A Self-Training Hierarchical Prototype-Based Approach for Semi-Supervised Classification

Xiaowei Gu

School of Computing and Communications, Lancaster University, Lancaster, UK

Email: x.gu3@lancaster.ac.uk; xw.gu@hotmail.com

Abstract: This paper introduces a novel self-training hierarchical prototype-based approach for semi-supervised classification. The proposed approach firstly identifies meaningful prototypes from labelled samples at multiple levels of granularity and, then, self-organizes a highly transparent, multi-layered recognition model by arranging them in a form of pyramidal hierarchies. After this, the learning model continues to self-evolve its structure and self-expand its knowledge base to incorporate new patterns recognized from unlabelled samples by exploiting the pseudo-label technique. Thanks to its prototype-based nature, the overall computational process of the proposed approach is highly explainable and traceable. Experimental studies with various benchmark image recognition problems demonstrate the state-of-the-art performance of the proposed approach, showing its strong capability to mine key information from unlabelled data for classification.

Keywords: self-training, prototype-based, hierarchical structure, semi-supervised learning, classification

1. Introduction

Classification is often considered as a supervised machine learning technique for predicting class labels of new observations. Traditional classification approaches use only labelled data to build recognition models. In reality, however, labelled samples are scarce and expensive to obtain [15], which usually requires slow human annotation and expensive laboratory experiments. Although unlabelled samples are plentiful and relatively easier to collect, supervised learning approaches are unable to utilize them [44],[48]. Semi-supervised learning approaches, on the other hand, go beyond traditional supervised learning and overcome the labelling bottleneck by involving a great amount of unlabelled data together with labelled ones to build stronger recognition models [37]. As semi-supervised learning approaches require less human labour and can exhibit better classification performance, they have been increasingly explored both in theory and in practice [4],[48].

Self-training is a simple and effective semi-supervised learning methodology following the idea of "pseudo label" [18],[23]. A standard self-training algorithm usually employs a mainstream classifier, such as neural network (NN) [18],[41], k-nearest neighbour (KNN) [14],[35], support vector machine (SVM) [19],[23],[24] or decision tree (DT) [23],[33], etc., as the base learner and iteratively enlarges the labelled training set with high-confidence predictions by its base learner, namely, pseudo-labelled samples [33]. However, standard self-training approaches require the base classifier to be fully retrained with both labelled and pseudo-labelled data in each iteration [40], which is computationally complex. Moreover, misclassifying a number of unlabelled samples is always inevitable during the self-training process because of insufficient labelled samples. Error detection and label modification usually are unable to correct all mistakes due to the lack of information [40]. Pseudo-labelling errors can easily propagate within the base classifier due to the iterative computational process, which, in turn, may significantly deteriorate classification performance. Another problem worthy of attention is that the generic classifiers employed by standard self-training approaches lack transparency and human-interpretability. Although complexity measures [12] can be applied to data for disclosing the geometrical characteristics of the class distributions in the original data space (as well as in the kernel space for SVM), NN and SVM are widely recognized as typical "black box" models. KNN and DT are also extremely hard to interpret when dealing with high-dimensional, large-scale, and complex problems. This makes the overall self-training process opaque. In addition, there is no easy way to fix the classifier when error occurs. To overcome these issues, one feasible solution is to replace the mainstream classifiers with a more advanced one, which has a transparent system structure and is capable of learning from data in a non-iterative, "one pass" manner.

In this paper, a novel self-training hierarchal prototype-based (STHP) approach is proposed for semi-supervised classification. The proposed approach employs the recently introduced hierarchical prototype-based (HP) classifier [9] as its base learner. Primed with a small amount of labelled training samples, the STHP classifier is able to pseudo-label unlabelled samples in a nature way by following the "nearest prototype" principle. It can continuously self-develop its prototype-based system structure without human supervision by identifying new prototypes from pseudo-labelled samples and aggregating them into the pyramidal hierarchies in a non-iterative manner. Thanks to its prototype-based nature and multi-layered system structure, the STHP classifier can offer higher transparency and human-interpretability than the state-of-the-art alternatives. Numerical examples on benchmark image datasets demonstrate the effectiveness and validity of the proposed approach as a powerful semi-supervised learning tool.

Key contributions of this paper include: (1) a new approach that can self-organize and self-develop its prototype-based hierarchical system structure from both labelled and unlabelled samples; (2) a fully explainable self-training paradigm for semi-supervised classification based on the pseudo-label technique; (3) the capability to visualize the learned knowledge base in a prototype-based hierarchal form.

The remainder of this paper is organized as follows. Section 2 provides a review of related works. The architecture, learning and decision-making processes of the recently introduced HP approach are summarized in Section 3. Section 4 presents technique details of the proposed approach followed by the computational complexity analysis given in Section 5. Experimental investigation is provided in Section 6 as the proof of concept. Section 7 concludes this paper and gives directions for future work.

2. Related Works

Semi-supervised learning is a hybrid machine learning technique combining elements of both supervised and unsupervised learning [14]. Thanks to its appealing capability to enhance classification models with unlabelled samples, semi-supervised learning is a hotly researched topic and has attracted extensive attentions in the recent decades [36],[39].

To effectively utilize unlabelled data, semi-supervised learning algorithms commonly exploit two semi-supervised assumptions, namely, the cluster assumption and the manifold assumption. Based on the underlying assumptions, existing algorithms in the literature may be categorized into three categories, which include cluster-based, manifold-based and ensemble learning [23]. A brief summary of mainstream semi-supervised approaches of the three categories is provided in Table 1.

Cluster-based approaches [20],[25] aim to make the decision boundaries between different classes pass through low-density regions and simultaneously maximize the margins between clusters. Well-known approaches of this category include: transductive support vector machine (TSVM) [34], semi-supervised support vector machine (S3VM) [2],[21], semi-supervised support vector machine using label mean (MeanS3VM) [20] and cluster-based regularization (ClusterReg) [31], etc. Manifold-based approaches [13],[16],[22],[45],[49] attempt to learn a low-dimensional manifold structure from the original input space to build a maximum-margin classifier. Mainstream manifold-based approaches include, but are not limited to, graph mincut [3], Gaussian field and harmonic function (GHF) [49], local and global consistency (LGC) [46], Laplacian support vector machine (LapSVM) [1] and anchor graph regularization (AnchorGraphReg) [22],[37]. Compared with cluster-based approaches, manifold-based approaches demonstrate stronger performance and are easier to implement [45], thus, they have gained more popularity. On the other hand, manifold-based approaches, in general, are highly computational complex and limited to small scale problems.

The third category is the ensemble learning methods [18],[23],[35],[47], which are developed from two semi-supervised assumptions. The most widely used ensemble learning methodologies include co-training and self-training. Co-training [28],[47] assumes that the feature space can be split into multiple conditionally independent and sufficient views (namely, sub-feature sets). In a standard co-training framework, multiple classifiers are firstly trained with labelled samples on the respective views and, then, each classifier uses its predictions on unlabelled samples to augment the training sets of others. However, traditional co-training relies on the strict requirement of different, conditionally independent views, which is rarely satisfied in real problems [28],[48]. Self-training [18],

as its name suggests, attempts to iteratively enlarge its labelled training set using unlabelled samples. During a standard self-training process, a base classifier is firstly trained with a small number of labelled samples. Then, the trained model is used for classifying the unlabelled samples [42]. The unlabelled samples with the highest classification confidence are selected out and assigned with class labels predicted by the base classifier. These class labels are so-called "pseudo labels". After this, the classifier is retrained with both the labelled and pseudo-labelled sets, and the procedure is repeated. In short, the classifier uses its own prediction to improve its classification effectiveness. Compared with other semi-supervised learning strategies, self-training [18] is simpler and does not impose any assumptions on the data generation model with user- and problem-specific parameters. Therefore, it has been successfully applied in many real-world scenarios [14],[19],[24]. However, as mentioned in Section 1, standard self-training approaches suffer from the problems of high computational complexity and error propagation caused by the iterative computational process. In addition, the transparency and explainability of mainstream classifiers employed by self-training approaches are also limited, especially for high-dimensional, large-scale and complex problems.

Table 1 Summary of mainstream semi-supervised classification algorithms

Category	Algorithm	Summary
Cluster-based (cluster assumption)	TSVM [34], S3VM [2],[21]	Regularize the decision boundaries and maximize the margins using unlabelled data.
	MeanS3VM [20]	Estimate label means of unlabelled samples and, then, maximize the margins between the label means.
	ClusterReg [31]	Regularize the base learner, e.g., a NN, with the posterior probability obtained by a clustering algorithm.
Manifold-based (manifold assumption)	Graph mincut [3]	Construct a graph from training samples and find the minimum cut on the graph by minimizing a quadratic loss function for classifying unlabelled samples.
	GHF [49], LGC [46]	Propagate label information from labelled samples to unlabelled samples over the graph Laplacian constructed from all training samples.
	LapSVM [1]	Regularize a standard SVM with the graph Laplacian constructed from all training samples.
	AnchorGraphReg [22],[37]	Select a small number of anchor points to approximately cover all training samples and construct an anchor graph with the training samples and anchor points.
Ensemble learning (cluster and manifold assumptions)	Co-training [28],[47]	Train multiple classifiers from different conditionally independent views of labelled samples and then let the trained classifiers to teach each other through classifying unlabelled samples.
	Self-training [14],[18],[19],[23],[24],[33],[38], [40]-[42] (and the proposed)	Train a base classifier from lablled samples and then, retrain the classifier with the enlarged labelled training set by its own most confident preductions.

The semi-supervised deep rule-based (SSDRB) approach presented in [7] attempts to tackle the aforementioned issues by using a deep rule-based (DRB) classifier [8] as its base learner for self-training. DRB is a zero-order evolving intelligent system (EIS) with a multi-layered architecture designed specifically for image classification. As a typical type of prototype-based models, zero-order EISs have been widely used for multi-class classification tasks. Compared with other mainstream classifiers (e.g., NN and SVM), zero-order EISs offer much higher transparency and human-interpretability, and can learn from streaming data on a sample-by-sample basis [30]. Nonetheless, it is frequently observed that zero-order EISs can be unfavourably obese and uninterpretable for

large-scale complex problems [11]. Under such circumstances, the transparency and explainability of zero-order EISs are also very limited.

The recently introduced HP classifier [9] is a generic approach for classification. It naturally simplifies complex problems by decomposing them into a series of local models, which are represented by meaningful prototypes. These prototypes are identified directly from data based on their mutual distances and ensemble properties; they represent the local peaks of multimodal distributions observed at multiple levels of granularity/specificity. The identified prototypes are naturally aggregated in the form of pyramidal hierarchies with meaningful links between successive layers. The HP classifier is capable of continuously self-evolving to capture new patterns in streaming data in a "one pass", computationally lean manner. More importantly, the rationales behind any decisions it makes can be explained clearly because its learning and decision-making processes strictly follows the "nearest prototype" principle.

This paper further extends the HP classifier with a self-training mechanism resulting in the STHP approach. The proposed approach is able to recursively self-update its system structure and meta-parameters using the pseudo-labelled samples without a full retraining or any iterative computation. Thus, error propagation can be effectively prevented, and the computational complexity is kept in a low level. A comparison between different base learners (the HP classifier and the mainstream ones) used by self-training approaches is summarized in Table 2.

Table 2. Comparison between different base classifiers

Base learners	Generic	Prototype-based	Online learning	Recursive updating	System transparency
SVM	Yes	Yes	No	No	Low
KNN	Yes	Yes			Depending on the problem
NN	Yes	No	No	No	Low
DT	Yes	No	No	No	Depending on the problem
DRB	No (for image classification only)	Yes	Yes	Yes	Depending on the problem
HP	Yes	Yes	Yes	Yes	High

In the next two sections, a summarization of technical details of the HP classifier will be provided, followed by the detailed description of the algorithmic procedure of the proposed STHP approach.

3. The HP Classifier

In this section, the general architecture, supervised learning and decision-making processes of the HP classifier [9] are briefly recalled to make this paper self-contained.

First of all, let $\boldsymbol{x}_k = \left[x_{k,1}, x_{k,2}, \ldots, x_{k,N}\right]^T$ be a particular sample in the N-dimensional data space, $\mathbf{R}^N$ and y_k be the corresponding class label, $y_k \in \{1,2, \ldots, C\}$, where C is the number of classes. The labelled set, $\mathbf{X}_L$ is composed of L samples, namely, $\mathbf{X}_L = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_L\}$ with known class labels, $\mathbf{Y}_L = \{y_1, y_2, \ldots, y_L\}$, and the unlabelled set, $\mathbf{X}_U$ is composed of U samples with unknown class labels, namely, $\mathbf{X}_U = \{\boldsymbol{x}_{L+1}, \boldsymbol{x}_{L+2}, \ldots, \boldsymbol{x}_{L+U}\}$. The labelled set can be further divided into C subsets according to their class labels, namely, $\mathbf{X}_L^i = \{\boldsymbol{x}_1^i, \boldsymbol{x}_2^i, \ldots, \boldsymbol{x}_{L^i}^i\}$ and $\mathbf{Y}_L^i = \left\{\overbrace{i, i, \ldots, i}^{L^i}\right\}$ $(i = 1,2 \ldots, C)$, and there is $\sum_{i=1}^{C} L^i = L$. In particular, the number of unlabelled samples is assumed to be much larger than the number of labelled samples, namely, $U \gg L$.

