Classification Of Polarimetric SAR Data By Complex-Valued Convolutional Neural Networks

I-Feng Lin[1], Maxim Spur[1], and Ronny Hänsch[1]

TU Berlin, Str. des 17. Juni 135, Berlin 10623, Germany,

Abstract. Convolutional Neural Networks (ConvNets) are, among other things, powerful pattern recognition tools. In the past years they were successfully applied to many different classification problems and are increasingly used in many domains, including land use analysis from optical aerial and satellite imagery. Polarimetric SAR (PolSAR) imaging is another popular source of geographic information with advantages over optical images, but could previously not be directly analyzed with ConvNets, since these were designed for the real domain only. This paper provides an overview of complex-valued ConvNets (CVCNNs) and introduces their adaptation to classifying PolSAR data with a demonstration on a real-world example.

Keywords: PolSAR, classification, machine learning, complex valued convolutional neural networks

1 Introduction

1.1 PolSAR

In contrast to conventional optical cameras, Synthetic Aperture Radar (SAR) is an active method of gathering remote sensing information. The microwave signals it sends out to the ground to be scattered back are impervious to weather and time of day, making it a comparably robust technique in earth observation.

Polarimetric SAR (PolSAR) is a refinement of that technique in that it uses different polarizations for both transmission and reception of radar signals, thereby gathering additional information about the scattered microwaves which would otherwise be unavailable or hard to distinguish. The most common version of it does this by sending out orthogonally polarized waves. The incoming reflections are again polarized in the same way, which results in a vector of four measurements:

$$\mathbf{k} = (S_{HH}, S_{HV}, S_{VH}, S_{VV})$$

where S_{TR} is a complex-valued measurement of the reflected signal with both polarizations during transmission (T) and reception (R) indicated by the index. This vector is usually reduced to three dimensions by assuming a redundancy of S_{HV} and S_{VH} when the waves are backscattered from natural targets.

One more transformation to that data is often done before further analysis, which is forming a complex covariance matrix from n neighboring pixels as described here:

$$\mathbf{C} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{k}_i$$

This covariance matrix has six unique entries, so the final PolSAR data used for analysis in the complex domain consists of vectors containing six complex numbers.

1.2 Classification Task

The goal of automatically interpreting PolSAR data is pixel-wise classification, i.e. assigning one class label to each pixel, like agricultural area, forest area, urban area, water and so on, based on features extracted from the neighborhood of this pixel.

There are many solutions to this kind of task, like Support Vector Machines (SVM, [1]), boosting [2], or Multi-Layer Perceptrons (MLP, [3]), but they all require the complex-valued PolSAR data to be projected into the real domain first, since these classifiers were designed only for real-valued data. Even when exploiting particular characteristics of PolSAR, this projection potentially leads to a loss of information.

That creating a classifier which natively works with complex-valued data throughout its whole processing chain is advantageous was recently shown in [4]. The authors adapted MLPs to the complex domain, and in using them to classify PolSAR data, these complex-valued networks of MLPs (CV-MLPs) performed significantly better than real-valued ones, which relied on pre-processed and projected data.

ConvNets have also recently been shown to be a good fit for semantic land use labelling [5], albeit for optical images. This paper builds on this success and adapts their approach to the complex-valued data of PolSAR.

1.3 ConvNets

ConvNets are an advancement over fully connected neural networks or multilayer perceptrons (MLPs), in that their hidden layers have receptive fields. These layers are either convolutional or subsampling layers, with only the last layers being fully connected. The convolutional layers create feature maps, which detect patterns in smaller regions of the input, while the subsampling (or pooling) layers reduce the size of the convolutional layers' output. Their output in turn can then be used as input for another convolutional layer.

Units of these feature maps all share the same weights, which in addition to fewer connections caused by their limited receptive fields leads to a great reduction in the number of parameters necessary to describe such a network. This model also more closely resembles biological neural networks and has been shown to be very successful in e.g. image recognition tasks [6].

