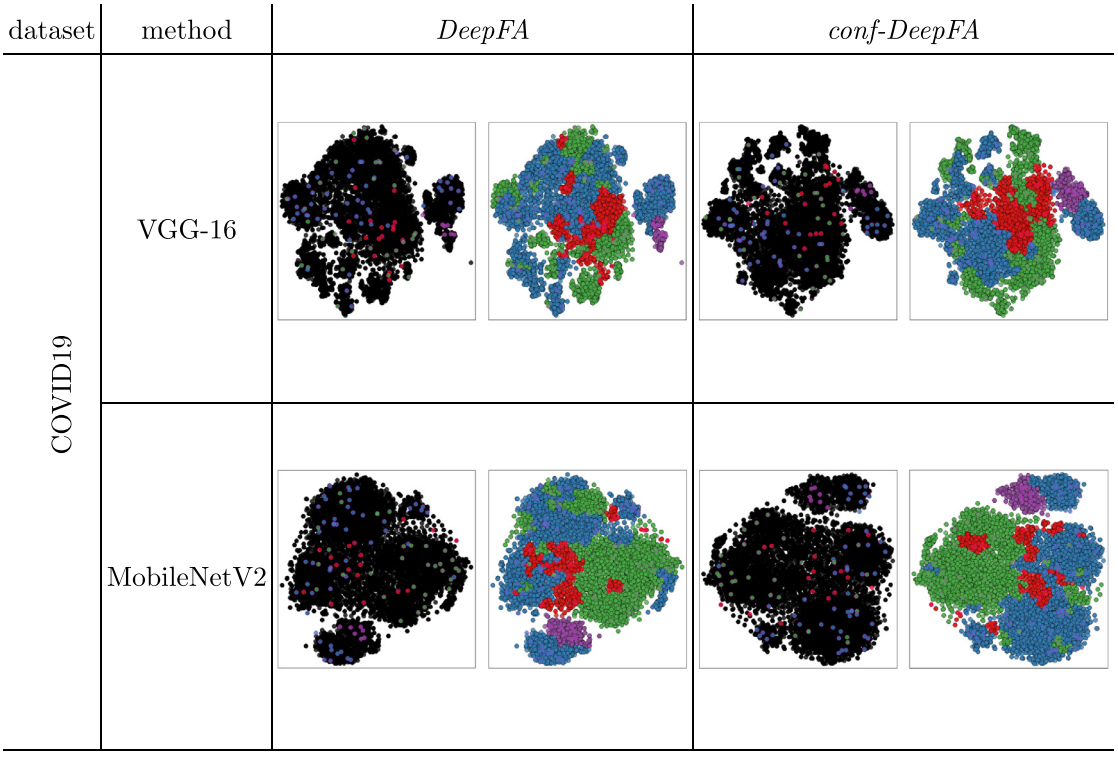


Fig. 8. Q4) Comparison of the feature spaces generated by *DeepFA* using VGG-16 and MobileNetV2 in pseudolabeling estimation for COVID19, 1% supervised samples and last iteration. 2D feature-space projections of training samples from left to right column per dataset: supervised samples colored by true labels, unsupervised ones are black; and samples colored by assigned pseudolabels.

a feature space in which OPFSemi propagates better labels than VGG-16.

5.4.2. Q4: Discussion

We investigate the MobileNetV2's feature space for the COVID19 dataset by showing its 2D projection and labeled samples (Fig. 8). We first notice that the projection presents a mixture between different classes. The supervised samples (colored points) are not clearly separated from the unsupervised ones (black), showing that this is a challenging dataset. For VGG-16, *conf-DeepFA* shows more groups having the same color, *e.g.*, the small groups around the larger one and red points grouped at the center. In contrast, *orig-DeepFA* shows a projection with more colors mixed in small groups and also red points are spread all over the larger group. For

MobileNetV2, there are no clear differences in the produced feature space; for the *orig-DeepFA* experiment, the red points seem to be more grouped at the projection center. MobileNetV2 shows fewer small groups having the same color around the larger group than VGG-16. This correlates with the accuracy and κ results for both architectures. It is possible that MobileNetV2 could achieve better results if considering more supervised samples. Too few supervised samples (from 10 to 100) for training this deep architecture is still a problem for its convergence in comparison to VGG-16.

