



Article

Few-Shot Hyperspectral Remote Sensing Image Classification via an Ensemble of Meta-Optimizers with Update Integration

Tao Hao ¹, Zhihua Zhang ^{1,*} and M. James C. Crabbe ² ¹ Interdisciplinary Data Mining Group, School of Mathematics, Shandong University, Jinan 250100, China² Wolfson College, University of Oxford, Oxford OX2 6UD, UK; james.crabbe@wolfson.ox.ac.uk

* Correspondence: zhangzhihua@sdu.edu.cn

Abstract: Hyperspectral images (HSIs) with abundant spectra and high spatial resolution can satisfy the demand for the classification of adjacent homogeneous regions and accurately determine their specific land-cover classes. Due to the potentially large variance within the same class in hyperspectral images, classifying HSIs with limited training samples (i.e., few-shot HSI classification) has become especially difficult. To solve this issue without adding training costs, we propose an ensemble of meta-optimizers that were generated one by one through utilizing periodic annealing on the learning rate during the meta-training process. Such a combination of meta-learning and ensemble learning demonstrates a powerful ability to optimize the deep network on few-shot HSI training. In order to further improve the classification performance, we introduced a novel update integration process to determine the most appropriate update for network parameters during the model training process. Compared with popular human-designed optimizers (Adam, AdaGrad, RMSprop, SGD, etc.), our proposed model performed better in convergence speed, final loss value, overall accuracy, average accuracy, and Kappa coefficient on five HSI benchmarks in a few-shot learning setting.

Keywords: few-shot HSI classification; meta-optimizer ensemble; meta-learning; update integration



Citation: Hao, T.; Zhang, Z.; Crabbe, M.J.C. Few-Shot Hyperspectral Remote Sensing Image Classification via an Ensemble of Meta-Optimizers with Update Integration. *Remote Sens.* **2024**, *16*, 2988. <https://doi.org/10.3390/rs16162988>

Academic Editors: Jie Feng and Salah Bourennane

Received: 7 May 2024

Revised: 29 July 2024

Accepted: 13 August 2024

Published: 14 August 2024



Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Hyperspectral remote sensing is a valuable monitoring technique in geoinformation that typically captures hundreds of narrow-spectral-band sets of information from the same region, revealing unique physical features of ground targets. The high spectral dimensions and spatial resolution in the hyperspectral imaging (HSI) process make accurate discrimination of different land-cover classes possible [1–3]. Since the HSI is formed pixel-by-pixel, its classification is to assign a unique label, representing a specific area, to each HSI pixel. At present, HSI classification has become an indispensable data-driven technique to achieve the United Nations Sustainable Development Goals (SDGs). For example, in agricultural settings, HSI classification is used to accurately capture the status of vegetation health, soil type, and moisture and then help to make reasonable decisions on fertilization, irrigation, and management [4,5]; in geological exploration, HSI classification is used to identify geological features, such as mineral and rock types, which aids in finding potential mineral deposits [6]; in environmental protection, HSI classification is used to monitor desertification processes and changes in land use [7–11]. HSI classification can also monitor natural disasters such as fires, floods, and earthquakes, making disaster response and mitigation efforts simpler and quicker [12,13].

Traditional HSI classifiers rely heavily on hand-crafted features [14–17]. Benediktsson et al. [18] constructed morphological profile features with the help of mathematical morphology. Jia et al. [19] used 3D Gabor filtering to extract spatial structure information. These carefully designed features were then fed into classifiers, such as support vector machines (SVMs) [20] and random forests [21]. Feature extraction techniques, such as principal component analysis (PCA) [22], independent component analysis (ICA) [23],

and linear discriminant analysis (LDA) [24], were also introduced to reduce redundant spectral information and produced more distinguishable features. By incorporating spatial contextual information, spectral–spatial features can further improve the classification performance [17,25]. Compared with traditional techniques, deep learning can automatically learn and capture high-level features from training data and achieve better classification accuracy [26–30]. Chen et al. [26] inputted spectral vectors and spatial features extracted by PCA into a stacked autoencoder to generate high-level joint spectral–spatial features and then implemented the classification by a logistic regression layer and achieved more accurate results than by traditional machine learning on KSC and Pavia HSI datasets. Mei et al. [30] used a 3D convolutional network to extract spatial and spectral features simultaneously and then achieved better results on Indian Pine, Salinas, and Pavia HSI datasets.

Manually labeling HSI samples is not only time-consuming and laborious but also requires much professional expertise. Due to the high annotation cost, only a limited number of annotated samples may be available, leading to a small training dataset size. Moreover, in hyperspectral images, the number of pixels for different land-cover classes can be significantly imbalanced, with some classes having far fewer samples than others. Severe overfitting occurs when training neural networks with insufficient samples. Current HSI classification faces the challenge of few-shot learning [31]. As an advance of deep learning, meta-learning [32,33] has become the most promising approach to deal with the issue of few-shot learning.

The idea of meta-learning is to enable models to not only acquire task-specific knowledge but also learn how to learn new tasks more effectively, so it allows models to make better use of limited training samples and demonstrate robust generalization ability on new tasks [34–36]. Although meta-learning has begun to be used in few-shot HSI classification [2,37], the impacts of certain optimizers on the final classification performance are often ignored. Since the popular human-designed optimizers, such as stochastic gradient descent (SGD) [38], RMSprop [39], AdaGrad [40], and Adam [41], are designed for general tasks, these optimizers cannot improve their performance through mining unique features of a specific HSI task, and they can possibly lead to serious overfitting, especially when dealing with insufficient training samples [42]. Incorporating prior knowledge of the training task into an optimizer can significantly reduce the size of the required training data and avoid overfitting [43,44]. Such an optimizer is called a meta-optimizer or meta-learning-based optimizer [45].

This study aimed to improve meta-optimizers and apply them in few-shot HSI classification. Since ensemble learning can improve the generalization ability by combining individual learners and then reduce overfitting risks and enhance the model robustness, we propose an ensemble of meta-optimizers that were generated one by one through utilizing periodic annealing on the learning rate during the meta-training process. Such a combination of meta-learning and ensemble learning not only has no significant added training costs but also demonstrates a powerful ability to optimize the deep network on few-shot HSI training. In each optimization iteration, a meta-optimizer ensemble can generate several candidates to update the parameters of the trained model. In order to integrate these candidate updates to effectively improve the model performance, we further developed an update integration technique to measure and incorporate the potential of each candidate. The real-world experiments on few-shot HSI classification consistently verified the effectiveness of our meta-optimizer ensemble with update integration over the widely used human-designed optimizers across five benchmark HSI datasets.

2. Related Works

During the HSI classification process, when sufficient labeled samples are provided, accurate classification can be easily implemented under a supervised classification framework [46]. Let a learning task $\mathcal{T}$ be specified by a dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$ and a loss function $\mathcal{L}$. The dataset $\mathcal{D}$ is split into a training set $\mathcal{D}^{train} = \{(x_i, y_i)\}_{i=1}^k$ and a test

set $\mathcal{D}^{test} = \{(x_i, y_i)\}_{i=k+1}^n$. The learning target is to obtain a predictive function f with parameters θ by solving the following optimization equation:

$$\theta^* = \operatorname{argmin}_{\theta} \sum_{(x,y) \in \mathcal{D}^{train}} \mathcal{L}(f_{\theta}(x), y) \quad (1)$$

where the sample subscript i is omitted for simplicity. The performance of f is finally evaluated on $\mathcal{D}^{test}$. When only limited labeled HSI samples are provided, Equation (1) is prone to becoming stuck in local optima.

Meta-learning aims to make the model learn how to learn so that it can more flexibly adapt to various learning scenarios. It consists of the learner, which is responsible for specific task learning, and the meta-learner, which guides the learner's learning process. The training of the meta-learner typically involves two stages: the first stage is to learn across a set of tasks, and the second stage is to evaluate and adjust the meta-learner on new tasks [42]. The optimal meta-knowledge ω^* is found by

$$\omega^* = \operatorname{argmin}_{\omega} E_{\mathcal{T} \sim p(\mathcal{T})} [\mathcal{L}(\omega; \mathcal{D})] \quad (2)$$

where $p(\mathcal{T})$ is a distribution of tasks, and a task specifies a dataset and loss function $\mathcal{T} = \{\mathcal{D}, \mathcal{L}\}$. In an actual implementation, only M source tasks $\mathcal{D}_{source} = \{(\mathcal{D}_{source}^{train}, \mathcal{D}_{source}^{test})^{(i)}\}_{i=1}^M$ are randomly sampled from the task distribution $p(\mathcal{T})$, so the expectation in the meta-training phase (2) is replaced simply by summation:

$$\omega^* = \operatorname{argmin}_{\omega} \sum_{i=1}^M \mathcal{L}(\mathcal{D}_{source}^{(i)}; \omega). \quad (3)$$

This meta-knowledge ω^* is further used to train the predictive model f with parameter θ on Q target tasks $\mathcal{D}_{target} = \{(\mathcal{D}_{target}^{train}, \mathcal{D}_{target}^{test})^{(i)}\}_{i=1}^Q$. For each target task i , the meta-testing stage is written as follows:

$$\theta^{*(i)} = \operatorname{argmin}_{\theta} \mathcal{L}(\mathcal{D}_{target}^{(i)}; \theta; \omega^*). \quad (4)$$

Compared with Equation (1) in typical supervised learning, the learning phase of Equation (4) benefits from the learned meta-knowledge ω^* .

The meta-knowledge ω^* can be in the form of the initial parameters [32] or optimization strategy [44], etc. By treating meta-knowledge ω^* as the optimization strategy, learning to optimize is proposed to optimize the learning process itself [47]. Specifically, Andrychowicz et al. [43] trained a two-layer LSTM network (LSTM optimizer) to generate dynamic updates. The idea is to replace the regular update with an update generated by an LSTM network. Guided by the meta-knowledge stored in the network parameters, the LSTM optimizer can incorporate historical gradient information to generate updates. Andrychowicz et al. [43] demonstrated that the LSTM optimizer outperforms the human-designed optimizers in a variety of tasks, including training neural networks, convex problems, and styling images with neural art. Ravi et al. [44] used LSTM networks to learn update rules for few-shot learning and set the cell state of LSTM networks as the learner's parameters, and the candidate cell state determined the gradient information for parameter updates. Li et al. [48] proposed a conceptually simple meta-optimizer, Meta-SGD, for few-shot learning. Meta-SGD learns the initialization and learning rate instead of the whole update rules. Compared with the LSTM optimizer, Meta-SGD is more straightforward to implement, but its generalization capacity needs to be improved. Chen et al. [49] trained an RNN-based meta-optimizer for global optimization of black-box functions. Their meta-optimizer, trained on synthetic functions, can optimize a broad class of black-box functions. Wang et al. [45] designed a meta-optimizer called HyperAdam, the parameter update

generated by which is an adaptive combination of multiple candidate updates produced by Adam using different decay rates.

Hyperspectral images have high spectral dimensionality and high spatial complexity, which may slow down the convergence speed of classification during training. A meta-optimizer can learn some knowledge from an HSI dataset and then use the learned knowledge to optimize a model on a new HSI dataset. To the best of our knowledge, no study has focused on learning a meta-optimizer for few-shot HSI classification.

3. Proposed Method

Hyperspectral images (HSIs) with abundant spectra and high spatial resolution can satisfy the demand for accurately determining specific land-cover classes. Due to the significant variance within the same class in hyperspectral images, few-shot HSI classification becomes especially difficult. To solve this issue without adding training costs, we propose a new approach: an ensemble of meta-optimizers with update integration for few-shot HSI classification.

3.1. Ensembles of Meta-Optimizers

The training in conventional machine learning can be expressed as the optimization in Equation (1). Its solution is always based on stochastic gradient descent and its variants through the following iterative process:

$$\theta_{t+1} = \theta_t - \alpha_t \cdot \nabla_{\theta_t} \mathcal{L} \quad (5)$$

where α_t is the learning rate at the time step t . Since Equation (5) usually requires thousands of iterations to find the optimum or local optimum, the performance of these optimizers will become very weak if the training samples are insufficient. To overcome this drawback, the meta-optimizer (i.e., an optimizer trained by meta-learning) is used to replace the human-designed stochastic gradient descent algorithms. It may take the form of a two-layer long short-term memory (LSTM) network, which can integrate information from the history of gradients to determine the parameter update. Such a meta-optimizer is also called an LSTM optimizer. Denote the hidden state of the LSTM with h and the output with g , and by using the LSTM optimizer G_ϕ to optimize the parameters θ of a model f , the sequence of updates will become

$$(g_t, h_t) = G_\phi(\nabla_{\theta_t} \mathcal{L}, h_{t-1}) \quad (6)$$

$$\theta_{t+1} = \theta_t + g_t. \quad (7)$$

In order to solve the challenge that different input gradients have very different magnitudes, making the meta-optimizers difficult to train, we adopted the standard normalization process in [43] to rescale the input gradients:

$$x \rightarrow \begin{cases} \left(\frac{\log(|x|)}{p}, \text{sgn}(x) \right) & \text{if } |x| \geq e^{-p} \\ (-1, e^p x) & \text{otherwise} \end{cases} \quad (8)$$

where $p > 0$ is the parameter controlling how small gradients are disregarded.

