GOTTFRIED WILHELM LEIBNIZ UNIVERSITÄT HANNOVER

INSTITUT FÜR INFORMATIONSVERARBEITUNG

INSTITUT FÜR PHOTOGRAMMETRIE UND GEOINFORMATION

Master Thesis

Investigations on Improving the Classification of Land Cover Based on Convolutional Neural Networks

Deyu Zhang

1st Supervisors (TNT):

Prof. Dr.-Ing. J. Ostermann, M.Sc. Felix Kuhnke

2nd Supervisors (IPI):

apl. Prof. Dr. Techn. F. Rottensteiner, M.Sc. Chun Yang

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbstständig angefertigt und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe. Ich stimme der Verwertung meiner Arbeit durch das Institut in den folgenden Punkten zu:

- Aufnahme der gedruckten und der elektronischen Fassung der Arbeit in die Institutsbibliothek,
- Vervielfältigung der gesamten Arbeit oder von Auszügen für Lehrzwecke und
- Wiedergabe der Arbeit durch Bild- und Tonträger

Hannover, den 22.01.19

Topic of the M.Sc. Thesis

Investigations on improving classification of land cover based on Convolutional Neural Networks

Classification of land cover is a standard task in remote sensing, in which each image pixel is assigned a class label indicating the physical material of the object surface (e.g. *grass*, *asphalt*). This task is challenging due to the heterogeneous appearance and high intra-class variance of objects. Recent work trying to solve this task has focused on convolutional neural networks (CNN), delivering considerably better results than traditional classifiers such as Random Forests, whereas such classifiers use hand-crafted features as input, CNN provide a framework in which these features (and, thus, a representation of the image) can be learned from training data, which explains much of the success of CNN in classification. Originally, CNN were designed to predict a single class label per image. In the meantime, they have also been expanded to the task of land cover classification, where a class label is to be predicted for each pixel. This is achieved by an encoder-decoder structure, where in the encoder part, the resolution is continuously reduced (as in common CNN), whereas the encoder part upsamples the resultant class scores to obtain per-pixel predictions. Existing methods mainly suffer from a poor representation of object boundaries due to the reduction of resolution in the encoder part of the CNN.

The goal of this master thesis is to investigate methods for improving the classification of land cover by an existing CNN encoder-decoder architecture. Generally speaking, there are two investigation variants. **The first one** is hard-negative mining. Mr. Zhang has to find out which pixels are hard to be classified. It can be done by setting a threshold of probabilistic score, the pixels whose correct class scores are above the threshold are considered as easy ones, otherwise are hard ones. Then investigations on how to improve the classification of them have to be conducted. It could be 1) retaining only the hard pixels in the same CNN model, by setting a mask in the groundtruth labels; 2) using a patch-based CNN model which requires patches as input, where the patches are extracted out by centering the hard pixels. For these both methods, data augmentation should be applied if needed. Meanwhile, Mr. Zhang needs to investigate the relationship of threshold and the overall performance. **The second one** is adding more prior knowledge to the actual CNN model. In this case, the prior object boundary information needs to be firstly extracted out from the groundtruth labels. Afterwards, Mr. Zhang needs to incorporate this knowledge into the model by modifying the structure. For this variant, an investigation of the width of boundary should be taken, in order to find out the relationship between performance and the boundary.

This master thesis is a cooperation of the Institute of Information processing (TNT) and the Institute of Photogrammetry and GeoInformation (IPI). The required methods shall be implemented in Python using the Tensorflow development environment for CNN. In this context, Mr. Zhang should use an existing encoder-decoder architecture for which both the source code and a pre-trained model are available at IPI. The developed methods shall be evaluated using test datasets for which a reference is available. Mr. Zhang shall use the Vaihingen benchmark dataset from the ISPRS semantic labelling challenge.

Contents

1. Introduction

Land cover is the description of the physical material on the earth's surface. For instance, land covers can be divided into *grass, asphalt, trees, bare ground,* etc. There are two most commonly used approaches to collect geoinformation for land cover: field survey and analysis of remote sensing data.

Classification of land cover is a standard task in remote sensing, in which each image pixel is assigned a class label indicating the physical material of the object surface (e.g. *grass, asphalt*) (Yang et al., 2018). This task is called land cover classification in photogrammetry and remote sensing and also termed as semantic segmentation in computer vision, which is implemented to recognize the objects and also find out the locations in the image.

Land cover classification is challenging due to the heterogeneous appearance and high intra-class variance of objects (Paisitkriangkrai et al., 2016), where the heterogeneous appearance describes the diverseness of object appearances in land cover. The intra-class variance can be also termed as within class variance, which describes the degree of dispersion in a segment, and in land cover classification it is reflected as how far a set of pixels values are spread out from their average value.

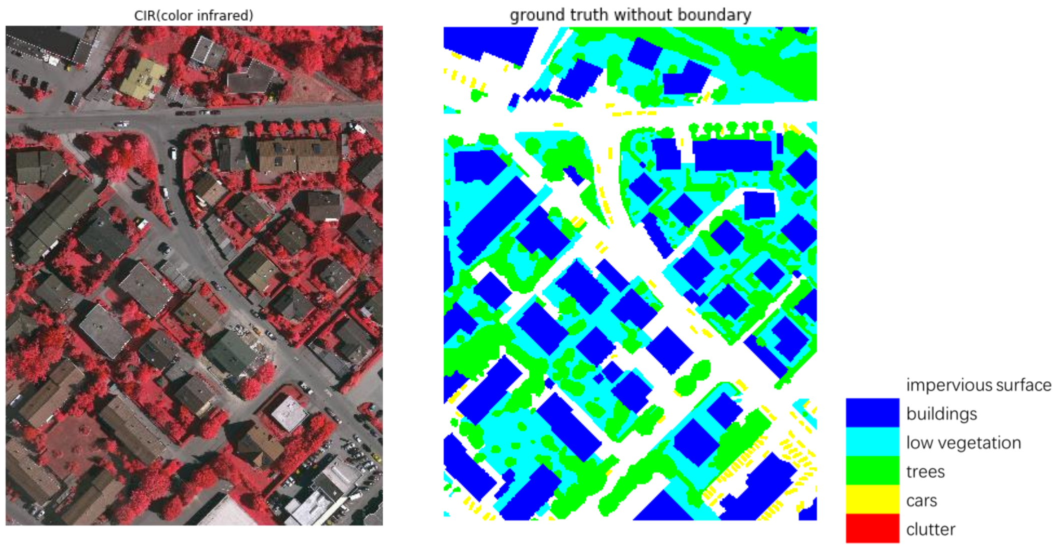

Figure 1-1 An example of land cover classification. On the left side is a color infrared photo and on the right is its land cover classification with six classes.

Recent work trying to solve this task has focused on convolutional neural networks, delivering considerably better results than traditional classifiers such as Random Forests, whereas such classifiers use hand-crafted features as input. Convolutional neural networks provide a framework in which these features (the representation of the image) can be learned from training data, which explains much of the success of convolutional neural network in classification.

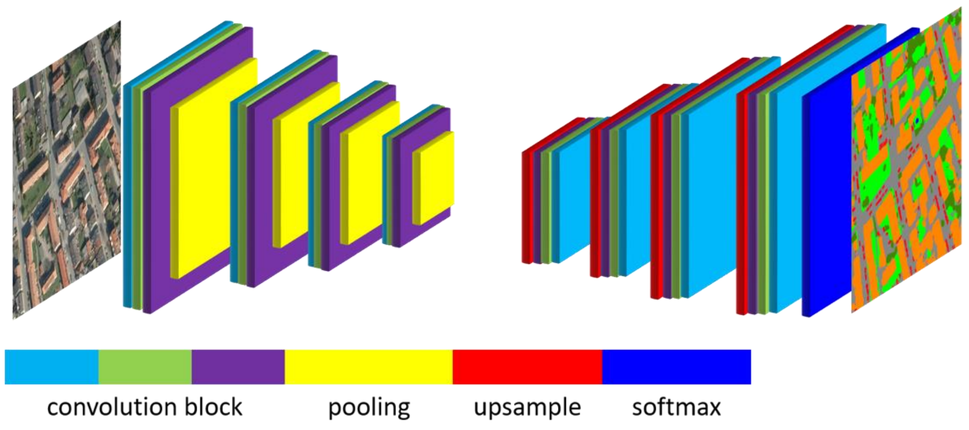

Figure 1-2 An example of encoder-decoder structure

Originally, convolutional neural networks were designed to predict a single class label per image (Krizhevsky et al., 2012). In the meantime, they have also been expanded to the task of land cover classification, where a class label is to be predicted for each pixel (Audebert et al., 2016). This can be achieved by an encoder-decoder structure, where in the encoder part, the spatial resolution is continuously reduced (as in common convolutional neural networks), whereas the decoder part upsamples the resultant feature maps to obtain per-pixel predictions (Yang et al., 2018), as shown in Figure 1-2, or a patch-based model structure. A patch-based model achieves per-pixel classification by packing a pixel and its surrounding pixels together and using the predicted label of this local "image" as its center pixel label. For instance, an image is cropped into many small parts, i.e. patches. The label of the center pixel of a patch is used as the class of this patch and the convolutional neural network converts the context information, which is provided by the rest pixels in this patch, into a feature vector. By using a sliding window approach, each pixel can be made as the center of such a patch (Längkvist et al., 2016). However, this implementation produces unnecessary computations due to patches overlapping. In order to reduce the impact of overlapping, bilinear interpolation is also used in patch-based model, which predicts every n^{th} pixel in the image, to convert sparse predictions to dense predictions (Paisitkriangkrai et al., 2016).

1.1 Motivation

With the development of photogrammetry and remote sensing, deep learning, especially convolutional neural network plays a more and more important role in land cover classification. Convolutional neural network is very suitable for land cover classification tasks due to the following reasons. Firstly, due to the rapid growth of the performance of graphics processing unit (GPU), the training and testing efficiency of the convolutional neural network has been greatly improved and not only that, its accuracy has also achieved a significant progress. Secondly, considering the size of aerial and satellite imagery is generally very large, the convolutional neural network can extract features automatically from the input data, while the traditional methods applied in land cover classification use hand-crafted features, which can be time-consuming, and the requirements vary from task to task. Thirdly, different from ground photogrammetry, aerial and satellite imagery can be captured in any azimuth. This means that the methods applied in land cover classification must be robust against rotation (Paisitkriangkrai et al., 2016). By applying augmentation of training dataset with different angles and mirroring, convolutional neural network can effectively manage to resist rotation issue.

An encoder-decoder structure was implemented to achieve land cover classification, i.e. semantic segmentation. Although the implemented structure made a great progress, there are still some challenges and imperfections, which need to be concerned and overcome. Existing methods mainly suffer from a poor representation of object boundaries due to the reduction of resolution in the encoder part of the convolutional neural network, as well as heterogeneous appearance and high intra-class variance of objects, as shown in Figure 1-3.

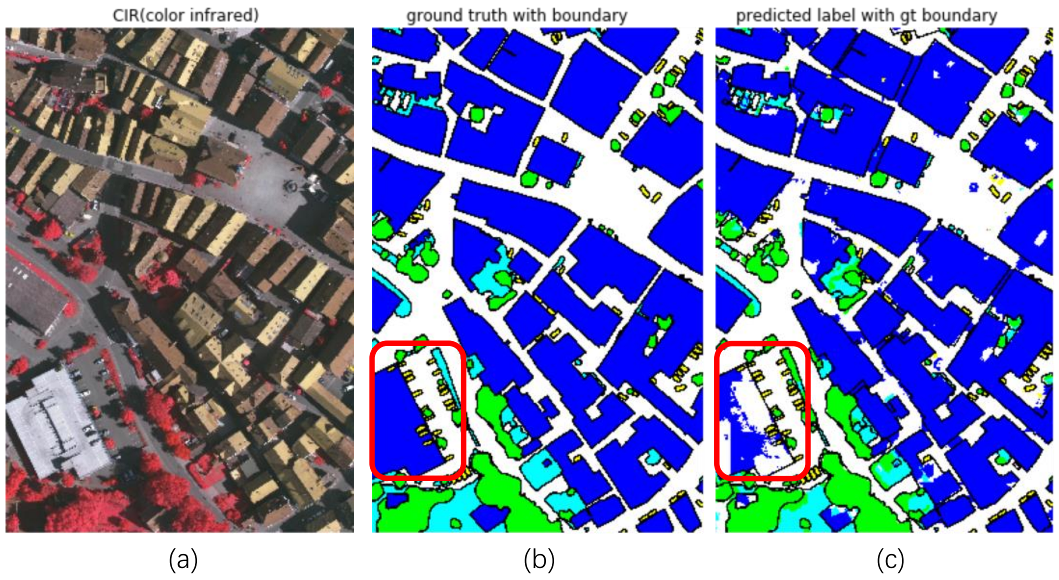

Figure 1-3 (a) is an example of Land cover classification task; (b) is the reference with boundary; (c) is the corresponding labels predicted by the model and the boundary generated from reference. Vaihingen Dataset of Benchmark Test of ISPRS.

One of the most conspicuous mistakes is the building at lower left corner (marked with red rectangle), colored as blue in (b), which indicates its class is *Building*, is classified incorrectly as *Impervious Surface*, as shown in (c). This mistake is caused by the combined impact of low inter-class variance and high intra-class variance of objects.

Compared to other buildings in the dataset, the features of this building are closer to the features of impervious surface. One method to solve this problem is to combine the existing model with another convolutional neural network architecture, in order to make the intra-class features more obvious. This purpose is achieved by merging the extracted features of two different models to obtain a stronger generalization. Another problem is the classification at boundaries. The reduction of resolution in the encoder parts is not beneficial for segmentation where boundary delineation is vital (Badrinarayanan et al., 2017). As shown in Figure 1-4, the mistakes at boundaries are not to be neglected. The model may achieve better performance at boundaries, when the boundary information is captured and stored before subsampling.

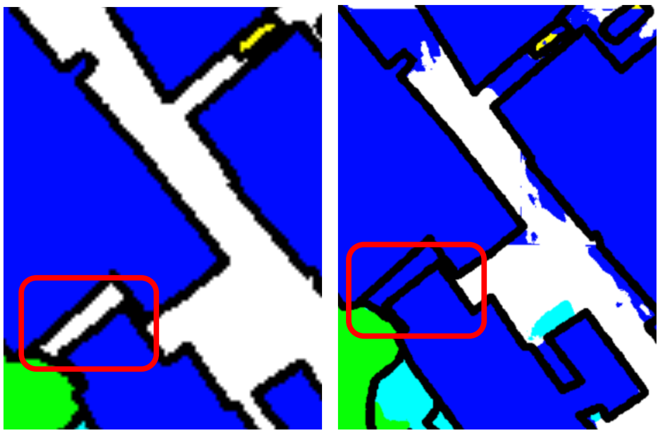
Figure 1-4 One local area of predicted labels with reference boundary, where the black line represents the boundary of objects

1.2 Goal of this thesis

The goal of this master thesis is to investigate methods for improving the classification of land cover based on an existing convolutional neural network encoder-decoder architecture. Generally speaking, there are two investigation variants. The first one is hard-negative mining. Pixels that are hard to be classified, need to be found out. It can be done by setting a threshold of probabilistic score, the pixels whose correct class scores are above the threshold are considered as easy ones, otherwise are hard ones. Then investigations on how to improve the classification of them have to be conducted. It could be:

- Retraining only the hard pixels in the same convolutional neural network model
- Using a patch-based convolutional neural network model which requires patches as input, where the patches are extracted out by centering the hard pixels

For these both methods, data augmentation should be applied if needed. The second one is adding more prior knowledge to the actual convolutional neural network model.

In this case, the prior object boundary information needs to be firstly extracted out from the reference. Afterwards, this knowledge needs to be incorporated into the model by modifying the structure, whereas a research on modification must be taken. Besides, an investigation of the width of boundary should be taken, in order to find out the relationship between performance and the boundary.

