UNIVERSITY OF COPENHAGEN

Computer Science - Master's thesis

Daniel Magleby Sørensen

Estimating glacier ice volume using image inpainting

Date: May 31, 2022

Advisors: Troels Christian Petersen and Niccolò Maffezzoli

Abstract

Deep learning based state of the art image inpainting models have in recent years shown great results in performing reconstruction on damaged images, seamlessly removing relatively large objects or obstructions from images, while remaining aware of the context of the image and creating realistic replacements. This thesis proposes the use of inpainting on Digital Elevation Models in order to "remove" glaciers from the image, revealing the bedrock underneath the glacier. With an accurate prediction of the bedrock, we can calculate the total volume of ice in the glacier, by subtracting the predicted bedrock levels from the original image. An accurate prediction of the total volume of ice is important for sea rise estimations and for estimating the total amount of drinking water, which is available in the glaciers.

In this thesis we find that image inpainting on its own does not perform very well at reconstructing the bedrock underneath the glaciers. However we also discover, that introducing real world ice thickness measurements into the base models improves the ability of the model to reconstruct the bedrock significantly. Future work should explore the ability to embed meta data about the glaciers into the models in order improve the volume predictions of the entire region, but also make the model capable of predicting volume between different glacier regions, such as RGI60-13,14 and 15, which is the Himalayas.

Contents

Introduction

Motivation

The motivation of this project is based on the following question: "How would the sea level change if all glaciers melted?". The currently available estimations of the full volume of the glaciers world wide are clouded in uncertainty. Glacial volumes are a great monitoring tool for climate changes, and their melting is thought to contribute to half the global sea rising, however with a very large uncertainty, since estimating glacier volume and monitoring them worldwide is hard. In this thesis, the proposal is to do exactly this based on altimetry satellite data. A model is trained on mountain terrain without glaciers, where a part of the data (corresponding to a potential glacier) is removed. Using supervised learning (inpainting), the topography of the missing part is then estimated, which yields a model for predicting glacier volumes.

The estimation of this volume is fundamental to predicting the global rise in sea level, melting glaciers from 1971-2018 is estimated to be responsible for 22% of the sea level rise observed in the same period [IPCC, 2021]. But the melting glaciers also impact the freshwater availability for an estimated 1.9 billion people, who rely on glaciers for freshwater [Immerzeel et al., 2020]. The current estimations are based on models relying on physical laws, but requires parameters, which are often poorly constrained. To improve on this, we will work in collaboration with Dr. Niccolo Maffezzoli (fellow at Univ. of Venice, who came with the original proposal), and use the ever increasing amount of satellite images to improve the current ice volume estimations of glaciers.

Thesis goal

The first stated goal of this thesis is to identify which satellite data is the most appropriate to use, when training a model to predict the bedrock underneath a glacier. The primary data comes in the form of Digital Elevation Models (DEM), which are publicly available at different resolutions, additionally information about the location, size and shape of the glaciers present in the DEMs is required for sampling training data, and performing the final volume prediction. This data can be found in the Randolph Glacier Inventory (RGI)[1]. [Consortium, 2017]

[1] Directory containing information about glacier shape, height in meters, area in km^2, etc.

The second stated goal is to experiment with state of the art deep learning based image inpainting, and asses the efficacy of these models and their ability to infer glacier ice volumes from the curated satellite data and supplemental data. The general hypothesis is that

an image inpainting model can learn to predict the bedrock level, of a de-glacierized area, which when applied to a glacierized area can be used to determine an estimate of the bedrock level beneath the glaciers from which an estimate ice volume can be computed.

Related work

In past works, the volume of glaciers have been estimated using numerous methods, such as modelling physical conditions, hybrid solutions estimating small sections of glaciers and interpolating the volume and large scale simulations used to match past observations with the present.

In [Huss and Farinotti, 2012] the authors deploy a physically based approach for estimating glacier ice thickness distribution and volume. The model in this work combines glacier outlines from the RGI with terrain elevation models from Shuttle Radar Topography Mission (SRTM) and Advanced Spaceborne Thermal Emission and Reflection Radiometer (ASTER) to obtain thickness of individual glaciers. The model builds on mass turnover and the flow law for ice. Another model based in physical characteristics is described in [Fürst et al., 2017], which uses a two stage approach first estimating the volume using mass conservation and secondly using velocity. The two stages is required because satellite remote sensing data often fails to cover an entire basin, in the first stage ice flux is translated into a glacier wide thickness, which is then supplemented in stage 2 in areas with reliable surface velocity information.

In [Frey et al., 2014] the GlabTop2 algorithm is used to calculate glacier volume. The algorithm works by randomly selecting DEM cells within the glacierized area. The ice thickness for all glacier cells is then interpolated from the ice thickness at the random DEM cells and from cells at the glacier margins, which is known to be zero. The algorithm requires as input a shapefile, with the glacier outline, a high resolution DEM, with a resolution greater than 100m and a mask of the same resolution as the DEM marking the glacierized cells. Additionally the algorithm has parameters, which changes the thickness estimations and interpolation process, these are τ, which is the basal shear force [2] and f, which is the shape factor. τ is in this work calculated as a function of vertical glacier extent and f is by default set to 0.8. The same algorithm is also used in [Ramsankaran et al., 2018], with different conditions for the parameter optimization.

Finally in [Eis et al., 2019] the initial glacier states used for the OGGM framework [3] is explored in order to produce better glacier volume estimates. In this work, a large set of physically plausible

[2] Basal shear stress is the component of stress directed parallel to the ground at the base of an ice sheet or glacier. On a global scale basal shear stress is the result of gravitational force pulling a mass of ice downhill. Locally, basal shear stress is the combination of gravitational force and the force exerted on a specific region of ice by adjacent ice.

[3] OGGM is an open source modelling framework for glaciers, which accounts for glacier geometry, ice dynamics and more. In order to simulate past and future mass-balance, volume and geometry of glaciers.

glaciers for a given year is generated and simulated to the present.

A common theme in these methods is a reliance on physical phenomena and/or a large amount of historical information about the glaciers, which themselves are subject to a certain degree of uncertainty. In this paper the method proposed relies solely on a high resolution DEM and glacier outlines from sources such as RGI, by using state of the art deep learning based image inpainting.

Deep Learning

In this section a theoretical view of deep learning and neural networks is given, since deep learning plays an integral part of the methods deployed in this paper. An overview of basic neural networks is given as well as the techniques back-propagation, which is the backbone of modern neural networks, additionally loss functions are discussed as well as regularization, which is used to combat overfitting. Finally a thorough investigation of convolutional networks and their layers, since these networks lay the foundation for image inpainting.

Neural networks

Deep learning algorithms are a subset of neural networks, machine learning and artificial intelligence, which imitates our current understanding of the function and structure of the human brain in order to learn[4] how to solve complex problems. Deep learning methods has proven to be a powerful machine learning method and has demonstrated great results in solving problems in fields such as natural language processing, image processing as well as speech recognition and generation. Deep learning methods have been derived from artificial neural networks (ANN), which is created by connecting multiple layers of neurons together in a large network. In this network of neurons, each neuron implements a nonlinear transformation, which is able to capture features at each layer and create a representation of the input data. One very powerful advantage of neural networks is that the network is able to learn features directly from the input data, without requiring manual design of features. This is in comparison to more simple machine learning methods, in which the features has to be extracted manually. [5]

[4] In [Mitchell, 1997] computer learning is defines as: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."

[5] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org

The building block of every neural is the neuron bearing the same name as the biological component it is trying to imitate. In figure 1 an illustration of a single neuron is given. As seen a neuron is a

simple mathematical function, which computes a weighted sum from the output of other neurons denoted by x_j, the summation multiplies x_j by the weight w_{ij} for each incoming signal.

$$a_i = \sum_{j=1}^{d} w_{ij} x_j + b_i$$

The neuron is then "activated" by an activation function $h(a_i)$, which is the output of that neuron.

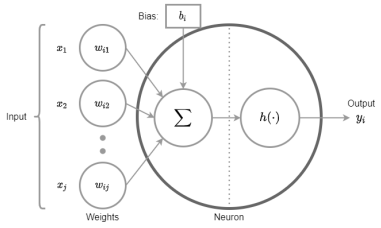


Figure 1: Single neuron with input vector x_j and implicit bias b_i, which is activated by the activation function $h(\cdot)$.

Activation functions

A neural network without an activation function would be a linear regression model, as the activation function is what creates the non-linear aspects of the network, which enables more complex tasks to be solved. An activation function in a neuron, is a function, which decides if the computed sum of inputs and weight should be activated or not. This can be seen in figure 1 denoted by $h(\cdot)$.

