UiT
THE ARCTIC UNIVERSITY OF NORWAY

Department of Physics and Technology

Reconstruction of the full-polarimetric covariance matrix from compact-polarimetric synthetic aperture radar data with convolutional neural networks

—

Umberto Bollo Del Rio

FYS-3941 Master's thesis in applied physics and mathematics

Abstract

The focus of this thesis is to find an alternative way to reconstruct a pseudo quadrature polarimetric (quad-pol) covariance matrix from compact polarimetric (compact-pol) data.

In the latest years, the compact polarimetry SAR mode was developed and used more and more widely. It provides a good compromise between area covered and information content per pixel [13].
The literature has focused for a long time on quad-pol data in the past. They contain more information compared to compact-pol data. Moreover, several ways to extract useful information from quad-pol SAR images have been developed [8].

Compact-pol data can be considered as a lossy compression from quad-pol data, which has inspired research to find ways to reconstruct the latter format from the former. This allows to apply all the methods and algorithms developed for data analysis of quad-pol data to a reconstructed pseudo quad-pol data.

The elaboration of more and more effective deep learning techniques in the last few years has guided us to consider convolutional neural networks (ConvNets) a suitable tool for our problem. ConvNets take advantage of the properties of grid-like topology data [7]. They are able to locate spatial and time local connections.

After making assumptions of reflection symmetry for the polarimetric covariance matrix, the reconstruction problem can be formulated as the regression from an image 224x224 with 4 channels, representing the compact-pol covariance matrix, to an image 224x224 with 5 channels, representing the quad-pol covariance matrix. This is the reason why we thought that ConvNets could be a good choice from the available suite of machine learning algorithms.

Our results were then compared with previous reconstruction methods, Souyris and Nord's [6, 37], applying the same data set. The methods developed in this thesis showed, on average, slightly worse results than those in the literature. However, we observed that, in same cases, they produced interesting outcomes, for example, a good generalization ability.

Contents

1 Introduction **1**

1.1 Previous works . . . 2

1.2 Objective and contribution . . . 3

1.3 Structure of the thesis . . . 3

2 Remote sensing images **5**

2.1 Synthetic aperture radars (SAR) . . . 5

2.1.1 Polarization . . . 6

2.2 The scattering coefficients . . . 7

2.3 Polarimetric covariance matrix . . . 8

2.3.1 Reflection symmetry . . . 10

2.4 Compact polarimetry . . . 10

2.4.1 Compact polarimetry modes . . . 12

2.4.2 Compact polarimetric covariance matrix . . . 13

2.4.3 Compact polarimetric information extraction . . . 14

2.4.4 Issues with the reconstruction of polarimetric information . . . 14

2.5 Reconstruction methods . . . 15

2.5.1 Souyris' reconstruction method . . . 15

2.5.2 Nord's reconstruction method . . . 17

3 Deep learning **19**

3.1 Artificial neural networks . . . 19

3.1.1 Modeling one neuron . . . 20

3.1.2 Layer structure . . . 22

3.1.3 Setting the dimension of the network: number of layers . . . 23

3.1.4 Loss function . . . 24

3.1.5 Backpropagation algorithm . . . 25

3.2 Convolutional neural networks . . . 26

3.2.1 The Convolution operation . . . 27

3.2.2 Pooling . . . 30

3.2.3 Some theoretical aspect about the implementation of convolution in neural networks . . . 32

3.2.4 ConvNet applications . . . 34

4 Methods **37**

4.1 Data structure and interpretation . . . 38

4.1.1 Sea ice test image . . . 39

4.1.2 Vegetation test image . . . 39

4.2 Calibration . . . 40

4.3 Covariance matrix calculation . . . 41

4.4 Data set configuration . . . 42

4.5 Compact-pol data . . . 43

4.6 Reconstruction methods: Souyris and Nord . . . 44

4.7 Convolutional neural network as a reconstruction method . . . 44

4.7.1 Caffe . . . 45

4.7.2 Implementation . . . 46

5 Results **49**

5.1 Error measure and comparison method 49

5.2 Souyris and Nord's methods performance over our data set 50

5.3 ConvNet method performance 51

5.3.1 1st ConvNet implementation 51

5.3.2 2nd ConvNet implementation 56

5.4 Summary of the results 62

6 Conclusions and further work **65**

1 Introduction

In the latest years, the development of more efficient and precise techniques to watch over the vast natural surfaces (e.g. seas, oceans, forests, lands) by satellite images allows analysts to have constant observations of the state of the Earth. This is very important to monitor climatic changes and useful for the mapping of natural resources [8].
The observation of wide and unpopulated areas can be done efficiently by space-borne microwave synthetic radar (SAR) sensors. They have the advantage of being independent of the local weather conditions, since the microwaves can propagate unaffected through the clouds, and they are not dependent on light conditions [2].
There are different configuration of SAR instruments. Depending on the information needed, the various modes have different resolution and spatial coverage (swath width and number of polarimetric channels). There is a trade-off between resolution and spatial coverage: high resolution will have a low spatial coverage and vice versa [5].
The quadrature-polarimetric (quad-pol) SAR mode, also known as full-polarimetric mode, is used to get the largest amount of polarimetric information about the back-scattering properties of the targets. It uses dual polarization at the transmitter (horizontal and vertical) and dual polarization at the receiver. The issue of this method is that the coverage area is small. Dual-polarimetric (dual-pol) SAR mode is used to cover a larger area, but it gives less information about the type and state of target surface objects than quad-pol modes. It uses single polarization at the transmitter and double at the receiver [11].
The Canadian RADARSAT-2 SAR sensor has a maximum swath of 50 km using the wide quad-pol mode, and the highest resolution is 5.2 m [5].
To have at the same time the polarimetric information content of quadratic-pol and the area coverage of dual-pol, the compact polarimetry SAR mode was introduced in the last decade [1]. The compact polarimetry consists of a SAR acquisition mode where only one polarization is transmitted, and two orthogonal polarizations are received. Unlike dual-polarimetric systems the polarization for the transmitter could be either circular or diagonal, instead of being horizontal or vertical. That allows the partial reconstruction of quad-pol data. Many research communities have concluded that the compact-pol mode is almost as good as quad-pol SAR systems for applications that look at natural terrain and incoherent scattering, where the underlying assumptions of compact-pol reconstruction are fulfilled[1, 6, 4]. We can assert that this last method will be used in a large scale in the near future, so it will be particularly interesting to find the best reconstruction to extract as much information as possible from compact-pol data.

The development of deep learning techniques has allowed solving problems that have resisted the best attempts of the classical machine learning methods for many years. It has turned out that deep learning is very good in application involving intricate structure in high-dimensional data and is therefore applicable to many domains of science [7]. In [7] the authors explain how they think that deep learning will have many more successes in the near future because it requires very little engineering by hand, aided furthermore by the increase of computational power and available data. New learning algorithms and architectures that are currently being developed for deep neural networks will only accelerate this progress.
LeCun, Bengio and Hinton noticed also that there was one particular type of deep, feed-forward network that was much easier to train and generalized much better than networks with full connectivity between adjacent layers. This was the convolutional neural network (ConvNet). It achieved many practical successes during the period when neural networks were out of favor and it has recently been widely adopted by the computer vision community and many others [7].
ConvNets are a specialized kind of neural network for processing data that have a known, grid-like topology, for example a color image composed of three 2D arrays. They employ convolution operation instead of general matrix multiplication [3].

For these reasons we thought it would be intuitively a promising idea to apply convolutional networks to the SAR image reconstruction problem.

1.1 Previous works

Concerning the quad-polarimetric data reconstruction, we studied and compared other methods developed before ours.

Reference [8] is a master thesis, submitted to UiT The Arctic University of Norway by Martine Mostervik Espeseth. It was used as a guide to work on satellite images, that composed our data set. The quad-pol data interpretation methods are defined, from the scattering vector to the creation of covariance matrices. Descriptions of peculiarities of sea ice images are also described, for example symmetry assumptions of the corresponding covariance matrices. In the end some reconstruction methods are indicated, with reference to them.

References [6] and [37] describe Souyris and Nord's methods. We applied these two methods to our data set as a comparison.

Reference [9] gave us some ideas on how to build a ConvNet. This is a paper written at the University of Berkeley, which is describing ConvNets as an appropriate solution to our problem. It had also helped to choose the methodology to follow. As suggested there, we used Caffe [35], a tool developed at the University of Berkley too, which allow to build complex neural network. [3] assisted us to enhance theoretical aspects.

1.2 Objective and contribution

The aim of this work is to find an alternative solution to the reconstruction problem from compact-pol to quad-pol data format. This was achieved by taking quad-pol covariance matrices, compressing them to compact-pol form, and then developing a method to expand back to a quad-pol approximation.
The reconstruction methods of previous literature are good attempts at solving this problem, but they are only effective under certain assumptions and in areas where these are fulfilled. As with any other approximation application, there is also an error from the reconstruction process. While we are going to work under these same assumptions, we wanted to try a different approach using a Convolutional Neural Network (ConvNet) to exploit the power of machine learning algorithms and compare the result with previous approaches.

We also would like to apply our trained ConvNet to images of different natural areas to test its ability to reconstruct matrices describing unfamiliar terrain.

1.3 Structure of the thesis

The thesis is structured into 6 chapters including the introduction.

Chapter 2 reviews some of the most important principles of SAR imaging, including the covariance matrix data format, both for quad-pol and compact-pol, and reconstruction methods. These aspects have been studied to process our data set.

Chapter 3 presents the theory of Deep Learning, focusing in particular on convolutional neural networks.

Chapter 4 describes the way in which data are processed, from calibration to covariance matrix construction. It has been also described how we have applied the reconstruction methods of the previous literature and the tools we used to set-up ours.

Chapter 5 presents the results obtained from the different reconstruction methods and settings. This chapter also includes a comparison of results, applying to the same data Souyris and Nord's methods and ours.

Chapter 6 summarizes the work and proposes some future work using ConvNets in SAR images.

2 Remote sensing images

Remote sensing is defined, for our purposes, as the measurement of object properties on the earth's surface using data acquired from satellites. It is, more in general, an attempt to measure something *at a distance*, rather than *in situ*. Since there is not a direct contact with the object of interest, it is necessary to rely on propagated signals of some sort, for example optical, acoustical or microwave [10]. One of this methods contemplates the use of synthetic aperture radars.

2.1 Synthetic aperture radars (SAR)

SAR systems are active sensors that were invented to allow high resolution monitoring of the Earth's surface. They are mounted on either space-borne or air-borne platforms from where they transmit a coherent electromagnetic pulse in the microwave region of the electromagnetic spectrum and measure the back-scattered response from the surface [8].
These pulses are partially reflected back to the radar by targets within the antenna beam (Figure 2.1). Since SAR sensors operate in the microwave frequency bands, the transmitted signal can penetrate clouds and most weather conditions.

SAR sensors can be divided into two main categories:

- Mono-static radars. They correspond to a system where the transmitter and the receiver share the same antenna.
- Bi-static radars. They correspond to a system where the transmitter and the receiver are separated by a considerable distance [11].

As the name implies, SAR instruments synthesize an aperture length in order to obtain a high resolution. This is achieved by utilizing the movement of the radar and further performing specialized signal processing to obtain a high resolution. As the platform travels and measure the response from a given target, the Doppler history will be considered for all the back-scattered signals from this target [11].
The spatial resolution is given by the minimum distance between two points on the surface that are still separable [12].

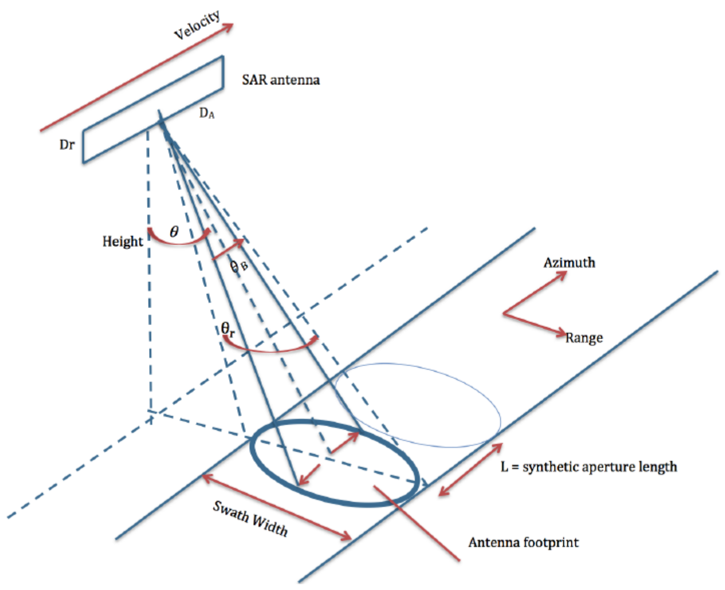


Figure 2.1: Illustration of the SAR geometry based on Figure 1.1 in [Lee and Pottier, 2009, p. 6].

2.1.1 Polarization

The polarization in SAR data is given by the orientation of the electric field of the wave which is reflected by the target we are observing, and it is expressed by the elements of the vector amplitude of the electric field [34]. Multiple polarization are useful to get more information about the shape of the target.

SAR instruments can be divided into three groups depending on the type of polarization:

- **Quad-polarization.** Also known as fully-polarized case, the radar transmits using two different polarized waves and measure the reflected wave in two different polarizations. The most common case is to transmit both horizontally and vertically polarized waves, and measure both horizontally and vertical polarizations. The result is a data with 4 channels (complex numbers) per pixel, horizontal-horizontal (HH), vertical-horizontal (VH), HV, and VV.
- **Dual-polarization.** This method employs the use of one polarization at the transmitter and two at the receiver.
- **Single-polarization.** Just one channel is used for the transmission and one at the receiver.

When the transmitter and the receiver have the same polarization, co-pol components are generated, i.e.: VV and HH. The cross-pol component is generated when the transmitter and the receiver have different polarization modes, i.e.: HV and VH.

Using one polarization at the transmitter allows to cover a larger swath width than using two polarization. This advantage is exploited when it is necessary to look at a large area on the ground without caring about small details [13].
In the other hand, choosing two polarization at the transmitter allow to obtain more information about the area we are monitoring, but a smaller surface can be covered [13].

The compact polarimetric mode was introduced to combine the positive aspects of both previous ones, it is the compact polarimetry. The compact-pol uses other polarizations than the common horizontal and vertical ones, and different combinations of the polarization of the transmitter and the receiver have been suggested [14]. These are chosen such that an approximation of full-pol data can be reconstructed under certain assumptions.

2.2 The scattering coefficients

The scattering coefficients describe the transformation of an EM field as the result of the wave interaction with one or multiple scatters at the target. The interaction processes depend on the polarization, phase, power, and frequency of the wave. The target's properties may alter the properties of the incoming field, such that the scattered field contains an unique signature reflecting the properties of the target. These unique signatures are of special interest in remote sensing [15].

The incident field at the surface is defines as:

$$\overrightarrow{E}^{i}_{surface} = \frac{e^{-jk_i r}}{r} \overrightarrow{E}^{i}_{sensor} \tag{2.1}$$

where $\overrightarrow{E}^{i}_{sensor}$ is the transmitted field from the sensor, $\overrightarrow{E}^{i}_{surface}$ is the incident field at the surface, k_i is the wave vector, and $\frac{e^{-jk_i r}}{r}$ takes into account the influence of the propagation medium on amplitude and phase, and r is the distance from the sensor to the surface [15].
Considering the scattered field at the receiver, the formula which describe it is:

$$\overrightarrow{E}^{s}_{sensor} = \frac{e^{-jk_i r}}{r} \overrightarrow{E}^{s}_{surface} \tag{2.2}$$

The transformation between the incident field, $\overrightarrow{E}^{s}_{surface}$, and the scattered field, at the surface is through the so called scattering wave vector (S), and is expressed as:

$$\overrightarrow{E}^{s}_{surface} = S\overrightarrow{E}^{i}_{surface} \tag{2.3}$$

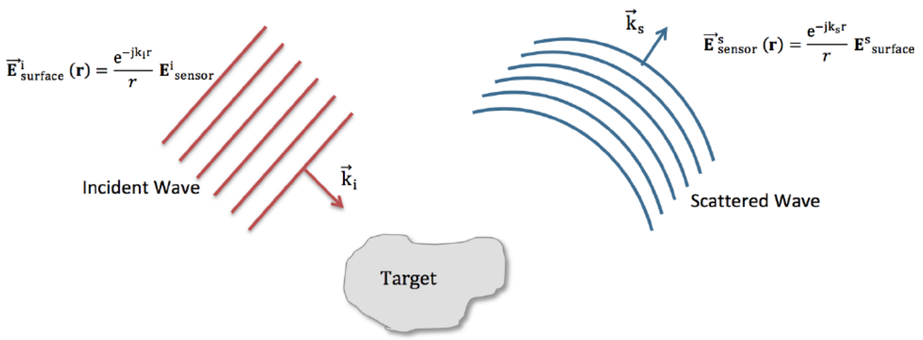

$\vec{E}^i_{surface}(\mathbf{r}) = \frac{e^{-jk_i r}}{r} \mathbf{E}^i_{sensor}$
Incident Wave
$\vec{k}_i$
Target
$\vec{k}_s$
$\vec{E}^s_{sensor}(\mathbf{r}) = \frac{e^{-jk_s r}}{r} \mathbf{E}^s_{surface}$
Scattered Wave

Figure 2.2: Interaction of an EM wave and a target in Espeseth, M. M., 2015, p. 21

Inserting for the $\overrightarrow{E}^s_{surface}$, yields:

$$\overrightarrow{E}^s_{surface} = \frac{e^{-jk_s r}}{r} S \overrightarrow{E}^i_{surface} \tag{2.4}$$

The scattering wave vector is usually written as a matrix (S), that represents the relation between the polarizations of the incoming and outgoing waves. These relationships are referred to as scattering coefficients. For quad-pol systems the relation between the transmitted fields and the measured field is [16]:

$$\begin{bmatrix} \overrightarrow{E}^s_H \\ \overrightarrow{E}^s_V \end{bmatrix} = \frac{e^{jkr}}{r} \begin{bmatrix} S_{HH} & S_{HV} \\ S_{VH} & S_{VV} \end{bmatrix} \begin{bmatrix} \overrightarrow{E}^i_H \\ \overrightarrow{E}^i_V \end{bmatrix} \tag{2.5}$$

where H and V denotes horizontal and vertical polarizations.

2.3 Polarimetric covariance matrix

The covariance matrix can usually provide information about the surface we are looking at. It is defined as the Hermitian outer product of the scattering vector. It is common to average over some area of the image, the squared window of pixels taken into consideration are usually called multilook cell, since it provides us multiple looks at the scattering process, that vary stochastically with the exact position of the scatterers relative to the transmitter and receiver. The random effects of the viewing geometry motivates the use of the covariance matrix, which is a second-order statistic. It does not make sense to look at first-order statistics, such as the mean scattering vector, since scattering coefficients are zero mean when measured over natural terrain (incoherent targets). The sample covariance matrix is created then by averaging over L pixels, i.e.:

$$C = \langle \overrightarrow{s}_i \overrightarrow{s}_i^{*T} \rangle = \frac{1}{L} \sum_{i=1}^{L} \boldsymbol{s}_i \boldsymbol{s}_i^{*T} \tag{2.6}$$

Where:

$\langle ... \rangle$ denotes spacial averaging
T denotes the transpose operator
* denotes the complex conjugate
L is the number of pixels we are considering (multilook size)
$\boldsymbol{s}_i$ is the vector defined as:

$$\boldsymbol{s}_i = [S_{HH}, S_{HV,} S_{VH}, S_{VV}]^T \tag{2.7}$$

For other applications coherency matrix is often used. It can be obtained by the covariance matrix using a linear transformation. The coherency matrix is a sample covariance matrix computed in a different basis, which is a linear transformation of the scattering vector above. I will not go deeper through it because it is not used in our experiment.