3.1. General architecture

The general architecture of the HP classifier is given in Fig. 1 [9]. The HP classifier consists of C prototype-based pyramidal hierarchies. Each hierarchy is self-organized from labelled samples of a particular class (one hierarchy per class). The zoom-in structure of the ith $(i = 1,2, \ldots, C)$ hierarchy is given in Fig. 1 as well, where $\boldsymbol{p}_{h,j}^i$ denotes

the jth prototype at the hth layer of the hierarchy; $h = 1,2,\dots,H$; H is the layer number; $j = 1,2,\dots,M_h^i$; M_h^i is the number of prototypes at the hth layer and there are $M_1^i \le \cdots \le M_h^i \le \cdots \le M_H^i$; $1 \le m_1 \le m_2 \le M_2^i$; and $1 \le m_3 \le m_4 \le m_5 \le M_H^i$. Without loss of generality, in this paper, the C hierarchies of the HP classifier have the same number of layers, namely, H [9].

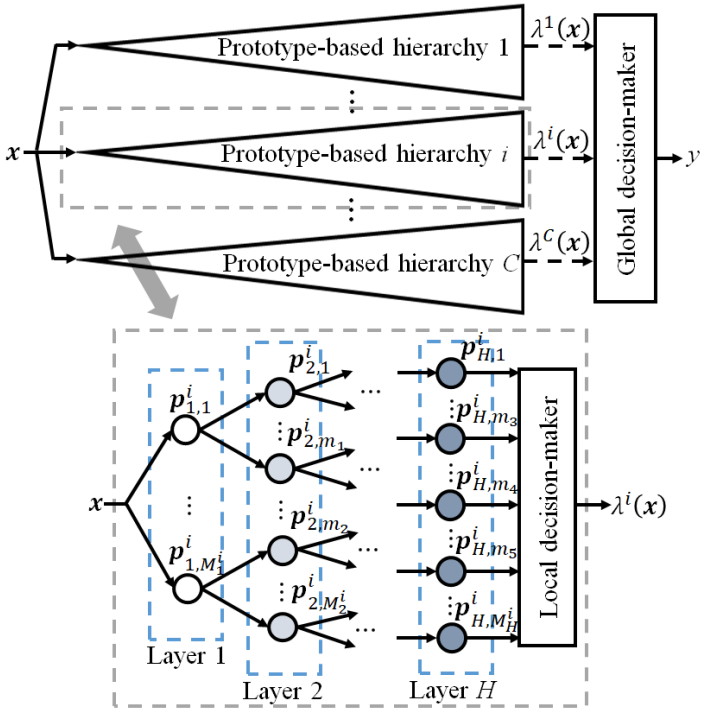


Fig. 1. General architecture of the HP classifier.

3.2. Supervised learning process

This subsection briefly recalls the supervised learning procedure of the HP classifier [9]. Since the prototype-based hierarchies are identified from labelled samples of each class separately, the learning process of the ith hierarchy is presented to avoid repetition ($i = 1,2,\dots,C$). The same principles can be applied to all other hierarchies within the classifier. Each observed sample, $\boldsymbol{x}$ is normalized by its Euclidean norm by default:

$$\boldsymbol{x} \leftarrow \frac{\boldsymbol{x}}{\|\boldsymbol{x}\|} \tag{1}$$

where $\|\boldsymbol{x}\| = \sqrt{\sum_{k=1}^{N} x_k^2}$. Following the mode of [9], the radii of area of influence, r_h around prototypes at different layers of the hierarchies are determined by the following expression ($h = 1,2,\dots,H$):

$$r_h = 2\left(1 - cos\left(\frac{\theta_o}{2^{h-1}}\right)\right) \tag{2}$$

In this paper, $\theta_o = \frac{\pi}{2}$ as a default setting. Note that the radii r_h ($h = 1,2,\dots,H$) are not problem- or user- specific parameters and can be decided without prior knowledge. One may specify the radius value setting based on preferences and specific requirements of the problems with the following constraint: $r_1 > r_2 > \cdots > r_H$. The supervised learning process is summarized as follows [9].

Stage 0. System initialization

The first sample, $\boldsymbol{x}_k^i \in \mathbf{X}_L^i$ ($k = 1$) of the ith class is used for initializing the hierarchy as the first prototype at each layer ($h = 1,2,\dots,H$):

$$M_h^i \leftarrow 1; \quad \boldsymbol{p}_{h,M_h^i}^i \leftarrow \boldsymbol{x}_k^i; \quad S_{h,M_h^i}^i \leftarrow 1 \tag{3}$$

where M_h^i denotes the number of prototypes at the *h*th layer; $S_{h,M_h^i}^i$ is the support of the prototype, $\boldsymbol{p}_{h,M_h^i}^i$, namely, the number of samples associated with $\boldsymbol{p}_{h,M_h^i}^i$.

The collection of apex prototypes, $\boldsymbol{\mathcal{L}}_0^i$ is defined as ($h = 1$):

$$\boldsymbol{\mathcal{L}}_0^i \leftarrow \left\{\boldsymbol{p}_{h,M_h^i}^i\right\} \tag{4}$$

and, the hierarchy is established by building links between prototypes of successive layers ($h = 2,3,\dots,H$):

$$\boldsymbol{\mathcal{L}}_{h-1,M_{h-1}^i}^i \leftarrow \left\{\boldsymbol{p}_{h,M_h^i}^i\right\} \tag{5}$$

where $\boldsymbol{\mathcal{L}}_{h-1,M_{h-1}^i}^i$ is the collection of immediate subordinates associated with $\boldsymbol{p}_{h-1,M_{h-1}^i}^i$ at the *h*th layer. By identifying $\boldsymbol{\mathcal{L}}_0^i$ and $\boldsymbol{\mathcal{L}}_{h-1,M_{h-1}^i}^i (h = 2,3,\dots,H)$, the subordinate relationships between these prototypes are established and the prototype-based hierarchy is initialized.

Stage 1. System dynamically evolving

With the next available sample, $\boldsymbol{x}_k^i \in \mathbf{X}_L^i$ ($k \leftarrow k+1$), the system evolving process is performed in a top-down manner starting from the top layer, $h = 1$. Firstly, the nearest prototype to $\boldsymbol{x}_k^i$ at the *h*th layer, denoted by $\boldsymbol{p}_{h,n_h^*}^i$, is identified by the following equation:

$$\boldsymbol{p}_{h,n_h^*}^i = \begin{cases} \operatorname{argmin}_{\boldsymbol{p}\in\boldsymbol{\mathcal{L}}_0^i}\left(\left\|\boldsymbol{x}_k^i - \boldsymbol{p}\right\|\right), & if\ h = 1 \\ \operatorname{argmin}_{\boldsymbol{p}\in\boldsymbol{\mathcal{L}}_{h-1,n_{h-1}^*}^i}\left(\left\|\boldsymbol{x}_k^i - \boldsymbol{p}\right\|\right), & if\ h = 2,3,\dots,H \end{cases} \tag{6}$$

Equation (6) enables the algorithm to identify the nearest prototype at each layer from only the immediate subordinates of the nearest prototype at the above layer. This allows the nearest neighbouring searching process to be performed in an extremely efficient manner.

Then, **Condition 1** is checked to see whether $\boldsymbol{x}_k^i$ is sufficiently distinctive to other prototypes at the *h*th layer and has the potential to become a new prototype:

$$\boldsymbol{Condition\ 1}: \quad \begin{array}{c} If\left(\left\|\boldsymbol{x}_k^i - \boldsymbol{p}_{h,n_h^*}^i\right\|^2 > r_h\right) \\ Then\left(\boldsymbol{x}_k^i\ is\ a\ new\ prototype\ at\ the\ h^{th}\ layer\right) \end{array} \tag{7}$$

If **Condition 1** is satisfied, $\boldsymbol{x}_k^i$ becomes a new prototype at the *h*th layer as well as the successive lower layers with meta-parameters initialized by equation (8) ($j = h, h+1, \dots, H$):

$$M_j^i \leftarrow M_j^i + 1; \quad \boldsymbol{p}_{j,M_j^i}^i \leftarrow \boldsymbol{x}_k^i; \quad S_{j,M_j^i}^i \leftarrow 1 \tag{8}$$

The structure of the hierarchy, then, is updated by adding a new branch formed by these newly added prototypes. If $\boldsymbol{x}_k^i$ is recognized as a new apex prototype, namely ($h = 1$),

$$\boldsymbol{\mathcal{L}}_0^i \leftarrow \boldsymbol{\mathcal{L}}_0^i \cup \left\{\boldsymbol{p}_{h,M_h^i}^i\right\} \tag{9}$$

$\boldsymbol{x}_k^i$ itself is the starting node of this new branch. Otherwise, the nearest prototype at the layer above, $\boldsymbol{p}_{h-1,n_{h-1}^*}^i$ is recognized as the starting node with $\boldsymbol{\mathcal{L}}_{h-1,n_{h-1}^*}^i$ updated as ($h \geq 2$):

$$\boldsymbol{\mathcal{L}}_{h-1,n_{h-1}^*}^i \leftarrow \boldsymbol{\mathcal{L}}_{h-1,n_{h-1}^*}^i \cup \left\{\boldsymbol{p}_{h,M_h^i}^i\right\} \tag{10}$$

The links between the new prototypes, $\boldsymbol{p}^i_{h,M^i_h}$, $\boldsymbol{p}^i_{h+1,M^i_{h+1}}$,..., $\boldsymbol{p}^i_{H,M^i_H}$ are established by equation (5), and the current system structure updating cycle is completed.

Otherwise, if **Condition 1** is unsatisfied, $\boldsymbol{x}^i_k$ is associated with the nearest prototype, $\boldsymbol{p}^i_{h,n^*_h}$ and the meta-parameters of $\boldsymbol{p}^i_{h,n^*_h}$ are updated as:

$$\boldsymbol{p}^i_{h,n^*_h} \leftarrow \frac{S^i_{h,n^*_h}\boldsymbol{p}^i_{h,n^*_h}+\boldsymbol{x}^i_k}{S^i_{h,n^*_h}+1}; \quad \boldsymbol{p}^i_{h,n^*_h} \leftarrow \frac{\boldsymbol{p}^i_{h,n^*_h}}{\left\|\boldsymbol{p}^i_{h,n^*_h}\right\|}; \quad S^i_{h,n^*_h} \leftarrow S^i_{h,n^*_h}+1 \tag{11}$$

Then, $\boldsymbol{x}^i_k$ is passed to the next layer ($h \leftarrow h+1$) of the hierarchy and the same procedure starting from equation (6) is repeated until the bottom layer is updated or being interrupted when **Condition 1** is satisfied at a particular layer resulting in a new branch adding to the hierarchy. After the structure and/or meta-parameter updating with $\boldsymbol{x}^i_k$, **Stage 1** is repeated for the next available labelled sample ($k \leftarrow k+1$).

The supervised learning process of the *i*th prototype-based hierarchy is also summarized by the following pseudo code [9].

Algorithm 1: supervised prototype-based hierarchy identification

Input: $\mathbf{X}^i_L$
Algorithm begins

\\ *Stage 0. System initialization* \\
a. $k \leftarrow 1$;
b. Read $\boldsymbol{x}^i_k$ from $\mathbf{X}^i_L$;
c. Normalize $\boldsymbol{x}^i_k$ by (1);
i. **For** $h = 1$ to H **do**
1. Initialize M^i_h, $\boldsymbol{p}^i_{h,M^i_h}$ and S^i_{h,M^i_h} by (3);
2. **If** ($h = 1$) **then**
* Initialize $\boldsymbol{\mathcal{L}}^i_0$ by (4);
3. **Else**
* Initialize $\boldsymbol{\mathcal{L}}^i_{h-1,M^i_{h-1}}$ by (5);
4. **End if**
ii. **End for**
\\ *Stage 1. System dynamically evolving* \\
d. **While** ($k < L^i$)
i. $k \leftarrow k+1$;
ii. Read $\boldsymbol{x}^i_k$ from $\mathbf{X}^i_L$;
iii. Normalize $\boldsymbol{x}^i_k$ by (1);
iv. **For** $h = 1$ to H **do**
1. Identify $\boldsymbol{p}^i_{h,n^*_h}$ by (6);
2. **If** (**Condition 1** is satisfied) **then**
* **For** $j = h$ to H **do**
- Update M^i_j and initialize $\boldsymbol{p}^i_{j,M^i_j}$ and S^i_{j,M^i_j} by (8);
* **End for**
* **If** ($h = 1$) **then**
- Update $\boldsymbol{\mathcal{L}}^i_0$ by (9);
* **Else**
- Update $\boldsymbol{\mathcal{L}}^i_{h-1,n^*_{h-1}}$ by (10);
* **End**
* **For** $j = h+1$ to H **do**
- Initialize $\boldsymbol{\mathcal{L}}^i_{h-1,M^i_{h-1}}$ by (5);
* **End for**
* Break **for** loop**;**
3. **Else**

- Update $\boldsymbol{p}_{h,n_h^*}^i$ and $S_{h,n_h^*}^i$ by (11); 4. **End if** *v.* **End for** *e.* **End while**
Algorithm ends ***Output: the ith prototype-based hierarchy***

3.3. Decision-making process

During the decision-making process, for a given unlabelled sample, $\boldsymbol{x}$, the local decision-maker of each pyramidal hierarchy of the HP classifier will firstly produce a score of confidence, $\lambda^i(\boldsymbol{x})$, which is calculated based on the similarity between $\boldsymbol{x}$ and the nearest prototype, $\boldsymbol{p}_{h,n_h^*}^i$ at the selected layer (assuming the hth one) for classification ($i = 1,2,\dots,C$) [9]:

$$\lambda_h^i(\boldsymbol{x}) = \max_{k=1,2,\dots,M_h^i}\left(e^{-\left\|\boldsymbol{p}_{h,k}^i-\boldsymbol{x}\right\|^2}\right) = e^{-\left\|\boldsymbol{p}_{h,n_h^*}^i-\boldsymbol{x}\right\|^2} \tag{12}$$

Then, the global decision-maker determines the class label of $\boldsymbol{x}$ by the "winner takes all" principle based on the C scores of confidence (one per hierarchy) [8]:

$$y \leftarrow class\ i^*; \quad i^* = \text{argmax}_{i=1,2,\dots,C}\left(\lambda_h^i(\boldsymbol{x})\right) \tag{13}$$

4. Proposed Approach

In this section, the self-training mechanism for the proposed STHP approach is presented in detail. The aim of self-training is to involve the unlabelled set, $\mathbf{X}_U$ to train a better classifier primed by the labelled set, $\mathbf{X}_L$. The flowchart of the proposed framework is depicted in Fig. 2.