Step function: The binary step function is one of the most simple activation functions, which implements a threshold and is defined as:

$$h(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

Sigmoid function: The sigmoid activation function is one of the most commonly used when implementing a neural network. The sigmoid activation function is defined as:

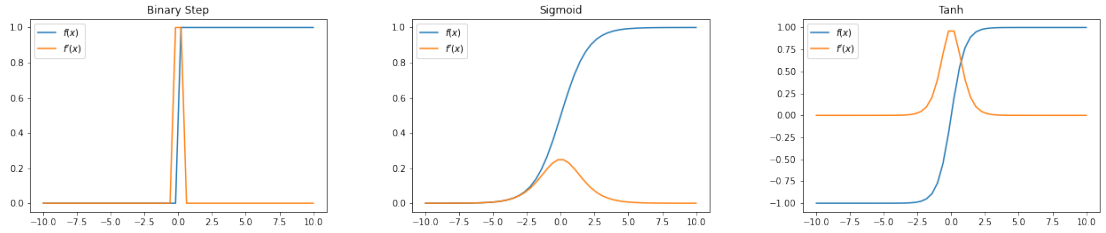
$$h(x) = \frac{1}{1 + e^{-x}}$$

This activation function squeezes the input into the range $[0, 1]$, this function has a smooth gradient, which makes it good for classification tasks. However the sigmoid activation function does have some drawbacks; In some configurations the network will stop learning and the sigmoids will saturate and kill gradients, this problem is well known ans is called vanishing gradient. With neural networks of increasing size the vanishing gradient problem becomes significantly worse. With sigmoid activation functions, the derivative lies between 0 and .25, which when multiplied during backpropagation causing the gradient to decrease very quickly.

Tanh function: The tanh activation function is very similar to the sigmoid activation function but squeezes the input values to $[-1, 1]$, which means the function is zero centered, unlike the sigmoid function. The gradient of the tanh function is stronger than the sigmoid

function due to the steeper derivative as seen in figure 2(c), but much like the sigmoid function, the tanh activation function suffers from the same vanishing gradient problem. The function is defined as:

$$h(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$



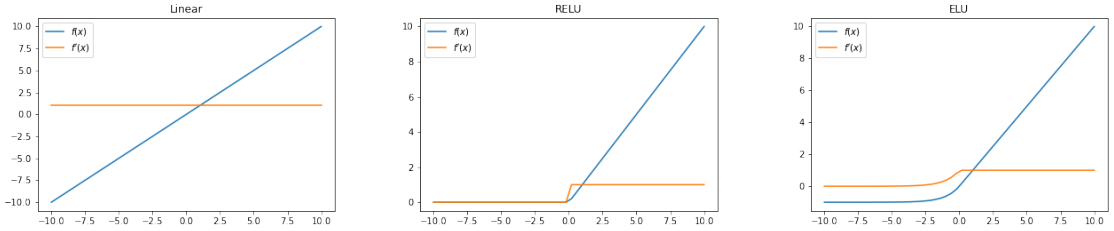
(a) (b) (c)

Figure 2: These graphs shows in (a) the binary step activation function, (b) the sigmoid activation function and (c) the tanh activation function.

Linear function: The linear activation function is given by:

$$h(x) = ax$$

The function can give a range of activation, hence the activation is not binary, but a major drawback is that the derivative is constant, this means the gradient has no relationship to the input and an error in prediction can not be solved during optimization.



(a) (b) (c)

Figure 3: These graphs shows in (a) the linear activation function, (b) the ReLU activation function and (c) the ELU activation function.

ReLU function: The rectified linear unit (ReLU) activation function is a piece wise linear function defined by:

$$h(x) = \max(0, x)$$

The ReLU activation function provides the same non-linear benefits as the sigmoid function, but with a better performance, since the function is less computationally expensive. Additionally the function rectifies the vanishing gradient problem. Since the activation function is unbound at $x > 0$, the activation function can suffer from exploding gradient, which is an explosive increase in the gradient value. This activation function has several versions, such as the Leaky ReLU, which is a ReLU function, which allows for small negative

values and parametric ReLU, which is a ReLU function, in which the slope at negative values can be increased or decreased.

Exponential LU function: The Exponential linear unit (ELU) activation function resembles the ReLU function, but allows for negative values with a smooth slope. The function is defined by:

$$h(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha(e^x - 1) & \text{if } x < 0 \end{cases}$$

where α is a parameter determining the slope of the negative part of the function.

Loss function

A loss function is a method used to measure the difference between the prediction produced by a model and the target output. The goal of the model is to learn a set of weights and bias, that minimizes the loss function. For classification tasks the combination of a softmax activation function and cross-entropy loss function can often be used to solve the task. In this paper image segmentation will briefly be mentioned for the purpose of generating masks for simulated glaciers, for this task a sigmoid activation function is combined with the binary cross-entropy, this will label every pixel with a value in the range $[0, 1]$ and a mask can be extracted using a threshold. Binary cross-entropy is given by:

$$C = -\frac{1}{N}\sum_{i=1}^{N}[y_i log(p_i) + (1 - y_i) log(1 - p_i)]$$

For image inpainting the complex loss functions used are based on root mean squared error (RMSE), mean squared error (MSE) and mean absolute error (MAE).

Backpropagation

Backpropagation is an efficient method used to compute the gradients in a directed graph, which is one way of representing a neural network. Backpropagation is a computational method, which uses the chain rule for derivatives, which allows for the computation of all partial derivatives of the neural network in linear time with respect to the depth of the graph. The use of this method is one of the things, that makes deep learning viable, since a naive approach for computing gradients would scale exponentially with the depth of the graph. Backpropagation is itself not a learning method, but rather a method often used by other learning methods.

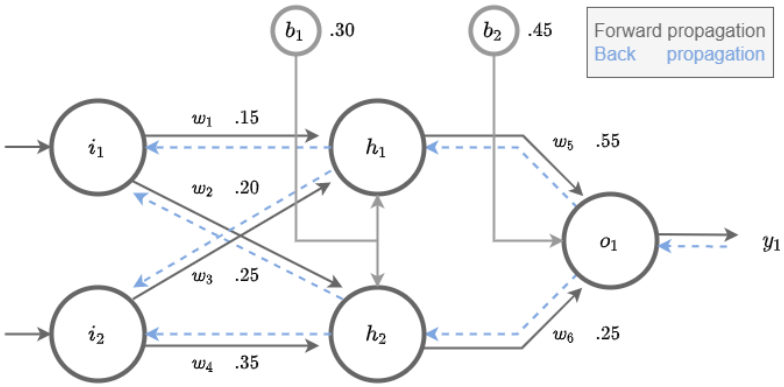


Figure 4: Simple multi layer perceptron with two inputs, two hidden layers and one output.

Performing forward propagation on the network in figure 4 with the initial weights: $i_1 = 0.1$ and $i_2 = 0.3$ and assuming a logistic activation function, we can calculate the net input to h_1 and h_2 in the following fashion:

$$in_{h1} = w_1 * i_1 + w_3 * t_2 + b_1 = .39$$
$$in_{h2} = w_2 * i_1 + w_4 * t_2 + b_1 = .425$$

The output of h_1 and h_2 is the calculated by applying the logistic activation function:

$$out_{h1} = \frac{1}{1+e^{-.39}} = 0.596, \quad out_{h2} = \frac{1}{1+e^{-.425}} = 0.604$$

Using the same procedure we calculate the net input and output of the output layer o_1:

$$in_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 = 0.929$$
$$out_{o1} = \frac{1}{1+e^{-0.929}} = 0.716$$

with a loss function defined as:

$$L = \sum \frac{1}{2}(target - output)^2$$

assuming that the target output for the network is .01 the total loss for this initial forward propagation would be 0.249.

Consider we would like to know the change the weight w_5 affects the total loss. For this we need to apply the back propagation for output layer case. By applying the chain rule we have:[6]

$$\frac{\partial L}{\partial w_5} = \frac{\partial L}{\partial out_{o1}} \cdot \frac{\partial out_{o1}}{\partial in_{o1}} \cdot \frac{\partial in_{o1}}{\partial w_5}$$

[6] Another approach is using the delta rule, which is written as $\delta_{o1} = \frac{\partial L}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial in_{o1}} = \frac{\partial L}{\partial in_{o1}}$ therefore $\frac{\partial L}{\partial w_5} = \delta_{o1} * out_{h1}$.