(2.7) contains four elements, but it can be reduced to three elements. There are some methods to do it, the most famous is to make a *reciprocity assumption* between S_{HV} and S_{VH} and replace them with a coherent average (that is, an average computed in the complex domain: $\frac{1}{2}(S_{HV} + S_{VH})$), which is then scaled by $\sqrt{2}$ in order to preserve power of the cross-pol measurements.[8]. I decided for this work to average those coefficients with the following formula to enable the use of integer arithmetic and to reduce the computational complexity in the convolutional neural network.:

$$S_{HV+VH} = \frac{1}{2}(S_{HV} + S_{VH}) \tag{2.8}$$

The vector used to create the covariance matrix I used for the experiment looked as follows:

$$\overrightarrow{s}_i = [S_{HH}, S_{HV+VH}, S_{VV}]^T \tag{2.9}$$

The corresponding covariance matrix is the one that follows:

$$C_L = \langle \overrightarrow{s}_L \overrightarrow{s}_L^{*T} \rangle = \begin{bmatrix} \left\langle |S_{HH}|^2 \right\rangle & \langle S_{HH} S_{HV+VH}^* \rangle & \langle S_{HH} S_{VV}^* \rangle \\ \langle S_{HV+VH} S_{HH}^* \rangle & \left\langle |S_{HV+VH}|^2 \right\rangle & \langle S_{HV+VH} S_{VV}^* \rangle \\ \langle S_{VV} S_{HH}^* \rangle & \langle S_{VV} S_{HV+VH}^* \rangle & \left\langle |S_{VV}|^2 \right\rangle \end{bmatrix} \tag{2.10}$$

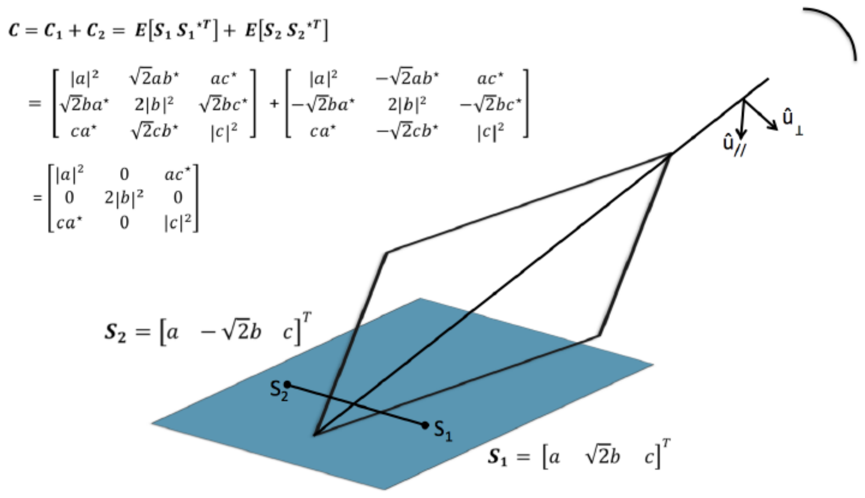

$C = C_1 + C_2 = E[S_1 S_1^{*T}] + E[S_2 S_2^{*T}]$
$= \begin{bmatrix} |a|^2 & \sqrt{2}ab^* & ac^* \\ \sqrt{2}ba^* & 2|b|^2 & \sqrt{2}bc^* \\ ca^* & \sqrt{2}cb^* & |c|^2 \end{bmatrix} + \begin{bmatrix} |a|^2 & -\sqrt{2}ab^* & ac^* \\ -\sqrt{2}ba^* & 2|b|^2 & -\sqrt{2}bc^* \\ ca^* & -\sqrt{2}cb^* & |c|^2 \end{bmatrix}$
$= \begin{bmatrix} |a|^2 & 0 & ac^* \\ 0 & 2|b|^2 & 0 \\ ca^* & 0 & |c|^2 \end{bmatrix}$
$S_2 = [a \quad -\sqrt{2}b \quad c]^T$
$S_1 = [a \quad \sqrt{2}b \quad c]^T$
S_2
S_1
$\hat{u}_{//}$
$\hat{u}_{\perp}$

Figure 2.3: Reflection symmetry around the line-of-sight direction (illustration based on Figure 3.9 in [Lee and Pottier, 2009, p. 69]).

2.3.1 Reflection symmetry

Reflection symmetry can be assumed when the distributed target has two points (S_1 and S_2) with equal contribution [17]. The scatterers (S_1 and S_2) are mirrored to each other as shown in the figure below [18]. The equations in this figure demonstrate how the two scattering targets within the resolution cell produce a decorrelation between the cross- and the co-pol scattering elements.

The following decorrelation takes place between the co- and cross-pol elements [11]:

$$\left\langle S_{HH} S^*_{HV+VH} \right\rangle = \left\langle S_{VV} S^*_{HV+VH} \right\rangle = 0 \tag{2.11}$$

This means that the orientation distribution is symmetrical about the vertical direction. Applying therefore the symmetry assumption, the resulting covariance matrix takes the form:

$$C_L = \begin{bmatrix} \left\langle |S_{HH}|^2 \right\rangle & 0 & \langle S_{HH} S^*_{VV} \rangle \\ 0 & \left\langle |S_{HV+VH}|^2 \right\rangle & 0 \\ \langle S_{VV} S^*_{HH} \rangle & 0 & \left\langle |S_{VV}|^2 \right\rangle \end{bmatrix} \tag{2.12}$$

2.4 Compact polarimetry

In the latest years, the use of compact polarimetry has become increasingly widespread. The reason is given by the fact that it exploits the benefit of both quad-pol and dual-pol setups, that is the amount

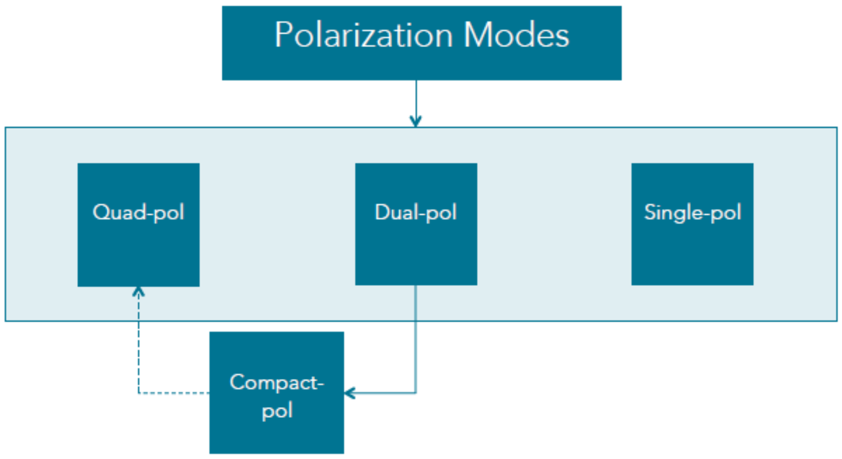


Figure 2.4: The different polarization architectures in Espeseth, M. M., 2015, figure 5.1 p. 51.

of information we can obtain from the first and the wider swath capacity of the second one.
The first satellite that operated with the compact-pol mode was the Mini-SAR sensor. This sensor was launched on October 22th 2008 on the Chandrayaan-1 mission, and operated for nine months [19].

The polarization mode defines the system with regards to the polarization of the transmitted and received electromagnetic field. Those modes can be divided mainly in three categories:

- single-pol. This system transmits waves using only one polarization, and receives in the same polarization.
- dual-pol. This system transmits waves using only one polarization, and receives in two polarizations.
- quad-pol. This system transmits waves with two orthogonal polarizations and measures the response coherently in two polarizations.

A compact-pol system is a subgroup of dual-pol system. In this systems, the transmitted signal is either a circularly polarized wave, or a linear combination of a horizontally and a vertically polarized wave ($\frac{\pi}{4}$-pol architecture) [6]. The back-scattered signals are either recorded in the horizontal and vertical polarization basis, or left- and right-hand circular polarization basis.

SAR radars can transmit just a single polarized wave at a time, so vertical and horizontal polarization must be time-multiplexed [13]. In dual- and single-pol systems only one polarization is used at the transmitter, it means that the swath width is double than in quad-pol systems (where two polarizations are used). In the other hand, compact-pol use just one polarization at the transmitter, that is how it can ensure the same coverage as single and dual pol.
Moreover, two polarizations at the transmitter require twice the average power compared to dual- and single-pol systems [13].

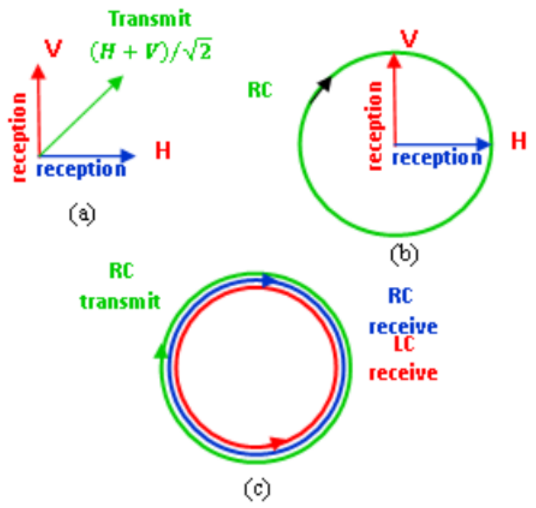


Figure 2.5: The three configurations of the compact polarimetry (a) π/4 , (b) CTLR and (c) DCP in [21], figure 2 p. 3.

2.4.1 Compact polarimetry modes

There are three different modes discussed in the literature:

- $\frac{\pi}{4}$-pol mode. The radar transmit a wave which is linearly polarized at 45 degrees to the horizontal and vertical directions [6].
- Circular mode. The radar transmits circularly polarized waves and receives linear, in the two orthogonal linear polarizations H and V. This type of configuration consists of two sub-classes; left- and right-hand circular polarizations at the transmitter. This mode is also called hybrid-pol mode [20]. In this work we focus in this kind of compact-pol data because they are widely used and perform better in the reconstruction of previous works [8].
- DCP mode. This system transmits circularly polarized pulses, and receives coherently dual-circular polarizations [20].

The scattering vector for $\frac{\pi}{4}$-pol mode:

$$\vec{k}_{\frac{\pi}{4}} = \begin{bmatrix} S_{\frac{\pi}{4}H} \\ S_{\frac{\pi}{4}V} \end{bmatrix} = \begin{bmatrix} S_{HH} & S_{VH} \\ S_{HV} & S_{VV} \end{bmatrix} \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} S_{HH} + S_{HV} \\ S_{VV} + S_{VH} \end{bmatrix} \tag{2.13}$$

The scattering vector for hybrid-pol mode:

$$\vec{k}_{LC/RC} = \begin{bmatrix} S_{LH/RH} \\ S_{LV/RV} \end{bmatrix} = \begin{bmatrix} S_{HH} & S_{VH} \\ S_{HV} & S_{VV} \end{bmatrix} \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ \pm j \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} S_{HH} \pm jS_{HV} \\ \pm jS_{VV} + S_{VH} \end{bmatrix} \tag{2.14}$$

The scattering vector for circular-pol mode:

$$\overrightarrow{k}_{LC/RC} = \begin{bmatrix} S_{LL} & S_{LR} \\ S_{RL} & S_{RR} \end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & j \\ j & 1 \end{bmatrix}\begin{bmatrix} S_{HH} & S_{HV} \\ S_{VH} & S_{VV} \end{bmatrix}\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & j \\ j & 1 \end{bmatrix} =$$

$$= \frac{1}{2}\begin{bmatrix} S_{HH} - S_{VV} + 2jS_{HV} & j(S_{HH} + S_{VV}) \\ j(S_{HH} + S_{VV}) & S_{VV} - S_{HH} + 2jS_{HV} \end{bmatrix} \tag{2.15}$$

The term $\sqrt{2}$ is present to conserve the power [6]. We remark that in this work coefficients S_{HV} and S_{VH} are averaged in a single value.

As we can observe in the equations above, all the scattering coefficients in the new representations contain a blend of co- and cross-pol terms, as defined with respect to the linear basis.

2.4.2 Compact polarimetric covariance matrix

The sample covariance matrix for the compact-pol modes is given as the averaged Hermitian outer product of the target vectors.

Covariance matrix for $\frac{\pi}{4}$-pol mode:

$$C_{\frac{\pi}{4}} = \left\langle \overrightarrow{k}_{\frac{\pi}{4}} \overrightarrow{k}^{*T}_{\frac{\pi}{4}} \right\rangle = \frac{1}{2}\begin{bmatrix} \left\langle |S_{HH}|^2 \right\rangle & \langle S_{HH}S_{VV}^* \rangle \\ \langle S_{VV}S_{HH}^* \rangle & \left\langle |S_{VV}|^2 \right\rangle \end{bmatrix} + \frac{\left\langle |S_{HV}|^2 \right\rangle}{2}\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} +$$

$$+\frac{1}{2}\begin{bmatrix} 2\Re(\langle S_{HH}S_{HV}^* \rangle) & \langle S_{HH}S_{HV}^* \rangle + \langle S_{HV}S_{VV}^* \rangle \\ \langle S_{HH}^*S_{HV} \rangle + \langle S_{VV}S_{HV}^* \rangle & 2\Re(\langle S_{VV}S_{HV}^* \rangle) \end{bmatrix} \tag{2.16}$$

Covariance matrix for hybrid-pol mode:

$$C_{Hybrid}^{LC/RC} = \left\langle \vec{k}_{LC/RC}\vec{k}^{*T}_{LC/RC} \right\rangle = \frac{1}{2}\begin{bmatrix} \left\langle |S_{HH}|^2 \right\rangle & \mp \mathrm{j}\langle S_{HH}S_{VV}^* \rangle \\ \pm j\,\langle S_{VV}S_{HH}^* \rangle & \left\langle |S_{VV}|^2 \right\rangle \end{bmatrix} + \frac{\left\langle |S_{HV}|^2 \right\rangle}{2}\begin{bmatrix} 1 & \pm j \\ j & 1 \end{bmatrix} +$$

$$+\frac{1}{2}\begin{bmatrix} \pm 2\Im\left(\langle S_{HH}S_{HV}^*\rangle\right) & \langle S_{HH}S_{HV}^*\rangle + \langle S_{HV}S_{VV}^*\rangle \\ \langle S_{HH}^*S_{HV}\rangle + \langle S_{VV}S_{HV}^*\rangle & \mp 2\Im\left(\langle S_{VV}S_{HV}^*\rangle\right) \end{bmatrix} \tag{2.17}$$

Covariance matrix for circular-pol mode:

$$C_{DCP}^{RC} = \left\langle \overrightarrow{k}_{DCP} \overrightarrow{k}_{DCP}^{*T} \right\rangle = \frac{1}{4}\begin{bmatrix} \left\langle |S_{VV} - S_{HH}|^2 \right\rangle & \langle -j(S_{VV} - S_{HH})(S_{HH} - S_{VV})^* \rangle \\ \langle j(S_{VV} + S_{HH})(S_{VV} - S_{HH})^* \rangle & \left\langle |S_{VV} + S_{HH}|^2 \right\rangle \end{bmatrix} +$$

$$+\frac{1}{4}\begin{bmatrix} 4\left\langle |S_{HV}|^2 \right\rangle & 0 \\ 0 & 0 \end{bmatrix} + \frac{1}{4}\begin{bmatrix} 4\Im\left(\langle (S_{VV} - S_{HH})\, S_{HV}^* \rangle\right) & 2\left\langle S_{HV}\left(S_{HH} + S_{VV}\right)^* \right\rangle \\ 2\left\langle (S_{HH} + S_{VV}^*) \right\rangle & 0 \end{bmatrix} \tag{2.18}$$

2.4.3 Compact polarimetric information extraction

There are different ways to extract information from compact polarimetric data. It is possible to divide this methods in three main approaches:

- The first one concern on the reconstruction of a pseudo quad-pol covariance matrix from the compact-pol. This reconstruction allows to apply well-known full-polarimetric methods to the reconstructed matrix. This is the goal of this work, we want to try to find a new efficient method to reconstruct full-polarimetric covariance matrices.
- The second group of methods contains decompositions applied directly on the compact-pol data. As in the quad-pol case, decomposition methods aim to decompose the data into multiple scattering types, such as surface, double bounce, and volume scattering [22].
- The third group uses the compact-pol parameters available for interpretation and classification purposes directly [23].

2.4.4 Issues with the reconstruction of polarimetric information

Regarding the quad-pol case, the interpretation of the features is well known and there is a strong theory in the background, especially for sea ice images. This is why it seems worthwhile to reconstruct the full polarimetric information instead of analyzing directly compact-pol data, where we still do not have available the same suit of effective and confirmed methods.

Although it is difficult to reconstruct quad-pol features from the compact-pol data, some relevant information can still be obtained from the compact-pol case.
One of them is the cross-pol intensity, that is an important feature to distinguish smooth ice and open water for example. This parameter is present just in the quad-pol polarization. This is an example which explain the importance of finding a method to construct a pseudo matrix.

2.5 Reconstruction methods

In state-of-the-art reconstruction methods some assumptions about the target must be made. Reflection symmetry (shown in the previous chapter) is an example of such assumption. The performance of the reconstruction will depend on the accuracy of these assumptions, namely the ones that will have the least negative impact on the restored data.

2.5.1 Souyris' reconstruction method

This reconstruction process is done by assuming reflection symmetry, and that the compact-pol data represents natural surfaces. This results in zero correlation between the cross- and co-pol scattering coefficients, i.e.:

$$\langle S_{HH} S_{HV}^{*} \rangle = \langle S_{HV} S_{HH}^{*} \rangle = 0 \tag{2.19}$$

$$\langle S_{VV} S_{HV}^{*} \rangle = \langle S_{HV} S_{VV}^{*} \rangle = 0 \tag{2.20}$$

The sample quad-pol covariance matrix will then take the following form:

$$C_3 = \begin{bmatrix} \left\langle |S_{HH}|^2 \right\rangle & 0 & \langle |S_{HH} S_{VV}^{*}| \rangle \\ 0 & 2\left\langle |S_{HH}|^2 \right\rangle & 0 \\ \langle |S_{VV} S_{HH}^{*}| \rangle & 0 & \left\langle |S_{VV}|^2 \right\rangle \end{bmatrix} \tag{2.21}$$

The sample hybrid-pol covariance matrix has the following form (considering the reflection symmetry):

$$C_{Hybrid}^{LC/RC} = \frac{1}{2} \begin{bmatrix} \left\langle |S_{HH}|^2 \right\rangle & \mp j \langle |S_{HH} S_{VV}^{*}| \rangle \\ \pm j \langle |S_{VV} S_{HH}^{*}| \rangle & 0 \left\langle |S_{VV}|^2 \right\rangle \end{bmatrix} + \frac{\left\langle |S_{HV}|^2 \right\rangle}{2} \begin{bmatrix} 1 & \pm j \\ j & 1 \end{bmatrix} \tag{2.22}$$

Compared to (2.17) , the last matrix containing the correlation between the cross- and co-pol elements is now canceled out, which leaves us with fewer unknown elements.