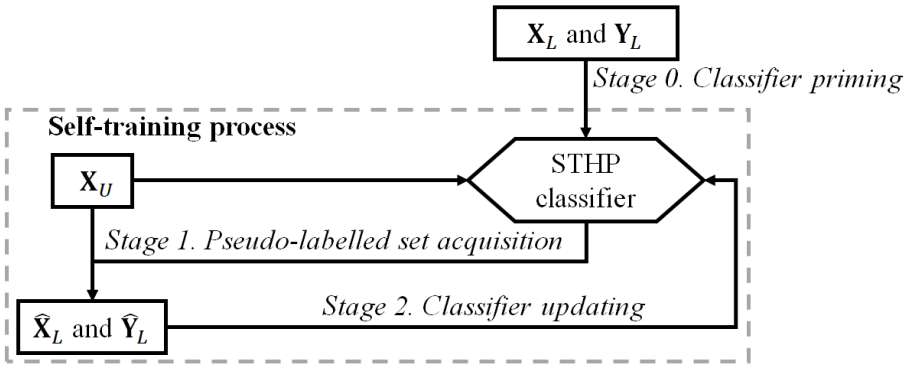


Fig.2. Flowchart of proposed self-training process.

One can see from Fig. 2 that the self-training process of the STHP classifier is divided into the following three stages. In **Stage 0**, the classifier is, firstly, primed with $\mathbf{X}_L$ and $\mathbf{Y}_L$ in a supervised manner. In **Stage 1**, the classifier compares $\mathbf{X}_U$ with its knowledge base and selects out a set of the most confident unlabelled samples, $\widehat{\mathbf{X}}_L$ from $\mathbf{X}_U$ with predicted class labels (pseudo-labels), $\widehat{\mathbf{Y}}_L$. Then, in **Stage 2**, the classifier self-develops its knowledge base using $\widehat{\mathbf{X}}_L$ and $\widehat{\mathbf{Y}}_L$. After this, the classifier goes back to **Stage 1** and starts a new learning cycle until no more suitable unlabelled samples can be utilized for system updating. The algorithmic procedure of the self-training process is described as follows.

Stage 0. Classifier priming

The STHP classifier is firstly trained with the labelled training samples, $\mathbf{X}_L$ and their labels, $\mathbf{Y}_L$ in a supervised manner using the same algorithmic procedure presented in subsection 3.2, resulting in C prototype-based hierarchies. Then, the classifier moves on to the next stage and the pseudo-labelling process starts.

Stage 1. Pseudo-labelled set acquisition

In this stage, a set of pseudo-labelled samples, $\widehat{\mathbf{X}}_L$ and the corresponding pseudo labels, $\widehat{\mathbf{Y}}_L$ will be selected out from $\mathbf{X}_U$ for updating the classifier. Before the selection process starts, $\widehat{\mathbf{X}}_L$ and $\widehat{\mathbf{Y}}_L$ are both initialized as: $\widehat{\mathbf{X}}_L \leftarrow \emptyset$ and $\widehat{\mathbf{Y}}_L \leftarrow \emptyset$.

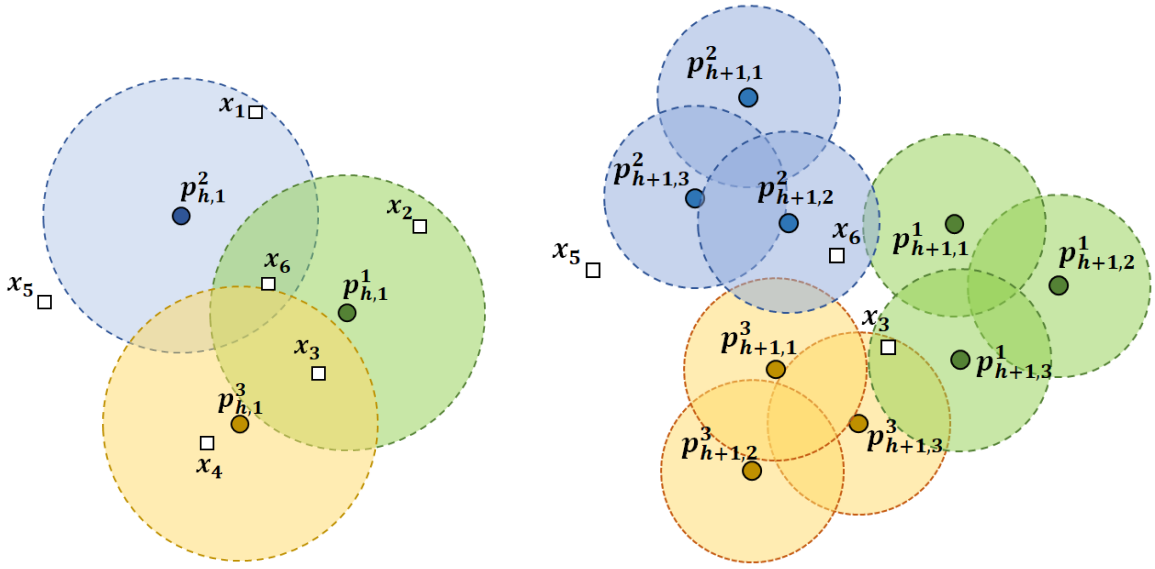
Then, for each unlabelled sample, $\boldsymbol{x}_k \in \mathbf{X}_U$ $(k = 1,2,\dots,U)$, it is compared with prototypes of the C hierarchies in a top-down manner starting from the top layer ($h = 1$). The following condition is examined to see whether $\boldsymbol{x}_k$ can be used for updating the classifier:

$$\textbf{\textit{Condition 2}}: \quad \begin{array}{l} If\ \left(\left\|\boldsymbol{x}_k - \boldsymbol{p}_{h,n_h^*}^{i}\right\|^2 < r_h\right)\ and\ \left(\min_{j=1,2,\dots,C;i\neq j}\left(\left\|\boldsymbol{x}_k - \boldsymbol{p}_{h,n_h^*}^{j}\right\|^2\right) > r_h\right) \\ Then\ \left(\widehat{\mathbf{X}}_L \leftarrow \widehat{\mathbf{X}}_L \cup \{\boldsymbol{x}_k\}\right)\ and\ \left(\widehat{\mathbf{Y}}_L \leftarrow \widehat{\mathbf{Y}}_L \cup \{\hat{y}_k = i\}\right)\ and\ \left(\mathbf{X}_U \leftarrow \mathbf{X}_U \backslash \{\boldsymbol{x}_k\}\right) \end{array} \tag{14}$$

where $\boldsymbol{p}_{h,n_h^*}^{j}$ is the nearest prototype to $\boldsymbol{x}_k$ at the *h*th layer of the *j*th hierarchy identified by equation (6). If **Condition 2** is satisfied at the *h*th layer, it means that $\boldsymbol{x}_k$ is closely associated with an existing data pattern observed from samples of the *i*th class and is distinctive from prototypes of other classes. In this case, the classifier is able to assign a class label $\hat{y}_k$ $(\hat{y}_k = i)$ to $\boldsymbol{x}_k$ with high confidence and, $\boldsymbol{x}_k$ will be used for updating the classifier in the next stage. Otherwise, $\boldsymbol{x}_k$ is passed to the next layer ($h \leftarrow h + 1$) and the same examining process is repeated.

The rationale behind **Condition 2** is very straightforward. Each prototype of the classifier has an area of influence in the form of a hypersphere occupying a part of the data space. If an unlabelled sample, $\boldsymbol{x}_k$ locates at the area of influence of a single prototype or the overlapping areas occupied by multiple prototypes of the same class, there is a very high likelihood that $\boldsymbol{x}_k$ belongs to that class as well. The reason for examining **Condition 2** at each layer in a "top-down" manner comes from the fact that prototypes at different layers are identified from data at different levels of granularity. Upper-layer prototypes contain highly generalized information of data, and they approximate the main data patterns. Lower-layer prototypes, on the other hand, contain finer details, and they represent the local models of data. These unlabelled samples satisfying **Condition 2** at lower layers are of great importance for the classifier to learn more precise decision boundaries for classification. Therefore, it is worthwhile examining **Condition 2** at different layers for each unlabelled sample.

An illustrative example is given by Fig. 3 to demonstrate the idea, where large dots "●" in three different colours are the prototypes of three different classes; white squares "□" represent unlabelled samples; the shadow area surrounding each prototype is the corresponding area of influence. As shown by Fig. 3 (a), there are six unlabelled samples in the data space, $\mathbf{X}_U = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, \boldsymbol{x}_4, \boldsymbol{x}_5, \boldsymbol{x}_6\}$ and three prototypes belonging to three different classes, namely, $\boldsymbol{p}_{h,1}^1$, $\boldsymbol{p}_{h,1}^2$ and $\boldsymbol{p}_{h,1}^3$. $\boldsymbol{x}_1$, $\boldsymbol{x}_2$ and $\boldsymbol{x}_4$ locate at the areas of influence of $\boldsymbol{p}_{h,1}^1$, $\boldsymbol{p}_{h,1}^2$ and $\boldsymbol{p}_{h,1}^3$, respectively. Thus, $\boldsymbol{x}_1, \boldsymbol{x}_2$ and $\boldsymbol{x}_4$ satisfy **Condition 2** and will be used for updating the classifier, namely, $\widehat{\mathbf{X}}_L \leftarrow \{\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_4\}$. $\boldsymbol{x}_3$ and $\boldsymbol{x}_6$ locate at the overlapping areas occupied jointly by prototypes from different classes. $\boldsymbol{x}_5$ lies outside of the area of influence of all prototypes at the *h*th layer. Therefore, $\boldsymbol{x}_3$, $\boldsymbol{x}_5$ and $\boldsymbol{x}_6$ will be passed to the *(h+1)*th layer to see whether they can satisfy **Condition 2** at the next layer. As one can see from Fig. 3(b), the data space is partitioned by the nine prototypes, $\{\boldsymbol{p}_{h+1,1}^1, \boldsymbol{p}_{h+1,2}^1, \dots, \boldsymbol{p}_{h+1,3}^3\}$ at the *(h+1)*th layer at a higher level of granularity. Nonetheless, $\boldsymbol{x}_3$ still sits at an overlapping area occupied by prototypes of classes 1 and 3, and $\boldsymbol{x}_5$ is, again, not associated with any of the prototypes. Meanwhile, $\boldsymbol{x}_6$ is associated closely with $\boldsymbol{p}_{h+1,2}^2$ thanks to the finer partitioning. As a result, $\boldsymbol{x}_6$ will be used for updating the classifier, namely, $\widehat{\mathbf{X}}_L \leftarrow \widehat{\mathbf{X}}_L \cup \{\boldsymbol{x}_6\}$, and $\boldsymbol{x}_3$ and $\boldsymbol{x}_5$ will be passed to the *(h+2)*th layer (if it exists) or be used for the next self-training cycle.



(a) *h*th layer (b) *(h+1)*th layer

Fig. 3. Illustration of **Condition 2**.

If $\boldsymbol{x}_k$ fails to satisfy **Condition 2** at the bottom layer (namely, $h = H$), it means that $\boldsymbol{x}_k$ locates at either an overlapping area that are occupied by prototypes from two or multiple classes or at a distant area that is out of reach for all previously identified prototypes. In such cases, the following condition is checked to see whether $\boldsymbol{x}_k$ can contribute to expanding the knowledge base of the classifier [7].

$$\textbf{\textit{Condition 3}}: \quad \begin{array}{c} If\ \left(\lambda_1^i(\boldsymbol{x}_k) > \gamma_o \cdot \max_{j=1,2,\ldots,C;i\neq j}\left(\lambda_1^j(\boldsymbol{x}_k)\right)\right) \\ Then\ \left(\hat{\mathbf{X}}_L \leftarrow \hat{\mathbf{X}}_L \cup \{\boldsymbol{x}_k\}\right)\ and\ \left(\hat{\mathbf{Y}}_L \leftarrow \hat{\mathbf{Y}}_L \cup \{\hat{y}_k = i\}\right)\ and\ \left(\mathbf{X}_U \leftarrow \mathbf{X}_U \backslash \{\boldsymbol{x}_k\}\right) \end{array} \tag{15}$$

where γ_o is a user-controlled parameter ($\gamma_o > 1$). If **Condition 3** is unsatisfied as well, $\boldsymbol{x}_k$ is put back to $\mathbf{X}_U$ for possible future use. Note that **Condition 2** is based on the cluster assumption and is mostly dealing with unlabelled samples that share the same and distinctive patterns with the labelled training samples. **Condition 3** exploits the manifold assumption and gives the classifier the ability to make inference. It effectively handles unlabelled samples that either share similar patterns with samples of different classes or lie outside of the area of influence of any prototype.

