ISPRS Journal of Photogrammetry and Remote Sensing 166 (2020) 241–254

Contents lists available at ScienceDirect

ISPRS Journal of Photogrammetry and Remote Sensing

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

Land-cover classification of multispectral LiDAR data using CNN with optimized hyper-parameters

Suoyan Pan[a], Haiyan Guan[b,c,*], Yating Chen[b], Yongtao Yu[d], Wesley Nunes Gonçalves[e], José Marcato Junior[f], Jonathan Li[g]

[a] School of Geographical Sciences, Nanjing University of Information Science and Technology, Nanjing, JS 210044, China
[b] School of Remote Sensing and Geomatics Engineering, Nanjing University of Information Science and Technology, Nanjing, JS 210044, China
[c] Suzhou Xiaoqi Information Technology Co., Ltd., 162 Renmin South Road, Chengxiang, Taicang, Jiangsu, JS 215400, China
[d] Faculty of Computer and Software Engineering, Huaiyin Institute of Technology, Huaian 223003, China
[e] Faculty of Computer Science and Faculty of Engineering, Architecture and Urbanism and Geography, Federal University of Mato Grosso do Sul, Brazil
[f] Faculty of Engineering, Architecture and Urbanism, and Geography, Federal University of Mato Grosso do Sul, Brazil
[g] Department of Geography and Environmental Management, University of Waterloo, Waterloo, ON N2L 3G1, Canada

ARTICLE INFO

Keywords:
Multi-spectral LiDAR
CNN
Land-cover classification
Hyper-parameters

ABSTRACT

Multispectral LiDAR (Light Detection And Ranging) is characterized of the completeness and consistency of its spectrum and spatial geometric data, which provides a new data source for land-cover classification. In recent years, the convolutional neural network (CNN), compared with traditional machine learning methods, has made a series of breakthroughs in image classification, object detection, and image semantic segmentation due to its stronger feature learning and feature expression abilities. However, traditional CNN models suffer from some issues, such as a large number of layers, leading to higher computational cost. To address this problem, we propose a CNN-based multi-spectral LiDAR land-cover classification framework and analyze its optimal parameters to improve classification accuracy. This framework starts with the preprocessing of multi-spectral 3D LiDAR data into 2D images. Next, a CNN model is constructed with seven fundamental functional layers, and its hyper-parameters are comprehensively discussed and optimized. The constructed CNN model with the optimized hyper-parameters was tested on the Titan multi-spectral LiDAR data, which include three wavelengths of 532 nm, 1064 nm, and 1550 nm. Extensive experiments demonstrated that the constructed CNN with the optimized hyper-parameters is feasible for multi-spectral LiDAR land-cover classification tasks. Compared with the classical CNN models (i.e., AlexNet, VGG16 and ResNet50) and our previous studies, our constructed CNN model with the optimized hyper-parameters is superior in computational performance and classification accuracies.

1. Introduction

Multi-spectral LiDAR (Light Detection And Ranging), as a new type of active remote sensing technology, has been widely used in land-cover classification, vegetation mapping, phreatic water depth measurement, and complex terrain survey (Scaioni et al., 2018; Yan et al., 2018; Ekhtari et al., 2018). Multi-spectral LiDAR system can provide point clouds coming from multiple channels with different wavelengths. More terms, such as multi-wavelength and multi-channel, are changeably used in the field of laser scanning. Airborne multi-spectral LiDAR data provide relatively complete and consistent spectral and spatial geometric information, contributing to land-cover and land-use classification (Bakuła et al., 2016; Matikainen et al., 2017; Shaker et al., 2019). Wichmann et al. (2015) studied the spectral patterns of some land covers in multi-spectral LiDAR data. Ekhtari et al. (2018) demonstrated that the recorded intensities of laser returns together with spatial metrics calculated from the three-dimensional locations of laser returns are sufficient for classifying the point cloud into ten distinct land cover classes, including grass, trees, two classes of soil, four classes of pavement, and two classes of buildings. Pan and Guan (2018), using multi-spectral LiDAR data, investigated a deep Boltzmann machine (DBM) model based high-level feature representation and two machine learning based low-level feature extraction and selection methods (principal component analysis (PCA) and random forest (RF)) in land

* Corresponding author at: School of Remote Sensing and Geomatics Engineering, Nanjing University of Information Science and Technology, Nanjing, JS 210044, China.
E-mail addresses: guanhy.nj@nuist.edu.cn (H. Guan), wesley.goncalves@ufms.br (W. Nunes Gonçalves), jose.marcato@ufms.br (J. Marcato Junior), junli@uwaterloo.ca (J. Li).

https://doi.org/10.1016/j.isprsjprs.2020.05.022
Received 30 November 2019; Received in revised form 24 May 2020; Accepted 27 May 2020

cover classification. These studies show that, compared with single-wavelength LiDAR data and optical images, multi-spectral LiDAR data are more suitable for conventional land-cover classification and mapping.

Currently, by using multi-spectral LiDAR data, many land-cover classification studies have been increasingly presented (Fernandez-Diaz et al., 2016; Hopkinson et al., 2016; Miller et al., 2016; Morsy et al., 2017; Chen et al., 2018). These land-cover classification methods are generally performed on either three-dimensional (3D) LiDAR points (Kumar et al., 2019b) or two-dimensional (2D) feature images interpolated from 3D points (Teo and Wu, 2017). Matikainen et al. (2017) demonstrated that the land-cover classification accuracies of 3D multi-spectral LiDAR points were better than those considering 2D interpolated feature images. However, with the development of LiDAR technologies, higher point density has been achieved, which implies in heavier computational burden when land-cover classification is performed using 3D LiDAR points (Kang and Yang, 2018). Therefore, converting 3D airborne multi-spectral LiDAR points into 2D feature images has been shown as an effective way to obtain land-cover maps by using established image processing algorithms (e.g., maximum likelihood, support vector machine, decision tree, and random forest) (Pan et al., 2019; Wang et al., 2019a, 2019b). Moreover, the increase in point density guarantees the details of land covers in 2D interpolated feature images.

Although the aforementioned methods have provided key technologies for land-cover classification by using multi-spectral LiDAR data, all of them conduct analysis based on traditional low-level LiDAR metrics (e.g., variance, entropy, and skewness), which are incapable of extracting comprehensive, high-level features from multi-spectral LiDAR data. Deep learning methods, which focus on how to increase the layer depth of the network architectures and reduce the fitting parameters of each layer, are capable of extracting deep, high-level features from the original data, thereby achieving satisfactory classification accuracy and better performance (Kumar et al., 2019a; Liu et al., 2019). In conventional deep learning architectures, the convolutional neural network (CNN) has gained its reputation in image processing (Wang et al., 2019a, 2019b; Manyala et al., 2019). CNN, a multi-layer deep learning model inspired by biology, uses a single neural network from training to prediction. The CNN was first introduced in (Fukushima, 1988), improved by Lecun et al. (1998), as well as refined and simplified posteriorly in (Tivive and Bouzerdoum, 2005; Li et al., 2006). A CNN uses stacked convolution kernels to learn spectral and spatial information from images, contributing to extract high-level abstract features. With the deepening of the neural network, the CNN implements classification tasks by establishing a connection between input samples and output labels. Due to the availability of large-scale training data and the implementation of high-performance workstations, the CNN outperforms some traditional neural network methods (e.g., Auto Encoder, Sparse Coding, and Restricted Boltzmann Machine) in many vision-related tasks, such as image classification (Jaswal et al., 2014), target detection (Soon et al., 2018), scene marking (Rangel et al., 2018), and face recognition (Hansen et al., 2018). Moreover, the CNN is also successfully applied to some non-vision-related tasks, such as speech recognition (Cai and Liu, 2016) and text classification (Zhu et al., 2018).

Recently, CNN has made many breakthroughs in high-resolution remote sensing image classification (Xu et al., 2018; Lu et al., 2018). Some CNN-based methods achieved better classification results by piling up a large number of repeated functional layers. Li et al. (2019) designed a deep CNN, with multiple layers, including four convolutional layers with 2 × 2 convolution kernels, a maximum pooling layer with 2 × 2 pooling windows, and two fully-connected layers. Experimental results demonstrated that Li's model outperformed support vector machine (SVM) in the Indian Pines and Salina dataset. Nevertheless, due to the relative complexity of the model, it took longer time to train. Moreover, Zhang et al. (2018a), Zhang et al. (2018b) proposed an innovative object-based CNN (OCNN) method, which combined a $OCNN_{128}$ model (a CNN model with the input size of 128 × 128 and eight layers of alternating convolutional and pooling layers) and a $OCNN_{48}$ model (a CNN model with the input size of 48 × 48 and six layers of alternating convolutional and pooling layers), to label general and linearly-shaped objects in complex urban environments. Experimental results showed that, compared to the $OCNN_{48}$ model, although the $OCNN_{128}$ model achieved the classification accuracy improvement of about 3.68%, it required more 1.03 h. Furthermore, to automatically classify mobile laser scanning (MLS) data, Kumar et al. (2019b), presented a single CNN (SCN) model (including an input layer, seven convolutional layers, five fully-connected layers, and an output layer) and a multi-faced CNN (MFC) model, which used multiple facets of an MLS sample as inputs to different SCNs for providing additional classification information. Experiments conducted on the KITTI dataset demonstrated that the overall accuracy and Kappa index reached 86.0% and 0.813 for SCN, 94.3% and 0.924 for MFC, respectively. Although the classification accuracies of the MFC model were obviously higher than those of the SCN model, the training time of the MFC model was also correspondingly longer than that of the SCN model due to the more complex CNN architecture of the MFC model. Therefore, to trade off the land-cover classification accuracy and computational efficiency, it is necessary to build an appropriate CNN architecture with the minimum functional layers, rather than a very complex architecture.

Although the CNN model and its variants have proven their feasibility and applicability in hyper-spectral data classification tasks (Guidici and Clark, 2017; Li et al., 2018a, 2018b), even in LiDAR data processing (Gao et al., 2018; He et al., 2019), they are very rarely applied to multi-spectral LiDAR land-cover classification tasks. Not to mention that some CNN parameters have not been thoroughly studied for land-cover classification. The CNN parameters are divided into hyper-parameters and non-hyper-parameters. Non-hyper-parameters are continuously adjusted during the model training stage. Hyper-parameters (e.g., dimension, input size, convolution kernel size, number of convolution kernels, pooling window size, and learning rate) determine the model type, which have a significant impact on the training and final predictions. Therefore, it is highly complicated and time consuming to build a powerful CNN model and find its appropriate parameters for land-cover classification tasks. However, at present, there are no clear rules to optimize the CNN hyper-parameters, and they are mostly determined by the experience and intuition of a designer (LeCun et al., 1998; Guo et al., 2016; Unnikrishnan et al., 2018). For example, for the convolution kernel size, all layers were set to the same value (e.g., 5) in the LeNet-5 architecture, while in the AlexNet and ZFNet models, different convolutional layers were set to different values. Dahou et al. (2019) used a differential evolution (DE) algorithm to optimize five CNN parameters, namely convolution kernel size, number of convolution kernels, number of neurons in fully-connected layer, and dropout rate. Experimental results showed that the DE-CNN model with the five optimal parameters achieved higher classification accuracy and cost less time than the traditional CNN models. Bergstra and Bengio, (2012) stated that, to obtain relatively high classification accuracy and computational efficiency, it is very important to test various values and select the most appropriate values for the CNN hyper-parameters.

