CONVOLUTIONAL NEURAL NETWORKS FOR CONTEXTUAL DENOISING AND CLASSIFICATION OF SAR IMAGES

CAROLYNE DANILLA
March, 2017

SUPERVISORS:
Dr. C. Persello
Dr. V.A. Tolpekin
Advisor: Mr. J.R. Bergado

CONVOLUTIONAL NEURAL NETWORKS FOR CONTEXTUAL DENOISING AND CLASSIFICATION OF SAR IMAGES

CAROLYNE DANILLA
Enschede, The Netherlands, March, 2017

Thesis submitted to the Faculty of Geo-Information Science and Earth Observation of the University of Twente in partial fulfilment of the requirements for the degree of Master of Science in Geo-information Science and Earth Observation.
Specialization: Geoinformatics

SUPERVISORS:
Dr. C. Persello
Dr. V.A. Tolpekin
Advisor: Mr. J.R. Bergado Msc

THESIS ASSESSMENT BOARD:
Prof. Dr. ir. A. Stein (Chair)
Mr. A. Strantzalis MSc (External Examiner, NEO, Netherlands Geomatics & Earth Observation B.V)

ABSTRACT

Classification of Synthetic Aperture Radar (SAR) images is a complex task because of the presence of speckle, which affects images in a way similar to a strong noise. As such, existing classification systems require that SAR images undergo a speckle filtering operation before classification. Additionally, SAR data are limited in terms of the number of bands to adequately discriminate between classes. Hence, additional features are commonly manually extracted from the images in order to characterize the spatial-contextual information. Classification systems may thus consider both feature extraction and speckle filtering to obtain better classification results. These processing steps are carried out sequentially, with speckle filtering followed by feature extraction and finally classification, but do not occur under a single framework. Convolutional neural networks enable us to perform these three SAR processing steps within a single framework.

In this thesis, we investigate the use of convolutional neural networks (CNNs) for automatic feature extraction, denoising, and classification of SAR images. We design the network architecture that consists of a sequence of convolutional layers with fully connected layers at the end, where the convolutional layers perform speckle filtering and feature extraction and the fully connected layers learn the classification rule from the extracted less noisy features. We carried out our experiments on 10m resolution Sentinel-1 multi-temporal images acquired in Flevoland for crop type classification. The sensitivity of the classifier to its hyperparameters was investigated, in addition to the speckle filtering mechanisms in the convolutional layers considering various aspects such as the pooling strategy and the effect of the varying hyperparameters. From the knowledge obtained in the experimental analysis, we formulated the optimal configuration of the proposed classification approach. The ability of our algorithm to reduce speckle was also evaluated against two speckle filters: Frost and Refined Lee filters using equivalent number of looks and ability to preserve edges as performance metrics. We then investigated the combination of our algorithm with Markov random fields (MRFs) for post-classification label smoothing, to further reduce the effect of speckle on the land-cover map and to improve classification accuracy. We compared convolutional neural network against two other state-of-the-art classifiers: support vector machines and random forests. Those techniques require speckle filtering and additional handcrafted spatial-contextual features. Several performance metrics were used such as the: overall accuracy, producer and user accuracy, map quality, computational time and complexity.

Experimental results show that the proposed convolutional neural network outperforms the alternative speckle filters in all performance metrics and that the addition of Markov Random fields for post classification improves map regularity and further reduce the residual noise. In addition, our approach outperforms the other two classifiers in all performance metrics considered, except for computational time and complexity. This confirms the ability of our approach to deal with speckle noise by learning spatial adaptive filter weights from the raw SAR image and for extracting useful spatial-contextual features for classification. Therefore, adopting such an approach can be beneficial in SAR image analysis by automating all the analysis tasks under a single framework for large scale monitoring systems in agriculture, land cover monitoring, and disaster monitoring, among others especially with freely available sentinel-1 images covering most parts of the world.

Index Terms: *Convolutional neural networks, synthetic aperture radar, speckle filtering, image classification, Sentinel-1*

ACKNOWLEDGEMENTS

I give thanks to the Almighty God for keeping me safe and healthy throughout the research period. I am forever grateful.
I thank the Netherlands Fellowship Program (NFP) for giving me this opportunity and for funding my studies. May God continue to bless you.
My deepest gratitude to NEO, Netherlands Geomatics and Earth Observation B.V for providing us pre-processed images and reference data. Your help was invaluable in realising this research.
Special thanks go to my supervisors Dr. C. Persello and DR.V. A. Tolpekin and to my advisor J.R. Bergado. Your comments and advice, technical or otherwise were very instrumental in the successful completion of this thesis. I really learnt a lot from all of you and I could never ask for better advisors.
I thank my fellow students, ITC staff, for giving me an opportunity to learn from you and to share my time with you. Special thanks go to my GFM classmates with whom I have shared most of my time in and out of class. Your companionship meant so much to me.
Finally, my outmost gratitude and love go to my grandmother. Thank you for always being there and for believing in me. And to the rest of my family, am truly grateful to have you all in my life.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1. MOTIVATION AND PROBLEM STATEMENT

1.1.1. Synthetic Aperture Radar systems and applications

Microwave remote sensing technologies such as SAR systems are quickly gaining popularity as unique tools for studying large geographical areas in a short time due to wide coverage area, high-resolution, very low sensitivity to weather, and a short revisit time (Lee, Pottier, & Thompson, 2009). Moreover, there is increased accessibility to free SAR data, such as those provided by Sentinel-1 under the European Space Agency open data policy, as well as from previous missions (ESA, 2010). In order for this imagery to be of value, they must be analyzed to extract useful information about the study area. Given the very large volume of available data, there is a need for automated algorithms that derive relevant information more quickly and efficiently.

SAR images are useful for many applications such as: in forestry for forest monitoring, classification, biomass estimation and tree height estimation, in agriculture for soil moisture extraction, crop type classification, and monitoring, and in disaster monitoring for oil spill detection, flood mapping, among others (ESA, 2010). For instance, in agriculture, information on crop acreage is important in the marketing of agricultural products both nationally and internationally, estimating regional productivity, validating crop insurance claims, and enforcing agricultural practices (McNairn & Brisco, 2004). However, gathering this information is no trivial task as the area is often too large for ground surveys, particularly at regional and national scales. When temporal information is required, as is the case for crop type classification, ground surveys are not feasible. Hence remotely sensed data such as SAR images are more suitable for deriving crop information as they are not affected by cloud cover and can, therefore, provide a reliable temporal coverage over large areas. The SAR signal responds to the crop structure: size, shape, and orientation of leaves, stalks, and fruit, and the dielectric properties of the crop canopy (McNairn & Brisco, 2004), thus providing a wide range of information that can be exploited to discriminate between different crop types.

1.1.2. SAR speckle filtering and contextual classification

SAR image classification is usually performed to produce land cover maps or as an intermediate step for further analysis. However, the classification is complicated by the presence of speckle. It is well known that speckle is a major problem that limits the exploitation of SAR images (Oliver & Quegan, 2004). Speckle is a characteristic of all coherent imaging systems. It behaves in a way similar to the presence of strong noise and complicates scene interpretation by reducing the accuracy of segmentation and classification. Speckle reduction filters are therefore commonly applied before scene classification (Dasari et al., 2016). Several approaches have been investigated for speckle reduction in SAR images. Lopez-Martinez et al. (2003) categorized them into two groups depending on which is their final purpose. The first group includes all those approaches assuming multi-dimensional SAR data as a type of diversity, combining all the channels to get speckle-free images (e.g. Novak & Burl, 1990; Lee, Grunes, & Mango, 1991). These approaches preserve spatial resolution, but result in the loss of correlation information

between channels. The second group includes techniques based on the spatial processing of SAR images (e.g. Lee et al., 1997; Frost et al., 1982) thereby degrading the spatial resolution. The authors concluded that these techniques either reduce the spatial resolution or do not preserve image information. Therefore, the behaviour of the filters based on these approaches has to be carefully studied in order to choose the most appropriate filter for a specific application. For instance, feature extraction requires good performance in preserving spatial resolution. In addition, the choice also depends on whether the images are to be subsequently processed automatically by image processing tools or manually by visual interpretation.

In classification, SAR images are limited in terms of the number of bands to adequately discriminate between classes. Hence, additional features are commonly extracted from the images in order to characterize the spatial-contextual information. Contextual information is useful for obtaining good results with noisy data, as is the case with SAR data (Frery, Correia, & Freitas, 2007). As such, classification systems may consider both feature extraction techniques and speckle filtering to obtain better classification results. These processing steps are carried out sequentially, with speckle filtering followed by feature extraction and then classification, but do not occur under a single framework. For crop type mapping, multitemporal images are required (Skriver, Svendsen, & Thomsen, 1999). The addition of the temporal component adds valuable information related to how the crop structure changes at different development stages. This information increases the ability of the classifier to differentiate between different crop types compared to single date crop type classification. Thus, SAR images acquired on different dates throughout the growing seasons are stacked together and used in the classification. Several studies on the multitemporal crop type classification with SAR images have been carried out achieving better results (Skriver et al., 2011; Skriver, 2012; Skriver et al., 1999; McNairn et al, 2009). Different classification approaches have been used to extract land cover and crop information from SAR data such as statistical methods based on the Wishart distribution (Lee, Grunes, & Kwok, 1994; Lee et al., 1997; Lee, Grunes, & Pottier, 2001), machine learning based approaches such as artificial neural networks (Chen & Huang, 1996), support vector machines (Fukuda & Hirosawa, 2001) and random forests (Du et al., 2015), and object-oriented classifiers (Jiao et al., 2014). Statistical based methods require that the data fulfil the relevant statistical distribution, such as the Gaussian and Wishart distribution, which is not always the case depending on the characteristics of scatterers in the scene, resulting in poor classification results. Object- oriented classifiers, on the other hand, reduce speckle by suppressing it to some extent during the image segmentation step, such that image segments are classified rather than individual pixels. However, this approach highly depends on the quality of image segmentation.

To further refine the results of classification, various regularization techniques have been used to improve the visual quality of the results and therefore further reduce noise in a post classification processing operation (Krylov & Zerubia, 2011). These include techniques based on mathematical morphology (Stevens, Monnier, & Snorrason, 2005), and Markov random fields (Krylov & Zerubia, 2011). Markov random fields consider label consistency within a defined neighbourhood and improve the classifier's robustness to the residue speckle after filtering. They have been used in the SAR image analysis, alongside existing classifiers such as support vector machines (Moser & Serpico, 2013) and maximum likelihood (Melgani & Serpico, 2003), and have proven to be effective in improving classification results.

1.1.3. Convolutional Neural Networks

In this thesis, we investigate the use of convolutional neural networks (CNNs) for automatic feature extraction, denoising, and classification of SAR images. In contrast to the classification methods described above, convolutional neural networks have the following characteristics that make them suitable for the denoising and classification task.

- They are made up of a set of adaptive filters which learn their weights directly from the raw data. The learned filters are equivalent to templates that are matched to every patch of the input image. When configured properly, the learnt filters can be used to discriminate between speckle and non-speckle, making the network to learn better representations of the data
- CNN classifiers are non-parametric in nature with high feature learning efficiency and automatic feature extraction abilities, unlike traditional classification methods which rely on manual feature engineering.
- They are able to learn from difficult data and to process large volumes of data

CNNs introduced by LeCun & Bengio (1995), represent an interesting method for image processing and form a link between the general feed-forward neural network and adaptive filters (Browne & Ghidary, 2003). They describe the architecture for applying neural networks to two-dimensional arrays, based on a spatially localized neural input. CNNs belong to the family of feedforward artificial neural networks (ANN), with one or more hidden layers as part of their architecture, high feature learning efficiency, and automatic feature extraction by training the network with representative feature samples (LeCun et al., 1998). They have been used in image classification and target recognition, with the advantage that there are several possibilities of combining the different neurons and learning rules to get better results (Nebauer, 1998). Here, speckle filtering, feature extraction, and classification can occur within a single framework. The beneficial properties of CNNs include weight sharing over the entire image which reduces the number of parameters, local connectivity for learning correlations between neighbouring pixels and equivariance to translation (i.e. if the input changes, the output changes similarly). These properties make it suitable to learn filters for speckle reduction from the images and to capture contextual information for the given classification task.

1.2. RESEARCH IDENTIFICATION

In a broader context, SAR images are useful for many applications that require image classification. However, the accuracy of the classification is always limited by the speckle inherent in SAR images. Existing classification methods, therefore, require speckle filtering to improve accuracy; a procedure that may lead to loss of information that is beneficial for the classification. Additionally, they rely on manually engineered features to provide additional spatial-contextual information to improve generalization capability. In this research, we develop a deep feature learning approach based on CNNs that can automatically perform speckle filtering, extract spatial-contextual features and classify SAR images under a single learning framework. Additionally, we combine the CNN approach with Markov random fields for post classification label refinement and study the effectiveness of this classification system in the analysis of multi-temporal dual polarized Sentinel-1 images for crop type classification.
We focus on the design, analysis, and evaluation of a CNN classifier for speckle reduction and classification. Different aspects of the CNN are analyzed such as the sensitivity to the hyperparameters and the effect of varying the hyperparameters on speckle filtering to come up with a better classifier that learns from a difficult data source. Finally, its performance is compared and evaluated against alternative speckle filters, and against classification methods using standard filtering methods (Refined Lee and Frost) and extracting handcrafted textural features such as gray level co-occurrence matrix statistics (Haralick et al., 1973) as a pre-processing step.

1.2.1. Research objectives

The main objective of the proposed research is to investigate a new method adopting a deep learning approach—based on convolutional neural networks—that learns the speckle reduction filters, extracts spatial-contextual features, and learns the classification rule for land cover classification directly from the data.

The main objective is achieved by the following sub-objectives.

1. To review the different network architectures used in literature and relate them to SAR image denoising and classification.
2. To design, implement and analyze the performance of a working CNN-based classifier
3. To compare the performance of the CNN speckle filter and classifier against the alternative speckle filters and classification methods.

1.2.2. Research questions

The following research questions will be answered for the sub-objectives above.

Sub-objective 1:

1. What network architectures have been proposed and studied in the literature and how do they work?
2. What is the network architecture to be used for SAR image denoising and classification?

Sub-objective 2:

1. What is the optimal CNN structure (e.g. number of hidden layers, the number of neurons in the hidden layer) of the classifier to address the speckle reduction and classification problem?
2. What are the optimal values for the model training parameters and the regularization parameters?
3. What performance measures (e.g. equivalent number of looks, classification accuracy, computation time and complexity) are relevant for assessing speckle filtering and the classifier?

Sub-objective 3:

1. Which approach performs better and in which aspects of the performance measures?
2. What is the difference between feature learning (with CNN) and manual feature engineering approaches in terms of performance?

1.2.3. Innovation

In this research, we develop a new convolutional learning system for speckle reduction and classification of SAR images. Very limited research has been done on the use of CNN in SAR image analysis, even though they have been successfully applied for various classification tasks in computer vision and pattern recognition. Moreover, there are only a few recent publications found on SAR image analysis with CNNs. Ding et al. (2016) and Chen et al. (2016) investigated CNNs for automatic target recognition of SAR images from the Moving and Stationary Target Acquisition and Recognition (MSTAR) benchmark dataset. Additionally, Zhao et al. (2016) studied CNNs for land cover classification of high resolution TerraSAR-X with patch-based training. Last but not least, Zhou et al. (2016) investigated CNNs for crop type classification on the AIRSAR San Francisco and Flevoland polarimetric SAR (PolSAR) dataset. These implementations only investigate CNNs for classification and target recognition with high resolution SAR images (at least up to 5m) and do not explore its ability to reduce speckle. This research exploits the advantage of CNNs speckle filtering and automatic feature extraction for classification of medium resolution SAR images. In our implementation, we also consider a multitemporal image series for testing the CNN in the context of a real world application such as crop type mapping with a large geographic extent using up to date reference data acquired in-situ. The classifier will be able to automatically perform speckle filtering, extract spatial-contextual features, and classify Sentinel-1 10m resolution SAR images end to end to produce land cover maps. Additionally, post classification processing with MRF will be applied to the results of a CNN soft classification so as to reduce the residual speckle that remains after filtering with CNN, and produce smoother land cover maps. This prototype has great potential to be used for monitoring systems in agriculture, land cover monitoring, and disaster monitoring, among others, especially with freely available sentinel-1 images covering most parts of the world. Better classification and speckle filtering results in terms of performance measures identified are expected. Hence, its performance was evaluated against other speckle filtering and classification methods.

1.3. PROJECT SET-UP

The research project was divided into the following phases:

- Review the different network architectures for CNNs
- Design, implementation, and analysis of the CNN classifier
- Set up and implement the alternative classification approaches
- Performance comparison

1.3.1. Project workflow

In the first stage of this research, we reviewed a number of CNN architectures. The focus of this review was to discuss the composition existing network architectures as used in previous implementations for classification both in competitions and for other related remote sensing image analysis tasks. We also reviewed relevant studies that have been carried out in CNN SAR image analysis. We chose one of the architectures deemed suitable for our SAR image analysis problem for the design and implementation phase.

In the second stage, we focused on design, implementation and performance analysis of a CNN classifier based on the selected suitable network architecture. This involved experiments to determine the overall optimal CNN architecture (number of hidden layers, filter size, patch size) with experiments on the network's sensitivity to hyperparameters and speckle filtering experiments, determining model training parameters; learning rate, learning rate decay and the regularization parameters. We also set up the Markov random fields (MRF) algorithm for post classification label smoothing. The MRF takes the soft classification results of a CNN as an input and provides the final classified map of the scene as an output. Finally, we designed two state-of-the-art classifiers: support vector machines (SVM) and random forests (RF) as compared approaches. The CNN and the compared classifiers were then trained with representative samples (training set) and assessed on independent samples that the classifiers had not seen before (test set). SAR images were then classified with the trained classifiers. Unfiltered multi-temporal images were used in CNN implementation while filtered multi-temporal images with handcrafted features were used for SVM and RF.

In the final stage, we compared CNN with existing speckle filters: Frost and Refined Lee filters. We also compared the performance of the CNN classifier against SVM and RF classifiers. Metrics such as the equivalent number of looks (ENL), edge preservation, computational time, overall classification accuracy, producer and user accuracy, and visual comparison of classification maps of the scene were used. Figure 1.1 shows the flowchart describing the main parts of the project workflow.

1.3.2. Thesis outline

The rest of the thesis is organized as follows:

- Chapter 2 presents a brief overview of existing network architectures as well as relevant studies in which they have been successfully applied
- Chapter 3 presents our formulation of the CNN-based classifier and the MRF algorithm for post classification processing, as well as the alternative approaches to be used for comparison
- Chapter 4 contains a description of the datasets used and data preparation, design and speckle filtering experiments for the CNN and MRF parameter tuning experiments.
- Chapter 5 includes a description of the final configuration of the CNN classifier and the comparison approaches and discuss the performance measures/metrics used for comparison
- Chapter 6 summarizes our findings in relation to the designed algorithm and discusses these findings
- Chapter 7 contains the conclusions and recommendations for future work based on the findings

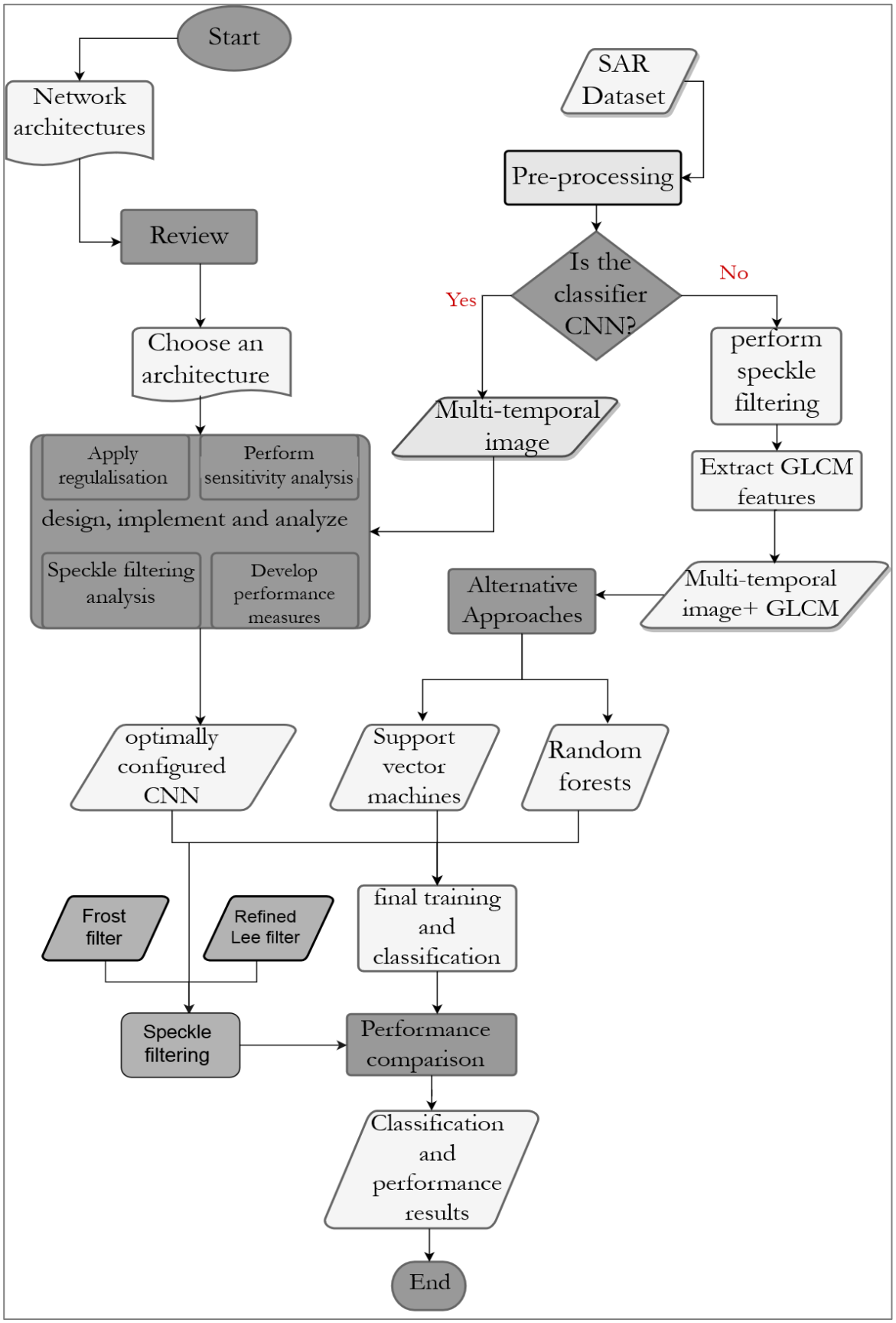


Figure 1.1: Flowchart of the project workflow.

CHAPTER 2

CONVOLUTIONAL NEURAL NETWORK ARCHITECTURES

This chapter presents a review of existing convolutional neural network architectures. The different network architectures, how they operate, and their composition and ability to solve the considered classification problem with SAR images are discussed, as well as relevant studies on which these architectures have been applied. One of the network architectures is then selected for the design and implementation phase.

2.1. CNN ARCHITECTURE OVERVIEW

A number of CNN architectures exist in the literature. This section reviews the commonly used architectures such CNN with Multilayer Perceptron (CNN), Fully Convolutional Networks (FCN), Recurrent CNN (RCNN), and Deconvolution networks. We discuss the architectural compositions as well as the relevant studies in which these architectures have been applied both in computer vision and in a remote sensing context.

2.1.1. CNN with Multilayer perceptron (CNN)

The CNN with Multilayer Perceptron; commonly known as Vanilla CNN is by far the most popular network architecture for traditional convolutional neural networks introduced by LeCun & Bengio (1995). This architecture was originally introduced to classify images, one label for one image as is the case for most image recognition tasks. It has since been adapted to perform pixel-wise labelling, making it suitable for classification tasks such as those in remote sensing applications. Typically, the architecture consists of a sequence of layers comprising the input layer, several hidden layers, and an output layer. The hidden layers are made up of one or more convolutional layers with non-linear activation layers, (optionally followed by a pooling layer for translation invariance) and finally one or more fully connected layers with non-linear activations stacked together to form the CNN architecture. The convolutional layers are the core building blocks in the architecture that learn spatial-contextual features. Each neuron in a convolutional layer is connected only to a local region of the input volume, the spatial extent of which is a hyperparameter called the receptive field of the neuron. The neurons are commonly known as filters or kernels and the receptive field the kernel size. Local connectivity, in addition to weight sharing among the filters, reduces the number of parameters to be estimated, which is very useful when dealing with high dimensional datasets. During the convolution operation, the filters slide across the input image, as it multiplies the values in the filters with the original pixel values of the image in an element wise multiplication operation. These multiplications are summed up to produce a single value for the central pixel and the process is repeated for every pixel in the image to produce a feature map for each filter.

A non-linear operation with an activation function such as rectified linear unit, hyperbolic tangent or the sigmoid function is then applied by the activation layer to the result from the convolution operation. A pooling layer usually follows the activation layer. The pooling layer summarizes the results of the convolution operation and takes the largest element (max pooling) or the average (average pooling) of all elements of the feature map within a window defining the spatial neighbourhood as output. Many pooling

strategies exist, but max and average pooling have been widely used. If down sampling is implemented by the pooling layer, then the subsequent layers perform pattern recognition at progressively larger spatial scales, with lower resolution. The motivation for subsampling feature maps obtained by previous layers is to make the CNN robust to noise and distortions (Jarrett et al., 2009). Additionally, reducing the spatial size of the representation is advantageous in a way that it largely reduces the number of parameters, controls overfitting and gradually builds up spatial invariance to translations of the input volume. Although this helps classification, subsampling results in output feature maps of lower resolution than the input feature maps as shown in Figure 2.1.