For better illustration, two three-layer prototype-based hierarchies derived from labelled samples of two different classes (namely, class 1 and class 2) are depicted in Fig. 4 as an example. As shown in Fig. 4, each hierarchy has one apex prototype, three prototypes at the second layer, and six leaf prototypes at the bottom layer. There are six unlabelled samples observed in the data space, namely, $\mathbf{X}_U = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, \boldsymbol{x}_4, \boldsymbol{x}_5, \boldsymbol{x}_6\}$ and there are overlaps between the areas of influence of prototypes of the two classes. It can be observed from Fig. 4 that, $\boldsymbol{x}_1$ lies outside of the areas of influence of the two hierarchies (**Condition 2** is unsatisfied), but $\left\|\boldsymbol{x}_1 - \boldsymbol{p}^1_{1,1}\right\|$ is much smaller than $\left\|\boldsymbol{x}_1 - \boldsymbol{p}^2_{1,1}\right\|$, thus, there is $\hat{y}_1 = 1$ based on **Condition 3** (assuming there is $\lambda_1^1(\boldsymbol{x}_k) > \gamma_o \cdot \lambda_1^2(\boldsymbol{x}_k)$). $\boldsymbol{x}_2$ can be easily pseudo-labelled with $\hat{y}_2 = 1$ based on **Condition 2** because it locates at a place that is occupied by $\boldsymbol{p}^2_{1,1}$ only. Despite that $\boldsymbol{x}_3$ locates at the overlapping area occupied by two apex prototypes, $\boldsymbol{p}^1_{1,1}$ and $\boldsymbol{p}^2_{1,1}$, $\boldsymbol{x}_3$ satisfies **Condition 2** at the second layer ($h = 2$) because its position is covered by the area of influence of $\boldsymbol{p}^1_{2,2}$ only, and thus, there is $\hat{y}_3 = 2$. $\boldsymbol{x}_4$ and $\boldsymbol{x}_5$ also locate at the overlapping area occupied by both $\boldsymbol{p}^1_{1,1}$ and $\boldsymbol{p}^2_{1,1}$. Although $\boldsymbol{x}_4$ is not close enough to any prototypes at lower layers, it manages to satisfy **Condition 3** because it is much closer to $\boldsymbol{p}^2_{1,1}$ and, thus, there is $\hat{y}_4 = 2$. $\boldsymbol{x}_5$ fails to satisfy **Condition 2** due to the same reason as $\boldsymbol{x}_4$, and it also fails to meet **Condition 3** because $\left\|\boldsymbol{x}_5 - \boldsymbol{p}^1_{1,1}\right\| \approx \left\|\boldsymbol{x}_5 - \boldsymbol{p}^2_{1,1}\right\|$. $\boldsymbol{x}_6$ lies outside of the areas of influence of the two prototype-based hierarchies, which is the same as $\boldsymbol{x}_1$, and it fails to satisfy **Condition 3** because $\left\|\boldsymbol{x}_6 - \boldsymbol{p}^1_{1,1}\right\| \approx \left\|\boldsymbol{x}_6 - \boldsymbol{p}^2_{1,1}\right\|$ similar to $\boldsymbol{x}_5$. As a result, $\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, \boldsymbol{x}_4$ are assigned with the respective pseudo-labels and will be

used for updating the two prototype-based hierarchies, meanwhile, $\boldsymbol{x}_5$ and $\boldsymbol{x}_6$ will be kept in $\mathbf{X}_U$ for the next learning cycle.

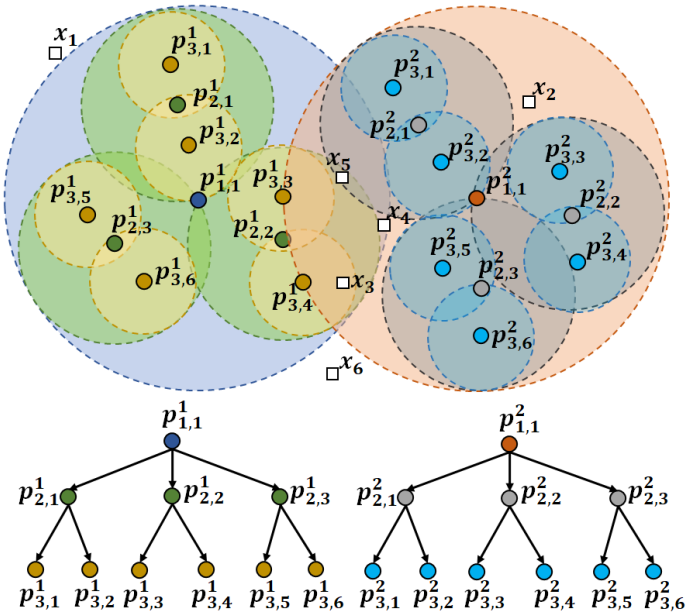


Fig. 4. Illustration of pseudo-labelling process.

The same process is repeated for the next unlabelled sample, $\boldsymbol{x}_k$ ($k \leftarrow k+1$) until all samples in $\mathbf{X}_U$ have been examined. Then, the self-training process goes to the next stage.

Stage 2. Classifier self-updating

In this stage, the classifier self-evolves with the newly obtained pseudo-labelled set $\widehat{\mathbf{X}}_L$ and the corresponding pseudo labels, $\widehat{\mathbf{Y}}_L$. $\widehat{\mathbf{X}}_L$ is, firstly, divided into a number of subsets denoted by $\widehat{\mathbf{X}}_L^1, \widehat{\mathbf{X}}_L^2, \ldots, \widehat{\mathbf{X}}_L^C$ based on $\widehat{\mathbf{Y}}_L$, and then each prototype-based hierarchy is updated with the corresponding pseudo-labelled subset using **Algorithm 1** (starting from ***Stage 1. System dynamically evolving process***). However, it has to be stressed that the classifier is self-updated on a sample-by-sample basis and the self-updating process involves $\widehat{\mathbf{X}}_L$ and $\widehat{\mathbf{Y}}_L$ only. Therefore, no full re-training is needed. After the system has been updated, the self-training process goes back to **Stage 1**, continuing to select out samples from $\mathbf{X}_U$ that can be used for system self-developing until all the remaining samples in $\mathbf{X}_U$ fail to meet both **Conditions 2** and **3**.

One attractive feature of the proposed approach is its higher tolerance to errors happened during the pseudo labelling process. Thanks to the "one pass" learning ability of the base learner, each pseudo-labelled sample will be used for updating the meta-parameters of one prototype at each layer only. Thus, such errors will not be propagated to neighbouring prototypes. Given a small amount of pseudo-labelling errors, the majority of the prototypes at the bottom layer will remain intact. The influence of pseudo-labelling errors on the affected prototypes at the upper layers is also very limited because such errors can only slightly shift the locations of upper-layer prototypes thanks to the much larger numbers of samples associated with them. In addition, even if these errors create new branches in the hierarchies, they can only alter decision boundaries locally without a heavy influence on the overall classification precision. Therefore, one may conclude that the classifier is highly robust to pseudo-labelling errors and can always maintain high classification accuracy.

To further demonstrate this, the structure updating process following the example given by Fig. 4 is presented in Fig. 5, where $\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3$ and $\boldsymbol{x}_4$ are used for updating the two hierarchies. As one can see from Fig. 5, $\boldsymbol{x}_1$ becomes a new apex prototype of the hierarchy corresponding to class 1 and initializes a new branch by itself, namely, $\boldsymbol{p}^1_{1,2}$, $\boldsymbol{p}^1_{2,4}$ and $\boldsymbol{p}^1_{3,7}$. $\boldsymbol{x}_2$ and $\boldsymbol{x}_4$ are used for updating meta-parameters of the apex prototype, $\boldsymbol{p}^2_{1,1}$ because they both are within the area of influence of $\boldsymbol{p}^2_{1,1}$, and the updated apex prototype is re-denoted as $\boldsymbol{p}^2_{1,1(a)}$. Then, they both initialize new branches and become new prototypes at the second and third layers of the hierarchy of class 2,

namely, $\boldsymbol{p}_{2,4}^2, \boldsymbol{p}_{2,5}^2$, $\boldsymbol{p}_{3,7}^2$ and $\boldsymbol{p}_{3,8}^2$. $\boldsymbol{x}_3$ is used for updating the meta-parameters of $\boldsymbol{p}_{1,1}^1$, $\boldsymbol{p}_{2,2}^1$ and $\boldsymbol{p}_{3,4}^1$, and the updated prototypes are re-denoted as $\boldsymbol{p}_{1,1(a)}^1$, $\boldsymbol{p}_{2,2(a)}^1$ and $\boldsymbol{p}_{3,4(a)}^1$, respectively. Other prototypes remain in the same positions. From this example one can also see that a pseudo-labelling error can only influence a branch of the hierarchy, while the vast majority of prototypes at the bottom layers are not influenced by the error.

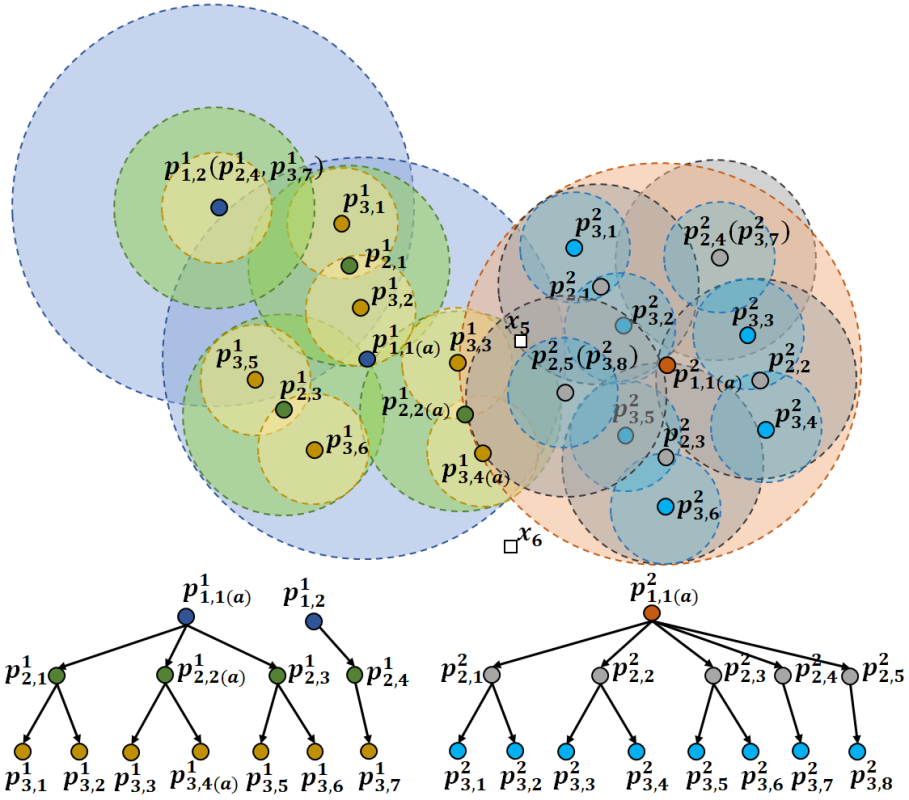


Fig. 5. Illustration of structure updating process.

Once the whole learning process is finished, for these remaining unlabelled samples $\boldsymbol{x}_k \in \mathbf{X}_U$ $(k = 1,2, \dots, U)$ that still fail to satisfy **Conditions 2** and **3** at the end of the self-training process, the classifier will predict the class label for each of them by equations (12) and (13), but these samples will not be used for updating the classifier. After this, the classifier is ready for out-of-sample prediction by following the same decision-making process as described in subsection 3.3.

The overall learning process of the STHP classifier is summarized in the form of pseudo code as follows.

