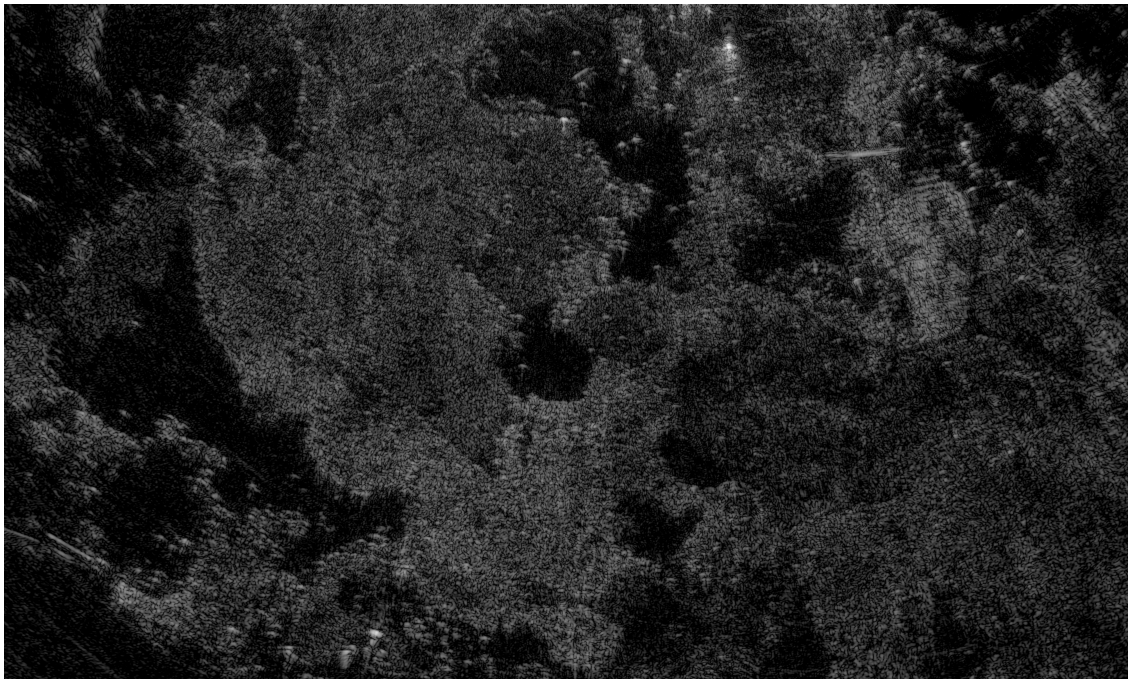


Reducing blur in SAR images using Deep Learning

Master's thesis in Computer Science – algorithms, languages and logic

VICTOR CHRISTOFFERSSON

JACOB LUNDBERG

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2019

Master's thesis 2019:NN

Reducing blur in SAR images using Deep Learning

Victor Christoffersson
Jacob Lundberg

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Signals and Systems
Chalmers University of Technology
Gothenburg, Sweden 2019

Reducing blur in SAR images using Deep Learning
Victor Christoffersson
Jacob Lundberg

Supervisor: Cristopher Zach, Electrical Engineering
Examiner: Cristopher Zach, Electrical Engineering

Master's Thesis 2019:NN
Department of Electrical Engineering
Signals and Systems
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: SAR-image created with a modified version of the RITSAR simulator [1].

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2019

Reducing blur in SAR images using Deep Learning
Victor Christoffersson
Jacob Lundberg
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Saab develops a system called CARABAS that is a Synthetic Aperture Radar (SAR). SAR relies on GPS equipment to correctly estimate the flight path during collection of SAR-data. If the flight path estimation is faulty, the processed SAR-image will contain blur and distorted targets. In order to remove the need for precise GPS technology, or to reduce errors, this thesis has investigated deep learning methods using convolutional neural networks to deblur processed SAR-images. Different architectures have been investigated along with different loss functions. Pixel-wise loss functions were tested against more complex losses like perceptual loss and adversarial models. The perceptual loss and adversarial models used in the thesis shows promising results on SAR-images created in a simulator, however when training on real SAR-images they perform worse. Real images contain patterns that are hard to classify as blurry or clear. The models were trained on crops where many of them contained a lot of these patterns that the networks struggled to learn. When using a sequential input to a recurrent network it was possible to improve the results, since this provided the network with higher-level information. The results suggests that deblurring SAR-images with deep learning is possible, however, further research is required for real images.

Keywords: SAR, deep learning, CNN, neural network, deblurring, adversarial model, perceptual loss, autoencoder, FCN.

Acknowledgements

We would like to express our gratitude to Anders Åhlander and Patrik Dammert at Saab AB for their guidance throughout the project, especially for their expertise regarding SAR. We would also like to thank the Innovation Lab team for their support and for providing us with necessary hardware. Lastly, we would like to thank our supervisor Christopher Zach at Chalmers University for his technical advice and suggestions regarding evaluation.

Victor Christoffersson, Jacob Lundberg, Gothenburg, June 2019

Contents

List of Figures

List of Tables

1
Introduction

This chapter provides a brief background to SAR and the problems that can occur when creating a SAR-image. This is followed by how the thesis aims to solve them and as well as a description of its delimitations.

1.1 Background

A SAR is mounted on a moving platform such as an aircraft to create high resolution images of the ground. The SAR-system collects data while flying over an area for a certain distance and then processes the data as if it came from a physically long antenna. In this way a long antenna aperture is synthetically created, and a high resolution image can be obtained. Saab AB develops a SAR system called CARABAS. The system, which operates in the VHF band, can be mounted on various aircrafts, from drones to airplanes.

The applications of SAR-images ranges from mapping surfaces of different planets to geology and oceanography on Earth. SAR can also be used to inspect forest biomass, deforestation, monitoring volcanoes and earthquakes, environmental monitoring such as oil spills, flooding and military surveillance. Many of these applications requires sharp images and blurry SAR-images can thus reduce the usability and leave out important details.

During SAR-image formation, the flight path is saved while collecting data from the ground. Since it is impossible to fly in a straight line, the flight path has to be estimated using GPS-technology. The algorithm is sensitive to errors in the flight path, therefore precise GPS-technology has to be used, making the equipment expensive. If the flight path supplied to the algorithm is not correctly estimated according to the actual flight path, positional or velocity errors can occur. Eliminating the need for an expensive GPS therefore reduces both the cost of the system and the need to rely on estimated information.

1.2 Problem description

Errors and uncertainties in the compensation for a non-linear flight path result in blurry SAR-images. Reducing the blur in the image increases the amount of visible details that an observer can distinguish. With the help of deep learning, the dependency on expensive GPS-technology could be reduced. Deep learning has shown

great promise blur reduction on normal images [2], [3]. This thesis aims to investigate if this is possible with SAR-images and to find a suitable architecture for the task.

The number of SAR-images captured with the CARABAS hardware is limited and considering the large amount of data required for supervised deep learning have to be addressed. The methods in this thesis will therefore first be evaluated on simulated SAR-images. These will be created with a simulator trying that resembles the CARABAS hardware. If the models are able to restore simulated images, they models will then be further developed to work with actual SAR-images provided by CARABAS.

1.3 Delimitations

This thesis will not evaluate methods for performing auto-focus during the creation of a SAR-image, but rather work on SAR-images that have been created with errors. Algorithms for SAR-image creation exists and work whenever the correct input is provided. The investigation aims to only improve SAR-images created with CARABAS hardware, hence other hardware will not be considered. This is due to the varying transmitting frequencies for different SAR systems.

SAR-images are represented by complex valued numbers. Although there is some research on how to handle this, it is not common input for neural networks [4]. Therefore, this will not be investigated in the thesis. Another approach is to split the complex numbers into two channels, one real and one imaginary [5]. This method will instead be used to handle all complex numbers as input.

1.4 Thesis outline

The thesis is divided into 6 chapters. In chapter 2, necessary theory is explained to understand the methods used in the report. The chapter also present previous research and anchors the thesis relevance and foundation.

Chapter 3 explains the methods used and motivates their choice. The different network architectures, loss functions as well as evaluation metrics are based on the theory in chapter 2.

The results are presented in chapter 4. The different models are evaluated according to different metrics. Examples predictions are also shown to visually be able to judge the quality of the methods.

In chapter 5 the strengths and weaknesses of the different method choices are analysed. The discussions are based on comparison of the results. The chapter also proposes what more can be done on the subject in the future. Finally, the conclusions of the results and discussions are presented in chapter 6.

1.5 Scientific contribution

This thesis introduces a new method to reduce blur in SAR-images. Currently there exists few algorithms that are able to successfully reduce blur in SAR-images. Processing SAR-data is usually time-intensive, however, a trained neural network can process the input in seconds. The thesis provides a fundamental baseline to continue further research within the SAR and deep learning field.

2
Theory

The first part of this chapter, section 2.1, consists of basic radar and SAR theory. The second and third part, section 2.2 and 2.3, serves as an introduction to artificial neural networks, deep learning, and other fundamental concepts to understand the methods used in the thesis.

2.1 Radar

Radar is an abbreviation that stands for Radio Detection and Ranging. Radar use electromagnetic waves to detect and determine distance from objects. The reflected signal from the source can be used to detect an object. The distance can be calculated with the time between transmission and reception from the signal. The received energy from the reflected signal is defined by the fundamental radar equation:

$$E = \frac{P_m G \sigma A_e t_{ot}}{(4\pi)^2 R^4 D} \tag{2.1}$$

where E is the energy of the received signal, P_m is transmitted average power, G antenna gain, the targets radar cross section σ, effective antenna area A_e and time on target t_{ot}. In the denominator, the range to the target R and losses D. It can be noted from the equation that the received energy decrease by a large factor with the distance [6].

2.1.1 Frequencies

For radar systems, transmitted frequencies determine its purpose. Most airborne radars transmit frequencies between 1-100 GHz. Laser radars normally transmits frequencies around 100 THz. CARABAS transmits electromagnetic waves in the low Very High Frequency (VHF) band, frequencies in the range 20-90 MHz, unlike most Synthetic Aparture Radar (SAR) systems that transmits in higher frequencies [7]. This is lower than the Ultra High Frequency (UHF) band and higher than the High Frequency band (HF). Lower frequency radars require large and heavy hardware and the opposite is true for high frequency radars [6].

2.1.2 Synthetic Aperture Radar

Since SAR transmits signals in the low VHF band this enables penetration through foliage and biomass. In order to obtain a high resolution, an unrealistically long

antenna is required. This is solved by synthetically creating the required aperture length. The antenna will then act as if it was physically longer. The synthetic antenna can be created by flying in a certain pattern along a selected area, where stripmap mode is common practice. Figure 2.1 shows an illustration of the stripmap pattern. During the flight, radar data is collected from different time intervals by sending pulses towards the ground. A complex valued SAR-image can then be processed from the collected radar bins in either the time or frequency domain. The frequency domain assumes a linear flight track, hence the time domain is the appropriate choice for SAR since a linear flight path is physically impossible [7].

In order to create a focused image, the distance from the source to the reflection point at each pulse has to be calculated with precision. The position can be estimated with GPS that tracks the position of the aircraft when the signal was transmitted. The precision of the distance will affect resulting image quality [8]. The distance between the source and the reflection has a small error margin. The maximum margin value is determined by the azimuth resolution. The azimuth resolution is the smallest distance between two points that can be distinguished on the ground. This resolution is approximately equal to half the length of the physical antenna. A longer antenna allows for a larger distance error. In conclusion, the distance error margin depends upon the equation:

$$\Delta L = \frac{\delta a}{4} \tag{2.2}$$

where δa is the azimuth resolution. The azimuth resolution can be calculated with the equation:

$$\delta a = \frac{\lambda_c}{2(v_2 - v_1)} * \frac{c}{2B} \tag{2.3}$$

where λ_c is the wavelength, B is the bandwidth and $(v_2 - v_1)$ is the angle between start and end as seen from the ground to the aircraft, also known as the integration angle.

Once the radar data has been collected along the flight path, it is gathered in an NxM matrix. The complex valued SAR-image can then be calculated with the backprojection algorithm from the radar data. Finally, the SAR-image can be calculated with the 10-logarithm of the magnitude to display for the human eye [7].

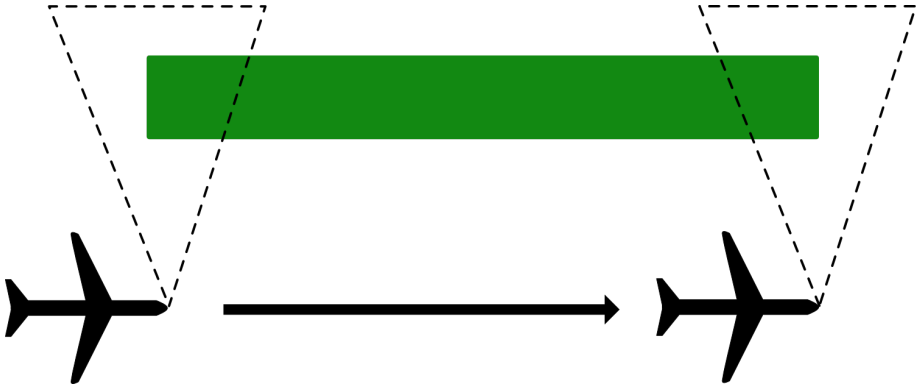


Figure 2.1: SAR in stripmap mode.