So it is necessary introduce another equation for linking the co- and cross-pol terms. In [6], we can find the following non-linear equation to solve the system:

$$\frac{X}{H+V} = \frac{1-|\rho_{HHVV}|}{4} \tag{2.23}$$

$$X = \left\langle |S_{HV}|^2 \right\rangle \; H = \left\langle |S_{HH}|^2 \right\rangle \; V = \left\langle |S_{VV}|^2 \right\rangle \; P = \langle S_{HH} S_{VV}^* \rangle \; \rho_{HHVV} = \frac{P}{\sqrt{HV}}$$

The equation (2.23) is named “Souyris’ linking”. The letters X, H, V and P were put in substitution to make the equation more readable.

The relationship given by (2.23) is extrapolated from the cases where the back-scattered wave is either fully polarized or fully depolarized. This method can cause some problems in case of double-bounce (like urban areas) or surface scattering.

Iterative method This algorithm was first proposed in [6]. This method consists mainly on searching an expression for the cross-pol intensity $\left\langle |S_{HV}|^2 \right\rangle = X$. So then, having all the others parameters, we can reconstruct the following system of equations based on hybrid-pol data (left-circular polarization as we used in our experiment):

$$C_{QP} = \begin{bmatrix} H & 0 & P \\ 0 & 2X & 0 \\ P^* & 0 & V \end{bmatrix} = \begin{bmatrix} 2C_{11} - X & 0 & +2jC_{12} + X \\ 0 & 2X & 0 \\ -2jC_{12}^* + X & 0 & 2C_{22} - X \end{bmatrix} \tag{2.24}$$

All the elements C_{ij} are directly taken by the compact-pol matrix. Assuming that we are using hybrid-pol data, our compact-pol matrix looks as follows:

$$C_{hybrid} = \begin{bmatrix} C_{11} & C_{12} \\ C_{12}^* & C_{22} \end{bmatrix} = \frac{1}{2} \begin{bmatrix} H + X & jP - jX \\ -jP^* + jX & V + X \end{bmatrix} \tag{2.25}$$

The degree of coherence can be then expressed as:

$$\rho_{HHVV} = \frac{P}{\sqrt{HV}} = \frac{+2jC_{12} + X}{\sqrt{(2C_{11} - X)(2C_{22} - X)}} \tag{2.26}$$

In the initial step ($i = 0$), the cross-pol term is set to 0:

$$X = \left\langle |S_{HV}|^2 \right\rangle = 0 \tag{2.27}$$

Consequently, we have:

$$\left|\widehat{\rho}^{(0)}_{HHVV}\right| = \frac{|-jC_{12}|}{\sqrt{C_{11}C_{22}}} \tag{2.28}$$

Then, the following steps will be calculated as follows:

$$\widehat{\rho}^{(i)}_{HHVV} = \frac{\mp 2C_{12} + \widehat{X}^{(i-1)}}{\sqrt{(2C_{11} - \widehat{X}^{(i-1)})(2C_{22} - \widehat{X}^{(i-1)})}} \tag{2.29}$$

$$\widehat{X}^{(i)} = \frac{(C_{11} + C_{22})(1 - \left|\widehat{\rho}^{(i)}_{HHVV}\right|)}{3 - \left|\widehat{\rho}^{(i)}_{HHVV}\right|} \tag{2.30}$$

The expression introduced in [6] is just an approximation, so it is possible that we encounter some errors. Some of these errors can be caused by the fact that $\left|\widehat{\rho}^{(i)}_{HHVV}\right|$ can be larger than one for certain pixels, or even the term in the squared root can be negative. In these cases, we have to interfere and set the parameters in the following way: $\left|\widehat{\rho}^{(i)}_{HHVV}\right| = 1$ and $\left|\widehat{X}^{(i)}\right| = 0$. The user has to decide how many iterations are to be executed.

One of the main limitations to this iterative process is given by the set up of the initial step to $X = 0$. It can cause significant errors in the reconstruction process going through the iterations. In [8] there is the suggestion to transform the determination of X into a optimization problem rather than using an iterative method.

We mentioned the other relevant limitation, it is given by the assumption of reflection symmetry from which we assume that some values are close to 0.
This assumption should theoretically be satisfied for natural terrain with a sufficient amount of scatterers, and in practice we see that values are actually close to zero. The exception is when the terrain has a slope, which is often information that we want to suppress anyway.

2.5.2 Nord's reconstruction method

Compared to Souyris' method, which is well suited for surface that exhibit volume scattering, Nord alters this method by replacing the 4 in the equation (2.23) with N, which is a function which will be updated step by step [37].

The pseudo quad-pol matrix to fill is the same as in Souyris' method (2.24), so then the goal of the iterative method: to find an approximation of the term $\left\langle |S_{HV}|^2 \right\rangle$. The initial step is the same as in Souyris' method, choosing $N = 4$. This may cause some deviation in the results, as the initial step assumes azimuthal symmetry for the data, so it could be worth to change this value and try different ones [24]. Changing the value of N should give a high reconstruction performance for open water areas.

The equations to use in the Nord's method are the following:

$$\widehat{X}^{(i)} = \frac{(2C_{11} + 2C_{22})(1 - \left|\widehat{\rho}^{(i)}_{HHVV}\right|)}{(N + 2(1 - \left|\widehat{\rho}^{(i)}_{HHVV}\right|))} \tag{2.31}$$

$$\left|\widehat{\rho}^{(i)}_{HHVV}\right| = \frac{\mp 2C_{12} + \widehat{X}^{(i-1)}}{\sqrt{(2C_{11} - \widehat{X}^{(i-1)})(2C_{22} - \widehat{X}^{(i-1)})}} \tag{2.32}$$

For the first step: $\widehat{\rho}^{(0)}_{HHVV} = \frac{|-2jC_{12}|}{\sqrt{C_{11}C_{22}}}$.

When the iteration is completed, the N parameter is calculated from the elements in the pseudo quad-pol covariance matrix, and X is updated.

The N parameter is dependent on the surface of interest. In Souyris' method the N parameter was set to 4, indicating that the surface is natural and exhibits strong azimuthal symmetry. Hence, one should therefore expect the N parameter to be low for natural surfaces, and high for surfaces that exhibit double bounce scattering, i.e., urban areas.
It is updated as follows: $N = \frac{\left\langle |S_{HH} - S_{VV}|^2 \right\rangle}{\left\langle |S_{HV}|^2 \right\rangle}$ after every single iteration.

3 Deep learning

Deep learning is a particular group of machine learning techniques which allows the algorithm to build complex concepts out of simpler concepts, for example it can represent the concept of an image of a person by combining simpler concepts, such as corners and contours, which are in turn defined in terms of edges [3]. One of the most common examples is a multi-layer perceptron or a feed-forward deep network, generally speaking.

The depth (meaning the number of hidden layers and their dimension) of the network allows the computer to learn a multi-step computer program. Each layer of the representation can be thought of as the state of the computer's memory after executing another set of instructions in parallel [3]. Not all the layers decode information about the input, the representation stores state information that helps to execute a program that can make sense of the input.

In short words, deep learning is the study of models that either involve a greater amount of composition of learned functions or learned concepts than traditional machine learning does.

In this work, the method of deep learning we chose is the convolutional neural network (ConvNet). As we will explain later on, this method represent a good solution to our problem.

In this chapter we are going to focus on ConvNets and more in general in neural networks.

3.1 Artificial neural networks

The idea behind Artificial Neural Networks is to imitate the human brain, which computes in an entirely different way from the conventional digital computer.
The brain is a highly complex, non-linear and a parallel computer. It has the capability to organize its structural constituents, the neurons, so as to perform certain computations with an astonishing speed [25].

The computing power of a neural network derives through its massively parallel distributed structure and its ability to learn and therefore generalize.
A machine is able to *generalize* when it produces reasonable outputs for inputs not encountered during the training (learning). This capability allows neural networks to solve complex problems that are currently intractable [25].

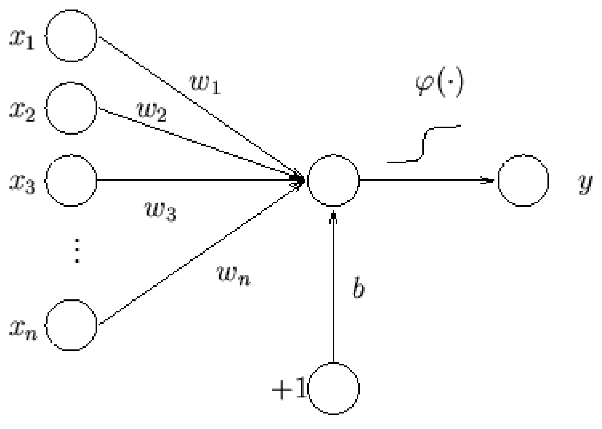


Figure 3.1: Model of one neuron (image taken from [38]).

3.1.1 Modeling one neuron

The basic computational unit of the brain is a **neuron** (Figure 3.1). Neurons are connected to each other by **synapses**. Each neuron receives input signals from its **dendrites** and produces output signals along its **axon**. In artificial neural networks, the signal travel along the axons ($\mathbf{x_i}$) interact multiplicatively ($\mathbf{w_i x_i}$) with the dendrites of the other neuron based on the synaptic strength at that synapse ($\mathbf{x_i}$). The idea is that the synaptic strengths (the weights $\mathbf{w}$) are learnable and control the strength of influence of one neuron an another. After summing all the weighted inputs, before sending to the output, the neuron applies an activation function $\varphi()$, which is usually non-linear [26].

The most common activation function is the sigmoid (Figure 3.2a):

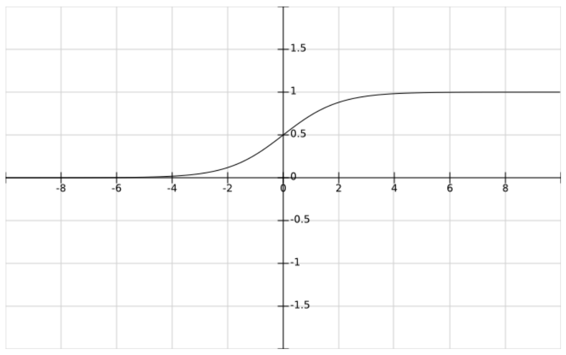
$$\varphi(x) = \frac{1}{1 + \mathrm{e}^{x}} \tag{3.1}$$

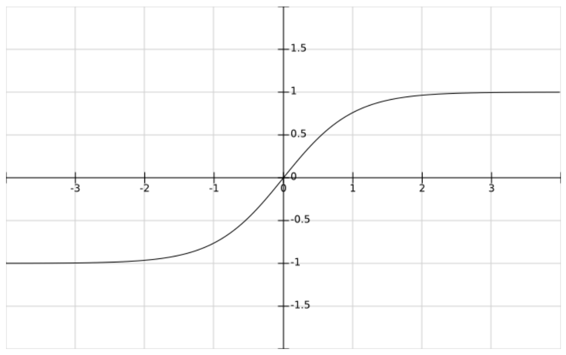
Where $x = \sum_{i=0}^{n} w_i x_i$. The bias $w_0 = b$ and $x_0 = 1$ to simplify the formula.

This function allows us to map all real values in a range between 0 and 1, with a strong incline close to 0, that make it look like a step function.

Another common activation function is *tanh(x)* (Figure 3.2b). It squashes a real-valued number to range [-1,1]. As the sigmoid neuron, its activation saturate. Its output is zero-centered, that is one of the reasons why it is often preferred to the sigmoid. We can represent tanh also as a function of sigmoid as follows:

$$tanh(\mathrm{x}) = 2\sigma(2\mathrm{x}) - 1 \tag{3.2}$$


(a) Sigmoid function.


(b) Tanh function.

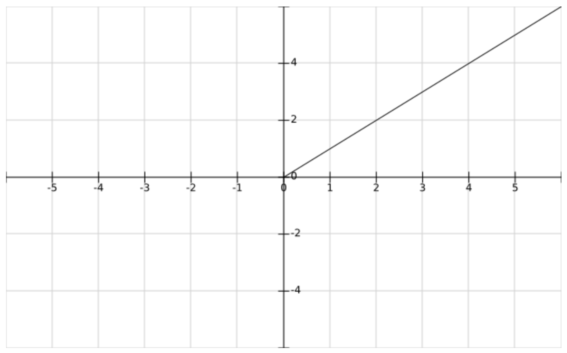

(c) ReLU function.

Figure 3.2: Activation functions.

Another attractive aspect of *tanh(x)* is its simple derivative computation:

$$\frac{d}{dx}tanh(x) = 1 - tanh^2(x) \tag{3.3}$$

The last activation function that is useful to mention for our work is the **ReLU** (Rectified Linear Unit) (Figure 3.2a). It is becoming very popular in the last years [26]. This activation function is thresholded at zero as follows:

$$f(x) = max(0, x) \tag{3.4}$$

One of the advantages of ReLU function is that it accelerate the convergence of stochastic gradient descent compared to the sigmoid/tanh functions. It is also faster, because it does not involve expensive operations as exponentials.
The issue of using ReLU could be, on the other hand, that is fragile during the training and can “die”. For example, a large gradient flowing through a ReLU neuron could cause the weights to update in such a way that the neuron will never activate on any datapoint again. If this happens, then the gradient flowing through the unit will forever be zero from that point on. That is, the ReLU units can irreversibly die during training since they can get knocked off the data manifold (the n-dimensional space where our network operates) [26].
We used some reLU layers in our experiments. As can be seen in chapter 5, to avoid that a large gradient affects these layers, we put a batch normalization (BatchNorm) layer after each reLU layer to normalize their output. In traditional deep networks, a too high learning rate may cause the gradients explode or vanish, as well as getting stuck in poor local minima. Batch Normalization helps address these issues. By normalizing activations throughout the network, it prevents small changes in layer parameters from amplifying as the data propagates through a deep network [43].

3.1.2 Layer structure

In a layered neural network, the neurons are organized in the form of layers. In the simplest form of a layered network, we have an **input layer** of sources nodes (our features) that projects onto an **output layer** of neurons (computational nodes), but not vice versa [25]. Cycles are not allowed since that would imply an infinite loop in the propagation forward (from input to output) of a network [26]. The most common neural network configuration is the fully-connected one, in which neurons between two adjacent layers are fully pairwise connected, but neurons within a single layer share no connections [26].

The output layer, unlike all the other layers in the network, most commonly do not have an activation function. It represent the class scores, if we are talking about classification or an arbitrary real-valued numbers, speaking about regression problems [25].

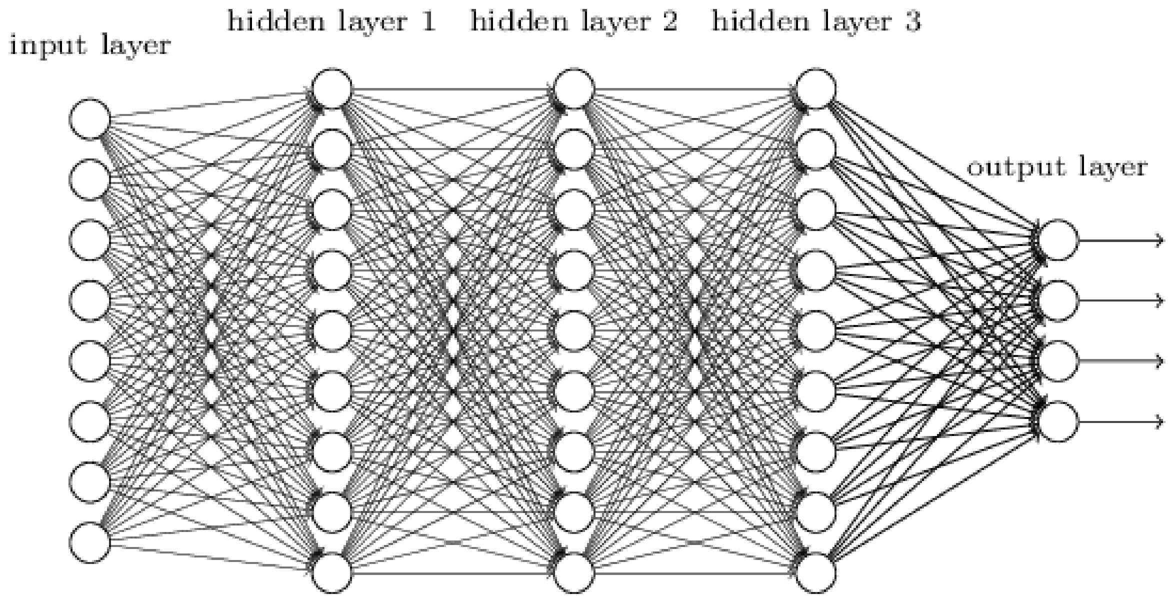


Figure 3.3: Neural Network [39].

We can built more complex neural networks adding layers between the input and the output, these are called hidden layers (Figure 3.3). By adding more layers the network is enabled to extract higher-order statistics [25].

The two metrics that people commonly use to measure the size of neural networks are the number of neurons or the number of parameters (all the connections between layers) [26].

3.1.3 Setting the dimension of the network: number of layers

How many neurons and layers are necessary for the task we want to compute? There is no appropriate answer to this question. The number of layers and the number of neurons are a hyper-parameter, it means that there is no method to understand how many of them give you the optimal configuration.

As we increase the size and number of layers in a Neural Network, the **capacity** of the network increases, this means that the space of the functions we can represent with our network increase.
Using more complex models bring also a disadvantage, the Network could learn too much from the training data set and fit also some noise that affect it. This is called **overfitting** [26].

To avoid overfitting there are some rules we can follow.
One is taking a look to the space of the data: bigger it is the space and more complex can be the model.
The amount of data we have is also an important parameter. A large data set is more difficult to overfit. This allows the designer to chose a bigger model.

All these considerations were studied by Vapnik and Chervonenkis on a theoretical point of view [27], but we will not go deeper.

Practically speaking we use different techniques to avoid overfitting. It means that we chose the most complex network our system can afford and then we apply: regularization, dropout and/or input noise [26, 48].

3.1.4 Loss function

The objective of Machine Learning is to make our model *learn* a pattern. This means that we want to *train* our model. Neural Networks are a supervised method, so our training consists of making a prediction and then compare it with the actual value, setting up an error function:

$$e_k(n) = d_k(n) - y_k(n) \tag{3.5}$$

Where $y_k(n)$ is the output of the network at the n iteration and k neuron of the output layer, and $d_k(n)$ is the desired output [25].

One of the simplest and most used loss functions is the **MSE** (Mean Squared Error):

$$e_k(n) = MSE = \frac{1}{k}\sum_{i=1}^{k}(d_i(n) - y_i(n))^2 \tag{3.6}$$

In this project we used the Euclidean distance that is very similar to MSE:

$$E(n) = \|\boldsymbol{d}(n) - \boldsymbol{y}(n)\|_2^2 = \sqrt{\sum_{i=1}^{q}(d_i(n) - y_i(n))^2} \tag{3.7}$$

Where q is the number of neuron of the layer.
The objective of the training is to make $e_k(n)$ smaller and smaller (avoiding meanwhile overfitting), this process is called **optimization**.

Since we want to *minimize* the quadratic error $e_k(n)$, we compute the gradient of it with respect to the weights.

The most common method is the stochastic gradient descent, through which we chose a direction to update the weights [28].

3.1.5 Backpropagation algorithm

The backpropagation algorithm is used to find a local minimum of the error function. The network is initialized with randomly chosen weights.