A limitation of the underlying neural network architecture is its dependence on particular activation functions of its neurons, which have to be non-linear, bounded, analytical and not constant. Such functions only exist for real numbers. Since PolSAR data resides in the complex domain, for which the only bounded and analytical functions are constant and therefore useless for learning tasks. To work around this issue, different activation functions that do not completely adhere to these rules may be used i.e., not analytical or not bounded functions. This has been shown to work if the input data is properly scaled and weights are not allowed to grow beyond a certain limit [7] [4].

Other research approached this problem by constructing a generalization of the real-valued ConvNet model, which is fully applicable to the complex domain with complex-valued versions of all necessary blocks: convolution, activation and backpropagation [8].

Building on these solutions to handle complex-valued data in ConvNets, this paper presents an approach to apply them to semantic pixel labelling of PolSAR data.

2 Structure

2.1 Architecture

Adopting a simplified version of the successful approach in [5], we use a ConvNet that takes a 16×16 pixel large neighborhood around the pixel that is to be classified as its input. Each of these pixels contains six channels of complex-valued data, which is calculated by the RAT software [9] from raw PolSAR data by taking the unique elements of the covariance matrix of a 3×2 pixel window.

The network consists of two convolutional layers, two fully-connected layers, and a 5-way classifier. The convolutional layer has three functions: convolution, activation, and pooling. Convolution sums the weighted inputs in a spatial area, implemented as kernels defined by x- and y-coordinate and channels, to be its output. Activation function will be discussed later in this paper. Pooling is max-by-magnitude function in this paper.

The inputs of the network are in dimension $16 \times 16 \times 6$. The first convolutional layer has 32 kernels of size $3 \times 3 \times 6$ with a stride of 1 pixel. The max-by-magnitude pooling function shrinks the dimension of inputs by 3×3 in x- and y-direction. Hence the outputs of this layer are in $5 \times 5 \times 32$ The second convolutional layer has 128 kernels of size $3 \times 3 \times 32$ with a stride of 1 pixel with the same pooling function and gives 128 singleton outputs. They are followed by two fully-connected layers that have 128 neurons and then a 5-way classifier.

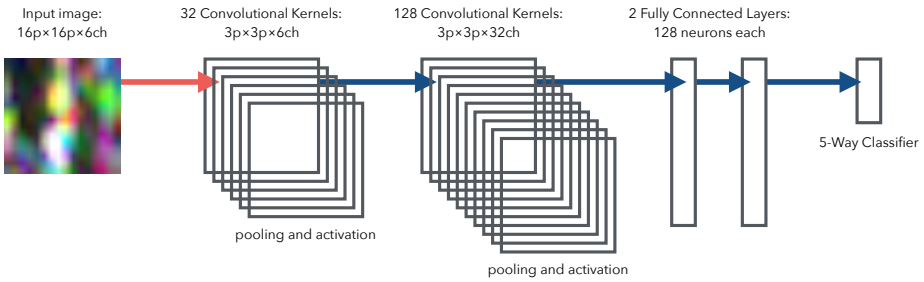

Fig. 1. The network consists of two convolutional layers and two fully-connected layers with a final 5-way classifier. The first convolutional layer has 32 kernels of size $3 \times 3 \times 6$ with a stride of 1 pixel. The second convolutional layer has 128 kernels of size $3 \times 3 \times 32$ with a stride of 1 pixel, followed by two fully-connected layers that have 128 neurons, then the 5-way classifier.

2.2 Activation Function

The activation function in convolutional layers was adopted from [4]. There, a complex-valued version tanh function is used because it is bounded and its gradient can be calculated with generalized complex derivative:

$$f(\mathbf{z}) = tanh(real(\mathbf{z})) + tanh(imag(\mathbf{z}))$$

2.3 N-way classifier

An N-way classifier classify the outputs of fully connected layers into classes. The N inputs of the classifier correspond to N classes. In order to determine how "wrong" an input is, target values of classes are preassigned in this paper. Every class has a target value and hence the distance between the inputs and their target values exist (square root of the magnitude of their difference.) They are their "loss" (negative score) of their corresponding classes. The class whose corresponding input has the least loss, i.e. the shortest distance to its target value, is the estimated class by the classifier.