The required methods shall be implemented in Python using the TensorFlow development environment (Abadi et al., 2016) for convolutional neural network.

1.3 Structure of this thesis

The structure of this thesis is as follows:

- In Chapter 2, the related work involving land cover classification based on traditional methods and convolutional neural networks will be discussed.
- In Chapter 3, the theoretical background about the convolutional neural network will be introduced to help understand our currently framework.
- In Chapter 4, the proposed methodology will be introduced.
- In Chapter 5, the training procedure of different models will be introduced as well as the results will be analyzed.
- In Chapter 6, we will conclude this thesis and state our prospects for the future.

2. Related work

In the long history of human development, knowing the geoinformations surround us is always very essential. How to identify the surface elements on earth (Land cover) is a highly concerned problem in geoinformation analyzing.

2.1 Traditional Methods

Land cover refers to description of the physical material on the earth's surface. It is undoubtedly a high value information supporting various environmental science and land management applications at global, regional and local scales (Foley et al., 2005; Sharma et al., 2018). Considering the significance of land cover, e.g. ecosystem services, agricultural monitoring and etc. (Hietel et al., 2004; Burkhard et al., 2012; Guidici & Clark, 2017), the remote sensing technology has been used for generating accurate land cover datasets at various scales (Bartholome & Belward, 2005; Gong et al., 2013; Jin et al., 2013). The remote sensing data has ideal spectral, spatial, radiometric and temporal characteristics (Sharma et al., 2018), but the land cover classification depends on not only the imagery appropriateness, but also the choice of classification methods (Lu & Weng, 2017).

Many classification methods have been proposed using remote sensing data, such as unsupervised algorithms, e.g. K-means, parametric supervised algorithms, e.g. maximum likelihood, and machine learning algorithms, e.g. random forest and neural networks (Li et al., 2014; Zhu et al., 2017). The unsupervised algorithms could achieve very potential performance but require extra knowledge to the task-specific study area, whereas supervised algorithms could produce high classification accuracies with sufficient training and proper settings (Li et al., 2014). Including context into the classification process by using context features (Hermosilla et al., 2012) and Markov or Conditional Random Fields (Albert et al., 2017) has improved the accuracy of land cover classification, but the contextual models requires a large amount of training data and the choice of optimizer is a challenge (Albert et al., 2017). The machine learning algorithms have been mostly applied in terrain classification, i.e. predicting a label for an overhead image (Paisitkriangkrai et al., 2016) due to limitations of computational efficiency. In last decades, the development of hardware made it possible to process a great number of high-resolution aerial imagery. Recent advances in image classification is attributed to the convolutional neural network, where it outperforms traditional classifiers using hand-crafted features in many areas (Krizhevsky et al., 2012; Girshick et al., 2014; Razavian et al., 2014) This improvement is also adapted in land cover classification.

2.2 Classification of Land Cover Based on Convolutional Neural Networks

The convolutional neural network is inspired by the nature of the mammalian visual cortex and showed a great success in vision applications. It can learn visual patterns directly from raw image pixels. The convolutional neural network uses a combination of a convolutional layer, a non-linear mapping and a pooling layer down-sampling and abstracting signals (LeCun et al., 1998) so that the spatial correlation present in the natural images is well exploited (Paisitkriangkrai et al., 2016). High-level features are extracted and learned from the training data. The deep convolutional neural network consists of a series of these combinations and is followed by several fully connected layers, which are used for integrating local features to global features (Krizhevsky et al., 2012).

Originally, the convolutional neural network is designed to predict a single class label per image (Krizhevsky et al., 2012). The convolutional layer and the pooling layer would reduce the dimension of the input and the elements of the output cannot one-to-one correspond to the pixels in the image. In addition, the fully connected layer would output the vectors with a fixed size.

2.2.1 Patch-based Land Cover Classification

Land cover classification is a pixel-level classification, which also called semantic segmentation in computer vision. To implement semantic segmentation in the convolutional neural network, one convenient method is the patch-based convolutional neural network (Paisitkriangkrai et al., 2016), where each pixel was separately classified into classes using a patch of image around it, which is labeled with its central pixel. In inference, the convolutional neural network naturally predicts a label for the patch, but this label is only assigned to the central pixel and the surrounding pixels are regarded as unpredicted. Längkvist et al. (2016) apply this procedure in a sliding window approach, making each pixel in the centre of such a patch. However, using the patch-based classification needs to generate a patch for each pixel and leads to two problems: computing consuming and insufficient use of information, i.e. only adjacent information of the pixel is considered. Computation consuming is due to overlapping of the patches. To overcome the problem of time consuming, Paisitkriangkrai et al. (2016) implement the simple linear interpolation in the patch-based classification. This method significantly alleviates the problem of time-consuming during evaluation, but has properties of the low pass filter, which can damage high frequency components, so it may blur the object boundaries to some extent.

2.2.2 Fully Convolutional Networks and Encoder-decoder architecture

Another structure for semantic segmentation is called fully convolutional networks (Long et al., 2015). This model transforms convolved feature maps into classes of each pixel. The network can accept input images of any size and produce output of the same size, one-to-one correspondence between the input image and the output. This network supports end-to-end, pixel-to-pixel training. To obtain dense predictions, Long et al. (2015) researched three methods: shift-and stitch, filter rarefaction and deconvolution. Using deconvolution to upsample is recommended.

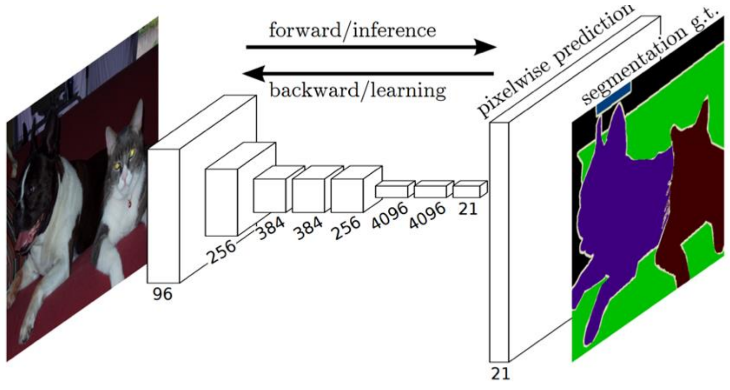

Figure 2-1 A simple fully convolutional network (Long et al., 2015), where the idea of solving the problem of image resolution reduction caused by convolution and pooling is upsampling

Noh et al. (2015) proposed the deconvolutional neural network where the encoder-decoder architecture is implemented. The encoder consists of series of building blocks of convolutional layers, pooling layers and ReLU layers like traditional convolutional neural networks, and the spatial dimension decreases gradually because of pooling layers, whereas the decoder needs to restore the spatial dimension and details from low-resolution signal. The decoder has symmetrical structure of the encoder, while the upsampling layer is used as the beginning of each block to increase the spatial dimension. Normally the shortcut connections between encoder and decoder are implemented to help decoder restore the losing location information.

SegNet (Badrinarayanan et al., 2015) is also based on the encoder-decoder architecture but the decoder uses pooling indices computed in the max-pooling step of the corresponding encoder to perform non-linear upsampling. This eliminates the need for learning to upsample.

These architectures for semantic segmentation are also implemented in land cover classification, achieving promising results. Marmanis et al. (2018) proposed a fully connected convolutional neural network for land cover classification with different resolution inputs.A fully convolutional network without pooling is proposed to deal with the dimension reduction (Sherrah, 2016) and achieved an increase of 2% in accuracy in the ISPRS labelling challenge. However, this method requires high computation ability. Yang et al. (2018) proposed an encoder-decoder structure based on SegNet and combine ensembles of classifiers with different input data, whereas the boundaries between objects are not very precise.

3. Theoretical background

In recent years, with the development of computing hardware and remote sensing technologies, convolutional neural networks work as a more and more important part in land cover classification. Traditional classifiers such as Random Forrest use hand-crafted features as input, which requires a lot of labor costs and different tasks require different features, especially involving with processing high-resolution aerial imagery. Compared to traditional methods, convolutional neural networks deliver better results and provide a framework in which high-level features can be learned from training data, whereas these high-level features contain useful spatial correlation among pixels.

Generally speaking, convolutional neural network is a hierarchical model, whose input is raw data, e.g. RGB image. Convolutional neural network, which is composed of different building blocks such as convolutional layer, pooling layer and non-linear activation function, abstracts the high-level semantic information layer by layer from the raw data. This process is called forward propagation.

When raw data is abstracted and propagated to the last layer, convolutional neural network fulfill the task such as classification with the help of objective function, also called loss function or cost function. The parameters of every layer are updated in the help of error back-propagation algorithm, where the error between predictions and true labels is backward propagated from the last layer.

Originally, convolutional neural networks were designed to predict a single class label per image (Krizhevsky et al., 2012). In the meantime, they have also been expanded to semantic segmentation, where a class label is to be predicted for each pixel (Long et al., 2015).

3.1 Convolutional Neural Networks

Convolutional neural network is a feedforward neural network, which means parameters propagate unidirectionally from input layer to the output layer. Its artificial neurons can respond to part of units in the surrounding area and have excellent performance for large image processing. A convolutional neural network consists of one or more convolutional layers and fully connected layers (classical structure of neural network), also includes normalization layers and pooling layers. This configuration enables the convolutional neural network to take advantage of the two-dimensional structure of the input data. Compared to other classification algorithms, such as k nearest neighbor algorithm and support vector machine, convolutional neural network achieves very attractive performance in terms of image recognition and computation speed. In other respects, convolutional neural network requires fewer parameters to be considered due to weight sharing (LeCun et al., 1998), making it an attractive deep learning structure. Weight sharing indicates that when each convolutional kernel is replicated across the entire image, these replicated kernels share the same parameterization (weights vector and bias) and output a feature map. This means that each kernel extracts a specific kind of feature from the entire image. This method dramatically reduces the number of parameters that needs to be trained, therefore it is to accelerate the training speed and reduce computational overhead.

Convolutional neural network is combined with a stack of building blocks that transforms the input data into an output prediction (e.g. probabilities and class scores). The commonly used layers are further discussed below.

3.1.1 Convolutional Layer

Convolutional layer is the fundamental unit of convolutional neural network. This layer is actually a series of filters, or be called convolution kernels, which have a limited perceptual field, but slide among the input data. Because of feedforward architecture, every kernel slides across the width and height of the input data to calculate the dot products of local inputs and kernels, and the feature map of this kernel is produced, as the input for the next layer operation.

If input data is the 5×5 matrix as shown in Figure 3-1 on the right side, and the corresponding convolution kernel is a 3×3 matrix on the left side. At the same time, we specify that each time a convolution operation is performed, the convolution kernel moves by one-pixel position, i.e. the stride is 1.

$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 0 \\ 9 & 8 & 7 & 6 & 5 \\ 4 & 3 & 2 & 1 & 0 \\ 1 & 2 & 3 & 4 & 5 \end{bmatrix}$$

Figure 3-1 Two-dimensional scenario: convolution kernel and input data

The kernel multiplies the input data from left to right and from top to bottom with given stride and at last we obtain a 3×3 feature map as the input of next layer.

$$\begin{bmatrix} 27 & 28 & 29 \\ 28 & 27 & 16 \\ 23 & 22 & 21 \end{bmatrix}$$

Figure 3-2 Two-dimensional scenario: feature map of the first convolution operation

Similarly, suppose in the three-dimensional situation, the input tensor of a convolutional layer l is $\boldsymbol{x}^l \in \mathbb{R}^{H^l \times W^l \times D^l}$ and the convolutional kernel is $\boldsymbol{f} \in \mathbb{R}^{h \times w \times D^l}$, where height, width and depth of the input and the kernel are respectively H^l, W^l, D^l and h, w, D^l (It must be specified that the depth of kernels must be consistent with inputs). The three-dimensional convolution operation actually extends 2D operation to all channels, i.e. D^l channels, at the corresponding position. The sum of $h \times w \times D^l$ elements, each of which is the multiplication result of the kernel and input at corresponding position, is the result of one convolution operation.

Furthermore, if there are D^{l+1} independent kernels $\boldsymbol{F} = \left(\boldsymbol{f}^0, \boldsymbol{f}^1, \ldots, \boldsymbol{f}^{D^{l+1}-1}\right)$ similar to $\boldsymbol{f} \in \mathbb{R}^{h \times w \times D^l}$, then the output is $\boldsymbol{x}^{l+1} \in \mathbb{R}^{H^{l+1} \times W^{l+1} \times D^{l+1}}$ while the input is $\boldsymbol{x}^l \in \mathbb{R}^{H^l \times W^l \times D^l}$ and one convolution operation can be described as below:

$$x^{l+1}_{i^{l+1}, j^{l+1}, d^{l+1}} = \sum_{i=0}^{h} \sum_{j=0}^{w} \sum_{d^l=0}^{D^l} f^{d^{l+1}}_{i,j,d^l} \cdot x^l_{i^{l+1}+i, j^{l+1}+j, d^l}, \tag{3.1}$$

where $x^{l+1}_{i^{l+1}, j^{l+1}, d^{l+1}} \in \boldsymbol{x}^{l+1}$, $f^{d^{l+1}}_{i,j,d^l} \in \boldsymbol{f}^{d^{l+1}}$, $x^l_{i^{l+1}+i, j^{l+1}+j, d^l} \in \boldsymbol{x}^l$, and $(i^{l+1}, j^{l+1}, d^{l+1})$ is the position vector of the convolution result, which satisfies:

$$0 \leq i^{l+1} < H^l - H + 1 = H^{l+1}. \tag{3.2}$$
$$0 \leq j^{l+1} < W^l - W + 1 = W^{l+1}. \tag{3.3}$$
$$0 \leq d^{l+1} < D^{l+1}. \tag{3.4}$$

It should be pointed out that $f^{d^{l+1}}_{i,j,d^l}$ in (3.1) can be regarded as learned weights and it can be found that weights are the same for a specific input in different positions. This is the *weight sharing* of convolutional layer. In addition, there are two important hyperparameters of convolutional layer: kernel size and stride. The settings of hyperparameters will affect the final result of convolutional neural networks.

In fact, parameters of kernels are learned through training and it can conclude numerous different patterns, e.g. boundary, shape and texture. By combining these filters and as the network proceeds to subsequent operations, the basic and general patterns are gradually abstracted into “conceptual” representations with high-level semantics, which can be applied into specific classification tasks.

3.1.2 Pooling Layer

Pooling is another important concept in convolutional neural networks. It is actually a form of non-linear down-sampling. There are two kinds of pooling operations that are commonly applied: average-pooling and max-pooling. It should be pointed out that different from convolutional layer, pooling layer does not contain parameters that need to be learned. It only needs to specify the hyperparameters, i.e. pooling strategy, pooling kernel size and pooling stride.

Same as the convolutional layer section, the pooling layer l can be described as $\boldsymbol{p}^l \in \mathbb{R}^{h\times w}$ (Normally the depth of the input and output of the pooling operation does not change). Suppose the input is $\boldsymbol{x}^l \in \mathbb{R}^{H^l\times W^l\times D}$ and the output is $\boldsymbol{x}^{l+1} \in \mathbb{R}^{H^{l+1}\times W^{l+1}\times D}$. At each average-pooling (max-pooling) operation, the average (maximum) of all values in the coverage area of pooling kernel is used as the result. If the pooling stride is (S_h, S_w) where S_h and S_w are the strides in the direction of height and width respectively, one pooling operation result is:

$$\text{Average} - \text{pooling:}\quad x^{l+1}_{i^{l+1},j^{l+1},d} = \frac{1}{hw}\sum_{i=0}^{h}\sum_{j=0}^{w} x^{l}_{i^{l+1}\cdot S_h+i,\,j^{l+1}\cdot S_w+j,\,d}\,. \tag{3.5}$$

$$\text{Max} - \text{pooling:}\quad x^{l+1}_{i^{l+1},j^{l+1},d} = \max_{0\le i<h,0\le j<w} x^{l}_{i^{l+1}\cdot S_h+i,\,j^{l+1}\cdot S_w+j,\,d}\,, \tag{3.6}$$

where $x^{l+1}_{i^{l+1},j^{l+1},d}$ and $x^{l}_{i^{l+1}\cdot S_h+i,j^{l+1}\cdot S_w+j,d}$ are an element of $\boldsymbol{x}^{l+1}$ and $\boldsymbol{x}^l$ respectively.