5.5. Q5: Choice of the deep layer for feature extraction

In the earlier experiments, we only investigate our proposed deep feature learning looping using the last convolutional layer's output. However, Rauber et al. [38] showed that the multilayer perceptron (mlp) layers, located after the convolutional layers, can create a 2D projected space with better separation among different classes for shallow architectures and without loading pre-trained weights. To test this for our *DeepFA* looping, we compare the result of deep features provided by the output of (i) the last convolutional layer and (ii) the last mlp layer. To facilitate our analysis, we use a fixed value of τ for *conf-DeepFA* and five iterations for all datasets. Instead of showing the propagation accuracy (which does not consider the class unbalance) we compute the propagation κ for the labeled samples in training set. Our experiments are described below:

- L_{conv}: VGG-16 is trained on S. Features from the last convolutional layer for samples in $S \cup U$ are projected in 2D with t-SNE and used by OPFSemi to pseudo labels from S to U_τ, for $\tau = 0.8$. VGG-16 is retrained on these pseudo labels and tested on T (this is one iteration of *conf-DeepFA* looping out of five);
- L_{mlp}: As above, but using features from the last hidden fully-connected layer in the DNN.

5.5.1. Q5: Results and discussion

Table 8 shows the mean label propagation accuracy, classification accuracy, κ, and their standard deviation over three splits after five iterations. For MNIST, H.eggs, and Coconut, using the last convolutional layer obtained the best results. In contrast, for H.eggs without impurities, P.cysts without impurities, H.larvae, and COVID, the last mlp layer obtained the best results. For P.cysts, the results of using either L_{conv} or L_{mlp} were similar when considering the standard deviation. The choice of the layer to be used in our *conf-DeepFA* looping also affect the κ results on the test set. All in all, the choice of which layer to use seems to depend on the dataset.

Table 8
Q5) Results from the last iteration for experiments comparing the label propagation in the t-SNE projected space of distinct layers L_{conv} (the last convolutional layer's output) and L_{mlp} (the last MLP layer's output) during five iterations of conf-DeepFA looping with $\tau = 0.8$. Best values per dataset in bold.

		distinct output layers	
dataset	metrics	L_{conv}	L_{mlp}
MNIST	prop. kappa	**0.560920 ±0.317**	0.523341 ± 0.058
	acc	**0.605778 ±0.282**	0.584000 ± 0.057
	kappa	**0.561751 ±0.314**	0.537411 ± 0.063
H.eggs (w/o imp)	prop. kappa	**0.980513 ±0.001**	0.977636 ± 0.003
	acc	0.976146 ± 0.008	**0.977401 ±0.000**
	kappa	0.971776 ± 0.009	**0.973232 ±0.000**
P.cysts (w/o imp)	prop. kappa	0.773862 ± 0.015	**0.807765 ±0.043**
	acc	0.851211 ± 0.011	**0.861592 ±0.033**
	kappa	0.797588 ± 0.018	**0.810277 ±0.048**
H.larvae	prop. kappa	0.819094 ± 0.043	**0.824582 ±0.029**
	acc	0.958610 ± 0.002	**0.961137 ±0.005**
	kappa	0.807432 ± 0.017	**0.823241 ±0.040**
H.eggs	prop. kappa	**0.904969 ±0.003**	0.861352 ± 0.033
	acc	**0.945024 ±0.006**	0.930465 ± 0.015
	kappa	**0.902488 ±0.010**	0.875675 ± 0.025
P.cysts	prop. kappa	0.631756 ± 0.057	**0.652409 ±0.060**
	acc	0.808545 ± 0.036	**0.812609 ±0.026**
	kappa	**0.682719 ±0.070**	0.681252 ± 0.051
Coconut	prop. kappa	**0.623280 ±0.057**	0.514769 ± 0.114
	acc	**0.887186 ±0.018**	0.872996 ± 0.024
	kappa	**0.652869 ±0.060**	0.577084 ± 0.116
COVID19	prop. kappa	0.430679 ± 0.036	**0.516374 ±0.034**
	acc	0.672126 ± 0.041	**0.699108 ±0.033**
	kappa	0.490593 ± 0.059	**0.546634 ±0.043**

5.6. Q6: Choice of the best DeepFA iteration

As in earlier work, all our experiments so far used only a fixed number of (five) iterations. However, as Section 5.3 and Fig. 5 show, there is no guarantee that the last iteration delivers the best model. Following Section 3.6, we propose to evaluate the produced feature space in each iteration by using an unsupervised clustering metric. For this, we select the Calinski-Harabasz Index (CHI) [39] which calculates the ratio of the sum of between-cluster and sum of within-cluster dispersion for all clusters, where cluster dispersion is defined as the sum of squared distances over the cluster points. Formally put

$$CHI = \frac{\frac{\sum_{k=1}^{K} n_k ||c_k - c||^2}{K-1}}{\frac{\sum_{k=1}^{K} \sum_{i=1}^{n_k} ||x_i - c_k||^2}{N-K}}, \quad (1)$$

where x_i a sample in a dataset with N samples; K the number of clusters; n_k is the number of samples in cluster k; c_k is the centroid of cluster k; and c is the global centroid of all samples. As *CHI* is high when the obtained clusters are dense and well separated from each other, we propose to choose the best label estimation in the produced feature space by selecting that one which achieves the best *CHI* value after several iterations of our method.