The learning problem consists of finding the optimal combination of weights so that the network function $\widehat{f}$, which represent the output of the network, approximates a given function f, the output we want, as closely as possible. It is easy to deduce that the function f is unknown. It means that we are not given the function f explicitly but only implicitly through some examples. The examples are given by a training set $\left\{(\boldsymbol{x}_1, \boldsymbol{y}_1), ..., (\boldsymbol{x}_p, \boldsymbol{y}_p)\right\}$ consisting of p ordered pairs of n- and m-dimensional vectors, which are called the input and output patterns.

$$\boldsymbol{y}_i = f(\boldsymbol{x}_i) \tag{3.8}$$

When the input pattern $\boldsymbol{x}_i$ from the training set is presented to this network, it produces an output vector $\boldsymbol{d}_i$ different in general from the target $\mathbf{y}_i$. What we want is to make $\mathbf{d}_i$ and $\mathbf{y}_i$ identical (or the closest as possible) for $i = 1, ..., p$, by using a learning algorithm [28].
We want to minimize the error function of the network, defined as:

$$E = \sqrt{\sum_{i=1}^{q} (\boldsymbol{d}_i(n) - \boldsymbol{y}_i(n))^2} \tag{3.9}$$

The gradient of the error function is computed and used to correct the initial weights. Our task is to compute this gradient recursively [28].

Every single j output units of the network is connected to a node which evaluates the function $\frac{1}{2}(d_{ij} - y_{ij})^2$, where d_{ij} and y_{ij} denote the *j-th* component of the output vector $\boldsymbol{d}_i$ and of the target $\boldsymbol{y}_i$. The m outputs are added in a single node and they give the sum E_i as its output (Figure 3.4).

The weights in the network are the only parameters that can be modified to make the quadratic error E as low as possible. We can minimize E by using an iterative process of gradient descent, for which we need to calculate the gradient:

$$\nabla E = (\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, ..., \frac{\partial E}{\partial w_l}) \tag{3.10}$$

Each weight is updated using the increment:

$\triangle w_i = -\gamma \frac{\partial E}{\partial w_i}$ for $i = 1, ..., l$ γ =learning constant.

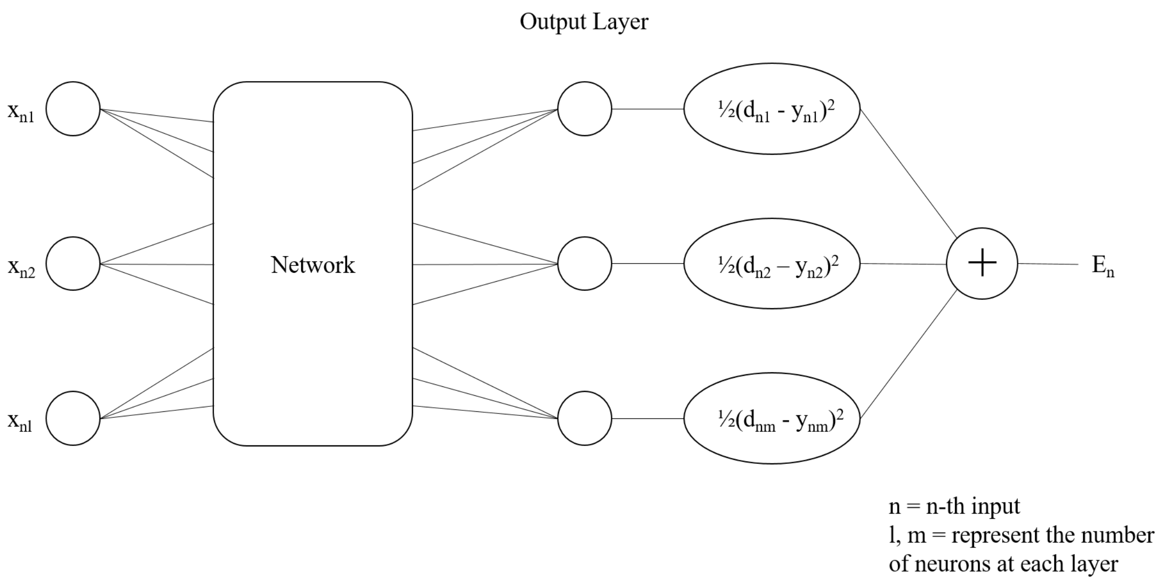

Output Layer
x_{n1}
x_{n2}
x_{nl}
Network
$\frac{1}{2}(d_{n1} - y_{n1})^2$
$\frac{1}{2}(d_{n2} - y_{n2})^2$
$\frac{1}{2}(d_{nm} - y_{nm})^2$
+
E_n
n = n-th input
l, m = represent the number of neurons at each layer

Figure 3.4: Extended network for the computation of the error function. In [28] fig 7.6 p. 156.

Using this method, we can adjust the network weights iteratively, using the backpropagation. In this way we expect to find a minimum of the error function.

The computational complexity of learning algorithm becomes the critical limiting factor when one envisions very large data sets. Stochastic gradient algorithms are recommended for large scale machine learning problems.
The stochastic gradient descent (SGD) algorithm is a drastic simplification of the standard gradient descent. Instead of computing the gradient of E exactly, each iteration estimates this gradient on the basis of a single randomly picked example [40].

In the backpropagation step the input from the right of the network is the constant 1. Incoming information to a node is multiplied by the weight stored in its left side. The result of the multiplication is transmitted to the next unit to the left.
The backpropagation step provides an implementation of the chain rule. Any sequence of function compositions can be evaluated in this way and its derivative can be obtained in the backpropagation step. We can think of the network as being used backwards, whereby at each node the product with the value stored in the left side is computed [28].

3.2 Convolutional neural networks

Convolutional Neural Networks (ConvNets) are a specialized kind of neural network for processing data that has a known, grid-like topology [29]. This made ConvNets an attractive choice for our problem, since our data set is composed by images.

There are four key ideas behind ConvNets that take advantage of the properties of natural signals: local connections, shared weights, pooling and the use of many layers [7].

ConvNets have been tremendously successful in practical applications [9]. The name "convolutional neural network" indicates that the network employs **convolution**.
Convolution is a specialized kind of linear operation. ConvNets are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers [3].

3.2.1 The Convolution operation

In its most general form, convolution is an operation on two functions of a real-valued argument.

Consider the function $x(t)$ and the function $w(a)$. The function $s(t)$ gives the result of the **convolution** between them:

$$s(t) = \int x(a)w(t-a)da \tag{3.11}$$

It is commonly denoted as follows: $s(t) = (x * w)(t)$.

In convolutional network terminology, the first argument to the convolution is often referred to as the **input** and the second argument as the **kernel**. The output is sometimes referred to as the **feature map**.

In real problems we usually do not have continuous function but sequences. So we need a discrete representation of the convolution operation:

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a) \tag{3.12}$$

In machine learning applications, the input is usually a multidimensional array of data and the kernel is usually a multidimensional array of parameters that are adapted by the learning algorithm. We will refer to these multidimensional arrays as **tensors**.

The commutative property of convolution arises because we have **flipped** the kernel relative to the input. The only reason to flip the kernel is to obtain the commutative property. It is not commonly applied.

In the Figure 3.5 is shown a convolutional operation example.

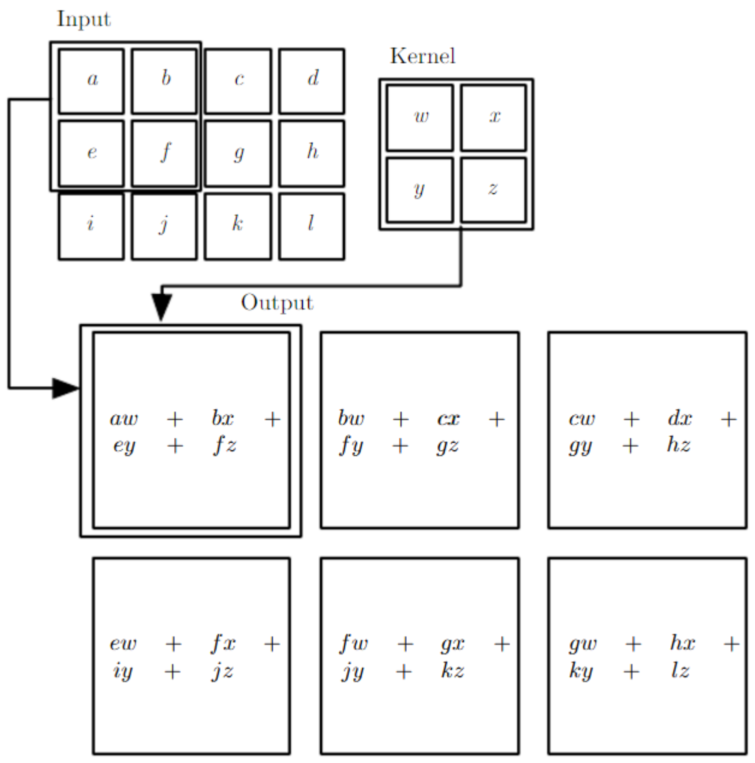


Figure 3.5: An example of 2-D convolution (Figure 9.1 p. 334 in [3]).

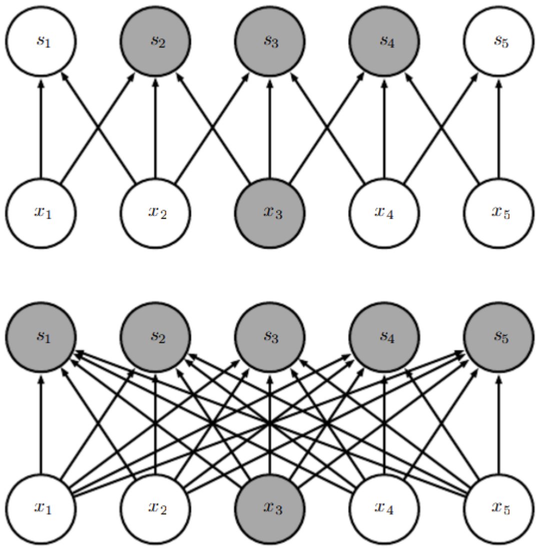


Figure 3.6: *Sparse connectivity, viewed from below.* The input $x_3$and the output units affected by it are highlighted. (Top) When s is formed by convolution with a kernel of width 3, only three outputs are affected by x.(Bottom) When s is formed by matrix multiplication, connectivity is no longer sparse, so all of the outputs are affected by x_3. (Figure 9.4 p. 337 in [3]).

It is rare for convolution to be used alone in machine learning; instead convolution is used simultaneously with other functions, and the combination of these functions does not commute regardless of whether the convolution operation flips its kernel or not [3].

The choice to use convolution in neural networks is supported by three main motivations:

- **Sparse interactions**: this is in opposition to the fully-connected neural network (where every neuron is connected to every single neuron of the following layer). This is realized making the kernel smaller than the input. For example, considering an image, the input image might have thousands or millions of pixels, but the network can detect small, meaningful features such as edges with kernels that occupy only tens or hundreds of pixels. The advantages of this aspect are computational (low memory usage and fast calculation) and statistical (avoiding meaningless elements). This allows the network to effectively describe complicated interactions between many variables by constructing such interactions from simple building blocks that each describe only sparse interactions (Figure 3.6).

- **Parameter sharing**: the network use the same parameter for more than one function in the model, as opposed to the traditional neural networks where each element is used just and always once when computing the output of a neuron. In a convolutional neural net, each member of the kernel is used at every position of the input, except if the architecture is avoiding some input elements as boundary pixels for example.

- **Equivariant representations**: the layer have a property of equivariance to translation. It means that if the input changes, the output changes in the same way. For example, in images, convolution creates a 2-D map of where certain features appear in the input. If we move the object in the input, its representation will move the same amount in the output.
  Convolution is not naturally equivariant to some other transformations, such as changes in the scale or rotation of an image. Other mechanisms are necessary for handling these kinds of transformations [3].

Moreover, convolution provides a means for working with inputs of variable size.

3.2.2 Pooling

A typical layer of a convolutional network consists of three stages:

- First stage: the layer performs several convolutions in parallel to produce a set of linear activations.
- Second stage: each linear activation is run through a nonlinear activation function, such as the rectified linear activation function (**detection**).
- Third stage: we use a **pooling** function to modify the output of the layer further.

A pooling function replaces the output of the net at a certain location with a summary statistic of the nearby outputs.
For example, the *max pooling* operation reports the maximum output within a rectangular neighborhood [30]. This is shown in figure 3.7.

In all cases, pooling helps to make the representation become approximately **invariant** to small translations of the input. Invariance to local translation can be a very useful property if we care more about whether some feature is present than exactly where it is.

The use of pooling can be viewed as adding an **infinitely strong prior** that the function the layer learns must be invariant to small translations. When this assumption is correct, it can greatly improve the statistical efficiency of the network.
Moreover, if we pool over the outputs of separately parametrized convolutions, the features can learn which transformations to become invariant to.

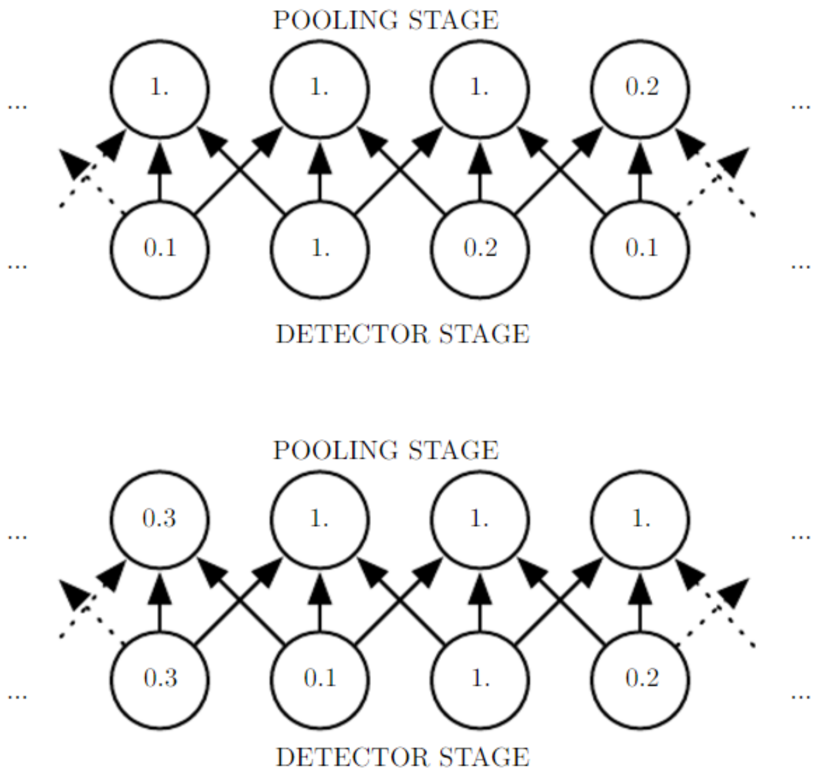


Figure 3.7: *Max pooling example.* (Top) A view of the middle of the output of a convolutional layer. The bottom row shows outputs of the nonlinearity. The top row shows the outputs of max pooling, with a stride of one pixel between pooling regions and a pooling region width of three pixels. (Bottom) A view of the same network, after the input has been shifted to the right by one pixel. Every value in the bottom row has changed, but only half of the values in the top row have changed, because the max pooling units are only sensitive to the maximum value in the neighborhood, not its exact location. (Figure 9.8 p. 343 in [3]).

Convolution and pooling as an infinitely strong prior This is a probability distribution over the parameters of a model that encodes our beliefs about what models are reasonable, before we have seen any data.

We can define a prior weak if its distribution has a high entropy, for example a Gaussian with high variance. In the other hand, a low variance of a Gaussian distribution and low entropy give us a strong prior.

An infinitely strong prior places zero probability on some parameters and says that these parameter values are completely forbidden, regardless of how much support the data gives to those values. We can consider the convolutional neural network as an infinitely strong prior because the weights for one hidden unit must be identical to the weights of its neighbor, but shifted in space and the weights must be zero,except for in the small, spatially contiguous receptive field assigned to that hidden unit.

One issue of pooling is that it can cause underfitting. The network is working correctly just if the assumptions made by the prior are accurate.
Some convolutional network architectures are designed to use pooling on some channels but not on other channels, in order to get both highly invariant features and features that will not underfit when the translation invariance prior is incorrect [31].

3.2.3 Some theoretical aspect about the implementation of convolution in neural networks

The convolution in neural network differs from the standard discrete convolution operation we find in the mathematical literature. The main characteristic of ConvNet is that neurons in each layer shares the weights, they are equal, but shifted. This allows the layer to search for a feature all along the space of the data, for example an image.

In the context of neural network the convolution is working in parallel. Convolution with just a single kernel (layer) is able to extract just one kind of feature (in many spatial locations anyway). Instead, we want each layer of our network to extract many kinds of features, at many locations.
Additionally, the input is usually not just a grid of real values. Rather, it is a grid of vector-valued observations. We do not have only spatial dimension but also channels, that in RGB images are the colors and in our case they represent the element of the covariance matrix. Moreover, we usually feed the network with not only 1 data per time, but we usually put some of them in parallel, every group of them is called **batch**, so at the end we have tensor of 4 dimensions (2 spatial, the number of channels and batch size).
Because convolutional networks usually use multi-channel convolution, the linear operations they are based on are not guaranteed to be commutative, even if kernel-flipping is used. These multi-channel operations are only commutative if each operation has the same number of output channels as input channels [3].

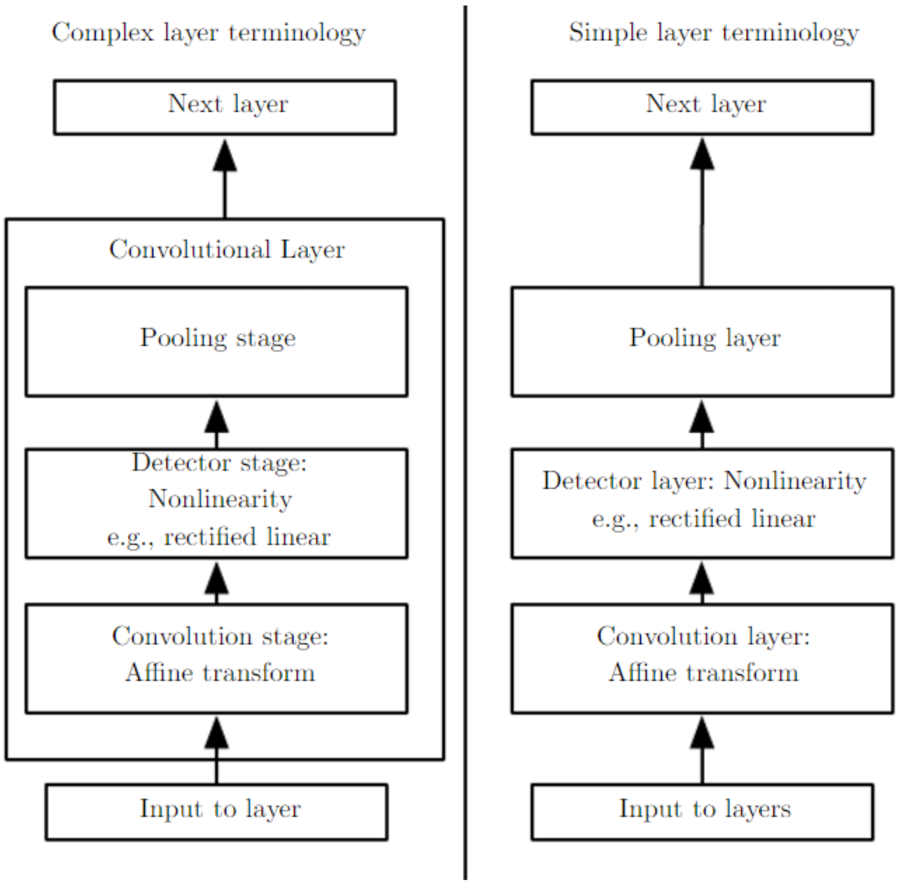

Complex layer terminology
Next layer
Convolutional Layer
Pooling stage
Detector stage: Nonlinearity e.g., rectified linear
Convolution stage: Affine transform
Input to layer
Simple layer terminology
Next layer
Pooling layer
Detector layer: Nonlinearity e.g., rectified linear
Convolution layer: Affine transform
Input to layers

Figure 3.8: The components of a typical convolutional neural network layer. On the left, the convolutional net is viewed as a small number of relatively complex layers, with each layer having many "stages". On the right, the convolutional net is viewed as a larger number of simple layers; every step of processing is regarded as a layer in its own right. (Figure 9.7 p. 341 in [3]).