Thus, to decrease computational complexity in the multi-spectral LiDAR land-cover classification task, we construct a light CNN model with seven fundamental layers. Next, to obtain high classification accuracies, we conduct an exploratory study on the CNN model to explore and discuss its hyper-parameters using the Teledyne Optech's Titan multi-spectral LiDAR data. In summary, the main contributions of our multi-spectral LiDAR land-over classification architecture are highlighted as follows:

(1) Our CNN model constructed for the multi-spectral LiDAR land-cover classification task is composed of only commonly used

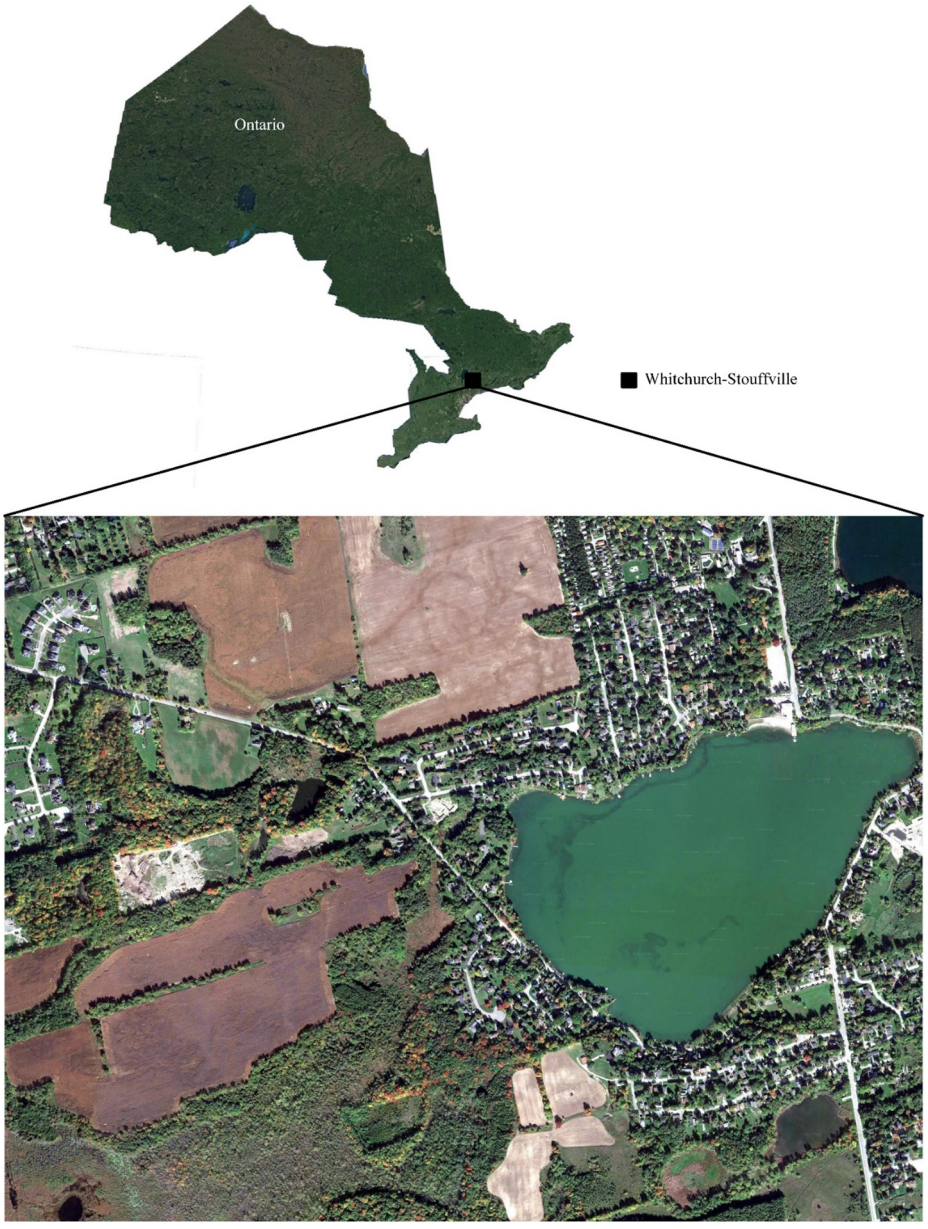

Fig. 1. Illustration of the study area.

functional layers, rather than piling up by a large number of repeated functional layers, contributing to the improvement of computational performance.

(2) By using a control variable method, the hyper-parameters of the constructed CNN model are discussed and optimized to provide guidelines for multi-spectral LiDAR land-cover classification.

The rest of this paper is organized as follows. Section 2 details the Titan multi-spectral LiDAR data, followed by data pre-processing. Section 3 describes our CNN model for the multi-spectral LiDAR land-cover classification task. Section 4 conducts a set of experiments including hyper-parameter optimization and algorithm comparison. Finally, the concluding remarks are presented in Section 5.

2. Data and data preprocessing

2.1. Multi-spectral LiDAR test data

As shown in Fig. 1, the study area is a small town located in Whitchurch-Stouffville, Ontario, Canada with an area of 2,052 m × 1,566 m and with the center position at latitude and longitude of 43°58′00″, 79°15′00″, respectively. The area contains a rich variety of objects, such as roads, trees, grass, soil, and water, which contribute to the implementation of our land-cover classification study.

The experimental data were collected using an airborne Titan multi-spectral LiDAR system, produced by the Teledyne Optech. The detailed specifications of the multi-spectral LiDAR system are presented in Table 1. The acquired multi-spectral LiDAR data, containing nineteen vertically intersecting flights, cover an area of about 25 square

Table 1
Specifications of the Titan Airborne System.

Parameters	Channel 1	Channel 2	Channel 3
Wavelength(nm)	1550 (SWIR)	1064 (NIR)	532 (GREEN)
Deflection Angle(°)	3.5 (forward)	nadir	7 (forward)
Flight Altitude(m)	~1000	~1000	~1000
Point Density(/m^2)	3.6	3.6	3.6

kilometers. Radiometric correction should be applied to the Titan multispectral LiDAR data (Briese et al., 2012) before we test them on land-cover classification tasks. Note that, in this study, because the system parameters and trajectories were unavailable, the three channels of intensities were directly used from the LiDAR outputs (as a file format of ASPRS LAS files) without intensity calibration. Iterative closest points (ICP) was used to roughly register these strips. Similarly, without control points or reference points, the geometric quality were not statistically reported. Thus, we selected the study area from the one strip for assessing our land cover classification method.

To improve the quality of training sample selection and classification accuracy evaluation, a high-resolution remote sensing image corresponding to the study area from Google Earth was used (See Fig. 1).

2.2. Data Pre-processing

The Titan multi-spectral LiDAR system generates three independent point clouds in three channels, i.e., 532 nm, 1064 nm, and 1550 nm. To improve the efficiency of point cloud data processing, especially for the Titan multi-spectral LiDAR data, we first merged the three independent point clouds into a single point cloud, where each point contains three spectral wavelengths, and then interpolated the merged point cloud into a set of 2D raster images, considering the elevation and multi-wavelength intensities. The specific data pre-processing is briefly described as follows. Specifically, each of the three single-wavelength point clouds was taken as the reference data, in which each point was processed to find its neighbors in the other two wavelengths of point clouds using a nearest neighbor searching algorithm. Because the average point density of single wavelength was about 3.6 points/m^2, the searching distance in this study was set to 1.0 m to obtain sufficient points in the two wavelengths of point clouds. To obtain the intensities of the two other wavelengths, an inverse-distance-weighted (IDW) interpolation method was used. If there were no neighboring points in the one of two wavelengths, the intensity value of this wavelength was set to zero. As such, three wavelengths were merged into a single, multi-spectral point cloud with higher point density than that of single wavelength. To efficiently perform feature extraction in land cover classification, through the IDW interpolation method, the merged multi-spectral LiDAR data were rasterized into 2D grid data sets, by using elevation and intensities from different single wavelength and the merged multi-wavelength. Based on the fact that the average point density of each wavelength was 3.6 points/m2, the resolution of the 2D image was set to 0.5 m. Fig. 2 (a) and (b) show the elevation and multi-spectral images of the merged point cloud, respectively. To improve the radiometric quality, the multi-wavelength intensities have been normalized and performed relative radiometric correction.

3. Method

3.1. Preliminaries

A CNN consists of a series of convolutional, pooling, and nonlinear operations. The CNN, describing the mapping relationship between different functional layers, contains two important characteristics: local connectivity and shared weights. Local connectivity simplifies the CNN model by limiting the number of connected neurons. Shared weights further reduce the parameters of the model and finally simplify the model by setting the same connected weights between some neurons in the same layer. In the following, some basic modules mentioned in the CNN model will be introduced.

Convolutional layer: the convolutional layer, the core part of a CNN model, uses convolution kernels to extract features from input images via a set of convolutional operations. The convolutional operations convert a local receptive field (the connected region of any convolution kernel on the input image) into the pixels of the next layer.

Pooling layer: the pooling layer mainly performs down-sampling to further reduce the dimension of the feature maps and increase the robustness of feature extraction. To reduce the dimension, the pooling layer combines a local receptive field into a single neuron. Generally, there are three forms of pooling (e.g., maximum pooling, mean pooling, and random pooling).

Fully-Connected layer: the fully-connected layer mainly functions as an integrator and a classifier. The integrator means that the fully-connected layer integrates the image features in the feature maps through multiple convolutional layers and pooling layers to obtain high-level meaning of the features. The classifier means that the feature map generated by the convolutional layers is mapped to a fixed-length feature vector, and then the feature vector is used to calculate the score of the class to which it belongs and the error between the output and real values.

Batch normalization layer: the batch normalization layer normalizes activations of a previous layer at each batch to keep the mean activation value close to 0 and the standard deviation activation value close to 1. It can greatly increase the convergence speed, reduce over-fitting, decrease the insensitivity of initial weights, and further allow us to use a higher learning rate (Li et al., 2018a, 2018b).

Flatten layer: the flatten layer, transforming the input from multi-dimensional space to one-dimensional space, is commonly used for conversion from a convolutional layer to a fully-connected layer.

Dropout layer: dropout operation randomly sets the neuron value to 0 with a probability of 50% in each training batch. By this operation, the CNN becomes less sensitive to specific sets of neurons, which helps to reduce the interaction among hidden layer neurons, avoids the over-fitting phenomenon, and improves the generalization ability of the

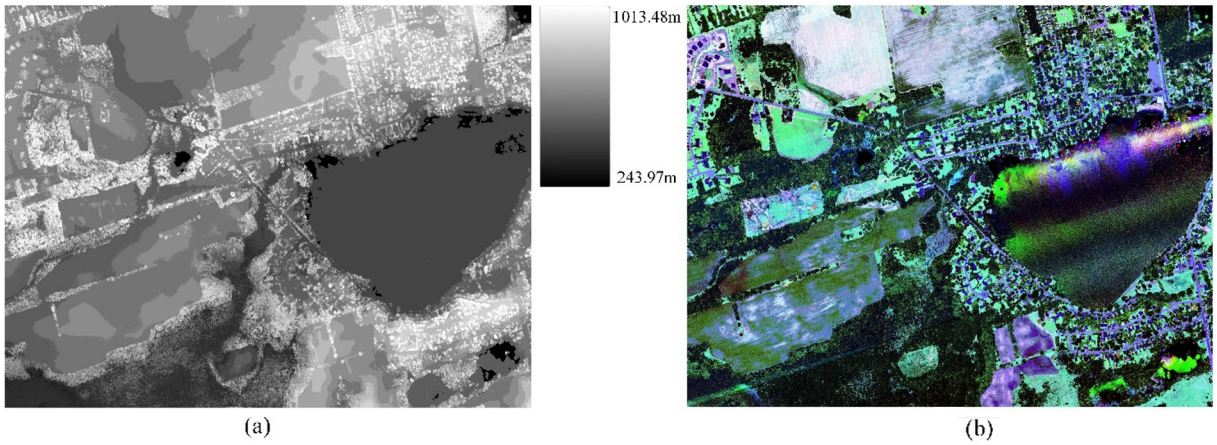

Fig. 2. 2D raster images, (a) elevation image, and (b) multi-spectral LiDAR image with three wavelengths.