To ease our analysis, we use a fixed $\tau = 0.8$ and *ten* total iterations for all datasets. Also, we compare the result of deep features provided by distinct layers (L) from the output of (i) the last convolutional layer (L_{conv}) and (ii) the last multilayer-perceptron layer (L_{mlp}). We compute the propagation κ instead of propagation accuracy for the labeled samples in the training set. The experiments are described below:

- *conf-DeepFA*$_{iter=5}$: VGG-16 is trained on S. Deep features for $S \cup U$ from L are projected in 2D with t-SNE, and used for OPFSemi's pseudo-labeling from S to U_τ, for samples with confidence above $\tau = 0.8$. OPFSemi's pseud labels are used to retrain VGG-16, and the network is tested on T (this is one iteration of *conf-DeepFA* looping out of *five*);
- *conf-DeepFA*$_{iter=best}$: As above, but we compute *CHI* on the 2D projections (over *ten* iterations) and select the model obtained from the iteration with the highest *CHI* value;
- *conf-DeepFA*$_{iter=worst}$: As above, but we select the iteration with the lowest *CHI* value.

5.6.1. Q6: Results

Table 9 shows the mean label propagation accuracy, classification accuracy, κ, and their standard deviation over three splits after five iterations, for the best among ten iterations, and for the worst among ten iterations. First, we consistently see that the highest *CHI* values lead to the best classification results (accuracy and κ) on the test set, except for P.cysts with and without impurities (for L_{conv}) and MNIST (for L_{mlp}). Also, the lowest *CHI* values lead to the worst classification results on the test set. The iteration with the best *CHI* value yields better results than those that we have found so far in our earlier experiments (last of five iterations). For MNIST, H.eggs with and without impurities, Coconut, and COVID19, the best iteration using L_{conv} obtains better classification results than

Table 9
Q5) Results for experiments comparing the label propagation in the t-SNE projected space of distinct layers (a) L_{conv} (last convolutional layer's output), and (b) L_{mlp} (last MLP layer's output) during *ten iterations* of *conf-DeepFA* looping with $\tau = 0.8$. The result for five iterations, and both the best and worst iterations chosen by the CHI value are presented. Best values per dataset and per layer L in bold.