One essential feature of any convolutional network implementation is the ability to implicitly zero-pad the input in order to make it wider. This means to add zeros to the data set to compute the convolution. It allows us to control the kernel width and the size of the output independently.

As we mention before, convolutional neural networks are not just composed by convolutional layers, but they may contain also different set up.
One of them is a layer of **unshared convolution**. It consists of small kernels which does not share their weights in different locations. This kind of layer is called **locally connected layer**. They are useful when we know that each feature should be a function of a small part of space, but there is no reason to think that the same feature should occur across all of space.

Tiled convolution offers a com-promise between a convolutional layer and a locally connected layer [32]. Rather than learning a separate set of weights at every spatial location, we learn a set of kernels that we rotate through as we move through space. This means that immediately neighboring locations will have different filters, like in a locally connected layer, but the memory requirements for storing the parameters will increase only by a factor of the size of this set of kernels, rather than the size of the entire output feature map.

Other operations besides convolution are usually necessary to implement a convolutional network. To perform learning, one must be able to compute the gradient with respect to the kernel, given the gradient with respect to the outputs. Multiplication by the transpose of the matrix defined by convolution is one such operation. This is the operation needed to back-propagate error derivatives through a convolutional layer, so it is needed to train convolutional networks that have more than one hidden layer.
Recall that convolution is a linear operation and can thus be described as a matrix multiplication, but we have to reshape the input tensor into a flat vector. The matrix involved is a function of the convolution kernel.
Transpose convolution is necessary to construct convolutional versions of those models.

Care must be taken to coordinate the transpose operation with the forward propagation. The size of the output that the transpose operation should return depends on the zero padding policy and stride of the forward propagation operation, as well as the size of the forward propagation's output map. In some cases, multiple sizes of input to forward propagation can result in the same size of output map, so the transpose operation must be explicitly told what the size of the original input was [33].

3.2.4 ConvNet applications

As we have already anticipated, ConvNets are thought to work on data composed by several channels, each channel being the observation of a different quantity at some point in space or time (for examples images or videos) [3, 9]. In our case the channels represent the elements of the covariance matrix.

They can also process inputs with varying spatial extents. In traditional neural networks the size of the input layer is fixed. In the other hand, ConvNets can apply its kernels a different number of times depending on the size of the input, and the output of the layer scales accordingly.
Sometimes also the output of the network is allowed to have variable size as well as the input.

Note that the use of convolution for processing variable sized inputs only makes sense for inputs that have variable size because they contain varying amounts of observation of the same kind of things [3]. This means: images with different space sizes in width and height and not different size of channels.

4 Methods

In this chapter the steps regarding the experimental process are described. The data set we used for our experiments is composed by RADARSAT-2 SAR images acquired in quad-pol configuration with fine resolution (RADARSAT-2 Data and Products (c) MacDonald, Dettwiler and Associates Ltd., 2011 & 2012 - All Rights Reserved.).

Our collection consists of different images of sea ice taken in different locations and dates as shown in Figure 4.1. Times and places were selected to provide diversity in geographic location and season in order to give the neural network a more robust ability to generalize and to avoid overfitting.

The data were processed by Python. When the data set was ready for the experiment we also added Caffe architecture, a framework used to build the neural network structure.

The flow every image file went through before the experiment is the following:

1. Extraction from .tif file and saved into numpy array.
2. Average of the two cross-pol elements into a single element, pixel by pixel, image by image.
3. Calibration of scattering matrices to sigma nought radar backscatter coefficients (area normalized with respect to a nominally horizontal plane on the ground).
4. Covariance computation with multi-look factor.
5. Subdivision of data sets into series of 224x224 pixels images .
6. Compression from quad-pol (target output of the experiment) to compact-pol (input).

The algorithms implemented for the reconstruction are the following:

- Souyris' method (reference method)
- Nord's method (reference method)
- Convolution Neural Network (our contribution)

4.1 Data structure and interpretation

The images we used for the experiments are described in the following table:

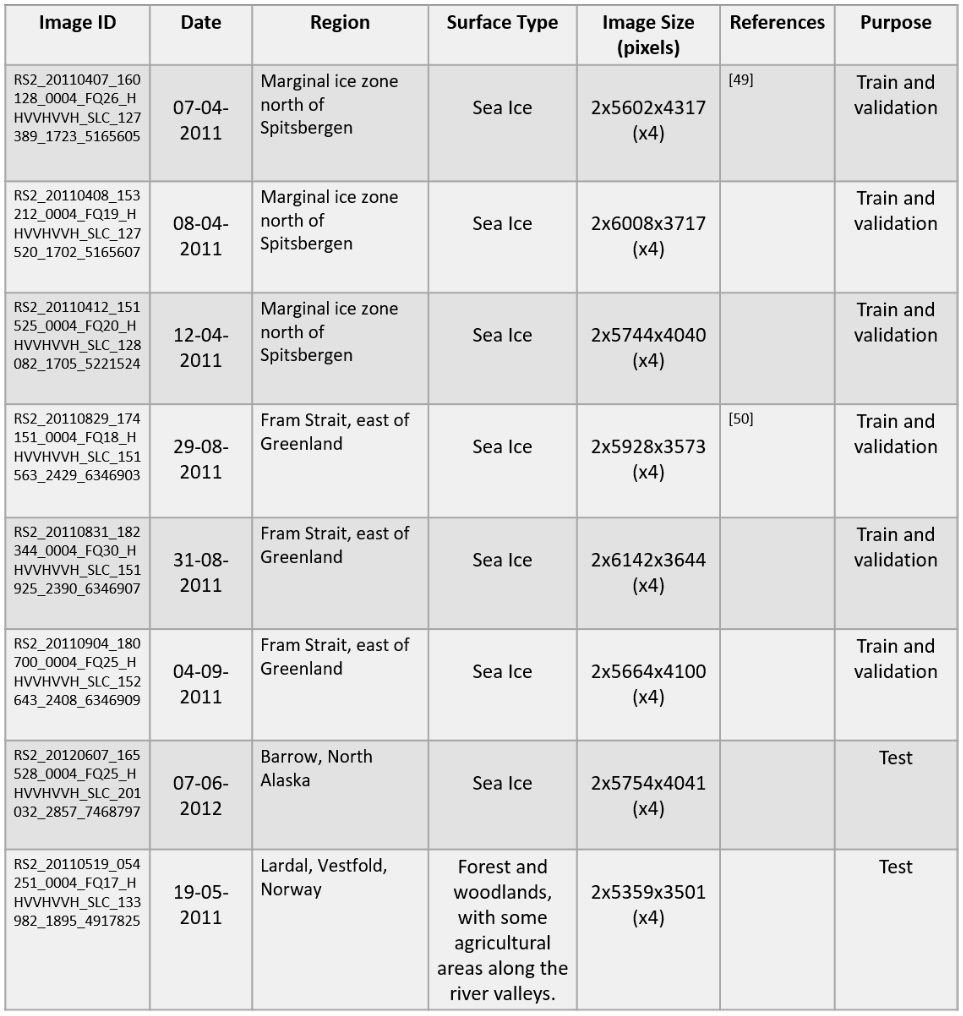


Image ID	Date	Region	Surface Type	Image Size (pixels)	References	Purpose
RS2_20110407_160128_0004_FQ26_HHVVHVVH_SLC_127389_1723_5165605	07-04-2011	Marginal ice zone north of Spitsbergen	Sea Ice	2x5602x4317 (x4)	[49]	Train and validation
RS2_20110408_153212_0004_FQ19_HHVVHVVH_SLC_127520_1702_5165607	08-04-2011	Marginal ice zone north of Spitsbergen	Sea Ice	2x6008x3717 (x4)		Train and validation
RS2_20110412_151525_0004_FQ20_HHVVHVVH_SLC_128082_1705_5221524	12-04-2011	Marginal ice zone north of Spitsbergen	Sea Ice	2x5744x4040 (x4)		Train and validation
RS2_20110829_174151_0004_FQ18_HHVVHVVH_SLC_151563_2429_6346903	29-08-2011	Fram Strait, east of Greenland	Sea Ice	2x5928x3573 (x4)	[50]	Train and validation
RS2_20110831_182344_0004_FQ30_HHVVHVVH_SLC_151925_2390_6346907	31-08-2011	Fram Strait, east of Greenland	Sea Ice	2x6142x3644 (x4)		Train and validation
RS2_20110904_180700_0004_FQ25_HHVVHVVH_SLC_152643_2408_6346909	04-09-2011	Fram Strait, east of Greenland	Sea Ice	2x5664x4100 (x4)		Train and validation
RS2_20120607_165528_0004_FQ25_HHVVHVVH_SLC_201032_2857_7468797	07-06-2012	Barrow, North Alaska	Sea Ice	2x5754x4041 (x4)		Test
RS2_20110519_054251_0004_FQ17_HHVVHVVH_SLC_133982_1895_4917825	19-05-2011	Lardal, Vestfold, Norway	Forest and woodlands, with some agricultural areas along the river valleys.	2x5359x3501 (x4)		Test

Figure 4.1: Data collection

Each single image was stored in 4 different .tif files, representing the separate layers (S_{HH}, S_{HV}, S_{VH}, S_{VV}) that compose the quad-pol scattering vector.
Each layer extracted, with "gdal" library, is read as a numpy array[L, M, N] where 16 bit signed integer values were stored.
As we mentioned before, each element of the scattering vector is a complex number.
M and N represents the number of rows and columns in the images, whereas L=2 indicates that the real and imaginary parts are stored separately.

After the extraction of the matrices, we averaged the cross-pol components (S_{HV}, S_{VH}), meaning we made the *reciprocity assumption*. This is a common assumption for sea ice images, as we mention in section 2.2. It declares that the interaction between the target and the incident wave is equal for the two polarization cross-pol channels.

The 8th image represent a forest. We wanted to test our trained networks also with this image, to check its ability to generalize on a different scenario. We cut the image in a square 1200x1200.

4.1.1 Sea ice test image

Some information about the sea ice image (Figure 4.2) we used to test our network.

In figure 4.1 it is identify by the code:

RS2_20120607_165528_0004_FQ25_HHVVHVVH_SLC_201032_2857_7468797

We will refer to this as *7th image* for simplicity.

This image represents the ocean next to Barrow, a small town in the north of Alaska. It is composed mostly of sea ice and frozen land (the corner at the bottom of example).

This image is very close to the ones we used to build the training and validation sets.

The original image was a bit bigger, we cut a section 3600x3600 pixels to make it squared and usable by the Network.

Figure 4.2 is a Pauli image [15]. The colors represent different conformations of the observed surface. In this case we have mainly dark blue areas. The blue symbolizes *surface scattering*. This condition occurs for flat and uniform surfaces, for example the sea.

4.1.2 Vegetation test image

This image, which is dominated by forest and vegetation, was used to test our network. It represents natural terrain and thereby conditions under which successful reconstruction from compact-pol data should be possible.

It is important to remark that this is a totally different image compared to the training and validation set samples. We wanted to apply also this test to verify the capability of the network to recognize the patterns also in an unknown surface type, and to examine the possibility of learning transfer..

In figure 4.1 it is identify by the code:

RS2_20110519_054251_0004_FQ17_HHVVHVVH_SLC_133982_1895_4917825

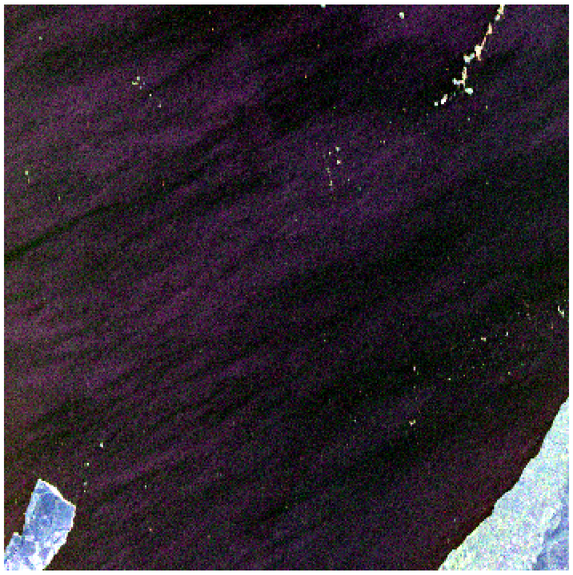


Figure 4.2: Sea ice mage used for testing. (RADARSAT-2 Data and Products (c) MacDonald, Dettwiler and Associates Ltd., 2011 - All Rights Reserved).

We will refer to this as *8th image* for simplicity.

This image represents the fields and the forest around Svarstad, Vestfold, south of Norway. It shows different patterns: there is a small village, some lakes, a river, a road, the forest and some agriculture fields by the river.

The original image was a bit bigger, we cut a section 3000x3000 pixels to make it squared and usable by the Network.

Figure 4.3 is a Pauli image [15]. The colors represent different conformations of the observed surface. In this case we have a wide range of colors. As we mention before, the blue symbolizes *surface scattering*, so mainly water. We can recognize a river and some lakes. The green symbolizes *volume scattering*, typical of forests. The red and pink colors represent *double bounce scattering*, that indicates building, artificial structures and more in general geometric structures [15].

4.2 Calibration

In the calibration process we divided the values of the image, column by column, by a factor stored in the sigma look-up table (LUT). These values are functions of the distance and the angle between the transmitter and the target [44, 45]. Calibration is required to put the pixels of the areas that were closer to the sensor and the farthest ones on the same scale. The wave have a different incidence angle when it touch the ground and come back to the sensor, depending on the distance.

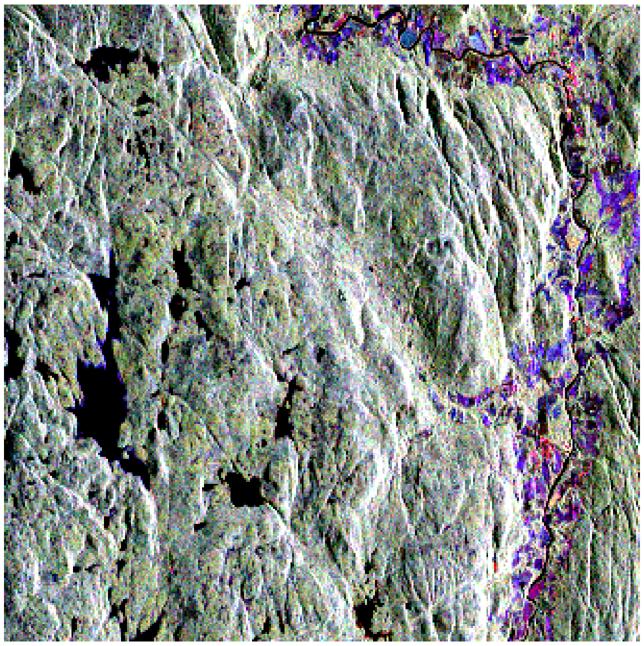


Figure 4.3: Vegetation image used for testing. (RADARSAT-2 Data and Products (c) MacDonald, Dettwiler and Associates Ltd., 2011 - All Rights Reserved).

After this manipulation, the previous 16 bit integers became 32 bit floating point, allowing us to work on decimal values with good precision.

4.3 Covariance matrix calculation

We computed the covariance matrix for each of the 6 images, using a multilook factor of 9. This means that we are averaging the covariance matrices over 9 single-look complex pixels, that represent the highest possible resolution of the Radarsat-2 SAR sensor. The resulting multilook pixels have a resolution of approximately 20m x 20m.

The input file is a numpy array [K, L, M, N] where K represent the 3-dimension scattering vector.

The formula used and described in section 2.3 is the following:

$$C = \langle \boldsymbol{s}_i \boldsymbol{s}_i^{*T} \rangle = \frac{1}{L} \sum_{i=1}^{L} \boldsymbol{s}_i \boldsymbol{s}_i^{*T} \tag{4.1}$$

For each pixel we have:

$$\boldsymbol{s}_i\boldsymbol{s}_i^{*T} = \begin{bmatrix} S_{HH} \\ S_{HV} \\ S_{VV} \end{bmatrix} [\ S_{HH} \quad S_{HV} \quad S_{VV}\ ]* = \begin{bmatrix} a_{HH}+jb_{HH} \\ a_{HV}+jb_{HV} \\ a_{VV}+jb_{VV} \end{bmatrix} [\ a_{HH}-jb_{HH} \quad a_{HV}-jb_{HV} \quad a_{VV}-jb_{VV}\ ] \tag{4.2}$$

Where a_{ii} is the real part and b_{jj} is the imaginary one. As we mention before, they are stored separately in the vector.

The *reflection symmetry assumption* is made, so the covariance matrix we want to obtain is the following:

$$C_L = \begin{bmatrix} \left\langle |S_{HH}|^2 \right\rangle & 0 & \langle S_{HH}S_{VV}^* \rangle \\ 0 & \left\langle |S_{HV+VH}|^2 \right\rangle & 0 \\ \langle S_{VV}S_{HH}^* \rangle & 0 & \left\langle |S_{VV}|^2 \right\rangle \end{bmatrix} \tag{4.3}$$

The values obtained were stored in a 5x1 vector for each 3x3 group of single-look complex (SLC) pixels. We went through the following steps to obtain that vector:

$$C_{fp} = \begin{bmatrix} c_1 = a_{HH}^2 + b_{HH}^2 & 0 & c_4 + jc_5 \\ 0 & c_2 = a_{HV}^2 + b_{HV}^2 & 0 \\ c_4 - jc_5 & 0 & c_3 = a_{VV}^2 + b_{VV}^2 \end{bmatrix} \tag{4.4}$$

Where $c_4 = a_{HH} * a_{VV} + b_{HH} * b_{VV}$ and $c_5 = a_{VV} * b_{HH} - a_{HH} * b_{VV}$

We put the non-zero values of the matrix in the following column vector, in order to : $\hat{C_{fp}} = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \end{bmatrix}$

The output file is a numpy array [J, M, N]. Where J is the dimension of the vector generated by the matrix C_{fp}.