We can then derive the three pieces of the partial derivative individually:

$$\frac{\partial L}{\partial out_{o1}} = out_{o1} - target_{o1} = 0.706$$
$$\frac{\partial out_{o1}}{\partial in_{o1}} = out_{o1}(1 - out_{o1}) = 0.203$$
$$\frac{\partial in_{o1}}{\partial w_5} = 1 * out_{h1} * w_5^{(1-1)} = 0.716$$
$$\frac{\partial L}{\partial w_5} = 0.706 * 0.203 * 0.716 = \underline{\underline{0.102}}$$

Next if we want to know the contribution of w_1 on the loss function, we have to apply back propagation for hidden layer case. This will often be written as:[7]

$$\frac{\partial L}{\partial w_1} = (\sum_o \frac{\partial L}{\partial out_o} * \frac{\partial out_o}{\partial in_o} * \frac{\partial in_o}{\partial out_{h1}}) * \frac{\partial out_{h1}}{\partial in_{h1}} * \frac{\partial in_{h1}}{\partial w_1}$$

[7] May be written as $\frac{\partial L}{\partial w_1} = \delta_{h1} i_1$, when using the delta rule.

Since the network in figure 4 only has one output node we start by computing:

$$\frac{\partial L}{\partial out_{h1}} = \frac{\partial L}{\partial in_{o1}} * \frac{\partial in_{o1}}{\partial out_{h1}}$$

Which can be split into two computations as follows, using many of the values above:

$$\frac{\partial L}{\partial in_{o1}} = \frac{\partial L}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial in_{o1}} = 0.706 * 0.203 = 0.143$$
$$\frac{\partial in_{o1}}{\partial out_{h1}} = w_5 = 0.55$$
$$\frac{\partial L}{\partial out_{h1}} = 0.143 * 0.55 = 0.078$$

with this we have:

$$\frac{\partial out_{h1}}{\partial in_{h1}} = out_{h1}(1 - out_{h1}) = 0.240$$
$$\frac{\partial in_{h1}}{\partial w_1} = i_1 = 0.1$$

With these partial derivatives we can find the contribution of w_1 to the loss as:

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial in_{h1}} * \frac{\partial in_{h1}}{\partial w_1} = 0.078 * 0.240 * 0.1 = \underline{\underline{0.001872}}$$

Gradient-based learning

Gradient-based learning refers to a range of gradient based optimization methods such as gradient descent and stochastic gradient

descent (SGD). The goal of these methods is to find the minimum of a function.

With back propagation explained we now have a means of defining every weight in a network with respect to the loss function. The gradients, that we calculate during back propagation can inform the algorithm of which direction to move, in order to find the minimum of the function.

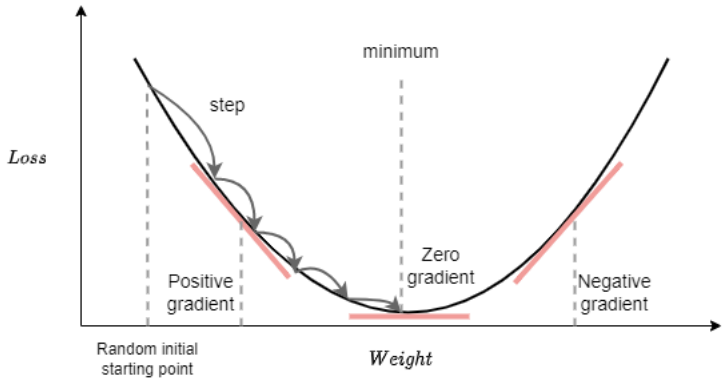


Figure 5: Gradient descent example graph.

Gradient descent is a step-wise method for finding a minimum of a function using a step-size η, which is taken in the opposite direction of the gradient, in order to find the minimum. For the weights of a neural network the change in weighs is defined as:

$$\Delta w_i = -\eta \frac{\partial L}{\partial w_i}$$

Which is then applied to the current weight in order to update the weight:

$$w_{updated} := w_{old} + \Delta w_{old}$$

Gradient descent in it's most basic form does however suffer from some quite significant drawbacks, in figure 6 and 7 we can see how the algorithm can get stuck above the global minimum if the learning rate is too high, and if the learning rate is too low, the algorithm can get stuck in a saddle point and never find a global minimum.

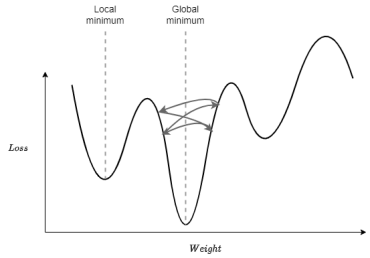


Figure 6: High learning rate.

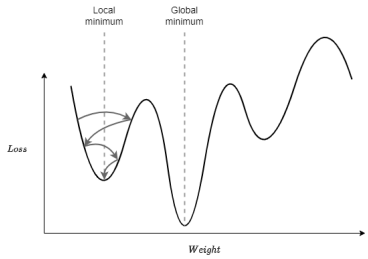


Figure 7: Low learning rate.

One solution to this problem is with the use of SGD algorithm over the the basic gradient descent algorithm. The key difference between the two algorithms is that while gradient descent runs through every sample in the dataset before parameters are updated, the SGD algorithm uses only a subset of the training data and as little as one observation before updating the parameters, This randomness allows the algorithm to escape saddle points. This does however come with the downside that the path to the minimum becomes visually noisy and instead of a smooth loss curve, we see a noisy version. This does not mean that the SGD algorithm is incapable of getting stuck at a

saddle point, it is still important that a good learning rate is chosen and that a sensible learning rate decay is picked.

Regularization

A central and pivotal problem in machine learning is how to make an algorithm that performs well on not only the training data, but also on new unseen data. As a central problem, many strategies has been developed, which reduces the test error in exchange for an increased training error, the goal of these strategies is to make the solution given by the machine learning algorithm more general. In order to talk about regularization methods, first we must visit two terms: overfitting and underfitting.

Overfitting is a term used, when the gab between the training and test error is too large. In general this often means that the model created a function, which is more complex, than the underlying function, which describes the test data. Example as seen in figure 10 where the function is more complex, than it has to, when compared to a more simple function with a better fit in figure 9.

Underfitting is a term describing a model, which is unable to obtain a sufficiently low training error. The causes of underfitting are numerous, but often insufficient training data is the cause. An example of an underfitting model can be seen in figure 8.

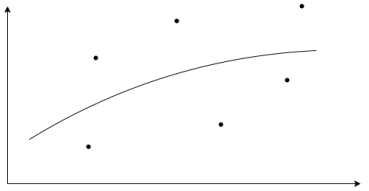


Figure 8: Example of a model underfitting.

Weight decay

One strategy to prevent a model producing a solution, which is too complex is to reduce the number of parameters in the model. However this strategy would be very limiting, and we would be unable to solve complex problems, since we want more parameters, which connects to various parts of the neural network, which create nonlinearity. In order to keep a large amount of parameters from becoming overly complex, we can deploy weight decay. Weight decay is a strategy, in which we add the a term to our loss function, which includes our model weights:

$$L + \gamma \frac{1}{2} ||w||^2$$

with the regularization hyperparameter $\gamma \geq 0$. This strategy of using weight decay will punish the network for having large weight parameters.

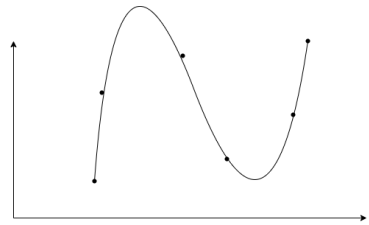


Figure 9: Example of a model fitting a problem properly.

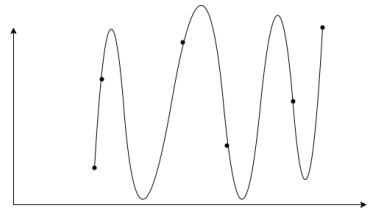
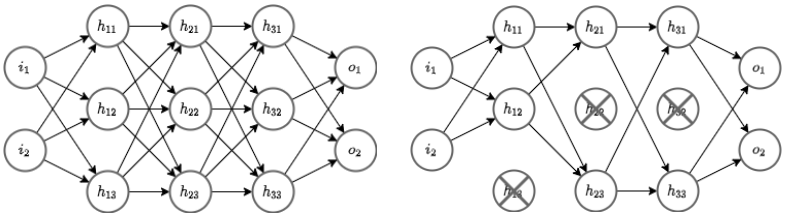


Figure 10: Example of a model overfitting.

Dropout

Another strategy for dealing with overfitting and making the model more generalized is the implementation of dropout. Dropout works

by making the model hold out a ratio of the neurons randomly. The idea is that this should stop the neurons from co-adapting, which is when a neuron relies on activation from other neurons. A model with dropout applied can be compared to an ensemble model, since the model samples from a large amount of thinned models. The model then learns multiple independent representation of the features of identical training data. An example of a model with dropout applied can be seen in figure 11.