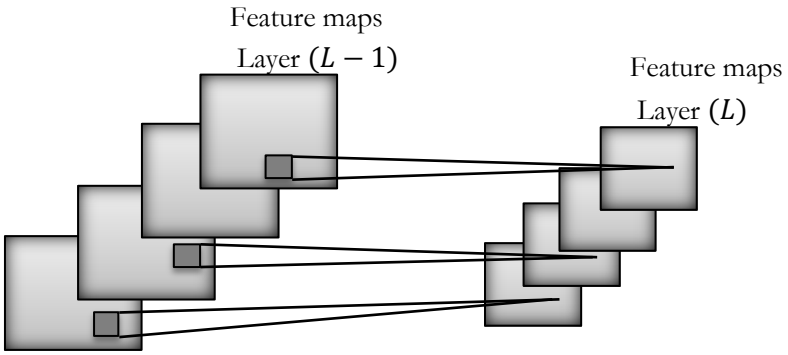


Figure 2.1: Illustration of pooling with subsampling.
If layer L is a pooling layer, all feature maps of the previous layer $L-1$ are pulled individually. Each unit in one of feature maps in layer L represents the average (average pooling) or the maximum (max pooling) within a fixed window of the corresponding feature map in the previous layer $L-1$.

In contrast to the convolutional layers, MLP layers have full connections to all neurons in the preceding and proceeding layers. Therefore, fully connected layers are associated with a lot of parameters that need to be estimated. The output of the convolutional layers is flattened and fed to the fully connected layers in a typical feed-forward network. Whereas convolutional layers perform local computations as defined by the size of the receptive fields, fully connected layers learn global image representations over all the activations in the preceding layers, subsequently learning the classification rule. In SAR image analysis, the convolutional layers perform feature extraction and speckle filtering concurrently while the MLP layers learn the classification rule from the extracted and filtered features. A typical CNN architecture is shown in Figure 2.2.

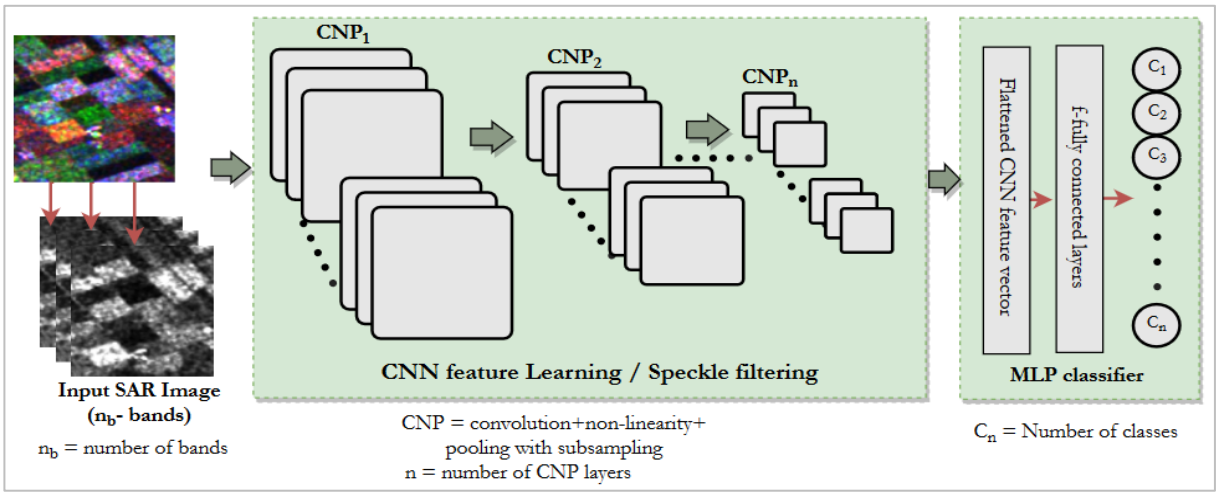


Figure 2.2: A typical CNN architecture in the context of SAR image analysis.

In general, the first layers of the CNN implement nonlinear template matching at a relatively fine spatial resolution, extracting basic features from the input dataset. Subsequent layers learn to recognize most of the spatial combinations found in previous features by generating patterns in the input, in a hierarchical manner (Bishop, 2006). This kind of architecture has been widely used for various classification tasks in computer vision and pattern recognition, in which it has been very successful, achieving state of the art results. The most widely recognized classification models implementing this architecture include AlexNet (Krizhevsky, Sutskever, & Geoffrey, 2012) for the ImageNet classification challenge and GoogLeNet (Szegedy et al., 2015), among others. Additionally, Bergado, Persello, & Gevaert (2016) classified the ISPRS 2D semantic labelling benchmark dataset (Vaihingen) with sub-decimeter resolution achieving state of the art results.

2.1.2. Fully convolutional networks

Fully convolutional networks (FCNs) consist of a sequence of convolutional layers with non-linear activation and pooling layers. The fully connected part in CNNs is discarded in this architecture. The last convolutional layer is usually connected to an activation function that performs multi-class labelling such as the softmax function (Bishop, 2006). Valid convolutions are also normally used in the convolutional layers, where only those parts of the convolution that are computed without the zero-padded edges are returned. Whereas in the standard CNN the label is assigned to a central pixel of the patch or image, FCNs learn from fully labelled patches. Additionally, FCNs accept inputs of arbitrary size, which is not possible with CNNs where all inputs must be of the same specified size, a constraint that comes with the addition of MLP at a deeper stage of the network. Another difference brought about by the addition of an MLP is that the fully connected layers in the CNN enable the network to learn global representations where the spatial relationships of the feature maps are discarded and each output of the last convolutional layer is connected to each neuron of the MLP layer. This is not the case with FCN, where the network tries to learn representations and make decisions based on the local spatial input.

FCNs produce coarse output maps when pooling with subsampling is applied, reducing the size of the input image by a factor equal to the pixel stride of the pooling layers. Several approaches are used to deal with this effect. For instance, the shift and stitch approach explained by Sherrah (2016) requires that the FCN be applied to shifted versions of the input which are then stitched together to form the output with the same dimensions as the input. Another approach gaining popularity for restoring the spatial resolution to that of the input image is the use of deconvolutional layers after all convolutional layers to upsample the downsampled output. The deconvolution operation is briefly explained in Subsection 2.1.3. FCNs have been used for semantic segmentation and scene labelling (Long, Shelhamer, & Darrell, 2015; Pinheiro & Collobert, 2014) and for feature detection (Sermanet et al., 2013). In a remote sensing context, Sherrah (2016) used FCNs for semantic labelling of aerial images for Vaihingen and Potsdam ISPRS benchmark datasets, achieving state of the art result.
For SAR image denoising and classification, the convolutional layers perform feature extraction, speckle filtering, and learn the classification rule. A typical architecture is as shown in Figure 2.3.

2.1.3. Deconvolutional networks

Similar to FCN, Deconvolutional networks (DNs) use the same idea, in which they convolve the input with a set of filters and generate feature hierarchies at each layer in the network (Zeiler et al., 2010). The two architectures are usually combined with FCNs followed by DNs to form a combined architecture that learns end to end. The last layer in the network is connected to an activation function such as softmax for multi-class labelling. Valid deconvolutions are normally used, with only those parts of the deconvolution operation computed without the zero-padded edges returned same as with FCNs. Whereas FCNs reduce the size of the input in the feedforward process involving several convolutional and pooling layers, DNs achieve the opposite by enlarging the feature maps with deconvolution and unpooling layers to produce

dense feature maps. The two main operations in deconvolution networks are unpooling and deconvolution. Unpooling layers perform the reverse operation of pooling in FCN in order to reconstruct the original size of the input. The deconvolution layers then densify the sparse feature maps from the unpooling operation in convolution-like operation, using learned filters. However, unlike convolutional layers which sum over multiple inputs within a window into a single output, deconvolution layers associate a single input with multiple outputs (Noh et al., 2015) as shown in Figure 2.4. The entire FCN-DN architecture is twice as deep and contains a lot of associated parameters, resulting in a harder optimization problem. It additionally requires that the pooled locations of the pooling operations be stored for the unpooling operation in deconvolution, which computationally intensive and increases the memory requirements. Thus, training such a deep network is not trivial and requires a lot of training examples. When used without FCNs, DNs can be trained in a way similar to FCNs to learn sparse representations of the input image (Zhang et al., 2013). Deconvolution networks are new in the computer vision and therefore very little research on them exists. They have been used to visualize the activated features in trained CNN models so as to understand their behaviour (Zeiler&Fergus, 2014), and for semantic segmentation (Noh et al., 2015). Zhang et al. (2013) implemented an all-Deconvolutional network for optical image restoration. The convolution and deconvolution operations are illustrated in Figure 2.4.

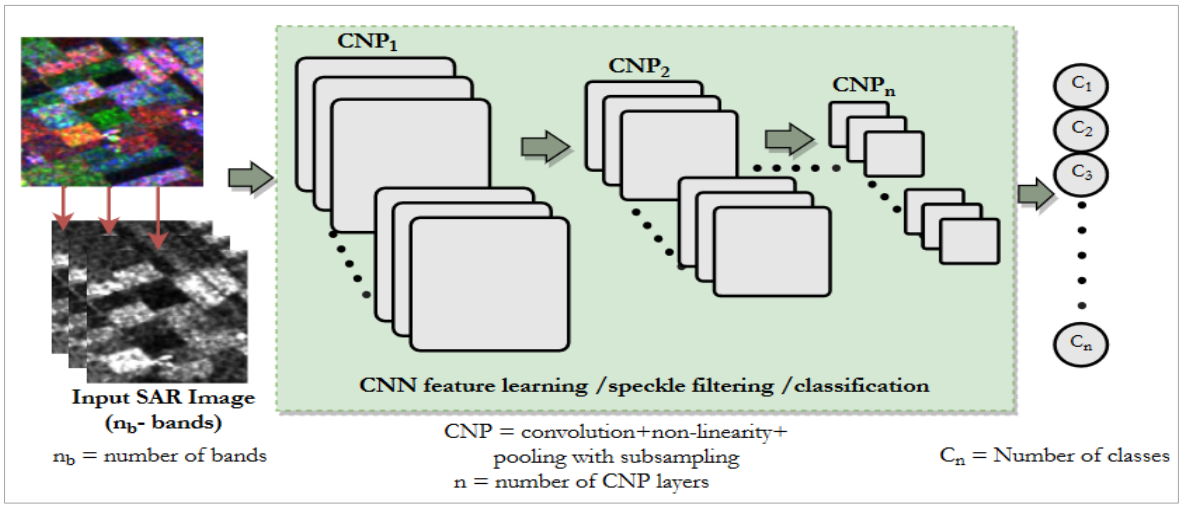


Figure 2.3: FCN architecture in the context of SAR image analysis. A softmax activation function can be added to the last convolutional layer for multi-class labelling.

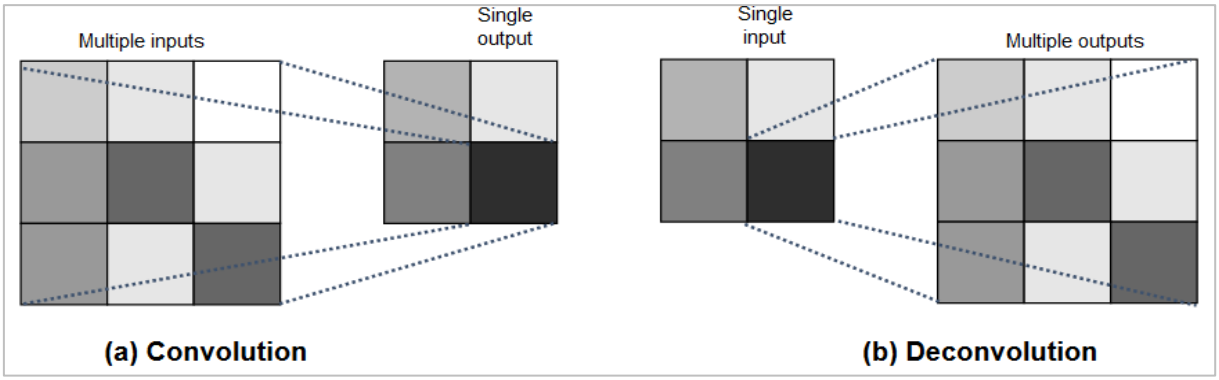


Figure 2.4: Illustration of the convolution (a) and deconvolution (b) operations.

2.1.4. Recurrent convolutional neural networks

Recurrent convolutional Neural Networks (RCNNs) take on a similar network architecture as the FCN shown in Figure 2.3 with one or more convolutional layers, non-linear activation, and pooling layers, but

with recurrent connections between layers (Pinheiro & Collobert, 2014). In the FCN architecture, both convolution and pooling operations are locally performed for image regions separately, with no contextual dependencies between the different image regions taken into consideration. For instance, convolutional layers only consider he local spatial patterns learned from the input data/image. RCNNs, on the other hand, are designed for learning connections between image regions at different spatial positions (also known as spatial-contextual dependencies) by using recurrent (feedback) connections. In this case, providing feedback from the output of a previous layer into the input of another layer allows the network to model label dependencies similar to an MRF-based classification, and to correct its own previous predictions. The network's hidden layer state is a function of the previous state, which can further be expanded as a function of all other previous states depending on the depth of the network. This makes RCNNs deep in time, and to retain all the past inputs, based on which, the network is able to discover correlations between the input data at different states of the sequence. Thus RCNNs require a larger memory so that outputs of all preceding layers can be stored in memory and their spatial correlations can be analyzed. For SAR image analysis, the convolutional layers perform similar tasks to those in FCN. The most successful applications of RCNNs in computer vision have been in modelling sequential data such as text and sound. Natural language processing (NLP) applications include language modelling (Mikolov et al., 2011), speech recognition (Chorowski et al., 2015; Graves, Mohamed, & Hinton, 2013), among others. A similar architecture was also applied for scene labelling by Pinheiro & Collobert (2014).

2.2. RELEVANT STUDIES IN SAR IMAGE ANALYSIS

As mentioned before in Subsection 1.2.3, SAR image analysis with CNNs is relatively new, with just a few recent publications on the subject. Ding et al. (2016) studied CNNs for target recognition with data augmentation. The Moving and Stationary Target Acquisition and Recognition (MSTAR) benchmark dataset was used in the implementation. While they consider the effect of speckle in the data augmentation process, the network's capability to adapt to the noise in the image were not fully studied or quantified. Zhao et al. (2016) implemented the patch-based CNN for land cover classification with TerraSAR-X images. A similar architecture is implemented in this thesis, where labelled patches are extracted from the input image to be classified for training, validation, and testing. The authors, on the other hand, used labelled images of the different classes as patches for training, validation, and testing. Zhou et al. (2016) investigated CNNs for crop type classification of the AIRSAR San Francisco and Flevoland Polarimetric SAR (PolSAR) data in the format of coherency and covariance matrix. Finally, Chen et al. (2016) implemented the FCN architecture for target classification with the Moving and Stationary Target Acquisition and Recognition (MSTAR) benchmark dataset.

2.3. CHOICE OF NETWORK ARCHITECTURE

As discussed in Chapter 1, Convolutional Neural Networks can automatically perform speckle filtering, extract contextual features and classify SAR images under a single learning framework by learning their filter weights directly from the data. We chose the CNN architecture to be the core algorithm for this research. This architecture is relatively easier to train using patch-based training, can be adapted easily to the specific needs of the data. The separation of the classification and speckle filtering in the network allows us to further explore and understand the CNN speckle filtering operation, which has not been studied before. Additionally, utilizing a fully connected layer at the end is a cheaper way to learn linear combinations of CNN features; effectively learning the classification rule. This network architecture has been explored for various classification tasks in computer vision for remote sensing image analysis, achieving state of the art results. In Chapter 3, we discuss the design of the classification algorithm exploiting the chosen architecture, as well as the alternative classification approaches for comparison and in Chapter 4 a description of the datasets, design experiments and final implementation of the classifier.

CHAPTER 3

ALGORITHMS DESIGN

This chapter discusses the design decisions of the selected network architecture as well as the MRF post processing algorithm and selected comparison approaches. The first section describes the network architecture and introduces the terms and notations that are used throughout the thesis. In the second section, we discuss the MRF algorithm and introduce the terms and notations related to the algorithm that are used throughout the thesis. Finally, in the last section, we introduce the classification approaches to be used for comparison with the CNN classifier. We briefly discuss the theory related to their design in accordance with existing literature. The design experiments and details on the composition of our chosen network architecture are intentionally left out and included in Chapter 4 and Chapter 5 respectively.

3.1. NETWORK ARCHITECTURE

3.1.1. CNN

A simple convolutional neural network is a sequence of layers transforming the input volume of an image into a classified map through a differentiable function in a pixel-wise classification. We use three main types of layers to build the convolutional network architecture: convolutional layers with non-linearities, pooling layers, and fully-connected layers. We stack these layers to form a complete architecture for performing pixel-wise classification on an image. The convolutional layers with non-linearities and pooling form a feature extractor, learning spatial-contextual features from the input image while reducing the inherent speckle noise. The first convolutional layer takes as input a square patch square patch P_{xy} of size $t \times t \times n_b$ centred on the pixel (x, y) to be classified. The square patch is taken from the input layer with n-dimensional multitemporal image of size $m \times m \times n_b$ where, $m \times m$ are the spatial dimensions of the input image and n_b is the number of bands in the input image. The convolutional layer has n_f filters of size $f_s \times f_s$ where f_s is smaller than the dimensions of the square patch. The size of the filters gives rise to the locally connected structure, which then perform a series of 2D convolutions over the input image to produce n_f feature maps. Border mode 'same' is applied to the input padded with zeros so that the output feature maps are the same size as the input.

The size of the feature maps is controlled by three parameters: the depth which corresponds to the number of filters we use for the convolution operation, the stride which is the number of pixels by which we slide our filters over the input image and zero padding. Padding the input images with zeros around the border enables us to apply the filter to the bordering elements of our input image. This feature allows us to control the size of the feature maps. Adding zero padding is also known as wide convolution and not using zero padding – narrow convolution.

An additional non-linear element-wise operation with an activation function is applied after every convolutional layer. This introduces non-linearity in the CNN since most of the real world data we want our network to learn from is usually non-linear while convolutional layers perform a linear operation. The pooling operation is defined by a spatial neighbourhood (window) of size $P_s \times P_s$ and takes the largest

element from the feature map within that window (Max pooling) as output. We set the stride for the convolution to 1 pixel and the stride for the pooling operation to P_s pixels for a network with subsampling. This produces non-overlapping pooling regions and downsamples the previous layer by a factor P_s. The pooling layer reduces the spatial size of the feature maps with a factor $(P_s)^n$ where n is number of pooling layers in the network with subsampling. Thus the size of the output from the convolution−activation−pooling layers is only affected by the stride of the pooling layers. Figure 3.1 illustrates the CNN hyperparameters with pooling.

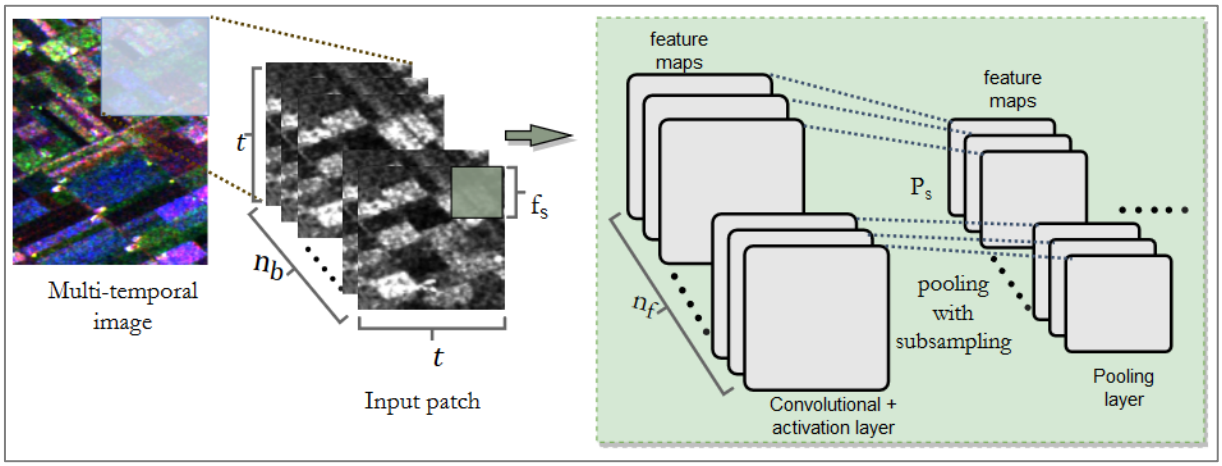


Figure 3.1: CNN hyperparameters.
The stride of the pooling operation Ps determines how much the spatial size of the feature maps is reduced in the pooling layer.

We then flatten the final output volume consisting of spatial-contextual features extracted by the CNN using a series of convolutions, nonlinearity, and pooling operations into a one-dimensional vector and connect it to the fully connected MLP layers. This part of the network architecture learns to classify the central pixel of the patch considering the feature maps extracted by the convolutional layers. It has full connections to the input and to its succeeding layer. The units in the MLP layers utilize the same activation function as the convolutional layers while the final output layer utilizes a softmax activation function for multi-class labelling (Bishop, 2006). Formally, the softmax function is given by:

$$h_i = \frac{e^{y_i}}{\sum_{c=1}^{c} e^{y_c}} \tag{3.1}$$

Where, $c \in \{1, 2, \ldots, C\}$ are the classes of interest, and h_i, y_i, y_c refer to class scores and values for each of the class labels (Goodfellow et al., 2016, pp. 176 – 180). The classifier then assigns the label with the highest score to the pixel being classified. The output layer consists of C units equivalent to the number of classes in the input. The class labels in this layer are transformed into vector encoding with all elements of the vector being zero except the index of the element corresponding to the assigned class label for a hard classification or into a vector encoding with all elements of the vectors being the probability of the pixel belonging to each of the class labels. In the former, for example, labels A, B, C, D are transformed into (1, 0, 0, 0), (0, 1, 0, 0), (0, 0, 1, 0), and (0, 0, 0, 1) vectors respectively for four pixels belonging to the four different classes of interest. In the latter, labels A, B, C, D are transformed into (P_a, P_b, P_c, P_d) where P_a, P_b, P_c, P_d are the probabilities of a pixel belonging to the classes A, B, C, D and they sum up to 1.

3.1.2. Training the CNN classifier

The network is trained by minimizing a categorical cross entropy objective (loss) function:

$$E_n(w) = -\sum_i z_i \log(x_i) \tag{3.2}$$

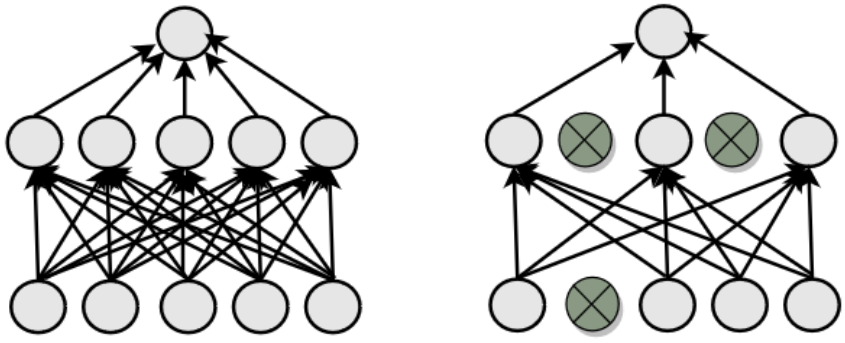
Where, z_i is the i^{th} element of the vector encoding of the true label and x_i is the i^{th} element of the output layer computed over n samples. Stochastic gradient descent (SGD) with momentum and learning rate is used for optimizing the loss function. This approach is discussed in detail by Goodfellow et al., (2016 pp. 288 - 292). An SGD algorithm minimizes the objective function $E_n(w)$ by updating the weights in the opposite direction of the gradient of the loss function with respect to the weights such that:

$$w^{(T+1)} = w^T - \eta \nabla_W E_n(w) \tag{3.3}$$

The learning rate η defines the proportion of the gradients used to update the weights and $\eta > 0$. Standard SGD evaluates the gradient of the loss function using a single sample taken from the training set. However, using a mini batch of several training samples as in batch SGD is preferred where the average gradient is computed. Each time the gradient is evaluated using a mini batch, an iteration is completed. The algorithm takes a step towards minimizing the loss function with each iteration, with a full pass over the entire training set making an epoch. Batch SGD is known to work better with noisy data since it computes the average gradient for the mini batch. Momentum is used to speed up the learning rate. Using this optimization approach, we set the values for the following parameters; learning rate η, momentum α and learning rate decay term η_d. Additionally, regularization is applied to prevent overfitting and improve the generalization capability of the network with an L2- weight decay term for CNN layers added to the loss function, early stopping criteria, and dropout (Srivastava et al., 2014) for both CNN and MLP layers. An L2 weight decay term λ_w acts as a penalty on the loss function and is proportional to the sum of the square of the weights. Hence the objective function in equation 3.2 takes the form:

$$J(w) = E_n(w) + \lambda_w ||w||^2 \tag{3.4}$$

Early stopping prematurely interrupts the training of the network in order to avoid overfitting, while as with dropout, hidden units of a layer along with their connections are randomly dropped (see Figure 3.2) with a probability of $1 - p_r$ in every epoch, where p_r is set as the probability of retaining a unit.