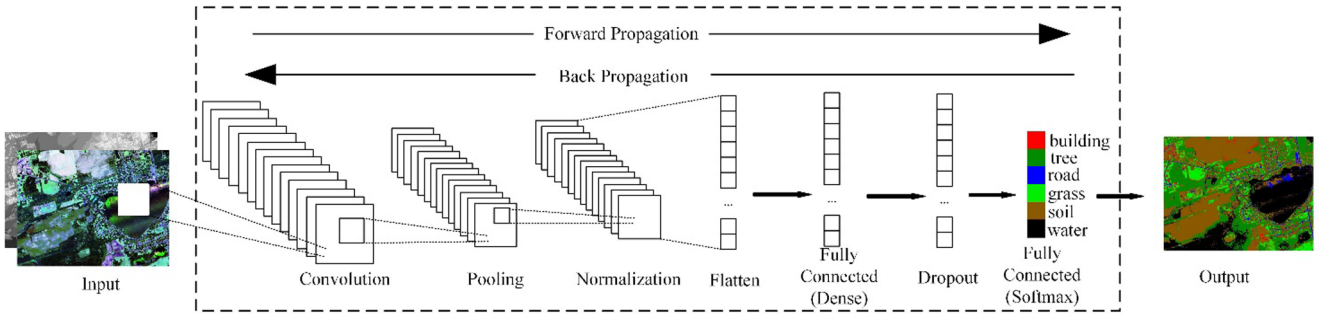

Fig. 3. Architecture of the constructed CNN model.

model (Srivastava et al, 2014).

3.2. Constructed CNN model

In this subsection, our CNN architecture is first described. Next, the learning algorithm is introduced to train the constructed CNN via an end-to-end training strategy. Finally, the inference stage is presented to obtain the land-cover classification results by using the Titan multi-spectral LiDAR data.

3.2.1. Our CNN architecture

For the multi-spectral LiDAR land-cover classification task, we design a CNN model which contains only seven fundamental hidden layers, namely, a convolutional layer, a pooling layer, two fully-connected layers, and three other functional layers (i.e., the batch normalization layer, the flatten layer, and the dropout layer), as shown in Fig. 3.

In the convolutional layer, the convolutional operation is calculated as follows:

$$y(n_1, n_2, \cdots, n_m) = \sum_{k_1=-\infty}^{\infty} \sum_{k_2=-\infty}^{\infty} \cdots \sum_{k_m=-\infty}^{\infty} x(k_1, k_2, \cdots\cdots, k_m) f(n_1 - k_1, n_2 - k_2, \cdots\cdots, n_m - k_m) \quad (1)$$

where m, x, f, and y are the dimension of the model, the input data, the local receptive field, and the output features, respectively. m can be set to 1, 2, and 3. For example, $m = 1$ means a 1D CNN model, indicating that the convolutional operation in this model moves only along one direction. To label each pixel, the depth direction is selected as the convolutional direction as shown in Fig. 4(a). Thus, this convolutional operation focuses on learning the relationships among only the input layers of each pixel. $m = 2$ represents a 2D CNN model, where the convolutional operation moves in two directions (the height and width directions) as shown in Fig. 4(b). Therefore, the 2D CNN model focuses attention on the correlations among pixels of the input layer. In a 3D CNN, m equals to 3, where the convolutional operation moves along all three directions (the height, width, and depth directions, where the depth is set to one in this study) as shown in Fig. 4(c). Thus, a 3D CNN model can better study both the relationships among the input layers of each pixel and the correlations among pixels of the input layer.

As shown in Fig. 4, CNN models with different dimensions have different convolutional directions. In order to obtain strong features from the input data fairly and effectively, it is necessary to adjust the input size in different dimensions. In the 1D CNN model ($m = 1$), because the convolution kernel moves along only one direction, the input size is defined as ($s_i \times s_i$, W), where s_i must be an odd number and greater than one to maintain the same pixel values of the input data. In the 2D CNN model, the input size is defined as (s_i, s_i, W) to use the neighboring information of the pixel to be processed. In the 3D CNN model, since the convolution kernel moves in three directions, the input size is defined as (W, s_i, s_i, 1), where W is the number of the raster dataset bands.

In the convolutional layer, convolution kernels are used to extract features from the input data. The convolution kernel size corresponds to the size of the local receptive field (f). The larger the convolution kernel size (the larger the value of f), the heavier the computational burden. The convolution kernel structures vary with the CNN dimensions. Note that the convolution kernel sizes are set to be odd and greater than one in the height, width, and depth dimensions, as well as the convolution kernel sizes in the height and width dimensions are set to the same value.

The convolution kernel sizes in the 1D and 2D CNN models are denoted as (s_c) and (s_c, s_c), respectively. The hyper-parameter, s_c is the kernel size in the height or width dimensions. The convolution kernel structure for the 3D CNN model is defined as (s_c, s_c, s_c'), where s_c' is the depth kernel size and the discussion of s_c' is consistent with that of s_c. Moreover, the number of convolution kernels (n_c) determines the

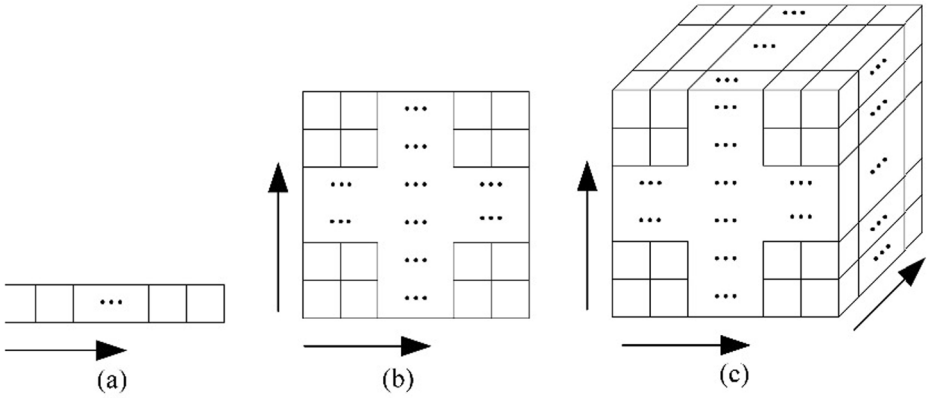

Fig. 4. Different convolutional directions of CNN with different dimensions (m), (a) m = 1, (b) m = 2 and (c) m = 3.

quantity of filters used in the convolutional layers and the number of feature images derived from these layers.

To ensure nonlinearity in the convolutional layer, we use Rectified Linear Unit (ReLU) as the activation function. The ReLU activation function (Cao et al., 2017) is formulated as follows:

$$y = \mathrm{ReLU}(x) = \max(0, x) \tag{2}$$

The ReLU activation function has the characteristics of unidirectionality and one-end saturation. It introduces sparsity into the CNN model to effectively handle disappearance of the gradient, thus greatly improving the convergence speed of the CNN model. Therefore, to enhance the feasibility and efficiency of the training process in the CNN model, we apply the ReLU activation function to the model (Cao et al., 2017).

The pooling layer is a key feature extraction operation in our CNN architecture. To reserve the strongest features, this study uses maximum pooling operations to down-sample the feature maps, by taking the maximum value of features in a specified window in each feature map. The pooling window size determines the amount of feature information involved and the computational complexity of the training model. Like the convolution kernel size described above, the pooling window size in 1D, 2D and 3D CNN models are defined as (s_p), (s_p, s_p), and (s_p, s_p, s_p'), respectively, where the hyper-parameter, s_p is the window size in the height or width dimensions and s_p' *is* the depth window size. Likewise, the discussion of s_p' is consistent with that of s_p.

As shown in Fig. 3, our CNN model includes two fully-connected layers: a dense layer and a softmax layer. The dense layer generates a 1D feature vector for further processing, and then roughly classifies the features extracted by the convolutional layer and down-sampled by the pooling layer. The number of the dense units (n_d) initially determines the coarse classification effect of the CNN model, thus it is necessary to discuss the hyper-parameter, n_d, in the dense layer. The softmax layer returns the original values of predictions. Because there are six types of land covers (building, tree, road, grass, soil, and water) in the study area as shown in Fig. 1, six neurons are designed in this layer, for the training set $\{(x_1, y_1), (x_2, y_2),\ldots, (x_n, y_n)\}$, where $y_i$is the sample label, $y_i \in \{1,2,\ldots,6\}$. When the input value x_i is given, the model calculates a k-dimensional vector ($k = 6$) as follows:

$$O_\theta(x_i) = \begin{bmatrix} p(y_i = 1|x_i; \theta) \\ p(y_i = 2|x_i; \theta) \\ \cdots \cdots \\ p(y_i = k|x_i; \theta) \end{bmatrix} = \frac{1}{\sum_{j=1}^{k} e^{\theta_j^T x_i}} \begin{bmatrix} e^{\theta_1^T x_i} \\ e^{\theta_2^T x_i} \\ \cdots \cdots \\ e^{\theta_k^T x_i} \end{bmatrix} \tag{3}$$

where $\sum_{j=1}^{k} e^{\theta_j^T x_i}$ normalizes the probability distribution to add all probability values up to 1, $\theta = [\theta_1, \theta_2, \ldots\ldots, \theta_k]$ is the parameter that the CNN model needs to be optimized, and the output value of each node describes the possibility of an input pixel belonging to a land cover. Therefore, the softmax layer provides the relative measurement of the likelihood that the input pixel falls into each target class (Vamplew et al., 2017).

3.2.2. Learning algorithm

For a CNN model, the parameters are divided into non-hyper-parameters and hyper-parameters (commonly including dimension, input size, convolution kernel size, number of convolution kernels, pooling kernel size, and learning rate, etc.). The non-hyper-parameters, referred to the weights and biases of the hidden layers, are constantly adjusted during the training process. The CNN training process aims to obtain the best combination of the model parameters. The most commonly used model training method is the Back Propagation algorithm, also known as BP algorithm.

The BP algorithm in this study can be divided into three steps:

(1) Forward Propagation, where the sample data are input to the network, and then transferred from the input layer to the output layer through layer-by-layer calculation to obtain the corresponding output results.
(2) Partial Derivative Calculation. The error term $\delta_i^{(l)}$ of neuron i in the l-th layer is inversely calculated, which represents the partial derivative of the loss function E of the network to the output value of the neuron.
(3) Parameters Update. According to the optimization algorithm, the gradient of each neuron parameter is calculated, and then each parameter is updated.