4.4 Data set configuration

The purpose of this stage is to create a database composed of several square image blocks (224x224 pixels). This process choice of image size involves a trade-off: a small size will reduce memory allocation

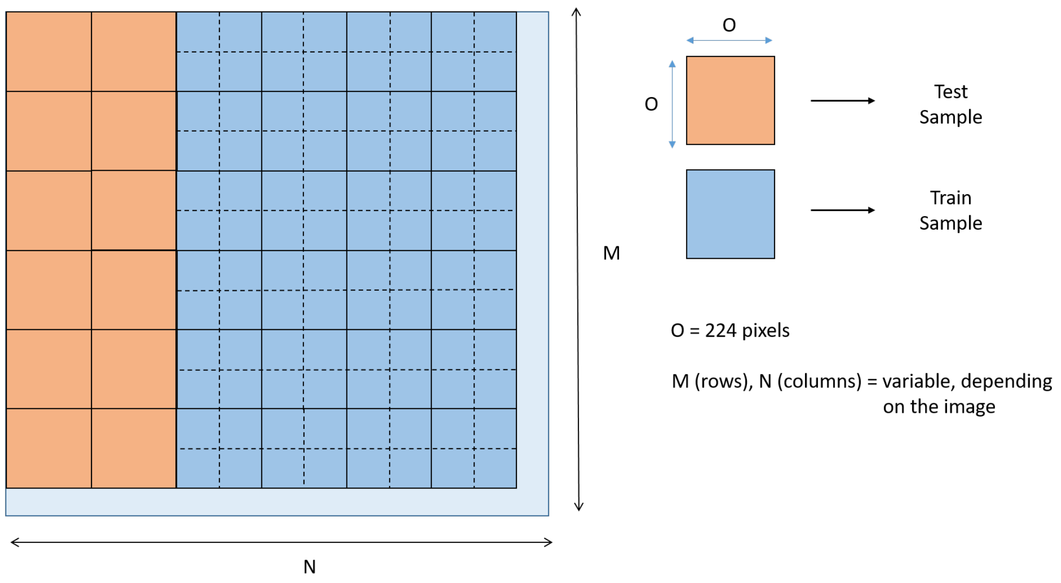


Figure 4.4: Image Cut

during the training phase, but the size should be large enough to permit the neural network to recognize some patterns in the ground structure and to capture and utilize contextual information.

The following steps were executed to obtain the suitable data set for the ConvNet. The same data set was then also applied to the previous reconstruction methods.

We split each image in two parts: one third for the test and two third for the training as shown in figure 4.4. With regards to the training, we also overlapped squares to increase the number of images we could use to feed the network.

An additional data augmentation of training data was performed by flipping the images and rotating them. This is a common approach to improve the performance of the network [42].

All of these manipulations have provided us with thousands of samples to process in the training phase.

4.5 Compact-pol data

The goal of our work is the reconstruction of quad-pol covariance matrices, meaning we had to build a simulated input, since our data were already quad-pol.

As we detailed in chapter 2, we chose to consider hybrid-pol data, since they are the most widely used. The relation between quad-pol and hybrid-pol is described by the following formula:

$$\boldsymbol{k}_{LC/RC} = \frac{1}{\sqrt{2}} \begin{bmatrix} S_{HH} \pm jS_{HV} \\ \pm jS_{VV} + S_{VH} \end{bmatrix} \tag{4.5}$$

We can then obtain a 3x2 matrix to transform quad-pol data to hybrid-pol following the steps below:

$$\boldsymbol{k}_{LC/RC} = \frac{1}{\sqrt{2}} \begin{bmatrix} S_{HH} \pm jS_{HV} \\ \pm jS_{VV} + S_{VH} \end{bmatrix} = A \begin{bmatrix} S_{HH} \\ S_{HV} \\ S_{VV} \end{bmatrix} \tag{4.6}$$

where $A = \begin{bmatrix} 1 & \pm j & 0 \\ 0 & 1 & \pm j \end{bmatrix}$is the transformation matrix. We can apply this matrix directly to the covariance matrix, according to the demonstration:

$C_{cp} = \left\langle \boldsymbol{k}_{LC/RC} \boldsymbol{k}^{*T}_{LC/RC} \right\rangle = \left\langle A\mathbf{s} \left\langle A\boldsymbol{s} \right\rangle^{*T} \right\rangle = AC_{fp}A^{*T}$

Where C_{cp} and C_{fp} are the covariance matrices, $\boldsymbol{k}_{LC/RC}$ and $\boldsymbol{s}$ are the scattering vectors.

We consider the left circular case, where $A = \begin{bmatrix} 1 & j & 0 \\ 0 & 1 & j \end{bmatrix}$

4.6 Reconstruction methods: Souyris and Nord

We implemented Souyris and Nord's methods in Python using the algorithms shown in Chapter 2.5. The result of these two experiments was a useful comparison to our convolutional neural network.

After the reconstruction method, we applied the MSE. This is the same error measure we used to asses the ConvNet, so we were able to compare them.

We observed that the Souyris and Nord's algorithms were making very small value updates ($\approx 10^{-8}$) after 180-200 iterations. We chose then to run this algorithms for 200 iteration each.

4.7 Convolutional neural network as a reconstruction method

This is the new method we wanted to develop to perform the reconstruction task from compact-pol data to quad-pol data.

As we mentioned previously, we used Caffe for the implementation of our architecture, which has also a Python interface, executed on Ubuntu OS .

Our data set was composed of 5448 images (224x224) for the training and 73 images for the test.

4.7.1 Caffe

Caffe is a deep learning framework made with expression, speed, and modularity in mind. It is developed by Berkeley AI Research (BAIR) and by community contributors. Yangqing Jia created the project during his PhD at UC Berkeley [35].
Caffe provides multimedia scientists and practitioners with a clean and modifiable framework for state-of-the-art deep learning algorithms and a collection of reference models. The framework is a BSD-licensed C++ library with Python and MATLAB bindings for training and deploying general-purpose convolutional neural networks and other deep models efficiently on commodity architectures [36]. Caffe is particularly powerful for image recognition, this is one of the reason why we chose it for our application. The reliability is confirmed by the fact that many important companies make use of it (as Facebook, Samsung, Nvidia, Intel and others).

The use of this tool is helped by the extensive documentation that creators have made available on the site [35]. The framework provides a good **modularity**, meaning that it allows to quickly change the dimension of the network by changing the number, the size and the type of layers the users want to use.
Many loss functions are usable as well.

Caffe stores and communicates data in 4-dimensional arrays called **blobs**. Blobs provide a unified memory interface, holding batches of images (or other data), parameters, or parameter updates [36].

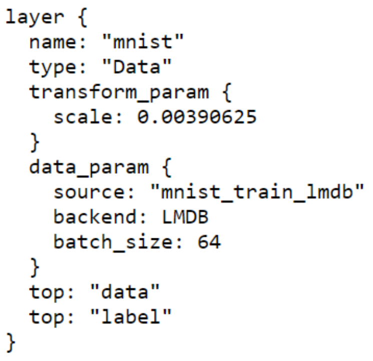
A Caffe layer is the frameworks representation of a neural network layer: it takes one or more blobs as input, and yields one or more blobs as output.

In our experiment we used the following layer types (refer to chapter 3 for the meaning of these layers):

- **ReLU**. Rectified Linear Unit.
- **conv**. Convolutional layer.
- **pool**. Pooling layer.
- **BatchNorm**. Batch normalization layer.

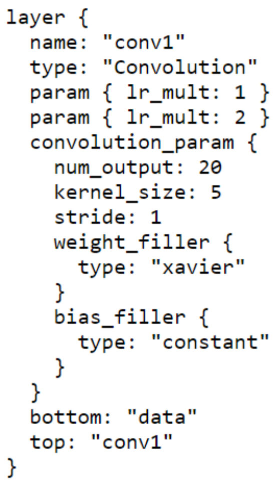
Caffe trains models by optimizing the loss function using gradient-based optimization techniques (refer to section 3.1.5) [36], in particular:

- Stochastic Gradient Descent (type: **SGD**).
- AdaDelta (type: **AdaDelta**).
- Adaptive Gradient (type: **AdaGrad**).



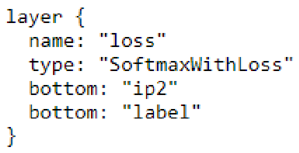
```
layer {
  name: "mnist"
  type: "Data"
  transform_param {
    scale: 0.00390625
  }
  data_param {
    source: "mnist_train_lmdb"
    backend: LMDB
    batch_size: 64
  }
  top: "data"
  top: "label"
}
```

(a) Data Layer.



```
layer {
  name: "conv1"
  type: "Convolution"
  param { lr_mult: 1 }
  param { lr_mult: 2 }
  convolution_param {
    num_output: 20
    kernel_size: 5
    stride: 1
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
  bottom: "data"
  top: "conv1"
}
```

(b) Hidden Layer.



```
layer {
  name: "loss"
  type: "SoftmaxWithLoss"
  bottom: "ip2"
  bottom: "label"
}
```

(c) Loss Layer.

Figure 4.5: Layers declaration example (screenshots from the website [35]).

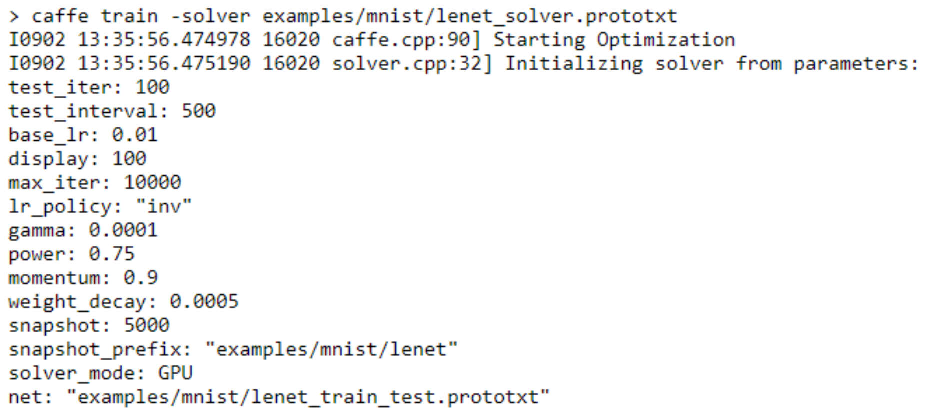
- Adam (type: **Adam**).
- Nesterov's Accelerated Gradient (type: **Nesterov**).
- RMSprop (type: **RMSProp**).

4.7.2 Implementation

The training and validation inputs are described using a prototxt file, google protocol buffer, listing the data input, their labels and the network architecture.

In this file there is the declaration of the layers we are going to use. In figure 4.5 a data layer, a hidden layer and a loss layer and a hidden are shown.

In the Data Layer (Figure 4.5a) we can set up some settings, for example specify the batch_size (number of images we want to use to feed the network at each iteration) and we can add a scale to normalize the values. In our case we did not need it, since our values were in the range [-1 , 1] after the calibration process.



```
> caffe train -solver examples/mnist/lenet_solver.prototxt
I0902 13:35:56.474978 16020 caffe.cpp:90] Starting Optimization
I0902 13:35:56.475190 16020 solver.cpp:32] Initializing solver from parameters:
test_iter: 100
test_interval: 500
base_lr: 0.01
display: 100
max_iter: 10000
lr_policy: "inv"
gamma: 0.0001
power: 0.75
momentum: 0.9
weight_decay: 0.0005
snapshot: 5000
snapshot_prefix: "examples/mnist/lenet"
solver_mode: GPU
net: "examples/mnist/lenet_train_test.prototxt"
```

Figure 4.6: Solver example (screenshot from the website [35]).

We can set up different attributes for each hidden layer (Figure 4.5b), depending on the type we chose. We then connect then to design our network architecture. The website documentation is really exhaustive.
As input and output at each layer, we have the processed data saved in blobs that are propagated forward and backward through the network.
Our network can be viewed as an encoder-decoder architecture, where the encoder produces a lower dimensional representation of the image and the decoder upsamples the low dimensional representation back to the original image size. We make use of pooling layers to subsample the feature maps between the convolutional layers in the encoder and utilize upsampling (fractional-strided convolutions) to perform upsampling in the decoder.
Finally, we specify if the layer has to be run in *TRAIN* or *TEST* phase.

The loss layer (Figure 4.5c) takes two blobs, the first one being the prediction and the second one being the label provided by the data layer. It does not produce any outputs, all it does is to compute the loss function value, report it when backpropagation starts, and initiate the gradient [35].

solver.prototxt (example in figure 4.6) is another prototxt file where many characteristics of the network can be set up.

The solver orchestrates model optimization by coordinating the network's forward inference and backward gradients to form parameter updates that attempt to improve the loss. The responsibilities of learning are divided between the Solver, for overseeing the optimization and generating parameter updates, and the Net, for yielding loss and gradients [35].

5 Results

In this chapter, the results obtained from the different reconstruction methods are described, compared and analyzed in term of loss function and image distance.

In Figure 5.1 is summarized the data process chain we described in the previous chapter.

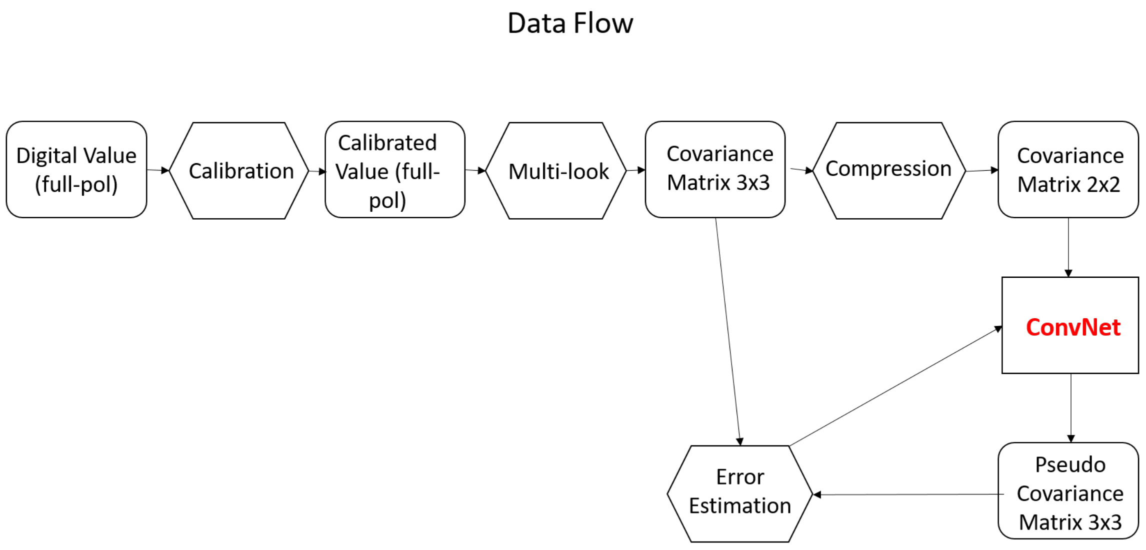


Figure 5.1: Scheme of our data processing

5.1 Error measure and comparison method

As we mention before, we applied the Euclidean distance to estimate the distance between the reconstruction and the target all over the tested method:

$$E = \|\boldsymbol{d}(n) - \boldsymbol{y}(n)\|_2^2 = \sqrt{\sum_{i=1}^{p}(d_i(n) - y_i(n))^2} \tag{5.1}$$

We also wanted to focus in particular in the reconstruction of the cross-pol correlation intensity ($\left\langle |S_{HV+VH}|^2 \right\rangle$), because it is the term which influence the reconstruction methods the most, since it has influence on every element of the reconstructed matrix (refer to section 2.5). Euclidean distance was applied also in this case.

5.2 Souyris and Nord's methods performance over our data set

Souyris' and Nord's methods were applied on our validation data set. The validation data set was composed by 73 images 224x224.

The Euclidean distance was applied between the array (73x5x224x224) of the real output values and the one of the predictions. The resulting Euclidean distances we obtained were:

- Souyris: 10,76
- Nord: 18,30

Notice that these values are high because they are not normalized. To obtain the average distance, element by element, we should divide by 73(samples)x5(channels)x224(rows)x224(columns). These results were then normalized and shown in Figure 5.11.

The same distance was applied to the 7th image (refer to Figure 4.1):

- Souyris: 1,66
- Nord: 2,16

To normalize, in this case, we should divide by 5x1200x1200.

And to the 8th image (refer to Figure 4.1):

- Souyris: 39,72
- Nord: 73,01

To normalize, in this case, we should divide by 5x1000x1000.

We can notice that Souyris' method is working better. This was expected because the images represent sea ice, and as we mention in chapter 2, it is a surface type for which Souyris' method has been reported to perform better [8].

The Euclidean distance was also applied between the array (73x224x224) of the real cross-pol values and the one of the predictions. The results we obtained was:

- Souyris: 8,81
- Nord: 9,14

Here the result about the 7th image:

- Souyris: 1,28
- Nord: 1,08

8th image:

- Souyris: 34,19
- Nord: 36,43

5.3 ConvNet method performance

Convolutional Neural Networks can be set-up in many different ways. This section is divided in subsection to show the results of two different configurations. The network was trained by a data set composed by 5448 square images of size 224 x 224 pixels..

5.3.1 1st ConvNet implementation

The representation of the structure, layer by layer, of this ConvNet is shown in figure 5.2.
Notice that every convolutional layer is followed by a rectified unit and a normalization layer [9]. We explained in chapter 3 that this is a good configuration, as the reLU can make faster the gradient descent and the normalization can keep the values in a smaller range.

Observe also that there is a dropout layer. This is used to avoid overfitting. The layer chose randomly a percentage (that has to be set-up) of weights that will not be affected at that iteration.

Each pooling layer has a kernel and a stride equal to 2. It means that is operating over a window 2 by 2 and moving the kernel two steps a time. This fact cause a compression of the original size of the image by a factor 2 for both axes.

As shown in Figure 5.2, the network has 4 pooling layers. It means that we have a sub-sampling factor of 16 through the Network. The upscore layer operates the resulting up-sampling of 16 at one time. Finally, the crop layer called *score* bring the blob back to the input dimension (224 by 224 pixels).

The dropout layer was set up at 20%.

The solver had the following settings:

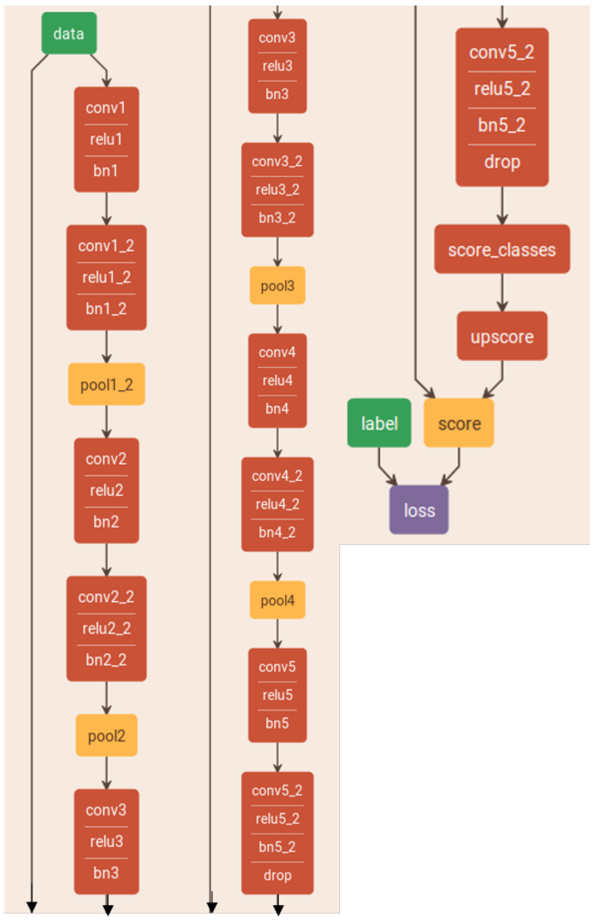


Figure 5.2: ConvNet 1st implementation. Graphical representation generated by [41].