(a) Standard Neural Network (b) After applying dropout

Figure 3.2: The effect of applying dropout for regularization.

We set the values of the parameters: learning rate, learning rate decay, weight decay, early stopping criteria and dropout using a holdout validation method as explained in Chapter 4 under Section 4.2.

3.2. MARKOV RANDOM FIELDS FOR POST-CLASSIFICATION PROCESSING

In this section, we discuss briefly the theory behind the design of the Markov random fields algorithm for post-classification label refinement.

3.2.1. Markov random fields

Markov random fields (MRFs) model the contextual dependencies of the labelled pixels in an image. The basic building blocks of an MRF are discussed are discussed as follows. Let us define set s (image) containing m pixels in which each random variable $w(1 \leq i \leq m)$ takes a label from label set C with user defined information classes $C_1, C_2, \ldots, C_n$. We define the configuration w for set s as $w = \{w_1, w_2, \ldots, w_m\}$, where $w_r \in C\ (1 \leq r \leq m)$. The configuration w with respect to a neighborhood system N is an MRF if its probability density function satisfies the following three properties (Tso & Mather, 2009 pp. 255 - 257):

- Positivity: $P(w) > 0$ for all possible configurations of w.
- Markovianity: $P(w_r|w_{s-r}) = P(w_r|w_{N_r})$, and
- Homogeneity: $P(w_r|w_{N_r})$, is the same for all sites

$s - r$ includes all pixels in the set s excluding r, w_{s-r} denotes the set of labels at the sites in $s - r$ and N_r denotes the neighbors of site r.

The first property is usually satisfied in practice with the joint probability $P(w)$ determined by the local conditional properties (Besag, 1974). Markovianity indicates the local characteristics of w. In other words, a label interacts with only the neighboring labels. Hence, a neighborhood system plays an important role in MRF as it defines the neighboring pixels that interact with the label given to a central pixel r. Finally, the homogeneity property specifies that the conditional probability for the label of a site r, with respect to the labels of the neighboring pixels is the same regardless of the relative position of site r in s.

The neighborhood system defining the first-order neighbors of a pixel as used in our MRF is as shown in Figure 3.3 (a). It can be extended to define second-order neighbors by including the diagonal neighboring pixels. A clique is part of a neighborhood system. It is a subset of neighboring sites. It can be a single site, a pair of neighboring sites or a triple of neighboring sites in a neighborhood system. In this research, we consider a pair of sites cliques C_2 of the horizontal and vertical neighbor pairs shown in Figure 3.3 (b) for the prior energy function of the MRF model.

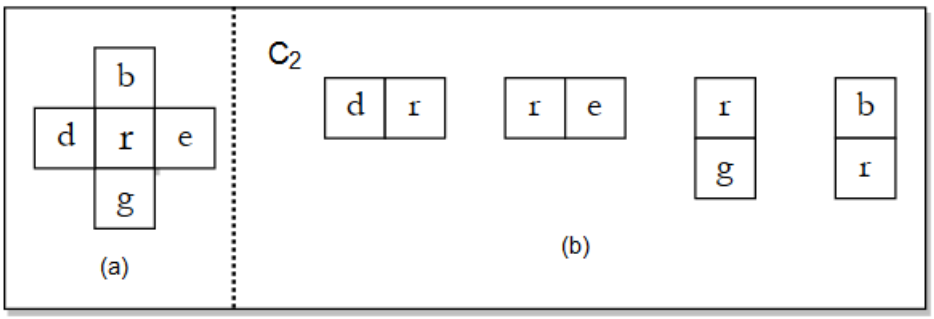


Figure 3.3: First-order neighbourhood system (a) with associated Cliques: pair-site cliques C_2 (b).

3.2.2. Maximum A Posteriori (MAP) estimate and MRF energy functions

To solve the pixel labelling problem, a Maximum A Posteriori (MAP) solution is obtained by minimizing the global posterior energy. In accordance with Bayesian formulae, the MAP estimate is obtained as follows:

$$\hat{w}_r = \text{argmin}\{U(w_r|d_r)\} \tag{3.5}$$

Where, $\hat{w}_r$ is the optimal class label and $U(w_r|d_r)$ is the posterior energy function. The posterior energy function for modelling the contextual and likelihood information in MRF is constructed using Bayesian formulae as follows:

$$U(w_r|d_r) = U(d_r|w_r) + U(w_r) \quad (3.6)$$

w is the membership value and d is the dataset. The prior energy function $U(w_r)$ is based upon pairwise clique potential functions in the neighbourhood system for the class label w_r in the MRF framework. The conditional energy $U(w_r|d_r)$ is based upon the likelihoods of the pixel r to belong to class w_r obtained as output of a soft classification with any classifier. An additional parameter λ is added to equation 3.6 which controls the balance between the two energy functions as shown in equation 3.8. This the MRF smoothness parameter and its value range between 0 and 1.

$$U(w_r|d_r) = (1-\lambda)U(d_r|w_r) + (\lambda)U(w_r) \quad (3.7)$$

We determine the value of the smoothing parameter as described in Section 4.4 in Chapter 4.

3.2.3. Simulated Annealing

Simulated annealing (SA) is an iterative relaxation algorithm introduced by Kirkpatrick et al. (1983 pp. 671-680) for solving optimization problems. We use the simulated annealing (SA) algorithm for minimizing the energy function in order to approximate the MRF-MAP estimate. The SA optimization algorithm starts at a high temperature T_0 (also known as the initial temperature), which is gradually decreased according to a pre-determined cooling schedule. This process is iterative until the system converges to a global minimum energy solution, i.e. $T_0 \to 0$ and no more pixels are updated. Then an optimal solution is found. Energy optimization is thus controlled by two parameters; initial temperature T_0 and updating temperature T_{upd}, which controls the temperature decrease. Therefore, we set a cooling schedule

$T_{k+1} = T_k \cdot T_{\text{upd}}$, that requires two parameters: initial temperature T_0 and updating temperature T_{upd}. The optimal values of the SA parameters T_0 and T_{upd} are determined as explained under Section 4.4 in Chapter 4.

3.3. COMPARISON ALGORITHMS

In this section, we briefly introduce the theory behind the design of the classification algorithms chosen for comparison with our proposed CNN algorithm.

3.3.1. Support vector machines

Support vector machines (SVMs) are non-parametric discriminative machine learning classifiers that have their roots in statistical learning theory (Cortes & Vapnik, 1995). They have been widely applied to various classification problems in computer vision such as for object (Pontil & Verri, 1998) and character (Nasien et al., 2010) recognition, and in SAR image classification by Fukuda & Hirosawa (2001), among others. The basic concept of SVMs for a linear, non-separable case aims at the definition of a hyperplane, in a multidimensional feature space mapped by a kernel function, in which all the computations are completed in the original feature space (Vapnik, 2006). The redistributed data enable the fitting of a linear hyperplane between the training samples of two classes. The optimization problem attempts to maximize the margins between the hyperplane and the closest training samples and to minimize the error of the training samples that cannot be differentiated (Vapnik, 1998). The influence of the non-separable samples is controlled by the regularization parameter C. A detailed introduction to the concept of SVMs is given by Burges (1998) and a brief discussion of key concepts is given as follows.

Consider a training set of L samples with (x_i, y_i) pairs. Let $x_i \in \Re^d(1,2,\dots,L)$ for a binary classification problem in d-dimensional feature space $\Re^d$ of the training sample with the corresponding class labels $y_i \in \{1,-1\}$. The separating hyperplane $f(x)$ is described by the norm vector $w \in \Re^d$ and the bias

$b \in \Re$, where $|b|/||w||$ is the distance between the hyperplane and the origin, with $||w||$ as the Euclidean norm from w:

$$f(x) = w \cdot x + b \tag{3.8}$$

The support vectors are located on two hyperplanes $w \cdot x + b = \pm 1$ that are parallel to the separating hyperplane. The goal is to maximize the margins between the closest samples and the hyperplane, which leads to the following optimization problem:

$$\min \left[\frac{1}{2} ||w||^2 + C \sum_{i=1}^{l} \xi_i \right] \tag{3.9}$$

Subject to:

$$y_i(w \cdot x_i + b) = 1 - \xi_i$$

Where, ξ_i are the slack variables and C is a regularization parameter, which handles misclassified samples in non-separable cases. The C parameter controls the number of training samples that lie on the wrong side of the hyperplane as well as the shape of the solution and consequently affects the generalization capability of the classifier. The linear SVM approach is extended to the nonlinear cases by a kernel function which nonlinearly maps the data into a high dimensional space such that it can be linearly separated. The final hyperplane decision function is then defined as:

$$f(x) = \sum_{i \epsilon sv} \alpha_i y_i K(x \cdot x_i) + b \tag{3.10}$$

And the final optimization problem is written as:

$$\max_{\alpha} \left\{ \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{j=1}^{l} y_i y_j K(x_i \cdot x_j) \right\} \tag{3.11}$$

Subject to:

$$\sum_{i=1}^{l} y_i \alpha_i = 0, \qquad 0 \le a_i \le C, \qquad \forall i = 1, \dots, l$$

Where, α_i, α_j are Lagrange multipliers for the dual form and a_i is bounded by the value of the regularization parameter C, and K denotes the kernel function.
A widely used kernel function is the radial basis function (RBF) kernel (Vapnik, 1998) defined as

$$K(x_i \cdot x_j) = e^{-\frac{||x_i - x_j||^2}{2\gamma^2}} \tag{3.12}$$

SVM training requires the estimation of the kernel parameter γ and the regularization parameter C. To extend the classification problem from binary to multi-class, as is the case for most remote sensing applications, two main strategies are commonly used. A classification problem with C classes is divided into several binary sub problems. The one-against-one (OAO) method generates $c(c-1)/2$ individual classifiers, one for each pair of classes and a majority vote is applied to the $c(c-1)/2$ rule to compute the final class membership. Alternatively, the one-against-all strategy generates C classifiers, separating each class from the remaining and the absolute maximum of equation 3.11 defines the final class label.

3.3.2. Random forests

Random forests (RF) is an ensemble classification method introduced by Breiman (2001) that constructs a set of classifiers instead of one classifier to form a forest of classifiers and then classifies new data points by taking a vote on their predictions. The classification strategy of the RF classifier is based on the assumption that independent classifiers produce individual errors, which may not occur in the other classifiers (Waske et al., 2009). A decision tree is grown using randomly selected features at a node. The node is then split using the best among the subset of the randomly selected features. Grown trees are not pruned (Archer & Kimes, 2008). The computational complexity is simplified by the reduction of the

number of input features at each internal node. This enables RF to deal with high-dimensional datasets (Waske & Braun, 2009).

To initialize RF algorithm, the user must define two parameters. These parameters are the number of trees to grow, N, and the number of features, M, used to split at each node respectively. Considering a training set with S samples, T bootstrap samples are drawn from two-thirds of the training samples while the remaining samples are left out and considered out-of-bag (OOB) data. The OOB data can be used as an independent check on the classification error rate, as the forest of trees is formed and to also measure importance of features as used in the classification. After the forest is complete, a case can be classified by taking the majority vote among all trees in the forest. An unpruned tree from each bootstrap sample is grown such that at each node, M features are randomly selected as a subset of the features, and the best split from among those features is chosen. Features that provide low a correlation between classes also have high predictive power (Horning, 2010), thus they are preffered. The split at each node is performed according to a criterion such as the Gini index (Breiman, 2001). Gini index measures class homogeneity which describes the level of purity for a given node. The aim is to find the largest homogeneous subclass within the training set and discriminate it from the remaining training dataset (Zambon et al., 2006). As Gini index increases, class heterogeneity also increases and as it decreases, class homogeneity increases. If a child node has a Gini index less than that of the parent node, then the split is successful. Tree splitting is terminated when Gini index is zero, which means that only one class is present per terminal node (Watts et al., 2011). Once all the trees are grown in the forest, a new dateset can then be predicted based on the predictions of the trees in the forest on the training set. The Gini index is given by

$$\text{Gini}\ (t) = \sum_{i=1}^{c} P_{w_i}\left(1 - P_{w_i}\right) \tag{3.13}$$

Where C is the number of classes, and P_{w_i} is the probability of class w_i at node t and is defined as:

$$P_{w_i} = \frac{S_{w_i}}{S} \tag{3.14}$$

Where, S_{w_i} is the number of samples belonging to class w_i and S is the total number of training samples.

In general, the RF algorithm for image classification works as follows. Suppose N is chosen as 100, the RF algorithm generates 100 trees which produce 100 different classification results for a particular pixel. If a particular pixel is classified as Maize by 80 trees, Potatoes by 15 trees and Grassland by 5 trees. The pixel is assigned to the class with the majority trees.

CHAPTER 4

DATASETS AND EXPERIMENTS

In this chapter, we describe the study area and the considered datasets for the implementation of the designed algorithms, as well as the setup for various experiments to determine values for CNN hyperparameters, the learning and regularization parameters, and other experiments relating to the design of the CNN algorithm for speckle filtering and classification. Also, experiments to determine the optimal values for the MRF parameters are briefly discussed.

4.1. STUDY SITE AND DATASET

4.1.1. Study site and dataset description

The study area comprises the municipalities of Dronten and Noordoostpolder in Flevoland, The Netherlands. Flevoland is a province located in the central part of The Netherlands, with large scale agriculture as the main economic activity. Some of the crops grown in the area include wheat, winter barley, and maize, among others with regular shaped crop fields. Figure 4.1 shows a map of the study area. As discussed in subsection 1.1.2, the structure of crops changes at different growth stages. As such, crop type classification requires a multitemporal dataset with images obtained at different dates in order to adequately discriminate between different classes of interest. Hence, we obtained five dual polarized Sentinel-1 C-band Ground Range Detected (GRD) SAR images with a pixel size of 10m and equivalent number of looks (ENL) of 4.9, covering Flevoland area. SAR images are of different dates corresponding to different stages of crop development stages such as early stage, mid-season, and near harvesting or late season. The image dates used for this study are as shown in Table 4.1.

Table 4.1: Acquired image with dates, polarizations, and growth stage

Image No.	Acquisition date	Polarization	Growth stage
1 &2	30/05/2016	HV/VV	Early season
3 &4	11/06/2016	HV/VV	Early season
5&6	05/07/2016	HV/VV	Mid-season
7&8	17/07/2016	HV/VV	Mid-season
9&10	10/08/2018	HV/VV	Late season

In addition to the images described above, a reference dataset was provided by NEO, Netherlands Geomatics and Earth Observation B.V., containing shape files that were used as representative samples for the crop types in the study area. This dataset contains 264 polygons for the 14 classes in the study site. Table 4.2 provides a summary of the polygons per crop contained in the shapefile and Figure 4.2 shows the all the reference polygons overlaid on the grayscale SAR image of the study area.

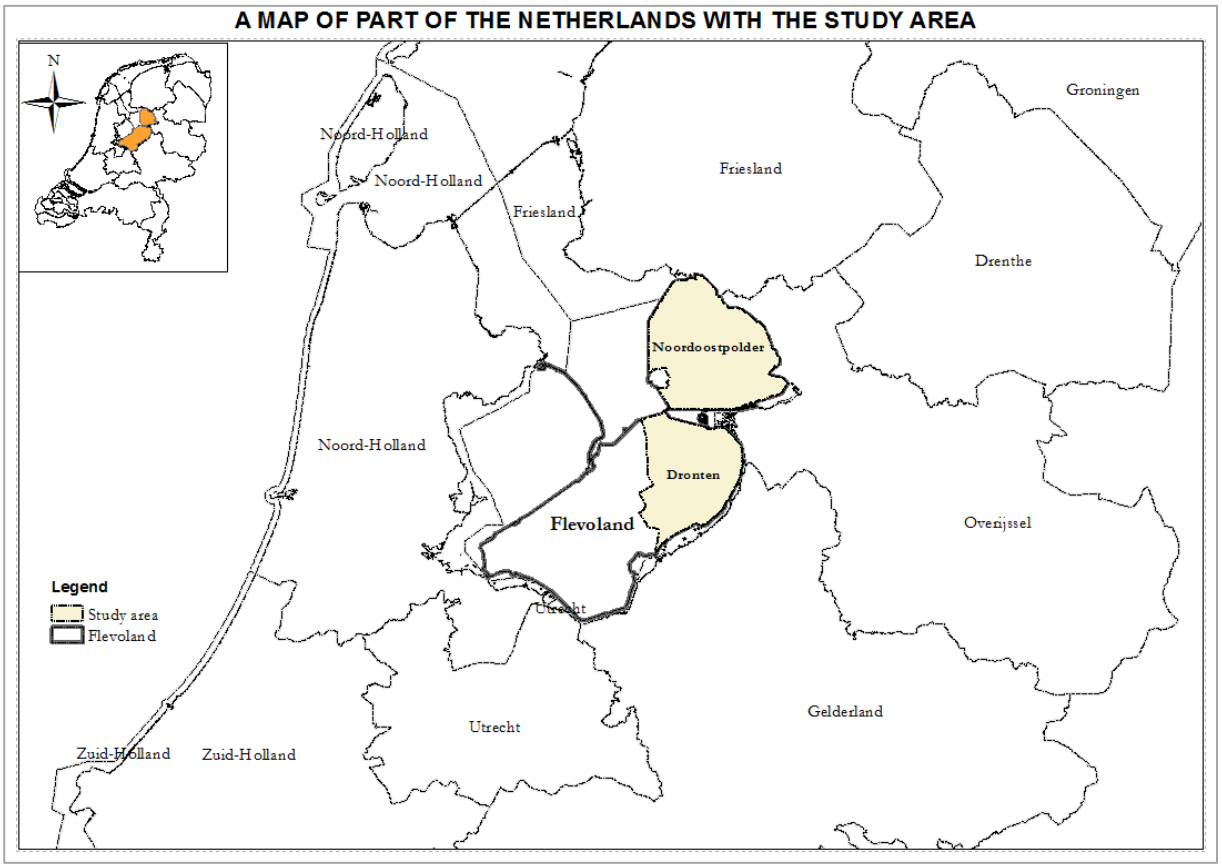


Figure 4.1: A map of the agricultural area for which datasets were acquired.

Table 4.2: Summary of the classes and their respective number of polygons in the reference dataset

Class	Number of Polygons
Potatoes	32
Beet	24
Fruit trees	13
Winter Barley	14
Summer Barley	14
Grassland	21
Alfalfa	12
Maize	18
Winter Wheat	21
Summer Wheat	14
Tulips&lillies	21
Onions	28
Carrots	15
Chicory root	17

4.1.2. Image preparation

The images obtained were pre-processed using the ESA SNAP toolbox for SAR image analysis. First, calibration of the images for each date was performed. Calibration of the radar backscatter is necessary to enable inter-comparison of radar images obtained at different dates by the same sensor or in different modes. Sigma nought backscatter coefficient images were obtained for each date. The calibrated images were geocoded and stacked together in ascending order from May to August to form a multi-temporal

image of the study site. The multitemporal image used for SVM and RF was filtered using a Frost filter. No speckle filtering was applied to the image used in the CNN classification.

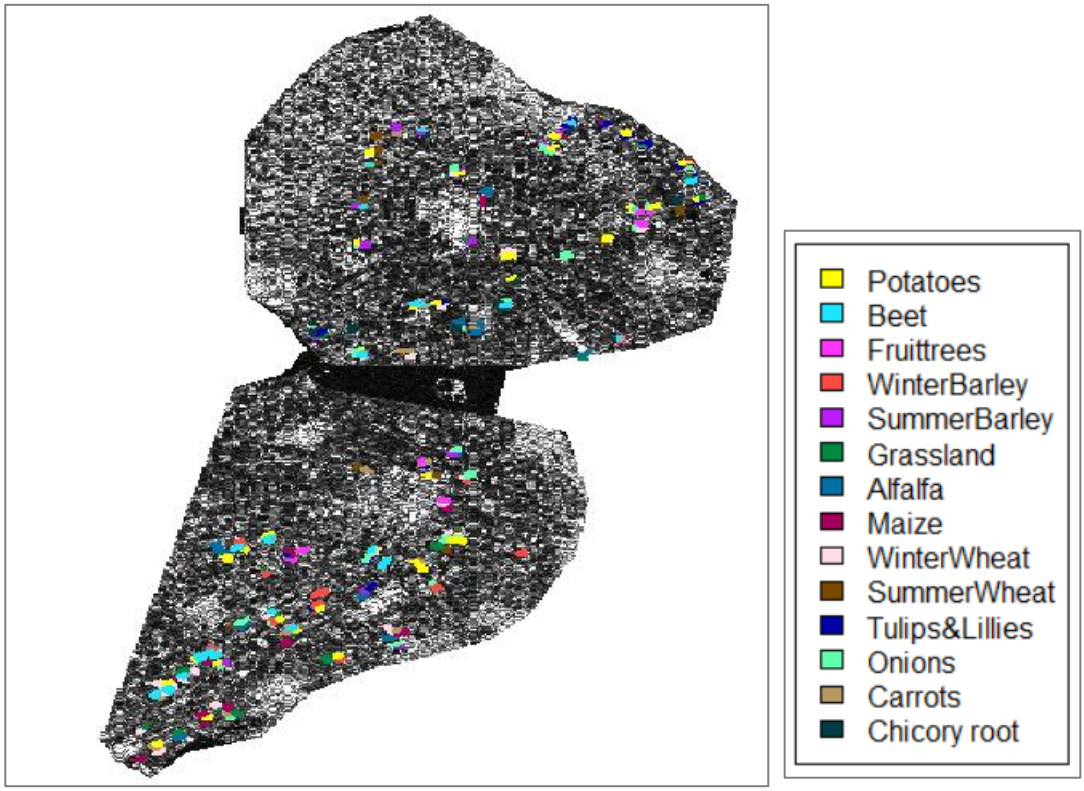


Figure 4.2: A grayscale image of the study area with reference polygons and the legend.

4.2. CNN DESIGN EXPERIMENTS

In this section, we discuss the experiments relating to the design, choice of hyperparameter values and learning and regularization parameter values of the CNN.

Data on hand

A subset of size 1243×1393 was extracted from the unfiltered multitemporal image for further experimentation. Only the polygons falling within the subset were used to extract the training, testing and validation set used for experimental analysis. The grayscale image of the subset with reference polygons is shown in Figure 4.3 and the training, validation and test sets were built on the basis of the sampling strategy described as follows.

Sampling

An equal random sampling strategy is used to build the training, validation and test set. Since there is no ground truth image available, it is not possible to determine the dominant crops in a subset. We draw an equal number of samples per class (500 patches) for the experimental analysis. For the final implementation, all the reference points within the subset are used for extracting training, validation, and testing samples.

Testing

Sparse testing (performing accuracy assessment on only selected pixels in the entire image) is used for all experiments and also for evaluating the classifier in the final implementation since no ground truth image is available in order to carry out full image testing.

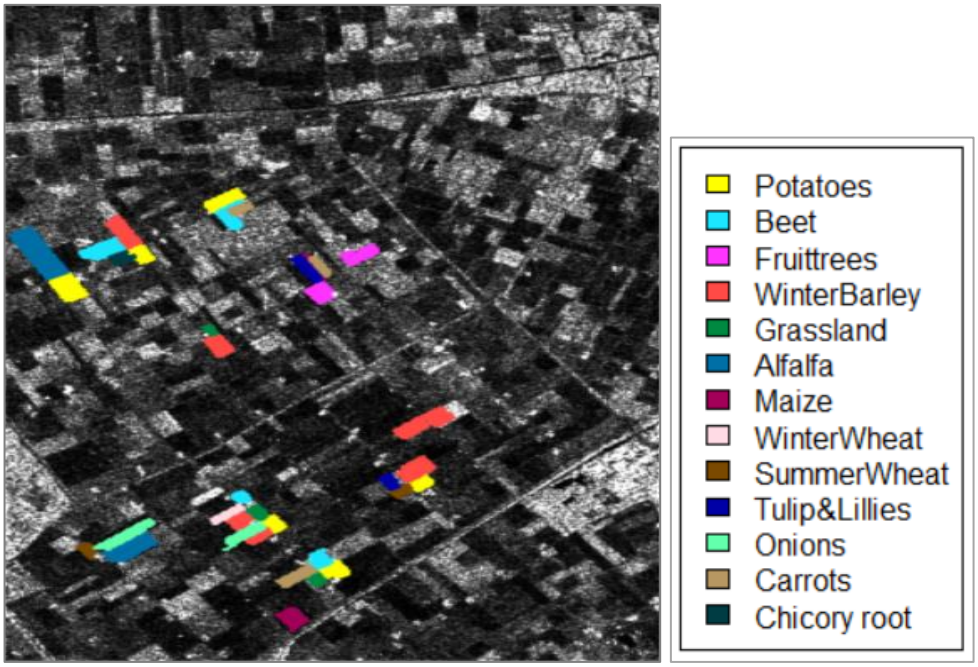


Figure 4.3: A grayscale subset image taken from the study area for experimental analysis and the legend.

Activation function

Three activations functions: the sigmoid, hyperbolic tangent ($tanh$), and rectified linear unit ($Relu$) are commonly used in deep learning. In our implementation, we use the $Relu$ activation function by Glorot et. (2011). $Relu$ has become the popular choice for most deep learning applications with the advantage that it can greatly accelerate the convergence of the stochastic gradient descent compared to sigmoid and $tanh$ functions (Krizhevsky et al., 2012).