2.2 Artificial Neural Networks

Artificial Neural Networks (ANN) are inspired by the functions of the human brain. The human brain has several million networks consisting of interconnected neurons. The brain sends electrical signals to learn and process information. ANNs are structured in a similar way, an input is passed trough neurons to be processed and output a result. An ANN can be trained to solve complex problems since it can contain a large amount of neurons and connections to estimate functions [9].

2.2.1 Feedforward Neural Networks

A feedforward neural network, or multilayer perceptron (MLP), is a subclass of ANNs that consists of at least three layers, an input layer which does not modify the input, a hidden layer that performs some operation on the input, and lastly an output layer. The name feedforward comes from that information only flows in one direction. The amount of hidden layers can be changed to an arbitrary number to fit the intended purpose. The layers in turn can consists of an arbitrary amount of neurons. Each neuron has a connection to all neurons in the previous layer and takes the output of those neurons as input. These inputs are multiplied with the neurons weights to calculate the activation:

$$z_{i,j} = \sigma(\boldsymbol{w_{i,j}} \boldsymbol{z_{i-1}} + b_{i,j}) \tag{2.4}$$

where $z_{i,j}$ is the output of neuron j in layer l_i and $\boldsymbol{z_{i-1}}$ are the outputs from layer l_{i-1}. $\boldsymbol{w_{i,j}}$ is the weight vector and $b_{i,j}$ the bias vector, both unique to each neuron. σ is called an activation function, which can introduce non-linearities to the network depending on the function. Examples of these are introduced in section 2.2.5 The calculation can be written for each layer as:

$$\boldsymbol{z}_i = \sigma(\boldsymbol{W}_i^T \boldsymbol{z}_{i-1} + \boldsymbol{b_i}) \tag{2.5}$$

Where $\boldsymbol{z_i}$ contains $z_{i,j}$ for all neurons, $\boldsymbol{W_i^T}$ is a matrix containing all weighs vectors $\boldsymbol{w_{i,j}}$, and $\boldsymbol{b_i}$ contains all biases $b_{i,j}$, for layer l_i [10].

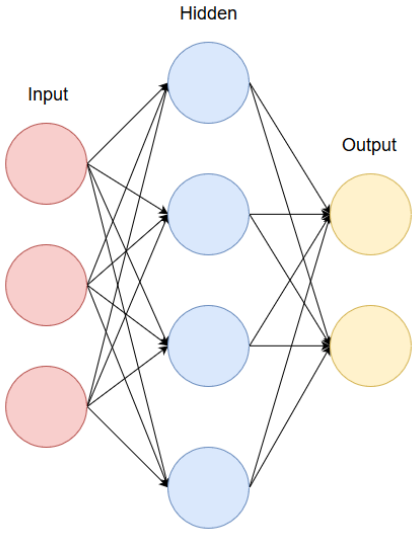


Figure 2.2: A fully connected Neural Network with one hidden layer.

2.2.2 Training a network

The weights and biases in the network are adjusted during the training process. To train the network, input $\boldsymbol{x}$ is first propagated forward through the entire network. The network produces output $\hat{y} = z_N$ that can be one or several scalar values. The aim of the network is to produce a prediction that minimises an error. The error is usually referred to as a cost or loss function, which outputs a scalar value and is a function chosen for the task.

The network does this with the help of the backpropagation algorithm [11]. After the forward pass through the network, the gradient of the loss $\mathcal{L}$ with respect to all weights and biases are calculated. When this is calculated the weights and biases can be updated in steps, according to gradient descent:

$$w_{i,j}(t+1) \longleftarrow w_{i,j}(t) - \lambda \frac{\partial \mathcal{L}(t)}{\partial w_{i,j}} \tag{2.6}$$

where t is the current state and $t+1$ the next state. λ is the learning rate that determines the step size. This is often not enough to guarantee an optimal solution. Other variations of gradient descent introduces a stochastic element to help getting to a minimum. One of them is the Adam optimiser [12]. Adam replaces the standard gradient descent update with the following calculations:

$$\begin{aligned} \boldsymbol{m_t} &= \beta_1 \boldsymbol{m}_{t-1} + (1-\beta_1)\nabla_\theta \mathcal{L}(t) \\ \boldsymbol{v}_t &= \beta_2 \boldsymbol{v}_{t-1} + (1-\beta_2)\left(\nabla_\theta \mathcal{L}(t)\right)^2 \\ \hat{\boldsymbol{m}}_t &= \boldsymbol{m}_t \,/\, (1-\beta_1^t) \\ \hat{\boldsymbol{v}}_t &= \boldsymbol{v}_t \,/\, (1-\beta_2^t) \\ \boldsymbol{W}_t &= \boldsymbol{W}_{t-1} - \lambda \hat{\boldsymbol{m}}_t \,/\, \left(\sqrt{\hat{\boldsymbol{v}}_t} + \epsilon\right) \end{aligned} \tag{2.7}$$

where $\boldsymbol{m}_t$ is an estimate of the mean and $\boldsymbol{v}_t$ an uncentered estimate of the variance for the gradients over time. β_1 and β_2 controls their decay rates respectively. $\hat{\boldsymbol{m}}_t$

and $\hat{\boldsymbol{v}}_t$ are the bias corrected versions of $\boldsymbol{m}_t$ and $\boldsymbol{v}_t$ because of initialisation to 0. ϵ is a small term to avoid division by 0. When $\hat{\boldsymbol{m}}_t / \sqrt{\hat{\boldsymbol{v}}_t}$ is small the stepsize will be close to 0, this is works well since the ratio is usually small when the gradient is close to an optimum. The parameters are calculated elementwise and thus allows Adam to have an adaptive learning rate for each parameter of the network [12].

2.2.3 Supervised learning

Supervised learning is a category in machine learning where the dataset is paired with some target. The desired output of the network is thus specified and the networks learns a mapping from the input space to the target space. The cost function can thus be based upon some distance between the networks prediction and the target. This leads to a well defined cost function to minimise [9].

2.2.4 Dropout

During training of a model, there is a risk that the model will be overfitting. This occurs when a model fits too well to a particular set of data. In neural networks it usually means that the accuracy of the network is better on seen than unseen data from the same dataset. To prevent this problem it is possible to use regularisation techniques like dropout. A dropout layer allows for a more dynamic network architecture by adding a probability that some nodes are excluded from the network during training. The optimal probability value for most networks and datasets is usually 0.5, thus a 50% chance that a node will be excluded [13].

2.2.5 Activation Functions

Activation functions are essential for neural networks. It enables complex non-linear mappings between input and output. The output from the activation function is then passed on to the next layer.

2.2.5.1 Sigmoid functions

The sigmoid function is a non-linear function that is derivable and defined for all real values which makes it well suited for neural networks. The name sigmoid usually refers to the logistic function that is calculated according to:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.8}$$

where large negative values are mapped close to 0 and large positive values are mapped close to 1. Sigmoid is often used to calculate probabilities like for binary classification where it is predicted whether the input belongs to class 0 or 1 [14].

The hyperbolic tangent (tanh) is a shifted and rescaled version of the sigmoid activation function. Unlike sigmoid it instead maps the input to the range -1 to 1 and thus allows for a negative output. It is defined by:

$$f(x) = \frac{2}{1 + e^{-2x}} - 1 \tag{2.9}$$

where large negative values are mapped close to -1 and large positive values close to 1 [9].

2.2.5.2 Rectified Linear Units

Rectified Linear Units (ReLU) is a common activation function for hidden layers and has been shown to outperform sigmoid and tanh in deep networks. The function sets a negative input to zero and can be written as:

$$f(x) = max(0, x) \tag{2.10}$$

ReLU efficiently accounts for non-linear effects, where non-linear effects are outputs that depend upon previous outputs [15].

A variation of ReLU is the leaky ReLU (LReLU). LReLU is calculated according to:

$$f(x) = \begin{cases} x & : \ if \ x > 0 \\ \alpha x & : \ otherwise \end{cases} \tag{2.11}$$

where α usually is a small number between 0 and 1. It allows for a small gradient when the value is lower than zero [16].

2.2.6 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are feedforward neural networks containing convolutional layers. These layers differs from standard fully connected layers since they use convolutions instead of standard matrix multiplications [9]. A convolution is a mathematical operation that takes a weighted average over an arbitrary input. In terms of images, the input is an image in the form of a tensor, where every pixel is one element. The weights are in the form of a tensor typically referred to as kernel. The kernel enables the network to catch spatial and temporal dependencies in the input, something fully connected layers can not. The convolution is calculated according to:

$$CONV(i, j) = (K * I)(i.j) = \sum_m \sum_n I(m, n)K(i - m, j - n) \tag{2.12}$$

with kernel K and image I. Like in standard neural networks, a kernel with the same size as the image could quickly cause the number of parameters in a network to explode. For instance, an 100x100 image would need 10000 weights for each neuron. This is solved by choosing a smaller kernel and traversing the image, thus reusing weights more than once and reducing the load. How much the kernel moves for every step is called stride, which also determines the output size along with the kernel size. The kernel size also defines the receptive field for each neuron, an $n \times m$ area in contrast to every input element in fully connected layers. The output for each

convolution is often referred to as a filter. It is possible to use several convolutions with different kernels in each layer and thus construct a feature map consisting of several filters. Different features can thus be detected in the input at every layer [9].

2.2.6.1 Pooling layers

Pooling layers are a common occurrence in CNNs. Pooling layers modifies the output of a layer which in turn enables the network be more invariant to translation [17]. This means that if features in an input image is for example rotated or moved, the pooling layer can still output the same values unchanged by the movement or rotation. Pooling layers outputs a summary of neighbouring values and can therefore be used with a stride to downsample the output while still preserving dominant features. This has the benefit of reducing the amount of parameters in succeeding layers. One common pooling layer is max pooling, which outputs only the maximum value for each $n \times m$ region [9].

2.2.6.2 Upsampling layers

Much like pooling layers reduce the size of data, upsampling layers increases the size. The pooling operations are replaced by either interpolation or transposed convolutions.

Bilinear Interpolation Bilinear interpolation takes a weighted average of the nearest neighbours with known values to find the value of an unknown function at a point (x, y). Using the four neighbours $(i_1, j_1), (i_2, j_2), (i_3, j_3), (i_4, j_4)$, the problem can be written as $f(x, y) \approx a_0 + a_1 x + a_2 y + a_3 xy$. The solution is then found by calculating the coefficients by solving the linear equation system:

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 1 & i_1 & j_1 & i_1 j_1 \\ 1 & i_1 & j_1 & i_1 j_1 \\ 1 & i_1 & j_1 & i_1 j_1 \\ 1 & i_1 & j_1 & i_1 j_1 \end{bmatrix}^{-1} \begin{bmatrix} f(i_1, j_1) \\ f(i_2, j_2) \\ f(i_3, j_3) \\ f(i_4, j_4) \end{bmatrix} \tag{2.13}$$

It is then possible to increase the resolution of an image by calculating pixels from the lower resolution image [18].

Transposed Convolutions In interpolation, the weights are predefined but it is also possible to upsample with learnable parameters. This is called transposed convolutions, deconvolutions or fractionally strided convolutions. Transposed convolution is the inverse of a standard convolution but instead of being a many-to-one operation it is instead one-to-many. The kernel traverses the image but by adding a zero padding it is possible to upsample the image [19].

2.2.6.3 Fully Convolutional Network

Fully Convolutional Network (FCN) is a type of architecture used in image-to-image translation tasks. FCN was originally developed for image segmentation [20], but has also proven useful for other tasks such as object tracking [21] and deblurring

of images [2], [3]. An FCN contains no fully connected layers and work with pixel-wise predictions and thus the output is an image rather than a label, making an FCN a regression network. An FCN uses the basic components of convolutional networks, convolutional, activation, pooling, and upsampling layers. All layers have spatial output of some size, $height \times width \times channels$. The target for an FCN loss function is then usually a reference image.

2.2.7 Residual connections

Residual connections or skip connections is often used in deeper networks. The concept, first introduced by [22], were to reduce the training error experienced by adding more layers to a model already suitable for a problem. The idea of residual connections is to make the deeper network perform at least as well as the more shallow network in learning a mapping. If the desired mapping is denoted as $H(x)$, and the layers are fit to another mapping $F(x) = H(x) - x$, the original mapping becomes $H(x) = F(x) + x$. With residual connections x should at least become an identity mapping from the input of a block to its output. Then if $F(x) = 0$ the network will at least preserve the input x and should theoretically do no worse than without the residual connections. In networks with deeper architectures they have proven to work well [2], [3], [22], [23].

2.2.8 Autoencoders

Autoencoders, or encoder-decoders are networks trained to learn a representation of a dataset. The input is first run through the encoder and then passed on to the decoder. The encoder receives the input and encodes it by changing its representation, images are scaled to a lower resolution, however the output can contain multiple feature maps. The output of the encoder is then passed on to the decoder that attempts to recover the information to its original format, thus performing some form of upsampling [9].