Algorithm 2: STHP classifier identification

Input: $\mathbf{X}_L$, $\mathbf{Y}_L$ and $\mathbf{X}_U$
Algorithm begins

\\ *Stage 0. Classifier priming* \\
a. **For** $i = 1$ to C **do**
i. Select out $\mathbf{X}_L^i$ and $\mathbf{Y}_L^i$ from $\mathbf{X}_L$ and $\mathbf{Y}_L$;
ii. Prime the *i*th hierarchy using $\mathbf{X}_L^i$ by ***Algorithm 1***;
b. **End for**
c. **While** $(\mathbf{X}_U \neq \emptyset)$
\\ *Stage 1. Pseudo-labelled set acquisition* \\
i. $\widehat{\mathbf{X}}_L \leftarrow \emptyset$ and $\widehat{\mathbf{Y}}_L \leftarrow \emptyset$;
ii. **For** $k = 1$ to U **do**
1. **For** $h = 1$ to H **do**
* **If** (**Condition 2** is satisfied) **then**
- $\widehat{\mathbf{X}}_L \leftarrow \widehat{\mathbf{X}}_L \cup \{\boldsymbol{x}_k\}$; $\widehat{\mathbf{Y}}_L \leftarrow \widehat{\mathbf{Y}}_L \cup \{\hat{y}_k = i\}$; $\mathbf{X}_U \leftarrow \mathbf{X}_U \backslash \{\boldsymbol{x}_k\}$;
- Break **for** loop;

* **End if**

2. **End for**

3. **If** (**Condition 2** is unsatisfied) **and** (**Condition 3** is satisfied) **then**

* $\widehat{\mathbf{X}}_L \leftarrow \widehat{\mathbf{X}}_L \cup \{\boldsymbol{x}_k\}$; $\widehat{\mathbf{Y}}_L \leftarrow \widehat{\mathbf{Y}}_L \cup \{\hat{y}_k = i\}$; $\mathbf{X}_U \leftarrow \mathbf{X}_U \backslash \{\boldsymbol{x}_k\}$;

* Break **for** loop;

4. **End if**

iii. **End for**

\\ *Stage 2. Classifier self-updating* \\

iv. **If** ($\widehat{\mathbf{X}}_L = \emptyset$) **then**

1. Break **while** loop;

v. **Else**

1. **For** $i = 1$ to C **do**

* Select out $\widehat{\mathbf{X}}_L^i$ and $\widehat{\mathbf{Y}}_L^i$ from $\widehat{\mathbf{X}}_L$ and $\widehat{\mathbf{Y}}_L$;

* Update the *i*th hierarchy using $\widehat{\mathbf{X}}_L^i$ by ***Algorithm 1***;

2. **End for**

vi. **End if**

vii. $U \leftarrow |\mathbf{X}_U|$;

d. **End while**

Algorithm ends

Output: the hierarchical prototype-based classifier

5. Computational Complexity Analysis

In this section, the computational complexity of the proposed STHP classifier is analysed.

5.1. Learning process

As the system structure and meta-parameters of the classifier are both dynamically evolving, it is impossible to derive an exact expression of computational complexity for the overall learning process. Thus, only the lower and upper bounds of the computational complexity of each algorithmic stage will be given.

The classifier is primed with the labelled set in **Stage 0**. For a particular labelled sample, $\boldsymbol{x}_k \in \mathbf{X}_L$ with $y_k = i$, the lower and upper bounds of the computational complexity of the learning cycle have been given in [7], which are $O(NM_1^i)$ and $O\left(N\left(M_1^i + \sum_{h=1}^{H-1} P_{h,n_h^*}^i + H\right)\right)$, respectively. Here $P_{h,n_h^*}^i$ is the cardinality of $\boldsymbol{\mathcal{L}}_{h,n_h^*}^i$. Therefore, the overall computational complexity of the supervised classifier priming process as described in **Stage 0** is between $O(CN)$ (when there is only one labelled sample per class available) and $O\left(LN\left(\max_{i=1,2,\dots,C}\left(M_1^i + \sum_{h=1}^{H-1} P_{h,n_h^*}^i\right) + H\right)\right)$.

In **Stage 1**, for each unlabelled sample, $\boldsymbol{x}_k \in \mathbf{X}_U$, it is firstly compared with apex prototypes at the top layer of each hierarchy, and then the same process is repeated in a top-down manner until **Condition 2** is met. If $\boldsymbol{x}_k$ fails to meet **Condition 2**, **Condition 3** is checked before putting it back to the pool. As a result, the lower bound of the computational complexity is reached when $\boldsymbol{x}_k$ satisfies **Condition 2** during the comparison with apex prototypes. In this case, the computational complexity is $O\left(N\sum_{i=1}^{C} M_1^i\right)$. Meanwhile, the maximum computational complexity is reached if $\boldsymbol{x}_k$ triggers both **Conditions 2** and **3** and, in this case, the computational complexity is $O\left(N\sum_{i=1}^{C}\sum_{h=1}^{H} M_h^i\right)$ (the computational complexity of **Condition 3** is negligible compared with **Condition 2**). Therefore, the lower and upper bounds of the computational complexity of **Stage 1** are $O\left(UN\sum_{i=1}^{C} M_1^i\right)$ (when all unlabelled samples meet **Condition 2** with $h = 1$) and $O\left(UN\sum_{i=1}^{C}\sum_{h=1}^{H} M_h^i\right)$ (when every unlabelled sample triggers both **Conditions 2** and **3**).

Stage 2 mostly concerns with classifier updating. Thus, the same conclusion on **Stage 0** can be applied. The computational complexity of this stage is between $O(0)$ (if no pseudo-labelled sample is selected for classifier updating) and $O\left(\hat{L}N\left(\max_{i=1,2,\dots,C}\left(M_1^i + \sum_{h=1}^{H-1} P_{h,n_h^*}^i\right) + H\right)\right)$, where $\hat{L}$ is the cardinality of $\widehat{\mathbf{X}}_L$.

5.2. Decision-making process

The computational complexity of the decision-making process is more straightforward. For a given data sample, $\boldsymbol{x}$, each prototype-based hierarchy will give a score of confidence by equations (12) and (13), and the computational complexity of the overall decision-making process is $O\left(N\sum_{i=1}^{C} M_h^i\right)$.

6. Experimental Investigation

In this section, numerical examples are presented to justify the effectiveness and validity of the proposed concept and method. The algorithms were developed using MATLAB2018a, and the performance was evaluated on a desktop with dual core i7 processor 3.60GHz × 2 and 32.0GB RAM.

6.1. Dataset description and experimental setting

Considering the significant interest in machine learning and computer vision communities to leverage the astronomical amount of unlabelled images existing on the Internet for building recognition models [10], the numerical experiments presented in this paper are focused on the image recognition perspective. Note that the proposed approach is generic and applicable to numerical data as well.

The following well-known challenging benchmark image sets are involved for demonstration. Example images of these datasets are given in Fig. 6 for illustration and key information is summarized in Table 3. Interested readers may find very detailed descriptions on these six benchmark problems from [7],[9],[43].

1) Singapore dataset[1];

2) WHU-RS dataset[2];

3) UCMerced dataset[3];

4) RSSCN7 dataset[4];

5) Caltech101 dataset[5]; and,

6) Caltech256 dataset[6].

In this paper, the pretrained VGG-VD-16 deep convolutional neural network (DCNN) model [29] is employed for feature extraction due to its simple structure and high performance [43]. Following the common practice, the 4096×1 dimensional activations from the first fully connected layer are extracted as the feature vectors of the images. Note that there is no further tuning involved. Moreover, one may consider using other feature descriptors for feature extraction, e.g., GoogLeNet [32], AlexNet [17], Gist [26], but this is beyond the scope of this paper.

[1] Available at: http://icn.bjtu.edu.cn/Visint/resources/Scenesig.aspx

[2] Available at: http://captain.whu.edu.cn/repository.html

[3] Available at: http://weegee.vision.ucmerced.edu/datasets/landuse.html

[4] Available at: https://sites.google.com/view/zhouwx/dataset

[5] Available at: http://www.vision.caltech.edu/Image_Datasets/Caltech101/

[6] Available at: http://www.vision.caltech.edu/Image_Datasets/Caltech256/

(a) Singapore dataset

(b) WHU-RS dataset

(c) UCMerced dataset

(d) RSSCN7 dataset

(e) Caltech101 dataset

(f) Caltech256 dataset

Fig. 6. Example images of the benchmark image sets for numerical experiments.

Table 3. Details of the benchmark datasets for numerical examples

Dataset	Resolution	# Classes	# Images	# Attributes
Singapore	256×256×3	9	1086	4096×1
WHU-RS	600×600×3	19	950	
UCMerced	256×256×3	21	2100	
RSSCN7	400×400×3	7	2800	
Caltech101	Roughly 300×200×3	101	8677	
Caltech256		256	29780	

For performance comparison, the following state-of-the-art semi-supervised classification approaches are involved in numerical examples.

1) AnchorGraphReg method with a regression matrix with kernel-defined weights (AGR) [22];

2) AnchorGraphReg method with a regression matrix optimized by local anchor embedding (AGRL) [22];

3) Local and global consistency method (LGC) [46];

4) Greedy gradient Max-Cut method (GGMC) [38];

5) Laplacian SVM classifier (LapSVM) [1]; and

6) SSDRB classifier [7].

Note that AGR, AGRL, LGC, GGMC and LapSVM are widely used semi-supervised approaches. The user-controlled parameter of AGR and AGRL, s (number of the closest anchors) is set to be 3; the iteration number of local anchor embedding for AGRL is set to be 10 as suggested in [22]. The user-controlled parameter, α of LGC is set to be 0.99 as suggested in [46]. The user-controlled parameter, μ of GGMC is set to be 0.01 as suggested in [38]. Both LGC and GGMC use the KNN graph with $k = 5$. LapSVM uses the "one-versus-all" strategy for all the benchmark problems. Since the performance of LapSVM is subject to the externally controlled parameters, in this paper, the following three parameter settings are considered, and the classifiers with the respective settings are re-denoted as LapSVM1, LapSVM2 and LapSVM3, respectively. For LapSVM1, the radial basis function kernel with $\sigma = 10$ is used; the two user-controlled parameters γ_I and γ_A are set to be 1 and 10^{-6}, respectively; the number of neighbour, k, for computing the graph Laplacian is set to be 15 as suggested in [1]. For LapSVM2, the following setting is used: $\sigma = 10$, $\gamma_I = 0.5$, $\gamma_A = 10^{-6}$ and $k = 15$. For LapSVM3, the following setting is considered: $\sigma = 1$, $\gamma_I = 1$, $\gamma_A = 10^{-5}$ and $k = 10$. SSDRB is of the same type as the proposed STHP classifier; the user-controlled parameter, Ω_1 of SSDRB is set to be 1.2 as suggested in [7].

In addition, the following supervised classification approaches are involved for a better comparison.

7) HP classifier [9];

8) Deep rule-based (DRB) classifier [8];

9) SVM classifier [5]; and

10) KNN classifier [6];

It is well known that both SVM and KNN classifiers are the two main generic classifiers used by pre-trained DCNN-based approaches and have demonstrated very strong performance on various benchmark problems [27],[43]. In the numerical examples presented in this paper, SVM uses the linear kernel function, and the value of k for KNN is set to be 1. Both HP and DRB serve as the baseline because STHP and SSDRB reduce to HP and DRB, respectively, if no self-training is performed. The experimental settings of HP and DRB follow the modes of [9] and [8], respectively; the layer number of HP is set as 3.

All the reported results are obtained after 25 Monte Carlo experiments by randomly dividing the involved benchmark datasets into labelled and unlabelled sets under certain ratios. In this paper, by default, the bottom layer (namely, the *H*th layer) of HP and STHP is used for decision-making because this layer contains a larger number of leaf prototypes with fine details of the problem, and thus, is able to perform classification with higher accuracy [9].

6.2. Experimental demonstration

First of all, the influence of the two user-controlled parameters, H and γ_o on classification accuracy and system complexity of the STHP classifier is investigated. This numerical example is based on Singapore dataset because of its smaller scale and simpler structure. During this experiment, six images ($L^i = 6$) from each class are randomly selected out to form the labelled set and the rest are used as the unlabelled set for training. The value of H varies from 1 to 5, and the value of γ_o varies from 1.05 to 1.35. The classification accuracy (Acc) on the unlabelled images and the average number of prototypes per layer per class (M_h; the subscript h denotes the hth layer; $h = 1,2,\dots,H$) are reported in Table 4. The performance of the HP classifier is also reported in the same table as the baseline.

Table 4. Classification accuracy and system complexity of STHP with different values of γ_o and H

Algorithm		STHP							HP
H	γ_o	1.05	1.10	1.15	1.20	1.25	1.30	1.35	
1	Acc	0.903	0.896	0.887	0.880	0.872	0.865	0.861	0.903
	M_1	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
2	Acc	0.927	0.938	0.944	0.944	0.944	0.942	0.939	0.916
	M_1	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	M_2	9.04	7.72	6.65	5.76	5.15	4.64	4.34	3.38
3	Acc	0.919	0.925	0.931	0.933	0.938	0.937	0.936	0.912
	M_1	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	M_2	9.04	7.73	6.67	5.78	5.19	4.68	4.37	3.38
	M_3	92.78	89.83	87.40	84.95	82.90	80.87	79.06	5.89
4	Acc	0.919	0.925	0.931	0.934	0.938	0.936	0.935	0.913
	M_1	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	M_2	9.04	7.73	6.67	5.78	5.19	4.68	4.37	3.38
	M_3	92.78	89.83	87.40	84.95	82.90	80.87	79.06	5.89
	M_4	118.15	115.05	112.51	109.91	107.83	105.72	103.74	6.00
5	Acc	0.919	0.925	0.931	0.934	0.938	0.936	0.935	0.913
	M_1	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	M_2	9.04	7.73	6.67	5.78	5.19	4.68	4.37	3.38
	M_3	92.78	89.83	87.40	84.95	82.90	80.87	79.06	5.89
	M_4	118.15	115.05	112.51	109.91	107.83	105.72	103.74	6.00

	M_5	118.45	115.32	112.75	110.13	108.06	105.95	103.89	6.00

Table 4 shows that the STHP classifier performs the best on this dataset with a two-layer structure ($H = 2$). Due to the problem of overfitting, when adding extra layers to the system, the classification accuracy of the proposed approach decreases slightly at first and then becomes stable. Thus, it can be concluded that the layer number has a marginal influence on the learning outcomes. The other user-controlled parameter, γ_o has a larger influence on both classification accuracy and system complexity (in terms of prototype numbers). In general, the classifier identifies less prototypes after the self-training process with a larger value of γ_o, and vice versa. It is also noticeable that the classification accuracy is not linearly related to the value setting of γ_o. If a smaller value of γ_o is used, the system tends to produce more mistakes when making inferences on unlabelled samples using **Condition 3**, and these pseudo-labelling errors deteriorate the overall performance. Meanwhile, a larger value of γ_o may impair the inferencing ability of STHP and stop the system from gaining new knowledge. In such cases, the STHP classifier may miss the valuable information hidden in the unlabelled samples. Based on Table 4, the best value range of γ_o is between 1.10 and 1.25.