Simply, we can assign $1 + 1i$ to the target value of the ith unit in a complex N-way classifier if the true class is class i and $0 + 0i$ otherwise. In this paper, we assign every class with different target values, which all distance 1 to $0 + 0i$. This approach is equivalent and provides a better visualization during training.

3 Training

3.1 Error Function

An error function, or loss function, is used in neural networks to evaluate the outputs given inputs and their true classes. The network is trained to minimize that

error. Here, we used the complex quadratic error functions, as [4] demonstrated it to perform well in a neural network.

Given all the weights $\mathbf{W}$ in the network, inputs $\mathbf{z}$. The output of the ith unit of the N-way classifier is $f(\mathbf{W}, \mathbf{z})_i$. The target value of class i is y_i, i.e. the ith unit contributes none to the error when $f(\mathbf{W}, \mathbf{z})_i = y_i$. The error function is formulated as:

$$L(f(\mathbf{W}, \mathbf{z}), \mathbf{y}) = \sum_{i=1}^{N} \tfrac{1}{2} \epsilon_i \bar{\epsilon_i} \text{ where } \epsilon_i = y_i - f(\mathbf{W}, \mathbf{z})_i$$

In addition, when the output is close enough to the target value within a hyperparameter threshold Δ, it can be considered as 0 to avoid over-fitting. Choice of threshold can have a big influence in classification performance. [10] Introduced with Δ the complex quadratic function becomes:

$$L(f(\mathbf{W}, \mathbf{z}), \mathbf{y}) = \sum_{i=1}^{N} \max(0, \tfrac{1}{2} \epsilon_i \bar{\epsilon_i} - \Delta) \text{ where } \epsilon_i = y_i - f(\mathbf{W}, \mathbf{z})_i$$

3.2 Backpropagation

Backpropagation is the mostly used method to adjust the weights $\mathbf{W}$ in neural networks. Gradient-based learning methods are an often successful approach in machine learning with backpropagation. As mentioned in section 3.1, the network is trained to minimize the loss given a set of inputs and weights by updating the weights. Rather than changing the weights randomly, the method update them with the gradient of loss function w.r.t weights. Since the weights exist in layers in metwork, the gradients are propagated back from the last layer to the first for learning. Among gradient-based learning methods, Stochastic gradient descent (SGD) is a popular method. An traditional (batch) SGD updates the weights with the sum gradient from the whole training set in iterations (epochs). This sequential approach is effective but challenging for distributed computing. On the other hand, an online SGD updates the weights with the gradient from a training sample before taking another. Online SGD may suffers from unstable changing resulted from a single sample. A practical solution is to employ mini-batch training. In each epoch, the training samples are aggregated as a mini-batch. The weights are updated with the sum gradient of a mini-batch. [11]

Mini-batch SGD is used in this paper. In one epoch there are 30 mini-batches of size 5000 samples, consisting of 1000 sample from 5 classes respectively. The whole training set has $30 \times 1000 \times 5 = 150,000$ samples, randomly chosen from source data. In each iteration the samples are randomly aggregated in batches. In this paper, every epoch is followed by a mini-test of 5000 samples to evaluate the intermediate status and generate learning curves of the network.

Given weights $\mathbf{W}^t$ at epoch t, the gradient of the error function w.r.t. W denoted as $\nabla_W L$, and the size of a mini-batch I, the weights after training is denoted as $\mathbf{W}^{t+1}$

$$\mathbf{W}^{t+1} = \mathbf{W}^t + \alpha \nabla_W \sum_{i \in I} L(f(\mathbf{W}, z_i), y_i)$$

where α is a hyperparameter learning rate

Learning rates decide how much the network learns from given data. While fast training is desired, large learning rates may result in a non-converging network. In practice, decaying learning rates over epochs gives both efficiency and stability. The learning rate of fully-connected layers starts at 10^{-6} and the decaying rate is 5% every epoch. Owing to the backward-pooling and backward-convolution, the gradient received in convolutional layers is much larger and therefore the learning rates of them should be significantly smaller.