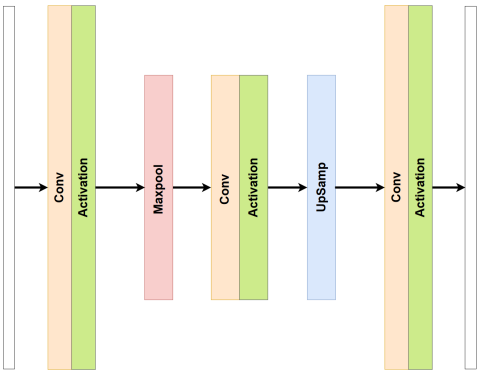


Figure 2.3: A simple autoencoder where the input is downsampled once and then upsampled.

2.2.9 Recurrent neural networks

Recurrent neural networks (RNNs) are used for processing sequential data. In an RNN parameters can be shared between different parts of the model and this allows for the usage of past information for future predictions. An RNN can be seen a sequence of hidden states starting with the initial state $h^{(0)}$. The following states are then calculated by updating the following equations for each time step t:

$$\begin{aligned} \boldsymbol{h}^{(t)} &= \sigma_h(\boldsymbol{W}_h\boldsymbol{x}^{(t)} + \boldsymbol{U}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}_h) \\ \boldsymbol{o}^{(t)} &= \sigma_o(\boldsymbol{W}_o\boldsymbol{h}^{(t)} + \boldsymbol{b}_o) \end{aligned} \tag{2.14}$$

$\boldsymbol{U}, \boldsymbol{V}$, and $\boldsymbol{W}$ are shared weights matrices, $\boldsymbol{b}$, and $\boldsymbol{c}$ shared bias vectors and σ_h, and σ_o activation functions. $\boldsymbol{o}^{(t)}$ is the output at time step t. The hidden state $\boldsymbol{h}^t$ is passed to the next unit in the RNN, thus propagating information through time. The total loss of the network is the sum of the loss for each output according to $\mathcal{L} = \sum_t \mathcal{L}^{(t)}$ [9]. The gradients can then be calculated with the backpropagation through time (BPTT) algorithm, a generalisation of the standard backprogation algorithm, and the parameters are updated accordingly [24].

A problem with RNN is exploding and vanishing gradients which makes it hard to train [25]. Exploding gradients are when the magnitude of long term gradients grow exponentially compared to short term ones and can arise during training when weights are repeatedly multiplied with values larger than 1. Vanishing gradients are in turn when the magnitude of long term gradients goes to zero resulting from weights being repeatedly multiplied with values lower than 1 [26].

2.2.9.1 Long Short-Term Memory

To resolve the problems with exploding and vanishing gradients it is possible to use a variation of RNN called Long Short-Term Memory (LSTM) [27]. LSTM is an RNN with gates which makes it possible to control how to use input and memory to update the next state. The gates themselves are learnable parameters and the network can thus control how much of the previous information it should use in the next state. The standard RNN units are replaced by LSTM units that contains a cell, an input gate, an output gate, and a forget gate [28]. The cell part is the memory unit of the network and has a self-loop. The input gate decides the flow if information in to the cell, the output gate decides the flow of information out from the cell, and the forget gate decides how much information that remains in the cell. The states of the LSTM are calculated according to:

$$\begin{aligned} \boldsymbol{f}^{(t)} &= \sigma_g(\boldsymbol{W}_f\boldsymbol{x}^{(t)} + \boldsymbol{U}_f\boldsymbol{h}^{(t-1)} + \boldsymbol{b}_f) \\ \boldsymbol{i}^{(t)} &= \sigma_g(\boldsymbol{W}_i\boldsymbol{x}^{(t)} + \boldsymbol{U}_i\boldsymbol{h}^{(t-1)} + \boldsymbol{b}_i) \\ \boldsymbol{o}^{(t)} &= \sigma_g(\boldsymbol{W}_o\boldsymbol{x}^{(t)} + \boldsymbol{U}_o\boldsymbol{h}^{(t-1)} + \boldsymbol{b}_o) \\ \boldsymbol{c}^{(t)} &= \boldsymbol{f}^{(t)} \circ \boldsymbol{c}^{(t-1)} + i^{(t)} \circ \sigma_c(\boldsymbol{W}_c\boldsymbol{x}^{(t)} + \boldsymbol{U}_c\boldsymbol{h}^{(t-1)} + \boldsymbol{b}_c) \\ \boldsymbol{h}^{(t)} &= \boldsymbol{o}^{(t)} \circ \sigma_h(\boldsymbol{c}_t) \end{aligned} \tag{2.15}$$

where $\boldsymbol{f}^{(t)}$ is the forget gate, $\boldsymbol{i}^{(t)}$ the input gate, $\boldsymbol{o}^{(t)}$ the input gate, $\boldsymbol{c}^{(t)}$ the cell state, $\boldsymbol{h}^{(t)}$ the hidden state. σ_g, σ_c and σ_h are activation functions and $\circ$ is the hadamard

product [9]. The control of information flow leads to a network that is able to have a longer memory without exploding and vanishing gradients [27].

2.3 Loss Functions

Loss functions are necessary to measure the error of a network. This measure can then be used to optimise the network parameters and thus reduce the error.

2.3.1 Pixel-wise Loss

Mean Square Error A commonly used loss function in deep learning is the Mean Square Error (MSE). It is calculated as:

$$\mathcal{L}_{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 \tag{2.16}$$

The difference between the true values $y_i \in Y$ and the predicted values $\hat{y}_i \in \hat{Y}$ are squared and then averaged over the whole set n to compute the final loss [9].

Huber Another regression loss is Huber. It is defined as:

$$\mathcal{L}_H = \begin{cases} \frac{1}{2}(\hat{y} - y)^2 & \text{if } |\hat{y} - y| \leq \delta \\ \delta|\hat{y} - y| - \frac{1}{2}\delta^2 & \text{otherwise} \end{cases} \tag{2.17}$$

which makes it linear for large values of $|\hat{y} - y|$ and quadratic for small values of $|\hat{y} - y|$. By combining properties from MSE loss with absolute error Huber becomes less sensitive to outliers than MSE [29].

2.3.2 Binary cross-entropy

The binary cross-entropy loss is used for measuring the error of predictions, usually used for binary classification. It is calculated according to:

$$\mathcal{L}_{BCE} = -\frac{1}{n}\sum_{i=1}^{n}(y_i(\log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)) \tag{2.18}$$

Where y_i is the true label and $\hat{y}_i$ the predicted label [30].

2.3.3 Perceptual Loss

Perceptual loss is similar to MSE but instead of calculating the loss for each pixel directly between output and target, perceptual loss instead calculates the loss by measuring the differences of feature maps. These feature maps are generated from a layer in a pre-trained convolutional network in order to catch semantic differences between images. This yields a loss with the aim to create an image with similar

features to the the original image. Using a feature map from an early layer makes the loss look at low-level features, while later layers catches more high-level features. The equation for the loss is as follows:

$$\mathcal{L}_P = \frac{1}{N * M} \sum_{n=1}^{N} \sum_{m=1}^{M} (f(y)_{n,m} - f(\hat{y})_{n,m})^2 \tag{2.19}$$

Where $N * M$ is the dimension of the feature map f obtained from convolution of the input image. The feature map is applied to both the sharp image y as well as the predicted image $\hat{y}$ [31].

2.3.4 Adversarial loss

Networks with adversarial loss has shown good results for deblurring images [2], [3], [32]. The idea of the discriminator is to evaluate the results of the regressor. This ends up as a system of two networks, a regressor and a discriminator with different objectives. The regressor's objective is to take samples from an input space and produce samples in some output space. The discriminator's objective however is to distinguish the output of the regressor from real samples. This creates an adversarial loss with the objective in the form of a min-max function:

$$\min_R \max_D \mathbb{E}_{y\ p_y(x)}[\log D(y)] + \mathbb{E}_{x\ p_x(x)}[\log(1 - D(R(x)))] \tag{2.20}$$

The regressor $\mathcal{R}$ is fed samples from some distribution P_x. It then tries to create a mapping from P_x as close as possible to the real samples in the distribution P_y. The discriminator $\mathcal{D}$ is fed samples both from the real samples as well as the fake samples generated by $\mathcal{R}$. The aim of $\mathcal{D}$ is to distinguish real samples from fake samples and providing feedback to $\mathcal{R}$. In turn $\mathcal{R}$ attempts to get better at generating realistic samples, and is thus trying to maximise the error of $\mathcal{D}$. Training is done by applying backpropagation to both networks in turns by first training $\mathcal{D}$ to apply the correct labels to samples and then $\mathcal{R}$ is trained to minimise $\log(1 - \mathcal{D}(\mathcal{R}(x))$. This is then repeated iteratively until a desired state is achieved. In theory $\mathcal{D}$ should be trained to optimality for each iteration, but this is not always feasible and $\mathcal{D}$ can be trained for some chosen amount of steps. $\mathcal{R}$ is optimal when it has created a mapping that transforms x_i into y_i. This is equivalent to $\mathcal{D}$ predicting 0.5 for all inputs [33].

3
Methods

The content of the real and generated data and how to process it is explained in this chapter. The different methods, network architectures, loss functions, evaluation metrics and the training procedure is also featured below. Implementation specific details can be seen in appendix B

3.1 Data

Two different datasets were used during the thesis, a simulated and a set with real images. Detailed properties of the different datasets are specified below.

3.1.1 Artificial SAR-images

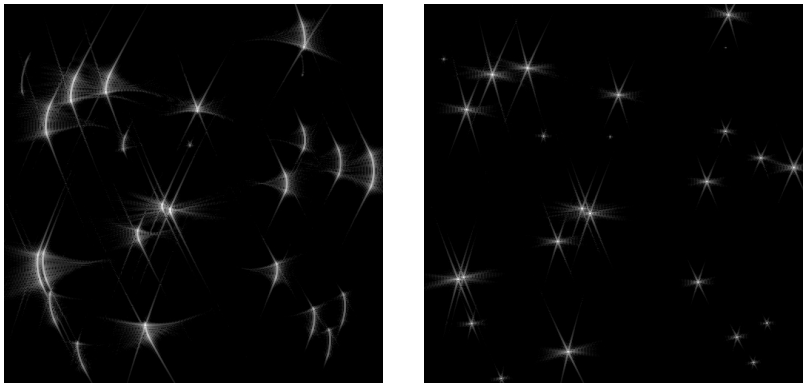
Artificial SAR-data was generated with the open source SAR-simulator RITSAR [1]. The final simulated dataset contained 8800 256x256 pairs of complex valued SAR-images. The software was modified to scatter points across the field by randomising x and y coordinates. These points were also created with randomised amplitudes between 0 and 1. The amount of points varied randomly between 200 and 500. All parameters were randomised in order to increase the variation within the dataset. The simulator generates data by emulating a physical SAR platform. The simulator offers two default platforms where both are modifiable. For the thesis, a default platform that transmits within the UHF band was used. This platform transmits frequencies similar to SAAB's CARABAS system and hence produce similar data. This platform can then used to generate a SAR-image that captures the scattered points. The resulting SAR-image is calculated with a signal processing chain (global backprojection) from the collected radar data. The resulting SAR-image is a complex valued matrix with dimensions 256x256. A logarithmic magnitude image can finally be calculated from the SAR-image with the equation:

$$I = \frac{10 * \log_{10} |p|}{max(|p|)} \tag{3.1}$$

where I is the resulting image and p is the complex valued SAR-image.

The collected radar data was generated by simulating movement along the y-axis with a velocity of 25m/s. The same radar data could then be used together with a modified UHF-platform that moved along the y-axis with a random speed in the

interval 20-30m/s selected uniformly. Two different SAR-images could then be produced for each platform from the same radar data, one blurry and one focused image. As a consequence, the platform with a random velocity results in a blurry SAR-image since it is out of sync with the generated radar data. Figure 3.1 shows examples of the resulting logarithmic magnitude images calculated from the SAR-image where 3.1a had incorrect velocity and 3.1b had the correct velocity.



(a) Blurry logarithmic SAR-image

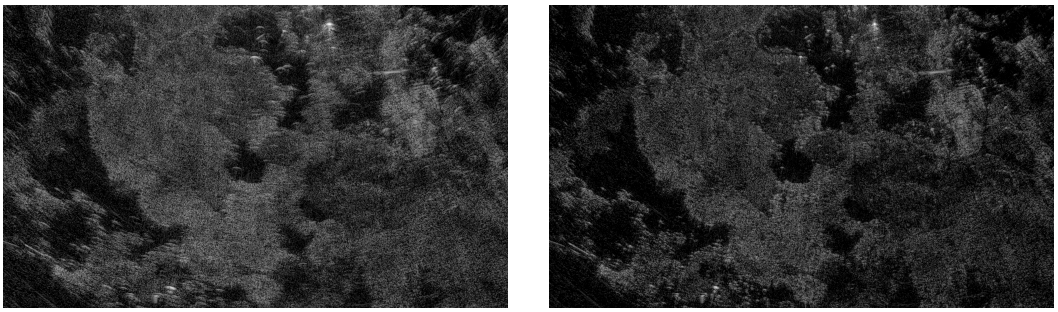
(b) Focused logarithmic SAR-image

Figure 3.1: Examples from the simulated dataset where every sample have a focused and a blurry version. The logarithmic magnitude for the images were calculated with equation 3.1.

3.1.2 Real SAR-images

The real dataset captured with CARABAS contains 11 different terrains. Each terrain consists of one image without a velocity error and 40 images with increasingly large velocity errors. All errors were added during the signal processing and not captured during the actual flight. A software available at Saab added errors to the recorded positions according to a specified velocity before the images were processed.