Following the example presented in Table 4, the influence of θ_o on classification accuracy and system complexity of the STHP classifier is investigated. In this example, a two-layer STHP classifier with $\gamma_o = 1.2$ is used for demonstration. During the experiment, six images per class are randomly selected out as the labelled set, and the remaining images are used as the unlabelled set. The value of θ_o varies from $\frac{\pi}{2}$ to $\frac{\pi}{16}$, and the classification accuracy on the unlabelled images and the average number of prototypes per layer per class are reported in Table 5. The same experiment is repeated by using 12 images per class to form the labelled set, and the results are reported in Table 5 as well. The performance of the HP classifier is also given as the baseline.

Table 5. Classification accuracy and system complexity of STHP with different values of θ_o

θ_o		$\frac{\pi}{2}$		$\frac{\pi}{4}$		$\frac{\pi}{8}$		$\frac{\pi}{16}$	
L^i	Algorithm	STHP	HP	STHP	HP	STHP	HP	STHP	HP
6	Acc	0.944	0.916	0.936	0.912	0.938	0.913	0.937	0.913
	M_1	1.00	1.00	5.84	3.38	78.74	5.89	103.47	6.00
	M_2	5.76	3.38	86.77	5.89	104.36	6.00	103.71	6.00
12	Acc	0.965	0.949	0.961	0.947	0.963	0.947	0.963	0.947
	M_1	1.00	1.00	6.43	4.67	79.20	11.56	104.78	12.00
	M_2	6.36	4.67	86.97	11.56	105.21	12.00	105.03	12.00

Table 5 shows that θ_o can significantly influence the system complexity of the STHP classifier. A smaller value of θ_o will reduce the area of influence of each prototype within the system, resulting in more prototypes being identified during both the supervised and semi-supervised learning processes. In addition, it can be observed from Table 5 that the classification accuracy deceases with a smaller value of θ_o. The main reason for this is as follows. As the area of influence of each prototype is reduced, **Condition 2** becomes less important during the pseudo-labelling process. At the same time, **Condition 3** starts to play as the dominant role in determining the pseudo-labels of unlabelled samples. Since **Condition 3** is less strict than **Condition 2**, more pseudo-labelling errors are inevitably introduced to the classifier during the semi-supervised learning process by **Condition 3**, which, in turn, deteriorates the system performance. It is also worth to be noticed that **Condition 3** performs pseudo-labelling based on the mutual distances between unlabelled samples and labelled prototypes, thus, this condition becomes more effective when more labelled samples are given.

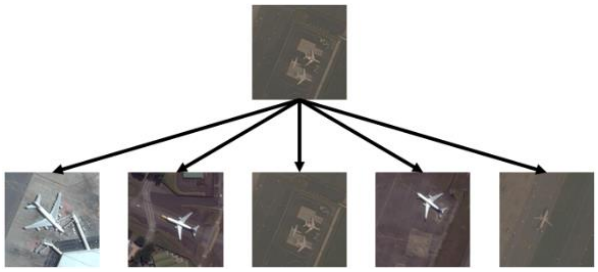
To further verify this, the following numerical example is performed. In this example, three two-layer STHP classifiers with different pseudo-labelling mechanisms are considered ($\gamma_o = 1.2$ for all of them). The first STHP classifier assigns pseudo-labels to unlabelled samples by using **Condition 2** only. The second one uses **Condition 3** for pseudo-labelling only. The third one uses both **Conditions 2** and **3**, which is the same as the previous numerical examples. The three STHP classifiers are re-denoted as STHP1, STHP2 and STHP3, respectively. The

same experiments conducted in Table 5 are repeated and the obtained results are tabulated in Table 6. As one can see, STHP2 identifies more prototypes than STHP1 and STHP3, and its classification accuracy is higher than STHP1 when more labelled samples are given. Nonetheless, STHP3 outperforms the other two in both experiments. This example further justifies the effectiveness and validity of the proposed self-training mechanism.

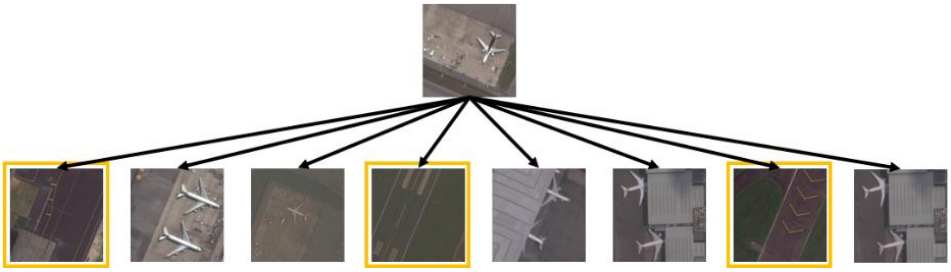
Table 6. Comparison of classification accuracy and system complexity between three different STHP classifiers

L^i	Algorithm	STHP1	STHP2	STHP3	HP
6	Acc	0.934	0.933	0.944	0.916
	M_1	1.00	1.00	1.00	1.00
	M_2	3.57	8.66	5.76	3.38
12	Acc	0.957	0.962	0.965	0.949
	M_1	1.00	1.00	1.00	1.00
	M_2	4.83	7.98	6.36	4.67

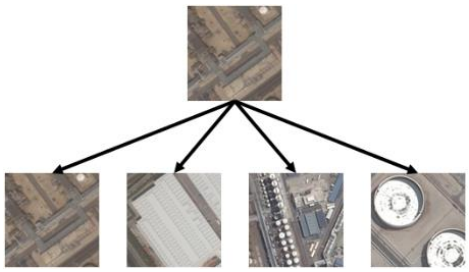
For better illustration, three prototype-based hierarchies corresponding to three classes "airplane", "industry" and "runway" of Singapore dataset obtained during a particular experiment with $H = 2$ and $\gamma_o = 1.2$ are given in Fig. 7. Since the STHP classifier performs semi-supervised learning and classification based on the 4096×1 dimensional feature vectors of images and the identified prototypes in their vector forms are not intuitive for visualization, images with the feature vectors that are the most similar to the identified prototypes are used for visualization in Fig. 7 for clarity.



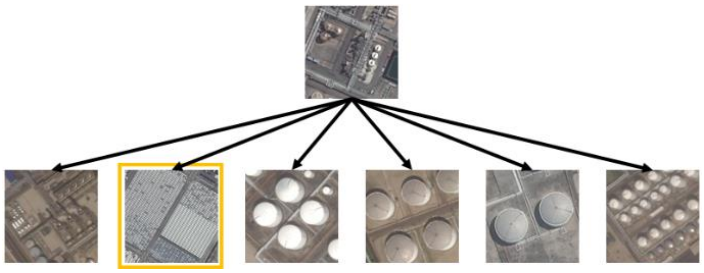
(a) the hierarchy of the class "airplane" after being primed with labelled samples



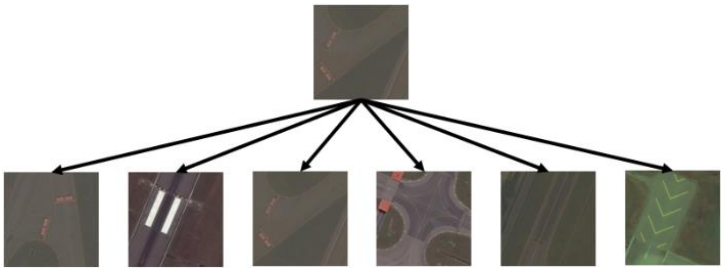
(b) the hierarchy of the class "airplane" after self-training with unlabelled samples



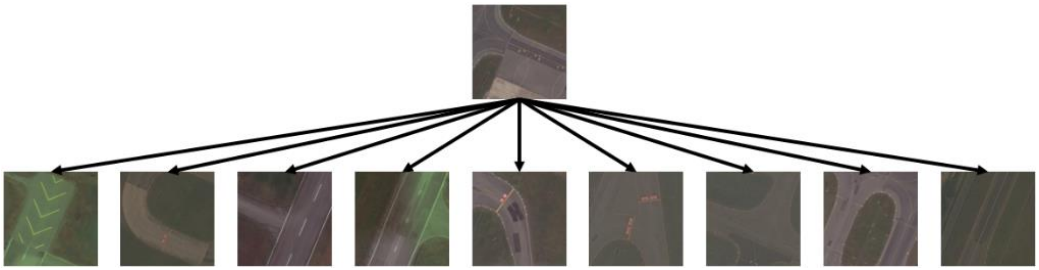
(c) the hierarchy of the class "industry" after being primed with labelled samples



(d) the hierarchy of the class “industry” after self-training with unlabelled samples



(e) the hierarchy of the class “runway” after being primed with labelled samples



(f) the hierarchy of the class “runway” after self-training with unlabelled samples

Fig. 7. Illustration of prototype-based hierarchies.

One can see from Fig. 7 that each prototype-based hierarchy is able to gain a few new prototypes through the self-training process by utilizing unlabelled images. In this way, the proposed STHP classifier effectively self-expands its knowledge base and self-improves its system structure without human supervision. However, one may notice that the self-training process also introduces some errors into the system (namely, the prototypes in the yellow boxes). This is due to the very high visual similarity between these unlabelled images and the prototypes identified from labelled images of a different class. For example, some images of the class “runway” are highly similar to the images of class “airplane”. Nonetheless, this issue could be addressed by using alternative feature descriptors with stronger descriptive abilities for feature extraction instead.

In the numerical examples presented in the rest of this section, the two user-controlled parameters of STHP classifier are set to be $H = 3$ and $\gamma_o = 1.2$ following the mode of [9] and [7]. However, it has to be stressed that this is only a general setting. The best parameter setting may differ from case to case and needs prior knowledge to be determined.

6.3. Performance comparison and discussions

In this subsection, the performance of the STHP classifier is compared with the state-of-the-art semi-supervised and supervised classification algorithms on the six benchmark datasets listed in subsection 6.1.

Firstly, for Singapore, WHU-RS, UCMerced and RSSCN7 datasets, $L^i = 2, 4, 6, \ldots, 12$ images from each class ($i = 1,2, \ldots, C$) are randomly selected out to form the labelled set, the remaining images are used as the unlabelled set. The classification accuracy rates of the STHP classifier and the 12 comparative algorithms on the unlabelled

sets of the four datasets are reported in Table 7 in the form of $mean \pm standard\ deviation$. The best results are highlighted.