Initialization

To initialize the weights of the CNN network for training, we use a normalized (Gaussian) initialization technique proposed by Glorot & Bengio (2010). In this approach, the weights are initialized by drawing them from a Gaussian distribution with zero mean and a specific variance:

$$Var(w) = \frac{2}{n_{in}+n_{out}} \tag{4.1}$$

Where w is the initialization of distribution for the neuron in question and n_{in}, n_{out} are the number of neurons fed into it and the number of neurons the result is fed into respectively.

Regularization

CNN models are prone to overfitting on the training dataset which results in poor performance of the trained network on unseen data. In order to prevent overfitting and improve the generalization capability of the classifier, three forms of regularization are applied as stated in Subsection 3.1, for all experiments as well as for the final implementation: L2 weight decay, dropout, and early stopping criteria. For early stopping, we monitor the value of the loss function over a validation set and save the best value from all of the previous epochs, and prematurely stop the training if the best value of the validation loss function does not change for e_n number of epochs after it has been saved (Bergado, 2016). For dropout, the rate of 'dropping a unit' $(1 - p_r)$ is set. The optimal values of the regularization parameters are determined using a holdout validation method described as follows.

Holdout validation scheme

In a holdout validation scheme, the training set is further split into training and validation sets by a pre-defined proportion. We define a set of possible values for model parameter/s and the algorithm forms all possible combinations of the parameter/s values with other model parameters. The validation set is then used to evaluate the criteria for selecting the optimal value of the hyperparameter/s, which is the overall accuracy. The number of models evaluated for each experiment is equal to the number of possible parameter combinations. We use this strategy for selecting the optimal parameters for the training the CNN model during the experimental phase and for the final implementation

4.2.1. Sensitivity to hyperparameters

We use the architecture described in Subsection 3.1.1 and the overall accuracy as a criterion for the evaluating sensitivity analysis experiments. A holdout validation method described above was used to select the optimal configuration of the hyperparameters of the model. Each sensitivity analysis experiment was designed with training, validation and test sets built using the sampling strategy described under Section 4.2. A small set was used for these experiments to minimize computational time with a CPU implementation which is generally computationally expensive. Three forms of regularization described in Section 4.2 are applied. The hyperparameters investigated in these experiments are the patch size, kernel size, number of filters and network depth. A similar configuration of the CNN is used for the first three hyperparameters as described in the experimental setting. Similar value sets are used for tuning learning and regularization parameters for all experiments such that we are able to determine the hyperparameter that our CNN is most sensitive to, based on a comparison of percentage increase in accuracy.

Patch size

We investigated the effect of the patch size t hyperparameter by varying the sizes of the input patches to a CNN with a fixed configuration shown in Table 4.3. We trained the network with stochastic gradient descent with momentum over 2918 training samples with learning and regularization parameters in table 4.4, tuned over 1437 validation samples in a holdout validation scheme and the overall accuracy is evaluated over 2145 test samples. The following patch size values are used in the experiments: 9, 11, 15, and 33.

Table 4.3: Patch size experiments: CNN configuration

Hyperparameter	Value
Layers	I-C-A-P-D1-C-A-P-D1-F-A-D2-O
Non-linearity used in A[3], O	$Relu, softmax$
Width of F	128
Pooling size	2
Patch size t	(9,11, 15, 33)
Kernel size f_s	5
Number of filters n_f	8

Layer notation: I= Input, C= convolution, A= Activation, D1=Dropout (1st stage), P=pooling, F= MLP hidden layer (fully-connected) D2 = Dropout (2nd stage) O = output.
Relu function is used in A, while the softmax function is used in the output layer O.
The convolution stride is set to 1 and the pooling stride is set to 2.

Table 4.4: Learning and regularization parameters

Parameter	Value
Learning rate η	(0.01, 0.001, 0.0001)
Learning rate decay η_d	(0.01, 0.001, 0.0001)
Momentum α	0.9
Weight decay λ_m	(0.01, 0.001, 0.0001)
Early stopping patience e_n	50
Dropout rate (D1, D2)	((0.0, 0.5), (0.0, 0.0))
Max number of epochs	1000

The learning rate decreases after each epoch defined by the function $\eta(e) = \frac{\eta_0}{1+\eta_d e}$, where η is the learning rate at epoch e, η_0 is the initial learning rate, and η_d is the learning rate decay term.
L2 parameter norm penalty (weight decay) is used to penalize the weights.
A fixed mini-batch size of 128 was used.

Kernel size

We investigated the effect of the kernel size f_s used in the convolutional layers. The configuration used for the network in patch size experiment is preserved, but the value of the patch size is fixed for these experiments as shown in Table 4.5. We trained the network with stochastic gradient descent with momentum as described in patch size experiments. The following values are used for f_s in the experiments: 3, 5, 7, and 9.

Table 4.5: Kernel size experiments: CNN configuration

Hyper parameter	Value
Layers	I-C-A-P-D1-C-A-P-D1-F-A-D2-O
Non-linearity used in A[3], O	$Relu, softmax$
Width of F	128
Pooling size	2
Pooling stride	1
Patch size t	11
Kernel size f_s	(3, 5, 7, 9)
Number of filters n_f	8

Layer notation: I= Input, C= convolution, A= Activation, D1=Dropout (1st stage), P=pooling, F= MLP hidden layer (fully-connected) D2 = Dropout (2nd stage) O = output.
Relu function is used in A, while the softmax function is used in the output layer O.
The convolution stride is set to 1 and the pooling stride is set to 2.

Number of filters

We also investigated the effect of the number of filters n_f used by the convolutional layers of the network. The network with the fixed configuration in Table 4.6 was trained using stochastic gradient descent with momentum as described in patch size experiments. The following values are used for n_f in the experiments: 8, 16, 32, and 64.

Table 4.6: Experiments on the number of filters: CNN configuration.

Hyperparameter	Value
Layers	I-C-A-P-D1-C-A-P-D1-F-A-D2-O
Non-linearity used in A[3], O	$Relu, softmax$
Width of F	128
Pooling size	2
Pooling stride	1
Patch size t	9
Kernel size f_s	5
Number of filters n_f	8, 16, 32

Layer notation: I= Input, C= convolution, A= Activation, D1=Dropout (1st stage), P=pooling, F= MLP hidden layer (fully-connected) D2 = Dropout (2nd stage) O = output.
Relu function is used in A, while softmax function is used in the output layer O.
The convolution stride is set to 1 and the pooling stride is set to 2.

Network depth

Finally, we studied the effect of the number of convolutional layers C_n and number of fully-connected layers F_n of a CNN network with the configuration in Table 4.7, trained using stochastic gradient descent as described in patch size experiments. The following C_n values were used: 2, 3, 4 while fixing F_n to 1; and F_n values 1, 2, 3 while fixing C_n to 3.

Table 4.7: Network depth experiments: CNN configuration.

Hyperparameter	Value
Layers	I-(C-A-P-D1) × C_n D1-(F-A-D2)× F_n - O
Non-linearity used in A[3], O	$Relu, softmax$
Width of F	128
Pooling size	2
Pooling stride	1
Patch size t	9
Kernel size f_s	5
Number of filters n_f	8

Layer notation: I= Input, C= convolution, A= Activation, D1=Dropout (1st stage), P=pooling, F= MLP hidden layer (fully-connected) D2 = Dropout (2nd stage) O = output.
Relu function is used in A, while the softmax function is used in the output layer O. The convolution stride is set to 1.

4.2.2. Pooling with and without subsampling

A pooling layer is usually added after the convolution and activation layers in the CNN network to produce translation invariant features. When subsampling is implemented in the pooling layer, the output of the network has a lower spatial resolution than the input image resolution. The amount of down sampling is controlled by the stride size (number of pixels the window moves along the input in a pooling operation). When the stride is set to 1, no subsampling occurs at the pooling layer. The subsampling factor from input to output is the product of the strides from all layers. For example, if there are N max-pooling layers each with a stride of 2, the overall downsampling factor of the CNN is $S = 2^N$. Hence, the network reduces the resolution of the original image by the factor 2^N. Since the resolution images used in our implementation was not high (10m), the effect of subsampling was investigated by training two CNN configurations with max pooling, which takes the largest element of the feature map within a spatial neighbourhood (window) defined by the pool size as the output. One of the CNNs implemented

subsampling with a stride equivalent to the pool size for each of the CNN layers and the other retaining the pooling layer but without subsampling by setting the pooling stride to 1.
The CNN configuration used for this experiment is shown in Table 4.8. We trained the network with stochastic gradient descent with momentum. The learning and regularization parameters are the same as in Table 4.4.

Table 4.8: Experiment on pooling with and without subsampling: CNN configuration

Hyperparameter	Value
Layers	I-C-A-P-D1-C-A-P-D1-F-A-D2-O
Non-linearity used in A[3], O	$Relu, softmax$
Width of F	128
Pooling size	2
Pooling stride	(1,2)
Patch size t	9
Kernel size f_s	5
Number of filters n_f	8

Layer notation: I= Input, C= convolution, A= Activation, D1=Dropout (1st stage), P=pooling, F= MLP hidden layer (fully-connected) D2 = Dropout (2nd stage) O = output.
Relu function is used in A, while the softmax function is in the output layer O.
The convolution stride is set to 1.

4.3. CNN SPECKLE FILTERING EXPERIMENTS

As discussed in Subsection 1.1.4, CNN networks consist of a bank of adaptive filters which when properly configured may filter the input noisy image. In this section, we introduce the experiments that were carried out on speckle filtering ability our designed CNN algorithm. For the different experiments, CNN networks were trained using stochastic gradient descent with learning and regularization parameters in Table 4.4 (see Section 4.2, Subsection 4.2.1. The training, validation and testing samples used are shown in Table 4.9 and were extracted from all the polygons within the subset image. Max pooling without subsampling was applied in all experiments so as to obtain feature maps at the same size as the original image. Only the CNN part (convolution-activation-pooling) of the network was re-built using the saved weights after training the networks in the previous step and convolved with an input image in a forward pass operation. The resulting feature maps were analyzed. In these experiments, equivalent number of looks (ENL) was used as a performance measure for speckle filtering. This requires calculating the statistics of a homogenous region as explained in Section 5.3 on performance metrics. To select a homogenous area, we overlaid the reference polygons on the original image and calculated the statistics of small subset polygons taken inside the reference polygons for different classes. Three classes with the lowest within-class variance were selected for further analysis as homogeneous areas. The homogeneous subset within a class was extracted in such a way that pixels at the edge, which are more likely to be mixed pixels at the transition between classes, were excluded. The selected classes for these experiments were Alfalfa and Winter Barley. We performed the following experiments on speckle filtering with CNNs.

Table 4.9: The labelled reference samples used as training, validation, and testing set

Class	Training	Validation	Testing
Potatoes	3546	1747	2067
Beet	2304	1135	1693
Fruittrees	1151	567	846
Winter Barley	4421	2177	3249
Grassland	705	347	518
Alfalfa	3479	1713	2557
Maize	896	442	659
Winter wheat	368	181	270
Summer Wheat	542	267	398
Tulips	741	365	544
Onions	1509	743	1109
Carrots	1143	563	840
Chicory root	321	158	235
TOTAL	**21125**	**10405**	**14985**

4.3.1. Patch size

While as the patch size does not directly relate to the filter operation, it does define the maximum area from which contextual information is extracted for a given pixel (central pixel in the patch) using the filters. Therefore, we consider the speckle filtering operation as contextual in nature, with some influence from the surrounding area as defined by the patch size. We investigated the effect of different patch sizes with CNNs trained with the following patch size values: 9 and 15. The configuration of the pre-trained CNNs is as shown in the table 4.10. Only the layers reconstructed for the forward operation are shown in the table.

Table 4.10: Speckle filtering experiments on patch size: CNN configuration

Hyperparameter	Value
Layers	C-A-P-C-A
Non-linearity used in A[2]	$Relu$
Pooling size	2
Pooling stride	1
Patch size t	(9, 15)
Kernel size f_s	5
Number of filters n_f	32

Layer notation: C= convolution, A= Activation
Relu function is used in A.
The convolution stride is set to 1.
Only the intermediate pooling layers are included, the last pooling layer is excluded to produce activated feature maps.

4.3.2. Kernel size

The kernel size is an important parameter in the speckle filtering operation as it defines the dimensions of the window that is centred over each pixel in the image. All the pixels values that fall within the window are used in the mathematical calculation of the new pixel value for the pixel over which the window is centred. Therefore, in addition to patch size experiments above, we investigated the effect of kernel size with the following kernel size values as used in the experiments: 3, 5, and 7. The configuration of the pre-

trained CNNs is as shown in the Table 4.11. Only the layers reconstructed for the forward operation are shown in the table.

Table 4.11: Speckle filtering experiments on the effect of kernel size: CNN configuration

Hyperparameter	Value
Layers	C-A-P-C-A
Non-linearity used in A[2]	$Relu$
Pooling size	2
Pooling stride	1
Patch size t	15
Kernel size f_s	(3, 5, 7)
Number of filters n_f	32

Layer notation: C= convolution, A= Activation
Relu function is used in A.
The convolution stride is set to 1.
Only the intermediate pooling layers are included, the last pooling layer is excluded to produce activated feature maps.

4.3.3. Number of filters

CNNs learn different features from the input using various filters. As a result, not all of the filters may have a noise reduction influence using the weights learnt from the data. The number of filters is equivalent to the number of feature maps that we obtain in the forward operation. This may have an influence in the number of filters performing speckle filtering within the network. Thus, we also investigated the effect of varying the number of filters on the network's ability to reduce noise with the following values for the number of filters used in the experiments: 8, 16, and 32. The configuration of the pre-trained CNNs is as shown in the Table 4.12. Only the layers reconstructed for the forward operation are shown in the table.

Table 4.12: Speckle filtering experiments on the effect of the number of filters: CNN configuration

Hyperparameter	Value
Layers	C-A-P-C-A
Non-linearity used in A[2]	$Relu$
Pooling size	2
Pooling stride	1
Patch size t	15
Kernel size f_s	5
Number of filters n_f	(8, 16, 32)

Layer notation: C= convolution, A= Activation
Relu function is used in A.
The convolution stride is set to 1.
Only the intermediate pooling layers are included, the last pooling layer is excluded to produce activated feature maps.

4.3.4. Number of convolutional layers

As explained in Subsection 2.2.1, the convolution operation in CNNs is hierarchical with the first layers learning lower order features and the level of abstraction increases as more layers are added. In this experiment, evaluate the effect of increasing the levels of abstraction on the speckle filtering process with the following values for the number of convolutional layers C_n used in the experiments: 2, 3, and 4. The configuration of the pre-trained CNNs is as shown in the Table 4.13. Only the layers reconstructed for the forward operation are shown in the table.

Table 4.13: Speckle filtering experiments on the number of convolutional layers: CNN configuration

Hyperparameter	Value
Layers	(C-A-P)× C_n
Non-linearity used in A[2]	$Relu$
Pooling size	2
Pooling stride	1
Patch size t	15
Kernel size f_s	5
Number of filters n_f	32

Layer notation: C= convolution, A= Activation, P=pooling
Relu function is used in A.
The convolution stride is set to 1.
Only the intermediate pooling layers are included, the last pooling layer is excluded to produce activated feature maps.

4.3.5. Average pooling versus max-pooling

Max pooling takes the largest value within a window and average pooling takes the average of all values within the window in the pooling operation. The operation of average pooling may be seen as averaging over the noise in the feature maps such that the next layer receives less noisy feature maps as input, while max pooling may, in fact, advance the noise to the next layer features, when the maximum values correspond to noisy pixels. On the basis of this intuition, we investigated the effect of the pooling strategy on the quality of speckle filtering. The network configuration for these experiments is shown in Table 4.14.

Table 4.14: Speckle filtering experiments on the effect of the pooling strategy used: CNN configuration

Hyperparameter	Value
Layers	C-A-P-C-A
Non-linearity used in A[2]	$Relu$
Pooling size	2
Pooling stride	1
Pooling strategy	(average pooling, max pooling)
Patch size t	9
Kernel size f_s	5
Number of filters n_f	32

Layer notation: C= convolution, A= Activation P=pooling
Relu function is used in A.
The convolution stride is set to 1. The CNN was trained with input patches of size 15×15.
Only the intermediate pooling layers are included, the last pooling layer is excluded to produce activated feature maps.

4.3.6. Summary CNN speckle filtering experiments

In this section, we have presented the various experiments performed to investigate various aspects of CNN as a speckle filter. In summary, we investigated the effect of varying the CNN hyperparameters (patch size, kernel size, the number of filters, and depth) on its ability to reduce noise, as well on the effect of the pooling strategy used in the network. The experiments were conducted on three homogenous classes of interest as described in Section 4.4. We postpone the reporting of results to Chapter 6.

4.4. FINAL IMPLEMENTATION

In section 4.2 and 4.3, we have shown how we investigated the components of our proposed approach, CNN. In summary, we performed hyperparameter sensitivity experiments (varying the patch size, kernel size, number of filters, and network depth) as well as experiments on pooling with and without subsampling, and finally experiments on the effect of the hyperparameters on the speckle filtering ability of the CNN. We postpone the final implementation details of the algorithm to be compared with 2 other classifiers to Chapter 5 and the results from the experiments and the discussion of results to Chapter 6.

4.5. PARAMETER TUNING EXPERIMENTS FOR MRF AND SA ALGORITHMS

In this section, we discuss the experiments that were conducted to determine the optimal values for the MRF smoothing parameter λ and the SA cooling schedule parameters initial temperature T_0 and updating temperature T_{upd}. The datasets used are the class probability maps obtained from a CNN soft classification and the testing set. We set the SA parameters first, followed by the MRF smoothing parameter as explained in the experimental set-up below.

SA parameter tuning

To determine the optimal value of T_{upd}, we classified a subset taken from the class probability maps of the image being classified with different values of T_{upd} at fixed values of λ and T_0 while observing the final energy value. For each value of T_{upd}, the experiment is repeated 100 times and the mean and variance of the resulting final energy values calculated. The values of T_{upd} investigated are shown in Table 4.15.

We also performed experiments to determine the optimal values for T_0 at fixed values of T_{upd} and λ while observing the final energy value on the same subset as for T_{upd} experiments. For each value of T_{upd}, the experiment is repeated 100 times and the mean and variance of the resulting final energy values calculated. The values of T_0 that were investigated are shown in Table 4.16.

Table 4.15: Updating temperature experiments: parameter values

Parameter	Value
Updating temperature T_{upd}	(0.9, 0.95, 0.99, 0.999)
Initial temperature T_0	4
Smoothing parameter λ	0.80

Table 4.16: Initial temperature experiments: parameter values

Parameter	Value
Updating temperature T_{upd}	0.99
Initial temperature T_0	(1, 3, 4, 5, 10)
Smoothing parameter λ	0.80

MRF smoothing parameter

Finally, we performed experiments to determine the value of the smoothing parameter λ while observing the overall accuracy evaluated with the test set at fixed values of T_0 and T_{upd}. Additionally, we inspect the resulting classified map for each value of λ to check for occurrences over smoothing. The following values of λ were used for the initial experiments: 0.2, 0.4, 0.6, 0.8, and 0.9. From the results of the initial experiments, the λ values were plotted against the obtained overall accuracy on the test set and the plot was then used to determine where to add more values for further experimentation based on the value that

gave the highest accuracy. To make a better decision on the choice of the best parameter value, the resulting maps were also visually inspected to check for over smoothing.

Final implementation

We have discussed the experiments to determine the optimal parameters for the final implementation of MRF and SA algorithms above. In total, we performed three experiments, the first two for determining cooling schedule parameters T_0 and T_{upd}, and finally the MRF smoothing parameter λ. We postpone the final implementation details to Chapter 5, and the results of parameter tuning experiments to Chapter 6.

CHAPTER 5

PERFORMANCE COMPARISON

In this chapter, we present the approaches that were compared for both speckle filtering and classification. In the first section, we describe each of the speckle filters used in the speckle filtering comparison, followed by a description of the classifiers we compared in Section 5.2. The final configuration of our designed CNN classifier based on the knowledge obtained in the design and speckle filtering experiments described in chapter four is presented, as well as the alternative approaches discussed in Section 3.3.
The respective performance metrics are discussed in Section 5.3 for the speckle filters and Section 5.4 for the classifiers respectively.

5.1. SPECKLE FILTERS

Speckle filtering involves moving a kernel of a specified window size over each pixel in the image and applying a mathematical calculation using the pixel values under the kernel and replacing the central pixel with the calculated value. The kernel is moved along the image in both row and column direction one pixel at a time until the entire image has been covered. By applying the filter, a smoothing effect is achieved and the visual appearance of the speckle is reduced. The speckle filtering techniques explained below were studied and compared. The performance metrics used for this comparison are discussed in Subsection 5.3.1.

5.1.1. CNN as a speckle filter

As discussed in Subsection 1.1.3, convolutional layers in a convolutional network not only capture spatial-contextual features; they also perform speckle filtering using a bank of filters that learn their weights from the raw SAR image patches during the training phase. CNN speckle filtering process is similar to the one described in Section 5.1, where a CNN filter of a specified kernel size is centred on the pixel to be filtered. Therefore, we chose CNN as our filter of primary interest with a kernel size of 5×5 that was compared with existing filters described below. The convolutional layers used are adopted from the final configuration of the CNN classifier described in Subsection 5.2.1, but without subsampling. Other experiments on speckle filtering with CNN are discussed under Section 4.4.

5.1.2. Refined Lee filter

First, we compare the CNN filter with the Refined Lee Filter. This is an adaptive speckle filter modified by Lopes et al. (1990) and is widely used for SAR image speckle filtering as it tends to preserve radiometric information and image texture by taking into consideration the heterogeneous areas. We apply the filter with a window size of 5×5, similar to the kernel size of the CNN filter for a fair comparison, to the raw SAR image.

5.1.3. Frost filter

We also compare the CNN filter with the Frost filter. The Frost filter is an adaptive filtering algorithm that uses a kernel which adapts itself to features based on local statistics (Frost et al, 1982). We apply the filter with a window size of 5×5, similar to CNN and Refined Lee filters to the raw SAR image.

5.2. CLASSIFIERS

5.2.1. CNN

In Section 2.3 we chose the network architecture CNN that utilizes a fully connected layer at the end to learn the classification rule as the classifier of our primary interest. In our final implementation, we adopt the values of hyperparameters showing relatively good results in the sensitivity analysis and speckle filtering experiments (see Subsections 6.1.2 and Section 6.2) for the CNN. We implemented the classifier of our primary interest with 3 convolutional layers and 1 fully connected layers with patch size 15×15 (input patch). Each convolutional layer applies 32 filters with kernel size 5×5, followed by $Relu$ activation function, and then a 2×2 max pooling operation. The convolutional stride is set to 1, while the pooling stride is to 1 for the first pooling layer and 2 for the other layers. The resulting output volume from a series of convolution, non-linear activation, and pooling operations is flattened into a one-dimensional vector and is connected to the succeeding fully connected layer. This layer consists of 128 units and utilises the same $Relu$ activation function. The fully connected layer is then connected to the output layer with C units where C is equivalent to the number of classes of interest, and a softmax activation function for multi-class labelling is used in this layer. The classifier is trained using stochastic gradient descent with momentum and learning rate, with learning and regularization parameters in Table 5.1 tuned over validation samples in a holdout cross validation scheme. All labelled samples within the subset image are used to extract training, validation and test sets (see Table 4.11). An unfiltered multitemporal SAR image is used, as discussed in Section 4.1.

Table 5.1: CNN: Learning and regularization parameters

Parameter	Value
Learning rate η	(0.01, 0.001, 0.0001)
Learning rate decay η_d	(0.01, 0.001, 0.0001)
Momentum a	0.9
Weight decay λ_m	(0.01, 0.001, 0.0001)
Early stopping patience e_n	50
Dropout rate (D1, D2)	((0.0, 0.5), (0.0, 0.0))
Max number of epochs	1000

The learning rate decreases after each epoch defined by the function $\eta(e) = \frac{\eta_0}{1+\eta_d e}$, where η is the learning rate at epoch e, η_0 is the initial learning rate, and η_d is the learning rate decay term.
L2 parameter norm (weight decay λ_m) is used to penalize the weights.
A fixed mini-batch size of 128 was used.

5.2.2. CNN with MRF post-processing

In order to further refine the results of the classification, we adopted an MRF approach that uses contextual information and class probability output from the CNN as input for label smoothing to further remove the noise that remains in the classified image. Thus, we compared the results of the CNN classifier described in Subsection 5.2.1 with a CNN classifier that uses MRF for post classification label smoothing. For a fair comparison, the architectural details of the CNN and hyperparameter values described in subsection 5.2.1 were preserved. However, in contrast to the CNN above where the output is the final class labels, the softmax activation function produces class probability maps which are fed as inputs to the MRF for final multi-class labelling. We obtained the optimal parameter values for the MRF and SA algorithms from the MRF parameter tuning experiments in Section 4.5 as $\lambda = 0.8$, $T_0 = 4$ and $T_{\mathrm{upd}} = 0.999$. The results of parameter tuning are discussed in Section 6.4. The MRF part of this framework produces the final classification map of the scene as output.

5.2.3. Support Vector Machines (SVM)

As discussed in Subsection 1.1.2, several classification algorithms have in the past been used to classify SAR images. We chose pixel-based support vector machines, a state of the art machine learning classifier that has achieved very good results in image classification, both computer vision and in remote sensing as one of the compared approaches. We set-up the C-SVM classifier for multi-class labelling described in Subsection 3.3.1, based on the implementation by Pedregosa et al. (2012) in a python environment. We select the values for the regularization parameter C and the kernel parameter γ using a holdout cross validation scheme, selecting 10 possible values for each on a logarithmic scale and evaluating 100 models with different parameter combinations. The criteria for evaluating and choosing the best model was overall accuracy on the validation set. Similar to CNN, the training, validation and testing samples in Table 4.11 were used in the implementation. The images used for classification with SVM were prepared differently from those used in CNN implementation. The SVM classifier receives an image with 10 bands of the filtered multi-temporal image for pixel-based SVM and additional bands of handcrafted texture features calculated from different statistics of the gray level co-occurrence matrix (GLCM) (Haralick et al., 1973) for the SVM+GLCM. The procedure for GLCM feature extraction and selection is described in Subsection 5.2.4.