Minimizing the loss function

$$\mathcal{L}_{meta}(\phi) = E_f[\sum_k^T \mathcal{L}(\theta_k; \mathcal{D})]. \quad (9)$$

can obtain the optimal parameters ϕ^* of meta-optimizer G , which can reduce the objective function $\mathcal{L}(\theta_i; \mathcal{D})$ as much as possible.

To build an ensemble of meta-optimizers without adding training costs, we performed periodic annealing on the learning rate. When the learning rate anneals to the minimum value, the meta-optimizer parameters are saved and added to the ensemble. In this way, we can obtain multiple meta-optimizers with different parameters without increasing

the training costs. The learning rate α changes in each annealing cycle according to the following formula:

$$\alpha = \alpha_{min} + \frac{1}{2}(\alpha_{max} - \alpha_{min})(1 + \cos(\frac{t_{cur}}{T_p})) \quad (10)$$

where α_{max} and α_{min} indicate the learning rate range, and t_{cur} is the number of iterations since the last restart. A good α_{max} can accelerate the training process and help the model escape from local minima. The α_{min} is a small enough number. The T_p is a hyper-parameter representing the period of cosine annealing. When the training process starts, $t_{cur} = 0$ and $\alpha = \alpha_{max}$. After T_p iterations, the learning rate will decrease to its minimum α_{min} , and one annealing cycle will be completed. At the beginning of the next annealing cycle, t_{cur} will become 0 again, and the learning rate will abruptly become α_{max} .

When the learning rate is small, the trained model tends to converge into the closest local minimum [50]. Once the learning rate reaches its minimum α_{min} , the corresponding meta-optimizer will be added to the meta-optimizer ensemble. Then, a large enough learning rate α_{max} is used to escape the current local minimum and restart a new annealing cycle. After N annealing cycles, we obtain an ensemble of N meta-optimizers, denoted as $MOE_N = \{G_{\phi_i}\}_{i=1}^N$.

The ensemble process of meta-optimizers with learning rate annealing is shown in Algorithm 1. The whole meta-training phase is a double-loop structure, which differs from the traditional training process.

Algorithm 1 The proposed ensemble of meta-optimizers.

Require:

- 1: The predictive model f with initial parameters θ_0
- 2: Source HSI dataset $\mathcal{D}_{source} = \{(x_i, y_i)\}_{i=1}^N$; batch size B .
- 3: Meta-optimizer G with initial parameters ϕ_0 , $MOE_N = \emptyset$.
- 4: Learning rate α , learning rate range α_{max} and α_{min} ; annealing period T_p .
- 5: loss function $\mathcal{L}$; meta-loss function $\mathcal{L}_{meta}$.

Begin:

```

6: While  $t_{cur} = 0, 1, 2, \dots$  do
7:    $\mathcal{L}_{meta} = 0$ 
8:    $\alpha = \alpha_{min} + \frac{1}{2}(\alpha_{max} - \alpha_{min})(1 + \cos(\frac{t_{cur}}{T_p}))$ 
9:   for  $t = 1, 2, \dots, T$  do
10:    Randomly draw  $B$  training samples  $\mathcal{D}_B = \{(x_i, y_i)\}_{i=1}^B$  from  $\mathcal{D}_{source}$ 
11:    Calculate the current loss value  $\mathcal{L}_t$  on  $\mathcal{D}_B$ 
12:    Calculate the gradients  $\nabla_{\theta_t} \mathcal{L}_t$  of  $f$ 
13:    Normalize  $\nabla_{\theta_t} \mathcal{L}_t \rightarrow [\nabla_{\theta_t} \mathcal{L}_t]_{norm}$  according to Equation (8)
14:     $(g_t, h_t) \leftarrow G_{\phi}([\nabla_{\theta_t} \mathcal{L}_t]_{norm}, h_{t-1})$ 
15:     $\theta_{t+1} \leftarrow \theta_t + g_t$ 
16:     $\mathcal{L}_{meta} \leftarrow \mathcal{L}_{meta} + \mathcal{L}_t$ 
17:  end for
18:  Calculate  $\nabla \mathcal{L}_{meta}(\phi)$ 
19:   $\phi \leftarrow \phi - \alpha \cdot \nabla \mathcal{L}_{meta}(\phi)$ 
20:  If  $\alpha = \alpha_{min}$  then
21:    Add  $G_{\phi}$  to the  $MOE_N$ 
22:     $t_{cur} = 0$ 
23:  end while
24: return  $MOE_N = \{G_{\phi_i}\}_{i=1}^N$ 
end

```

3.2. Update Integration

When optimizing a neural network, the meta-optimizer ensemble $MOE_N = \{G_{\phi_i}\}_{i=1}^N$ can generate N candidate updates in each optimization iteration. To optimally integrate the candidate updates $\{g^i\}_{i=1}^N$ into the final update g^* and achieve faster convergence,

we propose an update integration algorithm that can select a candidate update as the final update by estimating the quality of each update. The algorithm's core lies in how to measure the quality of each update. Typically, a better update at step t results in lower loss values for the predictive model from step $t + 1$ to the final step T . However, considering too many subsequent loss values is computationally expensive. As a compromise, only the loss value at step $t + 1$ is used to quantitatively measure the update at step t . At the same time, historical loss values are also used as a reference for measurement, since the meta-optimizer that produces better updates earlier will often have a higher probability of producing a better update later, and historical loss information can help reduce the stochasticity in calculating loss values using batch data. The details of the proposed update integration algorithm are as follows:

Suppose that the predictive model to be updated is f_θ . At the time step t , we sample a batch of samples from the target dataset and calculate the normalized gradients $[\nabla_{\theta_t} \mathcal{L}]_{norm}$. The normalized gradients are then inputted into the MOE_N to obtain N candidate updates $\{g_t^i\}_{i=1}^N$. Then, we sample another mini-batch of training data $\mathcal{D}_{mini} = \{(x_i, y_i)\}_{i=1}^b$ (b is a small number) to calculate N new loss values $\mathcal{L}(\theta_t + g_t^i; \mathcal{D}_{mini})$ ($i = 1, 2, \dots, N$) and the following equations:

$$m_t^i = \beta \cdot m_{t-1}^i + (1 - \beta) \cdot [\mathcal{L}(\theta_t + g_t^i; \mathcal{D}_{mini}) - \mathcal{L}(\theta_t)] \quad i = 1, 2, \dots, N, \quad (11)$$

$$\begin{cases} i^* = \underset{i}{\operatorname{argmin}} m_t^i \\ g_t^* = g_t^{i^*} \end{cases} \quad (12)$$

where $\beta \in [0, 1)$ is a hyper-parameter. Based on the newly sampled $\mathcal{D}_{mini}$, $\mathcal{L}(\theta_t + g_t^i; \mathcal{D}_{mini})$ can approximate the loss values $\mathcal{L}(\theta_{t+1})$ at step $t + 1$ by assuming g_t^i as the actual update. Consequently, $\mathcal{L}(\theta_t + g_t^i; \mathcal{D}_{mini}) - \mathcal{L}(\theta_t)$ measures which candidate update can drop the loss value the most. Each m_t^i is a real number iterating over the time step t , and its initial value is set to 0. The update integration algorithm determines the optimal update g_t^* at the time step t according to the minimum among m_t^i ($i = 1, 2, \dots, N$), as suggested in Equation (12). When $\beta = 0$, Equation (11) will become $m_t^i = [\mathcal{L}(\theta_t + g_t^i; \mathcal{D}_{mini}) - \mathcal{L}(\theta_t)]$. This means that the update integration algorithm determines the optimal update based only on the current loss reduction $\mathcal{L}(\theta_t + g_t^i; \mathcal{D}_{mini}) - \mathcal{L}(\theta_t)$. However, loss values can vary widely from one mini-batch of data to another, so we propose calculating the loss reductions with previous loss information to eliminate the stochasticity, which corresponds to the cases of $\beta \neq 0$ in Equation (11).

Averaging of the outputs of an ensemble is widely used. Compared with the average method, our proposed update integration algorithm can flexibly choose an update direction with a lower loss value by looking one step ahead at the loss changes.

3.3. Incorporating Update Integration into Meta-Optimizer Ensembles

The framework of the proposed method is shown in Figure 1. In each optimization iteration, we first input a batch of training samples to the neural network to be optimized and perform backpropagation to obtain the gradient information $\nabla_{\theta} \mathcal{L}$ of the neural network. The meta-optimizer ensemble $\{G_{\phi_i}\}_{i=1}^N$ then takes the normalized gradients $[\nabla_{\theta} \mathcal{L}]_{norm}$ as inputs, and each meta-optimizer G_{ϕ_i} outputs its suggested update g^i independently, directed by its parameters ϕ_i . Finally, the candidate updates $\{g^i\}_{i=1}^N$ are integrated into the final update g^* by using the proposed update integration algorithm.

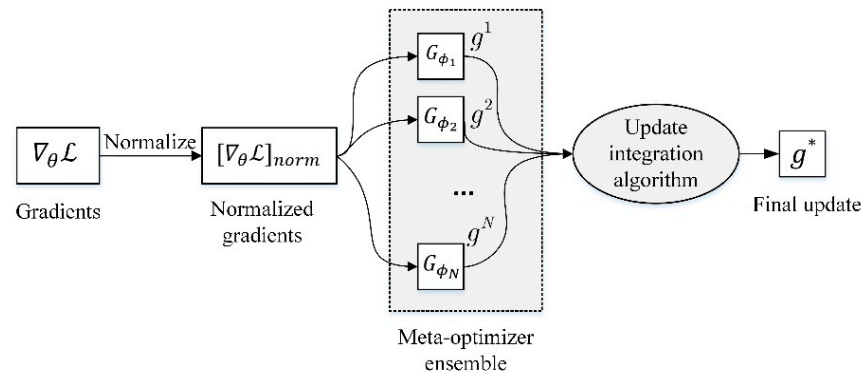


Figure 1. The framework of our proposed method.

Since the meta-optimizer ensemble has learned how to optimize from related tasks, it requires fewer training samples, leading to low computational costs and memory requirements and faster convergence in fewer iterations. Algorithm 2 illustrates the total pseudo-code for using the proposed meta-optimizer ensemble with the update integration algorithm to optimize a model f .

Algorithm 2 The proposed update integration algorithm.

Require:

1: The trained model f with initial parameters θ_1 .

2: Target dataset $\mathcal{D}_{target} = \{(x_j, y_j)\}_{j=1}^K$; batch size B , mini-batch size b .

3: $MOE_N = \{G_{\phi_i}\}_{i=1}^N$, hyper-parameter β .

Initialize: $m_0^i = \mathbf{0}$ $i = 1, 2, \dots, N$

4: **for** $t = 1, 2, \dots, T$ **do**

5: Draw random batch training data $\mathcal{D}^{train} = \{(x_j, y_j)\}_{j=1}^B$ from $\mathcal{D}_{target}$

6: Calculate the gradients $\nabla_{\theta_t} \mathcal{L}_t$ on $\mathcal{D}^{train}$

7: Normalize $\nabla_{\theta_t} \mathcal{L}_t \rightarrow [\nabla_{\theta_t} \mathcal{L}_t]_{norm}$ according to Equation (8)

8: Draw another random mini-batch $\mathcal{D}_{mini} = \{(x_j, y_j)\}_{j=1}^b$ from $\mathcal{D}_{target}$

9: **for** $i = 1, 2, \dots, N$ **do**

10: $g_t^i \leftarrow G_{\phi_i}([\nabla_{\theta_t} \mathcal{L}_t]_{norm}, h_{t-1}^i)$

11: Calculate $\mathcal{L}(\theta_t + g_t^i; \mathcal{D}_{mini})$

12: $m_t^i = \beta \cdot m_{t-1}^i + (1 - \beta) \cdot [\mathcal{L}(\theta_t + g_t^i; \mathcal{D}_{mini}) - \mathcal{L}(\theta_t)]$

13: **end for**

14: $i^* = \underset{i}{\operatorname{argmin}} m_t^i$

15: $g_t^* = g_t^{i^*}$

16: $\theta_{t+1} = \theta_t + g_t^*$

17: **end for**

18: **return** the final parameters θ_{T+1}

4. Real-World HSI Classification Experiments

We focused the real-world HSI classification under the N-way 10-shot learning setting, i.e., we randomly picked 10 samples from each of N land cover classes to form the training set. We illustrated the effectiveness of our proposed method on five real-world HSI datasets—PaviaU, PaviaC, Salinas, SalinasA, and KSC. Due to environmental changes and observation conditions, the spectral features of the same land-cover class may vary at different times or locations. Moreover, hyperspectral images of the same region acquired by different satellites may have some differences due to band configuration, spatial resolution, radiometric calibration, etc. Therefore, we designed the following seven source–target tasks to test the generalization and robustness ability of our proposed method. In each task, we

trained the meta-optimizer on the source dataset and then used it to optimize a predictive model on the target dataset with few labeled data. The real-world HSI classification experimental tasks consisted of different source–target combinations:

- (1) Trained on PaviaU and tested on PaviaC (PaviaU–PaviaC task);
- (2) Trained on PaviaC and tested on PaviaU (PaviaC–PaviaU task);
- (3) Trained on SalinasA and tested on Salinas (SalinasA–Salinas task);
- (4) Trained on PaviaC and tested on SalinasA (PaviaC–SalinasA task);
- (5) Trained on SalinasA and tested on PaviaC (SalinasA–PaviaC task);
- (6) Trained on PaviaC and tested on KSC (PaviaC–KSC task);
- (7) Trained on KSC and tested on PaviaC (KSC–PaviaC task).