Table 7. Performance comparison on benchmark remote sensing datasets

Dataset	Algorithm	L^i					
		2	4	6	8	10	12
Singapore	STHP	0.888±0.040	0.914±0.025	0.933±0.023	0.944±0.018	0.952±0.016	0.959±0.013
	AGR	0.823±0.038	0.849±0.032	0.858±0.024	0.868±0.035	0.876±0.033	0.887±0.021
	AGRL	0.833±0.051	0.859±0.043	0.882±0.028	0.900±0.026	0.906±0.026	0.915±0.020
	LGC	0.850±0.049	0.902±0.032	0.913±0.025	0.937±0.022	0.941±0.016	0.944±0.015
	GMMC	0.807±0.062	0.818±0.055	0.836±0.062	0.853±0.054	0.840±0.045	0.838±0.041
	LapSVM1	0.466±0.092	0.673±0.072	0.745±0.046	0.820±0.045	0.848±0.036	0.887±0.019
	LapSVM2	0.446±0.091	0.665±0.070	0.771±0.043	0.834±0.042	0.857±0.033	0.892±0.019
	LapSVM3	0.401±0.075	0.576±0.075	0.684±0.055	0.790±0.051	0.827±0.041	0.872±0.027
	SSDRB	**0.893±0.042**	**0.922±0.028**	**0.939±0.018**	**0.953±0.014**	**0.954±0.016**	**0.964±0.014**
	HP	0.839±0.036	0.884±0.026	0.912±0.014	0.930±0.018	0.934±0.014	0.947±0.012
	DRB	0.839±0.036	0.882±0.025	0.912±0.015	0.929±0.018	0.932±0.014	0.947±0.012
	SVM	0.680±0.066	0.817±0.042	0.854±0.036	0.898±0.023	0.898±0.025	0.922±0.019
	KNN	0.715±0.053	0.805±0.036	0.844±0.030	0.883±0.025	0.886±0.023	0.908±0.017
WHU-RS	STHP	**0.767±0.034**	**0.835±0.021**	**0.854±0.025**	**0.876±0.011**	**0.886±0.011**	**0.890±0.011**
	AGR	0.617±0.077	0.671±0.046	0.705±0.036	0.717±0.028	0.742±0.032	0.741±0.009
	AGRL	0.693±0.057	0.745±0.048	0.759±0.034	0.784±0.028	0.794±0.039	0.805±0.017
	LGC	0.738±0.035	0.804±0.033	0.831±0.020	0.849±0.018	0.863±0.011	0.864±0.015
	GMMC	0.785±0.023	0.805±0.030	0.820±0.019	0.825±0.022	0.826±0.019	0.842±0.032
	LapSVM1	0.614±0.041	0.772±0.030	0.832±0.018	0.858±0.011	0.871±0.012	0.880±0.027
	LapSVM2	0.606±0.041	0.764±0.035	0.837±0.019	0.860±0.013	0.872±0.011	0.879±0.010
	LapSVM3	0.601±0.037	0.754±0.034	0.830±0.016	0.862±0.011	0.882±0.010	0.888±0.010
	SSDRB	0.728±0.036	0.783±0.024	0.814±0.022	0.838±0.015	0.853±0.015	0.857±0.012
	HP	0.699±0.030	0.775±0.022	0.804±0.018	0.829±0.016	0.846±0.014	0.852±0.011
	DRB	0.699±0.030	0.775±0.023	0.804±0.018	0.828±0.017	0.844±0.015	0.848±0.011
	SVM	0.576±0.041	0.719±0.033	0.776±0.021	0.807±0.023	0.844±0.018	0.860±0.017
	KNN	0.602±0.032	0.697±0.033	0.739±0.019	0.764±0.015	0.795±0.022	0.802±0.013
UCMerced	STHP	**0.681±0.029**	**0.740±0.023**	**0.766±0.016**	**0.792±0.015**	**0.804±0.011**	**0.819±0.012**
	AGR	0.601±0.037	0.657±0.026	0.677±0.028	0.704±0.016	0.717±0.014	0.734±0.016
	AGRL	0.640±0.030	0.691±0.028	0.718±0.027	0.737±0.019	0.751±0.020	0.765±0.018
	LGC	0.631±0.033	0.694±0.022	0.718±0.020	0.743±0.021	0.762±0.020	0.773±0.016
	GMMC	0.606±0.042	0.635±0.036	0.649±0.030	0.670±0.026	0.677±0.027	0.683±0.024
	LapSVM1	0.442±0.038	0.590±0.027	0.668±0.019	0.722±0.020	0.760±0.014	0.787±0.012
	LapSVM2	0.428±0.043	0.576±0.033	0.674±0.020	0.724±0.022	0.764±0.014	0.788±0.012
	LapSVM3	0.419±0.045	0.566±0.031	0.669±0.023	0.725±0.023	0.767±0.016	0.790±0.013
	SSDRB	0.651±0.030	0.722±0.022	0.750±0.023	0.779±0.06	0.792±0.011	0.807±0.014
	HP	0.618±0.023	0.692±0.019	0.723±0.016	0.758±0.013	0.774±0.013	0.792±0.012
	DRB	0.618±0.023	0.692±0.019	0.722±0.016	0.757±0.012	0.773±0.013	0.791±0.012
	SVM	0.504±0.029	0.639±0.036	0.701±0.019	0.744±0.020	0.781±0.019	0.804±0.014
	KNN	0.524±0.027	0.610±0.024	0.645±0.024	0.681±0.016	0.712±0.015	0.730±0.014
RSSCN7	STHP	0.534±0.050	0.602±0.040	0.646±0.023	**0.676±0.033**	**0.697±0.027**	**0.700±0.015**
	AGR	0.545±0.052	0.590±0.043	0.645±0.026	0.651±0.032	0.655±0.032	0.675±0.022
	AGRL	**0.556±0.047**	**0.609±0.045**	**0.658±0.031**	0.668±0.030	0.666±0.032	0.678±0.029
	LGC	0.495±0.060	0.571±0.061	0.617±0.048	0.647±0.035	0.664±0.028	0.680±0.030
	GMMC	0.536±0.067	0.537±0.074	0.564±0.052	0.563±0.054	0.544±0.060	0.581±0.052
	LapSVM1	0.175±0.034	0.225±0.057	0.261±0.059	0.276±0.056	0.314±0.049	0.353±0.049
	LapSVM2	0.177±0.039	0.207±0.047	0.254±0.058	0.267±0.052	0.315±0.051	0.357±0.045
	LapSVM3	0.176±0.026	0.213±0.044	0.262±0.059	0.282±0.054	0.318±0.052	0.367±0.046
	SSDRB	0.501±0.050	0.573±0.055	0.624±0.029	0.656±0.035	0.671±0.031	0.687±0.026
	HP	0.488±0.042	0.549±0.042	0.598±0.023	0.628±0.029	0.637±0.023	0.649±0.024
	DRB	0.488±0.042	0.549±0.043	0.598±0.023	0.628±0.029	0.637±0.023	0.649±0.024
	SVM	0.448±0.047	0.554±0.035	0.617±0.030	0.655±0.033	0.684±0.018	0.697±0.021
	KNN	0.434±0.046	0.504±0.042	0.549±0.026	0.582±0.034	0.598±0.023	0.606±0.036

As one can see from Table 7, the proposed STHP classifier demonstrates very high classification accuracy on all four remote sensing datasets surpassing the best-performing alternatives in most of the cases.

Apart from the classification accuracy, it is also important to investigate whether the performance improvement of the proposed approach over the comparative approaches is of statistical significance. Therefore, statistical pairwise Wilcoxon tests between the STHP classifier and the selected comparative algorithms with higher classification precision (namely, AGRL, LGC, LapSVM3, SSDRB, HP and DRB) are conducted in the following example. The Fisher's method is employed to combine the *p*-values returned from the hypothesis tests on 25 Monte Carlo experiments:

$$X^2 = -2\sum_{j=1}^{25}\ln(p_j) \tag{16}$$

where p_j is the *p*-value returned from the *j*th hypothesis test; $j = 1,2,\ldots,25$. The X^2 values returned from the pairwise Wilcoxon tests between STHP and the alternatives are tabulated in Table 8, where "Inf" denotes infinite value. In addition, the pairwise Wilcoxon tests between the ground truth and all the algorithms are also performed and the results are reported in Table 8 as well. It is worth to be noticed that the value of X^2 tends to be large when the *p*-values are small, which suggests that the null hypotheses are not true for all the tests. If the obtained 25 *p*-values are all greater than 0.05, X^2 is smaller than $-2 \times 25 \times \ln(0.05) \approx 149.7866$. Based on the returned X^2 values from the 25 statistical tests tabulated in Table 8, one can conclude that the performance of the STHP classifier is significantly better than alternatives.

Table 8. X^2 values returned from pairwise Wilcoxon tests

	Dataset	L^i	STHP	AGRL	LGC	LapSVM3	SSDRB	HP	DRB
STHP vs	Singapore	2		734.43	1316.84	9324.46	703.45	318.56	311.79
		4		396.89	766.69	4966.00	323.08	256.92	255.16
		6		366.98	415.21	3337.22	228.39	291.77	275.75
		8		581.71	353.60	1524. 80	199.12	280.25	283.82
		10		381.02	285.98	1136.42	138.88	263.28	270.13
		12		501.42	345.54	697.75	182.23	233.88	230.57
Ground truth vs		2	644.46	768.69	1200.37	9023.28	698.26	639.33	661.63
		4	592.74	842.05	881.97	5685.27	602.80	754.76	751.04
		6	400.81	559.72	555.28	4167.45	572.84	688.55	677.48
		8	263.93	805.88	278.07	1795.50	387.90	467.52	470.92
		10	298.43	648.88	187.59	1510.24	402.53	514.86	516.78
		12	172.35	654.60	255.63	823.45	340.68	420.12	416.60
STHP vs	WHU-RS	2		306.75	723.85	1233.20	510.64	142.11	140.68
		4		309.38	500.61	848.58	332.68	181.28	178.80
		6		346.06	288.33	356.84	213.22	77.85	78.10
		8		184.98	229.31	114.63	168.92	95.39	98.34
		10		265.01	182.28	110.31	151.45	102.67	120.20
		12		230.03	138.36	80.84	89.08	50.14	65.22
Ground truth vs		2	376.24	261.76	590.45	1313.79	538.35	275.44	273.85
		4	270.79	236.52	536.57	695.11	372.89	211.81	210.08
		6	133.72	214.39	305.74	219.11	177.42	125.51	122.22
		8	98.25	176.96	222.20	96.34	106.87	81.04	78.18
		10	118.49	209.58	166.82	49.58	104.25	87.67	84.50
		12	91.73	183.47	173.16	59.28	115.92	85.92	100.86
STHP vs	UCMerced	2		796.77	1638.49	7541.93	1145.99	278.60	278.79
		4		348.68	925.52	4411.05	525.76	249.36	249.74
		6		313.13	330.09	1492.64	355.73	151.00	157.17
		8		188.19	330.99	823.86	308.54	238.09	248.44
		10		182.95	484.74	503.04	152.48	194.53	188.79
		12		179.67	178.65	287.62	202.56	126.31	121.17
Ground truth vs		2	934.58	529.36	1423.33	6574.41	932.48	533.98	532.78
		4	497.23	208.15	1023.79	4686.62	450.06	220.49	221.06
		6	452.88	146.21	510.84	2110.71	366.35	269.74	276.09

		8	328.46	163.30	444.80	1117.58	212.01	168.48	166.52
		10	339.14	145.40	650.56	776.35	234.97	158.99	163.26
		12	179.38	214.84	266.08	486.63	139.75	158.13	151.23
STHP vs	RSSCN7	2		2399.65	5512.90	Inf	4115.21	625.94	625.94
		4		1468.41	6662.87	Inf	2516.29	703.30	734.65
		6		1541.43	4433.36	Inf	1399.10	703.69	710.88
		8		954.25	3537.62	Inf	1215.88	284.55	293.22
		10		990.91	2580.95	Inf	1402.36	458.28	472.09
		12		864.54	3784.01	14330.77	1332.11	713.18	681.77
Ground truth vs		2	3071.55	1073.00	5827.33	Inf	6323.16	2433.81	2433.81
		4	1667.30	423.04	7499.36	Inf	2800.61	1076.93	1091.51
		6	1745.63	249.95	5340.46	Inf	2352.75	850.18	840.51
		8	980.20	217.34	4002.20	Inf	2129.19	770.23	766.71
		10	791.79	161.67	1642.37	14949.64	1262.06	662.96	665.82
		12	855.50	187.40	3774.69	13455.29	1327.41	480.35	464.88

In the last numerical example, for Caltech101 and Caltech256 datasets, $L^i = 1,2,3,\ldots,6$ images from each class ($i = 1,2,\ldots,C$) are randomly selected out to form the labelled set, the remaining images are used as the unlabelled set. The classification accuracy rates of the STHP classifier and the 12 comparative algorithms on the unlabelled sets of the two datasets are reported in Table 9 in the form of $mean \pm standard\ deviation$. The best results are highlighted.

Table 9. Performance comparison on Caltech datasets

Dataset	Algorithm	L^i					
		1	2	3	4	5	6
Caltech101	STHP	0.707±0.032	**0.771±0.020**	**0.792±0.018**	**0.812±0.014**	**0.819±0.013**	**0.827±0.011**
	AGR	0.549±0.035	0.545±0.073	0.565±0.080	0.611±0.048	0.607±0.068	0.634±0.036
	AGRL	0.533±0.075	0.545±0.088	0.5634±0.077	0.632±0.039	0.646±0.053	0.657±0.031
	LGC	0.643±0.035	0.715±0.026	0.745±0.028	0.764±0.022	0.773±0.021	0.783±0.015
	GMMC	0.610±0.037	0.646±0.020	0.655±0.024	0.660±0.023	0.671±0.023	0.673±0.023
	LapSVM1	0.244±0.044	0.464±0.049	0.585±0.033	0.641±0.034	0.672±0.022	0.682±0.031
	LapSVM2	0.264±0.050	0.473±0.038	0.516±0.071	0.527±0.080	0.602±0.069	0.628±0.051
	LapSVM3	0.240±0.051	0.458±0.044	0.526±0.059	0.530±0.090	0.609±0.057	0.623±0.050
	SSDRB	**0.717±0.017**	0.754±0.013	0.779±0.013	0.792±0.007	0.803±0.008	0.810±0.009
	HP	0.635±0.026	0.705±0.023	0.749±0.017	0.774±0.013	0.785±0.011	0.800±0.009
	DRB	0.635±0.026	0.705±0.023	0.748±0.017	0.773±0.013	0.783±0.013	0.798±0.010
	SVM	0.391±0.038	0.524±0.033	0.596±0.032	0.647±0.022	0.672±0.026	0.702±0.021
	KNN	0.503±0.029	0.586±0.027	0.637±0.021	0.679±0.016	0.687±0.022	0.714±0.015
Caltech256	STHP	0.426±0.017	0.524±0.012	**0.563±0.004**	**0.590±0.007**	**0.604±0.007**	**0.617±0.004**
	AGR	0.327±0.029	0.369±0.043	0.404±0.031	0.426±0.028	0.443±0.026	0.460±0.014
	AGRL	0.337±0.040	0.393±0.024	0.418±0.022	0.445±0.022	0.453±0.017	0.470±0.012
	LGC	Out of system memory					
	GMMC	Out of system memory					
	LapSVM1	0.140±0.013	0.262±0.033	0.349±0.028	0.391±0.048	0.403±0.031	0.427±0.040
	LapSVM2	0.149±0.013	0.239±0.025	0.314±0.025	0.375±0.034	0.415±0.027	0.467±0.010
	LapSVM3	0.139±0.021	0.249±0.035	0.290±0.036	0.319±0.047	0.391±0.039	0.450±0.034
	SSDRB	**0.465±0.012**	**0.526±0.010**	0.554±0.007	0.578±0.007	0.593±0.005	0.602±0.005
	HP	0.363±0.016	0.446±0.009	0.488±0.006	0.524±0.006	0.540±0.006	0.555±0.004
	DRB	0.363±0.016	0.446±0.009	0.488±0.006	0.523±0.006	0.539±0.006	0.554±0.004
	SVM	Out of system memory					
	KNN	0.274±0.012	0.350±0.009	0.392±0.008	0.424±0.006	0.444±0.006	0.462±0.006

From Table 9 one can see that, the proposed STHP approach is able to outperform or, at least, be on par with alterative semi-supervised and supervised classification approaches on the two challenging benchmark problems under different experimental settings. This demonstrates the promise of the STHP classifier as a powerful semi-supervised learning technique.