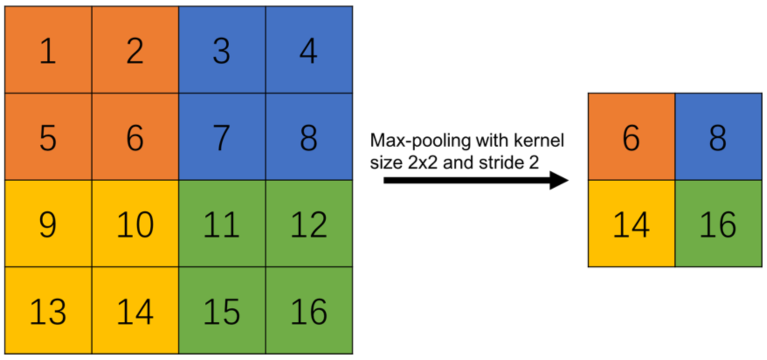

Figure 3-3 The most common used pooling strategy is max-pooling. Here is an example of max-pooling with kernel size 2x2 and stride 2, where the max is taken over 4 numbers.

Pooling layer is implemented for down-sampling and abstraction of input. In the past work of convolutional neural networks, researchers generally verify that the pooling layer has the following two effects:

1. Translation invariant. The pooling operation makes the model more concerned with the presence of certain features rather than the specific location of features and it makes that feature learning contain a degree of freedom that can tolerate some minor displacements. For example, as shown below, if the position of numbers is disrupted within the kernel size, the result does not change.

2. Dimension reduction. Due to the down-sampling of pooling layer, one element in the pooling result corresponds to a sub-region of the input data, so pooling operations are equivalent to implement spatially dimension reduction.

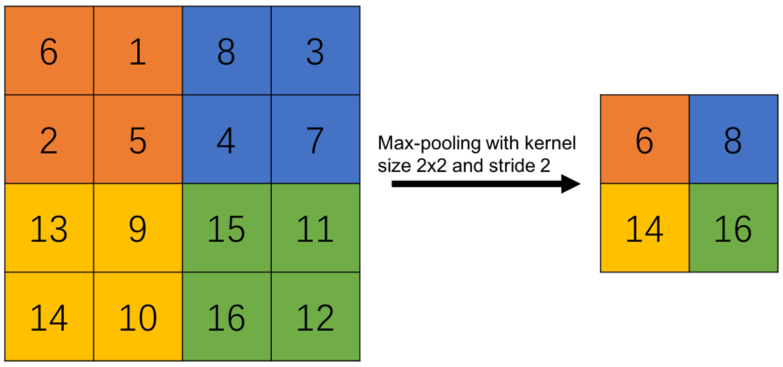

Figure 3-4 Feature invariant within pooling kernel size.

3.1.3 Activation Function

Activation function is also called non-linearity mapping. The purpose of activation function is to increase the non-linearity of the network, otherwise the stacking of several linear layers can only function as a linear mapping and cannot form complex functions. There are two activation functions that are commonly used: Sigmoid function and ReLU function.

Intuitively, the activation function simulates the characteristics of biological neurons: accept input signals and produce outputs. In neuroscience, biological neurons usually have a threshold. When the cumulative effect of the input signals obtained by the neuron exceeds the threshold, the neuron is activated, otherwise suppressed. In artificial neural network, the sigmoid function can simulate this biological process, which has a very important position in the development of neural network.

Sigmoid function is also known as Logistic function:

$$\sigma(x) = \frac{1}{1 + \exp(-x)}. \tag{3.7}$$

As shown in Figure 3-5, the range of the output is compressed between $[0,1]$ after processing by sigmoid function. But it must be pointed out at both ends of sigmoid function that the value bigger than 5 (or smaller than -5) will be compressed to 1 (or 0) no matter how big (or small) it is. This brings a serious problem, which is the vanishing effect of the gradient. As shown in Figure 3-5, the gradient of the part that is bigger than 5 (or smaller than -5) is very close to 0. The error is difficult or even impossible transferred to front layer during loss back propagation, which the entire network cannot be trained as a result (network parameters would have no updating if gradient equals zero).

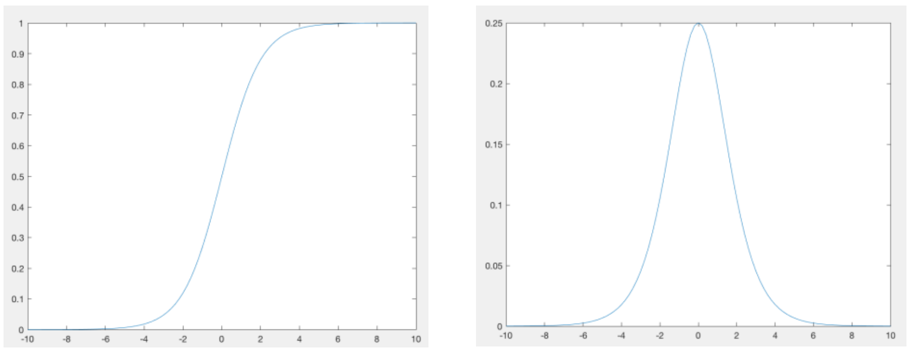

Figure 3-5 Sigmoid function and its gradient function

In order to avoid gradient vanishing, Nair and Hinton introduced Rectified Linear Unit (ReLU) into neural network in 2010 (Nair & Hinton, 2010). ReLU is currently one of the most commonly used activation functions in deep convolutional neural networks.

ReLU is actually a piecewise function:

$$\text{rectifier}(x) = \max\{0, x\} = \begin{cases} x & if\ \ x \geq 0 \\ 0 & if\ \ x < 0 \end{cases}. \tag{3.8}$$

The gradient of ReLU is 1 when $x \geq 0$, otherwise is 0. For $x \geq 0$, ReLU completely eliminates the gradient vanishing effect. In addition, ReLU contributes to the convergence of the stochastic gradient descent (SGD) method, and the convergence speed is about six times faster than another activation function, i.e. the hyperbolic tangent function, when both kinds of units applied in equivalent networks (Krizhevsky et al., 2012). Because of these high-quality characteristics, ReLU has become the first choice for convolutional neural network and other deep learning model, e.g. RNN.

3.1.4 Fully Connected Layer

Fully connected layer integrates the local features into global features. The operations we discussed above is to map the original data to the implicit feature space, while the fully connected layer connects the learned features to high-level inference. In actual use, the fully connected layer can be implemented by convolution operations: if the previous layer is a fully connected layer, the kernel size of current layer is set to 1×1; if the previous layer is not a fully connected layer, the kernel size is set to $H \times W \times D$, where H, W and D are respectively the height, width and depth of previous layer output.

Taking VGG-16 (Simonyan et al., 2015) as an example, the output of the last convolutional layer after pooling and activation function is a tensor of $7 \times 7 \times 512$, whereas the input is an image of $224 \times 224 \times 3$. If the next layer is fully connected layer and has 4096 neurons, there should be 4096 convolutional kernels and the kernel size should be set to $7 \times 7 \times 512$. After convolution an output of $1 \times 1 \times 4096$ is obtained.

3.1.5 Loss Function

The loss function is used to estimate the deviation between predictions and true labels. Various loss functions are implemented in convolutional neural network for different tasks and cross-entropy loss function (also named softmax loss function) is mostly used for classification problem.

There are two concepts that need to be specified: softmax and cross-entropy. Generally speaking, softmax normalizes the classification output (class scores) to a probabilistic class scores and cross-entropy characterizes the similarity between predictions and reference.

Softmax maps the outputs of multiple neurons to the interval (0,1) and that can be regarded as probability. If there is a vector $\boldsymbol{Z}$ and Z_i represents the i$^{\text{th}}$ elements in the vector, then the softmax value of this element is:

$$S_{Z_i} = \frac{e^{Z_i}}{\sum_j e^{Z_j}}. \tag{3.9}$$

The following figure shows the reason why softmax outputs probabilities more vividly:

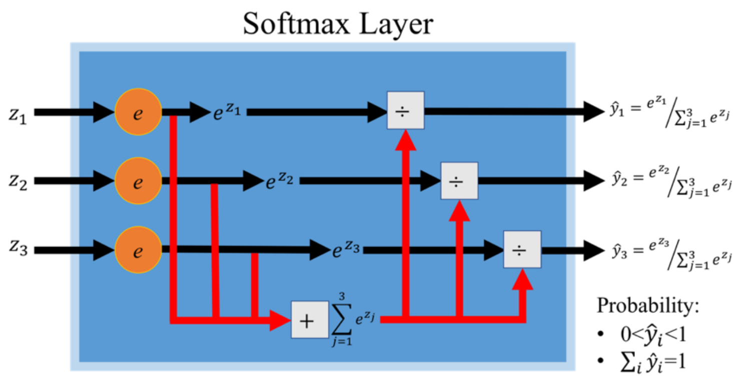

Figure 3-6 Softmax architecture with three input nodes

The original inputs are mapped to the value inside (0,1) and the sum of these values is 1, so the converted values can be regarded as probability. The predictions are made by choosing the class with the highest probabilistic class score.

Suppose a classification task has a total of N training samples. The input of the softmax layer, e.g. class scores, is $\boldsymbol{Z}_n = (Z_n^1, Z_n^2, \dots, Z_n^C)^{\mathrm{T}}$, the corresponding true label is $y_n \in \{y^1, y^2, \dots, y^C\}$ and the output of softmax is $\hat{\boldsymbol{y}}_n = (\hat{y}_n^1, \hat{y}_n^2, \dots, \hat{y}_n^C)^{\mathrm{T}}$, i.e. the predicted probabilities of sample n, where C is the number of class and $\hat{y}_n^c$ is the probability for class y^c. Now the loss of model can be calculated by similarity between predictions and reference and the cross-entropy is the instrument to achieve that.

The posterior probability $P_n(y^c|x)$, i.e. $\hat{y}_n^c$, for sample n to take class label y^c given the input data x:

$$\hat{y}_n^c = P_n(y^c|x) = \text{softmax}(\boldsymbol{Z}_n, y^c) = \frac{exp(Z_n^c)}{\sum_{j=1}^{C} exp(Z_n^j)}. \tag{3.10}$$

Therefore cross-entropy loss function can be described as below:

$$L = -\frac{1}{N}\sum_{n=1}^{N}\sum_{c=1}^{C} \delta(y_n, y^c) \cdot log(\hat{y}_n^c). \tag{3.11}$$

where $\delta(y_n, y^c)$ is *Kronecker delta*, of which the result is 1 if the variables are equal, and 0 otherwise.

3.1.6 Batch Normalization

Training deeper neural networks has always been the important method to improve the performance of models in deep learning (Szegedy et al., 2014; Simonyan & Zisserman, 2014). Batch normalization (Ioffe & Szegedy, 2015) proposed by Google in 2015 dramatically accelerates deep network training, thereby making it easier and more stable to train deep network models. In addition, the batch normalization is not only suitable for deep network, but for shallow neural network, it can also improve the generalization. At present, batch normalization has been essential for almost all convolutional neural networks.

As shown below is the batch normalization transform, where there are m values of a mini-batch. We normalize each activation by using mini-batches in stochastic gradient training, to make it have the mean of zero and the variance of 1.

$$\textbf{Input}\text{: Values of } x \text{ over a mini} - \text{batch: } \mathfrak{B} = \{x_{1\dots m}\};$$
$$\text{Parameters to be learned: } \gamma, \beta$$

$$\textbf{Output}\text{: } \{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$$

$$\text{mini} - \text{batch mean: } \mu_{\mathfrak{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$$

$$\text{mini} - \text{batch variance: } \sigma_{\mathfrak{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathfrak{B}})^2$$

$$\text{normalize: } \hat{x}_i \leftarrow \frac{x_i - \mu_{\mathfrak{B}}}{\sqrt{\sigma_{\mathfrak{B}}^2 - \epsilon}}$$

$$\text{scale and shift: } y_i \leftarrow \gamma\hat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i)$$

There are four steps in batch normalization. The first two steps calculate the mean and the variance of the mini-batch separately. The third step normalizes the mini-batch based on the calculated mean and variance. The last step scale and shift allows the batch normalization transform to represent identity (Ioffe & Szegedy, 2015), i.e. restore the distribution of data that is learned in the previous layer.

Batch normalization helps to properly initialize the neural networks by transforming the data fed to activation functions into a unit gaussian distribution at the beginning of the training. Except for that, it can be regarded as doing preprocessing at every layer of the network to help to accelerate the training. Batch normalization is added generally before the nonlinearity, e.g. ReLU.

3.2 Stochastic Gradient Descent

In deep learning, the loss function is minimized by using an optimization algorithm. The loss function also can be named as objective function. Gradient descent is a commonly used optimization method. Although gradient descent is rarely used directly in deep learning, the mechanism of gradient descent is the foundation of stochastic gradient descent.

3.2.1 Gradient Descent

By taking a simple one-dimensional gradient descent as an example, the reason why the gradient descent algorithm may reduce the value of the objective function is explained. Assume that the input and output of the continuously derivable function $f: \mathbb{R} \to \mathbb{R}$ are both scalars. Given a sufficiently small absolute value ϵ, according to the Taylor expansion formula, the following approximation can be acquired:

$$f(x+\epsilon) \approx f(x) + \epsilon f'(x), \tag{3.12}$$

where $f'(x)$ is the gradient of function f at x. The gradient of a one-dimensional function is a scalar, also known as the derivative.

Suppose there is a constant $\eta > 0$, and also $|\eta f'(x)|$ is sufficiently small. Then ϵ can be replaced by $-\eta f'(x)$:

$$f\big(x - \eta f'(x)\big) \approx f(x) - \eta f'(x)^2. \tag{3.13}$$

If $f'(x) \neq 0$, then $\eta f'(x)^2 > 0$:

$$f\big(x - \eta f'(x)\big) \lesssim f(x). \tag{3.14}$$

If x is Iteratively replaced by $x - \eta f'(x)$, the value of the function $f(x)$ can decrease. Thus, in gradient descent firstly an initial value x and a constant $\eta > 0$ is picked and x is iterated continuously by $x - \eta f'(x)$ until the stop condition is triggered, e.g. $f'(x)^2$ is sufficiently small or the number of iterations has reached a certain value.

The constant $\eta > 0$ is often called the learning rate in machine learning. This is a hypermeter that need to be specified manually. If a too small learning rate is implemented, it will cause x updates slowly and requires more iterations to achieve a optimal result. If a learning rate is excessive, $|\eta f'(x)|$ may be too large to make the mentioned Taylor expansion formula (3.9) no longer valid.