Let denote $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ as the training set, where y_i is the sample label and $x_i$is the input pixel. After forward propagation, for each input pixel, the output obtained at the softmax layer is the predicted value, denoted as $\bar{y}_i$. In this study, the loss function E is defined by Mean Squared Error (MSE), which is the sum of squared errors between the actual and predicted values of the training, as follows:

$$E(w, b) = \frac{1}{2}\sum_{i=1}^{n}(y_i - \bar{y}_i)^2 = \frac{1}{2}\sum_{i=1}^{n}(y_i - g[h_{w,b}(x_i)])^2 \tag{4}$$

where $h_{w,b}(x_i)$ is a function of input pixel x_i after forward propagation. The function includes the non-hyper-parameters: weights and biases.

For a model, the smaller the errors between the predicted and actual values, the better the prediction of the model. Therefore, a model training process can be seen as the process that minimizes the loss function $E(W, b)$. To solve the extreme points of the loss function $E(w, b)$, a gradient descent optimization method (Alagoz et al., 2016) is most commonly used. According to this method, the CNN parameter w is derived as follows. The calculation method of b is similar to w, and is omitted here.

$$w_{ij}^{(l)} \leftarrow w_{ij}^{(l)} - \eta\nabla w_{ij}^{(l)} = w_{ij}^{(l)} - \eta\frac{\partial E(W, b)}{\partial w_{ij}^{(l)}} \tag{5}$$

where $w_{ij}^{(l)}$ is the weight of neuron i from the l-th layer to the (l-1)-th layer, and ∇ is a gradient operator, and η is the step size, also known as the learning rate.

The chain derivation rules are presented as follows.

$$\nabla w_{ij}^{(l)} = \frac{\partial E(w,b)}{\partial w_{ij}^{(l)}} = \frac{\partial E(w,b)}{\partial h_i^{(l)}}\frac{\partial h_i^{(l)}}{\partial w_{ij}^{(l)}}$$
$$= \frac{\partial E(W,b)}{\partial a_i^{(l)}}\frac{\partial a_i^{(l)}}{\partial h_i^{(l)}}a_j^{(l-1)} = \frac{\partial E(W,b)}{\partial a_i^{(l)}}g'(h_i^{(l)})a_j^{(l-1)} \tag{6}$$

where $a_i^{(l)}$ and $a_j^{(l-1)}$ are the output values of node i in the l-th layer and node j in the (l-1)-th layer, respectively. $h_i^{(l)}$ is the calculation result of w for node i in the l-th layer, and $g(h_i^{(l)})$ is equal to $a_i^{(l)}$.

Let the error term $\delta_i^{(l)}$ of each neuron be:

$$\delta_i^{(l)} = \frac{\partial E(w, b)}{\partial a_i^{(l)}}g'(h_i^{(l)}) \tag{7}$$

Substituting Eq. (7) into Eq. (6), we get the following:

$$\nabla w_{ij}^{(l)} = \frac{\partial E(w, b)}{\partial w_{ij}^{(l)}} = \delta_i^{(l)}a_j^{(l-1)} \tag{8}$$

When the l-th layer is the output layer, then:

$$\frac{\partial E(w, b)}{\partial a_i^{(l)}} = \frac{\partial[\frac{1}{2}\sum_{i=1}^{n}(y_i - a_i^{(l)})^2]}{\partial a_i^{(l)}} = a_i^{(l)} - y_i \tag{9}$$

Substituting Eq. (9) into Eq. (7), we get the following:

$$\delta_i^{(l)} = \frac{\partial E(w, b)}{\partial a_i^{(l)}}g'(h_i^{(l)}) = (a_i^{(l)} - y_i)\bullet g'(h_i^{(l)}) \tag{10}$$

Substituting Eq. (10) into Eq. (8), the gradient update value of the weights in the output layer can be calculated.

When the l-th layer is a hidden layer, function $E(w, b)$ is a function

of $h^{(l+1)}$, where each element is a function of $x_i^{(l)}$. According to the following derivative formula of the compound function:

$$\frac{\partial E(w,b)}{\partial a_i^{(l)}} = \sum_{m=1}^{n+1} \frac{\partial E(w,b)}{\partial h_m^{(l+1)}} \frac{\partial h_m^{(l+1)}}{\partial a_i^{(l)}} = \sum_{m=1}^{n+1} \frac{\partial E(w,b)}{\partial a_j^{(l+1)}} \frac{\partial a_m^{(l+1)}}{\partial h_i^{(l)}} \frac{\partial h_m^{(l+1)}}{\partial a_i^{(l)}}$$
$$= \sum_{m=1}^{n+1} \frac{\partial E(w,b)}{\partial a_m^{(l+1)}} g'(h_m^{(l+1)}) w_{mi}^{(l+1)} = \sum_{m=1}^{n+1} \delta_m^{(l+1)} w_{mi}^{(l+1)}$$
$$= (w_{(,i)}^{(l+1)})^T \delta^{(l+1)} \tag{11}$$

We get the following formulation by substituting Eq. (11) into Eq. (7):

$$\delta_i^{(l)} = \frac{\partial E(W,b)}{\partial a_i^{(l)}} g'(h_i^{(l)}) = (w_{(,i)}^{(l+1)})^T \delta^{(l+1)} \bullet g'(h_i^{(l)}) \tag{12}$$

Similarly, the gradient update value of the weights in the hidden layer can be calculated by substituting Eq. (12) into Eq. (8).

Generally speaking, the training of a CNN model is mainly based on the BP algorithm. First, the loss function is defined to measure the difference between the outputs and the actual labels. Then, W and b are obtained by the stochastic gradient descent method (SGD) by minimizing the loss function.

3.2.3. Inference

With the constructed CNN model (See Fig. 3), we conduct a pixel-level classification study on 2D multi-spectral LiDAR feature images. Note that a large number of training samples are needed. Therefore, for each land cover class, we select 7680 pixel points, evenly and reasonably distributed from the 2D feature image to be classified. For each pixel, its neighborhood is extracted as a training image block input to the constructed CNN model. Then, with the features learned from the training samples, the entire image is to be predicted, and the multi-spectral LiDAR land-cover classification is performed through the constructed CNN model.

4. Experiments and analysis

To evaluate the performance of the constructed CNN based on the multi-spectral LiDAR data, a series of experiments were conducted. This section first described the hyper-parameter sensitivity analysis, and then performed comparative analysis with other traditional CNN models and our previous methods to demonstrate the feasibility of the optimal hyper-parameters. All experiments were trained and tested on the configured computer, with an Intel(R) Core(TM) i5-6500 CPU @ 3.20 GHz 3.19 GHz processor, a memory of 16.0G, and a graphics card for Intel(R) HD Graphics 530.

4.1. Hyper-parameters sensitivity analysis

Our CNN model involves six hyper-parameters in the model construction: dimension (m), input size (s_i), number of convolution kernels (n_c), convolution kernel size (s_c), pooling window size (s_p), and number of dense units (n_d). A control variable method was used to obtain their optimal values for the multi-spectral LiDAR classification task. Table 2 shows the initial values of the six hyper-parameters. In model implementation, our CNN model also involves two key hyper-parameters: data set dividing and learning rate. The data set was divided into the training set, which was used to train the model, the validation set, which was used to observe the performance of the model, and the test set, which was used to verify the effect of the model. The learning rate affects the speed at which our CNN model converges to a local minimum (that is, the speed at which the best precision is achieved), which plays an important role to model training time. Table 3 shows the default values of these two hyper-parameters in this study. To facilitate the implementation of our CNN model, some hyper-parameters (e.g., n_c and n_d) were set to the positive exponential power of two. A total of 50 epochs were run for each experiment.

Table 2
Hyper-parameters involved in the model establishment.

Hyper-parameters	Implemented Layers	Initial Values
Dimension (m)	Input Layer	2
Input size (s_i)	Input Layer	9
Number of convolution kernels (n_c)	Convolutional Layer	256
Convolution kernel size(s_c)	Convolutional Layer	3
Pooling window size(s_p)	Pooling Layer	2
Number of dense units(n_d)	the First Fully-Connected Layer	1024

Table 3
Default hyper-parameters set in the CNN model implementation.

Hyper-parameters	Implementation Phase	Initial Values
Data set dividing (train: validation: test)	Data Preparation	7:1:2
Learning rate	Training Process	0.001

Teo and Wu (2017) and Pan et al. (2018) have proven that the land-cover classification accuracies of multi-spectral LiDAR data are significantly better than those of single-wavelength LiDAR data. Therefore, we performed the constructed CNN model on the multi-spectral LiDAR data. The number of raster dataset bands, W, is four.

To achieve the optimal hyper-parameter values, the performance of our CNN model was evaluated by two metrics: overall accuracy (OA) and Kappa index (Congalton, 1991; Foody, 2010; Holtz, 2007). In this section, we designed six groups of experiments to investigate the sensitivity of our CNN model to the selection of m, s_i, n_c, s_c, s_p and n_d. Note that the model accuracies, obtained by the validation data set, were higher than the OA value of 95.0% and Kappa coefficient of 0.950, respectively. With the trained CNN model, the land-cover classification accuracies (i.e. OA and Kappa coefficient) were reported in the following experiments by using the test data sets.

4.1.1. Dimension, m

As discussed in Subsection 3.2, CNNs with different dimensions lead to different model structures, generating different convolutional operations in a local neighborhood. To determine the optimal dimension (m) of our CNN model, we varied m from 1 to 3 with an interval of 1, and maintained the remaining hyper-parameters $s_i = 9$, $s_c = 3$, $n_c = 256$, $s_p = 2$, and $n_d = 1024$. Table 4 shows the experimental results of the classification accuracies (i.e., OA and Kappa index) and the training time for the Titan multi-spectral LiDAR data, and Fig. 5 is the corresponding chart, where red circles represent the best Experiment 1–1-2 in Table 4.

As shown in Table 4 and Fig. 5, the CNN model with $m = 2$ (i.e., 2D CNN model) took the shortest training time, while the model with $m = 3$ (i.e., 3D CNN model) took the longest training time. The reason behind this phenomenon is that convolutional operations were greatly complex with different dimensions involved in CNN models. Among the three models, the 3D CNN model reached the highest computational complexity. Although the 1D CNN model is simplest in convolutional operations, its computational complexity was higher than that of the 2D CNN model. Correspondingly, the 3D CNN model achieved the best classification accuracies with an OA of 96.2% and a Kappa index of

Table 4
A comparison of model accuracy and training time for CNNs with different dimensions.

Experiment	m	OA (%)	Kappa	Time(mins)
1–1–1	1	91.6	0.900	149
1–1-2	**2**	**95.5**	**0.945**	**74**
1–1-3	3	96.2	0.955	1185

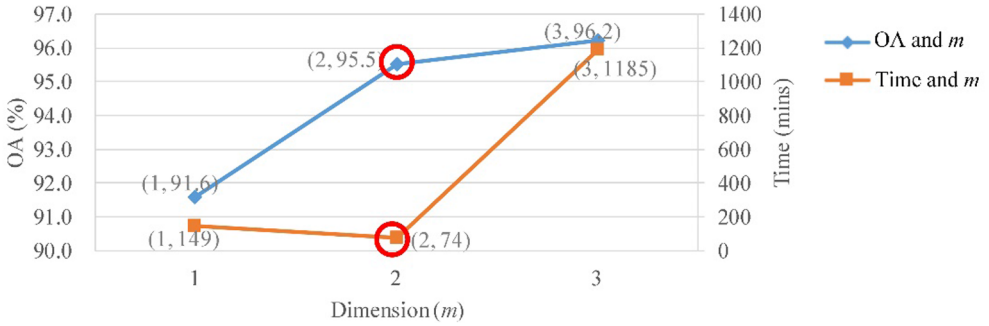

Fig. 5. A comparison of model accuracy and training time for CNNs with different dimensions.