5.2.4. Random Forests (RF)

In addition to SVM, we also chose the state-of-the-art Random forests (RF) classification algorithm for comparison with our proposed algorithm CNN. A detailed explanation of the RF classifier is explained in Subsection 3.4.2. We set up a random forest classifier in a python environment based on the implementation by Pedregosa et al. (2012) for ensemble classifiers. The main parameters to adjust were the number of estimators N and the maximum number of features M to consider when splitting a node. For the former, the larger the number of trees, the better the classification results, but also the longer it will take to compute. Additionally, results stop getting significantly better beyond a certain number of trees. Therefore, we explore different threshold values in a holdout cross validation scheme to set the optimal value while monitoring the overall accuracy on the validation set and the total model training time. For the latter, we set the value as recommended by Pedregosa et al. (2012) for classification, where M is equivalent to the square root of the number of bands in the input image.

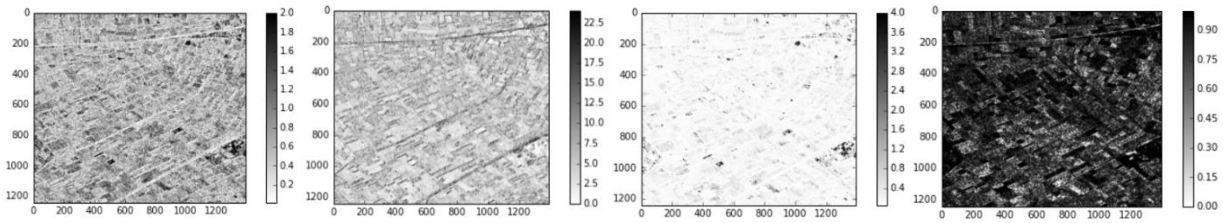
Similar to SVM, a filtered multitemporal image was used for the pixel-based RF and in additional GLCM texture features extracted for the RF+GLCM classifier. We explored different window sizes: 7×7, 9×9, 11×11 and 15×15 while monitoring the overall accuracy of classification. Window size 11×11 has the best performance (see Table 5.2) and was therefore chosen for subsequent feature extraction. GLCM matrices were computed in four offset directions: offsets: $(1,1)$, $(1,0)$, $(0,1)$, and $(-1,1)$. The average of the statistic (e.g. the average of the mean in the four directions) was used as the texture feature. Some of the extracted GLCM features are shown in Figure 5.1.

Table 5.2: Results of classification with different window sizes

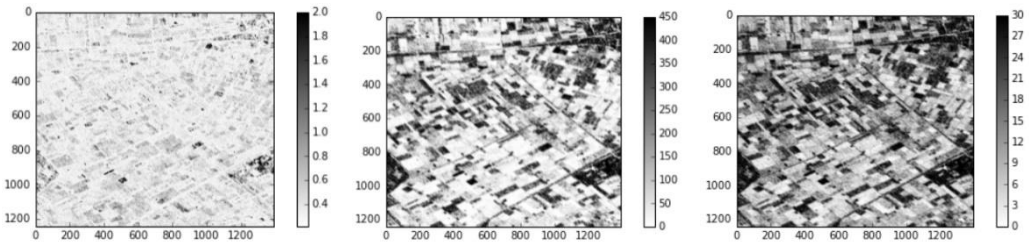
Window size	Overall accuracy (%)
7×7	93.18
9×9	93.46
11×11	94.37
15×15	93.83

Feature selection was performed using a random forest strategy of evaluating the importance of features in the input image in classification. This RF attribute ranks features based on the node of the tree on which they are selected for classification. For instance, features used at the top of the tree contribute to the final prediction decision of a larger fraction of the input samples. The expected fraction of the samples they contribute to can thus be used as an estimate of the relative importance of the features, which ranges

between from 0 to 1. We selected the best 30 features from the input image for the subsequent classification with SVM and RF algorithms.



(a) Contrast (b) Dissimilarity (c) Homogeneity (d) Entropy



(e) Angular second moment (f) mean (g) Entropy

Figure 5.1: Representative features of the extracted GLCM statistics.

5.3. SPECKLE SUPPRESSION METRICS

5.3.1. Equivalent Number of looks

As we have heavily used in the CNN speckle filtering experiments in Section 4.4, we also use the equivalent number of looks (ENL) as a performance metric in the comparison of the filters described in Section 5.1. The Equivalent Number of Looks (ENL) is measured over the homogenous region within a filtered image. It is defined as the ratio between the square of the mean to the variance. It can be used to characterize the smoothing effects of image processing operations such as speckle filtering. It is calculated as follows:

A homogenous region in an image is selected and its mean and standard deviation are calculated. ENL is given by:

$$\text{ENL} = (\mu/\sigma)^2$$

Where, μ is the mean calculated across all bands in the image and σ is the standard deviation. A high ENL value implies better performance of the filter in reducing speckle.

5.3.2. Edge preservation

For classification systems, strong edges are important to preserve the shape of the features within the scene and to adequately discriminate the classes of interest. Therefore, as a requirement, a good speckle filter must preserve edges in the image. In a SAR image, it is difficult to determine where the exact edge is due to variation caused by the presence of speckle. Therefore, we used the class reference polygons to determine the edge between two neighboring classes.

We measured the ability of each of the filters to preserve edges as follows. We overlaid the reference samples for all classes of the filtered images and the CNN feature maps. We then selected two neighbouring polygons for different with strong edges. The classes for which the polygons are selected were identified based the classification results with CNN and SVM, from which the least confused classes

with regular shapes in the classification map are considered to have strong edges. Additionally, classes with different crop structures are not easily confused, such as cereals and non-cereals. A small region of at least 30 pixels is drawn along the edges in the diagonal direction based on the orientation of most reference polygons as shown in Figure 4.3 is drawn at the boundary between the two classes for each of the classes (see Figure 5.2) and its statistics are computed. The absolute gradient of the two classes is computed from the mean as follows.

$$\text{Gradient} = \frac{|\mu_A - \mu_B|}{D/2} \tag{5.1}$$

Where, μ_A, μ_B are the mean of class A and the mean of class B which share a border, and $D/2$ is the number of pixels between the means.
A high gradient value indicates that the filter has a high ability to preserve the edge between two classes.

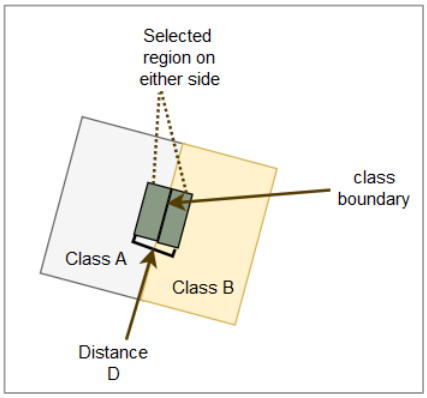


Figure 5.2: Two classes and the boundary along which the regions are drawn for analysis.

5.4. PERFORMANCE METRICS FOR CLASSIFIER COMPARISON

5.4.1. Overall accuracy

As already heavily used in the design experiments, we also used overall accuracy to compare the classification approaches described in Section 5.1. For this, the classifiers were evaluated with a test set and the resulting overall accuracies for each of the classifiers recorded and compared. The overall accuracy is computed by dividing the sum of total correctly classified samples for all classes of interest by the total number of pixels in the test set.

5.4.2. Producer and User accuracies

From the confusion matrix obtained by evaluating each of the classifiers above on the test set, we computed the producer accuracy and the user accuracy. Producer accuracy indicates the probability of a reference pixel being correctly classified. This corresponds to the sensitivity of the classifier to a particular class in the image. It is computed as follows.

$$\text{Producer accuracy} = \frac{correctly\ classified\ pixels\ (classA)}{total\ number\ of\ pixels\ of\ class\ A}$$

User accuracy is a measure of reliability. It indicates the probability that a pixel being classified in the image actually represents that category on the ground. It is computed as follows.

$$\text{User accuracy} = \frac{correctly\ classified\ pixels\ (classA)}{total\ number\ of\ pixels\ classifed\ as\ class\ A}$$

The confusion matrix expresses the number of samples assigned to a particular class relative to the actual class as verified on the ground using a reference dataset. It can be analyzed for classes that are being

confused with each other based on the distribution of their test samples. However, due to the large number of classes in the dataset, confusion matrix for each of the classifiers was not presented in Chapter 6.

5.4.3. Visual inspection and comparison of classification maps

We also compared the classification maps of the different classifiers. Given that speckle cannot be fully removed, some residual noise remains in the classified image. With this performance metric, we measure the classifier's robustness to noise based on the smoothness of the resulting classification maps. Additionally, the classification maps are inspected for classes that are getting confused with each other in the classification.

5.4.4. Computational time and complexity

Another metric we discussed is the computational time of the classification approaches. In this metric, we compared the time it takes for each of the classifiers above to train on the same number of training samples and subsequently classify an image subset. Bearing in mind that different libraries and programming languages were used to implement the approaches, it is difficult to quantitatively compare the performance of the algorithms in this aspect as it requires taking into account the differences between the technologies. Therefore, we only briefly discuss the qualitative comparison of the three approaches, for instance on the basis of the parameters that need to be estimated, among others.

5.5. PERFORMANCE COMPARISON RESULTS

In this chapter, we have shown how we compared the designed CNN as a filter to existing speckle filters as well as the CNN classifier (with automatic feature extraction) with other classification approaches that require filtered images as well as additional information in the form of GLCM texture features. We further introduced the various performance measures that have been used in the comparative study for both speckle suppression and classification. We postpone the results of these performance comparisons to Chapter 6.

CHAPTER 6

RESULTS AND DISCUSSIONS

In this chapter, we report the findings of the experiments and the performance comparison as described in Chapter 4 and Chapter 5 respectively. In the first section, we report on the design experiments of the CNN algorithm. In the next section, we discuss the results of the experiments performed on speckle filtering with the CNN algorithm. The succeeding sections then present the analysis done in evaluating the three speckle filters (CNN, Refined Lee and Frost filters) followed by the evaluation of our 3 classification approaches with several performance metrics discussed in Sections 5.3 and 5.4 respectively. All classified maps presented in this chapter were post processed in ArcMap to remove the non-crop areas using the crop parcel boundary shapefile obtained from the Dutch National spatial data infrastructure (PDOK) portal.

6.1. DESIGN EXPERIMENTS

6.1.1. CNN sensitivity analysis

In this Subsection, we present the results of the sensitivity analysis experiments performed with the CNN network configurations described in Subsection 4.2.1. The overall classification accuracy of the network was recorded as each of the four hyperparameters was varied.

Patch size

The patch size defines the area from which contextual information is extracted for a given central pixel. Table 6.1 shows how the classification performance of the CNN under different sizes of the input patches fed to the network. As we increase the patch size from 9×9 pixels, we can generally observe a trend of increasing overall accuracy from 88.20% peaking at 96.72% for patch size 33×33.

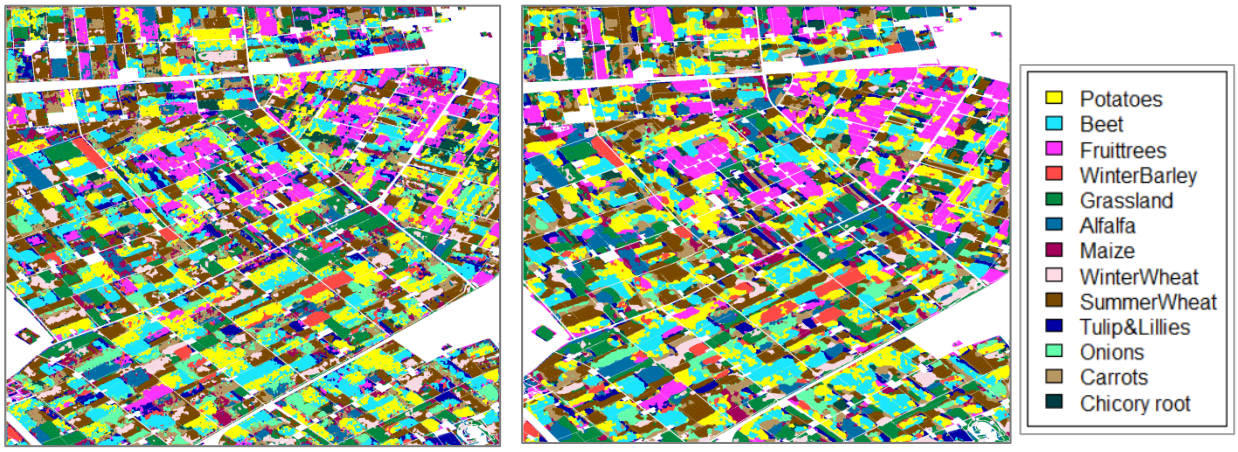
Table 6.1: Results of varying the patch size of the CNN.

Patch size	9×9	11×11	15×15	33×33
Overall accuracy (%)	88.20	90.66	93.92	96.72

Using overall accuracy, patch size 33×33 should have been the obvious choice for our implementation. However, on further evaluation of the classification maps, patch size 33×33 showed over-smoothing compared to 15×15, where the shapes of the crop fields in the scene were not preserved in the classification (see Figure 6.1).

The over smoothing with a larger patch size may be attributed to the resolution of the dataset used. At higher resolution, the objects of interest are considerably larger than the pixel size. This is not the case as the image resolution decreases. A related study on the effect of resolution on the window size for extracting texture information carried out by Chen, Stow, & Gong (2004) found that a small window size performed better only at lower resolutions with a large window size yielding different spatial distributions in the classified land cover map. While as this study was based on handcrafted texture extraction, we can

relate it to our study since the patch size is equivalent to the window size in texture extraction. Therefore, at a resolution of 10m, a smaller patch size (15×15) was considered for further analysis.



(a) Classified map for Patch size 15 (b) Classified map for patch 33 size (c) Legend

Figure 6.1: A comparison of classified maps at patch size 15 and 33

As seen in the maps above, the classification map for patch 33 CNN tends to over smooth along the edges of the crop fields. A patch size of 15×15 was then used in the final implementation of the CNN classifier.

Kernel size

The kernel size defines the size of the local receptive field from which the classifier can learn in order to discriminate the land cover classes of interest. Table 6.2 shows how the classification performance changes with different kernel sizes used by the convolutional layers of the network. Generally, we can observe an increase in overall accuracy as the kernel size increases starting with 87.92% using a 3×3 kernel, peaking at 91.95% using a 7×7 kernel and then drops to 90.79% for kernel size 9×9.

Table 6.2: Results of varying the kernel size of the CNN.

Kernel size	3×3	5×5	7×7	9×9
Overall accuracy (%)	87.92	90.66	91.95	90.79

The kernel size plays an important role in finding the key features in the input image. Theoretically, increasing the kernel size should result in an increase in accuracy as smaller patterns captured by small kernel sizes should be captured by large kernel sizes given enough training data. However, as the kernel size increases, so does the number of parameters of the CNN, making the network hard to train and increasing the computational cost. This could be the reason for the decline in accuracy for kernel size 9×9. In Table 6.2, the rate of increase in accuracy declines as the kernel size is varied before the accuracy finally declines. We also consider the results of the effect of varying the kernel size on speckle filtering (see Subsection 6.2.2) in choosing the kernel size for the final implementation.

Number of filters

The number of filters set the maximum number of contextual features (feature maps) the classifier can learn to look for in order to discriminate the classes of interest. Table 6.3 shows how the classification performance changed as we increased the number of filters. Generally, we can observe an increase in overall accuracy as the kernel size increases starting with 85.56% for 8 filters, peaking at 92.36 for 32 filters and then drops to 91.46 for 64 filters.

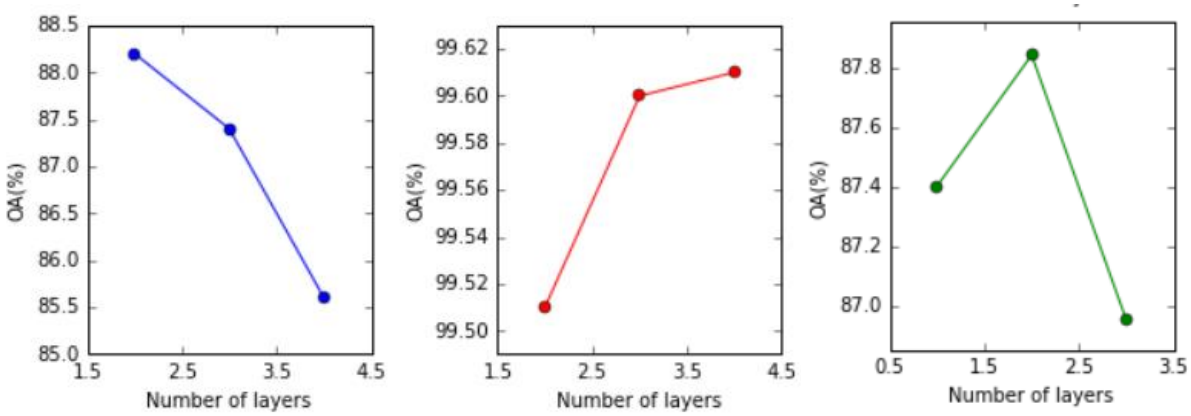
Table 6.3: Results of varying the number of filters in the CNN.

Number of filters	8	16	32	64
Overall accuracy (%)	85.67	89.87	92.36	91.46

Similar to the kernel size, a larger number of filters can theoretically learn all the contextual features that can be learned by a small number of filters. The increase in accuracy with the number of filters is due to the fact that more features are learnt by the CNN, which are then used to discriminate classes of interest. However, increasing the number of filters also increases the number of parameters in all layers of the CNN, resulting in a harder optimization problem especially with a fixed training set. This may be the reason for the drop in accuracy when 64 filters are used. For the CNN classifier, we used 32 filters, taking into account the results of the effect of the number of filters on speckle filtering on the explained in the next section.

Network depth

The depth of the CNN is divided into two parts: CNN depth (number of convolutional operations in the network), and MLP depth (number of fully connected layers in the network). Together, they define the number of operations on the dataset from the input layer to the output layer. Figure 5.2 shows the effect of varying the number of convolutional layers (plot (a) – 2918 training samples, plot (b) - 21125 training samples) and fully connected layers (plot (c)) of a CNN network on its classification accuracy.



(a) (b) (c)

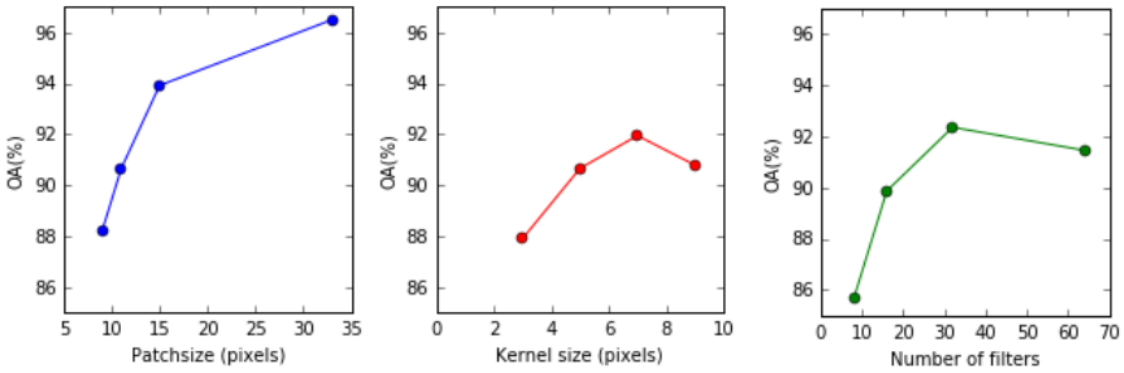
Figure 6.2: The effect of varying the number of layers: convolutional (a and b), MLP (a) in the CNN.

For convolutional layers, the number of parameters that need to be estimated increase as the number of layers increase. Therefore, deeper networks typically have a larger number of parameters which makes them more prone to overfitting, especially if the number of labelled samples in the training set is limited (Szegedy et al., 2015). Assuming the training samples used were sufficient, according to the findings of Bergado (2016), we can attribute the reduction in overall accuracy in (a) to the fact that the complexity of the classification problem at hand does not require the network to learn higher order features. To rule out this assumption, the experiment was repeated with larger training samples. As seen in plot (b) in Figure 6.2, the accuracy slightly increases as more layers are added. Also, the rate of increase is noticeably small from 3 to 4 layers as indicated by the gradient between the two points on the plot. On the basis of that, we can deduce that further increasing the number of layers may instead degrade the overall accuracy. On the other hand, for the MLP layers, there was a slight decrease in accuracy in accuracy from 1 MLP to 2 MLP layers of 0.31% when more training and testing samples were used, which deviates from plot (c) above where accuracy increases by about 0.4% from 1 MLP layer to 2. This implies that the size of the test set may have been inadequate to accurately measure the classifier performance for the results presented in

plot (c). From this, we can conclude that one MLP layer is enough to handle the complexity of our classification problem. Hence, for our final implementation, we use one fully connected layer. The decision on the number of convolution layers was made considering both sensitivity analysis results and the results from speckle filtering experiments presented in Section 6.2.

Comparison of the patch size, kernel size, and number of filters

An additional analysis was performed on the results of sensitivity analysis to CNN hyperparameters. Network depth was excluded from this analysis since a different size of labelled training samples was used the experiments on sensitivity to network depth. For the other hyperparameters, similar learning and regularization parameters, as well as the training and testing sets were used, paving way for further comparison to determine which hyperparameter the CNN is most sensitive to. The results of the various hyperparameter experiments are plotted in shown in Figure 6.3.



(a) (b) (c)

Figure 6.3: The effect of varying the patch size (a), kernel size (b) and number of filters (c) in the CNN.

From the plots above, we notice a similar trend for kernel size and number of filters, with the overall accuracy increasing, peaking at the third value and then declining further on. Concentrating on the gradient between the first three values for all the hyperparameters, we can say that the network is more sensitive to the number of filters (85.67% to 92.36%) followed by the patch size (88.2% to 93.92%). The number of filters is equivalent to the number of feature maps learnt by the CNN and the patch size defines the size of the area from which contextual information is learned for a given pixel. Thus, the network's performance in classification depends on the learned spatial-contextual features.

6.1.2. Results of experiments on pooling with and without subsampling

Pooling in CNN summarizes the results of the convolution operation so as to produce features that are invariant to translations of the input. Generally, we observed no change at all in classification accuracy. The subsampling operation was found to be beneficial to the network in terms of computation time. Since it reduces the size of the feature maps that are fed into the next layer, it substantially reduces the parameters of the proceeding layers, hence reducing the complexity of the network. The reduction in resolution had no effect on the classification as well. However, when the pre-trained network is used as a feature extractor, we found that a no-subsampling approach was better as it produced features with the same size as the input image, which was more useful for further analysis.

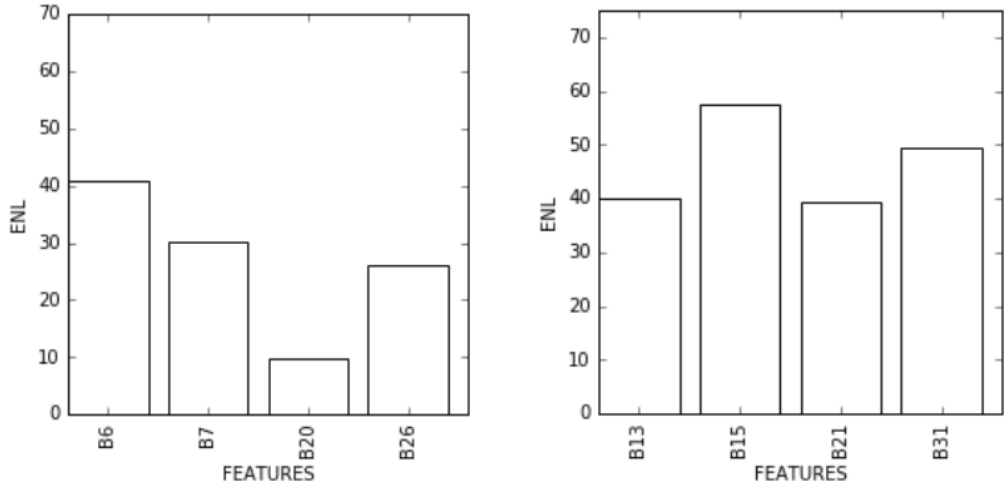
6.2. CNN SPECKLE FILTERING EXPERIMENTS

In the CNN feature extraction, each feature is the result of convolving a filter with the input image. The convolutional layers learn different features from the data, such as edges, shapes, and parts of objects, among others, with increasing levels of abstraction. Thus, not all the filters in convolutional layers

perform speckle filtering and filters performing speckle filtering may also be different for each of the networks, as determined during the learning process. Therefore, we calculated ENL for only the bands where the selected homogenous region appears, which differ for each experiment depending on the filters performing speckle filtering. Below, we present the ENL results for the best four speckle filters for each of the experimental set-ups except for the experiments on the number of filters, where ENL for all the bands for the speckle filters is presented.