		L_{conv}			L_{mlp}		
dataset	metric	iter= 5	iter = worst	iter = best	iter= 5	iter = worst	iter = best
	CHI	3805.08 ± 2544.0	1582.67 ± 1173.2	**5430.51 ±2356.6**	2807.49 ± 1172.5	1104.40 ± 198.9	**4055.42 ±83.5**
	prop. kappa	0.560920 ± 0.317	0.423657 ± 0.310	**0.704985 ±0.112**	0.523341 ± 0.058	0.379667 ± 0.056	**0.599035 ±0.023**
	acc	0.605778 ± 0.282	0.533556 ± 0.282	**0.743556 ±0.078**	**0.584000 ±0.057**	0.462000 ± 0.069	0.548000 ± 0.163
MNIST	kappa	0.561751 ± 0.314	0.479521 ± 0.316	**0.714611 ±0.087**	**0.537411 ±0.063**	0.401889 ± 0.076	0.496539 ± 0.183
	CHI	6161.35 ± 766.6	4200.59 ± 646.2	**6729.25 ±413.4**	**7599.88 ±308.6**	3434.74 ± 740.5	**7599.88 ±308.6**
	prop. kappa	0.980513 ± 0.001	0.978596 ± 0.006	**0.981472 ±0.002**	**0.977636 ±0.003**	0.961359 ± 0.008	**0.977636 ±0.003**
	acc	0.976146 ± 0.008	0.975518 ± 0.007	**0.979284 ±0.005**	**0.977401 ±0.000**	0.972379 ± 0.007	**0.977401 ±0.000**
H.eggs (w/o imp)	kappa	0.971776 ± 0.009	0.971012 ± 0.008	**0.975469 ±0.006**	**0.973232 ±0.000**	0.967319 ± 0.008	**0.973232 ±0.000**
	CHI	2447.19 ± 209.7	1453.61 ± 511.9	**3098.45 ±219.9**	2786.55 ± 198.5	1854.04 ± 253.9	**3534.84 ±260.9**
	prop. kappa	**0.773862 ±0.015**	0.672889 ± 0.062	0.684681 ± 0.032	**0.807765 ±0.043**	0.680152 ± 0.076	0.791739 ± 0.065
	acc	**0.851211 ±0.011**	0.793829 ± 0.036	0.791811 ± 0.025	0.861592 ± 0.033	0.816609 ± 0.041	**0.863898 ±0.031**
P.cysts (w/o imp)	kappa	**0.797588 ±0.018**	0.712134 ± 0.059	0.729677 ± 0.026	0.810277 ± 0.048	0.750887 ± 0.055	**0.817631 ±0.039**
	CHI	774.15 ± 214.7	499.97 ± 47.4	**920.94 ±340.6**	5546.44 ± 8201.1	570.23 ± 313.8	**8664.83 ±12964.6**
H.larvae	prop. kappa	0.819094 ± 0.043	0.775107 ± 0.050	**0.824502 ±0.038**	0.824582 ± 0.029	0.796519 ± 0.009	**0.830402 ±0.042**
	acc	0.958610 ± 0.002	0.952607 ± 0.009	**0.959242 ±0.006**	0.961137 ± 0.005	0.955134 ± 0.006	**0.964613 ±0.005**
	kappa	**0.807432 ±0.017**	0.755035 ± 0.068	0.806115 ± 0.031	0.823241 ± 0.040	0.785666 ± 0.054	**0.840656 ±0.035**
	CHI	1041.17 ± 8.9	724.86 ± 54.2	**1108.63 ±37.9**	1187.05 ± 47.2	846.19 ± 31.8	**1319.95 ±72.3**
	prop. kappa	**0.904969 ±0.003**	0.837934 ± 0.031	0.885585 ± 0.014	0.861352 ± 0.033	0.810120 ± 0.012	**0.870839 ±0.019**
	acc	0.945024 ± 0.006	0.917427 ± 0.022	**0.945676 ±0.005**	0.930465 ± 0.015	0.915906 ± 0.014	**0.931986 ±0.014**
H.eggs	kappa	0.902488 ± 0.010	0.851161 ± 0.040	**0.904525 ±0.009**	0.875675 ± 0.025	0.847946 ± 0.027	**0.878822 ±0.024**
	CHI	842.15 ± 212.7	737.011 ± 256.6	**1329.19 ±115.3**	1590.07 ± 128.3	1155.53 ± 289.7	**1861.46 ±197.0**
	prop. kappa	**0.631756 ±0.057**	0.626103 ± 0.075	0.631550 ± 0.063	0.652409 ± 0.060	0.632259 ± 0.017	**0.668595 ±0.040**
	acc	0.808545 ± 0.036	**0.811796 ±0.021**	0.792523 ± 0.039	0.812609 ± 0.026	0.800882 ± 0.032	**0.813886 ±0.020**
P.cysts	kappa	**0.682719 ±0.070**	0.682633 ± 0.055	0.665641 ± 0.056	0.681252 ± 0.051	0.668188 ± 0.052	**0.696163 ±0.029**
	CHI	4043.65 ± 1101.3	1338.66 ± 707.1	**5028.79 ±1068.9**	2811.05 ± 1232.2	930.54 ± 762.9	**4469.81 ±1436.5**
	prop. kappa	0.623280 ± 0.057	0.448743 ± 0.125	**0.623487 ±0.105**	0.514769 ± 0.114	0.370199 ± 0.228	**0.571377 ±0.092**
	acc	**0.887186 ±0.018**	0.863914 ± 0.016	0.886760 ± 0.015	0.872996 ± 0.024	0.834823 ± 0.041	**0.879239 ±0.013**
Coconut	kappa	0.652869 ± 0.060	0.537551 ± 0.071	**0.669792 ±0.040**	0.577084 ± 0.116	0.379283 ± 0.274	**0.634328 ±0.043**
	CHI	1112.59 ± 512.8	511.67 ± 110.9	**1516.63 ±270.7**	3712.00 ± 951.4	1827.87 ± 677.0	**4830.30 ±882.7**
	prop. kappa	0.430679 ± 0.036	0.410851 ± 0.046	**0.494343 ±0.025**	0.516374 ± 0.034	0.360516 ± 0.014	**0.538565 ±0.034**
	acc	0.672126 ± 0.041	0.646614 ± 0.046	**0.713491 ±0.012**	0.699108 ± 0.033	0.634016 ± 0.032	**0.699318 ±0.033**
COVID19	kappa	0.490593 ± 0.059	0.459456 ± 0.056	**0.555074 ±0.019**	0.546634 ± 0.043	0.436049 ± 0.044	**0.548049 ±0.044**