The PaviaU dataset and the PaviaC dataset were acquired by the ROSIS sensor, so tasks (1) and (2) can evaluate the generalization ability in different regions under the same sensor. The SalinasA dataset is a small subset of Salinas, so task (3) can test the generalization ability to deal with a large number of unseen classes. Robustness measures the ability of the model to maintain its functionality and performance under abnormal scenarios; so, to demonstrate the robustness of our model, tasks (4)–(7) adopted the strictest abnormal scenarios, where an HSI dataset from one sensor was used to train the model and an HSI dataset from a different sensor was used to test the robustness of the model. Our code is available at <https://github.com/lazyhaotao/MOE-U>.

4.1. Description of Datasets

The PaviaU HSI dataset was captured over Pavia using the ROSIS sensor. Its spatial size is 610×340 , and the number of spectral bands is 103. The PaviaU HSI dataset contains 42,776 labeled pixels in 9 land-cover classes (Table 1). The PaviaC HSI dataset was also acquired by the ROSIS sensor over the same region as PaviaU. Its spatial size is 1096×715 , and the number of spectral bands is 102. PaviaC has 148,152 labeled pixels in 9 land-cover classes (Table 2). The spatial resolution of these two datasets is 1.3 m.

Table 1. Land-cover classes with the number of samples per class in the PaviaU dataset.

No.	Class	Number
C1	Asphalt	6631
C2	Meadows	18,649
C3	Gravel	2099
C4	Trees	3064
C5	Painted metal sheets	1345
C6	Bare soil	5029
C7	Bitumen	1330
C8	Self-blocking bricks	3682
C9	Shadows	947
Total		42,776

Table 2. Land-cover classes with the number of samples per class in the PaviaC dataset.

No.	Class	Number
C1	Water	65,971
C2	Trees	7598
C3	Asphalt	3090
C4	Self-blocking bricks	2685
C5	Bitumen	6584
C6	Tiles	9248
C7	Shadows	7287
C8	Meadows	42,826
C9	Bare soil	2863
Total		148,152

The Salinas HSI dataset was collected by the 224-band AVIRIS sensor over Salinas Valley. Its spatial size is 512×217 , and the number bands is 204. It has 54,129 labeled samples classified into 16 land-cover classes (Table 3). The SalinasA HSI dataset is a small sub-scene of the Salinas dataset. It has a size of 83×86 and 6 land-cover classes (Table 4). The spatial resolution of these two datasets is 3.7 m.

Table 3. Land-cover classes with the number of samples per class in the Salinas dataset.

No.	Class	Number
C1	Brocoli_green_weeds_1	2009
C2	Brocoli_green_weeds_2	3726
C3	Fallow	1976
C4	Fallow_rough_plow	1394
C5	Fallow_smooth	2678
C6	Stubble	3959
C7	Celery	3579
C8	Grapes_untrained	11,271
C9	Soil_vinyard_develop	6203
C10	Corn_senesced_green_weeds	3278
C11	Lettuce_romaine_4wk	1068
C12	Lettuce_romaine_5wk	1927
C13	Lettuce_romaine_6wk	916
C14	Lettuce_romaine_7wk	1070
C15	Vinyard_untrained	7268
C16	Vinyard_vertical_trellis	1807
Total		54,129

Table 4. Land-cover classes with the number of samples per class in the SalinasA dataset.

No.	Class	Number
C1	Brocoli_green_weeds_1	391
C2	Corn_senesced_green_weeds	1343
C3	Lettuce_romaine_4wk	616
C4	Lettuce_romaine_5wk	1525
C5	Lettuce_romaine_6wk	674
C6	Lettuce_romaine_7wk	799
Total		5348

The KSC HSI dataset was collected by the AVIRIS sensor over the Kennedy Space Center (KSC), Florida. Its spatial size is 512×614 , and the number of spectral bands is 176. It has 5211 labeled pixels in 9 land-cover classes (Table 5). The spatial resolution of the KSC dataset is 18 m.

Table 5. Land-cover classes with the number of samples per class in the KSC dataset.

No.	Class	Number
C1	Scrub	761
C2	Willow-swamp	243
C3	CP-hammock	256
C4	Slash-pine	252
C5	Oak-broadleaf	161
C6	Hardwood	229
C7	Swap	105
C8	Graminoid-marsh	431
C9	Spartina-marsh	520
C10	Cattail-marsh	404

Table 5. Cont.

No.	Class	Number
C11	Salt-marsh	419
C12	Mud-flats	503
C13	Water	927
Total		5211

In summary, five HSI datasets (PaviaU, PaviaC, Salinas, SalinasA, and KSC) were used to test the performance of our proposed method. These five datasets from real-world scenarios contain a huge number of land-cover classes and have been widely used as test datasets in research on hyperspectral remote sensing image classification. Table 6 shows a comparison of these five real-world HSI datasets.

Table 6. Comparison of five real-world HSI datasets.

	PaviaU	PaviaC	Salinas	SalinasA	KSC
Pixel resolution	610×340	1096×715	512×217	83×86	512×614
Labeled pixels	42,776	148,152	54,129	5348	5211
Number of bands	103	102	204	204	176
Spectral range (nm)	430–860	430–860	400–2500	400–2500	400–2500
Sensor	ROSIS	ROSIS	AVIRIS	AVIRIS	AVIRIS
Number of classes	9	9	16	6	13
Spatial resolution (m)	1.3	1.3	3.7	3.7	18

4.2. Experimental Setup

The performance of various optimizers was evaluated with the convergence speed and final convergence value of the predictive model's loss function. We also evaluated the target HSI datasets for overall accuracy (OA), average accuracy (AA), and Kappa coefficient (κ), which indirectly reflect the performance of various optimizers and the degree of falling into overfitting. In detail, the OA is the proportion of correctly classified samples among all of the tested samples. The AA is defined as the average of all-class classification accuracy. Compared with OA, AA pays more attention to the classes with fewer samples. The Kappa coefficient is a statistical index evaluating the consistency and classification accuracy and is defined as $\kappa = (p_0 - p_e) / (1 - p_e)$, where p_0 is the OA and p_e is the hypothetical probability of the chance agreement.

The Adam optimizer with learning rate $\alpha = 0.01$ was used to train the meta-optimizer ensemble based on Algorithm 1. During the meta-training phase, we set $B = 128$ and $T = 100$. Then, we obtained an $MOE_3 = \{G_{\phi_i}\}_{i=1}^3$. For the update integration algorithm, all of the experimental results were reported with $\beta = 0.1$. The LSTM optimizer consisted of a two-layer LSTM network, with each layer containing 20 hidden units. It was trained by meta-learning without using ensemble learning. The LSTM optimizer's parameters were saved after 1000 meta-training episodes on the source dataset. For the normalization method, we set $p = 10$ as usual.

The predictive model f trained by the above optimizers on the target dataset was a four-layer fully connected network with a sigmoid activation function and each hidden layer containing 20 nodes. We input each HSI pixel of size $1 \times 1 \times \text{channel number}$ to the predictive model for classification. The loss function used in our experiments was cross-entropy loss. Our real-world HSI classification experiments concentrated on the N-way 10-shot learning setting, i.e., we randomly picked 10 samples from each of N land-cover classes to form the training set. We repeated each experiment ten times and reported the mean and the standard variation of various measurement indicators for classification performance.

4.3. Experimental Results

For brevity, we refer to the meta-optimizer ensemble with update integration as MOE-U, and to averaging all of the candidate updates of a meta-optimizer ensemble as MOE-A. We compared the performance of MOE-U against the LSTM optimizer, MOE-A, and a selection of popular human-designed optimizers: Adam, AdaGrad, RMSprop, SGD, and SGD with momentum, whose learning rates are 0.005, 0.1, 0.001, 0.5, and 0.5, respectively.

4.3.1. PaviaU–PaviaC Task

In this task, we first trained the meta-optimizers, including the LSTM optimizer, MOE-A, and MOE-U, on the PaviaU dataset. The meta-optimizers were then used to train the predictive model f on the PaviaC dataset. To avoid overfitting, we set the number of training iterations as 50. The averaged loss change curves are shown in Figure 2. Meta-optimizers had much faster convergence speed than the human-designed optimizers, indicating that the learned prior knowledge from other datasets acquired from the same sensor was used to accelerate the convergence speed and significantly reduce the final convergence loss. The final averaged convergence values and their standard deviations (Std) are shown in Table 7. Our MOE-U also achieved the lowest final convergence value of 0.3786 and the lowest Std of 0.1128, displaying better performance than the other two meta-optimizers. These numerical results confirm that our proposed update integration algorithm can effectively accelerate the convergence process and achieve lower loss simultaneously. The final convergence value of MOE-A was 0.3819, which was higher than that of MOE-U. The reason for this may be that averaging different parameter updates can bias the best update direction away from the correct direction, and our update integration process can determine the best update direction by estimating the change in loss.

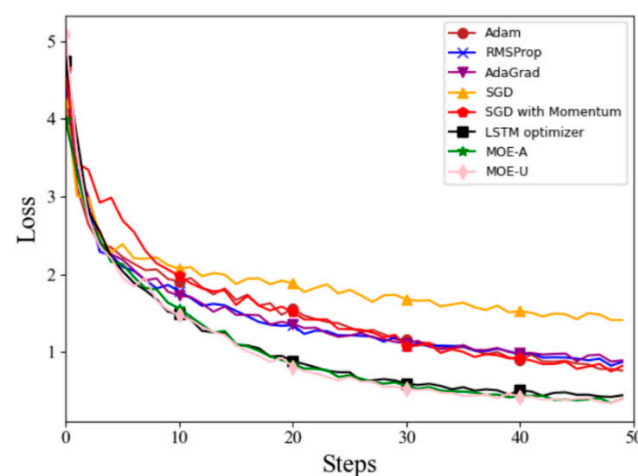


Figure 2. Loss changes on the PaviaC dataset.

Table 7. The loss function values of different optimizers on the PaviaC dataset.

Optimizer	Final Convergence Value	
	Mean	Std
Adam	0.8023	0.2267
AdaGrad	0.9133	0.1145
RMSprop	0.8824	0.1323
SGD	1.445	0.1436
SGD with momentum	0.8019	0.2074
LSTM optimizer	0.4367	0.1149
MOE-A	0.3819	0.1135
MOE-U	0.3786	0.1128

The classification performances by different optimizers are shown in Table 8, indicating that, compared with human-designed optimizers, meta-optimizers can achieve lower loss and better classification results while avoiding overfitting. The predictive model optimized by MOE-U achieved 0.8791, 0.8324, and 0.8340 on OA, AA, and Kappa, respectively, outperforming human-designed optimizers. Compared with the LSTM optimizer, our MOE-U increased the OA, AA, and Kappa values by 0.0277, 0.0294 and 0.0357, respectively. Compared with our MOE-A, our MOE-U increased these values by 0.0151, 0.0245, and 0.0249, respectively.

Table 8. Classification accuracy by different optimizers on the PaviaC dataset.

Class	Adam		AdaGrad		RMSprop		SGD		SGD with Momentum		LSTM Optimizer		MOE-A		MOE-U	
	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std
C1	0.9763	0.0368	0.8894	0.2945	0.9157	0.2223	0.6161	0.4633	0.9915	0.0074	0.9928	0.0025	0.9876	0.0102	0.9845	0.0119
C2	0.7692	0.2801	0.6898	0.3211	0.4484	0.3307	0.6452	0.4108	0.3957	0.4259	0.8055	0.0733	0.7905	0.0991	0.8146	0.0716
C3	0.6118	0.3800	0.6786	0.3367	0.8507	0.1639	0.3934	0.4704	0.6914	0.4464	0.8408	0.1377	0.8590	0.0879	0.8304	0.0940
C4	0.6409	0.3262	0.7002	0.2534	0.7315	0.1876	0.4538	0.4362	0.6988	0.2473	0.8142	0.1048	0.7548	0.2162	0.7986	0.0792
C5	0.2026	0.2708	0.2620	0.3278	0.3151	0.3434	0.3528	0.3785	0.4822	0.2206	0.4025	0.1930	0.5311	0.206	0.6089	0.1466
C6	0.8138	0.1262	0.3946	0.3704	0.7293	0.3589	0.4697	0.4290	0.6627	0.3159	0.8636	0.1185	0.8968	0.0783	0.8755	0.0885
C7	0.5769	0.3219	0.8887	0.0589	0.5799	0.3247	0.5457	0.4266	0.8368	0.0976	0.7998	0.0418	0.7365	0.1492	0.8057	0.0723
C8	0.7490	0.3631	0.6369	0.3284	0.6097	0.3932	0.3285	0.3831	0.4691	0.3907	0.7103	0.2944	0.7387	0.2296	0.7843	0.1785
C9	0.9942	0.0094	0.9896	0.0180	0.9943	0.0103	0.6075	0.4427	0.8024	0.3119	0.9979	0.0018	0.9925	0.0139	0.9892	0.015
OA	0.8225	0.098	0.7415	0.1353	0.7452	0.1717	0.5024	0.2719	0.7440	0.1239	0.8514	0.0888	0.8604	0.0686	0.8791	0.0503
AA	0.7039	0.0774	0.6811	0.0648	0.6860	0.0569	0.4903	0.0818	0.6701	0.0876	0.8030	0.0443	0.8097	0.0399	0.8324	0.017
Kappa	0.7588	0.1219	0.6649	0.1504	0.6737	0.1798	0.4250	0.2518	0.6612	0.1538	0.7983	0.1119	0.8091	0.0873	0.8340	0.0649

In total, the results demonstrate that MOE-U has good generalization ability. It can optimize a predictive model with limited training samples by using the learned knowledge from other HSI datasets from the same sensor. The best classification maps of various methods are displayed in Figure 3. The number after the optimizer name represents the accuracy on the test set. Our MOE-U also achieved the highest classification accuracy of 0.9408.