6.2.1. Patch size

Figure 6.4 shows the calculated ENL for Winter Barley at patch sizes 9 – plot (a) and 15 – plot (b). From the plots, we can generally observe high ENL values for patch size 15, with an average ENL value of 46.95 compared to patch size 9 with an average ENL of 26.65. Overall, ENL increases with increase in patch size.



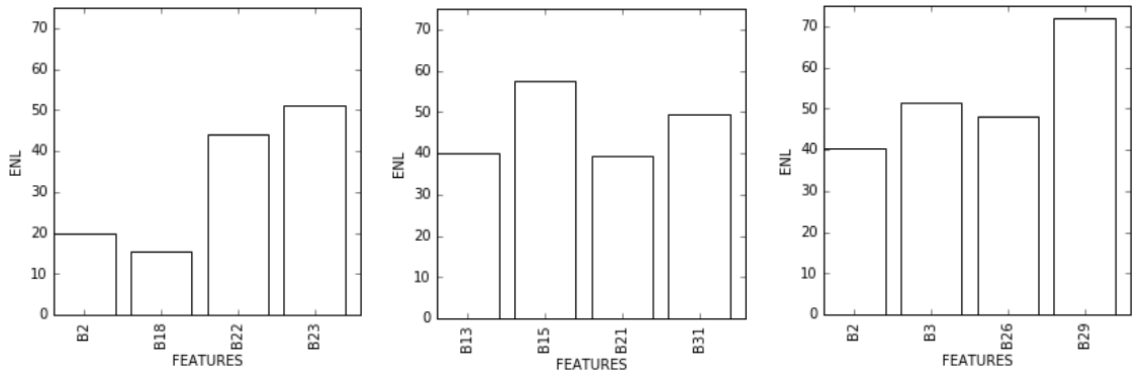
(a) (b)

Figure 6.4: ENL results with different patch sizes (9 – (a), 15 – (b)) used in the CNN.

The patch determines the size of the area surrounding a pixel, from which contextual information is extracted, and therefore, the size of the area over which the filter weights are learnt. These weights are then subsequently used in the computation of the new filtered value for the central pixel. We can interpret the results above as indicative of the influence of the size of the area over which the filter learns its weights. Thus, filter weights learnt over a wider area (patch size 15) result in better speckle filters and a better smoothing effect. Further increasing the patch size may result in better ENL values, but also in over smoothing as discovered in the design experiments on patch size. In our implementation, patch size of 15 was used.

6.2.2. Kernel size

The kernel size defines the dimensions of the window that is centred over each pixel in the image, where a mathematical calculation is performed using the pixel values under the kernel the central pixel value is replaced with the calculated value, similar to a typical operation with any averaging filter on the image. Figure 6.5 shows the calculated ENL for window sizes 3 – plot (a), 5 – plot (b), and 7 – plot (c). The average ENL value for kernel size 3 is 32.56, kernel size 5 46.59 and kernel size 7 52.95. Overall, the higher the kernel size, the better the ENL values.



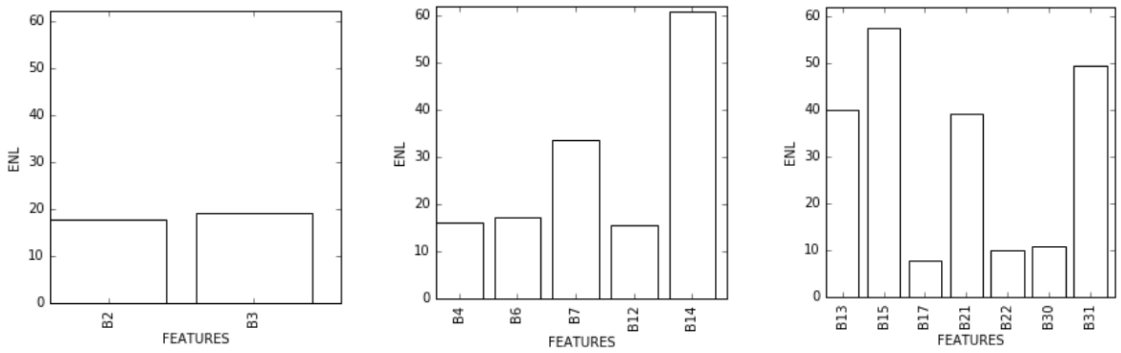
(a) (b) (c)

Figure 6.5: ENL results for different kernel sizes (3 – (a), 5 – (b), 7 – (c)) used in the CNN.

Theoretically, increasing the kernel size should result in a greater smoothing effect as more pixels are then utilized in the calculation of the central pixel value, therefore averaging over a larger number of pixels. This is in line with our findings as shown above. However, as discussed in the sensitivity analysis experiments on kernel size, a larger kernel size is associated with a large number of parameters to be estimated resulting in a difficult optimization problem. Therefore, we chose a kernel size of 5 which strikes a balance between preserving feature details, computational time and adequately reducing speckle.

6.2.3. Number of filters

Some of the filters in a CNN perform speckle filtering simultaneously while extracting the relevant features from the data. In this analysis, we present the ENL of all the features on which speckle filtering was performed for the Winter Barley. Figure 6.5 shows the ENL calculated as we varied the number of filters: plot (a) – 8 filters, plot (b) – 16 filters and plot (c) – 32 filters. Generally, we can observe from the plots that the number of speckle filters increases as we increase the number of filters in the network. The average ENL for 8 filters is 18.29, 16 filters 28.74, and 32 filters 30.76.



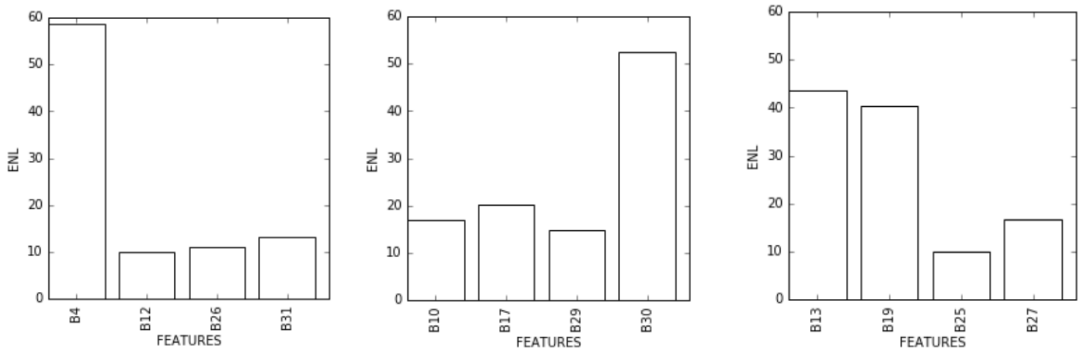
(a) (b) (c)

Figure 6.6: ENL results for different number of filters (8 – (a), 16 – (b), 32 – (c)) used in the CNN.

Since all speckle filters in the network contribute to the overall performance of a CNN in speckle filtering, we can say that increasing the number of filters, in this case, improves the network's ability to reduce noise with the various filters that are applied simultaneously on the dataset. Hence, in the final implementation, we use 32 filters in the network to maximize both speckle filtering and the accuracy of classification.

6.2.4. Number of convolutional layers

In addition to the previous experiments performed, we also calculated ENL for the class of interest while varying the number of convolutional layers in the network. Figure 6.7 shows the ENL results for different numbers of convolutional layers in the network. The average ENL for CNN with 2 convolutional layers is 23.22, 3 convolutional layers 26.07 and 4 convolutional layers 27.60.



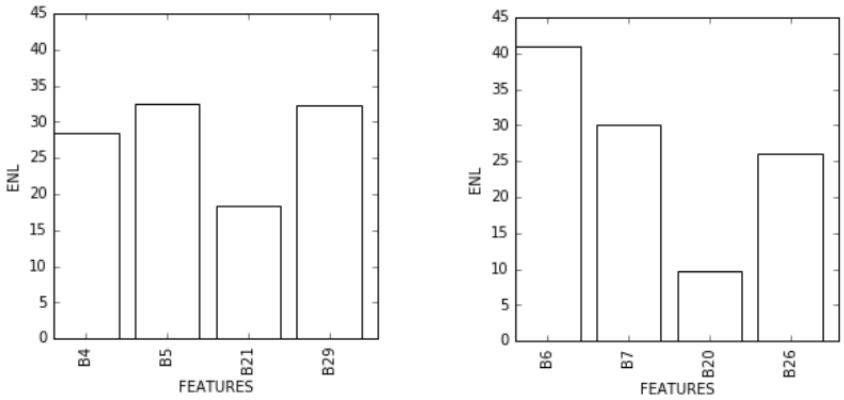
(a) (b) (c)

Figure 6.7: ENL results for different number of convolutional layers: (2 – (a), 3 – (b), 4 – (c)) in a CNN.

CNN feature extraction, and thus speckle filtering is hierarchical with different levels of abstraction. For instance, speckle filtering in the third convolutional layer is performed on features extracted and filtered by the second convolutional layer which are less noisy, and so on. Therefore, the ENL increases as more layers are added to the network as shown in Figure 6.7. A disadvantage of increasing the convolutional layers for speckle filtering is that a difficult optimization problem arises during network training as the parameters exponentially increase as explained in the results analysis for the network depth experiments. In order to balance between adequate speckle filtering and computational time resulting from making the network deeper, we chose 3 a CNN with three convolutional layers for our implementation.

6.2.5. Max pooling and average pooling

Figure 6.8 shows the ENL results computed for CNN with different pooling strategies. The average ENL value for average pooling is 27.93 and for max pooling 26.65. This confirms our suspicion that taking the average of all values within the pooling window is more beneficial for speckle filtering as it averages over speckle in the process.



(a) (b)

Figure 6.8: ENL results for different pooling strategies (average – (a), max – (b)) used in CNN.

However, convolutional neural networks implementing max pooling have been found to have small error rates compared to average pooling for most classification tasks according to the findings of Zeiler & Fergus (2013). The overall accuracies of the networks on which the ENL was measured confirm this with max pooling having an overall accuracy of 98.67% while average pooling 98.15%. Additionally, the network with average pooling was found to be computationally very expensive, taking a lot more time to train, hence max pooling was used in the final implementation.

6.3. PERFORMANCE ANALYSIS FOR THE SPECKLE FILTERS

6.3.1. Equivalent number of looks (ENL)

Figure 6.9 shows the ENL of each band calculated for Winter Barley, after applying the Refined Lee filter (a), Frost filter (b) and a CNN filter (c) with three convolutional layers on the SAR image. We can interpret the results for Frost and Refined Lee filter as the ENL calculated after applying the filter with a window of size 5×5 to the image with bands 1 to 10, and for the CNN filter as applying different filters with a kernel size 5×5 (in this case, 7 filters for the seven feature maps) to the same image. Therefore, the effect of the combination of different filters used in speckle filtering with CNN is far greater than for applying a single Frost filter with the same kernel size to the image. Overall, we can say that CNN with an average ENL value of 30.76 across filters outperforms Frost and Refined Lee filters with average ENL values of 17.87 and 9.89 respectively.

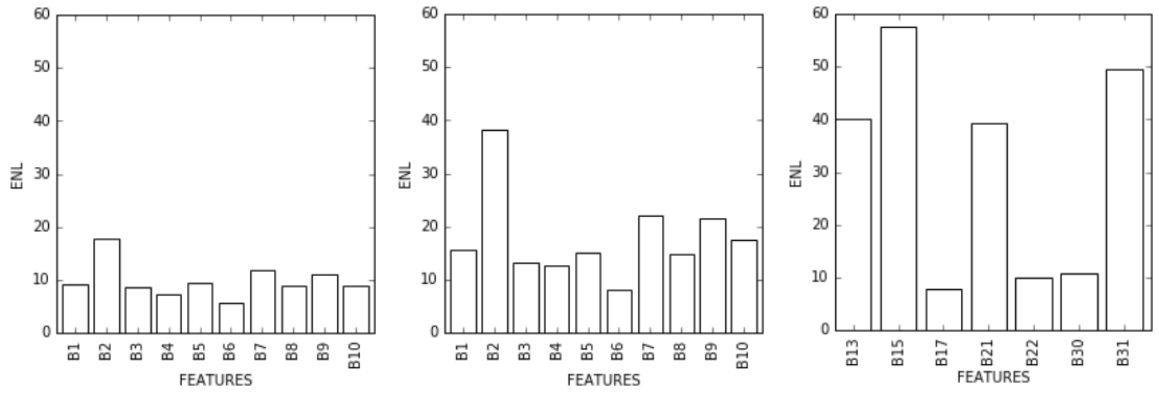


Figure 6.9: ENL results for speckle filtering with Refined Lee (a), Frost (b) and CNN (c) filters.

It is possible to apply the any of the traditional filters such as Refined Lee and Frost filters several times on the image. However, in a study by Collins (2000) on the effect of speckle filtering on spatial scale-dependent texture, it was found that speckle filtering does not retain most of the texture information as the resolution decreases. Since both Refined lee filter and Frost filter are spatial averaging filters, with each filtering operation, they degrade the resolution. Therefore, CNN as a speckle filter performs better as indicated by its calculated ENL for the various speckle filters in the network.

6.3.2. Edge preservation

The results of edge preservation assessment described in Subsection 5.3.2 between two neighboring classes for a subset taken near the shared edge are shown in Table 6.4. We selected Winter Barley (cereal) and potatoes (non-cereal) for this assessment as they are least likely to get confused since they have different crop structures, hence a strong gradient must exist between them.

Overall, we can say that CNN has a stronger gradient between neighboring classes with an average value of 0.4156 compared to frost filter with an average gradient of 0.0048 and Refined Lee filter with an

average gradient of 0.0037. The higher the gradient values, the better the edge preservation for a given filter.

Table 6.4: Edge preservation (gradient) results for Frost, Refined Lee and CNN filters

	Frost filter			Refined Lee filter				CNN filters		
Band	WB	PT	Gradient	WB	PT	Gradient	Band	WB	PT	Gradient
B1	0.0229	0.0188	0.0014	0.0233	0.0190	0.0014	C1	0.0078	0.1875	0.0599
B2	0.0470	0.0653	0.0061	0.0466	0.0638	0.0058	C4	0.0234	0.3438	0.1068
B3	**0.0389**	**0.0203**	**0.0062**	**0.0400**	**0.0198**	**0.0067**	**C7**	**9.8203**	**3.4531**	**2.1224**
B4	0.0585	0.0709	0.0042	0.0586	0.0690	0.0034	C8	4.5703	3.8672	0.2344
B5	0.0171	0.0186	0.0005	0.0170	0.0184	0.0005	**C16**	**4.3281**	**1.8672**	**0.8203**
B6	0.0905	0.0901	0.0001	0.0893	0.0904	0.0004	C17	0.3203	0.2578	0.0208
B7	0.0166	0.0255	0.0030	0.0159	0.0258	0.0033	C22	0.6875	0.0625	0.2083
B8	**0.0599**	**0.0918**	**0.0106**	**0.0575**	**0.0926**	**0.0117**	**C23**	**0.1016**	**1.5391**	**0.4792**
B9	0.0220	0.0286	0.0022	0.0220	0.0289	0.0023	C24	0.1484	0.1719	0.0078
B10	**0.1109**	**0.1514**	**0.0135**	**0.1104**	**0.1513**	**0.0136**	C27	0.3594	0.0703	0.0964

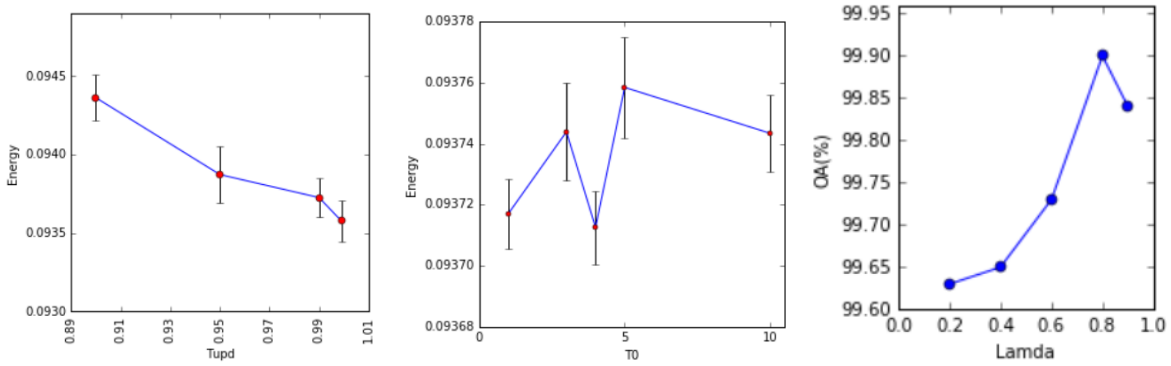
WB=winter barley (52 pixels), PT = potatoes (52pixels), Distance = 6 pixels

In the table above, the highlighted bands for Frost and Refined Lee have the strongest gradient values for CNN; each feature map gradient corresponds to each of the filters in the analysis. It is evident that CNN records the highest gradient for the filters related to C7, C16, and C23. This may be attributed to the fact that convolutional layers are able to automatically learn edges from the dataset, in addition to other spatial-contextual features learned by CNNs. Therefore, they have been used in edge detection studies in computer vision (e.g. Mohamed et al., 2013; Yoshida et al., 2004) and remote sensing image edge detection (e.g. Guo-bao et al., 2008). We can say that the CNN filter for C7, for instance, learns strongest edges in the image. On the other hand, Refined Lee and Frost filter have almost similar results, with the former performs slightly better. This is in line with the results obtained by Han, Guo, & Wang (2002) in their analysis on edge preservation by digital speckle filters. In general, contextual speckle filters such as the wavelet Bayesian denoising with Markov random fields (Xie, Pierce, & Ulaby, 2002) have been found to perform better than digital speckle filters even though they are not widely used due to the ease of use associated with filters such as Refined Lee and Frost. CNN filtering may still have an edge over the spatial contextual filters since it has the advantage of applying several filters to the same input with the combined effect resulting in better speckle filtering outcomes.

6.4. MRF AND SA PARAMETER ESTIMATION RESULTS

Figure 6.10 shows the results of the parameter tuning experiments for MRF and SA described in Section 4.4: for determining the optimal values of updating temperature T_{upd} (plot a), initial temperature T_0 (plot b), and smoothing parameter λ (plot c). For T_{upd} and T_0, the optimal values are the ones leading to lowest mean energy and standard deviation. For plots (a) and (b), as the values of T_{upd} and T_0 increase, the number of iterations also increases. This is because higher values result in a slower cooling schedule for the MRF. Therefore, further increasing T_{upd} towards 1 may result in even lower mean energy and standard deviation, but increase computational time. The results of T_0 show an unusual trend. We expected the mean energy to decrease with increase in T_0 up to a point and then remain constant onwards. This behavior shown in Figure 6.10 may be attributed to the fast cooling schedule used in order to speed up the processing. We chose the values with lowest mean energy and standard deviation as indicated on the graph as the optimal values for T_{upd} and T_0. The best value for λ on the other hand is taken as the value that leads to the highest overall accuracy: 0.8. To refine the value for λ, we tried other

values close to 0.8. However, these did not improve accuracy any further. Thus, in our implementation, we used T_{upd}: 0.999, T_0: 4.0, and λ: 0.80.



(a) (b) (c)

Figure 6.10: Plot of mean energy (blue line) with the respective error bars (black) derived from the standard deviation against Updating temperature (a), and initial temperature (b), and of overall accuracy against smoothing parameter (c).

6.5. PERFORMANCE ANALYSIS FOR THE CLASSIFIERS

In this Section, we present a comparative analysis of the classifiers using the performance metrics described in Section 5.4. A general discussion on computational time and complexity is presented in its own Subsection after the comparative analysis of the classifiers using the other performance metrics.

6.5.1. Comparison of CNN with CNN+MRF

In our first comparison, we analyzed the performance of the core algorithm without post processing, and when post processing with MRF was performed. The performance measures used for the analysis were overall accuracy, and visual inspection and comparison of classification maps.

Overall accuracy

Overall accuracy represents the percentage of pixels correctly classified by the classifier. The overall accuracy results for the CNN classifier and after post-processing with MRF are presented in Table 6.5. Generally, we can observe a slight increase in accuracy of only 0.34% with MRF post processing.

Table 6.5: Overall accuracy results for CNN and CNN with MRF

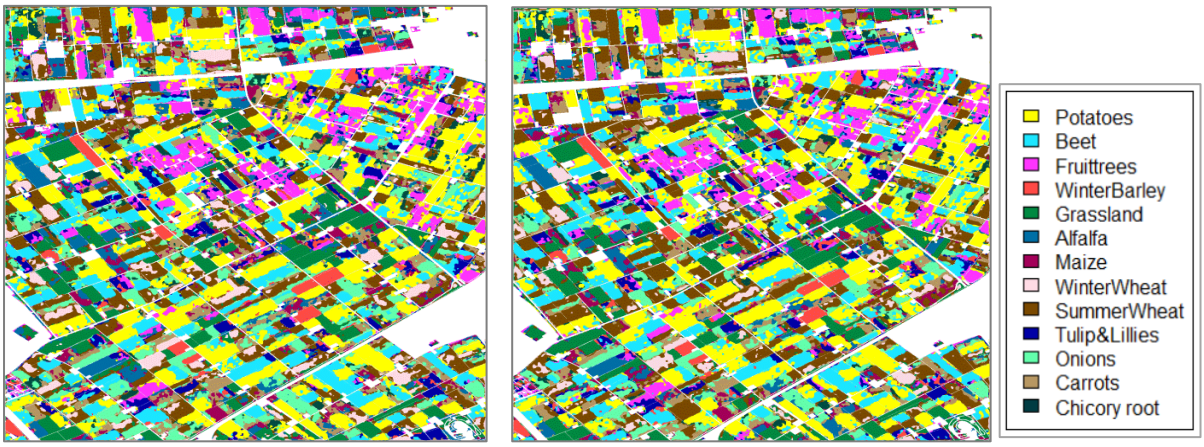
Classifier	Overall accuracy (%)
CNN	99.60
CNN+MRF	99.94

The results indicate that MRF improves the accuracy of classification. Thus, the addition of contextual label information in MRF processing produces in better classification results. Further analysis was performed on the resulting classification maps.

Visual inspection and comparison of classification maps

Since MRF essentially reduces the residual noise that remains after CNN speckle filtering, we compared the resulting classification maps as shown in Figure 6.11.

Concentrating on the classes Fruittrees and Summer Wheat in both images, CNN with MRF classified map has smoother and more regular crop fields for the classes compared to the CNN classified map. Other classes that benefit from the addition of the MRF are Potatoes and Beet. Hence, the addition of MRF for post-classification label refinement improves map regularity and increases the classifier's robustness to noise.



(a) (b) (c)

Figure 6.11: Classified maps for (a) CNN and (b) CNN with MRF, with the class legend (c). White spaces are non-crop areas.

6.5.2. Comparison of CNN with SVM and RF classifiers

Overall accuracy

Table 6.6 shows the overall classification accuracy results of the 5 classifiers over the subset image shown in Figure 4.1 Section 4.2 where all the reference polygons were used to extract training, validation and test samples. The CNN classifier implemented with an unfiltered multi-temporal SAR image outperforms the other 4 classifiers using a filtered multi-temporal SAR image.

Table 6.6: Overall accuracy results for CNN, SVM, and RF classifiers

Classifier	Overall Accuracy (%)
Pixel-based SVM	85.98
Pixel-based RF	87.98
SVM+GLCM	95.17
RF+GLCM	94.69
CNN	99.60

From the results, we noted a significant improvement in the overall classification accuracy with the addition of GLCM texture features. The accuracy of the SVM classifier improves by 9.4% when contextual information in the form of GLCM features is added and by 6.3% for RF. This demonstrates the importance of contextual information in the SAR image classification. On the other hand, CNN classifier still outperforms SVM and RF with a margin of 4% for SVM and 5% for RF, even with the addition of contextual information for the other two classifiers. As mentioned before, CNN automatically learns contextual features from the input for classification. The high accuracy results above are an indication that CNN classifier automatically learns more useful and less noisy features compared to GLCM features. Further analysis of the CNN and GLCM features in the classification with SVM and RF is provided in Subsection 6.5.3.

Producer accuracy

Producer accuracy measures the error of omission of the classifier. Errors of omission occur when a pixel is left out of the category/class being evaluated, to which it belongs. We infer from the producer accuracies, the classifier's sensitivity to a class of interest in the dataset. Table 6.7 presents the producer accuracies of the 3 classifiers. Overall, CNN classifier outperforms the other two classifiers for all classes in the image with an average producer accuracy of 99.60 % for CNN compared to 95.52% for SVM+GLCM and 92.73% for RF+GLCM.

Table 6.7: Producer accuracies computed for SVM+GLCM, RF+GLCM, and CNN classifiers

	Producer Accuracy (%)		
Class	SVM+GLCM	RF+GLCM	CNN
Potatoes	91.37	97.36	99.51
Beet	96.64	92.29	99.61
Fruittrees	96.69	93.2	100.00
Winter Barley	98.85	98.06	99.92
Grassland	97.80	95.51	99.47
Alfalfa	95.13	97.89	99.97
Maize	91.94	84.54	99.68
Winter wheat	98.84	93.39	99.80
Summer Wheat	99.46	93.98	100.00
Tulips	96.86	92.71	99.59
Onions	92.20	92.08	98.66
Carrots	90.21	84.4	98.83
Chicory root	95.81	90.05	99.77
Average Accuracy	95.52	92.73	99.60

SVM+GLCM classifier is least sensitive to Carrots (90.21%) and Potatoes (91.37%), RF+GLCM to carrots (84.4%) and Chicory root, and CNN to Carrots (98.83%) and Onions (98.66%). These classes, therefore, are more likely to have a higher number of misclassifications in the classified map than the other classes in the dataset. Additionally, CNN classifier has the lowest number of misclassifications with an average classifier sensitivity of 99.60% with RF+GLCM having the highest error rate with an average sensitivity of 92.73%. The high values reported can be attributed to the size of the test set, and are likely to change with a large sample of the test set. Overall, CNN classifier performs considerably better than the other classifiers.