In one set of training parameters, the learning rate of convolutional layers are set 0, i.e. the weights are not updated. In the other set, the learning rate of the second convolutional layer starts at 10^{-9} and that of the first starts at 10^{-12}.

As the name suggests, the gradient is calculated starting from the last layer of the network and propagated through layers back to the first. The complex derivative of the error function is the input of backpropagation. The gradient of the complex quadratic error function is:

$$\nabla_W L(f(\mathbf{W}, \mathbf{z}), \mathbf{y}) = \begin{cases} \sum_{i=1}^{n} \frac{1}{2} \bar{\epsilon_i} & \text{if } i \text{ is the true class} \\ \frac{1}{2} \bar{\epsilon_i} & \text{otherwise,} \end{cases}$$

where n is the number of classes such that $\epsilon_i \bar{\epsilon_i} \neq 0$.

Then a threshold Δ mentioned in section 3.1 is introduced. An gradient is applied to the class only when its error exceeds Δ to avoid overfitting.

$$\nabla_W L(f(\mathbf{W}, \mathbf{z}), \mathbf{y}) = \begin{cases} 0 & \text{if } \epsilon_i \bar{\epsilon_i} < \Delta \\ \sum_{i=1}^{n} \frac{1}{2} \bar{\epsilon_i} & \text{if } i \text{ is the true class} \\ \frac{1}{2} \bar{\epsilon_i} & \text{otherwise,} \end{cases}$$

where n is the number of classes such that $\epsilon_i \bar{\epsilon_i} > \Delta$.

The gradient in fully-connected layers can be easily calculated. The gradients in the convolutional layers go through three functions: pooling function, activation function, and convolution function. In pooling, the gradient is passed to the unit that is activated during forward propagation. In the activation function, the gradient is calculated as following:

$$\nabla_W ReLU(\mathbf{Z}) = \begin{cases} (1+1i)\mathbf{Z} & \text{if real}(\mathbf{Z}),\text{imag}(\mathbf{Z}) > 0 \\ 0 & otherwise \end{cases}$$

In the convolution function, the gradient is passed to those connected units in the kernels.

Δ is set 10^-1 in this paper.

3.3 Results and Discussion

Implementing the above in MATLAB we were able to train a network that shows significant success rates. Due to the high computational intensity involved in training, testing and evaluating, we did not exhaust all possibilities in different architectures, but instead opened a perspective for further research and refinement. The figures and tables below outline different parameters that were used and graphical representations of their outcomes.

Network A training	parameters
No. of samples in training set	150,000
No. of Epochs	10
No. of Batches	30
No. of samples in a batch	5000
Learning rate of conv. layers	0
Learning rate of affine layers	10^{-6}
Decaying rate of learning rate	5%

Network A testing	results
No. of samples	135,025
Overall correctness	56.62%
Correctness of city	68.12%
Correctness of field	70.32%
Correctness of forest	64.03%
Correctness of grass	37.34%
Correctness of street	43.27%

Table 1. training and testing parameters of Network A

Network B training	parameters
No. of samples in training set	25,000
No. of Epochs	8
No. of Batches	10
No. of samples in a batch	500
Learning rate of conv. layer 1	10^{-12}
Learning rate of conv. layer 2	10^{-9}
Learning rate of affine layers	10^{-6}
Decaying rate of learning rate	5%

Network B testing	results
No. of samples	135,025
Overall correctness	29.60%
Correctness of city	6.84%
Correctness of field	2.59%
Correctness of forest	41.90%
Correctness of grass	91.95%
Correctness of street	4.76%

Table 2. training and testing parameters of Network B

Network A is a convolution-augmented fully-connected neural network, which has no learning in convolutional layers.

Network B is a convolutional neural network. Owing to the heavy calculation in backpropagation, a smaller training set is used and the evaluation shows inferior results. However, the learning curve still increases in later epochs and we believe that a bigger training set and more epochs would result in better performance.