Moreover, it can be observed from Tables 7 and 8 that thanks to both **Conditions 2** and **3**, the self-training mechanism of the STHP classifier is generally more effective compared with the SSDRB classifier in terms of classification accuracy improvement, especially for the large-scale, complex problems. In addition, the advantage is more obvious if more labelled samples are given during the classifier priming stage. For example, in the experiments with Caltech101 and Caltech256 datasets with $L^i = 6$, the classification accuracy of the STHP classifier increases from 79.98% to 82.70% on Caltech101 dataset, and from 55.45% to 61.68% on Caltech256 dataset through self-training. In contrast, the classification accuracy of the SSDRB classifier increases from 79.76% to 81.04% on Caltech101 dataset, which is 60% less than STHP, and from 55.43% to 60.20% on Caltech256 dataset through self-training, which is around 30% less.

Based on the numerical examples and discussions presented in this section, one may conclude that the proposed approach is a strong alternative to the state-of-the-art approaches.

7. Conclusion and Future Works

This paper presented a novel approach for semi-supervised classification by extending the recently introduced HP classifier with a self-training mechanism based on the widely used pseudo-label technique. After being primed with labelled samples, the STHP classifier can continue to self-evolve its multi-layered structure from unlabelled samples via pseudo-labelling without human supervision. Compared with alternative semi-supervised learning approaches, unique features of the proposed approach include:

1) a highly transparent multi-layered system structure self-organized from both labelled and unlabelled samples;

2) a fully traceable, explainable self-training and decision-making mechanism;

3) the capability to self-learn from labelled and pseudo-labelled samples in a non-iterative manner;

4) the ability of visualizing the learned knowledge at multiple levels of specificity.

Numerical examples on various benchmark image classification problems demonstrate that the proposed STHP classifier can perform highly accurate classification given very few labelled samples surpassing or, at least, on par with the state-of-the-art semi-supervised learning approaches.

As future work, there are several considerations. Firstly, the self-training process of the STHP classifier presented in this paper is mostly limited to offline scenarios. Despite that the proposed approach can work in online scenarios on a chunk-by-chunk basis thanks to the online learning ability of the base learner, it will be more useful to develop an online, sample-by-sample self-training mechanism for STHP. Secondly, it will be a strong novelty if the STHP classifier can autonomously recognize unfamiliar data patterns within the unlabelled set and actively add them as new classes. This will also give STHP the capability to autonomously spot anomalies from unlabelled samples. Thirdly, users have to determine the layer number for the current STHP classifier. It will be a valuable modification if the STHP classifier can self-determine the optimal layer number during the learning process. Finally, the optimality of the STHP classifier needs to be analysed. It will be interesting to see how the classifier performs if all prototypes are optimized, e.g., by evolutionary computation algorithms, to the locally optimal positions.

References

[1] M. Belkin, P. Niyogi, and V. Sindhwani, "Manifold regularization: a geometric framework for learning from labeled and unlabeled examples," J. Mach. Learn. Res., vol. 7, no. 2006, pp. 2399–2434, 2006.

[2] K. P. Bennett and A. Demiriz, "Semi-supervised support vector machines," in Advances in Neural Information Processing Systems, 1999, pp. 368–374.

[3] A. Blum and S. Chawla, "Learning from labeled and unlabeled data using graph mincut," in International Conference on Machine Learning, 2001, pp. 19–26.

[4] V. Cheplygina, M. de Bruijne, and J. P. W. Pluim, "Not-so-supervised: a survey of semi-supervised, multi-instance, and transfer learning in medical image analysis," Med. Image Anal., vol. 54, pp. 280–296, 2019.

[5] N. Cristianini and J. Shawe-Taylor, An introduction to support vector machines and other kernel-based learning methods. Cambridge: Cambridge University Press, 2000.
[6] P. Cunningham and S. J. Delany, “K-nearest neighbour classifiers,” Mult. Classif. Syst., vol. 34, pp. 1–17, 2007.
[7] X. Gu and P. P. Angelov, “Semi-supervised deep rule-based approach for image classification,” Appl. Soft Comput., vol. 68, pp. 53–68, 2018.
[8] X. Gu, P. P. Angelov, C. Zhang, and P. M. Atkinson, “A massively parallel deep rule-based ensemble classifier for remote sensing scenes,” IEEE Geosci. Remote Sens. Lett., vol. 15, no. 3, pp. 345–349, 2018.
[9] X. Gu and W. Ding, “A hierarchical prototype-based approach for classification,” Inf. Sci. (Ny)., vol. 505, pp. 325–351, 2019.
[10] M. Guillaumin, J. J. Verbeek, and C. Schmid, “Multimodal semi-supervised learning for image classification,” in IEEE Conference on Computer Vision & Pattern Recognition, 2010, pp. 902–909.
[11] H. Hagras, “Toward human-understandable, explainable AI,” Computer (Long. Beach. Calif)., vol. 51, no. 9, pp. 28–36, 2018.
[12] T. K. Ho and M. Basu, “Complexity measures of supervised classification problems,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 24, no. 3, pp. 289–300, 2002.
[13] A. Iscen, G. Tolias, Y. Avrithis, and O. Chum, “Label propagation for deep semi-supervised learning,” in IEEE Conference on Computer Vision and Pattern Recognition, 2019, pp. 5070–5079.
[14] A. Iwayemi and C. Zhou, “SARAA: semi-supervised learning for automated residential appliance annotation,” IEEE Trans. Smart Grid, vol. 8, no. 2, p. 779, 2017.
[15] B. Jiang, H. Chen, B. Yuan, and X. Yao, “Scalable graph-based semi-supervised learning through sparse bayesian model,” IEEE Trans. Knowl. Data Eng., vol. 29, no. 12, pp. 2758–2771, 2017.
[16] M. Kim, D. Lee, and H. Shin, “Semi-supervised learning for hierarchically structured networks,” Pattern Recognit., vol. 95, pp. 191–200, 2019.
[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” in Advances In Neural Information Processing Systems, 2012, pp. 1097–1105.
[18] D.-H. Lee, “Pseudo-label: the simple and efficient semi-supervised learning method for deep neural networks,” in Workshop on Challenges in Representation Learning, ICML, 2013, p. 2.
[19] Y. Li, C. Guan, H. Li, and Z. Chin, “A self-training semi-supervised SVM algorithm and its application in an EEG-based brain computer interface speller system,” Pattern Recognit. Lett., vol. 29, no. 9, pp. 1285–1294, 2008.
[20] Y. F. Li, J. T. Kwok, and Z. H. Zhou, “Semi-supervised learning using label mean,” in International Conference on Machine Learning, 2009, pp. 633–640.
[21] Y. F. Li and Z. H. Zhou, “Towards making unlabeled data never hurt,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 37, no. 1, pp. 175–188, 2015.
[22] W. Liu, J. He, and S.-F. Chang, “Large graph construction for scalable semi-supervised learning,” in International Conference on Machine Learning, 2010, pp. 679–689.
[23] P. K. Mallapragada, R. Jin, A. K. Jain, and Y. Liu, “SemiBoost: boosting for semi-supervised learning,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 31, no. 11, pp. 2000–2014, 2008.
[24] U. Maulik and D. Chakraborty, “A self-trained ensemble with semisupervised SVM: an application to pixel classification of remote sensing imagery,” Pattern Recognit., vol. 44, no. 3, pp. 615–623, 2011.
[25] S. Mehrkanoon, C. Alzate, R. Mall, R. Langone, and J. A. K. Suykens, “Multiclass semisupervised learning based upon kernel spectral clustering,” IEEE Trans. Neural Networks Learn. Syst., vol. 26, no. 4, pp. 720–733, 2015.
[26] A. Oliva and A. Torralba, “Modeling the shape of the scene: a holistic representation of the spatial envelope,” Int. J. Comput. Vis., vol. 42, no. 3, pp. 145–175, 2001.
[27] A. B. Penatti, K. Nogueira, and J. A. Santos, “Do deep features generalize from everyday objects to remote sensing and aerial scenes domains ?,” in IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 44–51.
[28] S. Qiao, W. Shen, Z. Zhang, B. Wang, and A. Yuille, “Deep co-training for semi-supervised image recognition,” in European Conference on Computer Vision, 2018, pp. 135–152.

[29] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” in International Conference on Learning Representations, 2015, pp. 1–14.
[30] I. Škrjanc, J. Iglesias, A. Sanchis, D. Leite, E. Lughofer, and F. Gomide, “Evolving fuzzy and neuro-fuzzy approaches in clustering, regression, identification, and classification: a survey,” Inf. Sci. (Ny)., vol. 490, pp. 344–368, 2019.
[31] R. G. F. Soares, H. Chen, and X. Yao, “Semisupervised classification with cluster regularization,” IEEE Trans. Neural Networks Learn. Syst., vol. 23, no. 11, pp. 1779–1792, 2012.
[32] C. Szegedy et al., “Going deeper with convolutions,” in IEEE conference on Computer Vision and Pattern Recognition, 2015, pp. 1–9.
[33] J. Tanha, M. van Someren, and H. Afsarmanesh, “Semi-supervised self-training for decision tree classifiers,” Int. J. Mach. Learn. Cybern., vol. 8, no. 1, pp. 355–370, 2017.
[34] J. Thorsten, “Transductive inference for text classification using support vector machines,” in International conference on Machine learning, 1999, pp. 200–209.
[35] I. Triguero, J. A. Sáez, J. Luengo, S. García, and F. Herrera, “On the characterization of noise filters for self-training semi-supervised in nearest neighbor classification,” Neurocomputing, vol. 132, pp. 30–41, 2014.
[36] Z. Wang, B. Du, L. Zhang, L. Zhang, and X. Jia, “A novel semisupervised active-learning algorithm for hyperspectral image classification,” IEEE Trans. Geosci. Remote Sens., vol. 55, no. 6, pp. 3071–3083, 2017.
[37] M. Wang, W. Fu, S. Hao, D. Tao, and X. Wu, “Scalable semi-supervised learning by efficient anchor graph regularization,” IEEE Trans. Knowl. Data Eng., vol. 28, no. 7, pp. 1864–1877, 2016.
[38] J. Wang, T. Jebara, and S. F. Chang, “Semi-supervised learning using greedy Max-Cut,” J. Mach. Learn. Res., vol. 14, pp. 771–800, 2013.
[39] W. Wang, H. Wang, Z. Zhang, C. Zhang, and Y. Gao, “Semi-supervised domain adaptation via Fredholm integral based kernel methods,” Pattern Recognit., vol. 85, pp. 185–197, 2019.
[40] Y. Wang, X. Xu, H. Zhao, and Z. Hua, “Semi-supervised learning based on nearest neighbor rule and cut edges,” Knowledge-Based Syst., vol. 23, no. 6, pp. 547–554, 2010.
[41] H. Wu and S. Prasad, “Semi-supervised deep learning using pseudo labels for hyperspectral image classification,” IEEE Trans. Image Process., vol. 27, no. 3, pp. 1259–1270, 2018.
[42] D. Wu et al., “Self-training semi-supervised classification based on density peaks of data,” Neurocomputing, vol. 275, pp. 180–191, 2018.
[43] G. Xia et al., “AID: a benchmark dataset for performance evaluation of aerial scene classification,” IEEE Trans. Geosci. Remote Sens., vol. 55, no. 7, pp. 3965–3981, 2017.
[44] S. Xiang, F. Nie, and C. Zhang, “Semi-supervised classification via local spline regression,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 32, no. 11, pp. 2039–2053, 2010.
[45] Y. M. Zhang, K. Huang, G. G. Geng, and C. L. Liu, “MTC: a fast and robust graph-based transductive learning method,” IEEE Trans. Neural Networks Learn. Syst., vol. 26, no. 9, pp. 1979–1991, 2015.
[46] D. Zhou, O. Bousquet, T. N. Lal, J. Weston, and B. Schölkopf, “Learning with local and global consistency,” in Adv. Neural. Inform. Process Syst, 2004, pp. 321–328.
[47] Z. H. Zhou and M. Li, “Semi-supervised regression with co-training,” in International Joint Conference on Artificial Intelligence, 2005, pp. 908–913.
[48] X. J. Zhu, “Semi-supervised learning literature survey,” 2005.
[49] X. Zhu, Z. Ghahraman, and J. D. Lafferty, “Semi-supervised learning using gaussian fields and harmonic functions,” in International conference on Machine learning, 2003, pp. 912–919.