User accuracy

User accuracy is a measure of the error of commission of the classifier. Errors of commission occur when a pixel is incorrectly included in the category/class being evaluated. We infer from the user accuracies, the classifier's reliability for a class of interest in the dataset.

Table 6.8: User accuracies computed for SVM+GLCM, RF+GLCM, and CNN classifiers

	User Accuracy (%)		
Class	SVM+GLCM	RF+GLCM	CNN
Potatoes	98.27	92.34	99.87
Beet	94.85	96.29	98.94
Fruittrees	89.82	94.00	100.00
Winter Barley	98.4	98.09	99.55
Grassland	88.62	96.45	99.58
Alfalfa	97.16	94.38	99.85
Maize	82.97	92.24	99.36
Winter wheat	98.46	100.00	100.00
Summer Wheat	90.82	95.23	98.75
Tulips	97.54	94.61	99.80
Onions	92.97	91.24	99.47
Carrots	90.43	90.84	99.67
Chicory root	92.38	98.51	100.00
Average Accuracy	93.28	94.94	99.60

SVM+GLCM classifier has low user accuracies for Maize (82.97%) and Grassland (88.62%), RF+GLCM for Carrots (90.84%) and Onions (91.24%), and CNN for Summer Wheat (98.75%) and Beet (98.84%). These classes, therefore, are more likely to get confused with other classes during classification, with varying extents based on the classifier's sensitivity. Unlike producer accuracy where SVM+GLCM classifier outperforms RF+GLCM, the overall user accuracy indicates that the results of RF+GLCM are more reliable (94.94%). This means that SVM is more likely to have more confusion between classes despite the reported high accuracies. CNN classifier maintains a very high accuracy for all the classes, with an overall reliability of 99.60%, similar to the reported average producer accuracy above, and outperforms the other classifiers. The classes being confused for each of the classifiers are analyzed by visually inspecting the classification maps in the next step.

Visual inspection and comparison of classification maps

Figure 6.12 shows the classification maps of the subset using the classifiers CNN, SVM+GLCM, and RF+GLCM. Generally, we can observe that CNN classifier has a smoother and less noisy map compared to the other two classifiers.

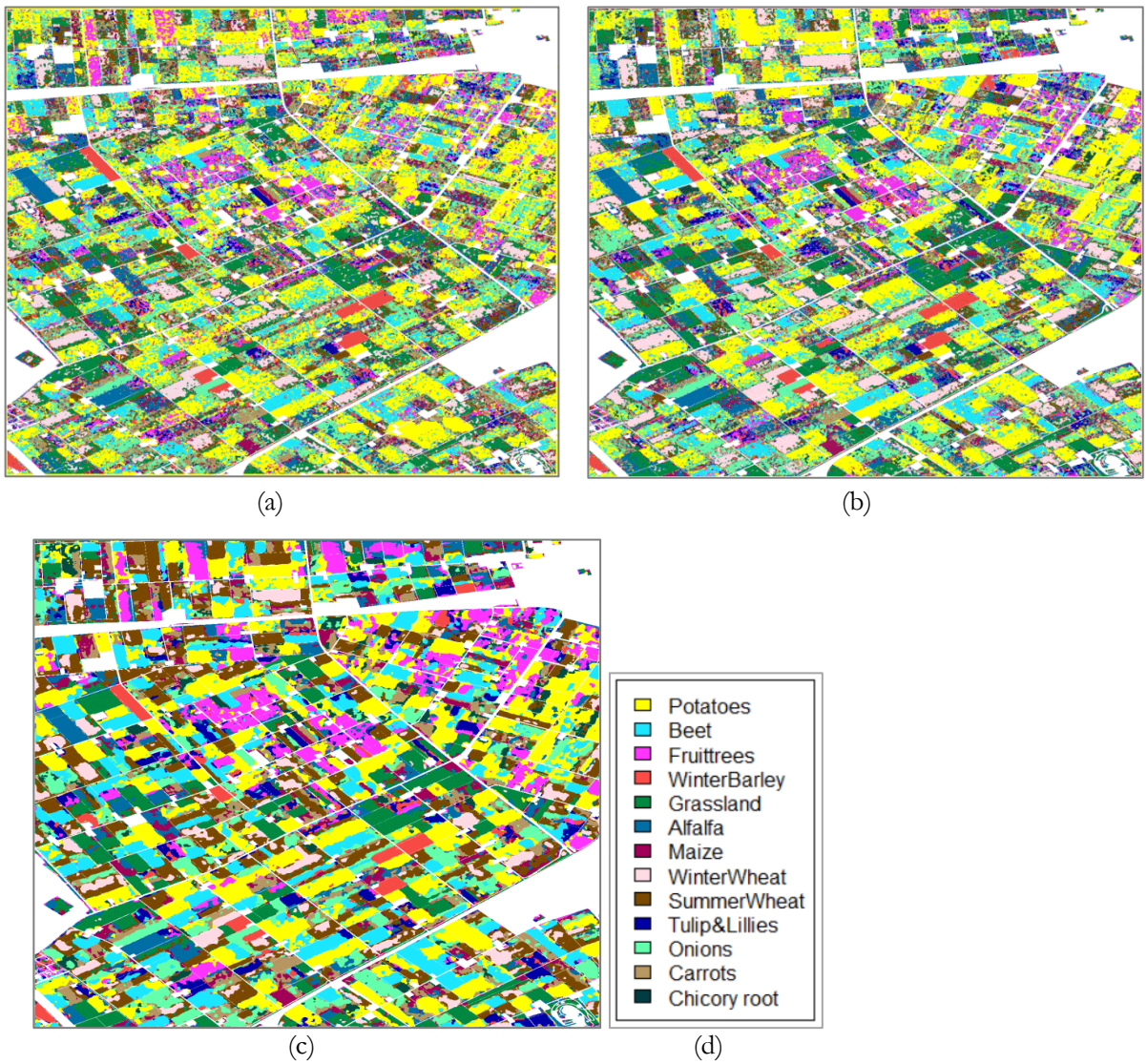


Figure 6.12: Classification maps for SVM+GLCM (a), RF+GLCM (b), CNN (c) classifiers, with the class legend (d). White spaces are non-crop areas

For SVM+GLCM and RF+GCLM, Potatoes (yellow) are confused with Beet (cyan). Given the producer accuracies in Table 6.6 for the two classes, we can say that SVM misclassifies Potatoes (90.21%) as Beet (96.64%), whereas RF+GLCM misclassifies Beet (92.29%) as Potatoes (97.36%). CNN classifier does not confuse the two classes as much as the other classifiers reflected by their respective producer accuracies. Other classes being confused are Winter Wheat and Summer Wheat; with Winter Wheat dominating for SVM+GLCM and RF+GLCM classifiers while Summer Wheat summer wheat dominates in the CNN classifier. The confusion between classes may be related to similarity in characteristics of the classes being confused, in that they have similar responses to the radar signal. The radar signal is sensitive to the large-scale crop structure (size, shape, and orientation of leaves, stalks, and fruit) and the dielectric properties of the crop canopy (McNairn & Brisco, 2004). Therefore, cereals of similar structure are more likely to get confused with each other depending on the classifier's generalization capability. The smoothness of the CNN classifier can be attributed to the classifier's ability to reduce noise to a larger extent with the various speckle filters in the network, making it more robust to the residual noise that remains after speckle filtering. The RF+GLCM classified map, on the other hand, has less noise compared to the SVM+GLCM classified map.

6.5.3. A comparison of CNN features with GLCM features using SVM and RF classifiers

To further investigate the effectiveness of CNN as a feature learning approach, we extracted CNN features using the trained network, stacked them to the filtered image and classified the resulting image with SVM and RF. The results of the performance analysis are presented below.

Overall accuracy

Table 6.9 shows the overall classification accuracy of the SVM and RF classifiers with GLCM features and features automatically learnt by a trained CNN classifier

Table 6.9: Overall accuracies results for SVM and RF with GLCM and CNN features

Classifier	Overall Accuracy (%)
SVM+GLCM	95.19
RF+GLCM	94.69
SVM+CNN	94.96
RF+CNN	97.55

SVM slightly performs better by 0.21% with GLCM features than with CNN features, whereas RF classifier improves by about 2.5% when CNN features are used as additional contextual information.

Producer accuracy

As explained, producer accuracy is used to estimate the classifier's sensitivity to the classes in the image being classified. Table 6.10 presents the producer accuracies of SVM and RF classifiers with GLCM and CNN features. Overall, the sensitivity of the SVM reduces by 1.88% when CNN features are used compared to the increase in sensitivity of 3.02% for RF with CNN features.

Table 6.10: Producer accuracies computed for SVM+GLCM, RF+GLCM, and SVM+CNN, RF+CNN classifiers

Class	Producer Accuracy (%)			
	SVM+GLCM	RF+GLCM	SVM+CNN	RF+CNN
Potatoes	91.37	97.36	96.53	98.91
Beet	96.64	92.29	93.24	95.73
Fruittrees	96.69	93.2	92.77	98.16
Winter Barley	98.85	98.06	97.66	98.63
Grassland	97.80	95.51	94.05	96.34
Alfalfa	95.13	97.89	97.67	99.49
Maize	91.94	84.54	93.83	94.7
Winter wheat	98.84	93.39	97.1	94.62
Summer Wheat	99.46	93.98	90.38	95.25
Tulips	96.86	92.71	91.03	97.28
Onions	92.20	92.08	92.41	94.69
Carrots	90.21	84.4	86.92	96.2
Chicory root	95.81	90.05	93.72	96.61
Average Accuracy	95.52	92.73	93.64	96.66

Similar to SVM+GLCM, SVM with CNN features is less sensitive to Carrots (86.92%). However, the performance of Potatoes improves by about 5%. RF+CNN classifier improves for all classes with an average producer accuracy of 96.66% and outperforms all the other classifiers, RF+GLCM, SVM+CNN, and SVM+GLCM.

User accuracy

Table 6.10 presents the user accuracies of the SVM and RF classifiers with GLCM and CNN features. Overall, the results of the SVM and RF classifiers are more reliable with CNN features performing better than GLCM features in the classification.

Table 6.11: User accuracies computed for SVM+GLCM, RF+GLCM, and SVM+CNN, RF+CNN classifiers

Class	User Accuracy (%)			
	SVM+GLCM	RF+GLCM	SVM+CNN	RF+CNN
Potatoes	98.27	92.34	94.75	96.53
Beet	94.85	96.29	93.29	97.02
Fruittrees	89.82	94.00	86.71	94.88
Winter Barley	98.4	98.09	98.53	99.31
Grassland	88.62	96.45	95.11	97.17
Alfalfa	97.16	94.38	96.99	98.32
Maize	82.97	92.24	88.76	96.69
Winter wheat	98.46	100.00	98.53	99.62
Summer Wheat	90.82	95.23	93.47	96.21
Tulips	97.54	94.61	94.31	98.89
Onions	92.97	91.24	94.39	97.68
Carrots	90.43	90.84	94.25	95.32
Chicory root	92.38	98.51	95.00	99.56
Average Accuracy	93.28	94.94	94.16	97.48

Unlike the producer accuracies, average use accuracy for SVM increases by 0.88% when CNN features are used. The user accuracy of Maize additionally improves by 5.79% as that of Fruit trees reduces by about

3%. Overall, basing on the individual class accuracies, we cannot conclude on which features are best for SVM since it performs slightly better in some classes with GLCM and in others with CNN. However, in the poorly performing classes such as Maize, CNN features result in a better classification result. On the other hand, RF performs considerably better for all classes except Winter Wheat which has a slight reduction in accuracy of 0.38%.

Visual inspection and comparison of classification maps

In addition to the performance metrics used above, we visually inspected and compared the classification maps for all the classifiers above. Figure 6.13 shows the classification maps of SVM and RF classifiers with GLCM and CNN features. Overall, the classified maps are less noisy for the CNN feature results compared to GLCM maps, implying that features automatically learned by CNN are less noisy compared to the GLCM features manually extracted from subset image after speckle filtering.

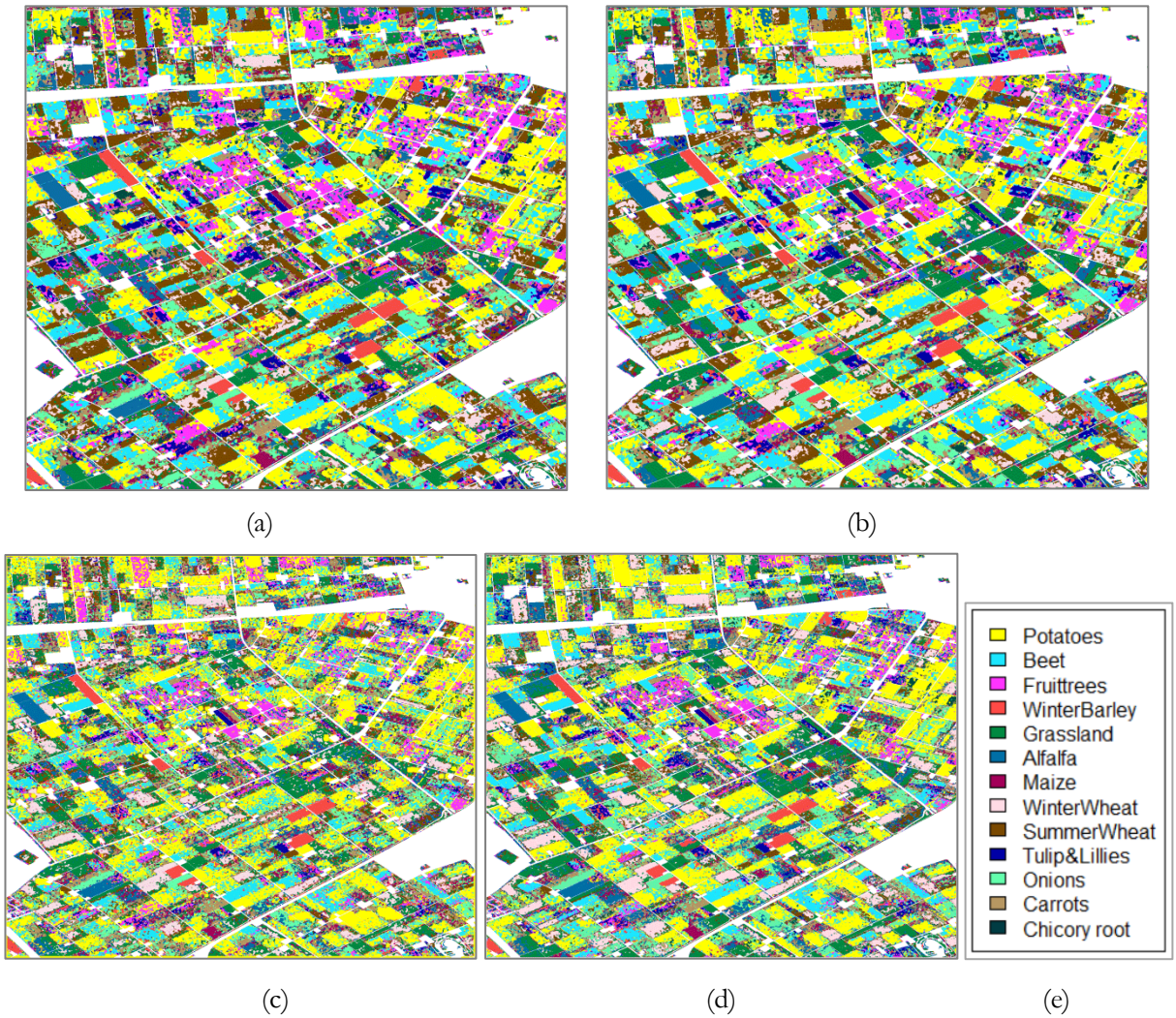


Figure 6.13: Classification maps for SVM+CNN (a), RF+CNN (b), SVM+GLCM(c), RF+GLCM (d), with the class legend (e).

The white spaces are non-crop areas.

From the classified maps, we observed that Potatoes and beet are now less confused and less noisy in the classification with CNN features for both SVM and RF. Additionally, Winter Wheat and Summer Wheat are also less confused in the in the classification maps for CNN features compared to the GLCM maps where Summer Wheat is confused with Winter Wheat for both SVM and RF classifiers. Overall, the use of CNN features results in classification maps with less noise, compared to GLCM features. This is not

surprising due to the very good performance of the CNN in the speckle filtering as reported in Section 6.3. Hence, we expect the features automatically learned by CNN to be comparably less noisy. On the other hand, the residual noise that remains in the filtered image with Frost and Refined Lee filters is a lot, as reflected in the GLCM features extracted after speckle filtering.

6.5.4. On CNN for feature learning and end-to-end CNN classification

We presented the results of classification with CNN trained end-to-end in Subsection 6.5.2 and with RF and SVM using features extracted when CNN is used as a feature learning approach in Subsection 6.5.3. End-to-end CNN outperforms the classifiers with CNN features in terms overall accuracy, producer, and user accuracies. Additionally, the CNN classified map is more regularized and less noisy in comparison. This implies that the proposed CNN trained end-to-end is more robust to the noise that remains after filtering. Overall, considering the performance measures used, CNN is superior to the other classifiers using features learned by the CNN.

6.5.5. On computational time and complexity

All analyses with the different classifiers were run on a Dell Precision M2800 laptop, with an Intel core i7-4710Mq CPU (2.50 GHz) processor, RAM of 8.00 GB and a 64-bit Windows operating system.

In the first part, the addition of MRF post processing increased the computational time of the CNN+MRF classifier. The MRF algorithm was prepared in RStudio (2015) environment with packages rgdal (Bivand et al., 2015) and Rcpp (Eddelbuettel & Francois, 2011). The computational time of the MRF algorithm is influenced by the value of the initial temperature T_0 and updating temperatureT_{upd}. However, the effect of T_{upd} is more significant. A small value results in a fast cooling schedule where convergence is achieved within a short time while as a large value results in a much slower cooling schedule. The MRF classification took about 19 minutes with the selected optimal parameters, with all computation graphs active for the subset image with dimensions 1243×1393.

The classifiers SVM, RF, and CNN were implemented in a python environment with their respective packages and their dependencies. SVM and RF outperformed CNN classifier in this performance metric. It took at least 1.5 days to train a CNN model when a holdout cross validation scheme was used to select the best learning and regularization parameters. Additionally, classifying the whole image subset took at least 30 minutes with a small patch size of 15. For SVM with GLCM, it took approximately 2.5 hours to train, with a holdout cross validation scheme for selecting the optimal values of the parameters C and γ. 1 hour was added for GLCM feature extraction and selection. RF registered the lowest computation time. This is because the computational complexity of RF is simplified by the reduction of the number of input features at each internal node. RF algorithm took at most 1.25 hours for classification, feature extraction and selection included. When a pre-trained CNN was used to extract features, the operation took at most 20 minutes. No feature selection was required for CNN, thus reducing the computational time for SVM+CNN and RF+CNN, without considering the training time for the CNN classifier that was modified into a feature extractor.

In terms of complexity, CNN is more complex than all the other classifiers, with convolutional layers having 650925 parameters each applying multiplication followed by the nonlinear activation and pooling operations in each unit of the succeeding layers compared to SVM and RF each with two parameters. The CNN classifier's poor performance can be attributed to the fact that CNN filters are learning their weights from the raw data, and perform feature extraction and speckle filtering, in addition, classification. Therefore, all the processing steps occur under a single framework, making it computationally heavy, unlike SVM and RF where the processing steps are carried out independent of each other.

6.6. SUMMARY OF RESULTS

In summary, we presented the results of all analysis carried out on our proposed algorithm and existing methods for speckle filtering, feature extraction, and classification. We presented and interpreted the results of the CNN sensitivity analysis to the hyperparameters: patch size, kernel size, the number of filters and network depth. Generally, with a fixed training set, the overall accuracy increases as the parameters are varied up to a certain value of the parameters and then drops. For the number of MLP layers, on the other hand, a downward trend was observed. Also, the network's complexity increases as the more layers are added to the network. We also find that a high value of patch size results in over smoothing even with increased accuracy. We also observed and interpreted the effect of the above hyperparameters on CNN's speckle filtering ability, where a similar trend as the sensitivity analysis was observed, with improved speckle filtering as the values of the hyperparameters were varied. We also find that CNN as a speckle filter is superior to existing speckle filters such as the Refined Lee filter and Frost filter. Furthermore, the addition of MRF to the CNN for post classification label refinement further reduces the noise that remains after CNN speckle filtering improves map regularity. When compared to existing classifiers such as SVM and RF, CNN classifier outperforms them on all performance metrics except computational time and complexity. Finally, we compared the manually engineered GLCM features to features learned automatically by CNN, in a classification with SVM and RF. Our findings show that CNN features are less noisy and thus improve map quality for SVM and RF classifiers.

With the results of the research discussed in this chapter, we conclude the thesis in the next chapter, where we review the questions posed in Chapter 1 in line with our findings and discussion from the experiments and analysis carried out, as well as the additional questions that were answered during our analysis

CHAPTER 7

CONCLUSIONS AND RECOMMENDATIONS

7.1. CONCLUSIONS

The main objective of this study was to develop a new method adopting a deep learning approach based on convolutional neural networks that learn from the data the speckle reduction filters, extract spatial-contextual features and learn the classification rule for land cover classification. We reviewed CNN network architectures and chose CNN architecture with an MLP at the end the SAR speckle filtering, feature extraction, and classification problem. The effect of varying CNN hyperparameters on the performance of the classifier and on the ability of the CNN to reduce speckle was investigated. From the knowledge obtained in the design analysis and speckle filtering experiments, we came up with the design of a CNN for speckle filtering and classification. The CNN learns spatial-contextual features useful for characterizing the various scatterers in the scene and simultaneously performs speckle filtering using weights of filters learned directly from the data. We evaluated CNN's speckle filtering ability against Frost and Refined Lee filters. We also investigated a post-classification processing approach, MRF for improving map regularity and further reducing residue speckle and evaluated the CNN classifier against SVM and RF approaches, that use speckle filtered images and hand crafted GLCM features in classification. Several performance measures were used in the comparison such as the equivalent number of looks (ENL) and edge preservation for speckle filtering, and overall accuracy, producer and user accuracy, visual inspection, and comparison of classification maps and computational time and complexity for the classifiers. CNN as a speckle filter outperforms Frost and Refined Lee filters in all the performance measures. The performance of CNN is attributed to the fact that it uses several filters to perform the speckle filtering operation in the various convolutional layers in the network, has the ability to learn edges from the data, hence good edge preservation compared to the other filters. We also found MRF to improve the map regularity by reducing the residue speckle. For the classifier comparison, we found that CNN outperforms SVM and RF in all performance measures except computation time and complexity. This may be attributed to the fact that CNN performs automatic feature learning, speckle filtering, and classification under a single framework while as for SVM and RF, the three processing steps are performed independently and the classifiers only perform classification. Finally, we compared the automatically extracted features by a CNN to handcrafted GLCM features and found them to improve accuracy for RF classification and better quality maps with less noise for both RF and SVM compared to classification with GLCM features.

We present the answers to the research questions posed in Section 1.2 as follows:

Network architectures

1. What network architectures have been proposed and studied in the literature and how do they work?
   In Chapter 2, we discussed the different network architectures for convolutional neural networks as found in the existing literature: CNN, fully convolutional networks (FCN), recurrent CNN and Deconvolutional networks.
2. What is the network architecture to be used for SAR image denoising and classification?

We chose the CNN architecture that utilizes one or more fully connected layers (MLP) at the end of the convolutional-non-linearity-pooling layers as the foundation of our convolutional neural network based approach. In this CNN architecture, the CNN part performs speckle filtering and feature extraction simultaneously. The MLP part learns the classification rule from the extracted and hopefully less noisy spatial-contextual features.

Design analysis and implementation of the CNN algorithm

1. What is the optimal CNN structure (e.g. number of hidden layers, the number of neurons in the hidden layer) of the classifier to address the speckle reduction and classification problem?

   We discussed the design decisions of the selected network architecture and introduced the terms and notations related to the design in Chapter 3, Section 3.1. To determine the optimal structure of the CNN classifier, we carried out design experiments. We varied the hyperparameters of the network such as patch size, kernel size, the number of filters, the number of convolutional layers and the number of fully connected MLP layers in the sensitivity analysis experiments. The results in Subsection 6.1.1 show the effect of varying these hyperparameters on the overall classification accuracy. In general, increasing the values of the hyperparameters with a fixed training set also increase the overall accuracy for the patch size, until such a point that the classifier starts to over smoothen the resulting classification map, with the regular crop field shapes getting distorted. This was attributed to the resolution of the images. Therefore, a smaller patch size produces reasonable results with medium resolution images. For kernel size and number of filters, the accuracy increases until reaching a peak with a certain hyperparameter value and then drops afterwards. On the other hand, increasing the number of convolutional layers with a larger training set only slightly improved the accuracy results, especially from the 2 to 3 layers, implying that the classifier benefited from higher order features extracted in the added convolutional layer, while as increasing the number of dense layers does not seem to improve classification accuracy. We also compared the results of patch size, kernel size and number of filters, keeping all the other parameters constant and found that the CNN was more sensitive to the change in patch size as indicated by the percentage change in overall accuracy. In addition to sensitivity analysis, the hyperparameters' effect on the ability of the CNN to reduce noise was also analyzed by varying the hyperparameters kernel size, patch size and the number of filters using the CNN part of the resulting trained network to extract features, which were used to compute the equivalent number of looks. Generally, increasing the values of the hyperparameters improves the network's ability to reduce noise as reflected by the results of the experiments shown in Section 6.2. On the basis of the results of these experiments, the designed CNN consists of three convolutional layers, 1 fully connected layer, with a patch size of 15, kernel size 5 and 32 filters.