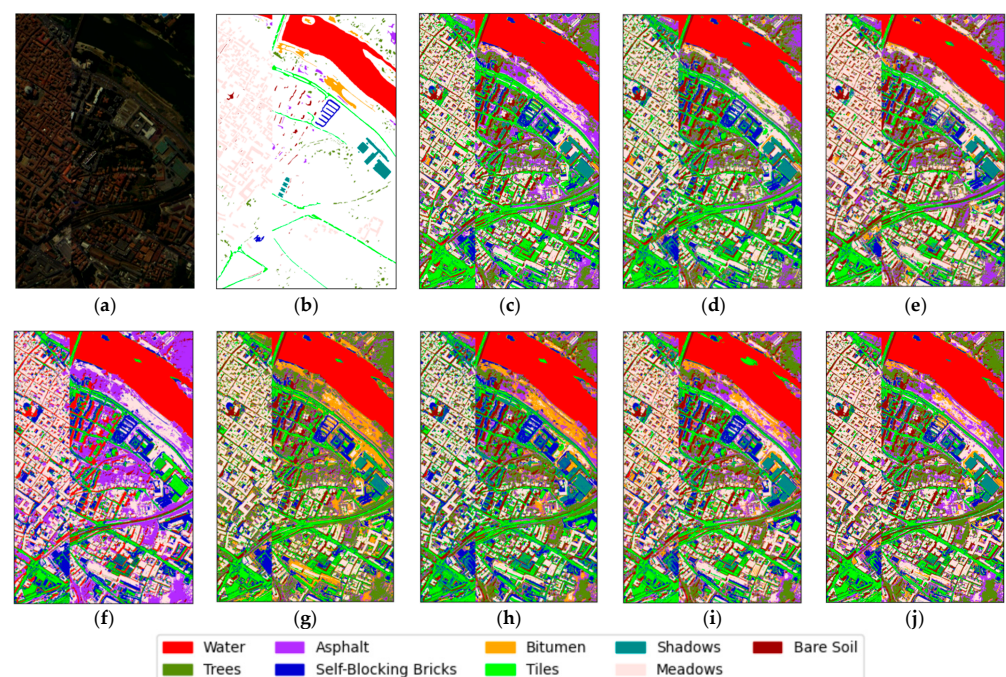


Figure 3. Visual classification results on the PaviaC dataset: (a) False-color image. (b) Ground-truth map. Classification maps obtained by (c) Adam (0.9216), (d) AdaGrad (0.9069), (e) RMSprop (0.8965),

(f) SGD (0.8253), (g) SGD with momentum (0.9291), (h) LSTM optimizer (0.9342), (i) MOE-A (0.9222), and (j) MOE-U (0.9408).

4.3.2. PaviaC–PaviaU Task

Following the above experiment, we then trained meta-optimizers on PaviaC and tested their performance on PaviaU. The averaged loss change curves are shown in Figure 4, and the final averaged convergence values and their standard deviations (Std) are given in Table 9. In this scenario, similar experimental results were achieved. MOE-U achieved the fastest convergence speed and the lowest convergence value of 0.6665.

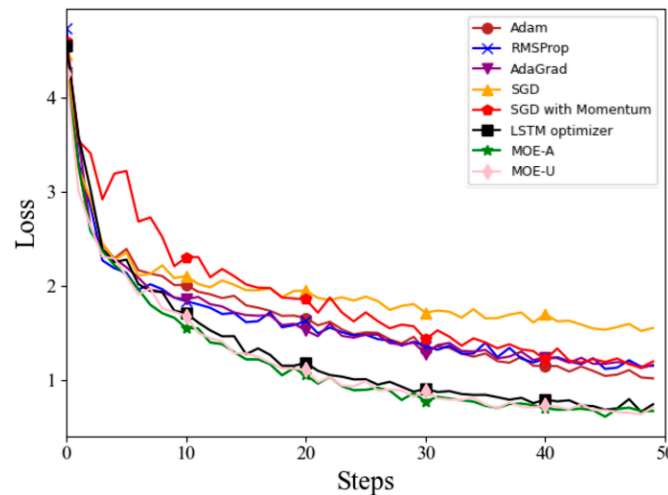


Figure 4. Loss changes on the PaviaU dataset.

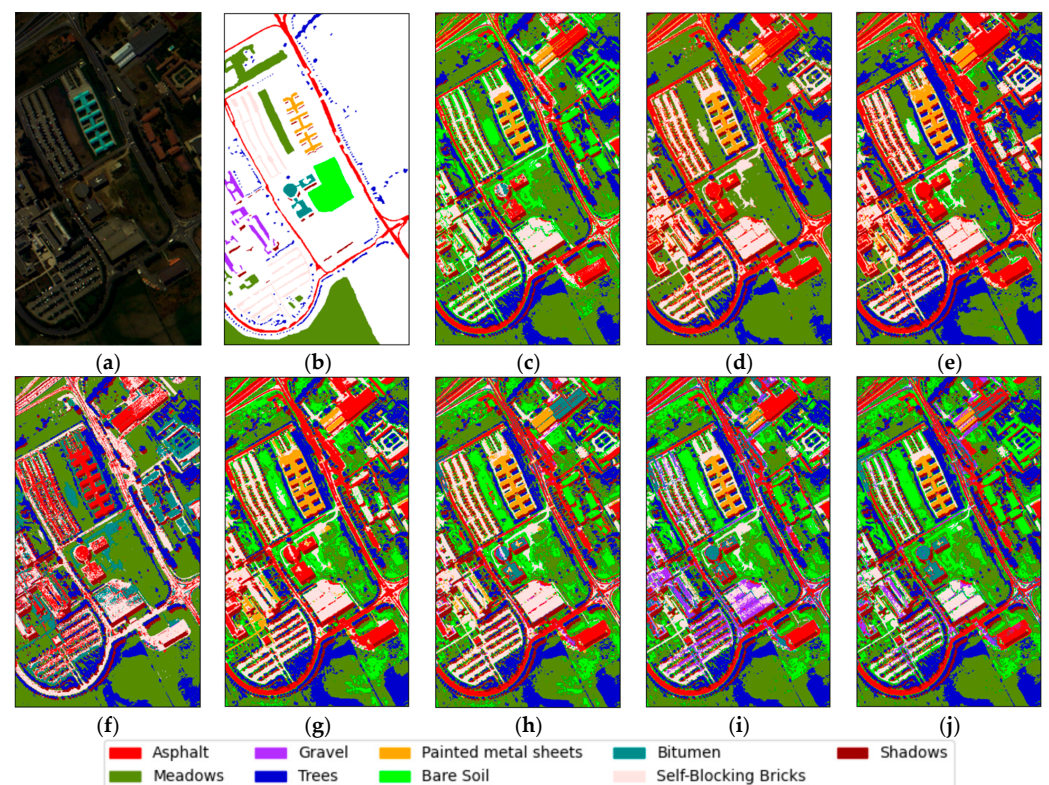
Table 9. The convergence values of different optimizers on the PaviaU dataset.

Optimizer	Final Convergence Value	
	Mean	Std
Adam	1.048	0.1693
AdaGrad	1.164	0.1103
RMSprop	1.148	0.1414
SGD	1.551	0.1254
SGD with momentum	1.181	0.3798
LSTM optimizer	0.7153	0.1213
MOE-A	0.6686	0.1155
MOE-U	0.6665	0.1627

The performance comparison of the classification results is shown in Table 10. The OA and Kappa values of our MOE-U were 0.6151 and 0.5195, respectively, outperforming all human-designed optimizers and meta-optimizers. The AA value of our MOE-U was just 0.0033 lower than that of our MOE-A. The distribution of samples across land-cover classes of PaviaU was highly unbalanced. For instance, meadows (C2) had 18,649 labeled samples while shadows (C9) had 947 labeled samples. This imbalance made training and testing very difficult, e.g., the Kappa of SGD was just 0.1991. Even in the face of such land-cover class imbalance, the meta-optimizers performed better than the human-designed optimizers, demonstrating that the learned meta-knowledge can effectively help optimize the network. The best classification maps by various optimizers are displayed in Figure 5.

Table 10. Classification accuracy by different optimizers on the PaviaU dataset.

Class	Adam		AdaGrad		RMSprop		SGD		SGD with Momentum		LSTM Optimizer		MOE-A		MOE-U	
	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std
C1	0.4130	0.2868	0.4614	0.3405	0.4577	0.3146	0.1844	0.3962	0.5825	0.3758	0.5907	0.1557	0.5571	0.1741	0.5626	0.1163
C2	0.4700	0.2504	0.5203	0.2812	0.5821	0.1823	0.2108	0.3238	0.5510	0.2571	0.5196	0.1929	0.5308	0.1537	0.5992	0.0603
C3	0.1538	0.3234	0.4175	0.4279	0.1015	0.232	0.3730	0.4199	0.2003	0.2535	0.4330	0.2587	0.4353	0.1865	0.2719	0.2183
C4	0.8614	0.2835	0.9641	0.0292	0.9664	0.0296	0.8609	0.2891	0.8836	0.1858	0.9354	0.0337	0.9279	0.0456	0.912	0.0553
C5	0.8933	0.2975	0.9920	0.0068	0.9886	0.0109	0.7851	0.3932	0.8929	0.2977	0.9903	0.0037	0.9910	0.0058	0.9928	0.0032
C6	0.5114	0.2727	0.3124	0.2140	0.2715	0.1919	0.1121	0.2019	0.3844	0.2295	0.4628	0.1810	0.4782	0.2236	0.4401	0.0901
C7	0.3456	0.3430	0.1363	0.3039	0.5056	0.4819	0.0994	0.2967	0.4331	0.4078	0.7783	0.2127	0.7705	0.1485	0.7816	0.2667
C8	0.4491	0.3440	0.3112	0.4121	0.3632	0.3790	0.2655	0.3466	0.3427	0.4246	0.4645	0.3680	0.5796	0.2303	0.6814	0.2203
C9	0.9966	0.0033	0.9982	0.0016	0.9987	0.0013	0.9992	0.0009	0.9959	0.0046	0.9997	0.0005	0.9976	0.0034	0.9967	0.0037
OA	0.4978	0.1072	0.5089	0.1404	0.5311	0.0928	0.2864	0.1374	0.5419	0.0708	0.5782	0.0707	0.5889	0.0514	0.6151	0.0322
AA	0.5660	0.0949	0.5682	0.0461	0.5817	0.0416	0.4323	0.082	0.5851	0.0775	0.6860	0.0233	0.6964	0.0291	0.6931	0.0439
Kappa	0.3935	0.1066	0.4057	0.1279	0.4211	0.0928	0.1991	0.1007	0.4340	0.0694	0.4851	0.0670	0.4959	0.0500	0.5195	0.0379

**Figure 5.** Visual classification results on the PaviaU dataset: (a) False-color image. (b) Ground-truth map. Classification maps obtained by (c) Adam (0.6918), (d) AdaGrad (0.7050), (e) RMSprop (0.6417), (f) SGD (0.5290), (g) SGD with momentum (0.6475), (h) LSTM optimizer (0.6444), (i) MOE-A (0.6515), and (j) MOE-U (0.6583).

4.3.3. SalinasA–Salinas Task

In this task, we trained the meta-optimizers on SalinasA and tested their classification performance on Salinas. Since the SalinasA dataset contains only 6 categories, while the Salinas dataset contains 16 categories, this task can test the generalization ability of our method in dealing with many unseen land-cover classes. The averaged loss change curves are shown in Figure 6, and the final averaged convergence values and their Stds are given

in Table 11. In this experiment, MOE-U converged significantly faster and had lower losses than other optimizers.

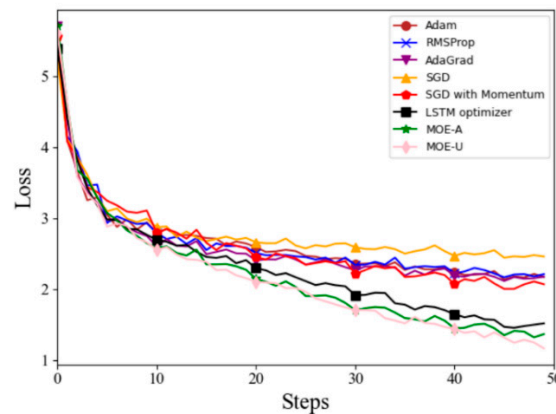


Figure 6. Loss changes on the Salinas dataset.