The images originally have dimensions 4001x4001. After inspecting all 11 terrains, the space where the aircraft was located during the pass was cropped out, resulting in a new dimension ranging between 2401-2601x4001 pixels. This was necessary since this area can not be properly calculated and was therefore excluded in the training set. Despite the reduced dimensions, the resulting images was too memory intensive for the available hardware and had to be cropped. The crops were taken randomly over the image of size 256x256 or 1024x1024 depending on the network. Additionally to prevent overload of the GPU memory, it also increased the variation in the dataset.



(a) SAR-image captured with CRABAS, processed with a velocity error

(b) Focused SAR-image captured with CARABAS

Figure 3.2: Example images from the training set captured with CARABAS. The blurry image was artificially blurred by adding a velocity error during the signal processing. The images can be seen in a higher resolution in appendix A.

3.2 Network architectures

Since the application of SAR-images for deep learning is a fairly unexplored area, several architectures had to be investigated. The regression models are all FCNs and autoencoders. The U-Net architecture is intended to work with smaller datasets and was therefore considered a suitable candidate for the real dataset. The other selected architectures combines some of the most successful techniques for blind image deblurring and other image-to-image translation tasks.

3.2.1 U-Net

The U-Net architecture was evaluated to deblur SAR-images. U-Net is an FCN originally used for image segmentation [23]. The architecture is inspired by the FCN in [20]. Due to data limitations in the biomedical field it was intended to work with smaller datasets which could make it suitable for the limited dataset of CARABAS SAR-images. The first half of the network consists of four blocks with two 3x3 convolutional layers with $64*2^n$ filters, where n is the block number starting from 0, and ReLU activation functions. For this thesis however, a filter size of $32*2^n$ was selected due to limited hardware. Each block is followed by a 2x2 max pooling layer that downsamples the output, halving the width and height. The second half of the U-Net is symmetrical to the first half but replaces the max pooling layers with upsampling layers symmetrical in size, thus doubling the width and height each time. Output from the high resolution filters from the early layers are copied and concatenated with the upsampled filters of the same size in the later layers. This is done by residual connections and allows propagation of high resolution context information. The last layer of the U-Net has a softmax activation function that outputs an image with the same size as the input. The loss function used in the original U-Net is a cross entropy pixel-wise loss. For this thesis the final output layer is changed to a linear output since it is not a probability.

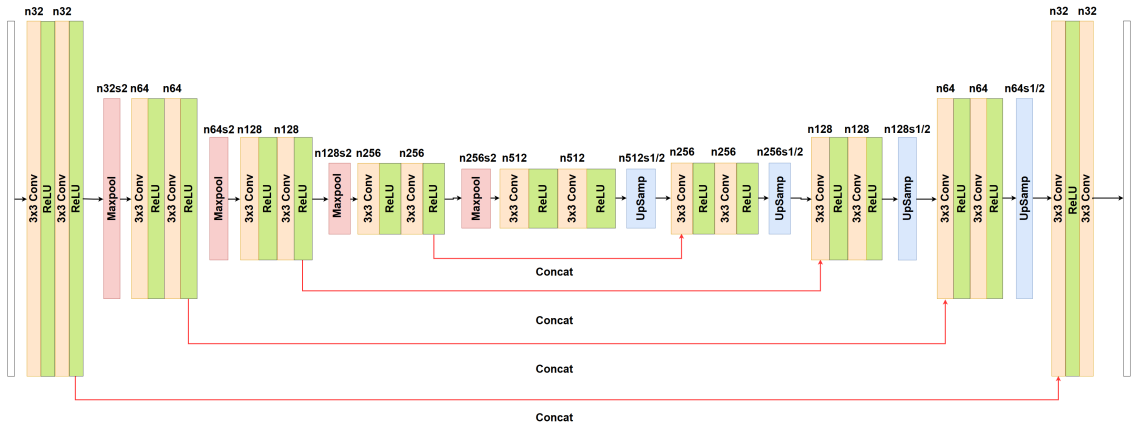


Figure 3.3: Architecture of the U-Net used in this thesis. The filter size is reduced and the output layer is changed to linear, aside from the original paper.

3.2.2 Deblur regressor

The deblur regressor is derived from the DeblurGAN generator, created with the purpose to perform blind image deblurring [2]. The first part of the regressor consists of a convolutional layer with 64 filters followed by two convolutional downsampling layers with 128 and 256 filters respectively. After the downsampling layers follows residual blocks, with two convolutional layers with 256 filters each. These blocks combine their input with their output with residual connections. The amount of residual blocks can be specified since they do not change the shape of their input. Seven blocks were used for all training, a smaller model than proposed in the original paper, since adding more residual blocks did not improve the performance. The first and the last convolutional layers use a kernel size of 7x7, the other convolutional layers have the kernel size 3x3. All convolutional layers are followed by ReLU activation functions, except the second layers in the residual blocks, and batch normalisation. Sigmoid was used as activation function in the final output layer.

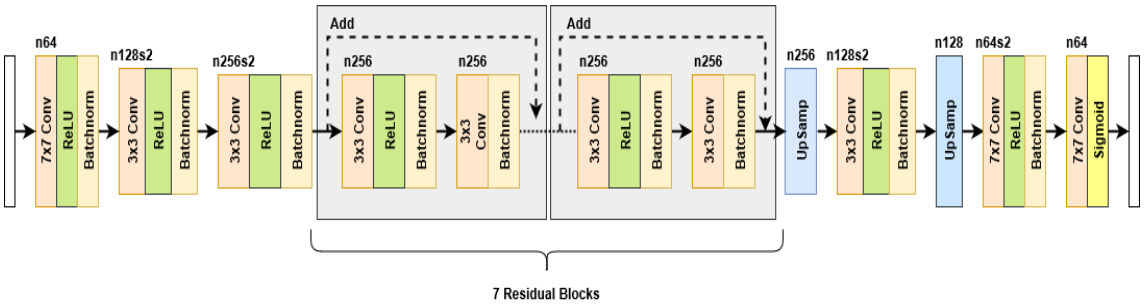


Figure 3.4: Architecture of the deblur regressor.

3.2.3 Effnet

Effnet is a convolutional neural network structure intended to be lightweight and efficient yet accurate [34]. Effnet consists of so called Effnet blocks. Each block

consists of a 1x3 depth-wise convolutional layer followed by 1x2 max pooling and a 3x1 depth-wise convolutional layer. The two convolutional layers replaces one 3x3 convolutional layer, with the benefit of being able to have a max pooling layer in between to reduce the amount of parameters. At the end of each block is a 2x1 convolutional layer. Each block doubles the amount of channels starting with 32 channels in the first convolution. On top of the Effnet is a 10 node fully connected layer. All hidden layers use LeakyReLU as activation function. This is followed by an output layer with sigmoid as activation function, suitable for binary classification.

3.2.4 Discriminator network

The discriminator network consists of 4x4 convolutional layers with stride 1, followed by LeakyReLU activation functions. On top of the convolutional layers is a fully connected layer with 1024 nodes with a tanh activation function, followed by a an output layer with a single neuron. The discriminator is a binary classifier with a sigmoid output activation function, evaluating whether the image was generated by the regressor or if looks similar to the distribution of the clear SAR-images.

3.2.5 U-Net with LSTM layers

The thesis proposes a modified version of U-Net that contains convolutional LSTM layers. The kernel size is 3x3 and the filter size is 32, for each LSTM block. The input to the network is a 1024x1024 image which is split into 16 256x256 segments, that is, 4 rows with 4 images each. These segments are fed to all 4 convolutional LSTM layers, where each layer takes one row as input. The output from the layers are concatenated into a 1024x1024x32 tensor that is then provided as input to the U-Net. U-Net produces a 1024x1024 linear output that is the restored image. The U-Net in this case used transposed convolutional layers instead of bilinear upsampling.

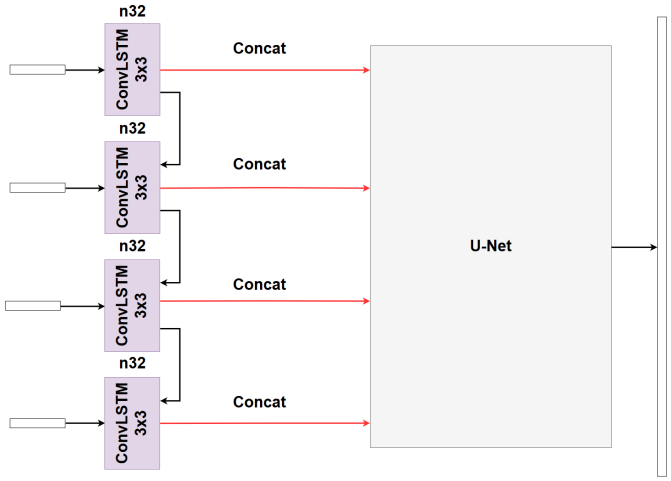


Figure 3.5: Architecture of the U-Net with convolutional LSTM layers.

3.3 Perceptual loss

Perceptual loss requires a network trained on relevant features for the dataset. To achieve this, part of the artificial dataset, 500 blurry and 500 clear images, was reserved for this task. The network was trained as a binary classifier between clear and blurry images. The architecture used for this task was the Effnet described in 3.2.3. The weights of the model that performed best on the validation set, a 10 percent subset from the full set, were selected. Feature maps from the first convolutional layer in the second convolutional block were extracted and used to calculate the perceptual loss.

3.4 Training

All training was done on a single GeForce RTX 2070 8GB GPU. The provided inputs were complex valued SAR-images supplied with logarithmic magnitude, derived from the complex values. It was possible to train on the logarithmic magnitude alone but when inspecting the images it was evident that information about target centers appeared outside the image. Early testing revealed that the networks struggled to capture these targets. When the complex values were supplied, the networks could capture these targets better and therefore the complex values were chosen as input. The complex values required processing since normally, a neural network is not able to handle imaginary numbers. This was solved by splitting the complex numbers in one real and one imaginary part, represented with real numbers of the same magnitude, and placing them in different input channels. The logarithmic magnitude was also supplied as a third channel to help the network in the initial phase of the training. The output of the network was a one channel logarithmic magnitude image.

All networks were first trained on the artificial dataset before being trained on the real dataset. This was necessary to ensure that the methods could successfully be evaluated independently of the limitation of the amount of SAR-images in the real dataset. U-Net and the deblur regressor were trained on simulated images with Huber, MSE and perceptual loss. Since the deblur regressor architecture was initially intended be used together with a discriminator, it was selected to investigate adversarial loss. Additionally, U-Net with LSTM layers were trained on real SAR-images to be able to draw further conclusions about the architectures.

3.4.1 Normalisation

The SAR-images were normalised before being fed to the network with the equation:

$$p_n = \frac{p}{max(|p|)} \tag{3.2}$$

where p_n is the resulting normalised SAR-image. The magnitude is calculated by:

$$m = \frac{|p_n|}{max(|p_n|)} \tag{3.3}$$

where m is the resulting magnitude matrix. The combined results from the input can be seen as a three channel tensor. The output from the network is the restored magnitude m_r, normalised.

3.5 Evaluation metrics

The models were evaluated with a few different evaluation metrics. The two metrics PSNR and SSIM are both full-reference methods, and compares an image to a baseline image, which in this case is the clear image. These metrics are often used to evaluate the performance of reconstruction algorithms and other deblurring methods [35], [36]. In addition to these metrics, differences between targets in the SAR-images has also been used.

3.5.1 PSNR

Peak Signal-to-Noise Ratio (PSNR) measures the ratio between the maximum possible value and the value of the noise that affects the quality. The signal is then the original image and the noise is the error of a test image compared to a reference image. The metric is calculated according to:

$$\mathcal{M}_{PSNR}(y, x) = 10 * \log_{10}\left(\frac{MAX_p^2}{MSE(x, y)}\right) \tag{3.4}$$

where x is the image to measure and y the reference image. MAX_p is the maximum possible value of a pixel in the image. In the case of 8bit images this is 255. $MSE(x)$ is the pixel-wise mean square error of image x calculated according to equation 2.16 [35].

3.5.2 SSIM

The Structural Similarity Index (SSIM) is intended to better measure visual quality according to the human visual system. It measures perceived change in structural information as opposed to absolute errors like MSE or PSNR. It bases its calculations on luminance, contrast and structure and is calculated according to:

$$\mathcal{M}_{SSIM}(y, x) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)} \tag{3.5}$$

Where x is the image to measure and y the reference image. μ_x and μ_y is the mean value for x and y respectively. σ_x^2 and σ_y^2 is the variance and σ_{xy} the covariance. $c_1 = (k_1 * MAX_p)^2$ and $c_2 = (k_1 * MAX_p)^2$, with $k_1 = 0.01$ and $k_2 = 0.03$ set accordingly to the original paper [37].

3.5.3 Target differences

SSIM and PSNR are two metrics that are usually used for normal images. To further investigate the quality features typical to SAR other metrics have been used. Since

the SAR-image consists of many targets the quality of the restored targets have been measured against targets in the reference images. The distance between the brightest pixel of targets in the restored images were measured against the brightest pixel of the corresponding reference targets. This aims to measure how well the network keeps positions in the image. The distance was calculated according to euclidean distance. To measure how well the networks pertains the strength of the different targets, the difference between amplitudes were also measured according to the same selection of pixels as the distance metric.