(a) (b)

Figure 11: (a) A fully connected neural network before applying dropout. (b) network after applying dropout.

Early stopping

Briefly a strategy, which is as effective as it is simple. Early stopping is a strategy, in which we monitor a condition in the model, what if we see no improvement, then we end the training. One very common stopping criteria is is the loss stops improving, then stop training after a number of epochs of no improvement.

Batch normalization

A strategy, which affects the neuron outputs is batch normalization. During the training of the model a batch normalization layer first determines the mean μ and the variance σ^2 of the activation values:

$$\mu = \frac{1}{N}\sum_i h(a_i), \qquad \sigma^2 = \frac{1}{N}\sum_i (h(a_i) - \mu)^2$$

The activation vector is then normalized such that each neurons output follows a standard distribution across the batch:

$$h(a_i)_{norm} = \frac{h(a_i) - \mu}{\sqrt{\sigma^2 - \epsilon}}$$

This process is shown in figure 12.

Finally each neurons output is calculated by applying a linear transformation with two trainable parameters, this allows the model to train the parameters to achieve an optimal distribution for every hidden layer. Here α adjusts the standard deviation and β adjusts the bias, which moves the peak of the distribution.

$$h(a_i)_{out} = \alpha \cdot h(a_i)_{norm} + \beta$$

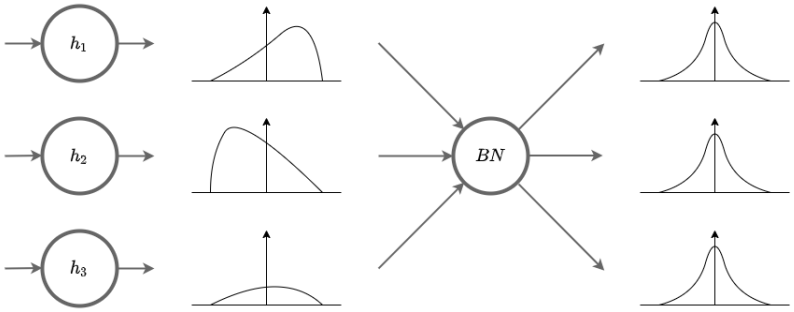


Figure 12: Illustration of a batch norm layer normalizing the signal to mean 0 and variance 1.

Convolutional neural networks

Convolutional neural networks (CNN) are a variation of the multi-player perceptron. These networks takes their name from the convolutional operation, which enables the network to digest images and extract features from the data. Since CNNs are pseudo-invariant[8] to translation, rotation and scaling, the networks are great for image recognition tasks and object detection. A canonical CNN consists of convolutional blocks, which are convolutional layers, each producing several feature maps, which are activated by a non-linear function followed by a pooling layer on top of a standard neural network, which is tasked with interpreting the output of the convolutional block.

[8] CNNs are not truly invariant, however when the separate neurons has seen several rotations of an object, the neurons are likely to fire even if the rotation is yet to be seen during training. The same principle is true for scaling.

Convolutional layers are layers which implements the convolutional operation, which is an operation, that takes two function for example f and g and produces a third function $(f * g)$, which expresses how the shape of one is modified by the other. 1D convolution with the filter kernel w is defined by:

$$s(t) = (x * w)(t) = \int x(a)w(t-a)da$$

2D convolution of a discrete image I with the filter kernel K is defined as:

$$\begin{aligned} S(i,j) = (I * K)(i,j) &= \sum_m \sum_n I(m,n)K(i-m, j-n) \\ &= \sum_m \sum_n I(i-m, j-n)K(m,n) \end{aligned}$$

The 2D convolution creates a new image $S(i,j)$ from the discrete image I and the kernel K, see figure 14 for an example of this operation, with a filter of height = 2, width = 2 and stride = 1, stride defines how many pixels the filter is moved.

The convolutional layer works in three steps; First a feature map is computed by convolution of the input with linear filters. Second,

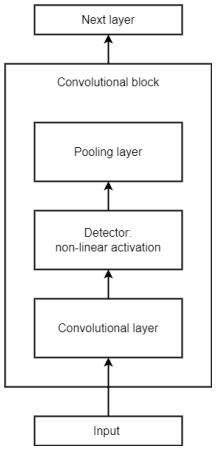


Figure 13: Illustration of terminology, to distinguish between convolutional layer and block.

The coefficients are learned as weights in the neural network, and third, a non-linear function is applied to the feature map, which one can view as the neurons sharing common weights. The non-linear activation function most often used for convolutional layers are the ReLU activation function, this function is particularly useful for images, since the values does not become negative after activating.

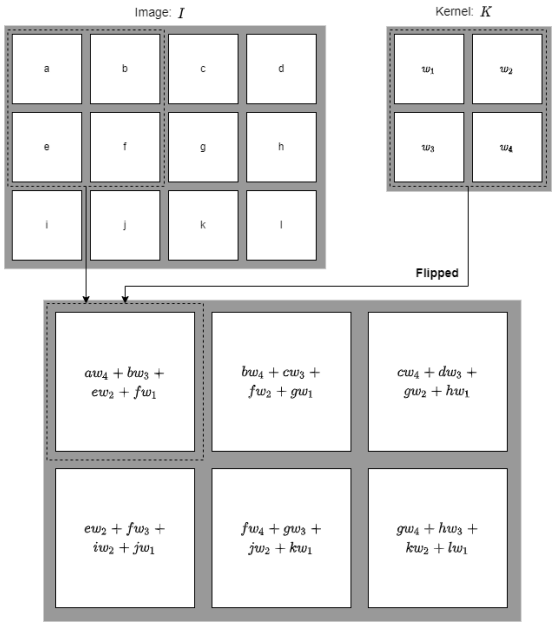


Figure 14: an example of 2D convolution over an image with kernel size (2,2) and stride 1, with kernel flipping. [Goodfellow et al., 2016]

After each convolutional layer and detector layer, which is the application of an activation function on the feature map. We have the **pooling layer** as seen in figure 13. A pooling layer is a layer which given an input from a detector layer creates a summary of the output, an example of a pooling layer can be seen in figure 15, which is a max pooling layer, which given a neighborhood returns the maximum value. Other pooling functions may be an average of a neighborhood or the L^2 norm of a neighborhood. Pooling layers helps reduce the dimensionality, increase the scale and support translation invariance. [Goodfellow et al., 2016]

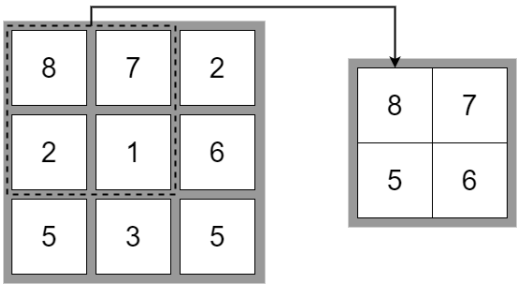


Figure 15: Max pooling example with kernel size (2,2) and stride 1.

The convolutional block leverages three important ideas, these are sparse interactions, parameter sharing and equivaraint. These three ideas makes it such that, image recognition and objects detection can be done with relatively simple models on relatively low specification hardware. This is important if you want to incorporate the power of these methods onto small devices.

Sparse interactions: the kernel of a convolutional layer is often much smaller than the input, this means each neuron is not fully connected, which can increase memory and statistical efficiency.

Parameter sharing: As mentioned above neurons that are part of the same feature map shares the same weights. This also increases memory and statistical efficiency.

Equivariance: When an input is translated, then the output is translated in the same way.

Diluted convolution

Diluted convolution is a method, in which the kernel is expanded by inserting holes between elements, this causes pixels to be skipped and allows for more information to be gathered without increasing the number of parameters in the network. This operation is often used for the benefit of increasing the receptive field without adding an excessive amount of convolutional layers.

Residual learning

Residual learning is a method in deep learning, initially developed with deep convolutional networks in mind [He et al., 2015], which has been shown to be effective in making the learning in the network easier. Let $H(x)$ be a fitting achieved by a series of layers with x denoting the input given to the first layer. The hypothesis is that, if a series of connected non-linear layers asymptotically can approximate complex functions, then the layers should just as easily be able to asymptotically approximate the residual function $H(x) - x$, so rather than having the network approximate $H(x)$, we can have the network instead approximate the residual function $F(x) := H(x) - x$, hence the original function becomes $F(x) + x$. The process can be defined as:

$$y = F(x, \{w_i\}) + x$$

where x and y are the input and output and the function $F(x, \{w_i\})$ is the residual mapping to be learned. Given the network described in figure 16 the function would be given by $F = h_2\sigma(h_1 x)$, where σ denotes the ReLU activation function. The operation $F + x$ is the element-wise addition between the stacked layers and the shortcut connection, this procedure does not add more parameters to the network.[He et al., 2015] Another version in the residual block is the

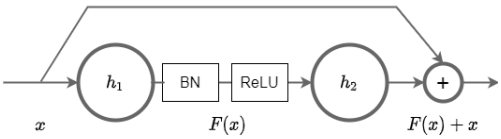


Figure 16: Illustration of residual building block, showing the shortcut created in the network.

diluted residual block, which uses the aforementioned diluted convolutional layer to increase the receptive field of the model. This type of diluted residual block is often used for GANs used for inpainting. [Demir and Unal, 2018]

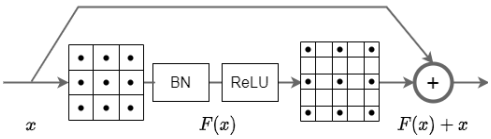


Figure 17: Illustration of diluted residual building block.