Table 11. The convergence values of different optimizers on the Salinas dataset.

Optimizer	Final Convergence Value	
	Mean	Std
ADAM	2.176	0.3160
AdaGrad	2.155	0.1448
RMSprop	2.198	0.1502
SGD	2.463	0.1173
SGD with momentum	2.053	0.2727
LSTM optimizer	1.486	0.1639
MOE-A	1.373	0.2306
MOE-U	1.255	0.1442

The performance comparison of the classification results is shown in Table 12, indicating that the Salinas dataset was so hard to classify with few training samples that all of the human-designed optimizers failed in this experiment. This might be attributed to the considerably large number of classes in the Salinas dataset, and the samples between each class were so similar that they were difficult to distinguish. Although the human-designed optimizers reduced the training loss, they made the predictive model severely overfit, resulting in the OA, AA, and Kappa all being near 0.2—a terrible value. However, the OA, AA, and Kappa values of our MOE-U reached 0.5333, 0.5516, and 0.4845, respectively—the best among all of the optimizers. Compared with the single meta-optimizer (LSTM optimizer), our MOE-U increased the OA, AA, and Kappa values by 0.0467, 0.0368, and 0.048, respectively. Compared with our MOE-A, our MOE-U increased these values by 0.0274, 0.0053, and 0.0233, respectively. These results indicate that our improvement of meta-optimizers can effectively enhance the generalization ability in dealing with many unseen classes. The best classification maps are displayed in Figure 7.

Table 12. Classification accuracy by different optimizers on the Salinas dataset.

Class	Adam		AdaGrad		RMSprop		SGD		SGD with Momentum		LSTM Optimizer		MOE-A		MOE-U	
	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std
C1	0.3998	0.4896	0.7984	0.3992	0.7858	0.3942	0.4517	0.4647	0.799	0.3995	0.4478	0.4552	0.7725	0.3889	0.5675	0.4257
C2	0.2960	0.4515	0.3903	0.3859	0.2929	0.4237	0.3082	0.4516	0.0003	0.0008	0.3179	0.4285	0.3812	0.3146	0.5832	0.3635
C3	0.1000	0.300	0.0733	0.1718	0.1558	0.3188	0.2819	0.4320	0.2423	0.3990	0.2052	0.2581	0.4340	0.3508	0.1757	0.2333
C4	0.1024	0.2993	0.5722	0.4658	0.5042	0.4760	0.2453	0.3954	0.2101	0.3954	0.9667	0.0683	0.8356	0.2852	0.8961	0.2988

Table 12. Cont.

Class	Adam		AdaGrad		RMSprop		SGD		SGD with Momentum		LSTM Optimizer		MOE-A		MOE-U	
	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std
C5	0.2000	0.3999	0.0002	0.0006	0.1127	0.2975	0.0001	0.0001	0.1047	0.2943	0.5880	0.3833	0.3880	0.4655	0.5782	0.3781
C6	0.4679	0.4644	0.7653	0.3895	0.6639	0.4033	0.1272	0.2948	0.6957	0.4144	0.8979	0.1351	0.9567	0.0760	0.9552	0.0667
C7	0.2930	0.4334	0.0159	0.0322	0.2571	0.3946	0.1062	0.2972	0.3078	0.4498	0.6105	0.3853	0.7449	0.2658	0.4794	0.4007
C8	0.1994	0.2907	0.3709	0.3832	0.2009	0.3266	0.1182	0.2602	0.1373	0.2115	0.4620	0.3303	0.3487	0.3202	0.6070	0.2226
C9	0.1997	0.3995	0.3567	0.4430	0.1130	0.2923	0.3007	0.4568	0.209	0.3963	0.4713	0.4746	0.4205	0.4054	0.5791	0.4731
C10	0.0784	0.2118	0.0800	0.2124	0.0696	0.1875	0.0025	0.0076	0.0894	0.2032	0.1951	0.2343	0.3351	0.2793	0.2083	0.2136
C11	0.0000	0.0000	0.2007	0.3997	0.2981	0.4554	0.038	0.1122	0.0000	0.0000	0.4175	0.4287	0.3186	0.379	0.4751	0.4109
C12	0.1996	0.3992	0.1620	0.3347	0.2648	0.4075	0.1000	0.3000	0.1778	0.359	0.1707	0.3027	0.3013	0.2653	0.2410	0.3297
C13	0.2746	0.383	0.3350	0.3999	0.1166	0.2121	0.0491	0.1474	0.3028	0.3495	0.8168	0.2841	0.8245	0.2843	0.8762	0.1684
C14	0.1574	0.2491	0.5389	0.4228	0.1794	0.3122	0.098	0.1477	0.3713	0.3786	0.6726	0.2499	0.5209	0.3732	0.6700	0.2840
C15	0.2089	0.2808	0.0598	0.1694	0.2384	0.3318	0.2729	0.4001	0.1034	0.2181	0.4313	0.3008	0.5338	0.3236	0.2958	0.1900
C16	0.2174	0.3049	0.1571	0.3069	0.1349	0.1436	0.0174	0.0245	0.1951	0.2512	0.5661	0.2205	0.6242	0.1530	0.6379	0.1198
OA	0.2243	0.1135	0.2932	0.0953	0.2555	0.0811	0.1744	0.0674	0.2175	0.0839	0.4863	0.0699	0.5056	0.0968	0.5333	0.0525
AA	0.2122	0.0786	0.3048	0.0675	0.2743	0.0508	0.1573	0.0429	0.2466	0.0912	0.5148	0.0587	0.5463	0.0895	0.5516	0.0469
Kappa	0.1678	0.1024	0.2403	0.0857	0.2010	0.0744	0.1118	0.0577	0.1679	0.0810	0.4365	0.0721	0.4612	0.0997	0.4845	0.0536

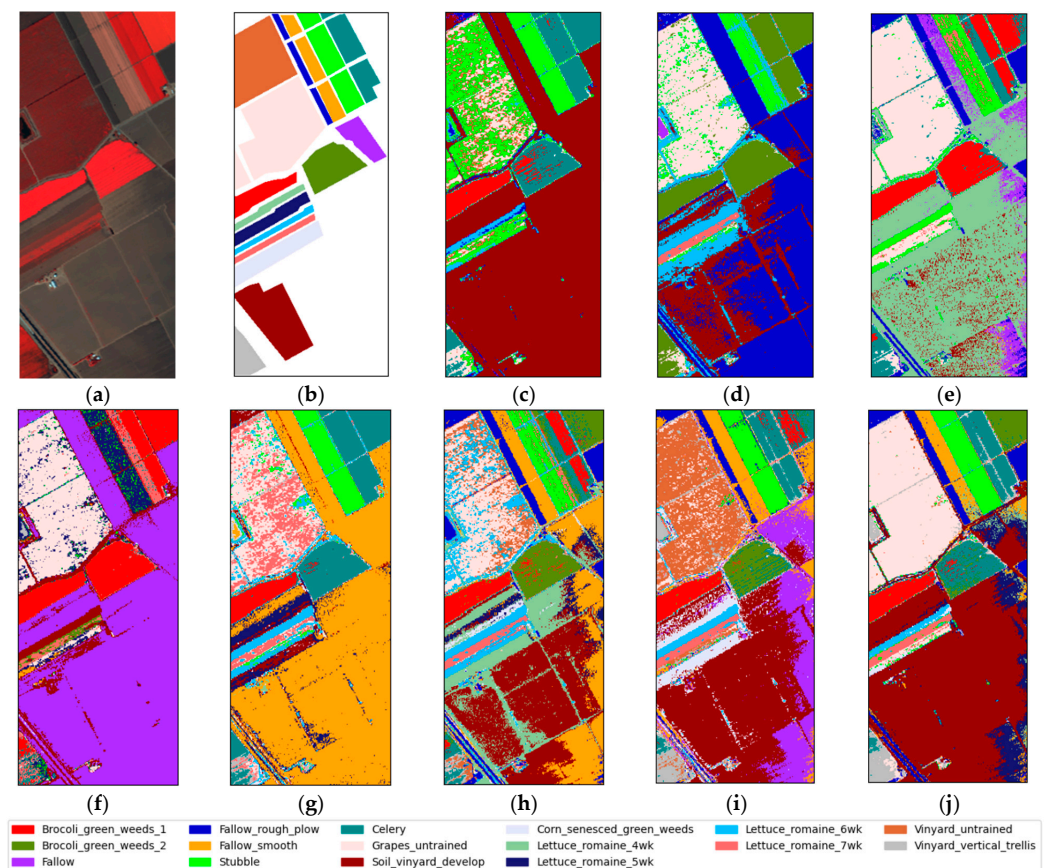


Figure 7. Visual classification results on the Salinas dataset: (a) False-color image. (b) Ground-truth map. Classification maps obtained by (c) Adam (0.3873), (d) AdaGrad (0.4571), (e) RMSprop (0.3873), (f) SGD (0.2727), (g) SGD with momentum (0.3885), (h) LSTM optimizer (0.5837), (i) MOE-A (0.6765), and (j) MOE-U (0.6236).

4.3.4. PaviaC–SalinasA Task

The PaviaC dataset and the SalinasA dataset were from different sensors, so they were completely different in terms of the number of bands, spectral range, spatial resolution, and land-cover types. Figure 8 shows the averaged loss change curves, and Table 13 shows the final convergence values. All of the meta-optimizers also performed better in convergence speed and the final convergence value. The averaged final convergence value of MOE-U was 0.2822, which was lower than all other meta-optimizers.

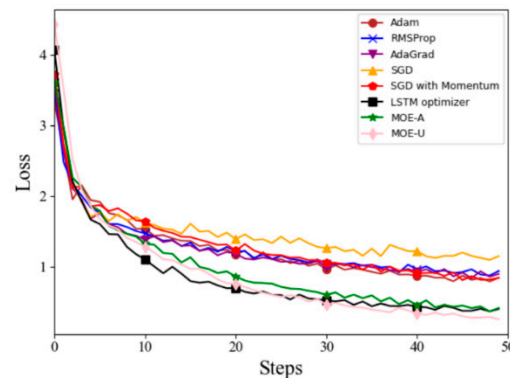


Figure 8. Loss changes on the SalinasA dataset.

Table 13. The convergence values of different optimizers on the SalinasA dataset.

Optimizer	Final Convergence Value	
	Mean	Std
Adam	0.8340	0.2434
AdaGrad	0.8905	0.1416
RMSprop	0.9189	0.1337
SGD	1.145	0.1298
SGD with momentum	0.8315	0.1449
LSTM optimizer	0.3872	0.1678
MOE-A	0.4187	0.1039
MOE-U	0.2822	0.1047

The classification results are shown in Table 14. All of the meta-optimizers had better classification results than the human-designed optimizers, and MOE-U achieved the best results in OA, AA, and Kappa again. This demonstrates that the learned knowledge can easily be generalized to HSI datasets from different sensors. Since the training dataset and test dataset in this task were from different sensors, these results effectively demonstrate the robustness of our methods. Finally, the best classification maps are displayed in Figure 9.

Table 14. Classification accuracy of the predictive model optimized by different optimizers on the SalinasA dataset.

Class	Adam		AdaGrad		RMSprop		SGD		SGD with Momentum		LSTM Optimizer		MOE-A		MOE-U	
	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std
C1	0.9951	0.0008	0.9951	0.0008	0.9959	0.0013	0.9954	0.0010	0.9959	0.0017	0.9951	0.0008	0.9949	0.0000	0.9949	0.0000
C2	0.085	0.1102	0.2235	0.1997	0.1377	0.1203	0.2137	0.2628	0.2051	0.1980	0.4778	0.2305	0.4640	0.1864	0.6827	0.1349
C3	0.1758	0.3139	0.6216	0.4067	0.6927	0.3677	0.5305	0.4759	0.7713	0.3098	0.8461	0.1602	0.8737	0.1004	0.6930	0.1686
C4	0.7527	0.3973	0.5201	0.4235	0.5530	0.3958	0.3605	0.4408	0.8736	0.1868	0.7308	0.3518	0.8089	0.2486	0.8912	0.1300
C5	0.7936	0.3969	0.8522	0.2945	0.6625	0.4158	0.4543	0.4529	0.8742	0.2588	0.9950	0.0035	0.9964	0.0022	0.9947	0.0033
C6	0.9099	0.1317	0.9627	0.0316	0.9584	0.0535	0.9369	0.0828	0.9446	0.0768	0.9637	0.0239	0.9594	0.0285	0.9625	0.0309

Table 14. Cont.