4

Results

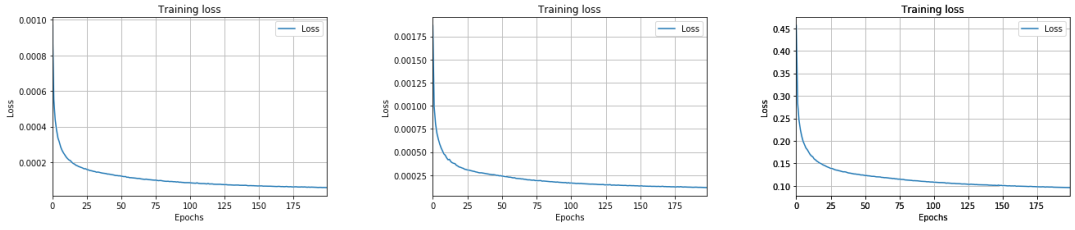
The results for the different methods are featured in this chapter. Plots for the losses are presented along with example images from the test set for visual inspection. The sections are divided in simulated and real images.

4.1 Simulated images

U-Net and the deblur regressor were trained on the simulated training dataset consisting of 8000 blurry and 8000 clear SAR-images. Different loss functions were evaluated for each architecture. The deblur regressor architecture was trained both with and without adversarial loss. Quantitative evaluation is presented as a summary for all models. All example plots are based on predictions of two different blurry images that were excluded from the training set.

4.1.1 U-Net

U-Net was trained with Huber, MSE and perceptual as loss functions. Adam was used for optimisation with a learning rate of 10^{-4}. The model was trained for 200 epochs with a batch size of 1, for each loss function. All losses had a steady downwards trend without any vanishing or exploding gradients during the training. The training took approximately two days to complete per loss function. The loss still decreased after 200 epochs for the different losses but due to time limitations it was enough to get a satisfactory result.



(a) U-Net Huber **(b)** U-Net MSE **(c)** U-Net Percept

Figure 4.1: Loss plot during training of the three different loss functions with U-Net.

4.1.1.1 Huber

U-Net with huber achieves a PSNR score of 21.44 and an SSIM score of 0.67. The predictions catches the overall structure of the targets and successfully predicts position and gets a target distance mean of 1 and amplitude mean of 5.67. As can be seen in the examples predictions, the targets appears to be smoother that in the clear images. Almost all targets at the edge of the images are captured.

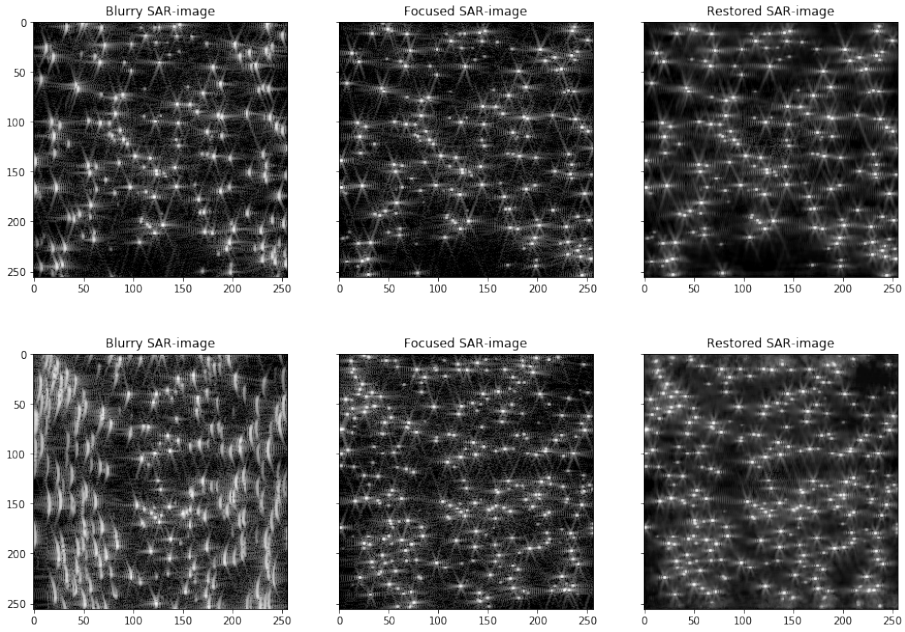


Figure 4.2: Two example plots of test data restored with U-Net architecture using Huber as loss function.

4.1.1.2 MSE

U-Net with MSE achieves a PSNR score of 21.52 and an SSIM score of 0.67. The model is able to restore the general structure of the targets and is able to successfully restore target amplitudes. The model receives a target distance mean of 1.75 and amplitude mean of 1.67. The target predictions seems to be overly smooth. Most targets at the edge of the images are captured.

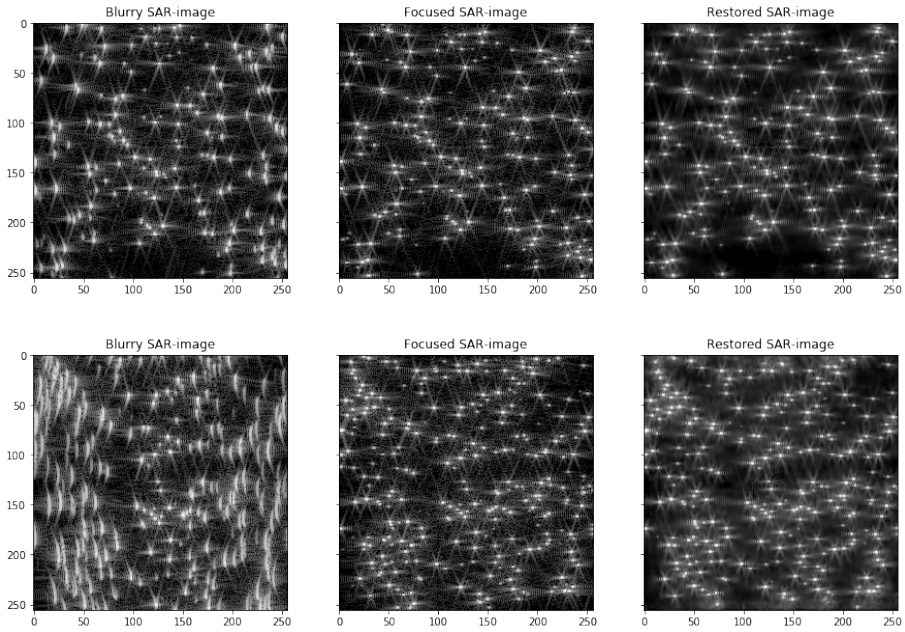


Figure 4.3: Two example plots of test data restored with U-Net architecture trained with MSE as loss function.

4.1.1.3 Perceptual

U-Net with Perceptual receives a PSNR score of 20.98 and SSIM score of 0.66. The predictions show that perceptual loss is able to capture the targets well and is able to restore them in an not overly smooth manner. The target distance mean is 1 and the amplitude mean is 3. The model is able catches most edge targets, however some are missed.

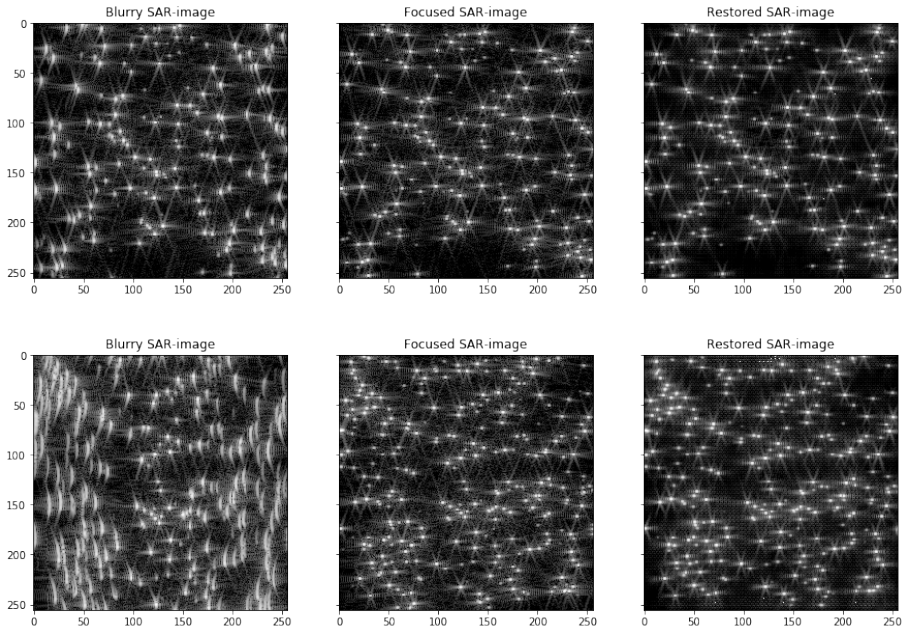
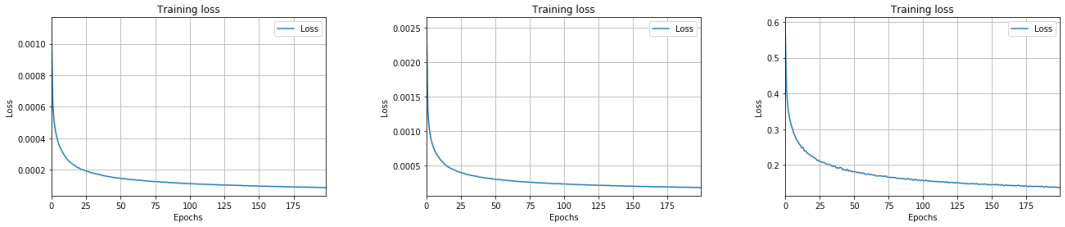


Figure 4.4: Two example plots of test data restored with U-Net architecture trained using perceptual loss.

4.1.2 Deblur regressor

The results for the deblur regressor explained in section 3.2.2 are presented below. The network was trained using Huber, MSE and perceptual loss. It was trained on 200 epochs with batch size 1. Adam was used for optimisation with a learning rate of 10^{-4}. The training took approximately 2 days to complete for each loss function.



(a) Deblur regressor Huber **(b)** Deblur regressor MSE **(c)** Deblur regressor Percept

Figure 4.5: Loss plot during training of the three different loss functions with the deblur regressor.

4.1.2.1 Huber

Huber gets a PSNR socre of 18.5 and SSIM of 0.48. Target distance mean is 9.03 and amplitude difference mean is 31.33. Huber is able to restore most of the targets but the edges look smooth as can be seen in the predicted examples.

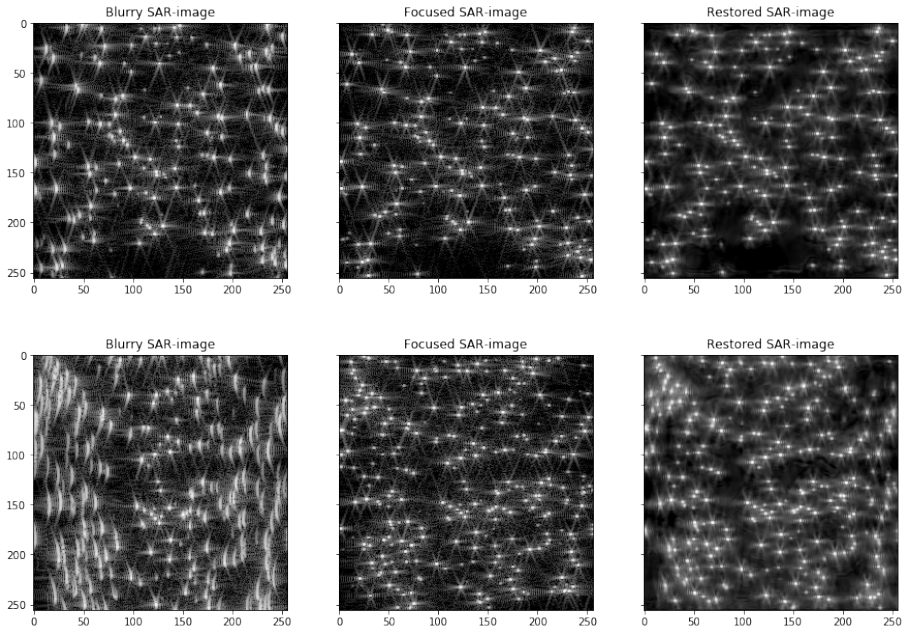


Figure 4.6: Two example plots of test data restored with the deblur regressor architecture, trained with Huber as loss function.

4.1.2.2 MSE

MSE receives a PSNR score of 18.67 and SSIM score of 0.48. It is possible to see from the predictions that MSE restores the targets but results in smoother targets in comparison to the clear images, and some of the targets in the edges are missing. Target distance mean is 8.36 and amplitude mean is 33.