Table 5
A comparison of the classification accuracies and training time with the different input size.

Experiment	s_i	OA (%)	Kappa	Time (mins)
1–2-1	3	92.1	0.905	18
1–2-2	5	93.4	0.920	39
1–2-3	7	94.7	0.936	59
1–2-4	**9**	**95.5**	**0.945**	**74**
1–2-5	11	95.3	0.943	86
1–2-6	13	92.5	0.910	101

Table 6
A comparison of model accuracy and training time for CNN models with different number of convolution kernels.

Experiment	n_c	OA (%)	Kappa	Time(mins)
1–3-1	64	94.8	0.938	24
1–3-2	128	95.2	0.943	50
1–3-3	**256**	**95.5**	**0.945**	**74**
1–3-4	512	95.4	0.944	138
1–3-5	1024	95.4	0.944	259

0.955, while the 1D CNN model achieved the worst classification accuracies with an OA of 91.6% and a Kappa index of 0.900. Although the OA accuracy and Kappa index of the 3D CNN model were slight higher than those of the 2D CNN model, the 3D CNN model took the training time 16 times that of the 2D CNN model. Therefore, to trade off the classification accuracy and the model training time, hyper-parameter m was set to 2 when our CNN model was performed on the Titan multi-spectral LiDAR data.

4.1.2. Input Size, s_i

The input size, s_i, determines the amount of the feature information used to input the CNN model. In this group of experiments, we varied s_i from 3 to 13 with an interval of 2, and maintained the remaining hyper-parameters $m = 2$, $n_c = 256$, $s_c = 3$, $s_p = 2$, and $n_d = 1024$. As we discussed above, the input size was defined as (s_i, s_i, W) when $m = 2$, that is, the 2D CNN model, where W was set to 4 (the number of raster dataset bands). Table 5 shows the Titan multi-spectral LiDAR land-cover classification accuracies (i.e., OA and Kappa) and the training time. Fig. 6 shows the corresponding chart, where red circles represent the best Experiment 1–2-4 in Table 5.

As shown in Table 5 and Fig. 6, the classification accuracies of the 2D CNN model smoothly increase when parameter s_i changed from 3 to 9. However, the classification accuracies decrease when parameter s_i varied from 9 to 13. The reason behind this phenomenon might be that a small value of s_i means a small neighborhood, leading to insufficient feature representation of land covers, thus reducing land-cover classification performance. On the contrary, a very large value of s_i indicates the neighborhood including rich and even redundant feature information, also leading to a degradation of the land-cover classification accuracies. In terms of time complexity, we found that the training time dramatically increased as s_i increased. Therefore, considering the classification accuracies (i.e., OA and Kappa) and the time consumed in training the model, the optimal configuration of s_i was 9.

4.1.3. Number of convolution Kernels, n_c

The number of convolution kernels (n_c) in the CNN model determines the number of feature images derived from the convolutional operation. To determine the optimal value of hyper-parameter n_c in our CNN model, we maintained $m = 2$, $s_i = 9$, $s_c = 3$, $s_p = 2$, $n_d = 1024$, and varied n_c from 64 to 1024 with a positive exponential power of two. Table 6 shows the Titan multi-spectral LiDAR land-cover classification accuracies (i.e., OA and Kappa) and the training time, and Fig. 7 is the corresponding chart, where red circles represent the best Experiment 1–3-3 in Table 6.

As shown in Table 6 and Fig. 7, the OA and Kappa values slight increased when n_c changed from 64 to 256, while the classification accuracies tended to be stable when n_c was larger than 256. This is because when n_c becomes smaller, that is, fewer filters are used in convolutional operations, features cannot be abundantly and comprehensively learned, resulting in under-fitting. However, if the value of n_c is set to be excessively large, the training samples are sparse in

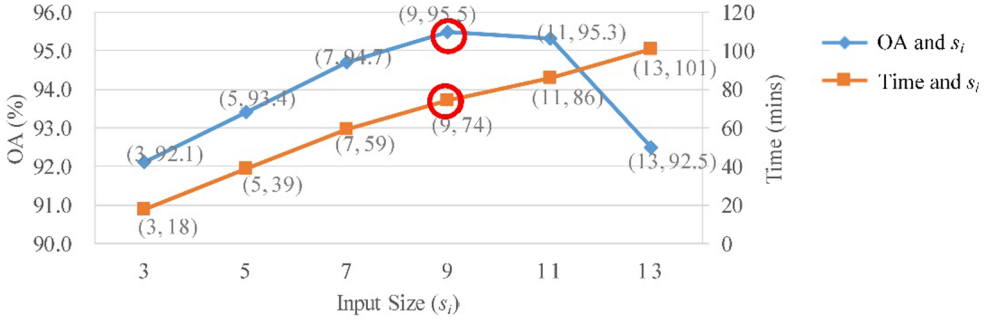

Fig. 6. A comparison of the classification accuracies and training time with the different input size.

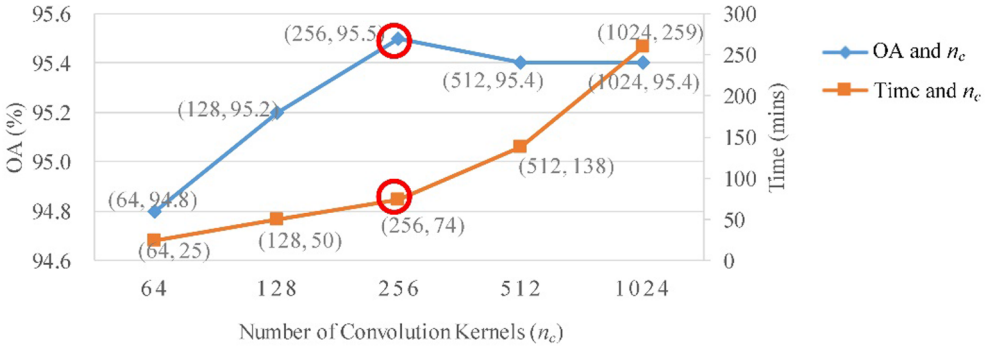

Fig. 7. A comparison of model accuracy and training time for CNN models with different number of convolution kernels.

Table 7
A comparison of the land-cover classification accuracy and training time complexity with different convolution kernel size.

Experiment	s_c	OA (%)	Kappa	Time(mins)
1–4-1	**3**	**95.5**	**0.945**	**74**
1–4-2	5	94.6	0.935	157
1–4-3	7	94.4	0.932	461
1–4-4	9	94.0	0.928	1286

Table 8
A comparison of the land-cover classification accuracy and training time complexity for CNN models with different pooling window size.

Experiment	s_p	OA (%)	Kappa	Time(mins)
1–5-1	**2**	**95.5**	**0.945**	**74**
1–5-2	3	94.1	0.930	51
1–5-3	4	93.6	0.923	34
1–5-4	5	93.5	0.922	23

convolutional operations, thus leading to over-fitting. In terms of time complexity, the training time dramatically increased as n_c increased. Therefore, to trade off the classification accuracies (i.e., OA and Kappa) and the time consumed in training the model, the optimal configuration of n_c was 256.

4.1.4. Convolution kernel Size, s_c

In convolutional operations, the convolution kernel size and number of convolution kernels (n_c) are equally important in feature image extraction. To determine the optimal convolution kernel size of our CNN model, we maintained hyper-parameters $m = 2$, $s_i = 9$, $n_c = 256$, $s_p = 2$, $n_d = 1024$, and varied s_c from 3 to 9 with an interval of 2. Table 7 shows the Titan multi-spectral LiDAR land-cover classification results with regard to classification accuracies (i.e., OA and Kappa) and training time, and Fig. 8 shows the corresponding chart, where red circles represent the best Experiment 1–4-1 in Table 7.

As shown in Table 7 and Fig. 8, with the increase of s_c, both the OA and Kappa values decreased accordingly. The reason is that a large amount of noise might be introduced when the value of s_c becomes excessively large. We found that s_c has a great influence on the training time of the CNN model. With the increase of s_c, the training time dramatically increased. Therefore, to balance the classification accuracies (i.e., OA and Kappa) and the time cost in model training, the optimal value of s_c is 3.

4.1.5. Pooling window Size, s_p

In this group of experiments, we maintained the hyper-parameters $m = 2$, $s_i = 9$, $n_c = 256$, $s_c = 3$, and $n_d = 1024$, and varied s_p from 2 to 5 with an interval of 1. Table 8 shows the experimental results of the Titan multi-spectral LiDAR classification accuracies (i.e., OA and Kappa) and the training time, and Fig. 9 shows the corresponding chart, where red circles represent the best Experiment 1–5-1 in Table 8.

As shown in Table 8 and Fig. 9, both the OA and Kappa values decreased when s_p increased from 2 to 5. Our CNN model achieved the best land-cover classification accuracies with an OA value of 95.5% and a Kappa value of 0.945 when $s_p = 2$. When the value of s_p is excessively large, the loss of the useful feature information might occur, resulting in under-fitting. In addition, with the increase of s_p, the model training time greatly decreased. Although the training time of our CNN model was about 74 min, when the pooling windows size was set to 2, this time cost was acceptable and the classification accuracies were satisfactory. Therefore, the value of s_p was set to 2.

4.1.6. Number of dense Units, n_d

The number of dense units (n_d) in our CNN model determines the number of neurons involved in rough land-cover classification. To obtain the optimal value of n_d of our CNN model, we maintained the hyper-parameters $m = 2$, $s_i = 9$, $n_c = 256$, $s_c = 3$, and $s_p = 2$, and varied n_d from 256 to 4096 with an interval of the positive exponential power of two. Table 9 shows the Titan multi-spectral LiDAR land-cover

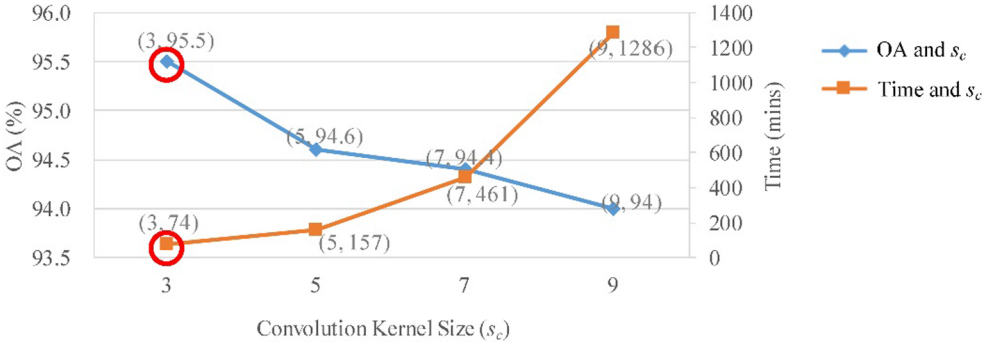

Fig. 8. A comparison of the land-cover classification accuracy and training time complexity with different convolution kernel size.