Class	Adam		AdaGrad		RMSprop		SGD		SGD with Momentum		LSTM Optimizer		MOE-A		MOE-U	
	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std
OA	0.5649	0.1504	0.600	0.1066	0.5716	0.1252	0.4876	0.0843	0.7136	0.1014	0.7680	0.1355	0.7895	0.0989	0.8473	0.067
AA	0.6187	0.1301	0.6959	0.0772	0.6667	0.0944	0.5819	0.0609	0.7774	0.0981	0.8348	0.0899	0.8496	0.0582	0.8698	0.0558
Kappa	0.4662	0.1689	0.5170	0.1206	0.4882	0.1334	0.3869	0.0799	0.6493	0.1266	0.7197	0.1579	0.7434	0.1156	0.8104	0.0830

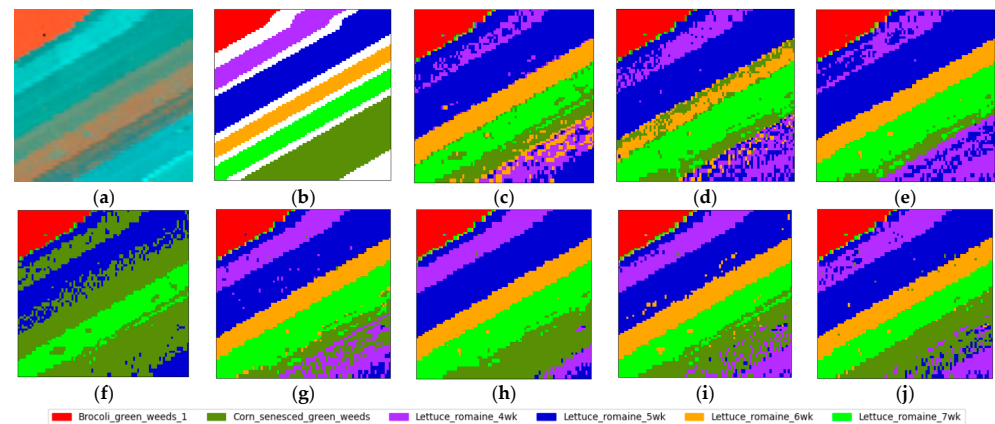


Figure 9. Visual classification results on the SalinasA dataset: (a) False-color image. (b) Ground-truth map. Classification maps obtained by (c) Adam (0.7556), (d) AdaGrad (0.7467), (e) RMSprop (0.7609), (f) SGD (0.6605), (g) SGD with momentum (0.8708), (h) LSTM optimizer (0.9476), (i) MOE-A (0.9139), and (j) MOE-U (0.9277).

4.3.5. SalinasA–PaviaC Task

In this task, we trained meta-optimizers on SalinasA and tested their classification performance on PaviaC. This task can further test the robustness of our method in the scenario where the source dataset and the target dataset are from different sensors. Figure 10 shows the averaged loss change curves, and the final convergence values are shown in Table 15. Based on the experimental results, we drew a similar conclusion that all of the meta-optimizers performed better than the human-designed optimizers in terms of convergence speed and the final loss, and our MOE-U also achieved the lowest loss among the meta-optimizers. The classification performance results are shown in Table 16, demonstrating again that the meta-optimizers can generalize well to other HSI datasets from different sensors. Our proposed MOE-U achieved 0.8445, 0.8112, and 0.7890 in OA, AA, and Kappa, respectively, which also outperformed all other optimizers. The best classification maps on PaviaC are displayed in Figure 11, while the best classification maps by the human-designed optimizers are shown in Figure 3.

Table 15. The convergence values of different optimizers on the PaviaC dataset.

Optimizer	Final Convergence Value	
	Mean	Std
Adam	0.8023	0.2267
AdaGrad	0.9133	0.1145
RMSprop	0.8824	0.1323
SGD	1.445	0.1436
SGD with momentum	0.8019	0.2074
LSTM optimizer	0.4803	0.1362
MOE-A	0.4924	0.1215
MOE-U	0.4246	0.1088

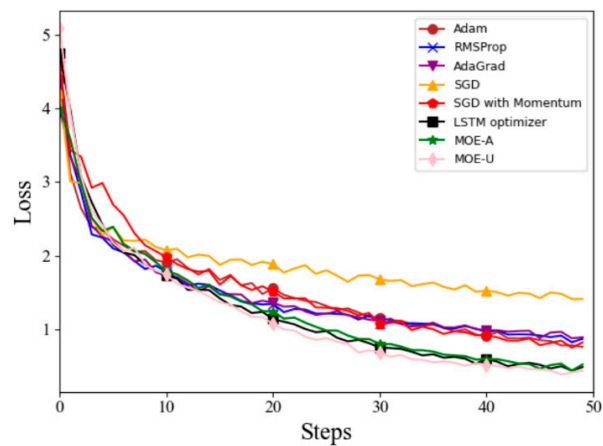


Figure 10. Loss changes on the PaviaC dataset.

Table 16. Classification accuracy by different optimizers on the PaviaC dataset.

Class	Adam		AdaGrad		RMSprop		SGD		SGD with Momentum		LSTM Optimizer		MOE-A		MOE-U	
	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std
C1	0.9763	0.0368	0.8894	0.2945	0.9157	0.2223	0.6161	0.4633	0.9915	0.0074	0.9902	0.0055	0.9839	0.0097	0.9802	0.0173
C2	0.7692	0.2801	0.6898	0.3211	0.4484	0.3307	0.6452	0.4108	0.3957	0.4259	0.7749	0.0816	0.7791	0.1418	0.7862	0.0875
C3	0.6118	0.3800	0.6786	0.3367	0.8507	0.1639	0.3934	0.4704	0.6914	0.4464	0.8847	0.0920	0.7278	0.2782	0.8709	0.0699
C4	0.6409	0.3262	0.7002	0.2534	0.7315	0.1876	0.4538	0.4362	0.6988	0.2473	0.7586	0.2077	0.7091	0.2624	0.7670	0.1321
C5	0.2026	0.2708	0.2620	0.3278	0.3151	0.3434	0.3528	0.3785	0.4822	0.2206	0.5737	0.209	0.5641	0.2197	0.5539	0.1709
C6	0.8138	0.1262	0.3946	0.3704	0.7293	0.3589	0.4697	0.429	0.6627	0.3159	0.8275	0.2294	0.8179	0.1327	0.8815	0.1079
C7	0.5769	0.3219	0.8887	0.0589	0.5799	0.3247	0.5457	0.4266	0.8368	0.0976	0.7812	0.0649	0.7554	0.1324	0.7826	0.0747
C8	0.7490	0.3631	0.6369	0.3284	0.6097	0.3932	0.3285	0.3831	0.4691	0.3907	0.6506	0.2739	0.6435	0.306	0.6863	0.2401
C9	0.9942	0.0094	0.9896	0.0180	0.9943	0.0103	0.6075	0.4427	0.8024	0.3119	0.9959	0.0073	0.9886	0.017	0.9919	0.0131
OA	0.8225	0.098	0.7415	0.1353	0.7452	0.1717	0.5024	0.2719	0.7440	0.1239	0.8358	0.0859	0.8245	0.0927	0.8445	0.0679
AA	0.7039	0.0774	0.6811	0.0648	0.6860	0.0569	0.4903	0.0818	0.6701	0.0876	0.8042	0.0584	0.7744	0.0593	0.8112	0.0286
Kappa	0.7588	0.1219	0.6649	0.1504	0.6737	0.1798	0.425	0.2518	0.6612	0.1538	0.7781	0.1101	0.7638	0.1163	0.7890	0.0861

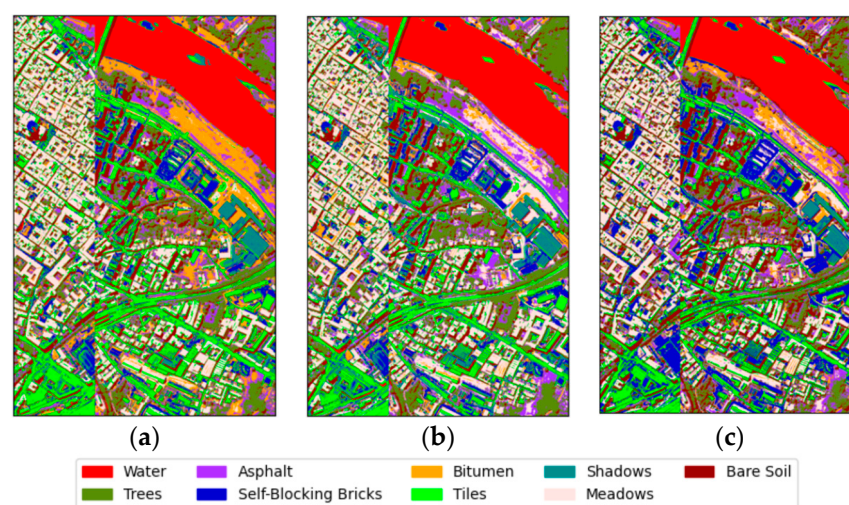


Figure 11. Visual classification results on the PaviaC dataset: (a) LSTM optimizer (0.9407), (b) MOE-A (0.9060), and (c) MOE-U (0.9194).

4.3.6. PaviaC–KSC Task

The PaviaC dataset and the KSC dataset were also from different sensors. Due to significant differences in spatial resolution (1.3 m and 18 m, respectively), the meta-knowledge from one HSI dataset is very difficult to use in classifying another HSI dataset. So, this task can further test the robustness ability of our proposed method. The averaged loss change curves are shown in Figure 12, and the final averaged convergence values and their Stds are reported in Table 17. In this scenario, our MOE-U achieved the fastest convergence speed and the lowest convergence value of 1.5106. The losses of the human-designed optimizers were all significantly higher than those of the meta-optimizers.

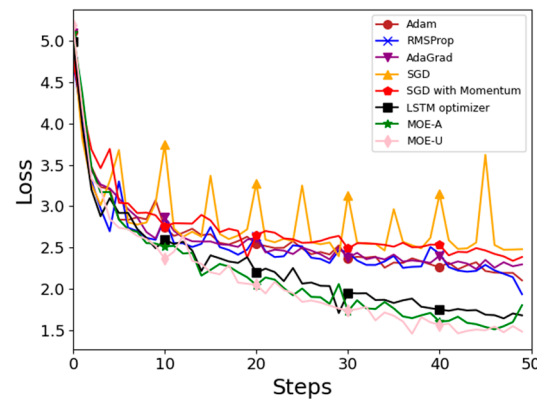


Figure 12. Loss changes on the KSC dataset.

Table 17. The convergence values of different optimizers on the KSC dataset.

Optimizer	Final Convergence Value	
	Mean	Std
Adam	2.1805	0.2329
AdaGrad	2.2995	0.1549
RMSprop	2.1549	0.2984
SGD	2.7160	0.6359
SGD with momentum	2.4010	0.2103
LSTM optimizer	1.6793	0.3079
MOE-A	1.6002	0.3768
MOE-U	1.5106	0.2350

The classification results are shown in Table 18. All of the meta-optimizers had better results than the human-designed optimizers, and MOE-U also achieved the best results in OA, AA, and Kappa again, which were 0.5361, 0.4278, and 0.4860, respectively. Among all of the human-designed optimizers, the highest values were 0.2757, 0.2131, and 0.2003, respectively, indicating that human-designed optimizers caused the network to fall into serious overfitting when the training samples were insufficient, thus failing to make the correct classification. The best classification maps are displayed in Figure 13.

Table 18. Classification accuracy by different optimizers on the KSC dataset.

Class	Adam		AdaGrad		RMSprop		SGD		SGD with Momentum		LSTM Optimizer		MOE-A		MOE-U	
	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std
C1	0.2771	0.4155	0.1000	0.3000	0.1760	0.3559	0.0000	0.0000	0.0995	0.2984	0.5811	0.3668	0.4622	0.3566	0.5882	0.3479
C2	0.1975	0.3951	0.2033	0.3985	0.1449	0.3143	0.1922	0.3848	0.0000	0.0000	0.2428	0.3188	0.3082	0.2597	0.4576	0.2739
C3	0.0984	0.2953	0.2984	0.4559	0.0000	0.0000	0.1008	0.2997	0.0992	0.2977	0.0910	0.2488	0.0184	0.0417	0.2148	0.3460
C4	0.0135	0.0405	0.1722	0.3485	0.0984	0.2939	0.0000	0.0000	0.3889	0.4764	0.0032	0.0095	0.1302	0.2591	0.1087	0.2182
C5	0.1665	0.3395	0.0000	0.0000	0.0770	0.2311	0.2000	0.4000	0.0000	0.0000	0.0193	0.0557	0.6211	0.3739	0.1627	0.2360

Table 18. Cont.