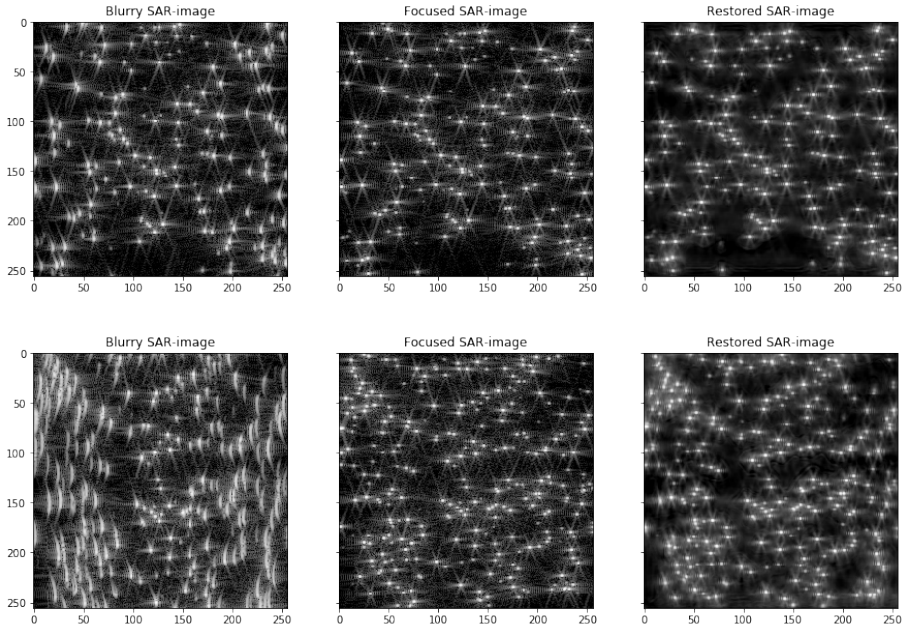


Figure 4.7: Two example plots of test data restored with the deblur regressor trained with MSE as loss function.

4.1.2.3 Perceptual loss

Perceptual loss receives a PSNR score of 19.14 and SSIM score of 0.57. It is possible to see from the predictions that perceptual loss restores the targets but fails to replicate their side lobes. The images also looks overly clear, compared to the clear reference images Target distance mean is 4.05 and amplitude mean is 8.67.

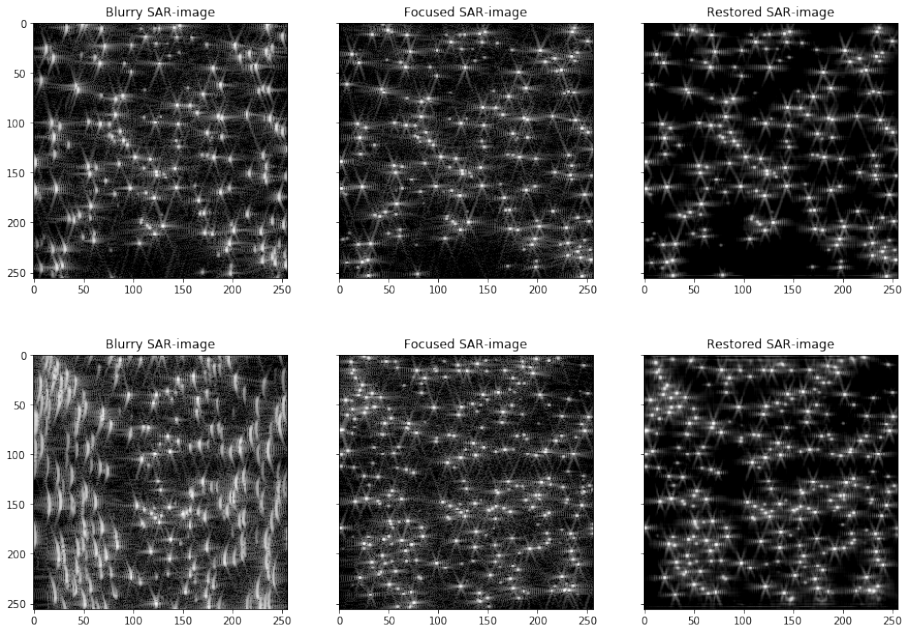


Figure 4.8: Two example plots of test data restored with the deblur regressor architecture, trained with perceptual loss.

4.1.2.4 Adversarial loss

The regressor was combined with the adversarial network explained in section 3.2.4 to create the adversarial loss training. Perceptual loss was used for the regressor along with the adversarial loss. The training took approximately 2 days to complete. Adam was used for optimisation with a learning rate of 10^{-4}. The batch size used was 8. The loss weights ratio was set to 1:1 between perceptual and adversarial loss. The training was stopped after 140 epochs, the generator collapsed and this resulted in increasingly worse predictions. The generator weights that are used for predictions in images 4.10 was achieved at epoch 65. The model receives a PSNR score of 14.32 and SSIM of 0.18. Target distance is 13.35 and amplitude mean 3.67. It is possible to observe from the example predictions that they look very similar to the clear images in style but fails to capture the targets well.

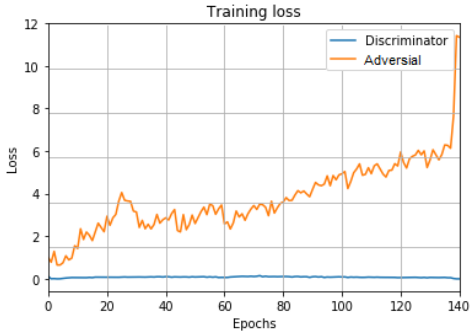


Figure 4.9: Loss plot of adversarial deblur regressor with a binary discriminator.

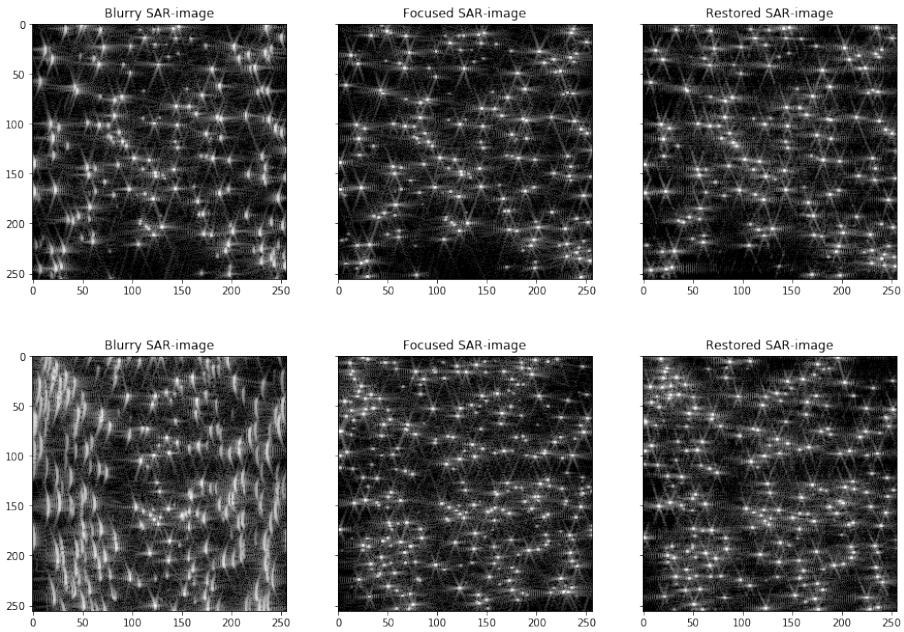


Figure 4.10: Two example plots of test data restored with the deblur architecture using a binary critic.

4.1.3 Evaluation

The different models were measured with the score metrics explained in section 3.5. U-Net with Huber loss performs the best for PSNR and SSIM while perceptual U-Net loss performs the best for target differences. The scores for PSNR and SSIM is the mean for predictions on 800 simulated SAR-images that were not included in the training set.

Method	PSNR	SSIM
No deblur	14.71	0.26
U-Net Huber	21.44	0.67
U-Net MSE	21.52	0.67
U-Net Percept	20.98	0.66
Reg Huber	18.5	0.48
Reg MSE	18.67	0.48
Reg Percept	19.14	0.57
Reg Disc	14.32	0.18

Table 4.1: Image quality scores for restored images with different methods measured against the clear image as reference. Higher PSNR and SSIM score is better.

Method	Mean	Median	Max
No deblur	17.37	26.02	26.08
U-Net Huber	1	1	2
U-Net MSE	1.75	2	2.24
U-Net Percept	1	1	2
Reg Huber	9.03	12.04	12.04
Reg MSE	8.36	11.05	13.04
Reg Percept	4.05	3.16	8.0
Reg Disc	13.35	18.03	21.02

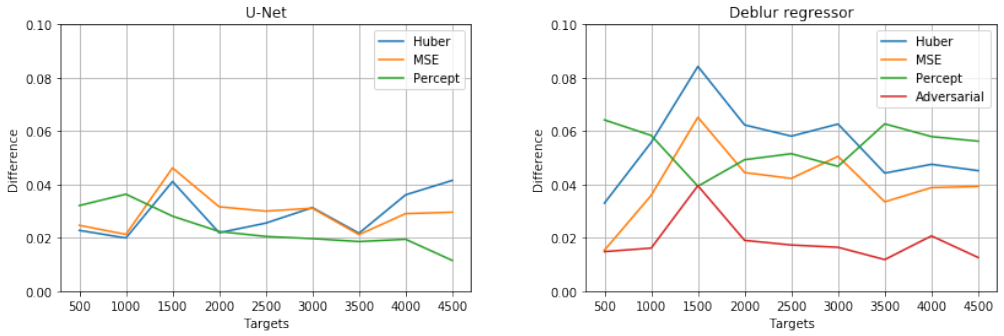
Table 4.2: Distance from the brightest pixel of the restored targets to the clear reference targets.

Method	Mean	Median	Max
No deblur	25.33	37	38
U-Net Huber	5.67	8	9
U-Net MSE	1.67	1	4
U-Net Percept	3	1	8
Reg Huber	31.33	40	54
Reg MSE	33	36	63
Reg Percept	8.67	9	15
Reg Disc	3.67	4	7

Table 4.3: Difference in amplitude between the brightest pixel of the restored targets to the clear reference targets.

4.1.4 Dense image evaluation

100 additional simulated SAR-images were created to compare network performance based on the amount of targets in an image. 10 images were created with 500 targets, 10 for 1000, and so on until 4500 targets is reached. The plot in figure 4.11 displays the average of the mean absolute error between the restored images and the focused reference images, based on logarithmic magnitude difference. All images had the same velocity error 20m/s, when the actual velocity was 25m/s. For U-Net the perceptual loss performs better than Huber and MSE. For the deblur regressor, adversarial performs the best, while the other loss functions performs slightly worse in comparison to any loss for U-Net.



(a) U-Net **(b)** Deblur regressor

Figure 4.11: Average of mean absolute error from predictions on simulated data based on the different models.

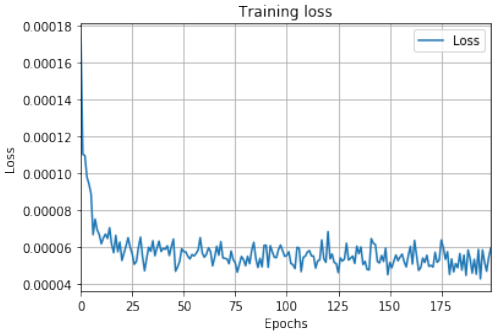
4.2 Real SAR-images

The performance of the two networks evaluated on the simulated dataset was significantly worse on the real dataset. Because of this an additional network was evaluated to be able to conclude the performance on real data. When perceptual loss was trained to find important features for the real images it was not possible to get good results, even with more complex network architectures than Effnet. Because of this, perceptual loss has not been evaluated for real SAR-images.

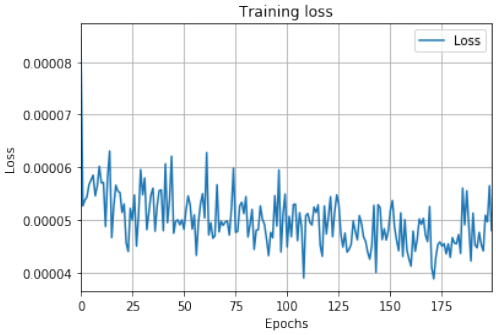
4.2.1 U-Net

The U-Net architecture was trained twice for 200 epochs on the real SAR-image dataset. The first model was trained with transfer learning with predefined weights from 4.1.1.1. In the transfer learning scenario, the learning rate was set to 10^{-6} in order to slightly adjust the weights. The other training scenario initialised the weights with Glorot uniform [38] and instead used learning rate 10^{-5}. Adam was used as optimiser in both scenarios together with batch size 1. Weights were selected when the loss was minimal in both scenarios. It is observable that the transfer learned model had a more stable training loss and converged faster. Both models

result in a blurry and smooth results that did not resemble the focused image. The two examples images were not included in the training set.