After understanding the one-dimensional gradient descent, a more general case needs to be considered: the input of objective function is vector and output is scalar, i.e. $f: \mathbb{R}^d \rightarrow \mathbb{R}$, where input vector $\boldsymbol{x} = [x_1, x_2, \dots, x_d]^{\mathrm{T}}$ is d-dimensional. The gradient of the objective function $f(\boldsymbol{x})$ about $\boldsymbol{x}$ is a vector of d elements:

$$\nabla_x f(\boldsymbol{x}) = \left[\frac{\partial f(\boldsymbol{x})}{\partial x_1}, \frac{\partial f(\boldsymbol{x})}{\partial x_2}, \dots, \frac{\partial f(\boldsymbol{x})}{\partial x_d}\right]^{\mathrm{T}}. \tag{3.15}$$

Every partial derivative element $\partial f(\boldsymbol{x})/\partial x_i$ represents the rate of change of the objective function $f(\boldsymbol{x})$ about x_i. In order to measure the rate of change of $f(\boldsymbol{x})$ along unit vector $\boldsymbol{u}$ ($\|\boldsymbol{u}\| = 1$), the directional derivative is defined as:

$$\mathrm{D}_u f(\boldsymbol{x}) = \lim_{h \to 0} \frac{f(\boldsymbol{x} + h\boldsymbol{u}) - f(\boldsymbol{x})}{h}. \tag{3.16}$$

According to the nature of the directional derivative, the above directional derivative can be rewritten as:

$$\mathrm{D}_u f(\boldsymbol{x}) = \nabla_x f(\boldsymbol{x}) \cdot \boldsymbol{u}. \tag{3.17}$$

Directional derivative gives the rate of change of $f(\boldsymbol{x})$ along every possible direction. In order to minimize $f(\boldsymbol{x})$, the hopeful way is to find the direction, along which the $f(\boldsymbol{x})$ decreases the fastest. The directional derivative $\mathrm{D}_u f(\boldsymbol{x})$ is minimized with the help of unit vector $\boldsymbol{u}$.

Since $\mathrm{D}_u f(\boldsymbol{x}) = \|\nabla_x f(\boldsymbol{x})\| \cdot \|\boldsymbol{u}\| \cdot \cos(\theta) = \|\nabla_x f(\boldsymbol{x})\| \cdot \cos(\theta)$, where θ is the included angle between the gradient $\nabla_x f(\boldsymbol{x})$ and the unit vector $\boldsymbol{u}$, and $\cos(\theta)$ reaches minimum value -1 when $\theta = \pi$, the directional derivative $\mathrm{D}_u f(\boldsymbol{x})$ is minimized when the unit vector $\boldsymbol{u}$ is in the opposite direction of the gradient $\nabla_x f(\boldsymbol{x})$. Therefore, the value of the objective function f can decrease iteratively by using gradient descent algorithm:

$$x \leftarrow x - \eta \nabla_x f(\boldsymbol{x}), \tag{3.18}$$

where $\eta > 0$ is the learning rate.

3.2.2 Stochastic Gradient Descent

Gradient descent algorithm needs to calculate the gradient of every training sample, when it optimizes the objective function in each iteration. If the training set is big (especially in deep learning), the efficiency of gradient descent will be very low. Also due to limitations of hardware resources (GPU memory, etc.), this method is basically unrealistic in practical applications. Thus, the stochastic gradient descent is commonly used to replace gradient descent in deep learning.

Suppose the objective function is defined as below:

$$f(\boldsymbol{x}) = \frac{1}{n}\sum_{i=1}^{n} f_i(\boldsymbol{x}), \tag{3.19}$$

where there are n training samples and $\boldsymbol{x}$ is the parameter vector of the model.

Then the gradient of the objective function is:

$$\nabla f(\boldsymbol{x}) = \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(\boldsymbol{x}), \tag{3.20}$$

If gradient descent is implemented, the computational overhead of every iteration is $O(n)$ and it grows linearly with number of training samples n. Therefore, when the number of training samples is large, each iteration computational overhead of the gradient descent is high.

Stochastic gradient descent computes one training sample every iteration to update model parameters, i.e. $\nabla f_i(\boldsymbol{x})$. Obviously, the computational overhead drops to $O(1)$ from $O(n)$ in every iteration.

Although the efficiency is raised, considering only one sample every time in stochastic gradient descent makes that the optimization may be not the overall optimization of the model. Thus, it is important to traverse the whole training samples. Every traversal of the entire input dataset is named as epoch. In deep learning, a simple change is implemented in stochastic gradient descent by using not a single training sample, but a batch of samples. With the gradient information of the sample batch, the parameters of the model are updated. This is called mini-batch based stochastic gradient descent. Due to that one batch consists of a quantity of samples, this strategy gains more robust gradient information compared to the standard stochastic gradient descent.

3.3 Regularization

Generalization ability describes the performance of the trained learning algorithm on test dataset. If one learning algorithm performs well on both training and test dataset, it has strong generalization ability. Otherwise if it performs well only on training dataset but does not achieve an ideal result on test dataset, its generalization is not acceptable. This situation is also called overfitting. In this case regularization is implemented to prevent overfitting.

3.3.1 Weight Decay

Although increase the training dataset may reduce overfitting, more training samples may not be available in every situation. The weight decay is the common method for dealing with overfitting problem, and also is named as L_2 regularization. L_2 regularization is to add a regularization in the loss function:

$$L = L_0 + \frac{\lambda}{2}\sum_{w} w^2, \tag{3.21}$$

where L_0 is the original loss function, n is the number of training samples, w represents the elements in the weights vector, $\frac{\lambda}{2}\sum_w w^2$ is the L_2 regularization and λ is the coefficient of the regularization, which weighs the proportion of the regularization and the original loss L_0. When $\lambda > 0$, the elements in weights vector will not grow too large. The mechanism is explained below.

First is the derivative:

$$\frac{\partial L}{\partial w} = \frac{\partial L_0}{\partial w} + \lambda w. \tag{3.22}$$

The stochastic gradient descent updates look as follows:

$$w \leftarrow w - \eta\frac{\partial L_0}{\partial w} - \eta\lambda w = (1 - \eta\lambda)w - \eta\frac{\partial L_0}{\partial w}. \tag{3.23}$$

w is becoming smaller because η and λ are positive and this is also the origin of the name weight decay. Weight decay adds constraints to the model to control the absolute value of the parameters and reduces therefore the complexity of the network. It can help avoid overfitting to constraint the parameters that they are not too large.

3.3.2 Dropout

Weight Decay prevents overfitting by adjusting loss functions, whereas dropout modifies the structure of network to achieve the goal. Its mechanism is very simple: several neurons, random picked with probability p, will be temporarily ignored in training and all neurons are activated in testing. Its process is shown below:

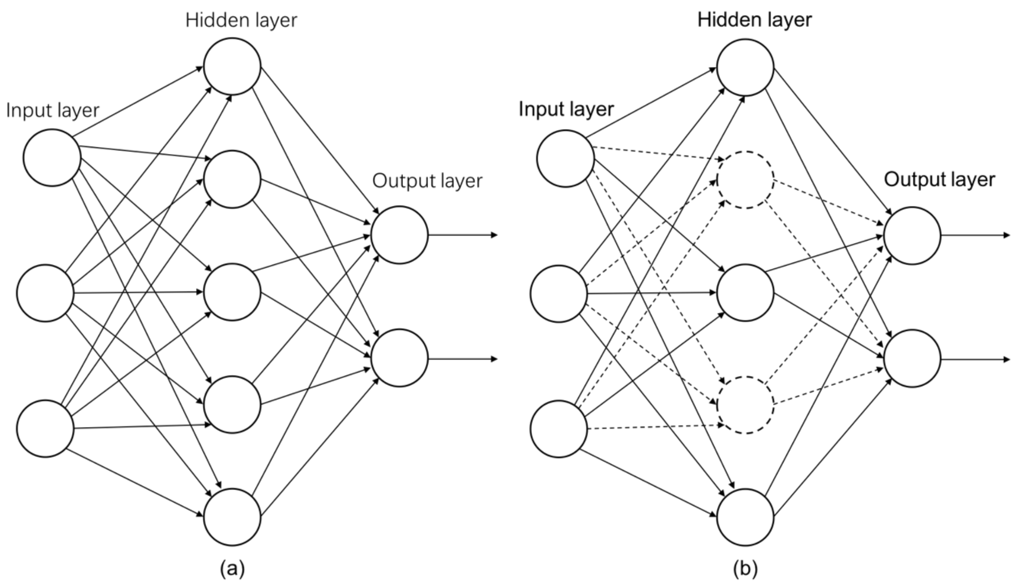

Figure 3-7 An example of dropout. Assume the network in (a) need to be trained. At the beginning of training, 40% neurons in hidden layer are randomly picked out and temporarily ignored, shown in (b). The other neurons update its weights. This is the process in one iteration. In next iteration, another 40% neurons are randomly picked out.

The training process that uses dropout is equivalent to training a lot of "mini-network" (combinations of different neurons in hidden layer). Every "mini-network" can present predictions and as the training progresses, most "mini-network" can give correct classification results.

3.4 Forward propagation and backpropagation

The neural network contains several neurons in every layer and the layers are connected with weights matrixes. The process, the information passed from the previous layer to the next layer, is called forward propagation. Backpropagation algorithm is actually the short of the backward propagation of errors. It is one of the most successful learning algorithms in neural networks.

3.4.1 Forward Propagation

Forward propagation is also mentioned as feedforward process. It refers to the variables (including outputs) of the model that are sequentially calculated and stored in the order of the input layer to the output layer. The figure below is more intuitive:

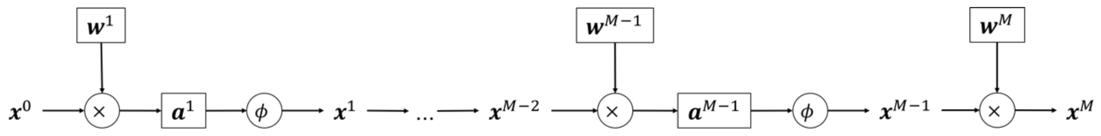

Figure 3-8 Forward propagation of a simplified $\boldsymbol{M}$ *layer neural network, where* $\boldsymbol{W}$ *is the weights vector,* $\boldsymbol{\phi}$ *is the activation function,* $\boldsymbol{x}^0 \in \mathbb{R}^d$ *is the input,* $\boldsymbol{x}^M \in \mathbb{R}^C$ *is the output and* d *and* C *is the dimension of the input and the number of classes respectively.*

3.4.2 Backpropagation

Suppose the loss function of the abovementioned network is ℓ:

$$L = \ell(\boldsymbol{x}^M, y), \tag{3.24}$$

where y is the class label. Backpropagation is an iterative learning algorithm, where arbitrary parameter is updated like follow:

$$w \leftarrow w - \eta \nabla w, \tag{3.25}$$

where $\eta > 0$ is the learning rate of the network. Backpropagation algorithm is based on gradient descent strategy, which adjusts the parameters and variables in the opposite direction of their gradients. Taking the last layer of the network above as an example:

$$\nabla \boldsymbol{w}^M = \frac{\partial L}{\partial \boldsymbol{w}^M}. \tag{3.26}$$

According to the chain rule in calculus:

$$\frac{\partial L}{\partial \boldsymbol{w}^M} = \frac{\partial L}{\partial \boldsymbol{x}^M} \cdot \frac{\partial \boldsymbol{x}^M}{\partial \boldsymbol{w}^M}, \tag{3.27}$$

In forward propagation, $\boldsymbol{x}^M = \boldsymbol{w}^M \boldsymbol{x}^{M-1}$, $\boldsymbol{x}^{M-1} = \phi(\boldsymbol{a}^{M-1})$ and $\boldsymbol{a}^{M-1} = \boldsymbol{w}^{M-1} \boldsymbol{x}^{M-2}$. Thus, the update of $\boldsymbol{w}^M$ is:

$$\nabla \boldsymbol{w}^M = \frac{\partial L}{\partial \boldsymbol{w}^M} = \frac{\partial L}{\partial \boldsymbol{x}^M} \cdot \boldsymbol{x}^{M-1^{\mathrm{T}}}. \tag{3.28}$$

where $\frac{\partial L}{\partial x^M}$ is available according to (3.21). To further simplify the calculations, Sigmoid function is used as the activation function due to its character $\phi'(\boldsymbol{a}) = \phi(\boldsymbol{a})\big(1 - \phi(\boldsymbol{a})\big)$. Along the order of the output layer to hidden layers, the gradient of variables in the hidden layer can be calculated:

$$\frac{\partial L}{\partial \boldsymbol{x}^{M-1}} = \frac{\partial L}{\partial \boldsymbol{x}^{M}} \cdot \frac{\partial \boldsymbol{x}^{M}}{\partial \boldsymbol{x}^{M-1}} = \frac{\partial L}{\partial \boldsymbol{x}^{M}} \cdot {\boldsymbol{w}^{M}}^{\mathrm{T}}. \tag{3.29}$$

$$\begin{aligned}\frac{\partial L}{\partial \boldsymbol{a}^{M-1}} &= \frac{\partial L}{\partial \boldsymbol{x}^{M-1}} \cdot \frac{\partial \boldsymbol{x}^{M-1}}{\partial \boldsymbol{a}^{M-1}} \\ &= \frac{\partial L}{\partial \boldsymbol{x}^{M-1}} \cdot \phi'(\boldsymbol{a}^{M-1}) \\ &= \frac{\partial L}{\partial \boldsymbol{x}^{M-1}} \cdot \phi(\boldsymbol{a}^{M-1}) \cdot \left(1 - \phi(\boldsymbol{a}^{M-1})\right) \\ &= \frac{\partial L}{\partial \boldsymbol{x}^{M-1}} \cdot \boldsymbol{x}^{M-1} \cdot (1 - \boldsymbol{x}^{M-1}). \end{aligned} \tag{3.30}$$

Then the update of $\boldsymbol{w}^{M-1}$ is:

$$\nabla \boldsymbol{w}^{M-1} = \frac{\partial L}{\partial \boldsymbol{w}^{M-1}} = \frac{\partial L}{\partial \boldsymbol{a}^{M-1}} \cdot \frac{\partial \boldsymbol{a}^{M-1}}{\partial \boldsymbol{w}^{M-1}} = \frac{\partial L}{\partial \boldsymbol{a}^{M-1}} \cdot {\boldsymbol{x}^{M-2}}^{\mathrm{T}}. \tag{3.31}$$

Follow this procedure until the weights vector $\boldsymbol{w}^{1}$ is updated. Then one iteration is finished. When the deep learning model is trained, forward propagation and backpropagation are both very essential. First the model needs to be fed with input data and forward propagate the data till the result is obtained; then the error is calculated and back propagated to hidden layers to adjust the parameters. Since the variables of forward propagation are used in back-propagation, then the reuse of the variables causes that the memory cannot be released immediately after the forward propagation.

3.5 Cost-sensitive Learning

In the classical hypothesis of machine learning, it is often assumed that different class samples are balanced, that is, the number of samples is the same or almost the same, but the actual tasks in our realistic scenarios often do not meet this assumption. In general, an imbalanced training sample distribution will result in a trained model that focuses on classes with a larger number of samples, while ignoring classes that has fewer samples, so that the generalization of the model is affected. An extreme example is that for a binary classification problem, there are 99 positive samples and only one negative sample in the training set. Without considering the imbalanced class distribution, the learning algorithm will cause the classifier to abandon the negative prediction, because it can obtain a very high accuracy that all samples are classified as positive in the training phase. But imagine that if the test set has 99 negative samples and only one positive sample, then the classifier only has a very low accuracy and completely failed on the test set. In fact, in addition to common classification and regression problems, pixel-level tasks such as image semantic segmentation (Long et al., 2015) and depth estimation (Liu et al., 2015) also have phenomena of the imbalanced class distribution. In order to further improve the generalization of the model and reduce the impact of the imbalanced class distribution, cost-sensitive learning is appended to the loss function.