Image segmentation

Typically a CNN is used to perform classification tasks, where an image is given a single class label. However for some visual tasks, it would be preferred, that a mask classifying every pixel is returned. An architecture, which seamlessly accomplishes this task, even with a relatively small training set is the U-net. The U-net is a network, which consists of a contracting path on the left and an expanding path in the right side, as seen in figure 18,

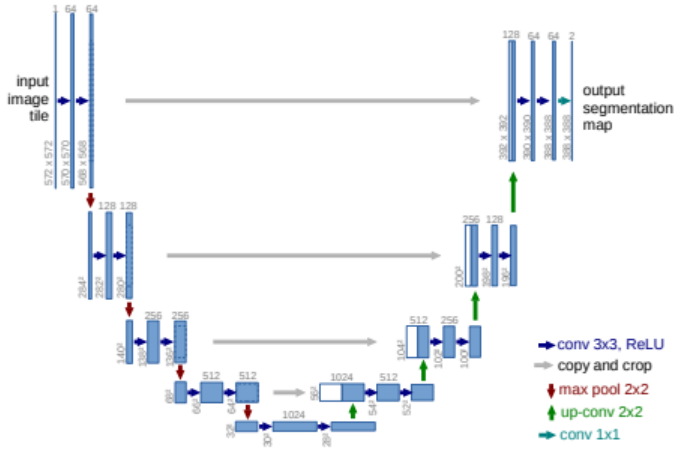


Figure 18: Illustration of the U-net architecture as defined in [Ronneberger et al., 2015].

The contracting path consists of convolution blocks, which works to down-sample the input. However, what makes the semantic segmentation possible is the introduction of transposed convolutional layers, which works to up-sample the classifications into a mask of

classifications for each pixel. An example of the transposed convolutional operation is given in figure 19.

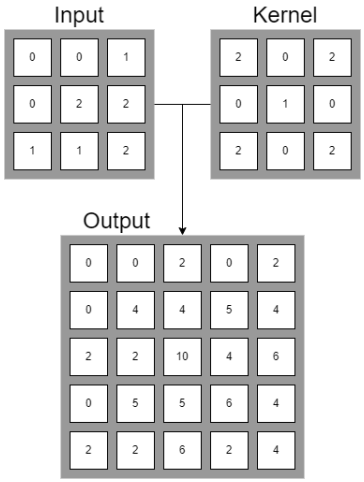


Figure 19: Example of transposed convolution given a (3x3) input and (3x3) kernel with stride 1, producing a (5x5) output.

Image inpainting

Inpainting refers to techniques used to restore damaged or severely deteriorated images. This can be the restoration of old scanned images with physical holes in them, but also images, which has been weathered with color deterioration. This means in general inpainting refers to methods, which restores damage to an image, in order to return the image back to the original state. In the realm of digital images, inpainting most commonly refers to the task of filling missing or invalid pixels on an image.

As inpainting has become more powerful using deep learning techniques, the method has also been used as an editing tool, used to remove unwanted objects from images by masking the objects and having the algorithm fill in the masked region, in effect removing the object from the image.

Non-learning methods

To fill in missing or invalid pixels, the simplest solution would be to copy a patch from the same or a similar image and paste to patch into the missing area. This however comes with the drawback, that the filled region is unlikely to be correct, and a search algorithm, which searches a large directory of images and computes similarity to match the missing area are likely to be very time-consuming and lacks the ability make semantically aware patch selections.

These non-learning approaches can only handle small or narrow holes in the image, where the color and texture variance is small. Un-

der these conditions several methods are able to perform remarkably well, such as in [Bertalmio et al., 2000] and [Telea, 2004], the latter is a method implemented in the Open CV library. With larger holes in the image or missing regions with large variance in color or texture, it is likely that the inpainting will cause blurring or artifacts in the filled region, such as Voronoi regions.

Deep learning methods

The deep learning methods for inpainting generally falls within two categories; convolutional networks with an adversarial encoder-decoder architecture and networks based on the u-net architecture. [Suvorov et al., 2021] A common concern with deep learning inpainting models are their ability to understand and reproduce the local and global context of the image. In order to train the network to account for this context several techniques have been proposed and tested with a variety of results.

Some examples are the introduction of dilated convolutions, which is a convolutional operation, that skips consecutive pixels, these can be used in general adversarial networks (GANs) which often use two discriminators, which then attempts to distinguish between real and generated data. Another development is the implementation of attention mechanisms, which you would normally see in translation and speech recognition networks, these range from general contextual attention to self attention[9] via Fourier transform. Other approaches split the network into parts and have each part learn different features allowing the networks to specialise, example one creates a course result and another network improves the restoration with fine detail. [Yu et al., 2018b]

Lastly in [Nazeri et al., 2019] the network tries to add more context to the masked region, by trying to detect edges in the image, reconstruct plausible edges and reconstruct the masked region based on the plausible edges under the mask.

[9] self-attention is often seen in networks, which process sentences. The self-attention module will look at every word in the sentence compared to every other word and re-weight the word embedding in order to include context, [Vaswani et al., 2017] self-attention for an inpainting network would instead of words compare the pixels in order to figure out the context.

Digital Elevation Models

A digital elevation model (DEM) is a digital cartographic dataset, which represents a continuous topographic elevation surface split into cell, where each cell represents the elevation on the z-axis at a given location on the x and y-axis. A DEM is commonly also referred to as a digital terrain model (DTM), this is identically with the DEM a representation of the bare earth, which is void of features, such as vegetation and buildings. The counterpart to the DTM is the digital surface model (DSM), which does contain vegetation and buildings.

DEMs are commonly build using remote sensing data [Abrams et al., 2020] and validated using ground surveying with GPS measurements. [Tighe and Chamberlain, 2009] Remote sensing data can be several things at once:

Stereogrammetry: This is a technique, in which photos are taken from aircrafts or drones at several different angles. Overlapping these images gives slight visual difference, which gives the images a sense of depth. From these overlapping images 3D information can be extracted and used to construct the DEM. This method can also use infrared imagery, the problem with this method is that visual and infrared light waves can not penetrate cloud coverings and can not be used at night.

Radar interferometry: This is a technique, which uses radar signals to infer the elevation of the terrain. Interferometry gathers elevation data by making several successive passes of the earth, measuring the same points more than once. These measurements are taken from either a space- or airborne synthetic aperture radar (SAR). The elevation data is captures by emitting a pulse of radar energy towards the surface and capturing the pulse again after being reflected back hitting sensors onboard, these sensors captures the amplitude and phase, when comparing these values at successive passes and computing geometric corrections a high quality DEM can be produces. This methods is used by the Shuttle Radar Topography Mission (SRTM), this technology is accurate within a few centimeters and can operate during bad weather and at night. [Farr et al., 2007]

LiDAR: Finally this technique involves installing a laser altimeter on either an aircraft, drone or satellite. The laser altimeter then emits a series of short laser lights in a sweeping motion, and the device then counts the time it takes for the laser light to be reflected. This procedure creates a laser dataset (LAS), from which a continuous raster dataset for elevation can be computed using the speed of light and interpolation.

The validation of these DEMs can then, as mentioned, be performed by conducting ground surveys of points on the ground and comparing these points to the points in the DEM being validated. Common validation metrics includes RMSE, NSSDA (95%)[10], standard deviation and mean. When measured on different terrain categories, such as slopes, shrubs and evergreen these metrics gives a good sense of the DEM quality. [Tighe and Chamberlain, 2009]

[10] National Standard for Spatial Data Accuracy at 95% confidence.

Materials and Methods

One goal of this thesis is to, using the elevation data from DEMs, predict the volume of the glaciers found in DEMs. For this we need DEMs and a method of identifying where in the DEM a glacier is present. Identifying the glaciers in the DEMs can be done using the Randolph Glacier Inventory, [Consortium, 2017] which contains the latitude and longitude of the glacier centers and a shapefile[11], which can be used to show the outline of the glaciers as well as the outline of every nunataks.[12]

[11] File type containing geometry defining the shape of an object.