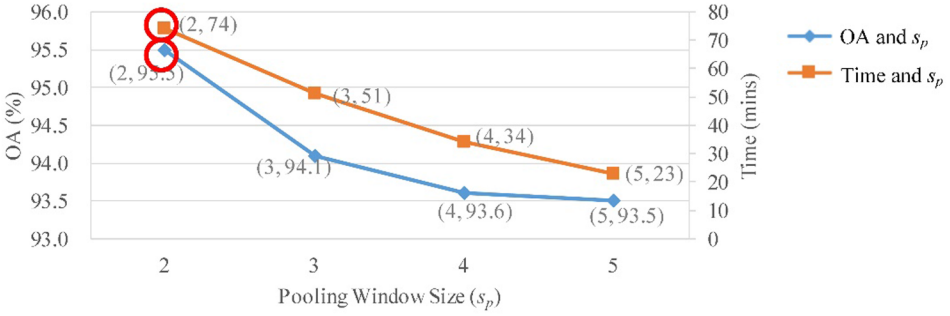

Fig. 9. A comparison of the land-cover classification accuracy and training time complexity for CNN models with different pooling window size.

Table 9
A comparison of the land-cover classification accuracies and training time complexity with different number of dense units.

Experiment	n_d	OA (%)	Kappa	Time(mins)
1–6-1	256	93.0	0.916	40
1–6-2	512	94.9	0.939	53
1–6-3	**1024**	**95.5**	**0.945**	**74**
1–6-4	2048	94.0	0.928	125
1–6-5	4096	94.3	0.931	243

classification accuracies (i.e., OA and Kappa) and the training time, and Fig. 10 is the corresponding chart, where red circles represent the best Experiment 1–6-3 in Table 9.

Table 9 and Fig. 10 demonstrate that both the OA and Kappa values dramatically increased when n_d varied from 256 to 1024. When n_d exceeded 1024, the land-cover classification accuracies of our CNN model greatly fluctuated with a downward trend, and then decreased. The purpose of the dense layer is to extract the correlation information between features through nonlinear changes and finally to obtain classification covers. Thus, if the dense layer contains fewer nodes, the extracted features might contain less useful information, resulting in under-fitting. However, if there are excessive nodes in the dense layer, the extracted features might contain redundant information, resulting in over-fitting. In terms of time complexity, the training time dramatically increased when the value of n_d increased. Therefore, to trade off the classification accuracies (i.e., OA and Kappa) and the training time, the optimal value of n_d was 1024.

4.2. Comparative experiments

After hyper-parameter sensitivity analysis, we obtained the optimal values of the six hyper-parameters for the multispectral LiDAR land-cover classification task, that is, $m = 2$, $s_i = 9$, $n_c = 256$, $s_c = 3$, $s_p = 2$, and $n_d = 1024$. Fig. 11 shows the classification results of our CNN model with the six optimal hyper-parameters. To evaluate the applicability of our CNN model with the analyzed optimal hyper-parameters, we compared it with three traditional CNN models (AlexNet, VGG16, and ResNet50) and our previous study (DBM, Random forest, and SVM).

These three traditional CNN models – AlexNet (Unnikrishnan et al., 2018), VGG16 (Zhang et al., 2018a, 2018b), and ResNet50 (Jiang et al., 2019) - have achieved great success in image classification (Akilan et al., 2018; Sharma et al., 2018). Because the above classic VGG16 model requires that each pixel of the input training sample must be interpolated to (48, 48) and must be in three wavelengths, the multispectral LiDAR data were subjected to nearest domain interpolation (Atamturktur et al., 2015) and principal component analysis (PCA). Finally, the s_i in these three classic CNN models (AlexNet, VGG16 and ResNet50) was set to 48, and W was set to 3. In the AlexNet model, the configurations of n_c and s_c in the first convolutional layer are 48 and 11, and the values of n_c and s_c in the second convolutional layer are 128 and 5, respectively. The two convolutional layers are followed by the pooling layer, where s_p was set to 3. Then, three identical convolutional layers are connected, where n_c and s_c were set to 192 and 3, respectively. In the VGG16 model, the convolutional layer and the pooling layer appear alternately, where s_p is 2, n_c in each convolutional layer is [4, 8, 16, 32], and s_c is fixed at 3. The ResNet50 model is deeper, with two basic blocks (identity block and convolution block), which is a stack of these two main blocks, and its parameters setting refers to the article Jiang et al. (2019). To ensure the fairness of the experimental comparative analysis, based on the same training data in these three classic CNN models (AlexNet, VGG16 and ResNet50), whose s_i was set to 48 and W was set to 3, and other optimal hyper-parameters ($m = 2$, $n_c = 256$, $s_c = 3$, $s_p = 2$, and $n_d = 1024$) described in Section 4.1, our constructed CNN model was used for comparative experiments of land-cover classification. These classical CNN models and our constructed CNN model were also iteratively trained by 50 epochs.

Fig. 11 shows the land-cover classification results by using the Titan multispectral LiDAR data, which is based our constructed CNN model. Visual inspections demonstrated that our CNN model with the optimal hyper-parameters is capable of labeling most land covers, especially land over water. However, our CNN model achieves a poor classification performance of distinguishing land cover soil from land cover building. To clearly demonstrate the comparative classification results, we selected two small areas (part A and part B) from the whole scene, as

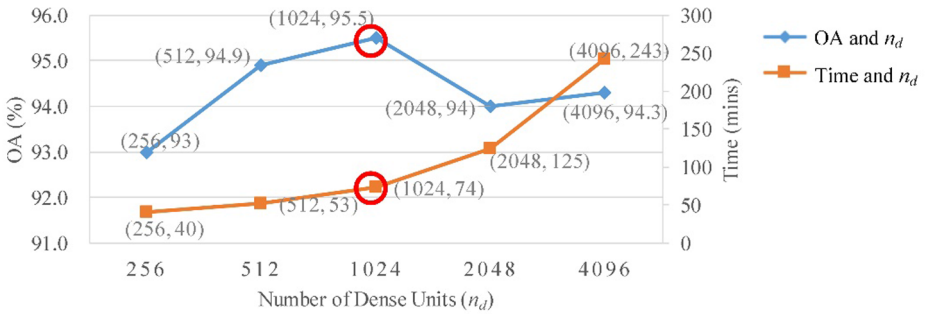

Fig. 10. A comparison of the land-cover classification accuracies and training time complexity with different number of dense units.

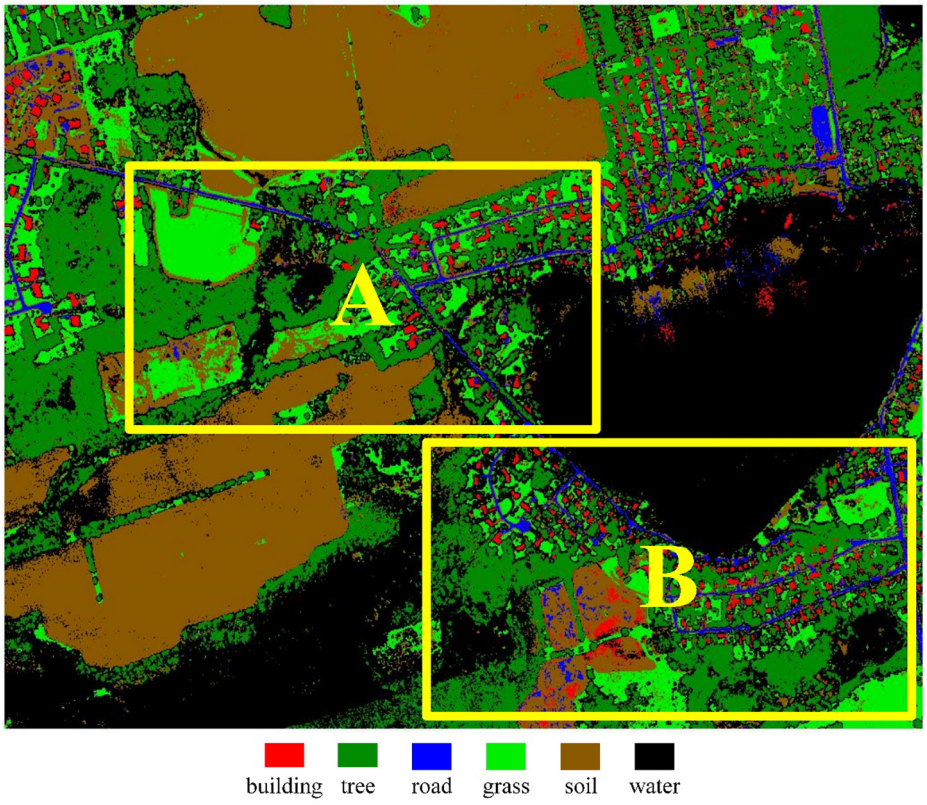

Fig. 11. Classification results of our constructed CNN.

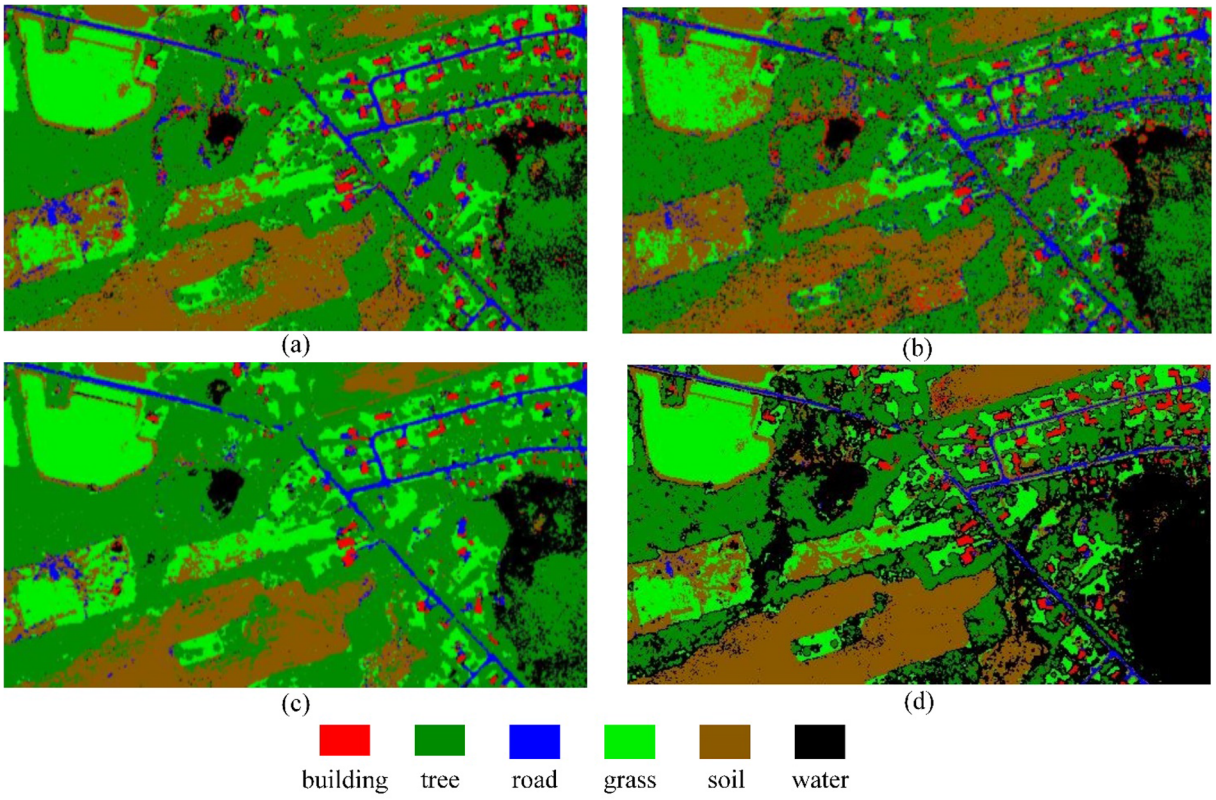

Fig. 12. Partial classification results (part A) of CNN models, (a) AlexNet, (b) VGG16, (c) ResNet50, (d) our constructed CNN.