3.5.1 Cost-sensitive Vector

One of the commonly used cost-sensitive methods is the cost-sensitive vectors. Suppose a cost-sensitive vector $\boldsymbol{\omega} = (\omega_1, \omega_2, \dots, \omega_C)$ and $\boldsymbol{\omega} \in \mathbb{R}_+^C$, where C is the number of classes. The elements of ω_c indicates the penalty that the sample is misclassified into c_{th} class. The new loss function updated from (3.11) is shown below:

$$L = -\frac{1}{N}\sum_{n=1}^{N}\sum_{c=1}^{C}\omega_c \cdot \delta(y_n, y^c) \cdot log(\hat{y}_n^c)\,. \tag{3.32}$$

3.5.2 Specify the Weights

According to (Eigen et al., 2015) and (Xie & Tu, 2015), the weights in cost-sensitive vector are specified in two ways. The first specification formula is described below:

$$\omega_c = \frac{median_{freq}}{freq(c)}, \tag{3.33}$$

where $freq(c)$ is the number of pixels of class c divided by the total number of pixels in images where c is present, and $median_{freq}$ is the median of these frequencies (Eigen et al., 2015).

For binary classification, Xie & Tu (2015) proposed another formula:

$$\omega_c = 1 - freq(c). \tag{3.34}$$

4. Methodology

An attractive methodology based on convolutional neural network for land cover classification has been proposed and achieved a great success (Yang et al., 2018). However, there are still some misclassifications and the accuracy at boundaries is obviously lower than overall accuracy. Deeper network structures and architectures are investigated to pay more attention on a precise delineation of the object boundaries. Generally speaking, there are two investigation variants. The first one is hard-negative mining. Pixels that are hard to be classified, need to be found out. It can be done by setting a threshold of probabilistic score, the pixels whose correct class scores are above the threshold are considered as easy ones, otherwise are hard ones. Then investigations on how to improve the classification of them have to be conducted. The second one is adding more prior knowledge to the existing convolutional neural network model. In this case, the prior object boundary information needs to be firstly extracted out from the reference. Afterwards, this knowledge needs to be incorporated into the model by modifying the structure, whereas a research on modification must be taken. Besides, an investigation of the width of boundary should be taken, in order to find out the relationship between performance and the boundary.

All networks are implemented based on tensorflow framework (Abadi et al., 2016).

4.1 SkipNet: A Deep Convolutional Neural Network

Yang et al. have proposed a new approach to determine land cover based on convolutional neural network. The input data are high-resolution digital aerial images. The proposed convolutional neural network architecture is based on SegNet architecture (Badrinarayanan et al., 2017), while the new model SkipNet is deeper but requires fewer parameters. SkipNet is a deep fully encoder-decoder convolutional neural network architecture for semantic segmentation, where semantic segmentation indicates that each pixel in the image is predicted with a label. The encoder part is similar to a standard convolutional neural network, whereas the decoder part is to restore low-resolution feature maps, the result of the encoder network, to full-scale feature maps as input for per-pixel classification.

4.1.1 Architecture

SkipNet applies a symmetric encoder-decoder structure to perform semantic segmentation. The encoder part of the model has four blocking units, where every blocking unit is composed of three convolution blocks and one max-pooling layer. A convolution block contains one convolutional layer, followed by batch normalization

(Ioffe & Szegedy, 2015) and a rectified linear unit (ReLU) adding non-linearity. The decoder part is symmetric to the encoder and has four blocking units, but each blocking unit is started with upsample layer, continued with three convolution blocks, of which the inside order does not change, i.e. the convolutional layer, batch normalization and a rectified linear unit. The upsample layer is realized with bilinear interpolation. At last, there is a skip connection at the end of each blocking unit of the decoder.

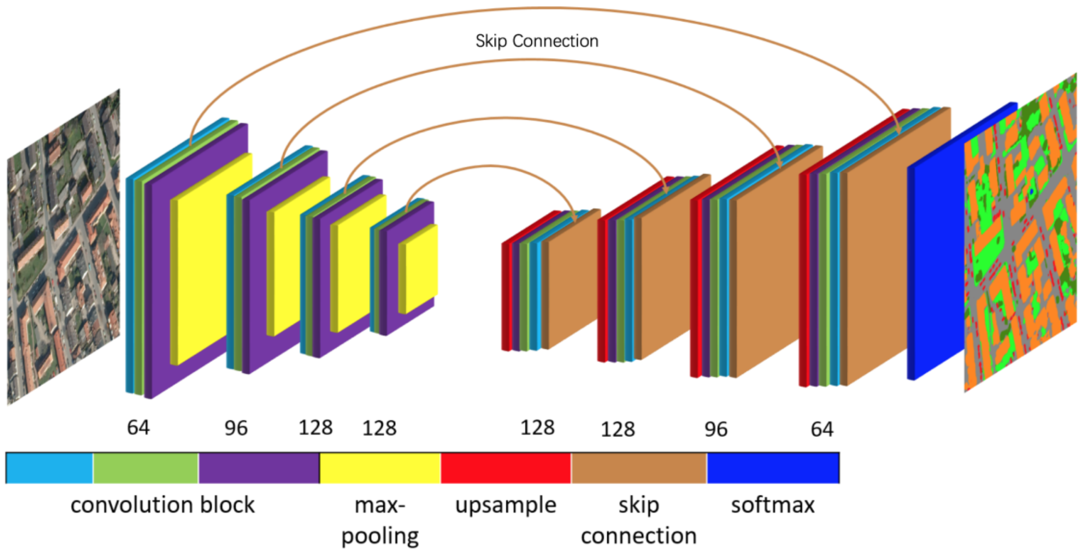

Figure 4-1 Architecture of SkipNet.

The skip connection is the mechanism to combine the features. The sub-sample, e.g. max-pooling, is not beneficial for boundary delineation, because the image becomes increasingly lossy (Badrinarayanan et al., 2017). Due to this reason, the feature maps before max-pooling in the encoder are concatenated with the feature maps in the decoder at corresponding positions. Trainable 1×1 convolutions are used to reduce the dimension of the combined feature maps. The implementation of the skip connection tries to restore the boundary information in the encoder before max-pooling, because the boundary information is preserved better in the encoder. The mechanism is illustrated below:

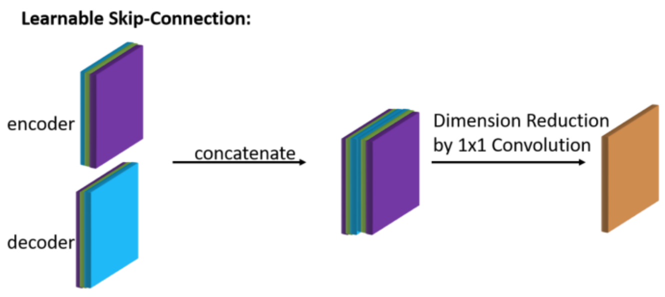
Learnable Skip-Connection:
encoder
decoder
concatenate
Dimension Reduction by 1x1 Convolution

Figure 4-2 Structure of a skip connection; color code: cf. Figure 4-1.

Each convolutional layer has the kernel size of 3×3 and a stride of 1 pixel. At each convolutional layer zero-padding is used to keep the spatial dimension of the feature maps. In the encoder part, the convolutional layers in the first convolution block have 64 convolutional kernels, whereas the convolutional layers in the second convolution block have 96 kernels. The convolutional layers in third and fourth convolution block have 128 convolutional kernels. In the decoder part, it is the mirror of the encoder part. Max-pooling has a pooling window size of 2×2 and its stride is 2 pixels.

At the end of the decoder, the softmax classifier is applied to convert the feature maps to the probabilities of every class. This is implemented by a 1×1 convolutional layer that outputs a tensor of dimension $C \times H \times W$, where H and W is the respectively the height and width of the input image and C is the number of classes.

Suppose the vector $\boldsymbol{Z}_i = \left(Z_i^1, Z_i^2, \dots, Z_i^C\right)^{\mathrm{T}}$ represents the class scores of the pixel i in the input image, where C is the number of classes, $y_i \in \{y^1, y^2, \dots, y^C\}$ is the corresponding true label and the output of softmax is $\hat{\boldsymbol{y}}_i = \left(\hat{y}_i^1, \hat{y}_i^2, \dots, \hat{y}_i^C\right)^{\mathrm{T}}$, i.e. the predicted probabilities of pixel i and $\hat{y}_i^c$ is the probability for class y^c:

$$\hat{y}_i^c = P_i(y^c|x) = \mathrm{softmax}(\boldsymbol{Z}_i, y^c) = \frac{exp(Z_i^c)}{\sum_{j=1}^{C} exp(Z_i^j)}, \tag{4.1}$$

where $P_i(y^c|x)$ is the posterior probability for pixel i to take class label y^c given the image data x.

In training, all the parameters are determined, and the batch size must be set to 1 due to the limit of the GPU. The parameters in the model are trained with the stochastic gradient descent and the backpropagation algorithm to optimize the objective function, where cross-entropy loss function is implemented and described below:

$$L = -\frac{1}{H \cdot W} \sum_{i=1}^{H \cdot W} \sum_{c=1}^{C} \omega_c \cdot \delta(y_i, y^c) \cdot log(\hat{y}_i^c), \tag{4.2}$$

where $\omega_c \in \boldsymbol{\omega} = \{\omega_1, \omega_2, \dots, \omega_C\}$ is a class weight computed according to (3.33) to compensate for an imbalanced class distribution in training data (Yang et al., 2018) and $\delta(y_i, y^c)$ is *Kronecker delta*, of which the result is 1 if the variables are equal, and 0 otherwise.

4.1.2 Implementation and Training

For training SkipNet we employed a stochastic gradient descent optimizer with weight decay 0.0005. The input size is 256×256 pixels. Due to the limitations of our GPU, the mini-batch size is set to 1. The base learning rate is set to 0.01 and decreased to 0.001 after 15 epochs in a total of 30 epochs training.

4.2 Investigations on Improving the Classification of Land Cover Based on SkipNet

Generally speaking, there are two investigation variants. The first one is hard-negative mining. Pixels that are hard to be classified, need to be found out. It can be done by setting a threshold of probabilistic score, the pixels whose correct class scores are above the threshold are considered as easy ones, otherwise are hard ones. Then investigations on how to improve the classification of them have to be conducted. It could be:

- Retraining only the hard pixels in the same convolutional neural network model
- Using a patch-based convolutional neural network model which requires patches as input, where the patches are extracted out by centering the hard pixels

The second one is adding more prior knowledge to the actual convolutional neural network model. In this case, the prior object boundary information needs to be firstly extracted out from the reference. Afterwards, this knowledge needs to be incorporated into the model by modifying the structure, whereas a research on modification must be taken. Besides, an investigation of the width of boundary should be taken, in order to find out the relationship between performance and the boundary.

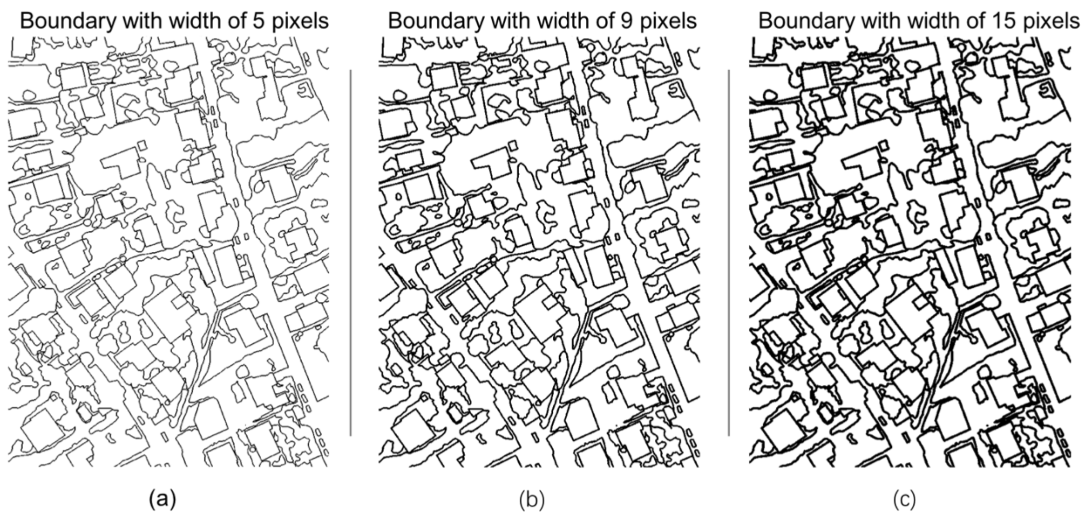
Boundary with width of 5 pixels
Boundary with width of 9 pixels
Boundary with width of 15 pixels
(a)
(b)
(c)

Figure 4-3 Class boundaries generated by reference with different width. Black and white represent respectively boundaries and others.

4.2.1 Hard-Negative Mining

Hard-negative mining is a method, which uses pre-trained models to make inference on the training dataset, and then collect the misclassified examples, i.e. negative samples, to build a hard-negative dataset. With this new dataset, a new model is trained, or the old model is fine-tuned (Felzenszwalb et al., 2010).

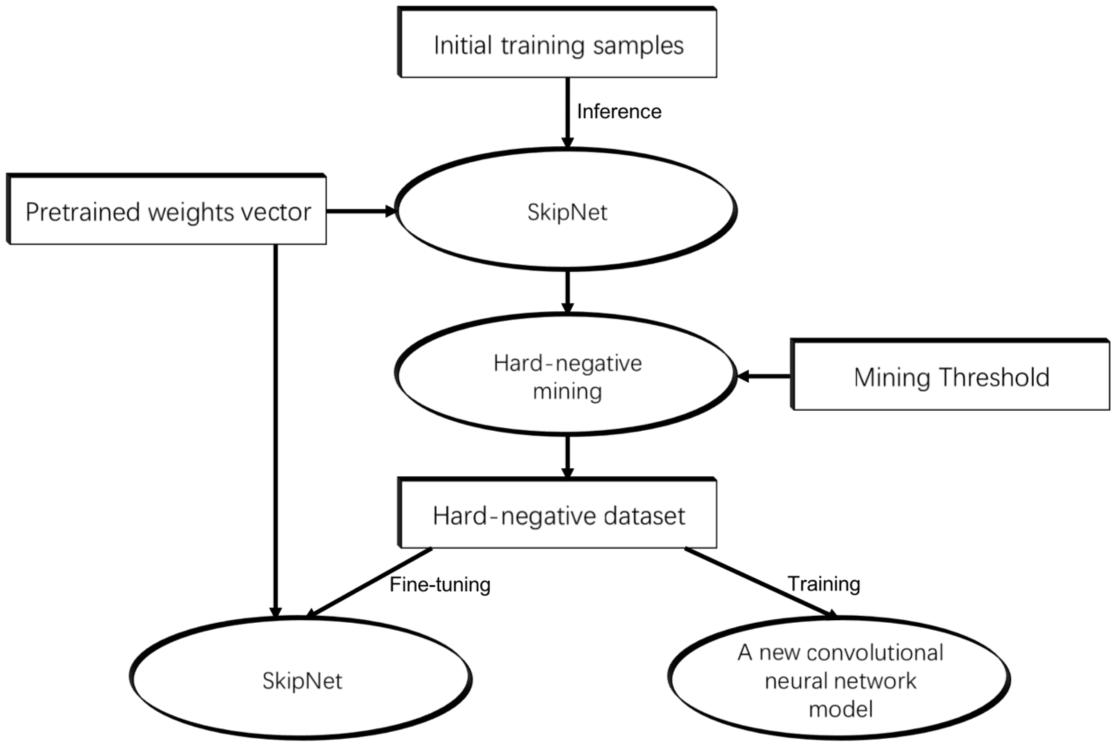

Figure 4-4 Illustration of hard-negative mining.

Our hard-negative dataset is built with an offline method, which tests the initial dataset to obtain probabilities with the pre-trained SkipNet model. After testing, the pixels in initial training dataset are summarized into three categories by setting a mining threshold $Thresh_M$. Suppose the true label is $y_i \in \{y^1, y^2, \dots, y^C\}$, where C is the number of classes, $\hat{y}_i^c$ is the highest probability of $\hat{\boldsymbol{y}}_i = \left(\hat{y}_i^1, \hat{y}_i^2, \dots, \hat{y}_i^C\right)^T$ and represents class y^c for pixel i:

- If $y_i = y^c$ and $\hat{y}_i^c \geq Thresh_M$, i.e. the pixel i is correctly classified and the probability is higher than the threshold, then the pixel i belongs to the group of strongly and correctly classified pixels.