(a) U-Net with transfer learning



(b) U-Net with random weight initialisation

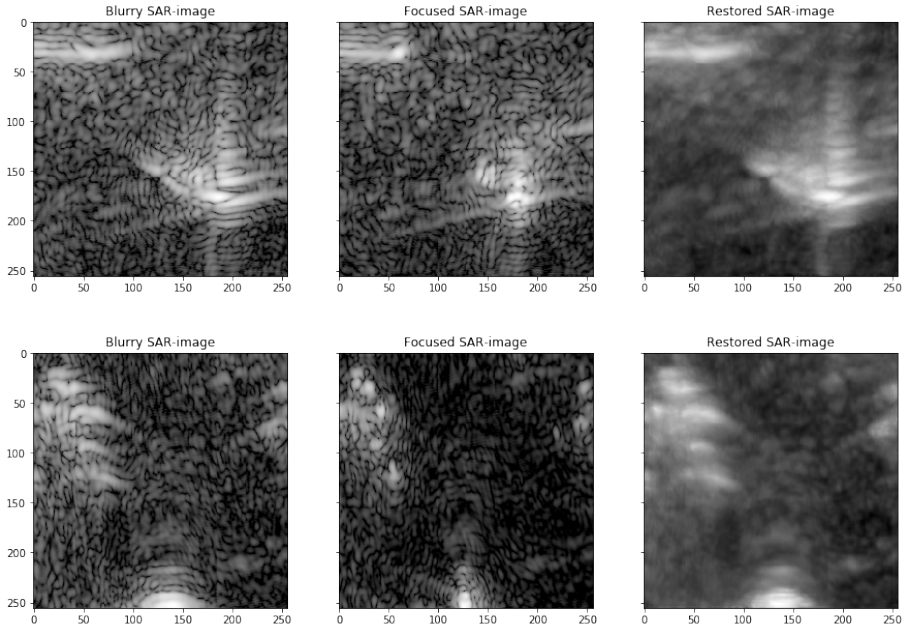


Figure 4.12: Two example plots of real SAR-images restored with a U-Net architecture that initially had the weights from 4.1.1.1.

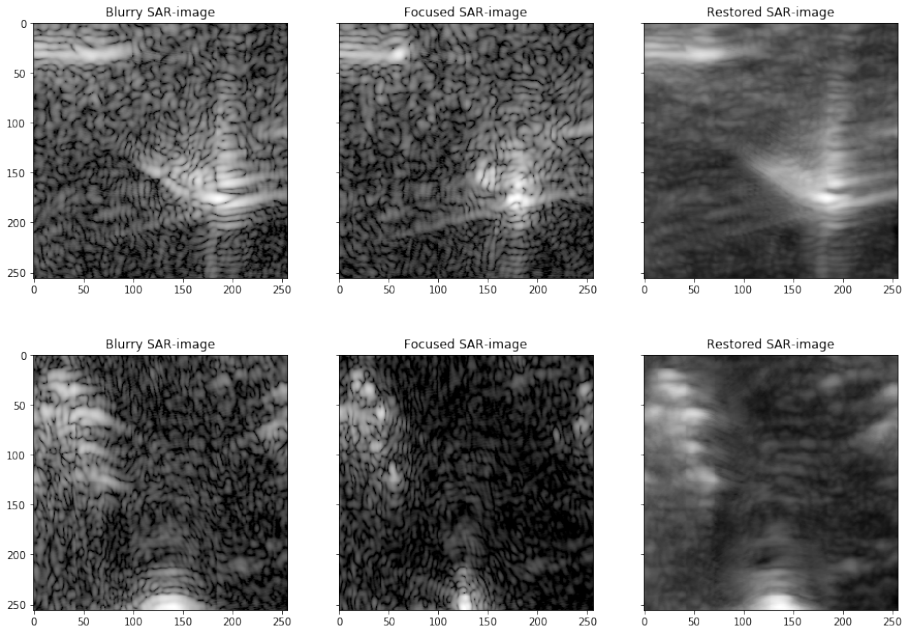


Figure 4.13: Two example plots of real SAR-images restored with a U-Net architecture that had random weight initialisation.

4.2.2 U-Net with LSTM layers

U-Net with LSTM layers was trained on real images captured by CARABAS. The network was trained for 600 epochs with batch size 1. The input dimensions 1024x1024 was the maximum size for the GPU memory on the available hardware therefore batch size 1 was the only option. The input dimensions from previous models were also increased to provide the network with more information. Adam was used for optimisation with the learning rate 10^{-4}. Huber was used as loss function. The example images are taken from the last epoch. The output manage to restore important targets but change the style of the dense terrain from the focused image. The two example images was not included in the training set.

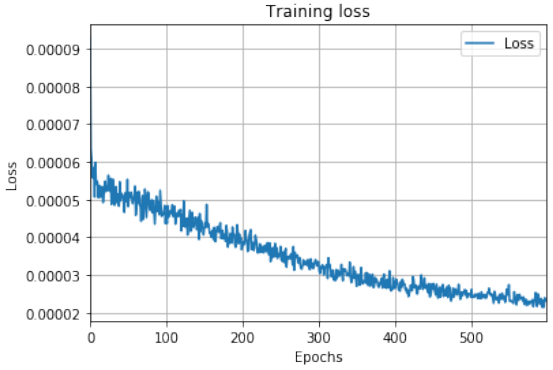


Figure 4.14: Training loss for real SAR-image input to U-Net with LSTM layers.

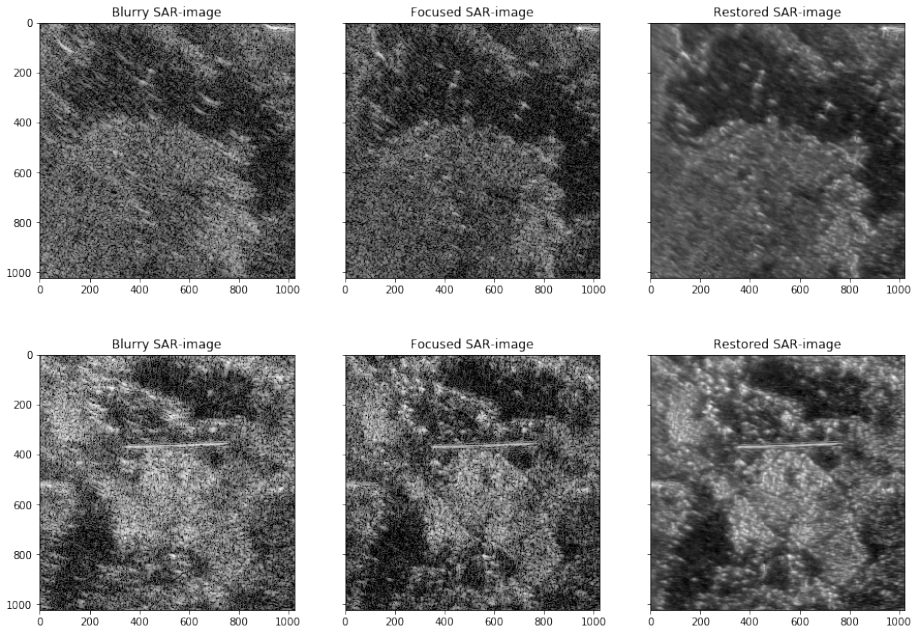


Figure 4.15: Two example plots of real SAR-images restored by U-Net with LSTM layers. The output is the magnitude and the input is a 3 channel tensor, complex and magnitude.

4.2.3 Evaluation

The models trained on the real dataset were evaluated based on a test set consisting of 100 real SAR-images with random velocity errors. The images were cropped randomly from 9 different full size images that were not included in the training set. Similar terrain could however have occurred in the training set. The box plot in figure 4.16 displays the mean absolute error between the restored images and the focused reference images. U-Net with and without transfer learning show only a small error and low variance. U-Net with LSTM has a much larger error, where the smallest value is higher than the other models.

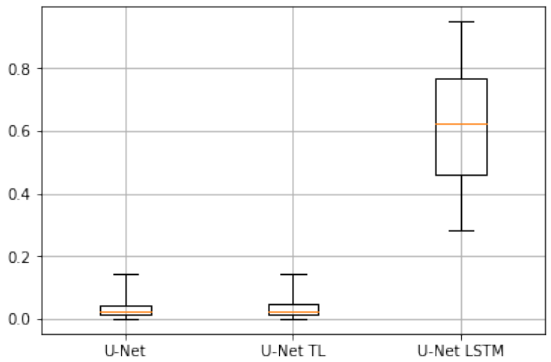
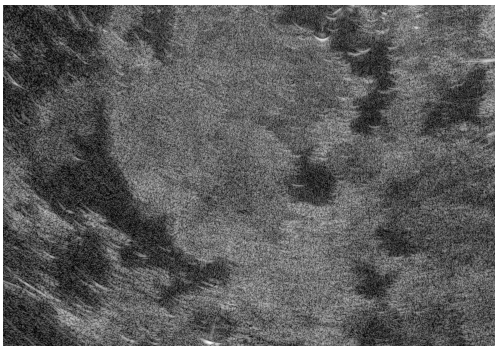


Figure 4.16: Box plot for the three models trained on CARABAS images, evaluated on the test set.

4.2.4 Satellite comparison

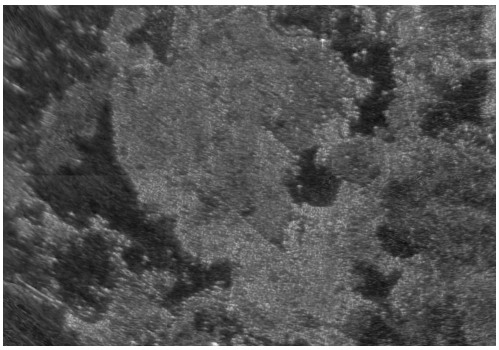
U-Net with LSTM layers was able to restore prominent features in the blurry SAR-image. It was the only model that was able to do this and was therefore selected for this comparison. An image of resolution 3072x2048 was restored by splitting the input in different segments of size 1024x1024. The restored image was then compared to a satellite image to observe how well it resembles the terrain. The blurred image is the most distinct from the satellite terrain and thus have the least practical use. When the focused and restored image is compared it is arguable that the restored image provide useful information. The restored image resembles the terrain more similar to the satellite image and it is also easier to pinpoint distinct trees in comparison to the focused image. Random parts of the terrain are potentially included in the training set, however, the specific error visible in the image are not.



(a) A blurry sample image from the test set that was given as input to the network

(b) The same image from the test set with original focus

(c) The corresponding field from the test set taken with satellite view from Google maps



(d) The same view restored with the U-Net LSTM network

Figure 4.17: SAR-image comparison to satellite image.

5
Discussion

The results show that it is possible to restore features from blurry SAR-images, both simulated and real (to a certain extent). This chapter will discuss the results for the different methods as well as compare their performance on the real images.

5.1 Simulated images

The models show that it is possible to restore simulated blurry SAR-images. The models successfully corrects the appearance of the targets and are able to position them correctly. The models also performed well with a large amount of targets in the image.

5.1.1 Model comparison

All non adversarial models were able to increase the PSNR as well as SSIM score of the images. In this regard, all U-Net models performed better than the deblur regressor models when comparing the same metrics to each other. The first main difference between the two architectures is the usage of max pooling layers in U-Net, rather than the deblur regressor's convolutional strides to downsample the input. Max pooling only keeps the most prominent features in each neighbourhood and this suggests that it better propagates important features rather than convolutional strides. The second main difference of the models is how residual connections are used. In the deblur model, add connections were used, rather than U-Net's concatenation connections. With add connections the information is iteratively propagated and refined through the network. The concatenation layers instead retains more of the original information deeper in the network. Both of these differences results in a network that keeps information rather than manipulating it. This also makes the network easier to train since it does not need to tweak parameters in order to do this.

5.1.2 Loss function comparison

When comparing the loss functions, results varied between the models. For U-Net the loss functions performed similarly between the pixel-wise losses Huber and MSE as well as perceptual. Huber performed better for PSNR and SSIM than perceptual loss, they performed equal on target distances, however, perceptual achieved a better score when amplitudes were compared. The results shows that all loss functions

are able to reduce the blur in the images. This also suggests that U-Net is robust for different loss functions. The deblur regressor had a higher variation between loss functions compared to U-Net. Even though the PSNR and SSIM was improved for the traditional loss functions. MSE achieved a worse score than perceptual in all regards, significantly so at restoring target positions. Perceptual was on par with the U-Net models for PSNR and SSIM but slightly worse at restoring target characteristics. This further suggests that the deblur regressor architecture is less robust than the U-Net architecture.

The adversarial loss models showed to be unstable to train which has also been suggested by earlier papers [33], [39]. This made it hard to get the model to converge. Looking at the loss plot for the model, the loss seemingly increases. The model resulted in worse evaluation scores, compared to the other loss functions, except for target amplitudes where it outperforms perceptual loss. Looking at the example predictions, the images look the most similar in style to the focused images in comparison to the other models, however, the model fails to correctly restore target positions. It is possible to weigh perceptual loss higher during training, but whether style or correct positions is most important has to be taken into consideration.

5.2 Real images

The models performed worse on real images. The U-Net model trained with and without transfer learning got a similar result where both models produced smoothed versions of the blurry images and fails to restore the targets, even more prominent ones. Both models got similar evaluation scores which is seemingly low. The transfer learned model was more stable to train but did not result in any better predictions. None of the models that were successful on simulated models were thus able to perform well on real images.

The U-Net with LSTM layers is the only model that resulted in improved predictions. The training of the model using the real dataset was more stable than the other models. When visually inspecting the results and comparing the most prominent reflectors, a majority of blurry features are restored, and closely resembles the focused reflectors. The image also appear to contain less blur around edged to darker areas. This suggests that the network is able to extract additional information by using sequences of images as input to LSTM layers. The model does however get a high evaluation score, which is contradictory since in comparison to the other models, the LSTM model is the one that looks visually the best.