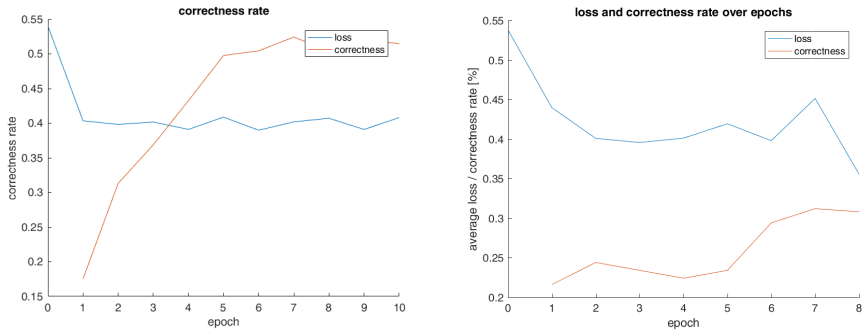

Fig. 2. average loss and correctness rate of a fixed subset of testing set after every epoch of Network A (left) and Network B(right)

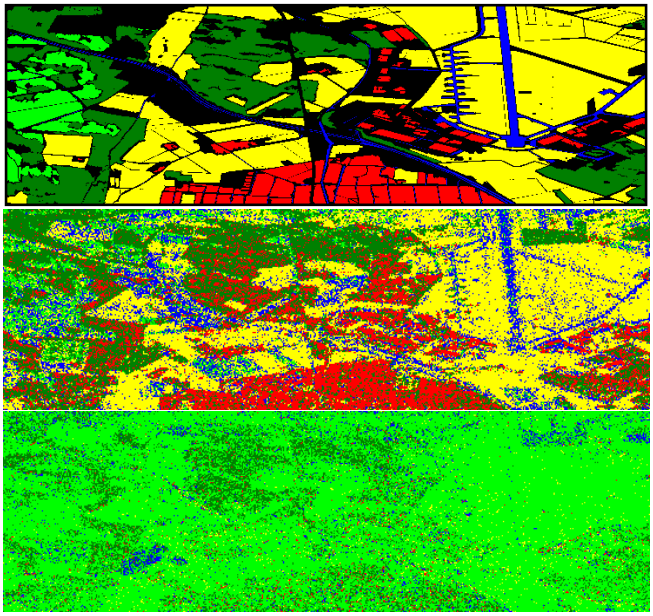

Fig. 3. Pre-labeled data (top) and data labeled by the trained network A (middle) and Network B (bottom) in stride of 4 pixels

References

1. Schoelkopf, B and Smola, A.J.,: Learning with Kernels - Support Vector Machines, Regularization, Optimization, and Beyond. MIT Press, ISBN-10:0-262-19475-9, 2001

2. Duda, R.O., Hart, P.E. and Stork, D.G.: Pattern Classification. Wiley-Interscience; 2nd Edition, ISBN-10: 0471056693, 2000
3. Bishop, C.M.: Neural Networks for Pattern Recognition. Oxford University Press, ISBN10: 0198538643, 1995
4. Hänsch, R. and Hellwich, O.: Classification of Polarimetric SAR data by Complex Valued Neural Networks
5. Paisitkriangkrai, S., Sherrah, J., Janney, P and Hengel, A. V.: Effective Semantic Pixel Labeling Convolutional Networks and Conditional Random Fields. CVPR, 2015
6. Krizhevsky, A., Sutskever, I. and Hinton, G.E.: Imagenet classification with deep convolutional neural networks. NIPS, 2012
7. Hirose, K: Postperovskite phase transition and its geophysical implications, 2006
8. Guberman, N: On Complex Valued Convolutional Neural Networks, 2016
9. Reigber, A. and Hellwich, O.: RAT (Radar Tools): A free SAR image analysis software packagez
10. Rong-En Fan and Chih-Jen Lin: A Study on Threshold Selection for Multi-label
11. Mu Li, Tong Zhang, Yuqiang Chen, Alexander J. Smola: Efficient Mini-batch Training for Stochastic Optimization