- If $y_i = y^c$ but $\hat{y}_i^c < Thresh_M$, i.e. the pixel i is correctly classified, but the probability is lower than the threshold, then the pixel i belongs to the group of weakly and correctly classified pixels.

- If $y_i \neq y^c$, then the pixel i belongs to the group of misclassified pixels

The last two kinds of pixels are used to build new training dataset.

4.2.2 Fine-tuning the SkipNet with Hard-Negative Mining

Fine-tuning the pre-trained SkipNet model is to use the target task data to continue the training process on the original pre-trained model. Although the SkipNet has already showed a great generalization ability, we hope to enhance its performance by preprocessing the dataset or modifying the model to pay more attention on the misclassified pixels.

Architecture

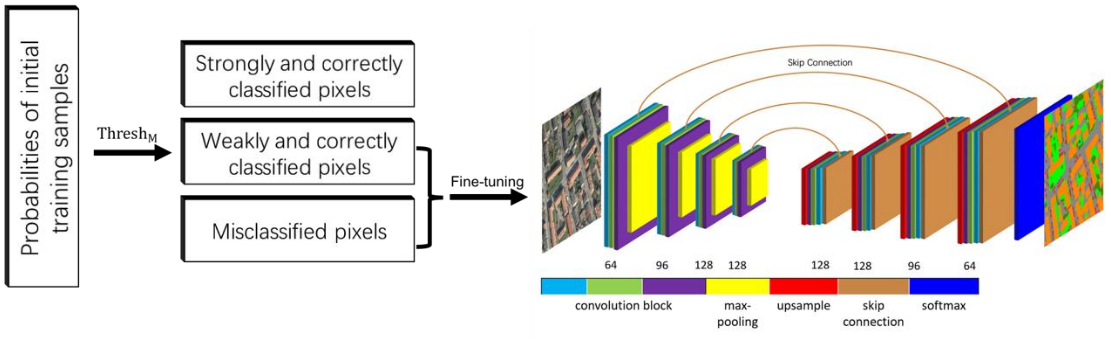

Figure 4-5 Procedure of fine-tuning the pre-trained SkipNet model with hard-negative mining

Two different standards to build hard-negative dataset are implemented by setting different mining thresholds $Thresh_M$. We investigate the impact of different mining thresholds to the model. In fine-tuning, a mask has been implemented on the initial training dataset to exclude the strongly and correctly classified pixels out of training. The mask is generated according to hard-negative mining. Furthermore, we also investigate a fine-tuning method, where the weights vectors of the encoder are frozen, i.e. the encoder cannot be trained. In testing, the overall accuracy of the fine-tuned model may decrease compared to the pre-trained SkipNet model, because the features of hard examples may not very representative for their classes. The predictions of the pre-trained SkipNet model are ensembled with the predictions of the fine-tuned model, which combines the results of the fine-tuned model and the pre-trained SkipNet model by multiplying the probabilistic class scores.

Fine-tuning

For fine-tuning the pre-trained SkipNet model, we employed a stochastic gradient descent optimizer with weight decay 0.0005. Due to the limitations of our GPU, the mini-batch size is set to 1 and the learning rate is set to 0.001 for 15 epochs.

4.2.3 Patch-based Classification: LiteNet

Patch classification is a deep learning method applied to semantic segmentation, where an image is divided into many patches, which are fed to the deep model as input. A patch takes the label of its central pixel as its label. In addition, context information is provided by the rest pixels in a patch and is converted into a feature map by convolutional neural network. In summary, a patch-based convolutional neural network converts the pixel values in an image patch to a one-dimensional class scores, and a softmax classifier is implemented to compute the class probabilities (Paisitkriangkrai et al., 2016). Pooling operation and activation function contain no trainable parameters because they perform a fixed operation, whereas the convolutional and fully connected layers extract and integrate features with neuron weights, which are trained with optimization algorithms so that the class of the highest class score is conformed to match the label in training dataset.

Architecture

Our patch-based convolutional neural network is composed with the strategy of (Krizhevsky et al., 2012), where it constructs a building unit with a convolutional layer, an activation function (ReLU) and a max-pooling layer. The convolutional layer computes the feature map with its convolutional kernel, whereas the activation function (ReLU) adds the non-linearity of the network and the max-pooling layer reduce the spatial resolution of the feature map. These two no parameter operations improve the robustness of the network to distortions and small translations (Paisitkriangkrai et al., 2016). Our network is constructed with four successive building units, continued with two fully connected layers, which integrates the local features into global features, and ended with a softmax classifier to convert the output of the last fully connected layer to the probabilities. The structure is illustrated below:

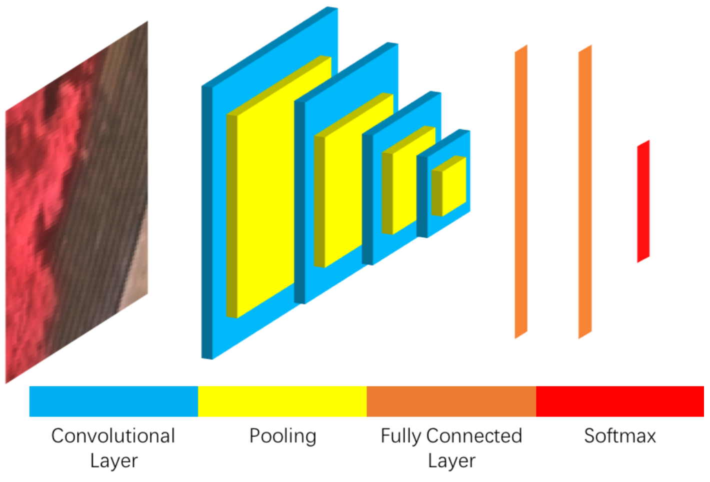

Figure 4-6 Patch-based convolutional neural network architecture according to (Paisitkriangkrai et al., 2016), where the difference of model is the softmax output, which has six output channels.

The inputs of this network are patches of size 64×64. The pooling layer is implemented with max-pooling technique and its kernel size is 3×3, where the stride is 2 pixels. All the convolutional layers in this network works with a stride of 1 pixel and zero padding. The first convolutional layer, whose input is the image with three channels, consists of 32 kernels, where its size is $5 \times 5 \times 3$. The second convolutional layer has 64 kernels of size $5 \times 5 \times 32$. The third convolutional layer is composed of 96 kernels, of which the size is $5 \times 5 \times 64$. The last convolutional layer has 128 kernels with size of $3 \times 3 \times 96$. Both fully connected layers have 128 neurons, followed by a 50% dropout. The last fully connected layer converts the feature vectors into a vector of class scores $\boldsymbol{Z}_i = \left(Z_i^1, Z_i^2, \dots, Z_i^C\right)^{\mathrm{T}}$, where C is the number of classes. For each patch i to be classified, its true label is$y_i \in \{y^1, y^2, \dots, y^C\}$. The output of softmax is $\hat{\boldsymbol{y}}_i = \left(\hat{y}_i^1, \hat{y}_i^2, \dots, \hat{y}_i^C\right)^{\mathrm{T}}$, i.e. the predicted probabilities of pixel i and $\hat{y}_i^c$ is the probability for class y^c. The softmax layer normalizes the class scores into posterior probability $P_i(y_i|x_i)$ for patch i, given the image data x_i:

$$\hat{y}_i^c = P_i(y^c|x_i) = \mathrm{softmax}(\boldsymbol{Z}_i, y^c) = \frac{exp(Z_i^c)}{\sum_{j=1}^{C} exp(Z_i^j)}. \tag{4.3}$$

Training uses mini-batch stochastic gradient descent and back-propagation algorithm optimize the cross-entropy loss function, which is:

$$L = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C} \delta(y_i, y^c) \cdot log(\hat{y}_i^c). \tag{4.4}$$

where $\delta(y_i, y^c)$ is *Kronecker delta*, of which the result is 1 if the variables are equal, and 0 otherwise, and N is the number of patches in a mini-batch.

Training and Inference

The purpose our test is to improve the performance of the existing convolutional neural network model, so the model is trained with the help of hard-negative mining. For training the LiteNet, we employed a stochastic gradient descent optimizer with weight decay 0.0005 and a step learning policy. The mini-batch size is set to 16, while the base learning rate is set to 0.01 and decreased to 0.001 after 15 epochs in a total of 30 epochs training.

In testing, the inputs of this network are several complete images, which are divided into patches by a sliding window approach. Although the sliding window approach can be implemented effectively in the model, it can be very time-consuming due to overlapping. In order to reduce the unnecessary calculations caused by overlapping in testing, bilinear interpolation is added in the model. First, the sliding window is specified with a step size of 4 pixels to evaluate the entire test image, however, the result is not a pixel-level classification as shown below.

$$\begin{bmatrix} \ddots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \cdots & P_{x_1y_1} & np & np & np & P_{x_1y_2} \\ \cdots & np & np & np & np & np \\ \cdots & np & np & np & np & np \\ \cdots & np & np & np & np & np \\ \cdots & P_{x_2y_1} & np & np & np & P_{x_2y_2} \end{bmatrix}$$

Figure 4-7 A simple example of the result processed by the patch-based model, which contains a sliding window with a step size of 4 pixels. "np" represents "not predicted" and "P_{xy}" represents the probabilities.

Secondly, in order to acquire a pixel-level classification, bilinear interpolation of probabilities is implemented to obtain dense predictions, which can be described as below:

$$P(x,y) = \frac{1}{(x_2 - x_1)(y_2 - y_1)} [x_2 - x \quad x - x_1] \begin{bmatrix} P_{x_1y_1} & P_{x_1y_2} \\ P_{x_2y_1} & P_{x_2y_2} \end{bmatrix} \begin{bmatrix} y_2 - y \\ y - y_1 \end{bmatrix}. \quad (4.5)$$

This method significantly alleviates the problem of time-consuming during evaluation, but bilinear interpolation algorithm has properties of the low pass filter, which can damage high frequency components, so it may blur the object boundaries to some extent.

With the dense predictions, which is focused on hard-negative examples, we may combine the results of the pre-trained SkipNet model and the patch-based convolutional neural network by applying ensemble method, where the probabilistic vectors of the pre-trained SkipNet are multiplied with the corresponding probabilistic vectors of the LiteNet model.

4.2.4 Adding Prior-Knowledge of Boundary in SkipNet

The result of the pre-trained SkipNet model has been carefully analyzed, and one special misclassification has drawn our attention. The boundaries between objects, e.g. between *Building* and *Impervious Surfaces,* are not very precise.

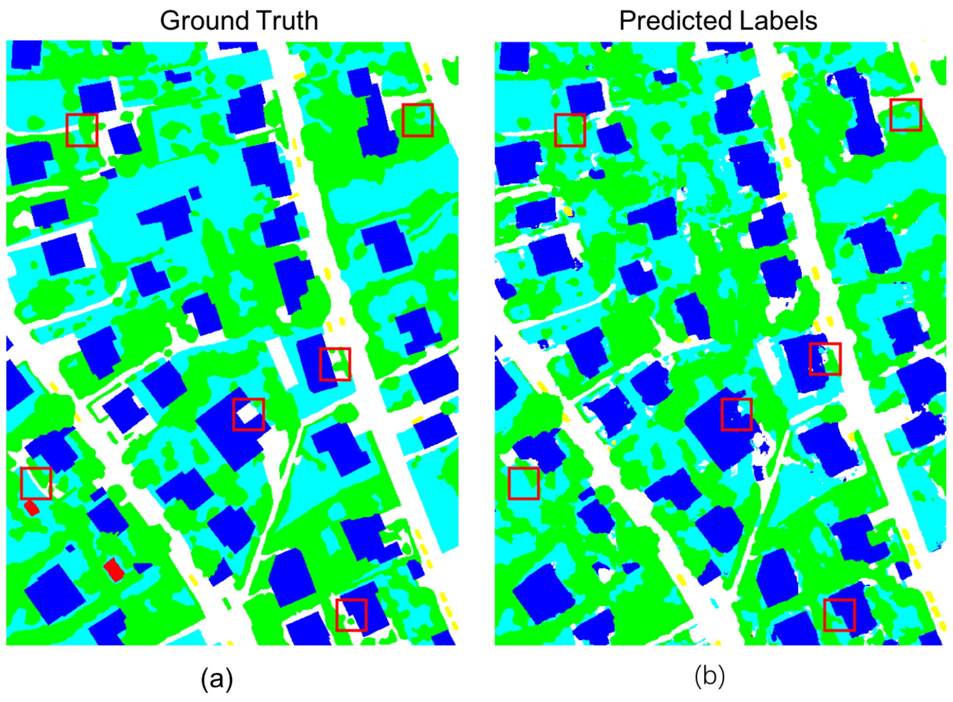

Figure 4-8 Misclassifications of boundaries. (a) is the reference, whereas (b) is the predicted labels.

The third investigation is adding more prior knowledge to the SkipNet. In this case, the prior object boundary information needs to be firstly extracted out from the reference. Afterwards, this knowledge needs to be incorporated into the model by modifying the structure, whereas a research on modification must be taken. Besides, an investigation of the width of boundary should be taken, in order to find out the relationship between performance and the boundary.

Architecture

After the boundaries have been extracted, the model structure needs to be modified to incorporate the knowledge. The SkipNet is a single task classifier, where it only predicts the label of each pixel. Now a new task is implemented with the purpose of the classification of the class boundaries. To achieve this goal, a new softmax layer is parallelly added into the model. The new structure is shown below:

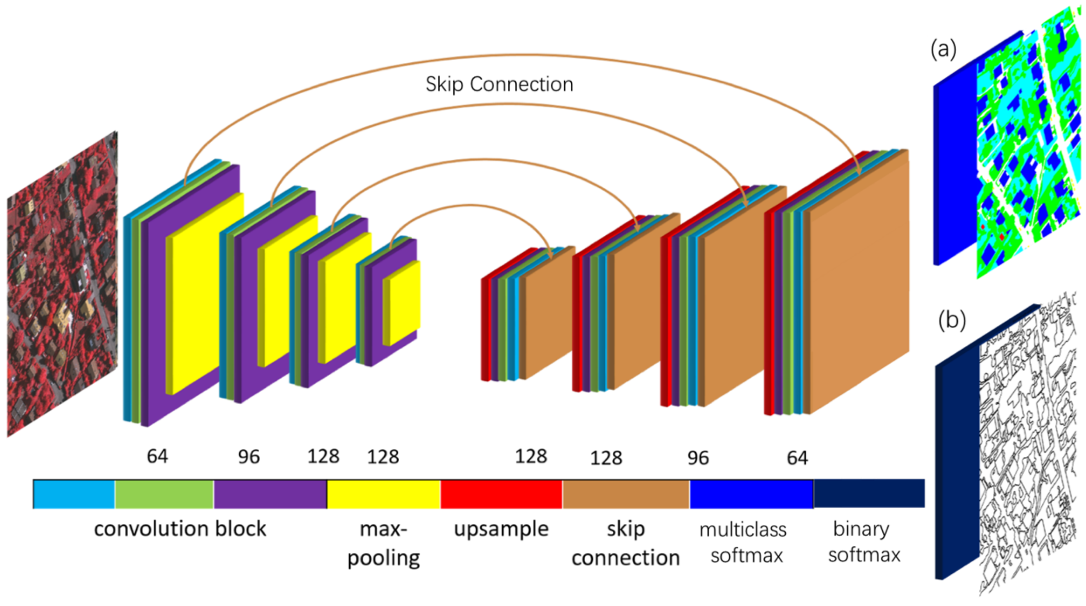

Figure 4-9 Illustration of the dual task model. The dual task model has two classifiers connected to the last layer of decoder, of which (a) classifier still performs the multiclass classification and (b) classifier performs binary classification of boundaries. The input of both classifiers is the dense feature map from the last layer of the decoder part.