Fig. 9. Q6) Results of CHI (top) and κ (bottom) for the studied datasets, considering the last convolutional layer (left) and the last mlp layer (right) for *conf-DeepFA* with fixed certainty value ($\tau = 0.8$). The worst (red) and the best (green) iterations out of *ten* iterations were chosen based in the worst and best CHI values on the training set respectively. The results after five iterations (orange) are also shown. The datasets are ordered by higher κ values in x axis (from left to right). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

when using L_{mlp}. For this dataset, using L_{conv} with the highest-*CHI* iteration selection strategy increases κ by 0.15 as compared to using the last iteration.

5.6.2. Q6: Discussion

Using *CHI* to select the labeled feature space: Fig. 9 shows *CHI* and κ for the last five, best, and worst iterations for L_{conv} and L_{mlp}. The worst values (red) are always lower than the last-five iterations (orange) and also lower than the best values (green) for both *CHI* and κ. We conclude that using *CHI* values computed from the produced 2D feature space in training set and using pseudo-labels (even prone to errors) to select the best and worst iterations yields good classification results on the test set. This supports the choice of the best iteration of *conf-DeepFA* looping even for different datasets. Also, we notice that higher *CHI* values lead to higher κ values. For example, for MNIST, both *CHI* and κ are higher for the L_{conv} setup than for the L_{mlp} setup. We also see that L_{mlp} yields to higher *CHI* (and κ) values than L_{conv} for all datasets, except for MNIST. **Added-value of *CHI* evaluation in *conf-DeepFA*:** Fig. 10 plots, for each dataset, the best κ chosen by the best *CHI* value

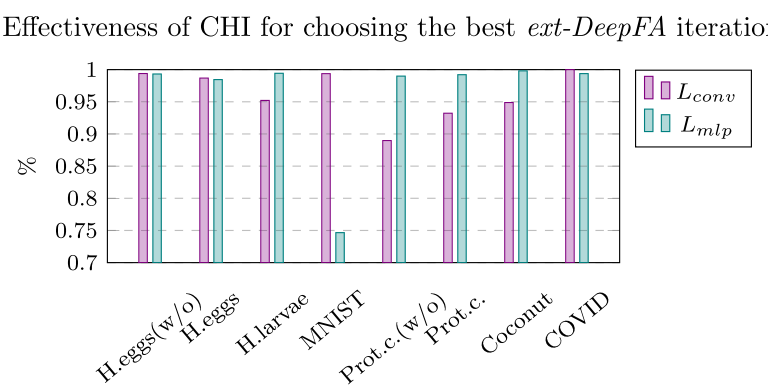


Fig. 10. Q6) Effectiveness of using CHI for choosing the best iteration in *ext-DeepFA* given by the best κ chosen by the best CHI in each iteration over the best possible κ in the test set.

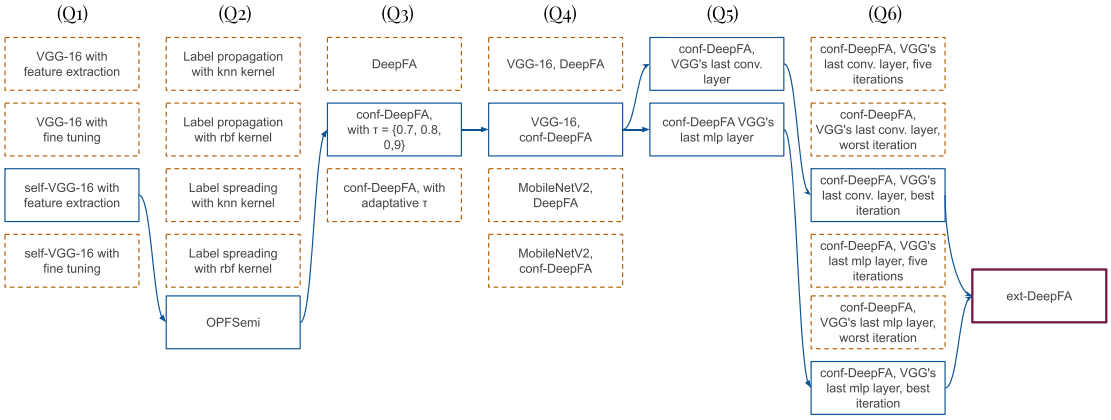


Fig. 11. Brief summary of the raised questions. For each addressed question, the compared experiments (orange, dashed) and the best result (blue, solid) are presented. Each resulting answer leads and composes the final *ext-DeepFA*. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

divided by the highest κ obtained on the test set for both L_{conv} and L_{mlp}. We see that the *CHI*-based selection approach achieves at least 95% of the best possible result in the test set for all datasets. This confirms the added-value of using *CHI* for choosing the best iteration of *conf-DeepFA*.