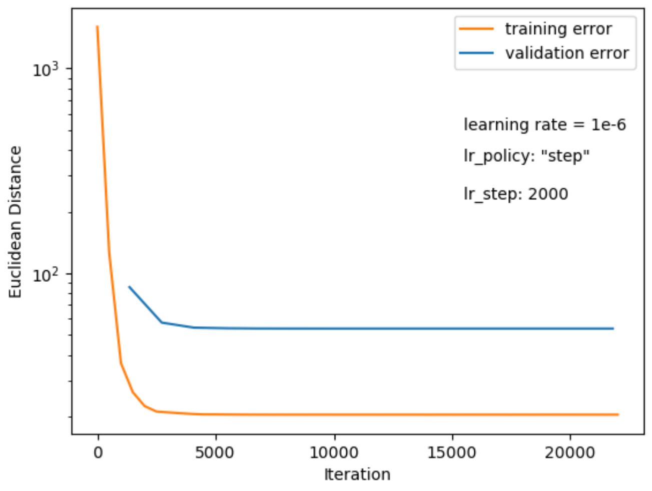


Figure 5.3: Loss function 1st network experiment.

- minimum research algorithm: Stochastic Gradient Descent
- learning rate at the beginning: 10^{-6}
- learning step: 2000
- gamma: 0.1

It means that every 2000 iteration the learning rate was multiplied by the gamma value.

As we can see in figure 5.3, after 20.000 iterations the loss function was not changing anymore.

We applied the same tests to the neural network we used for Souyris' and Nord's methods.
The Euclidean distance, applied between the array of the real output values and the one of the predictions, gave as a result:

- Validation set: 78,10
- 7th Image: 42,62
- 8th Image: 258,97

The euclidean distance, applied between the array of the real cross-pol values and the one of the predictions, gave as a result:

- Validation set: 5,89

- 7th image: 2,72
- 8th image: 34,74

As we can notice, they perform, on average, worse than Souyris and Nord's methods.

We display then the relative error, regarding the cross-pol component, in figure 5.4. It has been computed on the 7th image. We can notice how the methods from the literature perform better for low cross-pol terms ($\left\langle |S_{HV+VH}|^2 \right\rangle$). However, when this value is high (for example the corner at the right-bottom), our method is performing better.

Notice that Nord's method shows a flat error, close to 100%, almost everywhere this is due to the fact that the cross-pol values obtained by the iterative method are close to 0. This method differs from Souryis' one for the N variable at the denominator2.31. This value is equal to 4 at the beginning, but could become higher and higher, iteration by iteration, related to a consequent decrease in the predicted value of ($\left\langle |S_{HV+VH}|^2 \right\rangle$), since N is updated as follows: $N = \frac{\langle |S_{HH} - S_{VV}|^2 \rangle}{\langle |S_{HV}|^2 \rangle}$. Considering a smaller number of method iterations could improve this result.

The relative error e_{rel_i} was calculated as follows:

$$e_{rel_i} = \left| \frac{x_i - \hat{x_i}}{x_i} \right| \tag{5.2}$$

Where x_i is the target value of the matrix and $\hat{x_i}$ is the predicted value.

After noticing that our method was performing better for higher values of the cross-pol intensity (in this case they indicate sea ice), we decided to test it on an image representing a forest. The cross-pol term of the scattering vector is higher in vegetation images because of the reflections from randomly oriented structures in canopy layers (e.g. trees) [46, 47]. For sea ice, the cross-pol intensity increases with surface roughness, and is maximum for highly deformed ice, such as ridges and rubble ice.

It is important to remark that our training data set was composed mainly by sea ice images. This means that the forest image is a completely different and unknown input for the network.

The average error computed over the whole covariance matrix is significantly higher than Souyris and Nord's methods. However, if we focus on the cross-pol term, the error is close to the Souyris one. The figure 5.5 shows the absolute squared error pixel by pixel.

Notice that we had to adapt the scale for the cross-pol intensity, with respect to the scale used for sea ice. This is due to the higher values for this kind of area. Nord's method produces the same relative error almost all over the region. Souyris has some difficulties to estimate the area close to the river, this can be related to the surface type and the fullfilment of the assumptions underlying the methods.

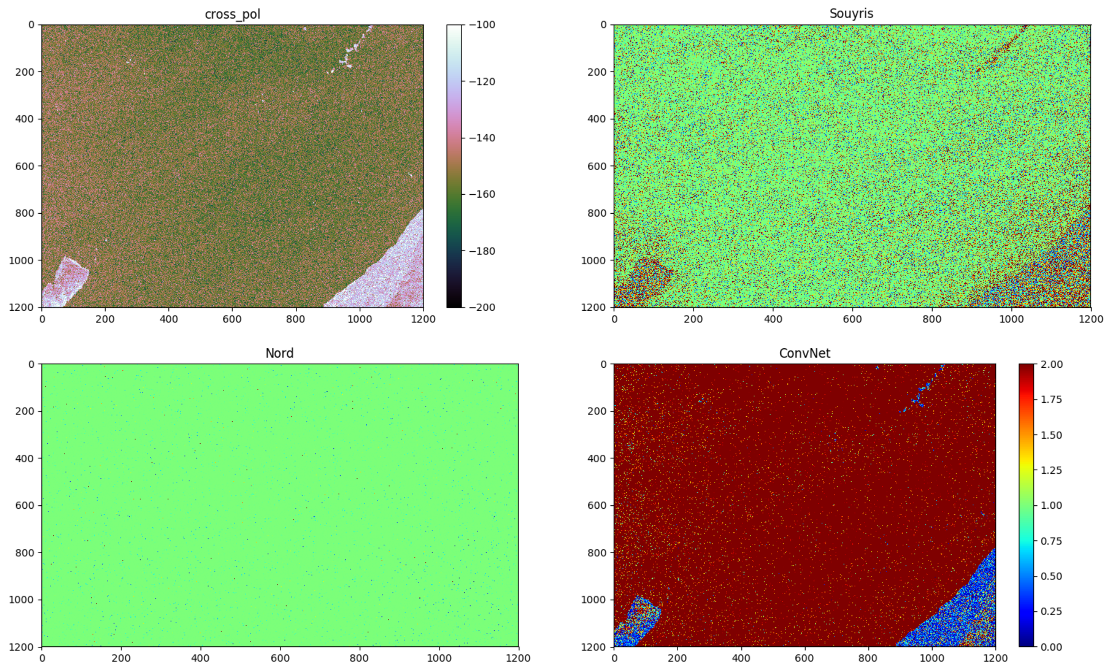


Figure 5.4: Relative error comparison, 1st network, 7th image (sea ice) of the data set. The first image represent the cross-pol value of the quad-pol covariance matrix (it is expressed in dB). The other three images represent the relative error pixel by pixel. We are comparing: Souyris, Nord and our trained network. The displayed error has a scale range between 0 and 200%.

Our method is working better than Souyris and Nord's when the value of the cross-pol intensity is in the range [-85dB, -65dB]. These cross-pol intensity values are present in the portions of the image containing sparse vegetation, next to a river and a road. Our training data set contains mainly this kind of surface, so it is familiar with the ConvNet.

5.3.2 2nd ConvNet implementation

Once we have analyzed the results of the first network, we decided to modify it to increase its power. It means that we increase its ability to elaborate regression functions. The second network we built is shown in Figure 5.6.

The changes we made regard the sub-sampling and the up-sampling. We thought that the up-sampling operated at the end of the previous network (which increased the size of the blob by 16 times) was too strong to be made at one time. This could have affected the network in term of resolution. It may cause a loss of information.

We decided then to make it gradually, operating two different up-sampling in different times. The layers upscore3 and upscore2 (Figure 5.6) carry out this task. The first one multiply the size the network by a factor 2, the second one by a factor 8.

We also decided to add a crop layer, called *score_classes2c*, between the 3rd pooling layer and the output of the first up-sampling. This could also avoid the down-sampling from causing an information loss, because we are considering a layer that did not cross the 4th pooling layer.

These modifications increase the complexity of the network.

The dropout layer was set up at 20%.

The solver had the following settings:

- minimum research algorithm: Stochastic Gradient Descent
- learning rate at the beginning: 10^{-6}
- learning step: 7000
- gamma: 0.1

Notice in Figure 5.7 that the training loss starts lower than in the first network. This is due to the initialization of the weights of the first part of the network. Since it has the same structure of the first network, we loaded there the weights we had calculated before.

The Euclidean distance, applied between the array of the real output values and the one of the predictions, gave as a result:

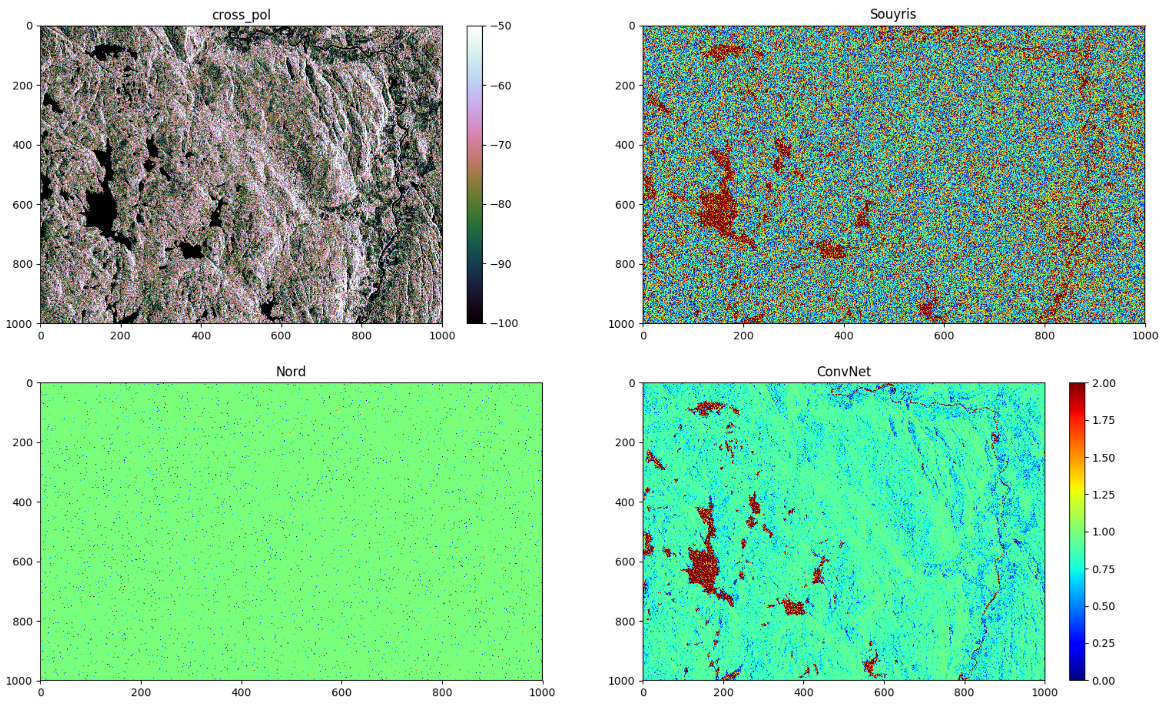


Figure 5.5: Relative error comparison, 1st network, 8th image (vegetation) of the data set. The first image represent the cross-pol value of the quad-pol covariance matrix (it is expressed in dB). The other three images represent the relative error pixel by pixel. We are comparing: Souyris, Nord and our trained network. The displayed error has a scale range between 0 and 200%.

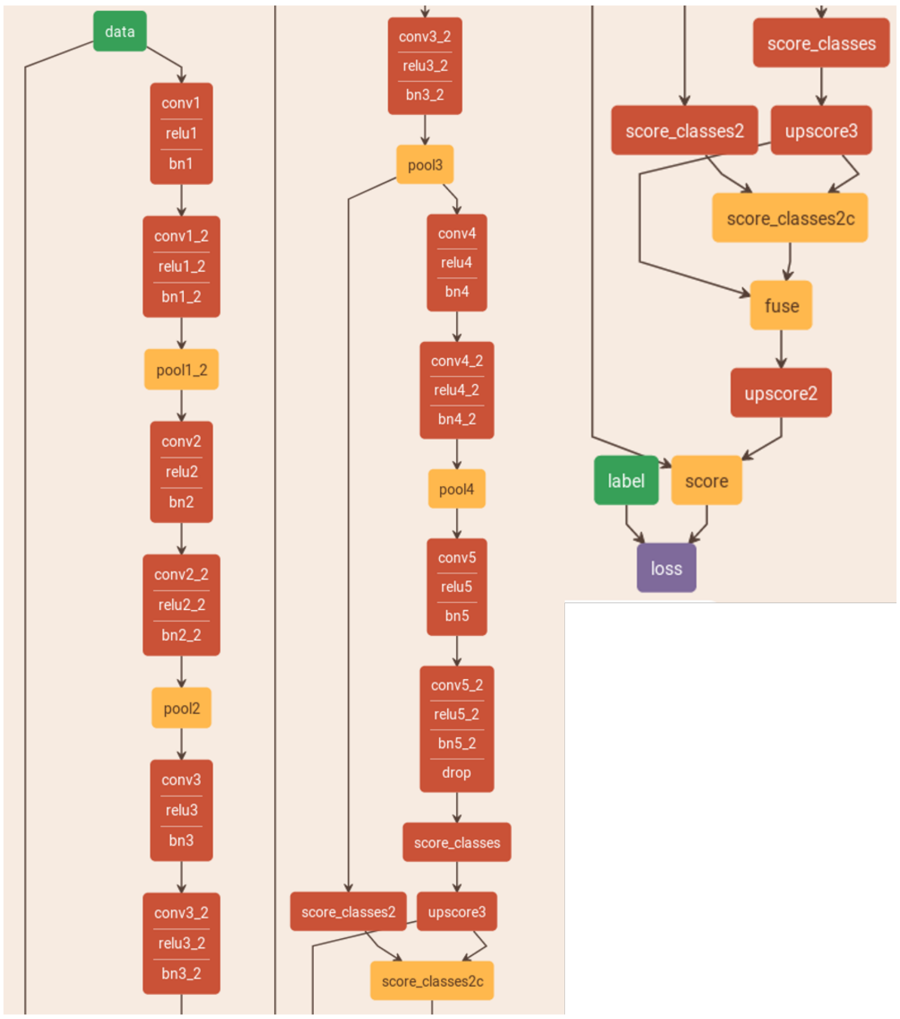


Figure 5.6: ConvNet 2nd implementation. Graphical representation generated by [41].

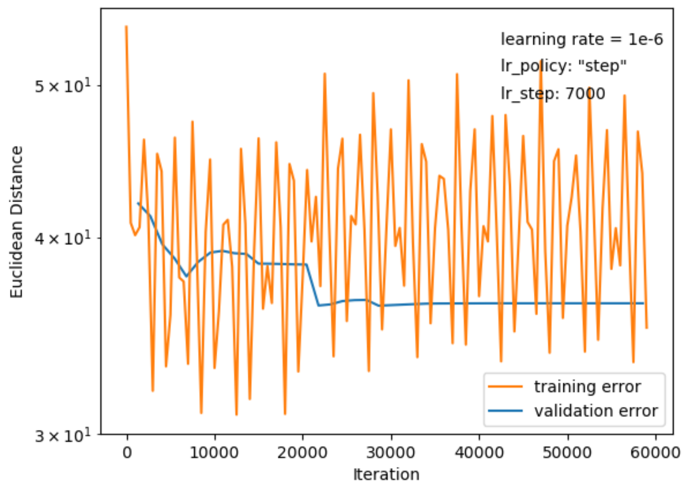


Figure 5.7: Loss function 2nd network experiment.

- Test set: 101,68
- 7th image: 29,00
- 8th image: 277,56

The euclidean distance, applied between the array of the real cross-pol values and the one of the predictions, gave as a result:

- Test set: 18,63
- 7th image: 12,44
- 8th image: 31,02

Considering the results, our network is performing quite worse than the first one unfortunately. This means that we were not able to adjust the parameters to obtain a better result.

We display then the relative error, regarding the cross-pol component, in figure 5.8. It has been computed on the 7th image. We can notice that this network is not able to perform as the previous image in this kind of data.

We tested then the network on the vegetation image. The result is shown in Figure 5.9. This is a surprising result, because this network was the worst one in all the previous comparison. In this case this network is making the smallest error. We can clearly see that the relative error of the cross-pol intensity is very high only in the areas where there is water (cross-pol value small).

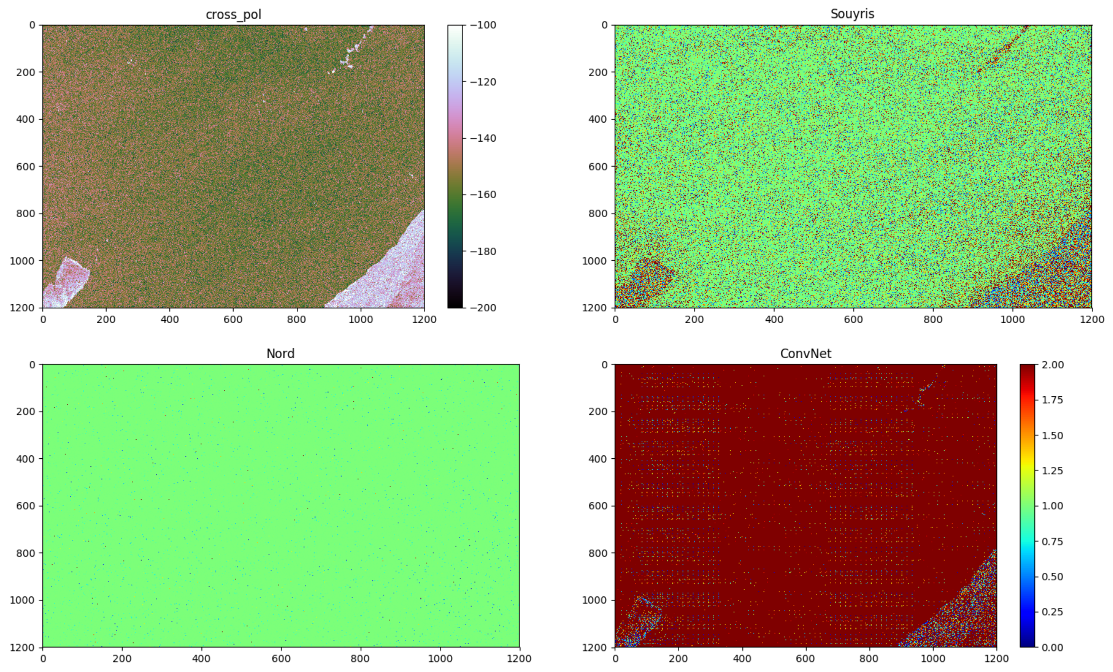


Figure 5.8: Relative error comparison, 2nd network, 7th image (sea ice) of the data set. The first image represent the cross-pol value of the quad-pol covariance matrix (it is expressed in dB). The other three images represent the relative error pixel by pixel. We are comparing: Souyris, Nord and our trained network. The displayed error has a scale range between 0 and 200%.