In training, the network is fed with the original training dataset, whereas the reference of the class boundaries is automatically generated with the new implemented algorithm.

For multiclass classification, suppose the vector $\boldsymbol{Z}_i = \left(Z_i^1, Z_i^2, \dots, Z_i^C\right)^{\mathrm{T}}$ represents the class scores of the pixel i, where C is the number of classes, $y_i \in \{y^1, y^2, \dots, y^C\}$ is the corresponding true label and Z_i^c is the class score for class y^c. For boundary classification, the vector $\boldsymbol{\mathcal{Z}}_i = (\mathcal{Z}_i^1, \mathcal{Z}_i^2)^{\mathrm{T}}$ represents the class scores of the pixel i. $\mathcal{Y}_i \in \{\mathcal{Y}^1, \mathcal{Y}^2\}$ is the true label, and $\mathcal{Z}_i^c$ is the class score for class $\mathcal{Y}^c$.

The new loss function is the combination of two classifiers, shown as below:

$$L_{multiclass} = -\frac{1}{H \cdot W} \sum_{i=1}^{H \cdot W} \sum_{c=1}^{C} \omega_c \cdot \delta(y_i, y^c) \cdot log\left(\frac{exp(Z_i^c)}{\sum_{j=1}^{C} exp\left(Z_i^j\right)}\right). \tag{4.6}$$

$$L_{boundary} = -\frac{1}{H \cdot W} \sum_{i=1}^{H \cdot W} \sum_{c=1}^{2} \gamma_c \cdot \delta(\mathcal{Y}_i, \mathcal{Y}^c) \cdot log\left(\frac{exp(\mathcal{Z}_i^c)}{\sum_{j=1}^{2} exp\left(\mathcal{Z}_i^j\right)}\right). \tag{4.7}$$

$$L = \alpha \cdot L_{multiclass} + \beta \cdot L_{boundary}, \tag{4.8}$$

where α and β are the new hypermeters to modify the ratio of two loss functions, H and W is respectively the height and width of the input image, and $\omega_c \in \boldsymbol{\omega} = \{\omega_1, \omega_2, \dots, \omega_C\}$ is a class weight computed according to (3.33). $\delta(y_i, y^c)$ and $\delta(\mathcal{Y}_i, \mathcal{Y}^c)$ are *Kronecker delta*, of which the result is 1 if the variables are equal, and 0 otherwise. γ_c is a class weights for binary classification (3.34).

Training and Inference

For training this network, we load the parameters of the encoder, decoder and multiclass classifier from the pre-trained SkipNet and initialize the parameters of boundary classifier normally. We employed a stochastic gradient descent optimizer with weight decay 0.0005. The input size is 256×256 pixels. Due to the limitations of our GPU, the mini-batch size is set to 1. The learning rate is 0.001 for 15 epochs. In testing, the classifier (b) does not output predictions.

5. Experiments

In this chapter, we will test our three architectures using Vaihingen Dataset of the ISPRS 2D Semantic Labeling Challenge (Wegner et al., 2017). First, a brief introduction of the dataset is presented. Then we will share our investigation experiments in two aspects, i.e. hard-negative mining and adding prior knowledge of the boundary. The results of each architecture will be evaluated. Our architectures are implemented based on TensorFlow framework (Abadi et al., 2016). We use a GPU (Nvidia GTX 1060, 3GB) to accelerate the training and inference.

The pre-trained SkipNet model is provided by Institute of Photogrammetry and GeoInformation, where the model is implemented based on TensorFlow framework (Abadi et al., 2016). All results will be compared with the pre-trained SkipNet model and for the sake of brevity, we call it SkipNet-B (B refers to baseline).

5.1 Dataset

The proposed investigations were applied to Vaihingen Dataset of the ISPRS 2D Semantic Labeling Challenge (Wegner et al., 2017), which is an open benchmark dataset provided online. Overall there are over 168 million pixels in 33 images. 16 of 33 are provided with labeled reference for training and validation and the others are withheld for testing. The pixel-based reference was generated by manual labelling and has 6 labels: *Impervious Surfaces (Imper.), Building (Build.), Low Vegetation (Low Veg.), Tree, Car,* and *Clutter/Background.* We take 4 images (image numbers 5, 7, 23, 30) to build our validation datasets and the rest 12 images to build our training datasets for the investigations.

The evaluation is based on overall accuracy (OA), i.e. the percentage of pixels that are assigned the correct class label by the classification process, and the average F1 score, i.e. the average of the harmonic means of the completeness and the correctness per class. In the evaluation of F1 scores, class *Clutter/Background* will be ignored due to that there are very few pixels of class *Clutter/Background* in the validation datasets.

5.1.1 Setup for Fine-tuning the SkipNet-B

The 12 images for building training datasets are divided into tiles of size 256×256 with overlapping of 50%, which corresponds to input size required by our SkipNet variants. These tiles are used for training, whereas the other 4 images are for testing. The tiles in the training dataset are flipped in horizontal and vertical directions and also rotated to $90°, 180°$ and $270°$. There 3,328 tiles before data augmentation and

19,968 tiles afterwards.

According to the probabilistic scores predicted by the SkipNet-B, we separate the pixels of the tiles in the training dataset by the mining threshold $Thresh_M$, cf. 4.2.1, to obtain the hard-negative examples. There are two different mining thresholds $Thresh_M \in \{0.85, 0.95\}$ in our fine-tuning experiment. We fine-tune the SkipNet-B in two different ways: directly fine-tuning and encoder-frozen fine-tuning. Directly fine-tuning means we fine-tune the weights of the encoder and the decoder, whereas the encoder-frozen fine-tuning indicates that we make the weights of the encoder remain unchanged and the weights of the decoder fine-tuned.

In our evaluation, SkipNet-D_{Thresh_M} represents the model is directly fine-tuned with different $Thresh_M$. SkipNet-E_{Thresh_M} refers to the model is encoder-frozen fine-tuned with different $Thresh_M$. We compared different ensembles, where we drop the term SkipNet to denote the classifiers that we combined. For instance, EN (B, D_{Thresh_M}) refers to an ensemble that combines the chosen output of SkipNet-B and SkipNet-D_{Thresh_M}.

5.1.2 Setup for Training the LiteNet

Our LiteNet is based on the architecture according to Paisitkriangkrai et al., 2016. In order to enhance the performance of SkipNet-B, our LiteNet model is arranged to pay more attention to the hard-negative examples. This focus has been reflected in the design of the training datasets.

With the probabilistic scores of 12 training images predicted by SkipNet-B, we classified the pixels with $Thresh_M = 0.95$ to build different training datasets, cf. 4.2.1. The original training dataset of SkipNet-B, cf. 5.1.1, contains low percentage of hard-negative examples, which has been modified at different levels in our new training datasets. The training samples of LiteNet are patches, whose central pixel is selected according to the hard-negative mining as shown below. A patch is generated with the central pixel and its surrounding pixels and its size is 64×64, which corresponds to input size required by LiteNet. There are five new training datasets. The patches in training datasets are flipped in horizontal and vertical directions and also rotated to $90°, 180°$ and $270°$ for data augmentation.

The category of the patch central pixel	Number of patches of every class in the training datasets				
	Dataset V	Dataset W	Dataset X	Dataset Y	Dataset Z
Strongly & correctly classified pixels	15000	15000	15000	30000	0
Weakly & correctly classified and misclassified pxels	0	5000	15000	30000	15000
Total number of patches after data augmentation	540000	720000	1080000	2160000	540000

Table 1 Five training datasets for LiteNet. The category of pixels is cf. 4.2.1.

In our evaluation, LiteNet-V, W, X, Y, Z represent the network variants trained with datasets V, W, X, Y, Z respectively. We compared different ensembles, where we drop the term LiteNet to denote the classifiers that we combined. For instance, EN(B, V) refers to an ensemble that combines the output of SkipNet-B and LiteNet-V. We implement the ensemble of not only probabilistic scores of all pixels, but also a certain Due to that of pixels, whose highest probabilistic score is smaller than a given threshold. This can be regarded as a hard-negative mining for post-processing in predictions in order to find out the probabilistic outputs that need to be combined.

5.1.3 Setup for the Boundary-aware SkipNet

The accuracy inside boundaries of SkipNet-B varies from the width of the boundaries but is significantly lower than the overall accuracy as shown below. The overall accuracy of SkipNet-B is 86.29%

Boundary width [pixel]	5		10		15	
	Inside	Outside	Inside	Outside	Inside	Outside
Accuracy [%]	57.15	89.30	63.81	90.82	68.58	91.80
Pixel percentage [%]	9.40	9.60	16.82	83.18	23.80	76.20

Table 2 Evaluation of accuracies inside or outside the boundaries with different boundary widths based on SkipNet-B. Inside or outside refers to the area inside or outside the boundary.

The 12 large images with reference data of Vaihingen dataset are divided in the same way, cf. 5.1.1. The training dataset is the same as the fine-tuning experiments, whereas the boundary reference is generated according to the reference of the Vaihingen dataset. In this experiment we trained six boundary-aware SkipNet with different classifier weights, i.e. $(\alpha, \beta) \in \{(1,1), (1,0.1)\ \}$, (α, β) cf. 4.2.4.

In our evaluation, we use SkipNetB- $\mathrm{Width}_{(\alpha,\beta)}$ to indicate our network variants. SkipNetB refers to boundary-aware SkipNet, Width refers to the boundary width and Classifier weight refers to (α, β). We compared different ensembles, where we drop the term SkipNet and SkipNetB to denote the classifiers that we combined. For instance, EN(B, $10_{(1,1)}$) refers to an ensemble that combines the chosen output of SkipNet-B and SkipNetB-$10_{(1,1)}$.

5.2 Evaluation of Hard-Negative Mining

In the investigation of hard-negative mining, there are two architectures that will be evaluated, i.e. SkipNet and LiteNet. These two architectures are trained with different training datasets and parameter settings.

5.2.1 Evaluation of Fine-tuning the SkipNet-B

Table 3 presents the land cover classification results for all variants except for ensembles of networks described in section 5.1.1. SkipNet-D has achieved a better performance than SkipNet-E both in F1 scores and OA regardless of the mining thresholds. This may be caused by that not all parameters are trainable in the SkipNet-E. In addition, compared to SkipNet-B, SkipNet-$E_{0.95}$ shows a decrease of 0.24% and 0.30% in average F1 scores and OA respectively, and SkipNet-$E_{0.85}$ shows a decrease of 7.15% and 1.83% in average F1 score and OA respectively. Although the OA of SkipNet-$D_{0.95}$ and SkipNet-$D_{0.85}$ decreased 0.11% and 0.15% respectively, the average F1 scores show a little improvement (0.05% and 0.07% respectively). For SkipNet-$D_{0.95}$, two classes, i.e. *Impervious Surfaces* and *Building* are better classified by a margin of F1 scores (0.68% and 0.56% respectively), compared to SkipNet-B. The reason why F1 scores of these two classes are improved is that the fine-tuning with hard-negative examples increases the ability to distinguish the samples, where the samples of these two classes may be similar as shown in Chapter 1.1. While F1 scores of SkipNet-$D_{0.95}$ are improved for two classes, SkipNet-$D_{0.85}$ achieves a better average F1 score and F1 score of class *car* with 83.35% and 78.49% respectively. After fine-tuning with hard-negative mining, the F1 scores of class *car*, which has fewer training samples (covering 1.40% of the test area), have been improved due to that the percentage of training samples of class *car* has increased.

Network variant	F1 [%]					avg. F1 [%]	OA [%]
	Imper.	*Build.*	*Low Veg.*	*Tree*	*Car*		
SkipNet-B	87.08	92.36	**74.71**	**86.42**	75.81	83.28	**86.29**
SkipNet-$D_{0.95}$	**87.76**	**92.92**	71.77	85.76	78.43	83.33	86.18
SkipNet-$D_{0.85}$	87.64	92.76	72.05	85.78	**78.49**	**83.35**	86.14
SkipNet-$E_{0.95}$	87.51	92.67	71.71	85.64	77.69	83.04	85.99
SkipNet-$E_{0.85}$	85.66	91.81	70.58	84.84	47.77	76.13	84.46

Table 3 Results of land cover classification. Network variant: cf. Section 5.1.1. F1: F1 score, OA: overall accuracy, both evaluated on the basis of pixels. Best scores are printed in bold font.

Table 4 presents the land cover classification results for ensemble variants described in section 5.1.1. We choose network variant SkipNet-D to implement ensemble due to that directly fine-tuning outperformed the other way according to Table 1. It is notably that the F1 of class *Low Vegetation* and class *Tree* is higher, compared to other network variants. OA and average F1 score of EN(B, $D_{0.95}$, $D_{0.85}$) outperformed the other network variants with 86.40% and 83.66% respectively.

Network variant	F1 [%]					avg. F1 [%]	OA [%]
	Imper.	*Build.*	*Low Veg.*	*Tree*	*Car*		
EN(B,$D_{0.95}$)	87.45	92.61	**74.01**	**86.29**	77.66	83.60	86.39
EN(B,$D_{0.85}$)	87.41	92.57	73.95	86.26	77.50	83.54	86.35
EN(B,$D_{0.95}$,$D_{0.85}$)	**87.60**	**92.72**	73.55	86.17	**78.24**	**83.66**	**86.40**

Table 4 Results of land cover classification. Network variant: cf. Section 5.1.1. F1: F1 score, OA: overall accuracy, both evaluated on the basis of pixels. Best scores are printed in bold font.

5.2.2 Evaluation of LiteNet

According to Table 5, our baseline SkipNet has achieved best performance among these network variants. It is in our expectations due to the design of training datasets, because our datasets except dataset V contain much higher percentage of samples, of which features can be hardly summarized. LiteNet-V has the best performance in our LiteNet variants due to that the dataset contains only the samples, of which the features can be easily extracted. However, due to this design, the generalization ability of LiteNet-V is not as good as SkipNet-B.

Network variant	F1 [%]					avg. F1 [%]	OA [%]
	Imper.	*Build.*	*Low Veg.*	*Tree*	*Car*		
SkipNet-B	**87.08**	**92.36**	**74.71**	**86.42**	**75.81**	**83.28**	**86.29**
LiteNet-V	79.26	87.29	68.22	80.90	46.66	72.47	79.42
LiteNet-W	77.81	86.74	66.45	80.43	42.45	70.78	78.24
LiteNet-X	78.26	86.86	67.73	81.21	46.78	72.17	79.02
LiteNet-Y	75.42	85.35	60.14	80.77	47.22	69.78	77.23
LiteNet-Z	11.92	66.04	0.00	71.95	36.63	37.31	51.35

Table 5 Results of Results of land cover classification. Network variant: cf. Section 5.1.2. F1: F1 score, OA: overall accuracy, both evaluated on the basis of pixels. Best scores are printed in bold font.

Table 6 presents the ensembles of LiteNet variants and SkipNet-B. EN(B, Y) shows a little increase (0.04%) in OA, whereas SkipNet-B still has the better average F1 score of 83.28%. In addition, EN(B, Y) outperformed other network variants in F1 scores of class *Impervious Surfaces* and class *Building* with 87.09% and 92.53% respectively, whereas the F1 score of class *Low Vegetation* of EN(B, V) is higher than others with 74.94%. It is remarkably that OA of EN(B, Z) achieves 85.87%, whereas OA of LiteNet-Z is only 51.38%.