6. Answers to the studied questions

Fig. 11 summarizes our targeted questions, performed experiments, and best obtained results per question. We evaluate (Q1) the feature space generated by VGG-16 by feature extraction and fine-tuning strategies and compare (Q2) OPFSemi label propagation with other semi-supervised methods within the annotation looping. Next, we include (Q3) a confidence sampling strategy to OPFSemi's pseudo-labeling to define the most confident samples for training VGG-16 and evaluate (Q4) a feature space produced by MobileNetV2, a recent deep architecture with much fewer parameters to optimize. We also compare (Q5) two feature spaces provided by distinct layers of the deep network and investigate (Q6) the usage of a clustering metric to choose the best iteration in the proposed looping.

Considering the proposed experimental setup for *orig-DeepFA*, *conf-DeepFA*, and *ext-DeepFA* evaluation, we show that (Q1) self-trained VGG-16 models based on feature extraction only are usually better than fine-tuned models. (Q2) OPFSemi label propagation yielded the best performance on the 2D projected space for all tested datasets. (Q3) Confidence-based sampling showed clear added value, with different confidence values for each dataset though. (Q4) MobileNetV2 was not able to learn a feature space in which OPFSemi propagated better labels than VGG-16. (Q5) Propagating labels in different layers of the deep architecture led to different label propagation and classification accuracies depending on the dataset. (Q6) High values of the chosen clustering metric consistently led to the best propagation accuracy and classification results.

7. Limitations

We can split our limitations into those related to the experimental validation and the proposed technique. In validating our work, we explored eight datasets (from toy to real scenarios), two deep-learning approaches (VGG-16 and MobileNetV2), five iterations of the deep annotation loop, and one projection method (t-SNE). Exploring more combinations of such techniques is definitely of extra added value. Using more than five looping iterations could help to understand how much the learned feature space can be improved.

Related to the limitations of the proposed technique, we see that the selection of the confidence threshold τ and layer depth may vary on the dataset. To solve that observed dataset dependency, we plan to include user knowledge to select those values and layers. Although we have shown that the 2d projection provided by the t-SNE algorithm is suitable to offer relevant information to OPFSemi's label propagation, the t-SNE projection errors were not considered to improve the performed label propagation and feature learning. Additionally, using the t-SNE algorithm can be an issue due to its scalability: projecting more than dozens of thousands of samples can cost minutes. We intend to analyze

further the impact of the t-SNE projection errors and other projection techniques in our proposed pipeline.

8. Conclusion

We proposed an approach for labeling unsupervised samples and increasing the quality of image classification and extracted feature spaces when using very few supervised samples and many unsupervised ones for training. To cover the weaknesses of an existing earlier approach, we designed a new approach by evaluating six questions that aim to explore the space of possible improvements. Our investigation led to several findings. First, we showed that OPFSemi's label propagation by minimum graph paths conducted over t-SNE projections is better than other label propagation methods, even those using graphs, neighborhood distances, or kernel tricks. Using a confidence sampling strategy, we selected the best-labeled samples to retrain the model and minimized the effect of wrongly assigned labels in the learned feature space. To our knowledge, we used for the first time the Calinski-Harabasz Index (CHI) to evaluate the learned feature space and pseudo-labels of t-SNE projected features in a 2D space. CHI reveals a correlation among the best 2D feature space, best pseudo-labels, and best classification results, even if these are pseudo-labels prone to errors. When using CHI values to choose the best iteration, we achieve at least 95% of the best κ value achievable in the test set. This insight opens new ways for using CHI as an evaluation strategy in the learned and projected feature space and its pseudo labels.

To solve the observed dataset dependency in the selection of the confidence threshold τ and layer depth, we plan next to include user knowledge to provide a semi-automatic pseudo-labeling along the lines in [11], but now considering the proposed looping of the deep feature annotation method. Also, considering CHI can provide relevant information about the deep annotated features to support the user in a semi-automatic fashion. We intend to consider self-supervised learning strategies in deep feature learning to improve the learned feature space. Additionally, we plan to evaluate other projection methods (beyond t-SNE) with high neighborhood preservation qualities for potential label propagation improvement. Ultimately, we expect that our automatic-and-interactive deep feature-based pseudo-labeling combination will lead to higher quality and more explainable deep learning methods.