Class	Adam		AdaGrad		RMSprop		SGD		SGD with Momentum		LSTM Optimizer		MOE-A		MOE-U	
	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std
C6	0.0825	0.2476	0.1000	0.3000	0.3803	0.4668	0.0983	0.2948	0.2000	0.4000	0.2450	0.3498	0.0694	0.1136	0.1843	0.2201
C7	0.0000	0.0000	0.0000	0.0000	0.1981	0.3962	0.1000	0.3000	0.0000	0.0000	0.3086	0.3883	0.1914	0.2573	0.4114	0.4295
C8	0.2044	0.3052	0.1090	0.2673	0.0007	0.0021	0.0007	0.0021	0.0501	0.1503	0.1387	0.1583	0.2323	0.1860	0.2056	0.2708
C9	0.1238	0.2993	0.1096	0.2982	0.4415	0.4497	0.1000	0.3000	0.0994	0.2983	0.5406	0.3132	0.5533	0.2615	0.4479	0.3298
C10	0.0106	0.0319	0.2599	0.3938	0.0983	0.2883	0.0000	0.0000	0.0027	0.0057	0.2938	0.3036	0.3374	0.2648	0.3443	0.2510
C11	0.2477	0.3895	0.0482	0.1438	0.0427	0.1282	0.1000	0.3000	0.1000	0.3000	0.4687	0.4338	0.6418	0.3389	0.7888	0.2629
C12	0.5750	0.3728	0.1861	0.2737	0.5153	0.3863	0.1026	0.2985	0.3972	0.4306	0.4676	0.2653	0.6463	0.2487	0.6531	0.2041
C13	0.5975	0.4879	0.7931	0.3966	0.5977	0.4881	0.5995	0.4895	0.5949	0.4845	0.9965	0.0039	0.9954	0.0062	0.9937	0.0075
OA	0.2757	0.0990	0.2545	0.0850	0.2715	0.1203	0.1610	0.0842	0.2135	0.1032	0.4667	0.0716	0.5068	0.0705	0.5361	0.0883
AA	0.1996	0.0364	0.1831	0.0413	0.2131	0.0781	0.1226	0.0384	0.1563	0.0449	0.3382	0.051	0.4006	0.0717	0.4278	0.0592
Kappa	0.1974	0.0912	0.1803	0.0769	0.2003	0.1222	0.0851	0.0723	0.1402	0.1002	0.4049	0.0733	0.4555	0.0763	0.4860	0.094

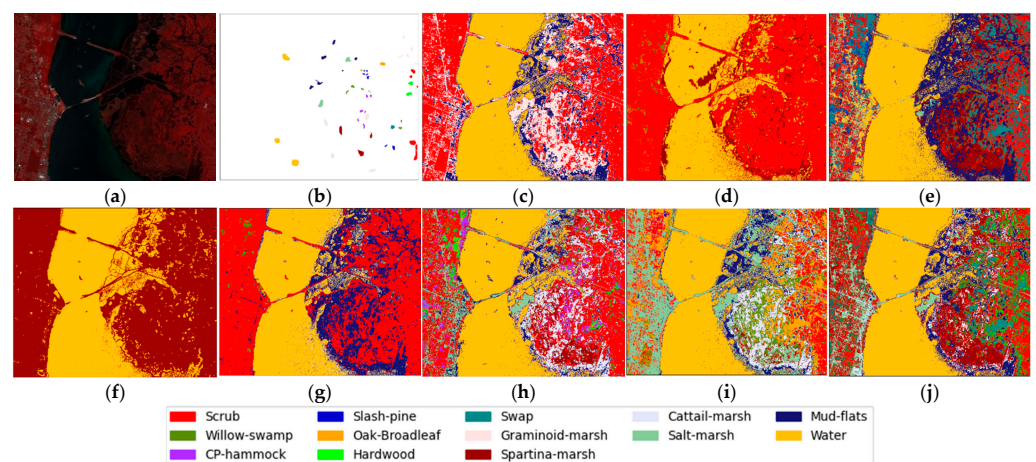


Figure 13. Visual classification results on the KSC dataset: (a) False-color image. (b) Ground-truth map. Classification maps obtained by (c) Adam (0.4481), (d) AdaGrad (0.3373), (e) RMSprop (0.5296), (f) SGD (0.2810), (g) SGD with momentum (0.4123), (h) LSTM optimizer (0.6173), (i) MOE-A (0.5878), and (j) MOE-U (0.6319).

4.3.7. KSC–PaviaC Task

For the KSC–PaviaC task, the averaged loss change curves are shown in Figure 14, and the final averaged convergence values and their Stds are reported in Table 19. In this scenario, all of the meta-optimizers had similar convergence speeds and final convergence values, which were also significantly better than those of the human-designed optimizers. The classification results are shown in Table 20. Our MOE-U achieved the highest OA, AA, and Kappa of 0.9024, 0.8388, and 0.8652, respectively. The best classification maps by the meta-optimizers are displayed in Figure 15. The best classification maps by the human-designed optimizers are shown in Figure 3.

Table 19. The convergence values of different optimizers on the PaviaC dataset.

Optimizer	Final Convergence Value	
	Mean	Std
Adam	0.7734	0.1765
AdaGrad	0.9039	0.0941
RMSprop	0.9133	0.1145
SGD	1.4272	0.1492

Table 19. Cont.

Optimizer	Final Convergence Value	
	Mean	Std
SGD with momentum	0.7634	0.2265
LSTM optimizer	0.3463	0.1168
MOE-A	0.3941	0.1040
MOE-U	0.3710	0.1002

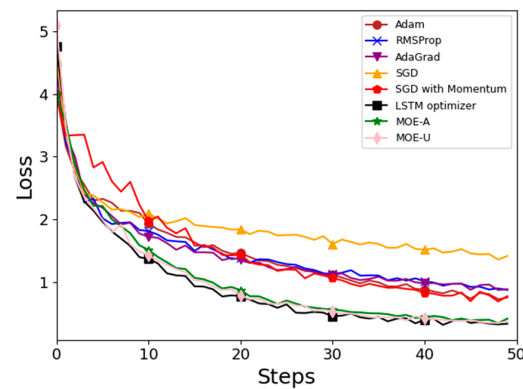


Figure 14. Loss changes on the PaviaC dataset.

Table 20. Classification accuracy by different optimizers on the PaviaC dataset.

Class	Adam		AdaGrad		RMSprop		SGD		SGD with Momentum		LSTM Optimizer		MOE-A		MOE-U	
	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std
C1	0.9763	0.0368	0.8894	0.2945	0.9157	0.2223	0.6161	0.4633	0.9915	0.0074	0.9941	0.0029	0.9862	0.0121	0.9890	0.0075
C2	0.7692	0.2801	0.6898	0.3211	0.4484	0.3307	0.6452	0.4108	0.3957	0.4259	0.7859	0.0708	0.7921	0.1134	0.8235	0.0661
C3	0.6118	0.3800	0.6786	0.3367	0.8507	0.1639	0.3934	0.4704	0.6914	0.4464	0.8819	0.0924	0.8191	0.1732	0.8270	0.1230
C4	0.6409	0.3262	0.7002	0.2534	0.7315	0.1876	0.4538	0.4362	0.6988	0.2473	0.8228	0.1017	0.8124	0.0550	0.8160	0.0768
C5	0.2026	0.2708	0.2620	0.3278	0.3151	0.3434	0.3528	0.3785	0.4822	0.2206	0.5186	0.1952	0.5533	0.1382	0.5514	0.2171
C6	0.8138	0.1262	0.3946	0.3704	0.7293	0.3589	0.4697	0.429	0.6627	0.3159	0.8874	0.0449	0.8890	0.0981	0.8554	0.1217
C7	0.5769	0.3219	0.8887	0.0589	0.5799	0.3247	0.5457	0.4266	0.8368	0.0976	0.8168	0.0376	0.8208	0.0260	0.8238	0.0404
C8	0.7490	0.3631	0.6369	0.3284	0.6097	0.3932	0.3285	0.3831	0.4691	0.3907	0.7373	0.2658	0.7678	0.2266	0.8648	0.2156
C9	0.9942	0.0094	0.9896	0.0180	0.9943	0.0103	0.6075	0.4427	0.8024	0.3119	0.9990	0.0013	0.9916	0.0117	0.9982	0.0041
OA	0.8225	0.098	0.7415	0.1353	0.7452	0.1717	0.5024	0.2719	0.7440	0.1239	0.8673	0.0754	0.8732	0.0680	0.9024	0.0590
AA	0.7039	0.0774	0.6811	0.0648	0.6860	0.0569	0.4903	0.0818	0.6701	0.0876	0.8271	0.0276	0.8258	0.0261	0.8388	0.0267
Kappa	0.7588	0.1219	0.6649	0.1504	0.6737	0.1798	0.425	0.2518	0.6612	0.1538	0.8193	0.0962	0.8268	0.0872	0.8652	0.0753

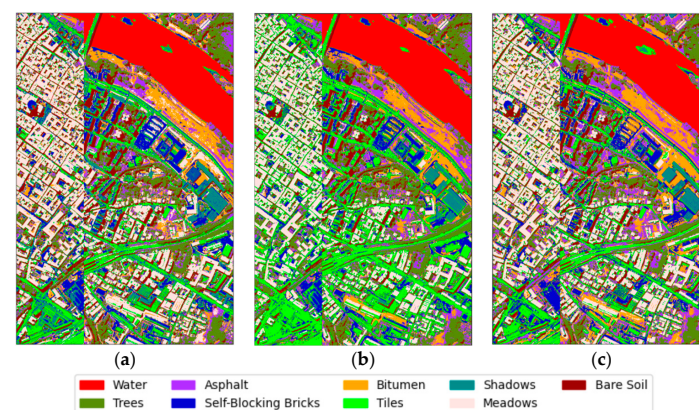


Figure 15. Visual classification results on the PaviaC dataset: (a) LSTM optimizer (0.9351), (b) MOE-A (0.9401), and (c) MOE-U (0.9423).

5. Discussion

Manually labeling HSI samples is not only time-consuming and laborious but also requires much professional expertise, so HSI classification always faces the challenge of few-shot learning. The main difficulty is to improve the generalization ability of few-shot classifiers and avoid overfitting, where traditional human-designed optimizers (e.g., Adam, Adagrad, SGD) are widely recognized as being unable to achieve this aim. As an advance of deep learning, meta-learning becomes a powerful tool to deal with the issue of few-shot learning. It always runs fast, since only very limited samples are used to train the meta-learning model, leading to low computational costs and memory requirements. In this study, we focused on the N-way 10-shot learning setting, i.e., we randomly picked 10 samples from each of N land-cover classes to form the training set. We proposed an improvement of meta-optimizers through an ensemble of meta-optimizers and a novel update integration process. We performed periodic annealing on the learning rate during the meta-training process, leading to building an ensemble of meta-optimizers without adding training costs. The incorporation of the update integration process increased the computational costs during the iterative process but brought a significant enhancement in classification accuracy.

In seven classification tasks on five real-world HSI datasets, we trained meta-optimizers on the source HSI dataset and tested their performance on different datasets from the same or different sensors. The real-world HSI classification experimental results showed that, with the help of the learned knowledge, the meta-optimizers outperformed the human-designed optimizers in terms of convergence speed, final convergence value, OA, AA, and Kappa. Moreover, our proposed MOE-U performed better than the single meta-optimizer (LSTM optimizer) and MOE-A, proving that an ensemble of meta-optimizers could achieve better results using our proposed update integration algorithm. Multiple meta-optimizers in an ensemble contain more parameters than a single meta-optimizer, thus containing more useful knowledge. When optimizing the predictive model, multiple meta-optimizers in an ensemble have a higher probability of generating good updates. The proposed update integration algorithm is more likely to choose good updates. Finally, our MOE-U achieved the best results in all real-world HSI classification experiments.

6. Conclusions

When lacking training samples, the widely used human-designed optimizers, such as Adam, AdaGrad, RMSprop, SGD, and SGD with momentum, can cause the model to fall into severe overfitting. To solve this issue, we present a meta-optimizer ensemble for HSI classification, which learns prior knowledge from the source HSI dataset and then uses the learned knowledge to train the predictive model on the target HSI dataset with limited training samples. By combining the advantages of ensemble learning and meta-learning, the meta-optimizer ensemble is improved in terms of overall performance and generalization ability. Moreover, we propose an effective update integration algorithm to incorporate the candidate updates generated by the meta-optimizer ensemble into the final update. The experimental results on multiple few-shot HSI classification tasks demonstrate the superiority and effectiveness of our proposed methods. Compared with the widely used human-designed optimizers, our meta-optimizer ensemble makes the predictive model converge faster on target HSI datasets while achieving better OA, AA, and Kappa coefficient results. Except for HSI classification, our improvement of meta-optimizers has the potential to improve various few-shot learning models in broad fields, including numerical analysis, computation, medical imaging, industrial detection, and other applications.

Author Contributions: T.H. and Z.Z. are co-first authors. Conceptualization, Z.Z.; Methodology, T.H. and Z.Z.; Software, T.H.; Validation, T.H.; Formal analysis, Z.Z., T.H. and M.J.C.C.; Investigation, Z.Z.; Resources, T.H.; Writing—original draft, T.H. and Z.Z.; Writing—review & editing, Z.Z. and M.J.C.C. All authors have read and agreed to the published version of the manuscript.

Funding: The corresponding author was supported by the European Commission Horizon 2020 Framework Program No. 861584 and the Taishan Distinguished Professor Fund No. 20190910.