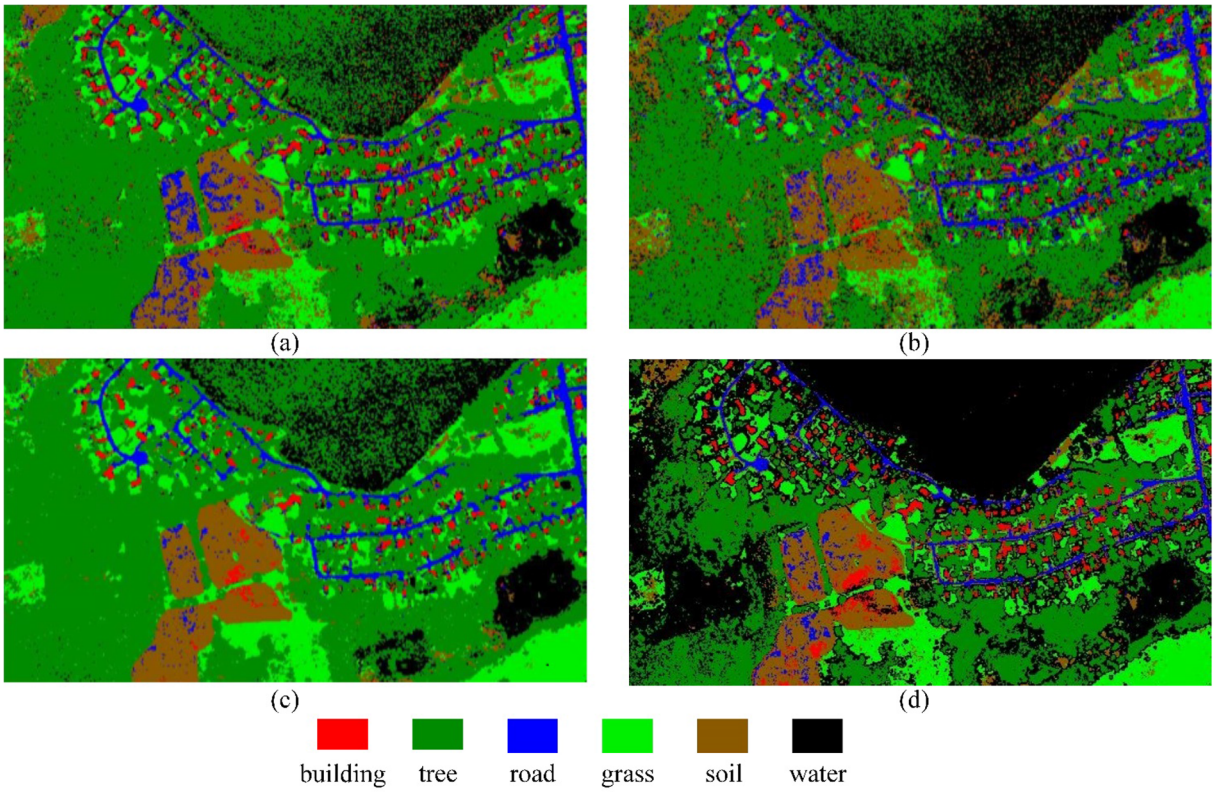

Fig. 13. Partial classification results (part B) of CNN models, (a) AlexNet, (b) VGG16, (c) ResNet50, (d) our constructed CNN.

Table 10
A comparison of five deep learning models in classification accuracies and time complexity.

Experiment	Model	OA (%)	Kappa	Time(mins)
2–1	AlexNet	91.8	0.902	661
2–2	VGG16	93.0	0.916	1006
2–3	ResNet50	94.2	0.930	1860
2–4	**our CNN model**	**94.9**	**0.939**	**523**
2–5	our previous method-DBM	89.3	0.866	34.675 × 60≈2080

shown in Fig. 12 (part A) and 13 (part B). We found that the AlexNet model was incapable of distinguishing soil and roads (see Figs. 12(a) and 13(a)); the VGG16 model achieved a poor classification performance on land covers soil, water, and tree (see Fig. 12(b) and 13(b)); the ResNet50 model failed to separate land cover grass from land cover soil, and accurately extract roads (see Figs. 12(c) and 13(c)).

In our previous study (Pan et al, 2019), we performed random forests (RF), support vector machine (SVM), and one of deep learning methods (Deep Boltzmann Machine, DBM) on the multi-spectral LiDAR data. Our previous studies focused on the land-cover classification feature analysis and demonstrated that the two-layer DBM model achieved the best classification accuracies with OA = 89.3% and Kappa = 0.866. Thus, the DBM model was selected for the comparison. In the two-layer DBM model, there are three important parameters: the neighborhood size (N_s = 9×9 pixels), and the numbers of hidden units N^1_{hU} = 300 and N^2_{hU} = 80 for the first and second layers, respectively.

To further quantitatively verify the classification performance of these comparative methods, we listed their classification accuracies with regard to the OA and Kappa values and the training time in minutes in Table 10, and Fig. 14 shows the corresponding chart, where red circles represent the best Experiment 2–4 in Table 10.

As shown in Table 10 and Fig. 14, in terms of time complexity, the DBM model (Experiment 2–5) took a longer time of 34.675 h (about 2080 min) to learn the features compared to all CNN models (Experiments 2–1, 2–2, 2–3, and 2–4). This is because the CNN models are featured by local connectivity and shared weights, which reduce the model complexity and the number of parameters, and finally simplify the model. Then, we compared different CNN models (Experiments 2–1, 2–2, 2–3, and 2–4), where the AlexNet model achieved the lowest land-

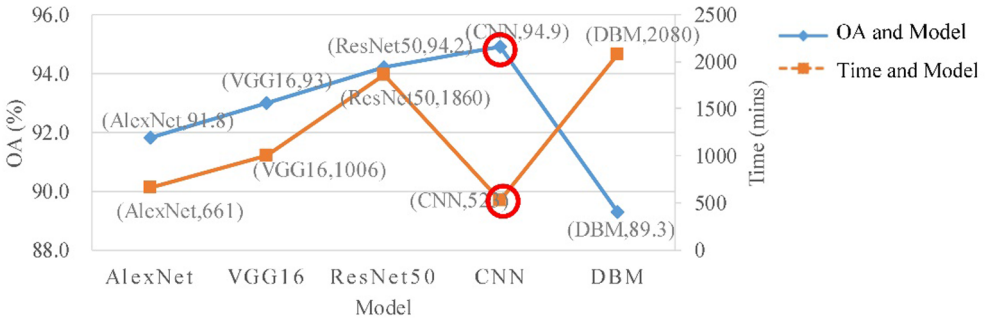

Fig. 14. A comparison of five deep learning models in classification accuracies and time complexity.

cover classification accuracy, while our CNN model with the optimal hyper-parameters achieved the highest classification accuracy with the OA and Kappa values improved by about 0.7%-3.1% and 0.009–0.037, respectively. With the most number of the functional layers, the ResNet50 model contains the most number of hyper-parameters and non-hyper-parameters, leading to the highest training time complexity. Comparatively, our CNN model has the least computational burden because the model contains only seven fundamental function layers, rather than the deeper and repetitive functional layers.

Qualitative and quantitative test results indicate that our CNN model achieved higher classification accuracies than the classical CNN models (i.e., AlexNet, VGG16 and ResNet50) and our previous study-DBM. The main reason is that, (1) our CNN model contains mainly fundamental function layers, which guarantees the classification accuracies by using the Titian multispectral LiDAR data, and (2) our CNN model is not piled up by a large number of implementation layers, which greatly shortens the model training time to improve the land-cover classification efficiency.

5. Conclusion

To classify the Titan multi-spectral LiDAR data into the six land-cover types (i.e., building, tree, road, grass, soil, and water), this paper presented a CNN model with optimized hyper-parameters. The main contributions of this study can be summarized as follows: (1) by stacking seven main functional layers and maintaining the optimized hyper-parameters, the time complexity of our CNN model is obviously reduced to 74 min, and (2) in our CNN model, six hyper-parameters, including dimension, input size, number of convolutional kernels, convolutional kernel size, pooling window size, and number of dense units, were investigated to obtain the optimal values by the control variable method; with the optimal values of the hyper-parameters, we obtained the satisfactory classification results with the OA of 95.5% and Kappa index of 0.945. Comparative experiments showed that, our CNN model achieved the classification accuracies with OA increased by 0.7%-3.1% and Kappa index increased by 0.009–0.037 better than the traditional CNN models (e.g., AlexNet, VGG16, and ResNet50), and our CNN model's time performance is significantly better than these traditional CNN models and the DBM. Therefore, our CNN model with the optimal hyper-parameters is practical and feasible for the multispectral LiDAR land-cover classification task. However, because the test Titan multispectral LiDAR data were provided without system parameters, trajectory data, and control points, their data preprocessing, such as fine strip registration and radiometric correction, cannot be done in our study. In future, the feasibility and applicability of our CNN model should be tested on different multispectral LiDAR data sets with a complete set of meta data, covering different rural and urban environments.

Declaration of Competing Interest

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

Acknowledgments

This research was supported by the National Natural Science Foundation of China under Grants 41671454, 61603146, and 41971414. We would like to thank the anonymous reviewers for their careful reading and thorough comments.

References

Akilan, T., Wu, Q.M., Zhang, H., 2018. Effect of fusing features from multiple DCNN architectures in image classification. IET Image Proc. 12 (7), 1102–1110.

Alagoz, B.B., Alisoy, H.Z., Koseoglu, M., Alagoz, S., 2016. Modeling and Analysis of Dielectric Materials by Using Gradient Descent Optimization Method. Int. J. Model. Simul. Scient. Comput. 08 (1).

Atamturktur, S., Egeberg, M.C., Hemez, F.M., Stevens, G., 2015. Defining coverage of an operational domain using a modified nearest-neighbor metric. Mech. Syst. Sig. Process. 349–361. https://doi.org/10.1016/j.ymssp.2014.05.040.

Bakuła, K., Kupidura, P., Jełowicki, Ł., 2016. Testing of Land Cover Classification from Multispectral Airborne Laser Scanning Data. ISPRS-Int. Arch. Photogram., Remote Sens. Spatial Inform. Sci. 161–169. https://doi.org/10.5194/isprs-archives-XLI-B7-161-2016.

Bergstra, J., Bengio, Y., 2012. Random search for hyper-parameter optimization. J. Mach. Learn. Res. 13 (1), 281–305.

Briese, C., Pfennigbauer, M., Lehner, H., Ullrich, A., Wagner, W., Pfeifer, N., 2012. Radiometric calibration of multi-wavelength airborne laser scanning data. ISPRS Annals Photogram., Remote Sens. Spatial Inform. Sci. 1, 335–340. https://doi.org/10.5194/isprsannals-I-7-335-2012.