Network variant	F1 [%]					avg. F1 [%]	OA [%]
	Imper.	*Build.*	*Low Veg.*	*Tree*	*Car*		
SkipNet-B	87.08	92.36	74.71	**86.42**	**75.81**	**83.28**	86.29
EN(B,V)	86.94	92.47	**74.94**	86.38	74.79	83.10	86.29
EN(B,W)	86.70	92.26	74.77	86.35	74.71	82.96	86.12
EN(B,X)	86.69	92.15	74.92	86.40	74.51	82.93	86.12
EN(B,Y)	**87.09**	**92.53**	74.42	86.40	75.06	83.10	**86.33**
EN(B,Z)	86.47	91.96	73.13	86.39	75.03	82.60	85.86

Table 6 Result of land cover classification. Network variant: cf. Section 5.1.2. F1: F1 score, OA: overall accuracy, both evaluated on the basis of pixels. Best scores are printed in bold font.

We want to investigate the relationship between OA and the threshold for ensembles, which is used similar to the mining threshold. We implement the EN(SkipNet-B, LiteNet) under a certain condition. If the highest probabilistic score of a pixel predicted by SkipNet-B is under the threshold, then we multiply the probabilistic vectors of SkipNet-B and LiteNet. The results are shown below:

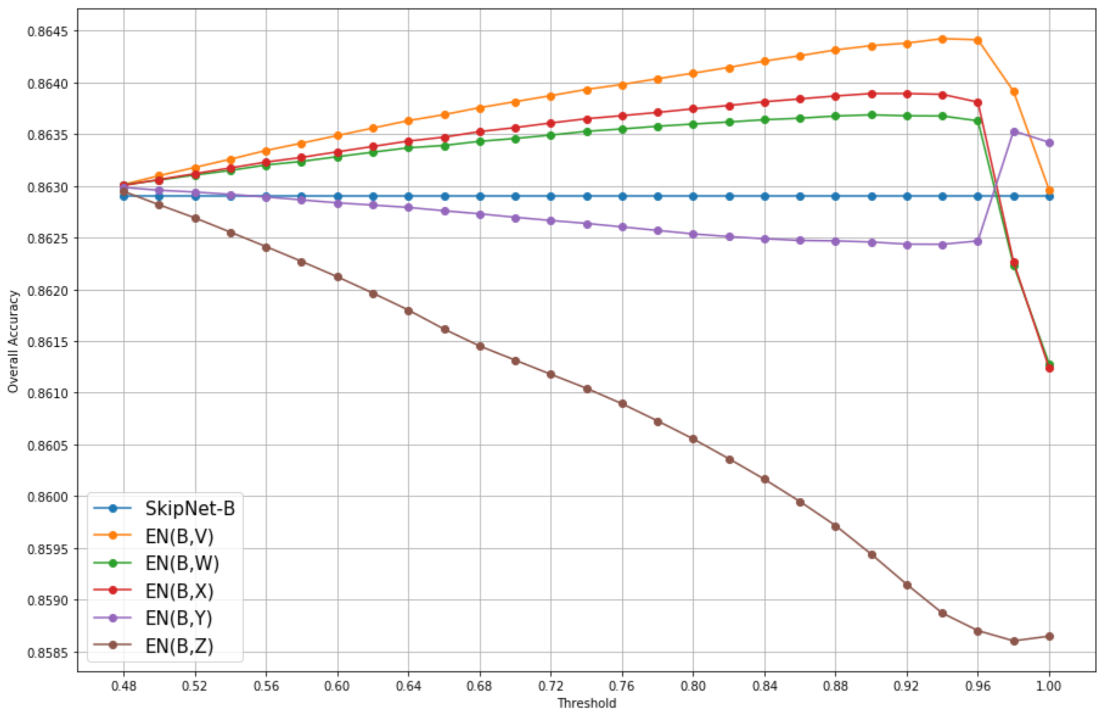

Figure 5-1 Overall accuracies of land cover classification. Network variant: cf. Section 5.1.2.

If the threshold reaches 1.0, all pixels are implemented with ensemble, where OA is the same as Table 5. If the threshold is under 0.47, then no pixels are implemented with ensemble method. According to Figure 5-1, when threshold equals 0.94, OA achieves its peak with 86.44%. Training datasets with high percentage of strongly and correctly classified pixels would achieve a good performance. Training with datasets of high percentage of the hard-negative examples would damage the generalization of the models.

5.3 Evaluation of the Boundary-aware SkipNet

Table 7 presents the land cover classification results for all variants except for ensembles of networks described in section 5.1.3. SkipNet-B still has better F1 scores of class *Low Vegetation* and class *Tree* than other network variants. SkipNetB-10$_{(1,1)}$ has a better average F1 score and OA of 82.93% and 85.48% respectively than other boundary-aware network variants. SkipNetB-5$_{(1,1)}$ outperformed SkipNet-B in the F1 scores of class *Impervious Surfaces* and class *Building* with a little improvement (0.14% and 0.09% respectively). It is notably that the F1 score of class *car* of SkipNetB-15$_{(1,0.1)}$ has increased with 2.48% compared to SkipNet-B.

Network variant	F1 [%]					avg. F1 [%]	OA [%]
	Imper.	*Build.*	*Low Veg.*	*Tree*	*Car*		
SkipNet-B	87.08	92.36	**74.71**	**86.42**	75.81	**83.28**	**86.29**
SkipNetB-5$_{(1,1)}$	**87.22**	**92.45**	72.10	85.11	75.77	82.53	85.47
SkipNetB-5$_{(1,0.1)}$	86.67	91.45	72.24	85.89	77.61	82.77	85.34
SkipNetB-10$_{(1,1)}$	86.67	91.44	72.97	85.97	77.62	82.93	85.48
SkipNetB-10$_{(1,0.1)}$	85.41	91.01	70.59	85.97	77.71	82.14	84.75
SkipNetB-15$_{(1,1)}$	86.10	91.19	68.23	85.30	77.55	81.67	84.60
SkipNetB-15$_{(1,0.1)}$	86.28	91.44	68.00	85.10	**78.29**	81.82	84.67

Table 7 Results of Results of land cover classification. Network variant: cf. Section 5.1.3. F1: F1 score, OA: overall accuracy, both evaluated on the basis of pixels. Best scores are printed in bold font.

According to Table 8, we implement ensembles as shown below. The average F1 score and OA of EN(B,10$_{(1,1)}$) has achieved a little increase of 0.45% and 0.14% respectively, where the F1 scores of class *Low Vegetation,* class *Tree,* and class *Car* are also higher than other network variants. EN(B,5$_{(1,1)}$) outperformed other network variants in F1 scores of class *Impervious Surfaces* and class *Building* with 87.44% and 92.65% respectively.

Network variant	F1 [%]					avg. F1 [%]	OA [%]
	Imper.	*Build.*	*Low Veg.*	*Tree*	*Car*		
SkipNet-B	87.08	92.36	74.71	86.42	75.81	83.28	86.29
EN(B,5$_{(1,1)}$)	**87.44**	**92.65**	74.61	86.29	76.44	83.49	86.39
EN(B,10$_{(1,1)}$)	87.39	92.45	**74.72**	**86.52**	**77.57**	**83.73**	**86.43**
EN(B,15$_{(1,0.1)}$)	87.27	92.46	72.89	86.27	77.42	83.26	86.15
EN(B,5$_{(1,1)}$,10$_{(1,1)}$,15$_{(1,0.1)}$)	87.29	92.39	73.45	86.23	77.56	83.38	86.13

Table 8 Results of Results of land cover classification. Network variant: cf. Section 5.1.3. F1: F1 score, OA: overall accuracy, both evaluated on the basis of pixels. Best scores are printed in bold font.

6 Conclusion and outlook

In our experiments, we have investigated three methods based on SkipNet for the pixel-wise classification of land cover based on aerial images. We compared different variants of the SkipNet, LiteNet and boundary-aware SkipNet architecture. Our experiments have shown that an emsemble of SkipNet and LiteNet achieves the best performance with an overall accuracy of 86.44% for five land cover classes. The other two investigations have also improved overall accuracy of the basis SkipNet architecture with an increase of 0.11% and 0.15% respectively, whereas the overall accuracy of the basis SkipNet architecture is 86.29%.

We have investigated the impact of fine-tuning our basis SkipNet with hard-negative mining and also the mining threshold is set to different levels, meanwhile the SkipNet is fine-tuned in two different ways: directly fine-tuning and encoder-frozen fine-tuning. The results show that directly fine-tuning outperformed encoder-frozen fine-tuning regardless of the mining thresholds. We also investigated an ensemble of SkipNet and LiteNet, where LiteNet is trained with different datasets. These datasets are designed with different percentage of hard-negative examples. In this investigation, we also try to find out the impact of post-processing with hard-negative mining to overall accuracy. In the last experiment, we modified the SkipNet architecture for adding prior knowledge of boundary to it, in order to improve accuracy inside boundaries. The relationship of boundary width and performance has also been investigated.

These three investigations have all improved the performance of the basis SkipNet, but there are still some insufficient. For instance, SkipNet has better performance of class *Low Vegetation* and class *Tree* the most time, except for an ensemble of SkipNet and LiteNet and an ensemble of SkipNet and boundary-aware SkipNet. The improvement of these investigations is not by much. In future work, we may test more mining thresholds to find out the relationship between mining threshold and performance of the architectures using hard-negative mining. In addition, we have the same work for boundary width and performance. In our second experiments, the post-processing with hard-negative mining shows a promising result. This relationship will be further discussed. Besides, we have improved different accuracies of different classes in different network variants. We will further investigate the reasons of the impact and combine the advantage of different network variants.

7 References

Albert, L., Rottensteiner, F. and Heipke, C., 2017. A higher order conditional random field model for simultaneous classification of land cover and land use. ISPRS Journal of Photogrammetry and Remote Sensing, 130, pp. 63-80.

Bartholome, E. and Belward, A.S., 2005. GLC2000: a new approach to global land cover mapping from Earth observation data. International Journal of Remote Sensing, 26(9), pp.1959-1977.

Badrinarayanan, V., Kendall, A. and Cipolla, R., 2015. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. IEEE Transactions on Pattern Analysis and Machine Intelligence, 39(12), pp. 2481-2495.

Burkhard, B., Kroll, F., Nedkov, S. and Müller, F., 2012. Mapping ecosystem service supply, demand and budgets. Ecological Indicators, 21, pp. 17-29.

Foley, J.A., DeFries, R., Asner, G.P., Barford, C., Bonan, G., Carpenter, S.R., Chapin, F.S., Coe, M.T., Daily, G.C., Gibbs, H.K. and Helkowski, J.H., 2005. Global consequences of land use. Science, 309(5734), pp. 570-574.

Girshick, R., Donahue, J., Darrell, T. and Malik, J., 2014. Rich feature hierarchies for accurate object detection and semantic segmentation. In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 580-587.

Glorot, X. and Bengio, Y., 2010, March. Understanding the difficulty of training deep feedforward neural networks. In Proceedings of the thirteenth international conference on artificial intelligence and statistics, PMLR 9, pp. 249-256.

Gong, P., Wang, J., Yu, L., Zhao, Y., Zhao, Y., Liang, L., Niu, Z., Huang, X., Fu, H., Liu, S. and Li, C., 2013. Finer resolution observation and monitoring of global land cover: First mapping results with Landsat TM and ETM+ data. International Journal of Remote Sensing, 34(7), pp. 2607-2654.

He, K., Zhang, X., Ren, S. and Sun, J., 2016. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770-778.

Hermosilla, T., Ruiz, L.A., Recio, J.A. and Cambra-López, M., 2012. Assessing contextual descriptive features for plot-based classification of urban areas. Landscape and Urban Planning, 106(1), pp.124-137.

Hietel, E., Waldhardt R., Otte, A., 2004. Analysing land-cover changes in relation to environmental variables in Hesse, Germany. Landscape Ecology, 19, pp. 473–489.

Hinton, G.E., Srivastava, N., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R.R., 2012. Improving neural networks by preventing co-adaptation of feature detectors. arXiv preprint arXiv:1207.0580.

Krizhevsky, A., Sutskever, I. and Hinton, G.E., 2012. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems, 25(1), pp. 1097-1105.

Längkvist, M., Kiselev, A., Alirezaie, M. and Loutfi, A., 2016. Classification and segmentation of satellite orthoimagery using convolutional neural networks. Remote Sensing, 8(4), pp. 329-350.

LeCun, Y., Bottou, L., Bengio, Y. and Haffner, P., 1998. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), pp. 2278-2324.

Li, C., Wang, J., Wang, L., Hu, L. and Gong, P., 2014. Comparison of classification algorithms and training sample sizes in urban land classification with Landsat thematic mapper imagery. Remote Sensing, 6(2), pp. 964-983.

Liu, F., Shen, C. and Lin, G., 2015. Deep convolutional neural fields for depth estimation from a single image. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 5162-5170.

Long, J., Shelhamer, E. and Darrell, T., 2015. Fully convolutional networks for semantic segmentation. In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 3431-3440.

Lu, D. and Weng, Q., 2007. A survey of image classification methods and techniques for improving classification performance. International journal of Remote sensing, 28(5), pp.823-870.

Jin, S., Yang, L., Danielson, P., Homer, C., Fry, J. and Xian, G., 2013. A comprehensive change detection method for updating the National Land Cover Database to circa 2011. Remote Sensing of Environment, 132, pp. 159-175.

Marmanis, D., Schindler, K., Wegner, J.D., Galliani, S., Datcu, M. and Stilla, U., 2018. Classification with an edge: Improving semantic image segmentation with boundary detection. ISPRS Journal of Photogrammetry and Remote Sensing, 135, pp.158-172.

Meyer, W., Turner, B, 1994. Changes in land use and land cover: a global perspective, volume 4.

Noh, H., Hong, S. and Han, B., 2015. Learning deconvolution network for semantic segmentation. In Proceedings of the IEEE international conference on computer vision, pp. 1520-1528.

Paisitkriangkrai, S., Sherrah, J., Janney, P. and van den Hengel, A., 2016. Semantic labeling of aerial and satellite imagery. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, 9(7), pp.2868-2881.

Rowley, H.A., Baluja, S. and Kanade, T., 1998. Neural network-based face detection. IEEE Transactions on pattern analysis and machine intelligence, 20(1), pp.23-38.

Rowley, H., Baluja, S. and Kanade, T., 1998, June. Rotation invariant neural network-based face detection. In Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, p. 38.

Sherrah, J., 2016. Fully convolutional networks for dense semantic labelling of high-resolution aerial imagery. ArXiv:1606.02585.

Sermanet, P., Chintala, S. and LeCun, Y., 2012, November. Convolutional neural networks applied to house numbers digit classification. Pattern Recognition (ICPR), 2012 21st International Conference, pp. 3288-3291.

Sharif Razavian, A., Azizpour, H., Sullivan, J. and Carlsson, S., 2014. CNN features off-the-shelf: an astounding baseline for recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition workshops, pp. 806-813.

Sharma, A., Liu, X. and Yang, X., 2018. Land cover classification from multi-temporal, multi-spectral remotely sensed imagery using patch-based recurrent neural networks. Neural Networks.

Simonyan, K., Zisserman, A., 2015. Very deep convolutional networks for large-scale image recognition. In: International Conference for Learning Representations.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V. and Rabinovich, A., 2015. Going deeper with convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 1-9.

Xie, S. and Tu, Z., 2015. Holistically-nested edge detection. In Proceedings of the IEEE international conference on computer vision, pp. 1395-1403.

Wegner, J.D., Rottensteiner, F., Gerke, M., Sohn, Gunho, 2017. The ISPRS labeling challenge. Available in the WWW: http://www2.isprs.org/commissions/comm3/wg4/semantic-labeling.html (accessed 11/01/2019).

Yang, C., Rottensteiner, F. and Heipke, C., 2018. Classification of land cover and land use based on convolutional neural networks. ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences 4 (2018), 4(3), pp. 251-258.

List of Figures

List of Tables