[12] Ice-free areas within glaciers, commonly referred to as glacier islands.

In order to evaluate the inpainting models, we also require benchmarks to compare the volume predictions, for this we will be using the results from four other models, which have been mentioned in the related works section and summarised in [Farinotti et al., 2019]. Additionally for evaluating the models at points in the glacier we use real world observation from portable radar measurements from sled and helicopter, much like, how a DEM is validated in the section above.

Data acquisition and description

The **DEM** used to create the training and validation dataset for the models is sourced from the Advanced Spaceborne Thermal Emission and Reflection Radiometer (ASTER), which is an instrument launched onboard the NASA Terra spacecraft in December 1999, this instrument uses it's stereoscopic capabilities to gather data. [Abrams et al., 2020] The current version of the DEM in parts generated by this instrument is called the ASTER Global Digital Elevation Model (ASTER GDEM) version 3. This DEM has been generated from data gathered between March 1, 2000 and November 30, 2013. by stacking all cloud-masked scenes with the non-cloud-masked scenes from the period and applying various algorithms to remove abnormal data. This approach is not always enough to create a high quality DEM, hence existing DEMs are used to replace anomalies caused by the lack of source data. [Abrams et al., 2020].

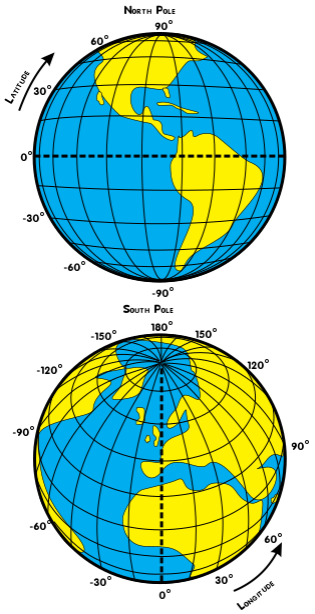


Figure 20: Latitude and Longitude lines. Image: Djexplo - Wikimedia Commons, public domain.

The DEMs are sourced as 3601 x 3601 tiles, which corresponds to 1 degree by 1 degree with a resolution of 30m (1 arc-second). The x and y-axis are denominated in latitude and longitude, and the tiles uses the EPSG:4326 projection which is equal to the World Geodetic System (WGS) 1984, known as WGS84. A consequence of this reference system is that every pixel in the DEM is not a perfect 30 by 30 meter square, but rather a rectangle, where the longitude (y-axis) is

an almost constant 30m and the latitude (x-axis) is based on the great circle distance, which can be calculated as $\texttt{cos}(\pi/180 \cdot \texttt{longitude}) \cdot 30$ this means a pixel at longitude 47 would be $30m$ by $20.45m$. Notice in figure 20 the sides of the squares changes in length depending on the latitude. Another more accurate way to describe the DEM tiles are that every tile is 1 degree by 1 degree, since every tile is 3601x3601 every pixel within the ASTER GDEM is 1/3600 degrees by 1/3600 degrees.

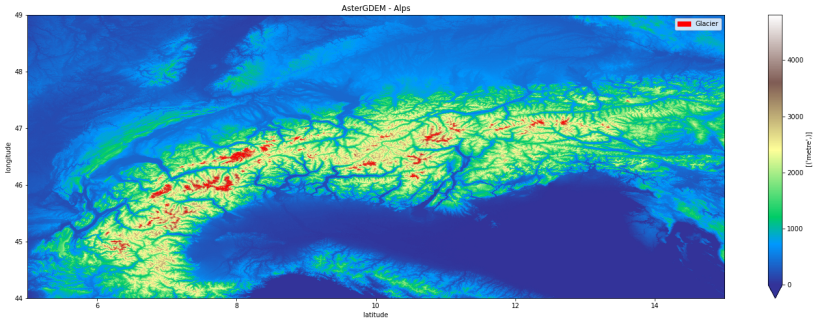


Figure 21: mosaic of the 50 DEM tiles of the Alps with glaciers in red.

For this thesis a total of 176 tiles have been sourced and assembled into three DEM mosaics, from which we can sample data. We use a DEM mosaic of the Alps comprised of 50 tiles, which is the primary area of study, a DEM mosaic of the Himalayas in Kyrgyzstan comprised of 60 tiles and a DEM mosaic of the Himalayas in Tajikistan comprised of 66 tiles. The mosaic DEM, which is the main area of study can be seen in it's complete state, with glaciers highlighted in figure 21.

The ASTER GDEM v3 does not perfectly express the height at any point and is subject to a not insignificant amount of error. A validation study using GPS benchmarks, performed over several location in the United States, found that the ASTER GDEM has a mean error of $-1.20m$ with a standard deviation of $8.44m$, this shows a negative bias in the GDEM overall. The RMSE of the GDEM is $8.52m$ with an LE95 of $16.70m$. This means that when evaluation the inpainting result of the models, a consideration of this error should be made.[Gesch et al., 2016]

The glacier and nunataks outlines have been sourced from the **RGI dataset** through the OGGM framework. For this thesis we use region 11 covering the Alps and region 13 covering the northern part of the Himalayas, these regions are illustrated in figure 22. With this dataset, we can create masks of the glaciers and avoid the glaciers

when sampling for training data. Additionally this dataset also contains information such as the area of the glacier in km^2, the minimum and maximum elevation of the glacier and several ids, which can be used to link the glaciers to other datasets, such as observations made by survey teams.

From the RGI dataset we extract 3865 glaciers from a total of 3927 glaciers from the Alps, 62 glaciers were removed because they were larger than a 256x256 image, which means no meaningful inpainting could be performed. We split these glaciers into seven categories, for the purpose of measuring performance based on glacier size, since inpainting performs noticeably differently based on the size of the masked region of the image.

Class	Area	Total
I	<0.05	1561
II	0.05 - 0.1	585
III	0.1 - 0.5	1056
IV	0.5 - 1.0	282
V	1.0 - 2.0	190
VI	2.0 - 5.0	137
VII	5.0 - 10.0	50
VIII	>10.0	4
Total		**3865**

Table 1: Glacier classes based on their total surface area, notice that the number of small glaciers outweighs the number of large glaciers by magnitudes.

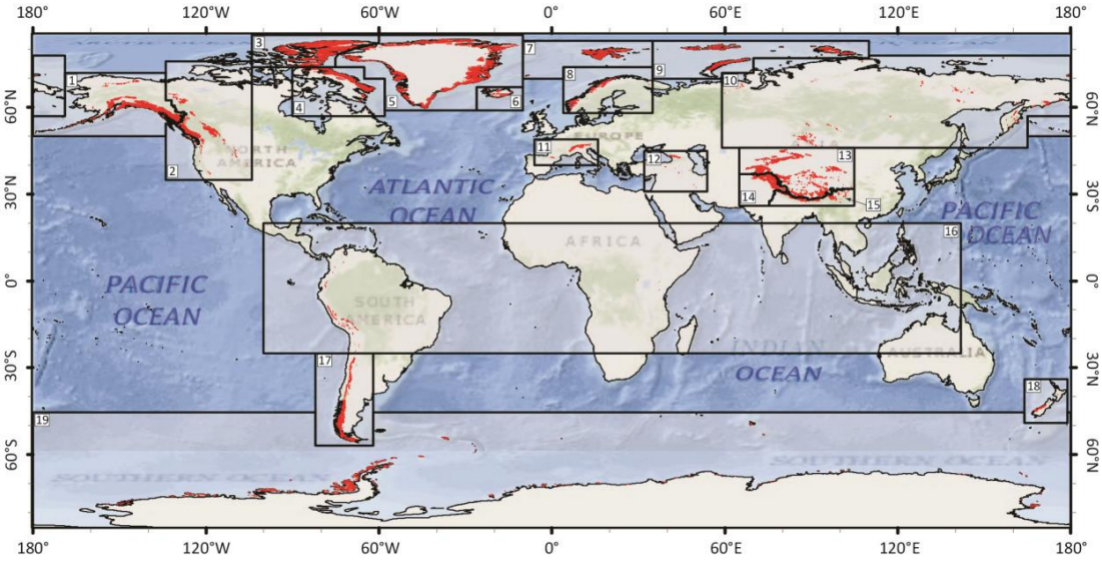


Figure 22: Regions of the Randolph Glacier Inventory. [Consortium, 2017]

The data from **other models** have been sourced from the paper [Farinotti et al., 2019] summarizing five different models, of which four have predictions, that match our area of study, the Alps. The results from these five models are available from the supplemental material. The results from the models are given in a masked format, where every glacier is a mask of the glacier shape, where every pixel is the predicted height of the glacier. The results are given in EPSG:32633 (WGS84) format at a 25m resolution, and since the format is geometrically equivalent for every pixel, the area of the glacier is given by: $25 \cdot 25 \cdot M$, where M is a count of every pixel within the prediction, which is inside the glacier outline and the volume is given by: $25 \cdot 25 \cdot H$, where $H = \texttt{sum}(\texttt{pred})$. An example of a prediction by model 1 of 5 is shown in figure 23.