Cai, M., Liu, J., 2016. Maxout neurons for deep convolutional and LSTM neural networks in speech recognition. Speech Commun. 77, 53–64.

Cao, J.L., Pang, Y.W., Li, X.L., Liang, J.K., 2017. Randomly translational activation inspired by the input distributions of ReLU. Neurocomputing 275, 859–868.

Chen, X.Q., Ye, C.M., Li, J., Chapman, M.A., 2018. Quantifying the carbon storage in urban trees using multispectral ALS data. IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens. 11 (9), 3358–3365.

Congalton, R.G., 1991. A review of assessing the accuracy of classifications of remotely sensed data. Remote Sens. Environ. 37 (1), 35–46.

Dahou, A., Elaziz, M.A., Zhou, J.W., Xiong, S.W., 2019. Arabic sentiment classification using convolutional neural network and differential evolution algorithm. Comput. Intell. Neurosci. 1–16.

Ekhtari, N., Glennie, C., Fernandez-Diaz, J.C., 2018. Classification of Airborne Multispectral Lidar Point Clouds for Land Cover Mapping. IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens. 11 (6), 2068–2078.

Fernandez-diaz, J.C., Carter, W.E., Glennie, C., Shrestha, R.L., Pan, Z.G., Ekhtari, N., Singhania, A., Hauser, D., Sartori, M., 2016. Capability assessment and performance metrics for the Titan multispectral mapping Lidar. Remote Sensing 8 (11), 936–970.

Foody, G.M., 2010. Assessing the Accuracy of Remotely Sensed Data: Principles and Practices. Photogram. Rec. 25 (130), 204–205.

Fukushima, K., 1988. Neocognitron: A hierarchical neural network capable of visual pattern recognition. Neural Networks 1 (2), 119–130.

Gao, H.B., Cheng, B., Wang, J.Q., Li, K.Q., Zhao, J.H., Li, D.Y., 2018. Object classification using CNN-based fusion of vision and LiDAR in autonomous vehicle environment. IEEE Trans. Ind. Inf. 14 (9), 4224–4231.

Guidici, D., Clark, M.L., 2017. One-dimensional convolutional neural network land-cover classification of multi-seasonal hyperspectral imagery in the San Francisco Bay area, California. Remote Sens. 9 (629), 1–25.

Guo, Y.M., Liu, Y., Oerlemans, A., Lao, S.Y., Wu, S., Lew, M.S., 2016. Deep learning for visual understanding. Neurocomputing 187, 27–48.

Hansen, M., Smith, M.L., Smith, L.N., Salter, M., Baxter, E.M., Farish, M., Grieve, B., 2018. Towards on-farm pig face recognition using convolutional neural networks. Comput. Ind. 145–152.

He, X., Wang, A.L., Ghamisi, P., Li, G.Y., Chen, Y.S., 2019. LiDAR data classification using spatial transformation and CNN. IEEE Geosci. Remote Sens. Lett. 16 (1), 125–129.

Holtz, T.S., 2007. Introductory Digital Image Processing: A Remote Sensing Perspective. Third Edition. Environ. Eng. Geosci. 13 (1), 89–90.

Hopkinson, C., Chasmer, L., Gynan, C., Mahonry, C., Sitar, M., 2016. Multisensor and multispectral LiDAR characterization and classification of a forest environment. Canad. J. Remote Sens. 42 (5), 501–520.

Jaswal, D., Sowmya, Soman, K. P., 2014. Image classification using convolutional neural networks. Int. J. Scient. Eng. Res., 5(6), 1661-1668.

Jiang, Y.N., Li, Y., Zhang, H.K., 2019. Hyperspectral image classification based on 3-D separable ResNet and transfer learning. IEEE Geosci. Remote Sens. Lett. https://doi.org/10.1109/lgrs.2019.2913011.

Kang, Z.Z., Yang, J.T., 2018. A probabilistic graphical model for the classification of mobile LiDAR point clouds. ISPRS J. Photogramm. Remote Sens. 143, 108–123.

Kumar, B., Lohani, B., Pandey, G., 2019a. Development of deep learning architecture for automatic classification of outdoor mobile LiDAR data. Int. J. Remote Sens. 40 (9), 3543–3554.

Kumar, B., Pandey, G., Lohani, B., Misra, S.C., 2019b. A multi-faceted CNN architecture for automatic classification of mobile LiDAR data and an algorithm to reproduce point cloud samples for enhanced training. ISPRS J. Photogramm. Remote Sens. 147, 80–89.

Lecun, Y., Bottou, L., Bengio, Y., Haffner, P., 1998. Gradient-based learning applied to document recognition. Proc. IEEE 86 (11), 2278–2323.

Li, J.J., Zhao, X., Li, Y.S., Du, Q., Xi, B., Hu, J., 2018a. Classification of hyperspectral imagery using a new fully convolutional neural network. IEEE Geosci. Remote Sens. Lett. 15 (2), 292–296.

Li, Q.D., Yang, X.S., Yang, F.Y., 2006. Hyperchaos in a simple CNN. Int. J. Bifurcation Chaos 16 (08), 2453–2457.

Li, W., Chen, C., Zhang, M.M., Li, H.C., Du, Q., 2019. Data augmentation for hyperspectral image classification with deep CNN. IEEE Geosci. Remote Sens. Lett. 16 (4), 593–597.

Li, Y.H., Wang, N.Y., Shi, J.P., Hou, X.D., Liu, J.Y., 2018b. Adaptive batch normalization for practical domain adaptation. Pattern Recogn. 80, 109–117.

Liu, W. P., Sun, J., Li, W. Y., Hu, T., Wang, P., 2019. Deep learning on point clouds and its application: a survey. Sensors, 19(19), 4188, DOI.10.3390/s19194188.

Lu, C., Yang, X.M., Wang, Z.H., Li, Z., 2018. Using multi-level fusion of local features for land-use scene classification with high spatial resolution images in urban coastal zones. Int. J. Appl. Earth Obs. Geoinf. 70, 1–12.

Manyala, A., Cholakkal, H., Anand, V., Kanhangad, V., Rajan, D., 2019. CNN-based gender classification in near-infrared periocular images. Pattern Anal. Appl. 22 (4), 1–12.
Matikainen, L., Karila, K., Hyyppä, J., Litkey, P., Puttonen, E., Ahokas, E., 2017. Object-based analysis of multispectral airborne laser scanner data for land cover classification and map updating. ISPRS J. Photogramm. Remote Sens. 128, 298–313.
Miller, C.I., Thomas, J.J., Kim, A.M., Metcalf, J.P., Olsen, R.S., 2016. Application of image classification techniques to multispectral lidar point cloud data. Proc. SPIE 9832.
Morsy, S., Shaker, A., El-Rabbany, A., 2017. Multispectral LiDAR data for land cover classification of urban areas. Sensors 17 (5), 958–979.
Pan, S.Y., Guan, H.Y., 2018. Object classification using airborne multispectral LiDAR data. Acta Ueodaetica et Cartographica Sinica 47 (2), 198–207.
Pan, S.Y., Guan, H.Y., Yu, Y.T., Li, J., Peng, D.F., 2019. A comparative land-cover classification feature study of learning algorithms: DBM, PCA, and RF using multispectral LiDAR data. IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens. 12 (4), 1314–1326.
Rangel, J.C., Martinezgomez, J., Romerogonzalez, C., Garciavarea, I., Cazorla, M., 2018. Semi-supervised 3D object recognition through CNN labeling. Appl. Soft Comput. 65, 603–613.
Scaioni, M., Hofle, B., Kersting, A.P., Barazzetti, L., Previtali, M., Wujanz, D., 2018. Methods from information extraction from LiDAR intensity data and multispectral LiDAR technology. ISPRS Archives XLII-3 1503–1510, DOI.10.5194/isprs-archives-XLII-3-1503-2018.
Shaker, A., Yan, W.Y., LaRocque, P.E., 2019. Automatic land-water classification using multispectral airborne LiDAR data for near-shore and river environments. ISPRS J. Photogramm. Remote Sens. 152, 94–108.
Sharma, N., Jain, V., Mishra, A., 2018. An analysis of convolutional neural networks for image classification. Proc. Comput. Sci. 132, 377–384.
Soon, F.C., Khaw, H.Y., Chuah, J.H., Kanesan, J., 2018. Hyper-parameters optimization of deep CNN architecture for vehicle logo recognition. IET Intel. Transport Syst. 12 (8), 939–946.
Srivastava, N., Hinton, G.E., Krizhevsky, A., Sutskever, L., Salakhutdinov, R., 2014. Dropout: a simple way to prevent neural networks from overfitting. J. Mach. Learn. Res. 15 (1), 1929–1958.
Teo, T., Wu, H., 2017. Analysis of land cover classification using multi-wavelength LiDAR system. Appl. Sci. 7 (7), 663–683.
Tivive, F.H., Bouzerdoum, A., 2005. Efficient training algorithms for a class of shunting inhibitory convolutional neural networks. IEEE Trans. Neural Networks 16 (3), 541–556.
Unnikrishnan, A., Sowmya, P, S. K., 2018. Deep AlexNet with reduced number of trainable parameters for satellite image classification. Proc. Comput. Sci., pp. 931–938.
Vamplew, P., Dazeley, R., Foale, C., 2017. Softmax exploration strategies for multi-objective reinforcement learning. Neurocomputing 263, 74–86.
Wang, A.L., Wang, Y., Chen, Y.S., 2019a. Hyperspectral image classification based on convolutional neural network and random forest. Remote Sens. Lett. 10 (11), 1086–1094.
Wang, C.S., Shu, Q.Q., Wang, X.Y., Guo, B., Liu, P., Li, Q.Q., 2019b. A random forest classifier based on pixel comparison features for urban LiDAR data. ISPRS J. Photogramm. Remote Sens. 148, 75–86.
Wichmann, V., Bremer, M., Lindenberger, J., Rutzinger, M., Georges, C., Petrinimonteferri, F., 2015. Evaluating the potential of multispectral airborne LiDAR for topographic mapping and land cover classification. ISPRS Annals II-3/W5 113–119, DOI:10.5194/isprsannals-II-3-W5-113-2015.
Xu, X.D., Li, W., Ran, Q., Du, Q., Gao, L.R., Zhang, B., 2018. Multisource remote sensing data classification based on convolutional neural network. IEEE Trans. Geosci. Remote Sens. 56 (2), 937–949.
Yan, W.Y., Shaker, A., Larocque, P.E., 2018. Water mapping using multispectral airborne LiDAR data. ISPRS Archives XLII-3, 2047–2052. https://doi.org/10.5194/isprs-archives-XLII-3-2047-2018.
Zhang, C., Sargent, I., Pan, X., LI, H. P., Gardiner, A., Hare, J., Atkinson, P. M., 2018. An object-based convolutional neural network (OCNN) for urban land use classification. Remote Sens. Environ., vol. 261, pp. 57-70.
Zhang, R., Li, G.Y., Li, M.L., Wang, L., 2018b. Fusion of images and point clouds for the semantic segmentation of large-scale 3D scenes based on deep learning. ISPRS J. Photogramm. Remote Sens. 143, 85–96.
Zhu, Y.H., Gao, X., Zhang, W.L., Liu, S.K., Zhang, Y.Y., 2018. A bi-directional LSTM-CNN model with attention for aspect-level text classification. Future Internet 10, 116. https://doi.org/10.3390/fi10120116.