2. What are the optimal values for the model training parameters and the regularization parameters? To determine the optimal values of the learning and regularization parameters: learning rate η, learning rate decay η_d, weight decay λ_w, early stopping criteria e_n, and dropout we used a hold-out cross validation scheme as explained in Chapter 4 under Section 4.2. The parameter space was limited to only the set values and a full parameter search was not conducted was it would be very costly in terms of computation time. As noted in the classifier comparisons, CNN is at a big disadvantage when it comes to the computation time.

3. What performance measures (e.g. equivalent number of looks, classification accuracy, computation time and complexity) are relevant for assessing speckle filtering and the classifier?

   In this study, we first evaluated the CNN as a speckle filter against two other speckle filtering approaches using performance measures: ENL and edge preservation. Then, we evaluated the CNN classifier against SVM and RF classifiers with GLCM features using performance measures:

overall accuracy, producer accuracy, user accuracy, visual inspection and comparison of classification maps and computational time and complexity.

Performance comparison

1. Which approach performs better and in which aspects of the performance measures?

   In the comparison of the speckle filters, our findings show that CNN performs better than Frost and Refined Lee filter for both ENL and edge preservation. This can be attributed to the fact that for a CNN, several convolutional layers perform speckle filtering using several filters, depending on the configuration of the network. Additionally, CNN has the ability to automatically learn edges from the raw images, making it better at edge preservation.

   In the comparison with SVM and RF which use speckle filtered images and additional handcrafted GLCM features, our findings show that the CNN classifier outperforms the other classifiers for all metrics except computational time and complexity where it is at a disadvantage. The classified map for CNN was less noisy compared to the other classified maps of SVM and RF, which is further proof of the CNN's filtering ability and robustness to noise that remains after filtering.

2. What is the difference between feature learning (with CNN) and manual feature engineering approaches in terms of performance?

   We compared the features learnt by a CNN to the manually engineered GLCM features with an SVM and RF classification. Our findings show that for RF, CNN features outperform GLCM features in all performance metrics considered. Specifically, the resulting classification map is less noisy, confirming our suspicion that CNN extracts less noisy features which improve a classifier's robustness to noise. For SVM, GLCM features outperform slightly better than CNN features in overall accuracy and producer accuracy (sensitivity), with CNN performing better in user accuracy (reliability) and with a less noisy classification map.

We also answered additional questions we encountered in the course of our analysis such as:

- How subsampling affect the classification results of the CNN, given that the image resolution is medium?

  We found out that subsampling only helps to improve the performance of CNN in terms of computational time but has no effect on accuracy. No subsampling approach is more desirable for CNN feature extraction for further analysis (see Subsection 6.1.2).

- How does the pooling strategy affect the network's ability to deal with noise?

  Our findings show that the average pooling outperforms max pooling in speckle filtering with max pooling performing better in classification (Subsection 6.2.5).

- How can the results of the CNN be improved?

  We investigated MRFs for post classification label refining and our findings show that MRF improves map regularity and therefore the CNN classifier's robustness to noise (Subsection 6.5.1)

- How does end-to-end CNN compare with CNN as a feature learning approach?

  We compared the results of SVM and RF classification utilizing features learnt by A CNN with the results of our CNN classification. We found that end-to-end CNN still outperformed the other two classifiers using CNN features and resulted in a smoother and more regularized classified map, making it more robust to the noise that remains after speckle filtering with CNN.

We presented in this study a convolutional learning approach for automatic feature extraction, denoising, and classification of SAR images, all within a single framework. We investigated the various architectural elements for the both speckle filtering and classification. We analyzed the performance of our approach with existing methods and found that apart from computational time and complexity, it outperforms the alternative classifiers. In conclusion, we can say our approach has great potential and can be beneficial in SAR image analysis by automating all the analysis tasks under a single framework for large scale monitoring systems in agriculture, forestry, and disaster monitoring among others.

7.2. RECOMMENDATIONS

- Explore other network architectures such as FCN and RCNN using knowledge obtained from this study as a starting point. Since other networks discussed in Chapter 2 also have some similar properties to the CNN (utilizing filters with weights learned from the data). Their unique characteristics may also be useful in speckle filtering and classification, although they were not explored further in this study.
- Investigate other combinations of values for learning and regularization parameters since due to time constraints, it was not possible to adequately search the parameter space
- Combining Sentinel-1 SAR images with Setinel-2 optical images for a crop type classification. The spectral information obtained from optical images may complement SAR image information, thereby producing better classification results.
- Compare CNN with other contextual speckle classifiers which have been known to perform better than the standard digital speckle filters that are commonly used.
- Consider other handcrafted features such as Local Binary Patterns (LBP) features and other sources of additional information such as vegetation indices.
- Further investigate the effect of resolution for such a classification with images obtained at various resolutions.

LIST OF REFERENCES

Archer, K. J., & Kimes, R. V. (2008). Empirical characterization of random forest variable importance measures. *Computational Statistics and Data Analysis*, *52*(4), 2249–2260. http://doi.org/10.1016/j.csda.2007.08.015

Bergado, J. R. A. (2016). *A deep feature learning approach to urban scene classification*. University of Twente Faculty of Geo-Information and Earth Observation (ITC). Retrieved from http://www.itc.nl/library/papers_2016/msc/gfm/bergado.pdf

Bergado, J. R., Persello, C., & Gevaert, C. (2016). A deep learning approach to the classification of sub-decimetre resolution aerial images. In *2016 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)* (pp. 1516–1519). IEEE.

Besag, J. (1974). Spatial interaction and the statistical analysis of lattice systems. *Journal of the Royal Statistical Society. Series B (* …, *36*(2), 192–236.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (information science and statistics)*. *Journal of Chemical Information and Modeling*. New York: Springer-Verlag.

Bivand, R., Keitt, T., & Rowlingson, B. (2015). rgdal: Bindings for the Geospatial Data Abstraction Library. R package version 1.1-3. Retrieved from https://cran.r-project.org/package=rgdal

Breiman, L. (2001). Random forests. *Machine Learning*, *45*(1), 5–32.

Browne, M., & Ghidary, S. S. (2003). Convolutional Neural Networks for Image Processing : An application in Robot Vision. *Image Processing*, 641–652.

Burges, C. J. (1998). A Tutorial on Support Vector Machines for Pattern Recognition. *Data Mining and Knowledge Discovery*, *2*(2), 121–167.

Chen*, D., Stow, D. a., & Gong, P. (2004). Examining the effect of spatial resolution and texture window size on classification accuracy: an urban environment case. *International Journal of Remote Sensing*, *25*(11), 2177–2192.

Chen, K., & Huang, W. P. (1996). Classification of multifrequency polarimetric SAR imagery using a dynamic learning neural network. *IEEE Transactions on Geoscience and Remote Sensing, 34(3), 814–820*.

Chen, S., Member, S., Wang, H., Xu, F., & Member, S. (2016). Target Classification Using the Deep Convolutional Networks for SAR Images. *IEEE Transactions on Geoscience and Remote Sensing*, *54*(8), 4806–4817.

Chorowski, J., Bahdanau, D., Serdyuk, D., Cho, K., & Bengio, Y. (2015). Attention-Based Models for speech recognition, 1–19. *arXiv preprint arXiv:1506.07503v1, 2015*

Collins, M. J. (2000). The effect of speckle filtering on scale-dependent texture estimation of a forested scene. *IEEE Transactions on Geoscience and Remote Sensing*, *38*(3), 1160–1170.

Cortes, C., & Vapnik, V. (1995). Support-Vector Networks. *Machine Learning*, *20*(3), 273–297.

Dasari, K., Anjaneyulu, L., Jayasri, P. V., & Prasad, A. V. V. (2016). The importance of speckle filtering in image classification of SAR data. *International Conference on Microwave, Optical and Communication Engineering, ICMOCE*, 349–352.

Ding, J., Chen, B., Liu, H., & Huang, M. (2016). Convolutional Neural Network with Data Augmentation for SAR Target Recognition. *IEEE Geoscience and Remote Sensing Letters*, *13*(3), 364–368.

Du, P., Samat, A., Waske, B., Liu, S., & Li, Z. (2015). Random Forest and Rotation Forest for fully polarized SAR image classification using polarimetric and spatial features. *ISPRS Journal of Photogrammetry and Remote Sensing*, *105*, 38–53.

Eddelbuettel, D., & Francois, R. (2011). Rcpp: Seamless R and C++ I ntegration. *Journal of Statistical*

Software, *40*(8). Retrieved from http://www.jstatsoft.org/v40/i08/

ESA. (2010). Revised ESA Data Policy for ERS , Envisat and Earth Explorer missions. *Earth*, , 1–3.

Frery, A. C., Correia, A. H., & Freitas, C. D. C. (2007). Classifying multifrequency fully polarimetric imagery with multiple sources of statistical evidence and contextual information. *IEEE Transactions on Geoscience and Remote Sensing*, *45*(10), 3098–3109.

Frost, V. S., Stiles, J. a, Shanmugan, K. S., & Holtzman, J. C. (1982). A model for radar images and its application to adaptive digital filtering of multiplicative noise. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *4*(2), 157–166.

Fukuda, S., & Hirosawa, H. (2001). Support Vector Machine Classification of Land Cover : Application to polarimetric SAR Data. *IEEE Geoscience and Remote Sensing Symposium (IGARSS)*, *1*(5), 187–189.

Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)*, *9*, 249–256.

Glorot, X., Bordes, A., & Bengio, Y. (2011). Deep sparse rectifier neural networks. *AISTATS '11: Proceedings of the 14th International Conference on Artificial Intelligence and Statistics*, *15*, 315–323.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. Book in preparation for MIT press. Retrieved December 21, 2016, from http://www.deeplearningbook.org/version-2016-01-30/index.html

Graves, A., Mohamed, A., & Hinton, G. (2013). Speech Recognition with Deep Recurrent Neural Networks. *Icassp*, (3), 6645–6649.

Guo-bao, X., Gui-yan, Z., Lu, Y., Yi-xin, Y., & Yu-li, S. (2008). A CNN-based Edge Detection algorithm for Remote Sensing Image. *Control*, 2558–2561.

Han, C., Guo, H., & Wang, C. (2002). Edge preservation evaluation of digital speckle filters. In *IEEE International Geoscience and Remote Sensing Symposium* (Vol. 4, pp. 2471–2473). IEEE.

Haralick, R. M., Shanmugam, K., & Dinstein, I. (1973). Textural features for image classification. *IEEE Transactions on Systems, Man, and Cybernetics*, *3*(6), 610–621.

Horning, N. (2010). Random Forests: An algorithm for image classification and generation of continuous fields data sets. *International Conference on Geoinformatics for Spatial Infrastructure Development in Earth and Allied Sciences 2010*, 1–6.

Jiao, X., Kovacs, J. M., Shang, J., McNairn, H., Walters, D., Ma, B., & Geng, X. (2014). Object-oriented crop mapping and monitoring using multi-temporal polarimetric RADARSAT-2 data. *ISPRS Journal of Photogrammetry and Remote Sensing*, *96*, 38–46.

Kirkpatrick, S., Gelatt, C. D., & Vecch, M. P. (1983). Optimization by Simulated Annealing. *Science*, *220*(4598), 671–680.

Krizhevsky, A., Sutskever, I., & Geoffrey E., H. (2012). Imagenet. *Advances in Neural Information Processing Systems 25 (NIPS2012)*, 1–9.

Krylov, V. A., & Zerubia, J. (2011). Synthetic aperture radar image classification via mixture approaches. In *2011 IEEE International Conference on Microwaves, Communications, Antennas and Electronic Systems (COMCAS 2011)* (pp. 1–8). IEEE.

LeCun, Y., & Bengio, Y. (1995). Convolutional networks for images, speech, and time series. *The Handbook of Brain Theory and Neural Networks*, *3361*(April 2016), 255–258.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2323.

Lee, J.-S., Pottier, E., & Thompson, B. J. (2009). *Polarimetric Radar Imaging : From Basics to Applications*. Baton Rouge: CRC Press.

Lee, J. S., Grünes, M. R., Ainsworth, T. L., Du, L. J., Schuler, D. L., & Cloude, S. R. (1999). Unsupervised classification using polarimetric decomposition and the complex wishart classifier. *IEEE Transactions on Geoscience and Remote Sensing*, *37*(5 pt 1), 2249–2258.

Lee, J. S., Grunes, M. R., & Grandi, G. De. (1997). Polarimetric SAR speckle filtering and its impact on classification. *IGARSS'97. 1997 IEEE International Geoscience and Remote Sensing Symposium Proceedings. Remote Sensing - A Scientific Vision for Sustainable Development*, *2*(5), 2363–2373.

Lee, J. S., Grunes, M. R., & Kwok, R. (1994). Classification of multi-look polarimetric SAR imagery based on complex Wishart distribution. *International Journal of Remote Sensing*, *15*(11), 2299–2311.

Lee, J. Sen, Grunes, M. R., & Mango, S. A. (1991). Speckle Reduction in Multipolarization, Multifrequency Sar Imagery. *IEEE Transactions on Geoscience and Remote Sensing*, *29*(4), 535–544.

Lee, J. Sen, Grunes, M. R., & Pottier, E. (2001). Quantitative comparison of classification capability: Fully polarimetric versus dual and single-polarization SAR. *IEEE Transactions on Geoscience and Remote Sensing*, *39*(11), 2343–2351.

Long, J., Shelhamer, E., & Darrell, T. (2015). [Slices] Fully convolutional networks for semantic segmentation. *Cvpr 2015*, 3431–3440.

Lopes, A., Touzi, R., & Nezry, E. (1990). Adaptive Speckle Filters and Scene Heterogeneity. *IEEE Transactions on Geoscience and Remote Sensing*, *28*(6), 992–1000.

Lopez-Martinez, C., Fàbregas, X., López-martínez, C., Member, S., & Fàbregas, X. (2003). Polarimetric SAR Speckle Noise Model. *IEEE Transactions on Geoscience and Remote Sensing*, *41*(10), 2232–2242.

McNairn, H., & Brisco, B. (2004). The application of C-band polarimetric SAR for agriculture: A review. *Canadian Journal of Remote Sensing*, *30*(3), 525–542.

McNairn, H., Jiali Shang, Xianfeng Jiao, & Champagne, C. (2009). The Contribution of ALOS PALSAR Multipolarization and Polarimetric Data to Crop Classification. *IEEE Transactions on Geoscience and Remote Sensing*, *47*(12), 3981–3992.

Melgani, F., & Serpico, S. B. (2003). A Markov Random Field Approach to Spatio-Temporal Contextual Image Classification. *IEEE Transactions on Geoscience and Remote Sensing*, *41*(11), 2478–2487.

Mikolov, T., Kombrink, S., Burget, L., Cernocky, J., & Khudanpur, S. (2011). Extensions of recurrent neural network language model. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, 5528–5531.

Mohamed A. El-Sayed Yarub A. Estaitia, M. a K. (2013). Automated Edge Detection Using Convolutional Neural Network. *International Journal of Advanced Computer Science and Applications(IJACSA)*, *4*(10), 11–17. Retrieved from http://ijacsa.thesai.org/

Moser, G., & Serpico, S. B. (2013). Combining support vector machines and Markov random fields in an integrated framework for contextual image classification. *IEEE Transactions on Geoscience and Remote Sensing*, *51*(5), 2734–2752.

Nasien, D., Haron, H., & Yuhaniz, S. S. (2010). Support Vector Machine (SVM) for english handwritten character recognition. *2010 2nd International Conference on Computer Engineering and Applications, ICCEA 2010*, *1*, 249–252.

Nebauer, C. (1998). Evaluation of convolutional neural networks for visual recognition. *IEEE Transactions on Neural Networks / A Publication of the IEEE Neural Networks Council*, *9*(4), 685–696.

Noh, H., Hong, S., & Han, B. (2015). Learning Deconvolution Network for Semantic Segmentation. *Iccv*, *1*, 1520–1528.

Novak, L. M., & Burl, M. C. (1990). Optimal speckle reduction in polarimetric SAR imagery. *Aerospace and*

Electronic Systems, IEEE Transactions on, *26*(2), 293–305.

Oliver, C., & Quegan, S. (2004). *Understanding Synthetic Aperture Radar Images*. Raleigh, NC: Scitech Publishing, Inc.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., … Duchesnay, É. (2012). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Pinheiro, P., & Collobert, R. (2014). Recurrent convolutional neural networks for scene labeling. *Proceedings of The 31st International Conference on Machine Learning*, *32*(June), 82–90.

Pontil, M., & Verri, A. (1998). Support vector machines for 3D object recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *20*(6), 637–646. http://doi.org/10.1109/34.683777

RStudio Team. (2015). RStudio: Integrated Development for R. Boston, MA: Inc. Retrieved from http://www.rstudio.com/.

Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., & LeCun, Y. (2013). OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks. *arXiv Preprint arXiv*, 1312.6229.

Sherrah, J. (2016). Fully Convolutional Networks for Dense Semantic Labelling of High-Resolution Aerial Imagery. *arXiv*, 1–22.

Skriver, H. (2012). Crop Classification by Multitemporal C- and L-Band Single- and Dual-Polarization and Fully Polarimetric SAR. *IEEE Transactions on Geoscience and Remote Sensing*, *50*(6), 2138–2149.

Skriver, H., Mattia, F., Satalino, G., Balenzano, A., Pauwels, V. R. N., Verhoest, N. E. C., & Davidson, M. (2011). Crop classification using short-revisit multitemporal SAR data. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, *4*(2), 423–431.

Skriver, H., Svendsen, M. T., & Thomsen, A. G. (1999). Multitemporal C- and L-band polarimetric signatures of crops. *IEEE Transactions on Geoscience and Remote Sensing*, *37*(5), 2413–2429.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: prevent NN from overfitting. *Journal of Machine Learning Research*, *15*, 1929–1958.

Stevens, M. R., Monnier, C., & Snorrason, M. S. (2005). Parameter adaptation for target recognition in LADAR. In F. A. Sadjadi (Ed.), (p. 201).

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., … Rabinovich, A. (2015). Going deeper with convolutions. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, *07–12–June*, 1–9.

Tso, B., & Mather, P. M. (2009). *Classification methods for remotely sensed data* (second edi). CRC Press.

Vapnik, V. (2006). *Estimation of Dependences Based on Empirical Data*. Springer New York.

Vapnik, V. N. (1998). Statistical Learning Theory. *Adaptive and Learning Systems for Signal Processing, Communications and Control*, *2*, 1–740.

Waske, B., Benediktsson, J. A., Árnason, K., & Sveinsson, J. R. (2009). Mapping of hyperspectral AVIRIS data using machine-learning algorithms. *Canadian Journal of Remote Sensing*, *35*(SUPPL. 1). http://doi.org/10.5589/m09-018

Waske, B., & Braun, M. (2009). Classifier ensembles for land cover mapping using multitemporal SAR imagery. *ISPRS Journal of Photogrammetry and Remote Sensing*, *64*(5), 450–457.

Watts, J. D., Powell, S. L., Lawrence, R. L., & Hilker, T. (2011). Improved classification of conservation tillage adoption using high temporal and synthetic satellite imagery. *Remote Sensing of Environment*, *115*(1), 66–75.

Xie, H., Pierce, L. E., & Ulaby, F. T. (2002). SAR speckle reduction using wavelet denoising and Markov random field modeling. *IEEE Transactions on Geoscience and Remote Sensing*, *40*(10), 2196–2212.

Yoshida, T., Kawata, J., Tada, T., Ushida, a., & Morimoto, J. (2004). Edge detection method with CNN. *SICE 2004 Annual Conference*, *2*, 1721–1724.

Zambon, M., Lawrence, R., Bunn, A., & Powell, S. (2006). Effect of Alternative Splitting Rules on Image Processing Using Classification Tree Analysis. *Photogrammetric Engineering & Remote Sensing*, *72*(1), 25–30.

Zeiler, M., & Fergus, R. (2015). Visualizing and understanding Convolutional Networks), 1–11. *In: Fleet D., Pajdla T., Schiele B., Tuytelaars T. (eds) Computer Vision – ECCV 2014. ECCV 2014. Lecture Notes in Computer Science, vol 8689. Springer, Cham*

Zeiler, M. D., & Fergus, R. (2013). Stochastic Pooling for Regularization of Deep Convolutional Neural Networks. *International Conference on Representation Learning*, 1–9.

Zeiler, M. D., Krishnan, D., Taylor, G. W., & Fergus, R. (2010). Deconvolutional networks. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2528–2535.

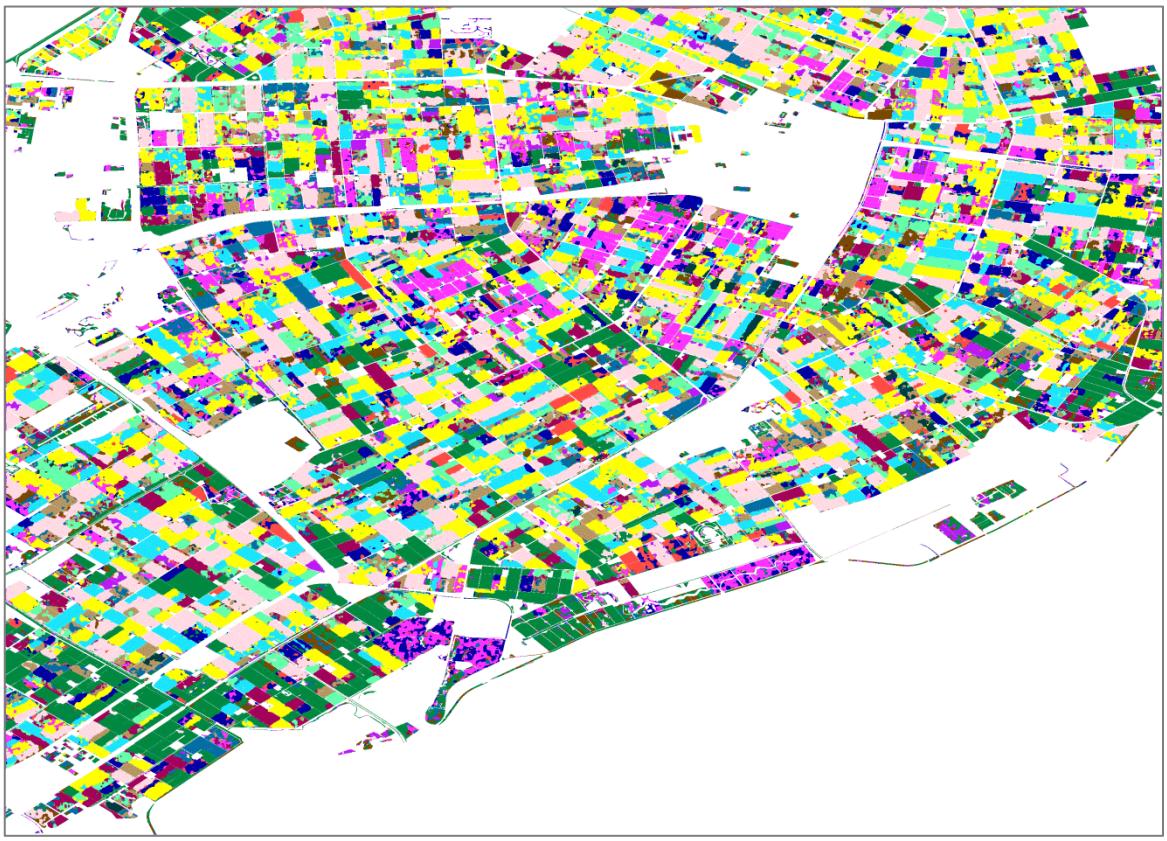
Zhang, J., Zhong, P., Chen, Y., & Li, S. (2013). For the Representation and Restoration of Optical Remote Sensing Images. *IEEE Transactions on Geoscience and Remote Sensing*, *52*(5), 1–11.

Zhao, J., Guo, W., Cui, S., Zhang, Z., & Yu, W. (2016). Convolutional Neural Network for Sar Image Classification At Patch Level. *IEEE International Geoscience and Remote Sensing Symposium*, (2015), 945–948.

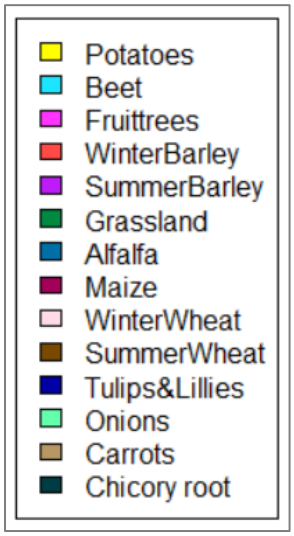
Zhou, Y., Wang, H., Xu, F., & Jin, Y. (2016). Polarimetric SAR Image Classification Using Deep Convolutional Neural Networks. *IEEE Geoscience and Remote Sensing Letters*, 1–5.

APPENDIX A

CLASSIFIED MAP FOR A LARGER SUBSET



(a)



(b)

Figure 8.1: CNN classified map for a large subset (a) (74714 training points), with the legend (b).