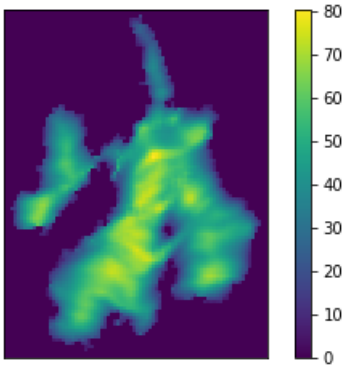


Figure 23: Height prediction of RGI60-11.0047 made by model 1 [Huss and Farinotti, 2012].

Lastly, we use **real ice thickness observations** from [Welty et al., 2020], using these observations we can perform validation on the glacier height predictions. This dataset contains measurements of many glaciers, from this, we can extract 68 glaciers, which are glaciers from the Alps with 328.094 total observations, which translates into 12.117 DEM pixels. Since many observations are taken within a 30 by 30 meter square, we find the minimum, average and maximum thickness for each pixel.

Training data and Pre-processing

The training data used by the models are created from a random sampling of the three large mosaic DEMs. The sample patches of the DEM are 256x256, and each individual patch is only allowed to overlap with other patches by a small margin. If a patch is sampled and contains any parts of the glacierized region the patch is dropped, since we don't want the models to learn anything about the terrain, which contains glaciers. In order to not sample parts of the DEM, which does not contain any mountains, an additional condition is enforced, which is that the middle of the image, which is a 128x128 box must have an average high, which is higher than the lowest glacier altitude. By enforcing this condition, we hope to constrain the knowledge of the model to mountain sides. The patches, which have been sampled from the Himalayas have additionally been constrained to contain no pixels, which have values higher than the highest value of the Alps mosaic DEM. In total there are $\approx$ 20.000 patches from the Alps, $\approx$ 100.000 patches from the Himalayas in Kyrgyzstan and $\approx$ 100.000 patches from the Himalayas from Tajikistan. Given a total of $\approx$ 220.000 training samples, of which 10% are used for validation. In figure 24 the sampling area of the Alps has been visualised, notice that we can clearly see that the sampling avoids the flat areas of the DEM and the areas, which contains glaciers, as seen in figure 21.

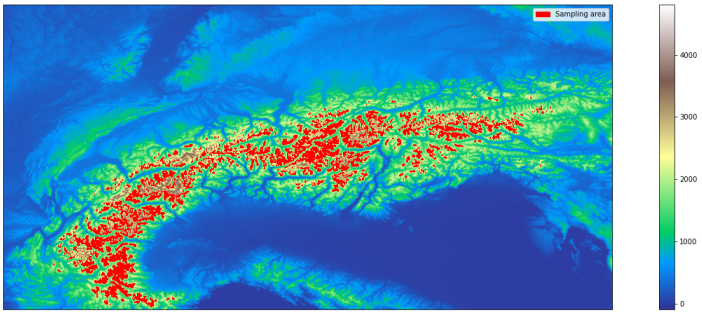


Figure 24: Sampling area of the training data from the Alps. Total of 22.000 samples extracted from DEM in figure.

Mask generation:
On order to train the models, we require masks, that we can apply to the DEM patches. These masks can be generated in a myriad of ways, but for this thesis we have selected two methods, first a random mask consistent of a large box area with brush-strokes covering between 25 and 30 percent of the original DEM patch. An example of such a patch can bee seen in figure 25. These masks promotes that the model learn the underlying terrain in it's entirety, since the mask can occlude entire valleys and mountains in the DEM patch.

Figure 25: Example of a randomly generated box and brush stroke mask.

Another approach to masks is to attempt and generate realistic glacier masks, that mean masks, which generally are glacier sized, glacier shaped and placed in places, where it is plausible a glacier could be found. For this a segmentation model using the u-net architecture, has been trained on the real glaciers and then tasked with segmenting glaciers in every DEM patch of the training set. In order to ensure that the generated glaciers adhere to our understanding of where glaciers can exists on mountains a simple rule has also been implemented, which is, whether true or not, that glaciers does not appear on mountain tops or global mountain ridges. With this rule we erase any mask segments that overlap these features. The features themselves have been identified by performing a Gaussian blur over each individual patch[13], then splitting the images into vertical and horizontal lines, and finding the local maxima within every line and drawing them as a mask, which is subtracted from the segmented mask of glaciers. Using these masks, as seen in figure 26, the goal is to teach the model how to predict mountain sides, which has been deemed to be plausible glacier locations, while avoiding teaching the model how to create mountain tops or ridges. In figure 27 we can see, how the real glaciers does not significantly overlap the tops or ridges in the DEM.

[13] The reason we apply a Gaussian blur is to remove small local maxima, since a glacier could overlap these features.

Figure 26: Example of a mask generated by the glacier segmentation model, showing a highly realistic glacier mask.

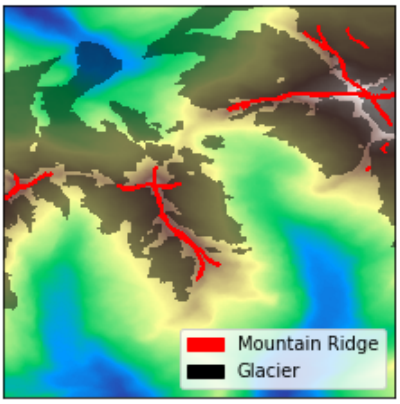


Figure 27: Example of a real glacier mask in black and mountain ridge in red, showing that the real glaciers have very little overlap with the ridges and tops of the mountains.

Partial Convolution Network

The first model, that have been used for performing image inpainting in this thesis, is the model proposed in [Liu et al., 2018]. This model uses the u-net architecture, with our DEM patches and masks as input. However every convolutional layer has been swapped with partial convolutional layers, and the up-sampling operations use nearest neighbor. The shortcut links in the model concatenates two DEM patches and masks together, which is used as input for the next layer. The last layer of the model concatenates the original DEM and mask, which allows the model to copy non-hole pixels and removes the need for extensive post-processing of the model output. The implementation in this thesis has been modified to work with the file format used to store every DEM patch and reduced to take images of the size 256x256 instead of 512x512 as used in the paper, this allows for the model to run with a reasonable batch size on consumer hardware. A graph of the model can be found in the appendix, figure 49, and an overview of the modified architecture used for this thesis is given in figure 28.