Data Availability Statement: The data presented in this study are available on request from the corresponding author due to privacy reasons.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Kumar, B.; Dikshit, O.; Gupta, A.; Singh, M.K. Feature extraction for hyperspectral image classification: A review. *Int. J. Remote Sens.* **2020**, *41*, 6248–6287. [\[CrossRef\]](#)
2. Deng, B.; Jia, S.; Shi, D.M. Deep Metric Learning-Based Feature Embedding for Hyperspectral Image Classification. *IEEE Trans. Geosci. Remote Sens.* **2020**, *58*, 1422–1435. [\[CrossRef\]](#)
3. Li, S.T.; Song, W.W.; Fang, L.Y.; Chen, Y.S.; Ghamisi, P.; Benediktsson, J.A. Deep Learning for Hyperspectral Image Classification: An Overview. *IEEE Trans. Geosci. Remote Sens.* **2019**, *57*, 6690–6709. [\[CrossRef\]](#)
4. Ahmad, M.; Shabbir, S.; Roy, S.K.; Hong, D.F.; Wu, X.; Yao, J.; Khan, A.M.; Mazzara, M.; Distefano, S.; Chanussot, J. Hyperspectral Image Classification-Traditional to Deep Models: A Survey for Future Prospects. *IEEE J. Sel. Top. Appl. Earth Observ. Remote Sens.* **2022**, *15*, 968–999. [\[CrossRef\]](#)
5. Wang, C.Y.; Liu, B.H.; Liu, L.P.; Zhu, Y.J.; Hou, J.L.; Liu, P.; Li, X. A review of deep learning used in the hyperspectral image analysis for agriculture. *Artif. Intell. Rev.* **2021**, *54*, 5205–5253. [\[CrossRef\]](#)
6. Govender, M.; Chetty, K.; Bulcock, H. A review of hyperspectral remote sensing and its application in vegetation and water resource studies. *Water SA* **2007**, *33*, 145–151. [\[CrossRef\]](#)
7. Zhang, Z.H.; Huisingh, D. Combating desertification in China: Monitoring, control, management and revegetation. *J. Clean Prod.* **2018**, *182*, 765–775. [\[CrossRef\]](#)
8. Chutia, D.; Bhattacharyya, D.; Sarma, K.K.; Kalita, R.; Sudhakar, S. Hyperspectral remote sensing classifications: A perspective survey. *Trans. GIS* **2016**, *20*, 463–490. [\[CrossRef\]](#)
9. Hong, D.F.; Yokoya, N.; Ge, N.; Chanussot, J.; Zhu, X.X. Learnable manifold alignment (LeMA): A semi-supervised cross-modality learning framework for land cover and land use classification. *ISPRS-J. Photogramm. Remote Sens.* **2019**, *147*, 193–205. [\[CrossRef\]](#)
10. Mishra, N.B.; Crews, K.A. Mapping vegetation morphology types in a dry savanna ecosystem: Integrating hierarchical object-based image analysis with Random Forest. *Int. J. Remote Sens.* **2014**, *35*, 1175–1198. [\[CrossRef\]](#)
11. Ghamisi, P.; Plaza, J.; Chen, Y.S.; Li, J.; Plaza, A. Advanced Spectral Classifiers for Hyperspectral Images A review. *IEEE Geosci. Remote Sens. Mag.* **2017**, *5*, 8–32. [\[CrossRef\]](#)
12. Cheng, G.; Guo, L.; Zhao, T.Y.; Han, J.W.; Li, H.H.; Fang, J. Automatic landslide detection from remote-sensing imagery using a scene classification method based on BoVW and pLSA. *Int. J. Remote Sens.* **2013**, *34*, 45–59. [\[CrossRef\]](#)
13. Martha, T.R.; Kerle, N.; van Westen, C.J.; Jetten, V.; Kumar, K.V. Segment Optimization and Data-Driven Thresholding for Knowledge-Based Landslide Detection by Object-Based Image Analysis. *IEEE Trans. Geosci. Remote Sens.* **2011**, *49*, 4928–4943. [\[CrossRef\]](#)
14. Yang, J.X.; Zhao, Y.Q.; Chan, J.C.W. Learning and Transferring Deep Joint Spectral-Spatial Features for Hyperspectral Classification. *IEEE Trans. Geosci. Remote Sens.* **2017**, *55*, 4729–4742. [\[CrossRef\]](#)
15. Zhang, L.F.; Zhang, L.P.; Tao, D.C.; Huang, X. On Combining Multiple Features for Hyperspectral Remote Sensing Image Classification. *IEEE Trans. Geosci. Remote Sens.* **2012**, *50*, 879–893. [\[CrossRef\]](#)
16. Chen, Y.; Nasrabadi, N.M.; Tran, T.D. Hyperspectral Image Classification Using Dictionary-Based Sparse Representation. *IEEE Trans. Geosci. Remote Sens.* **2011**, *49*, 3973–3985. [\[CrossRef\]](#)
17. Li, J.; Marpu, P.R.; Plaza, A.; Bioucas-Dias, J.M.; Benediktsson, J.A. Generalized Composite Kernel Framework for Hyperspectral Image Classification. *IEEE Trans. Geosci. Remote Sens.* **2013**, *51*, 4816–4829. [\[CrossRef\]](#)
18. Benediktsson, J.A.; Palmason, J.A.; Sveinsson, J.R. Classification of hyperspectral data from urban areas based on extended morphological profiles. *IEEE Trans. Geosci. Remote Sens.* **2005**, *43*, 480–491. [\[CrossRef\]](#)
19. Jia, S.; Shen, L.L.; Li, Q.Q. Gabor Feature-Based Collaborative Representation for Hyperspectral Imagery Classification. *IEEE Trans. Geosci. Remote Sens.* **2015**, *53*, 1118–1129.
20. Camps-Valls, G.; Bruzzone, L. Kernel-based methods for hyperspectral image classification. *IEEE Trans. Geosci. Remote Sens.* **2005**, *43*, 1351–1362. [\[CrossRef\]](#)
21. Ham, J.; Chen, Y.C.; Crawford, M.M.; Ghosh, J. Investigation of the random forest framework for classification of hyperspectral data. *IEEE Trans. Geosci. Remote Sens.* **2005**, *43*, 492–501. [\[CrossRef\]](#)
22. Licciardi, G.; Marpu, P.R.; Chanussot, J.; Benediktsson, J.A. Linear Versus Nonlinear PCA for the Classification of Hyperspectral Data Based on the Extended Morphological Profiles. *IEEE Geosci. Remote Sens. Lett.* **2012**, *9*, 447–451. [\[CrossRef\]](#)
23. Villa, A.; Benediktsson, J.A.; Chanussot, J.; Jutten, C. Hyperspectral Image Classification With Independent Component Discriminant Analysis. *IEEE Trans. Geosci. Remote Sens.* **2011**, *49*, 4865–4876. [\[CrossRef\]](#)
24. Bandos, T.V.; Bruzzone, L.; Camps-Valls, G. Classification of Hyperspectral Images with Regularized Linear Discriminant Analysis. *IEEE Trans. Geosci. Remote Sens.* **2009**, *47*, 862–873. [\[CrossRef\]](#)

25. Falco, N.; Benediktsson, J.A.; Bruzzone, L. Spectral and Spatial Classification of Hyperspectral Images Based on ICA and Reduced Morphological Attribute Profiles. *IEEE Trans. Geosci. Remote Sens.* **2015**, *53*, 6223–6240. [\[CrossRef\]](#)
26. Chen, Y.S.; Lin, Z.H.; Zhao, X.; Wang, G.; Gu, Y.F. Deep Learning-Based Classification of Hyperspectral Data. *IEEE J. Sel. Top. Appl. Earth Observ. Remote Sens.* **2014**, *7*, 2094–2107. [\[CrossRef\]](#)
27. Zhang, M.M.; Li, W.; Du, Q. Diverse Region-Based CNN for Hyperspectral Image Classification. *IEEE Trans. Image Process.* **2018**, *27*, 2623–2634. [\[CrossRef\]](#)
28. Yu, S.Q.; Jia, D.; Xu, C.Y. Convolutional neural networks for hyperspectral image classification. *Neurocomputing* **2017**, *219*, 88–98. [\[CrossRef\]](#)
29. He, N.J.; Paoletti, M.E.; Haut, J.M.; Fang, L.Y.; Li, S.T.; Plaza, A.; Plaza, J. Feature Extraction With Multiscale Covariance Maps for Hyperspectral Image Classification. *IEEE Trans. Geosci. Remote Sens.* **2019**, *57*, 755–769. [\[CrossRef\]](#)
30. Mei, S.H.; Ji, J.Y.; Geng, Y.H.; Zhang, Z.; Li, X.; Du, Q. Unsupervised Spatial-Spectral Feature Learning by 3D Convolutional Autoencoder for Hyperspectral Classification. *IEEE Trans. Geosci. Remote Sens.* **2019**, *57*, 6808–6820. [\[CrossRef\]](#)
31. Xue, Z.H.; Zhou, Y.Y.; Du, P.J. S3Net: Spectral-Spatial Siamese Network for Few-Shot Hyperspectral Image Classification. *IEEE Trans. Geosci. Remote Sens.* **2022**, *60*, 5531219. [\[CrossRef\]](#)
32. Finn, C.; Abbeel, P.; Levine, S. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. In Proceedings of the 34th International Conference on Machine Learning, Sydney, Australia, 6–11 August 2017; pp. 1126–1135.
33. Hospedales, T.; Antoniou, A.; Micaelli, P.; Storkey, A. Meta-Learning in Neural Networks: A Survey. *IEEE Trans. Pattern Anal. Mach. Intell.* **2022**, *44*, 5149–5169. [\[CrossRef\]](#)
34. Wang, Y.Q.; Yao, Q.M.; Kwok, J.T.; Ni, L.M. Generalizing from a Few Examples: A Survey on Few-shot Learning. *ACM Comput. Surv.* **2020**, *53*, 1–34.
35. Lee, Y.; Choi, S. Gradient-Based Meta-Learning with Learned Layerwise Metric and Subspace. In Proceedings of the 35th International Conference on Machine Learning (ICML), Stockholm, Sweden, 10–15 July 2018; pp. 2927–2936.
36. Gao, K.L.; Liu, B.; Yu, X.C.; Zhang, P.Q.; Tan, X.; Sun, Y.F. Small sample classification of hyperspectral image using model-agnostic meta-learning algorithm and convolutional neural network. *Int. J. Remote Sens.* **2021**, *42*, 3090–3122. [\[CrossRef\]](#)
37. Li, X.R.; Cao, Z.Y.; Zhao, L.Y.; Jiang, J.F. ALPN: Active-Learning-Based Prototypical Network for Few-Shot Hyperspectral Imagery Classification. *IEEE Geosci. Remote Sens. Lett.* **2022**, *19*, 5508305. [\[CrossRef\]](#)
38. Bottou, L. Large-Scale Machine Learning with Stochastic Gradient Descent. In Proceedings of the 19th International Conference on Computational Statistics (COMPSTAT'2010), Paris, France, 22–27 August 2010; pp. 177–186.
39. Mukkamala, M.C.; Hein, M. Variants of RMSProp and Adagrad with Logarithmic Regret Bounds. In Proceedings of the 34th International Conference on Machine Learning, Sydney, Australia, 6–11 August 2017; pp. 2545–2553.
40. Duchi, J.; Hazan, E.; Singer, Y. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *J. Mach. Learn. Res.* **2011**, *12*, 2121–2159.
41. Kingma, D.P.; Ba, J. Adam: A method for stochastic optimization. *arXiv* **2014**, arXiv:1412.6980.
42. Huisman, M.; van Rijn, J.N.; Laat, A. A survey of deep meta-learning. *Artif. Intell. Rev.* **2021**, *54*, 4483–4541. [\[CrossRef\]](#)
43. Andrychowicz, M.; Denil, M.; Colmenarejo, S.G.; Hoffman, M.W.; Pfau, D.; Schaul, T.; Shillingford, B.; de Freitas, N. Learning to learn by gradient descent by gradient descent. In Proceedings of the 30th Conference on Neural Information Processing Systems (NIPS), Barcelona, Spain, 5–10 December 2016.
44. Ravi, S.; Larochelle, H. Optimization as a model for few-shot learning. In Proceedings of the International Conference on Learning Representations, San Juan, Puerto Rico, 2–4 May 2016; pp. 1–11.
45. Wang, S.P.; Sun, J.; Xu, Z.B. HyperAdam: A Learnable Task-Adaptive Adam for Network Training. In Proceedings of the 33rd AAAI Conference on Artificial Intelligence/31st Innovative Applications of Artificial Intelligence Conference/9th AAAI Symposium on Educational Advances in Artificial Intelligence, Honolulu, HI, USA, 27 January–1 February 2019; pp. 5297–5304.
46. Tian, Y.J.; Zhao, X.X.; Huang, W. Meta-learning approaches for learning-to-learn in deep learning: A survey. *Neurocomputing* **2022**, *494*, 203–223. [\[CrossRef\]](#)
47. Wichrowska, O.; Maheswaranathan, N.; Hoffman, M.W.; Colmenarejo, S.G.; Denil, M.; de Freitas, N.; Sohl-Dickstein, J. Learned Optimizers that Scale and Generalize. In Proceedings of the 34th International Conference on Machine Learning, Sydney, Australia, 6–11 August 2017; pp. 5744–5753.
48. Zhenguo, L.; Fengwei, Z.; Fei, C.; Hang, L. Meta-SGD: Learning to Learn Quickly for Few Shot Learning. *arXiv* **2017**, arXiv:1707.09835.
49. Chen, Y.T.; Hoffman, M.W.; Colmenarejo, S.G.; Denil, M.; Lillicrap, T.P.; Botvinick, M.; de Freitas, N. Learning to Learn without Gradient Descent by Gradient Descent. In Proceedings of the 34th International Conference on Machine Learning, Sydney, Australia, 6–11 August 2017.
50. Gao, H.; Yixuan, L.; Pleiss, G.; Zhuang, L.; Hopcroft, J.E.; Weinberger, K.Q. Snapshot Ensembles: Train 1, get M for free. *arXiv* **2017**, arXiv:1704.00109.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.