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

The authors acknowledge FAPESP grants #2014/12236 – 1, #2019/10705 – 8, #2022/12668-5, CAPES grants with Finance Code 001, and CNPq grants #303808/2018 – 7.

References

[1] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, C.L. Zitnick, Microsoft COCO: Common objects in context, in: Proc. ECCV, 2014, pp. 740–755.
[2] C. Sun, A. Shrivastava, S. Singh, A. Gupta, Revisiting unreasonable effectiveness of data in deep learning era, in: Proc. ICCV, 2017, pp. 843–852.
[3] J. Bergstra, Y. Bengio, Random search for hyper-parameter optimization, J. Mach. Learn. Res. 13 (null) (2012) 281–305.
[4] D.H. Lee, Pseudo-label : The simple and efficient semi-supervised learning method for deep neural networks, in: Proc. ICML-WREPL, 2013.
[5] T. Miyato, S.-i. Maeda, M. Koyama, S. Ishii, Virtual adversarial training: a regularization method for supervised and semi-supervised learning, IEEE PAMI 41 (8) (2018) 1979–1993.
[6] L. Jing, Y. Tian, Self-supervised visual feature learning with deep neural networks: a survey, IEEE PAMI (2020), doi:10.1109/TPAMI.2020.2992393. 1–1
[7] H. Pham, Z. Dai, Q. Xie, Q.V. Le, Meta pseudo labels, in: Proc. CVPR, 2021, pp. 11557–11568.
[8] W. Amorim, A. Falcão, J. Papa, M. Carvalho, Improving semi-supervised learning through optimum connectivity, Pattern Recognit. 60 (2016) 72–85.
[9] B.C. Benato, A.C. Telea, A.X. Falcão, Semi-supervised learning with interactive label propagation guided by feature space projections, in: Proc. SIBGRAPI, 2018, pp. 392–399.
[10] W. Amorim, G. Rosa, Rogério, J. Castanho, F. Dotto, O. Rodrigues, A. Marana, J. Papa, Semi-supervised learning with connectivity-driven convolutional neural networks, Pattern Recognit. Lett. 128 (2019) 16–22.
[11] B.C. Benato, J.F. Gomes, A.C. Telea, A.X. Falcão, Semi-automatic data annotation guided by feature space projection, Pattern Recognit. 109 (2021) 107612.
[12] B.C. Benato, J.F. Gomes, A.C. Telea, A.X. Falcão, Semi-supervised deep learning based on label propagation in a 2D embedded space, in: Proc. CIARP, Springer, 2021, pp. 371–381.
[13] V. Sindhwani, P. Niyogi, M. Belkin, Beyond the point cloud: From transductive to semi-supervised learning, in: Proc. ICML, 2005, pp. 824–831.
[14] L. van der Maaten, Accelerating t-SNE using tree-based algorithms, J. Mach. Learn. Res. 15 (1) (2014) 3221–3245.
[15] B.C. Benato, A.C. Telea, A.X. Falcao, Iterative pseudo-labeling with deep feature annotation and confidence-based sampling, in: Proc. SIBGRAPI, IEEE, 2021, pp. 192–198.
[16] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, 2014. arxiv.org/abs/1409.1556.
[17] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, L.-C. Chen, Mobilenetv2: Inverted residuals and linear bottlenecks, in: Proc. CVPR, 2018, pp. 4510–4520.
[18] Y. Wang, Q. Yao, J.T. Kwok, L.M. Ni, Generalizing from a few examples: a survey on few-shot learning, ACM Comput. Surv. 53 (3) (2020) 1–34.
[19] Z. Yu, L. Chen, Z. Cheng, J. Luo, Transmatch: a transfer-learning scheme for semi-supervised few-shot learning, CVPR, 2020.
[20] Y. Li, X. Chao, Semi-supervised few-shot learning approach for plant diseases recognition, Plant Methods 17 (2021) 1–10.
[21] A. Zhmoginov, M. Sandler, M. Vladymyrov, Hypertransformer: model generation for supervised and semi-supervised few-shot learning, in: International Conference on Machine Learning, PMLR, 2022, pp. 27075–27098.
[22] A. Iscen, G. Tolias, Y. Avrithis, O. Chum, Label propagation for deep semi-supervised learning, in: Proc. ICCV, 2019, pp. 5070–5079.
[23] E. Arazo, D. Ortego, P. Albert, N.E. O'Connor, K. McGuinness, Pseudo-labeling and confirmation bias in deep semi-supervised learning, in: Proc. IJCNN, IEEE, 2020, pp. 1–8.