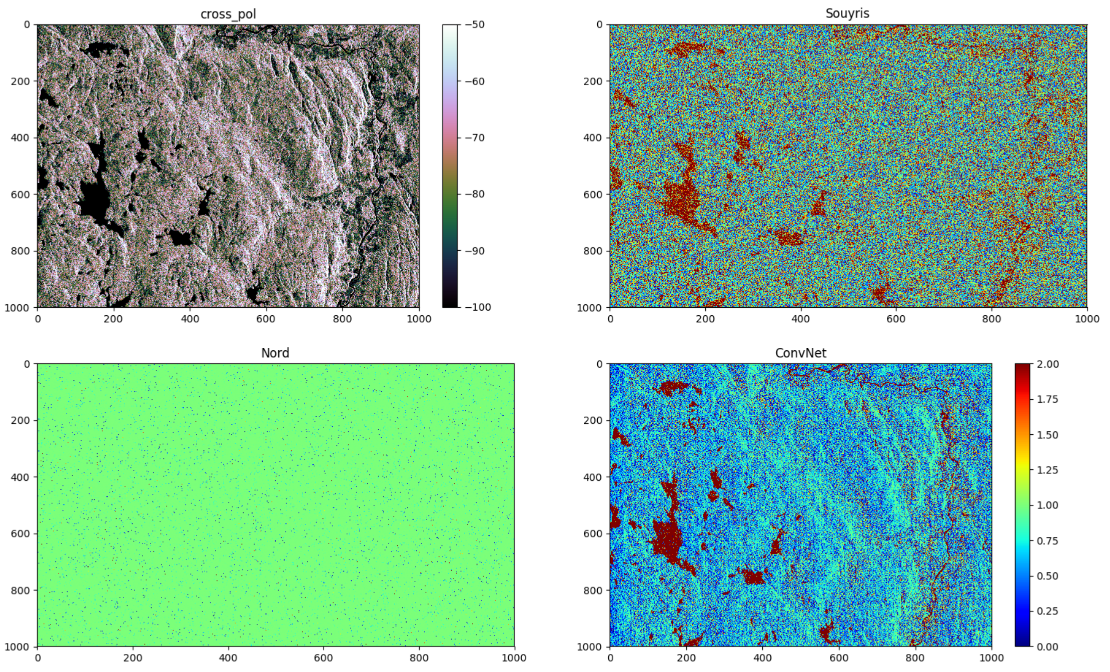


Figure 5.9: Relative error comparison, 2nd network, 8th image (vegetation) of the data set. The first image represent the cross-pol value of the quad-pol covariance matrix (it is expressed in dB). The other three images represent the relative error pixel by pixel. We are comparing: Souyris, Nord and our trained network. The displayed error has a scale range between 0 and 200%.

5.4 Summary of the results

We compare finally the results of the network in the reconstruction of the cross-pol intensity (Figure 5.10).

Notice that the first network is more capable to interpret the cross-pol intensity for sea ice, meanwhile the second network is more able to work on vegetation images.

In Figure 5.11 we resumed all the results we obtained.

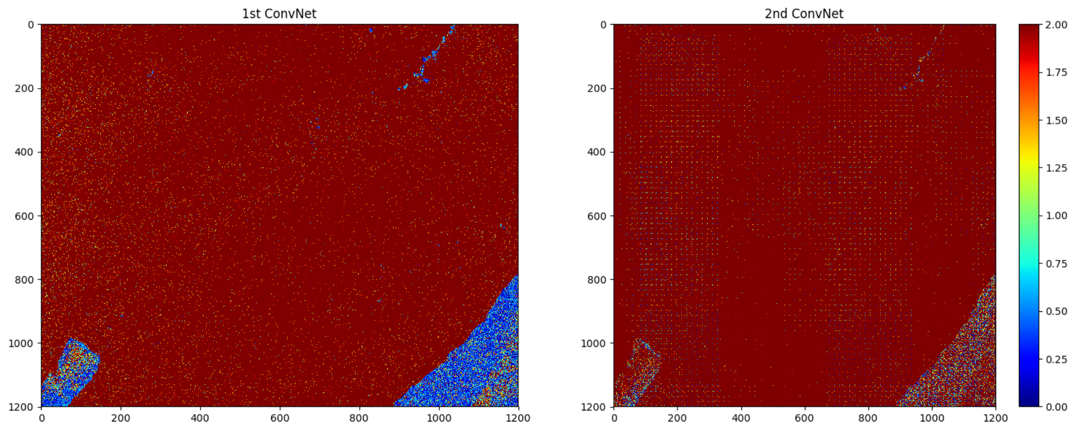
We display the Euclidean distance and the average distance value by value. The best performance for each measure are highlighted with yellow. The worst performance for each measure are highlighted with red.

Souyris is the method which is performing better over the data set. The second network is the one which is performing worse in general.

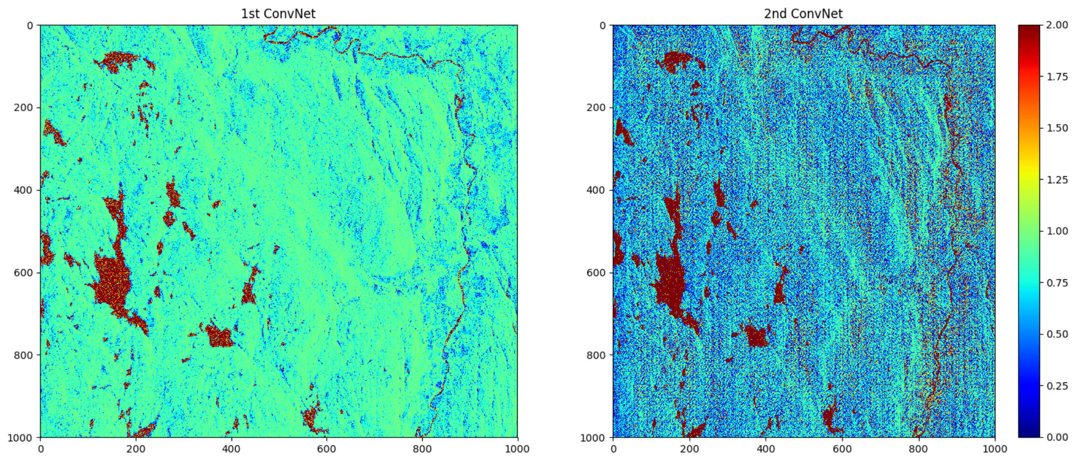
Considering the error computed on the validation test, it is surprising that the ConvNets perform so badly. Remember that this set derives by cropping one third of the 6 sea ice images in the table in figure 4.1, this should have caused a bit of correlation between train and validation set. Our first ConvNet is the best to predict the cross-pol intensity anyway.

Regarding the test over the 7th image, our networks were again the worst reconstruction methods, specially in the total euclidean distance. Our first network is doing a good work again on the cross-pol term. Analyzing deeper this comparison, it is the best to predict cross-pol intensity in the range [-130dB,-100dB], this is clear in figure 5.4.

With respect to the 8th image, we have to most surprising result. The 2nd ConvNet we built, that was performing very bad in all other situation, is the best one on reconstructing the cross-pol intensity, this can be seen in figure 5.9.

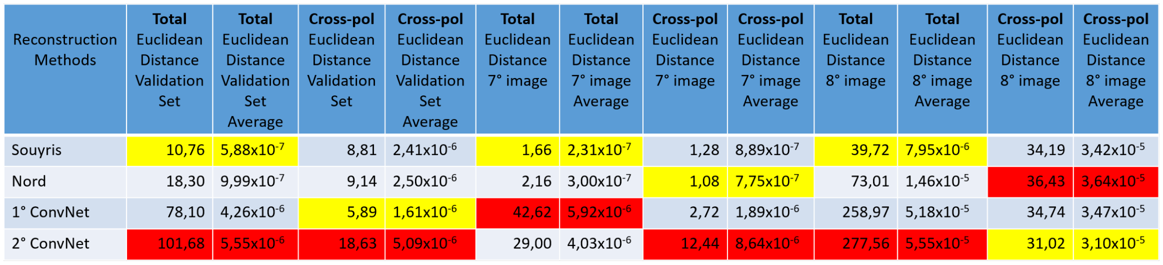


(a) 7th image



(b) 8th image.

Figure 5.10: Comparison between 1st and 2nd Network.



Reconstruction Methods	**Total** Euclidean Distance Validation Set	**Total** Euclidean Distance Validation Set Average	**Cross-pol** Euclidean Distance Validation Set	**Cross-pol** Euclidean Distance Validation Set Average	**Total** Euclidean Distance 7° image	**Total** Euclidean Distance 7° image Average	**Cross-pol** Euclidean Distance 7° image	**Cross-pol** Euclidean Distance 7° image Average	**Total** Euclidean Distance 8° image	**Total** Euclidean Distance 8° image Average	**Cross-pol** Euclidean Distance 8° image	**Cross-pol** Euclidean Distance 8° image Average
Souyris	10,76	5,88x10^{-7}	8,81	2,41x10^{-6}	1,66	2,31x10^{-7}	1,28	8,89x10^{-7}	39,72	7,95x10^{-6}	34,19	3,42x10^{-5}
Nord	18,30	9,99x10^{-7}	9,14	2,50x10^{-6}	2,16	3,00x10^{-7}	1,08	7,75x10^{-7}	73,01	1,46x10^{-5}	36,43	3,64x10^{-5}
1° ConvNet	78,10	4,26x10^{-6}	5,89	1,61x10^{-6}	42,62	5,92x10^{-6}	2,72	1,89x10^{-6}	258,97	5,18x10^{-5}	34,74	3,47x10^{-5}
2° ConvNet	101,68	5,55x10^{-6}	18,63	5,09x10^{-6}	29,00	4,03x10^{-6}	12,44	8,64x10^{-6}	277,56	5,55x10^{-5}	31,02	3,10x10^{-5}

Figure 5.11: Results resume.

6 Conclusions and further work

In this thesis a known problem in remote sensing has been faced: the reconstruction from compact-pol covariance matrix to a pseudo quad-pol one [8]. Convolutional neural networks (ConvNets) [9] are innovative systems which have given encouraging results in the field of image classification. We therefore thought to apply the ConvNets to the mentioned preexisting problem.

To the best of our knowledge, no previous works can be found in the literature where ConvNets have been used to perform regression anlysis for reconstruction from compact-pol data. So we had to design our method from scratch, and we could not benchmark it against similar methods as a basis for comparison.

The final results are encouraging and revealing that the intuition was right, even though they are not outstanding when compared to the outcomes of the other iterative methods.

Many choices can be done along the way of designing this method, for example in the preliminary data set manipulation and in the neural network composition, and many ways can be followed to try to achieve better results in the future. Given the complexity of the system, it is not so easy to tune its parameters and settings to optimize the obtained results.

Regarding the remote sensing data, it would have been possible to choose a different multi-look number (section 4.3), instead of using 9. We could also have gone deeper in the feature analysis of the reconstructed matrices. We only consider the euclidean distance between the covariance matrices and cross-pol intensity values, but we could also have taken into account other parameters analyzed in this thesis [8] to test the reconstruction performance.

Considering the convolutional neural network, there is a great number of hyper-parameters to be tuned and small changes to be applied to the structure of the system. All these modifications could make the network more efficient. We could for example increase the deepness of the network, since we did not encounter any overfitting problem. It could mean that our network was not complex enough to learn more. We believe however that the number of iterations in the training phase was adequate. Increasing the learning rate or the step-size (section 5.3.1) did not help. There are also some other parameters that could be changed in the network layers, for example the kernel sizes of the convolution layer or of the pooling layer.

As for the data set, lastly, some comments can be made: we chose some sea ice images for the training because we knew that the compared methods were working efficiently there [8]. It would be interesting to see how the network would perform if images of different types were included in the training set, such as images covering forests or urban areas. It would be another good idea to increase the size of the training set, together with a larger network.

References

[1] Attest, G. and Collins, M. J. (2013). On the use of Compact Polarimetry SAR for Ship Detection. ISPRS Journal of Photogrammetry and Remote Sensing, 80:1–9.

[2] Chuvieco, E. and Huete, A. (2010). Fundamentals of Satellite Remote Sensing. CRC Press Taylor and Francis Group.

[3] Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. MIT Press.

[4] Li, Y., Zhang, Y., Chen, J., & Zhang, H. (2014). Improved compact polarimetric SAR quad-pol reconstruction algorithm for oil spill detection. IEEE geoscience and remote sensing letters, 11(6), 1139-1142.

[5] MacDonald, Dettwiler and Associates Ltd (2014). Radarsat-2 product description. http://mdacorporation.com/docs/default-source/technical-documents/geospatial-services/52-1238_rs2_product_description.pdf?sfvrsn=10, Accessed: 3 September 2017.

[6] Souyris, J.-C., Imboa, P., Fjørtoft, R., Mingot, S., and Lee, J.-S. (2005). Compact Polarimetry Based on Symmetry Properties of Geophysical Media: The π/4 Mode. IEEE Transactions on Geoscience and Remote Sensing, 43(3):634–646.

[7] LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. Nature, 521(7553):436-444.

[8] Espeseth, M. M. (2015). Synthetic aperture radar compact polarimetry for sea ice surveillance (Master's thesis, UiT The Arctic University of Norway, Department of Physics and Technology).

[9] Long, J., Shelhamer, E., & Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 3431-3440).

[10] Schowengerdt, R. A. (2006). Remote sensing: models and methods for image processing. Academic press.

[11] Lee, J. S., & Pottier, E. (2009). Polarimetric radar imaging: from basics to applications. CRC press.

[12] Elachi, C., & Van Zyl, J. J. (2006). Introduction to the physics and techniques of remote sensing (Vol. 28). John Wiley & Sons.

[13] Raney, R. K. and Hopkins, J. (2011). A Perspective on Compact Polarimetry. IEEE Geoscience and Remote Sensing Newsletters, http:// www.grss-ieee.org/wp-content/uploads/2010/03/ngrs_SeptForWeb.pdf, Accessed: 31 May 2017.

[14] Charbonneau, F. J., Brisco, B., Raney, R. K., McNairn, H., Liu, C., Vachon, P. W., ... & Geldsetzer, T. (2010). Compact polarimetry overview and applications assessment. Canadian Journal of Remote Sensing, 36(S2): 298-315.

[15] Cloude, S. (2010). Polarisation: applications in remote sensing. Oxford University Press.

[16] Taylor, W. and Boerner, M. (2007). Basic Concepts in Radar Polarimetry. PolSARpro Lecture Notes, 3.

[17] Cloude, S. R., & Pottier, E. (1996). A review of target decomposition theorems in radar polarimetry. IEEE Transactions on Geoscience and Remote Sensing, 34(2):498-518.

[18] Moreira, A., Prats-Iraola, P., Younis, M., Krieger, G., Hajnsek, I., & Papathanassiou, K. P. (2013). A tutorial on synthetic aperture radar. IEEE Geoscience and remote sensing magazine, 1(1):6-43.

[19] Spudis, D. B. J., Butler, B., Carter, L., Chakraborty, M., Gillis-Davis, J., Goswami, J., ... & Patterson, W. (2010). RESULTS OF THE MINI-SAR IMAGING RADAR, CHANDRAYAAN-1 MISSION TO THE MOON PD.

[20] Raney, R. K. (2007). Hybrid-polarity SAR architecture. IEEE Transactions on Geoscience and Remote Sensing, 45(11):3397-3404.

[21] Boularbah, S., Ouarzeddine, M., & Belhadj-Aissa, A. (2012). Investigation of the capability of the compact polarimetry mode to reconstruct full polarimetry mode using RADARSAT2 data. Advanced Electromagnetics, 1(1):19-28.

[22] Cloude, S. R., Goodenough, D. G., & Chen, H. (2012). Compact decomposition theory. IEEE Geoscience and Remote Sensing Letters, 9(1):28-32.

[23] Shirvany, R., Chabert, M., & Tourneret, J. Y. (2012). Ship and oil-spill detection using the degree of polarization in linear and hybrid/compact dual-pol SAR. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, 5(3):885-892.

[24] Collins, M. J., Denbina, M., & Attest, G. (2013). On the reconstruction of quad-pol SAR data from compact polarimetry data for ocean target detection. IEEE Transactions on Geoscience and Remote Sensing, 51(1):591-600.

[25] Haykin, S. S. (2001). Neural networks: a comprehensive foundation. Tsinghua University Press.

[26] http://cs231n.github.io/neural-networks-1/ Accessed: 8 August 2017.

[27] Vapnik, V. N., & Chervonenkis, A. J. (1974). Theory of pattern recognition.

[28] Rojas, R. (2013). Neural networks: a systematic introduction. Springer Science & Business Media.

[29] LeCun, Y. (1989). Generalization and network design strategies. Connectionism in perspective, 143-155.

[30] Zhou, Y. T., & Chellappa, R. (1988, July). Computation of optical flow using a neural network. In IEEE International Conference on Neural Networks (Vol. 1998, pp. 71-78).

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

[32] Gregor, K., & LeCun, Y. (2010). Emergence of complex-like cells in a temporal product network with local receptive fields. arXiv preprint arXiv:1006.0448.

[33] Goodfellow, I. J. (2010). Technical report: Multidimensional, downsampled convolution for autoencoders. Technical report, Université de Montréal. 357.

[34] van Zyl, J. J. (2011). Synthetic aperture radar polarimetry (Vol. 2). John Wiley & Sons.

[35] http://caffe.berkeleyvision.org/ Accessed: 23 August 2017.

[36] Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., ... & Darrell, T. (2014, November). Caffe: Convolutional architecture for fast feature embedding. In Proceedings of the 22nd ACM international conference on Multimedia (pp. 675-678). ACM.

[37] Nord, M. E., Ainsworth, T. L., Lee, J. S., & Stacy, N. J. (2009). Comparison of compact polarimetric synthetic aperture radar modes. IEEE Transactions on Geoscience and Remote Sensing, 47(1):174-188.

[38] https://www.hiit.fi/u/ahonkela/dippa/node41.html. Accessed: 25 August 2017.

[39] http://searchnetworking.techtarget.com/definition/neural-network. Accessed: 25 August 2017.

[40] Bottou, L. (2010). Large-scale machine learning with stochastic gradient descent. In Proceedings of COMPSTAT'2010 (pp. 177-186). Physica-Verlag HD.

[41] http://ethereon.github.io/netscope/#/editor

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

[43] Ioffe, S., & Szegedy, C. (2015, June). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In International Conference on Machine Learning (pp. 448-456).

[44] http://mdacorporation.com/docs/default-source/technical-documents/geospatial-services/radarsat-2-application-look-up-tables.pdf?sfvrsn=4

[45] http://mdacorporation.com/docs/default-source/technical-documents/geospatial-services/radarsat-2-product-format-definition.pdf?sfvrsn=4

[46] Schuler, D. L., Lee, J. S., & De Grandi, G. (1996). Measurement of topography using polarimetric SAR images. IEEE Transactions on Geoscience and Remote Sensing, 34(5): 1266-1277.

[47] Hong, S. H., & Wdowinski, S. (2014). Double-bounce component in cross-polarimetric SAR from a new scattering target decomposition. IEEE Transactions on Geoscience and Remote Sensing, 52(6): 3039-3051.

[48] Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. Journal of machine learning research, 15(1): 1929-1958.

[49] Moen, M. A. (2015). Analysis and interpretation of C-band polarimetric SAR signatures of sea ice (Doctoral thesis, UiT The Arctic University of Norway, Department of Physics and Technology).

[50] Fors, A. S. (2017). Investigations of summer sea ice with X and C-band multi-polarimetric synthetic aperture radar (SAR) (Doctoral thesis, UiT The Arctic University of Norway, Department of Physics and Technology).