When comparing the simulated images to the real images it is possible to explain why the results were worse. Simulated images have separate distinct targets between focused and blurry images. This made it easy for the network to find features and predict corrections. The real SAR-images lacked the distinctness between features and only larger points could be distinguished as focused targets like the simulated images. Most of the terrain looked similar in structure between focused and blurry

images. This can be explained by the fact that when collecting SAR-data there are a lot of reflectors.

In the simulated images it is only possible to create one target per pixel, while in the real images, it is possible to get more that one reflector per pixel, resulting in denser images. Terrains such as dense forests visible in figure 3.2 have a high target density and thus targets are merged and creates structures that are hard to identify as distinctly focused or blurry. As can be seen in 4.1.4 the networks also have a slight problem with simulated images when the targets are densely packed. The more dense the points are in an image, up to a certain point, the worse is the restoration. This suggests that it is hard to restore real images, since they primarily consists of these structures. During the training process of perceptual images the networks were unable to consistently classify images as blurry or focused, further suggesting that there is a problem with finding distinct features. A network using recurrent layers was not tried to binary classify between focused or blurry images.

Inspecting the full SAR-images it is possible to identify the images as blurry when looking at a reference image. When focusing on smaller parts without bright independent reflectors it is however not possible to distinguish the structure in the images. This suggest that it could be possible for a network to produce better results when training on full images, thus getting a better view of the larger structure. This however would require better hardware. Another problem with the real images were the information that the terrains provided, primarily forested areas, proved to be challenging. SAR-images taken over cities often contain more distinguishable features such as buildings and roads. Another way to get around the problem of non-distinguishable features could be to only use parts of the images with distinguishable features. This would however require a larger dataset to train a model that is able to generalise well.

5.3 Future work

The methods show great promise for blur reduction in both simulated and real images. Application on real images requires further investigation however. One potential solution is to collect more real SAR-image data which could lead to better training and generalisation. Another approach is to test the models with better hardware to provide full image input. This could lead to a model that can handle dense terrain since the differences are more apparent in the full images.

Another area without that require further investigation is how to handle complex numbers as input to neural networks. This thesis demonstrated that it is possible to split the real and imaginary parts of complex numbers and feed them as separate channels and get good results. Further research could therefore focus on neural networks that can handle complex numbers without imaginary and real separation in order to compare these results.

Further work could also include correcting flight paths rather than images. Since

the faulty flight path is available along with the blurry images, a model could take these as input and then output a correct flight path, or a correction to the faulty path. From this new flight path the image could then be processed from the raw SAR data into a focused image.

6
Conclusion

The purpose of this thesis has been to evaluate deep learning methods for deblurring SAR-images. The results showed that is was possible to get good performance on simulated images for several methods. It was evident that both pixel-wise loss functions and perceptual loss could reduce blur for the proposed models. Blurry targets could be restored to look like clear targets, while keeping the positions fairly intact. The adversarial model showed worse results for restoring targets, but can be useful for preserving the style of the images.

For real images the results were significantly worse. None of the models that performed well on simulated images could restore real images to a satisfactory level. The images contained a lot of features that were not distinctly blurry. Many of the cropped images contained a pattern that were caused by dense trees which made it hard for the networks to learn. Because of hardware limitations it was not possible to use full-sized images as input. It is believed that this could potentially reveal more information for the network and thus yield better results on the real images. The model with LSTM layers provides the network with more information of the full image since it receives a sequence of cropped images. This was the only model that was able to restore some of the more prominent targets in the images. This suggests that models with recurrent layers are a promising candidates for further investigation. However further tests are needed using better hardware to make use of the full images along with more data to be able to draw more conclusions about real images.

The thesis also found that when working with complex numbers it is possible to divide the real and imaginary parts into different channels and get a good result when using convolutional layers. Using convolutional networks that can take complex numbers is another approach that has not been investigated in this thesis, but could possibly improve the results.

References

[1] D. Macdonald, *Ritsar*, `https://github.com/dm6718/RITSAR`, 2015.

[2] O. Kupyn, V. Budzan, M. Mykhailych, D. Mishkin, and J. Matas, "Deblurgan: Blind motion deblurring using conditional adversarial networks", *CoRR*, vol. abs/1711.07064, 2017. arXiv: `1711.07064`. [Online]. Available: `http://arxiv.org/abs/1711.07064`.

[3] X. Tao, H. Gao, Y. Wang, X. Shen, J. Wang, and J. Jia, "Scale-recurrent network for deep image deblurring", *CoRR*, vol. abs/1802.01770, 2018. arXiv: `1802.01770`. [Online]. Available: `http://arxiv.org/abs/1802.01770`.

[4] C. Trabelsi, O. Bilaniuk, Y. Zhang, D. Serdyuk, S. Subramanian, J. F. Santos, S. Mehri, N. Rostamzadeh, Y. Bengio, and C. J. Pal, "Deep complex networks", *ArXiv preprint arXiv:1705.09792*, 2017.

[5] M. Wilmanski, C. Kreucher, and A. Hero, "Complex input convolutional neural networks for wide angle sar atr", in *2016 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, IEEE, 2016, pp. 1037–1041.

[6] G. W.Stimson, *Introduction to airborne radar, Second edition*. SchiTech Publishing, 1998.

[7] C. Oliver and S. Quegan, *Understanding synthetic aperture radar images*. SciTech Publishing, 2004.

[8] A. Ahlander, H. Hellsten, K. Lind, J. Lindgren, and B. Svensson, "Architectural challenges in memory-intensive, real-time image forming", in *Parallel Processing, 2007. ICPP 2007. International Conference on*, IEEE, 2007, pp. 35–35.

[9] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT Press, 2016, `http://www.deeplearningbook.org`.

[10] K. L. Priddy and P. E. Keller, *Artificial neural networks: An introduction*. SPIE press, 2005, vol. 68.

[11] D. E. Rumelhart, G. E. Hinton, R. J. Williams, *et al.*, "Learning representations by back-propagating errors", *Cognitive modeling*, vol. 5, no. 3, p. 1, 1988.

[12] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization", *ArXiv preprint arXiv:1412.6980*, 2014.

[13] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting", *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[14] J. Han and C. Moraga, "The influence of the sigmoid function parameters on the speed of backpropagation learning", in *International Workshop on Artificial Neural Networks*, Springer, 1995, pp. 195–201.

[15] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks", in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, 2011, pp. 315–323.
[16] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models", in *Proc. icml*, vol. 30, 2013, p. 3.
[17] M. Riesenhuber and T. Poggio, "Hierarchical models of object recognition in cortex", *Nature neuroscience*, vol. 2, no. 11, p. 1019, 1999.
[18] A. C. Bovik, *Handbook of image and video processing*. Academic press, 2010.
[19] V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning", *ArXiv e-prints*, Mar. 2016. eprint: 1603.07285.
[20] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation", *CoRR*, vol. abs/1411.4038, 2014. arXiv: 1411.4038. [Online]. Available: http://arxiv.org/abs/1411.4038.
[21] L. Wang, W. Ouyang, X. Wang, and H. Lu, "Visual tracking with fully convolutional networks", in *The IEEE International Conference on Computer Vision (ICCV)*, Dec. 2015.
[22] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.
[23] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation", in *International Conference on Medical image computing and computer-assisted intervention*, Springer, 2015, pp. 234–241.
[24] P. J. Werbos *et al.*, "Backpropagation through time: What it does and how to do it", *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.
[25] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks", in *International conference on machine learning*, 2013, pp. 1310–1318.
[26] Y. Bengio, P. Simard, P. Frasconi, *et al.*, "Learning long-term dependencies with gradient descent is difficult", *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 157–166, 1994.
[27] S. Hochreiter and J. Schmidhuber, "Long short-term memory", *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
[28] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with lstm", 1999.
[29] P. Huber, "Robust estimation of a location parameter", *Annals of Mathematical Statistics*, vol. 35, pp. 73–101, 1964.
[30] K. P. Murphy, *Machine learning: A probabilistic perspective*. MIT press, 2012.
[31] J. Johnson, A. Alahi, and L. Fei-Fei, "Perceptual losses for real-time style transfer and super-resolution", in *European conference on computer vision*, Springer, 2016, pp. 694–711.
[32] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, *et al.*, "Photo-realistic single image super-resolution using a generative adversarial network", *ArXiv preprint*, 2017.

[33] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets", in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[34] I. Freeman, L. Roese-Koerner, and A. Kummert, "Effnet: An efficient structure for convolutional neural networks", *CoRR*, vol. abs/1801.06434, 2018. arXiv: 1801.06434. [Online]. Available: http://arxiv.org/abs/1801.06434.

[35] A. Hore and D. Ziou, "Image quality metrics: Psnr vs. ssim", in *2010 20th International Conference on Pattern Recognition*, IEEE, 2010, pp. 2366–2369.

[36] S. Winkler and P. Mohandas, "The evolution of video quality measurement: From psnr to hybrid metrics", *IEEE Transactions on Broadcasting*, vol. 54, no. 3, pp. 660–668, 2008.

[37] Z. Wang, A. C. Bovik, H. R. Sheikh, E. P. Simoncelli, *et al.*, "Image quality assessment: From error visibility to structural similarity", *IEEE transactions on image processing*, vol. 13, no. 4, pp. 600–612, 2004.

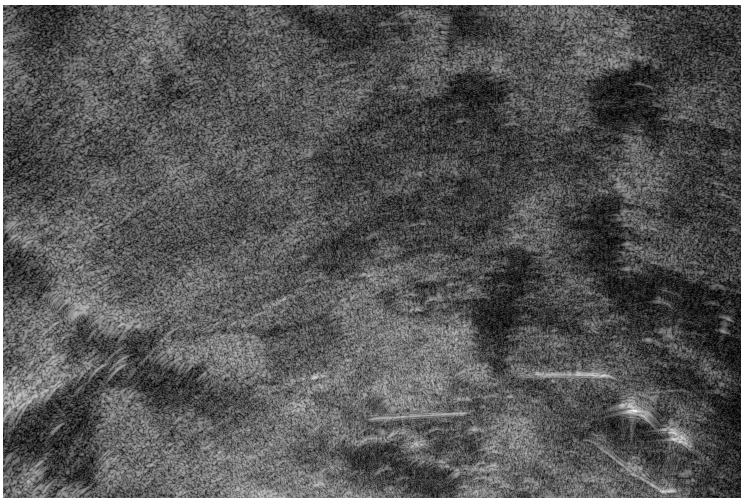
[38] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks", in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, 2010, pp. 249–256.

[39] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein gan", *ArXiv preprint arXiv:1701.07875*, 2017.

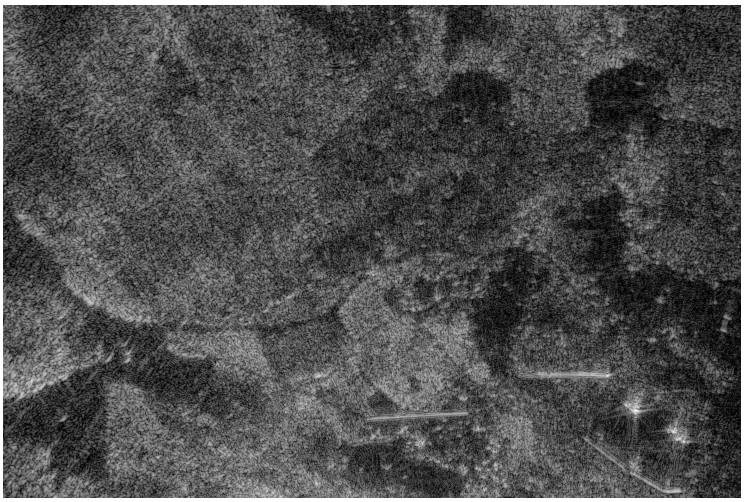
A

Real SAR-images

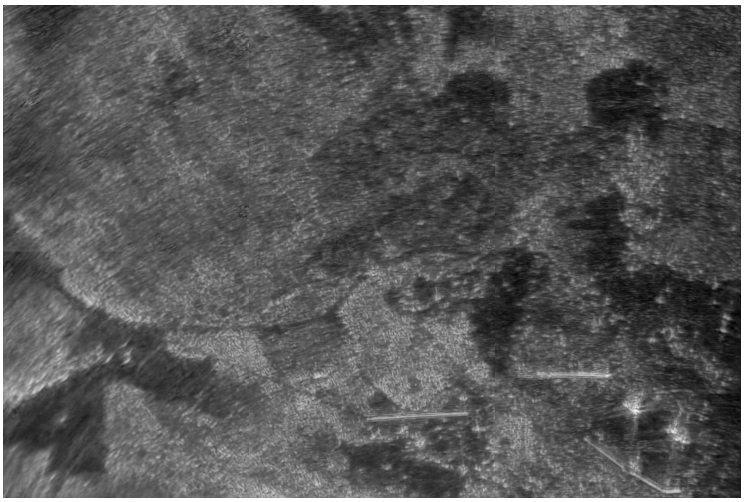
Examples of SAR-images from the real dataset in higher resolution.



(a) Blurry SAR-image captured with CARABAS



(b) Focused SAR-image captured with CARABAS



(c) SAR-image restored with U-Net with LSTM layers

Figure A.1: Blurry and focused SAR-image of the same motive as well as a restored version.

B

Frameworks

All development was done in python. The following frameworks were used for the neural network training process and data pre-processing:

- Tensorflow
- Keras
- Numpy
- Pandas
- Pillow
- Scipy
- Matplotlib