[24] W. Shi, Y. Gong, C. Ding, Z.M. Tao, N. Zheng, Transductive semi-supervised deep learning using min-max features, in: Proc. ECCV, 2018, pp. 299–315.
[25] X. Zhai, A. Oliver, A. Kolesnikov, L. Beyer, S4l: Self-supervised semi-supervised learning, in: Proc. ICCV, 2019, pp. 1476–1485.
[26] P. Cascante-Bonilla, F. Tan, Y. Qi, V. Ordonez, Curriculum labeling: revisiting pseudo-labeling for semi-supervised learning, arXiv preprint arXiv:2001.06001 (2020).
[27] N. Das, S. Chaba, R. Wu, S. Gandhi, D.H. Chau, X. Chu, GOGGLES: Automatic image labeling with affinity coding, in: Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data, in: SIGMOD '20, Association for Computing Machinery, 2020, pp. 1717–1732.
[28] R. Wu, N. Das, S. Chaba, S. Gandhi, D.H. Chau, X. Chu, A cluster-then-label approach for few-shot learning with application to automatic image data labeling, J. Data and Information Quality 14 (3) (2022).
[29] B.C. Benato, A.X. Falcao, A.C. Telea, Linking data separation, visual separation, and classifier performance using pseudo-labeling by contrastive learning, in: Proc. VISAPP, IEEE, 2023 (to appear).
[30] M. Espadoto, R. Martins, A. Kerren, N. Hirata, A. Telea, Toward a quantitative survey of dimension reduction techniques, IEEE TVC 27 (3) (2019) 2153–2173.
[31] Z. Tian, X. Zhai, G. van Steenpaal, L. Yu, E. Dimara, M. Espadoto, A. Telea, Quantitative and qualitative comparison of 2D and 3D projection techniques for high-dimensional data, Information 12 (2021).
[32] Y. LeCun, C. Cortes, MNIST handwritten digit database(2010). yann.lecun.com/exdb/mnist.
[33] C. Suzuki, J. Gomes, A. Falcão, S. Shimizu, J. Papa, Automated diagnosis of human intestinal parasites using optical microscopy images, in: Proc. Symp. Biomedical Imaging, 2013, pp. 460–463.
[34] J.E. Vargas-Muñoz, P. Zhou, A.X. Falcão, D. Tuia, Interactive coconut tree annotation using feature space projections, in: Proc. IGARSS, 2019, pp. 5718–5721, doi:10.1109/IGARSS.2019.8899005.
[35] M.E. Chowdhury, T. Rahman, A. Khandakar, R. Mazhar, M.A. Kadir, Z.B. Mahbub, K.R. Islam, M.S. Khan, A. Iqbal, N. Al Emadi, et al., Can ai help in screening viral and covid-19 pneumonia? IEEE Access 8 (2020) 132665–132676.
[36] T. Rahman, A. Khandakar, Y. Qiblawey, A. Tahir, S. Kiranyaz, S.B.A. Kashem, M.T. Islam, S. Al Maadeed, S.M. Zughaier, M.S. Khan, et al., Exploring the effect of image enhancement techniques on covid-19 detection using chest x-ray images, Comput. Biol. Med. 132 (2021) 104319.

[37] F. Chollet, et al., Keras, 2015, (https://keras.io).
[38] P. Rauber, A. Falcão, A. Telea, Projections as visual aids for classification system design, Inf Vis (2017).
[39] T. Caliński, J. Harabasz, A dendrite method for cluster analysis, Commun. Stat. 3 (1) (1974) 1–27.

Bárbara C. Benato received her M.Sc. (2019) in Computer Science from University of Campinas, Brazil. She is currently a Ph.D. student in Computer Science at University of Campinas, Brazil. Her research interests include machine learning, deep learning, pattern recognition and visual analytics applications in machine learning.

Alexandru C. Telea received his Ph.D. (2000) in Computer Science from the Eindhoven University of Technology, the Netherlands. He is currently a professor in visual data analytics at the Department of Information and Computing Sciences, Utrecht University. His interests include 3D multiscale shape processing, information visualization, software visualization, machine learning, and visual analytics.

Alexandre X. Falcão received his Ph.D. (1996) in Electrical Engineering from the University of Campinas, Brazil. He is a Professor in Computer Science at the Institute of Computing, University of Campinas and his research interests include image processing and analysis, machine learning and pattern recognition, data visualization, medical imaging and remote sensing applications.