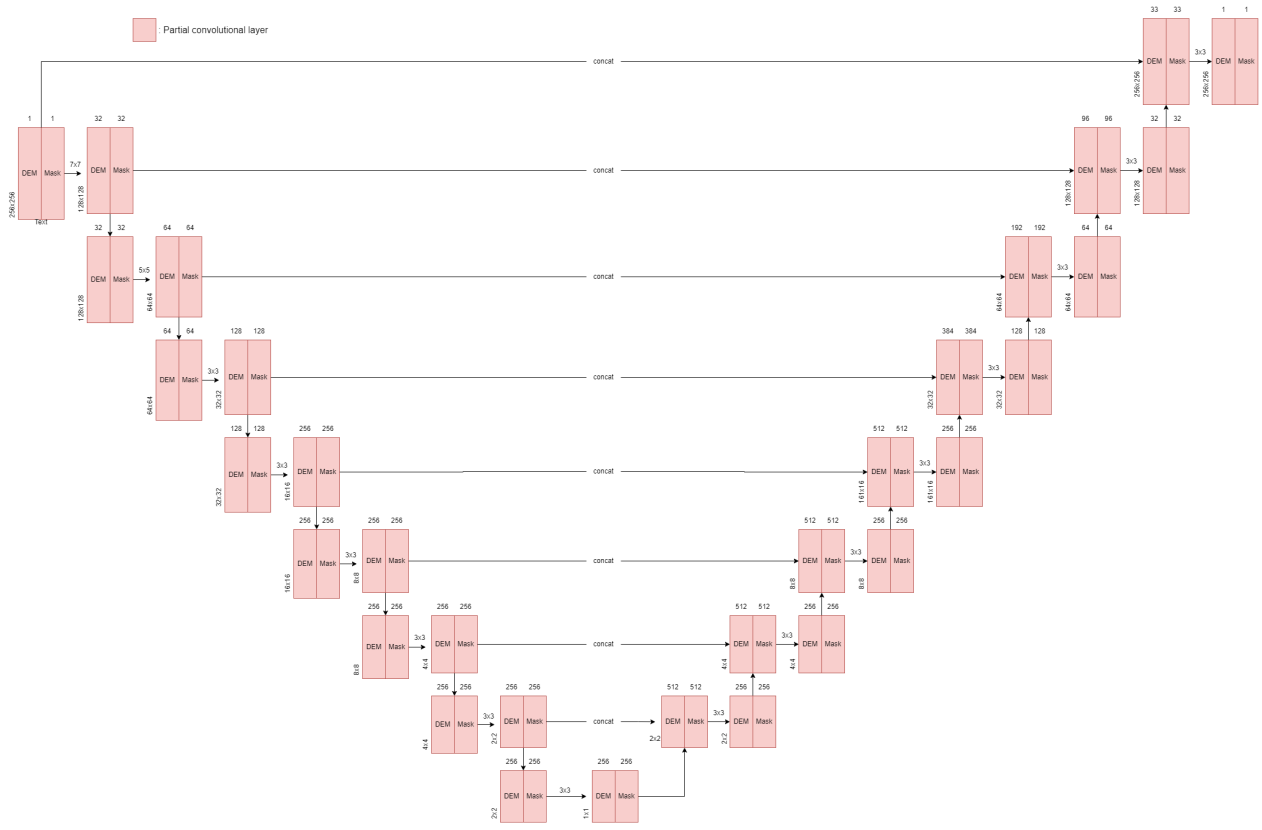


Figure 28: Modified architecture of the partial convolutional network creating a 256x256 input and 1x1x256 bottleneck.

The partial convolutional layer refers to the partial convolution operation and the mask update function. Let $\mathbf{W}$ be the convolutional filter weights for the convolutional filter and b be the bias. Let $\mathbf{X}$ be the feature values for the convolutional window and $\mathbf{M}$ be the binary mask, then the operation can be expressed as:

$$x' = \begin{cases} \mathbf{W}^T(\mathbf{X} \odot \mathbf{M}) \frac{\texttt{sum}(\mathbf{1})}{\texttt{sum}(\mathbf{M})} + b, & \text{if } \texttt{sum}(\mathbf{M} > 0) \\ 0, & \text{otherwise} \end{cases}$$

where $\odot$ is element-wise multiplication and $\mathbf{1}$ is an array containing only ones, with the same shape as a $\mathbf{M}$. The output is determined by the unmasked values and a scaling factor $\texttt{sum}(\mathbf{1})/\texttt{sum}(\mathbf{M})$, which is in place to appropriately scale adjust for the number of valid pixels. This operation can be seen in figure 29, in which the operation has been applied to a 5x5 image, with a masked 3x3 region in the middle and a 3x3 kernel. After each partial convolution operation the mask is updated by the following criteria:

$$m' = \begin{cases} 1, & \text{if } \texttt{sum}(\mathbf{M} > 0) \\ 0, & \text{otherwise} \end{cases}$$

This mask update is also illustrated in figure 29, which shrinks the mask, increasing the number of valid pixels for the next layer.

The partial convolutional network uses a composite **loss function** with four terms, which all cover different desired properties of the reconstruction. The first term is the L1 loss, which is a loss term, which ensures the pixel-wise reconstruction accuracy, given an image with a masked hole $\mathbf{I}_{in}$, an initial binary mask $\mathbf{M}$, the network prediction based in the input and mask $\mathbf{I}_{out}$ and the ground truth image $\mathbf{I}_{gt}$, we can define the first loss term as:

$$L_{hole} = \frac{1}{N_{\mathbf{I}_{gt}}} ||(1 - M) \odot (\mathbf{I}_{out} - \mathbf{I}_{in})||_1$$

$$L_{valid} = \frac{1}{N_{\mathbf{I}_{gt}}} ||M \odot (\mathbf{I}_{out} - \mathbf{I}_{in})||_1$$

where $N_{\mathbf{I}_{gt}}$ is the total number of elements in $\mathbf{I}_{gt}$. L_{hole} and L_{valid} denotes the L1 loss of the hole pixels and valid pixels in the prediction respectively.

The second loss term is the perceptual loss, also called the VGG loss, this loss term uses the pretrained VGG-16 model to extract features. The layers from the VGG-16 model that are used for this loss term are the three pooling layers. This loss term works by feeding the ground truth and filled image to the pretrained VGG-16 model and compute the L1 distance between the features at the pooling layers of the VGG-16. The perceptual loss is defined as:

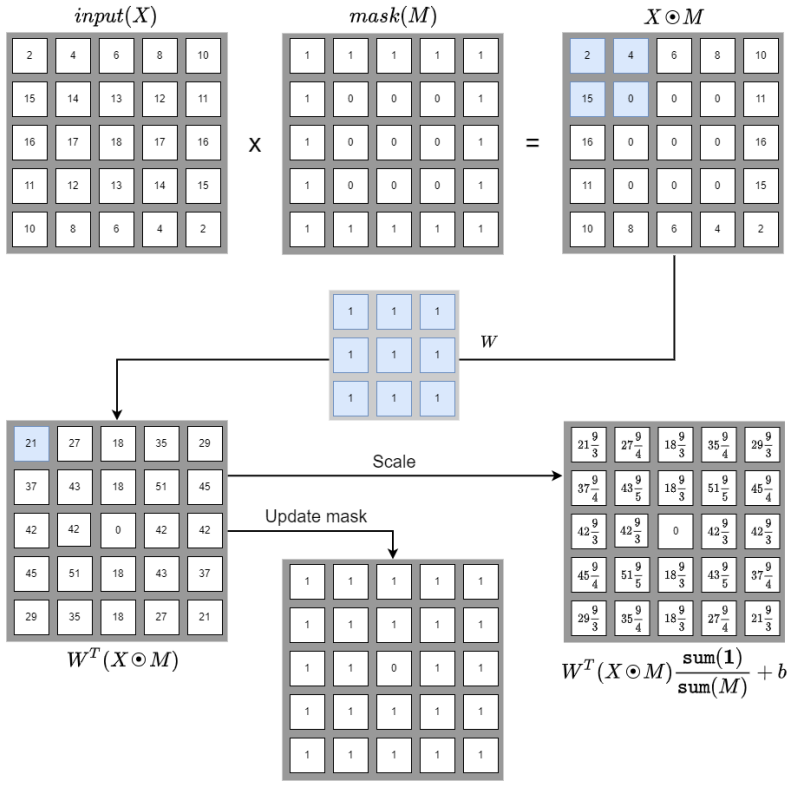


Figure 29: Partial convolutional layer example with $\mathbf{W} = (\mathbf{1})$.

$$L_{perceptual} = \sum_{n=0}^{N-1} ||\Psi_n(\mathbf{I}_{out}) - \Psi_n(\mathbf{I}_{gt})||_1 \quad + \sum_{n=0}^{N-1} ||\Psi_n(\mathbf{I}_{comp}) - \Psi_n(\mathbf{I}_{gt})||_1$$

where $\mathbf{I}_{comp}$ is the network prediction with valid pixels replaced with ground truth pixels and Ψ is the VGG-16 model. This loss function ensures that the predicted image is semantically close to the ground truth image.

The third loss term is the style loss, this loss term, like the perceptual loss also relies on the feature extraction from the VGG-16 model. The style loss is similar to the perceptual loss, however before calculating the L1 loss, we use a Gram Matrix to perform auto-correlation

on the features. The style loss is defined as:

$$L_{style_{out}} = \sum_{n=0}^{N-1} \frac{1}{C_n C_n} ||K_n((\Psi_n(\mathbf{I}_{out})^T \Psi_n(\mathbf{I}_{out})) - (\Psi_n(\mathbf{I}_{gt})^T \Psi_n(\mathbf{I}_{gt})))||_1$$

$$L_{style_{comp}} = \sum_{n=0}^{N-1} \frac{1}{C_n C_n} ||K_n((\Psi_n(\mathbf{I}_{comp})^T \Psi_n(\mathbf{I}_{comp})) - (\Psi_n(\mathbf{I}_{gt})^T \Psi_n(\mathbf{I}_{gt})))||_1$$

where $C_n C_n$ is the Gram Matrix, under the assumption that the high level features of the VGG-16 model has the shape $(H_n W_n) C_n$, and K_n is the normalization factor $1/C_n H_n W_n$.

Lastly, the forth loss term is the total variation loss, this loss term is used to ensure the smoothness of the final prediction. The total variation loss is defined as:

$$L_{tv} = \sum_{(i,j) \in R, (i,j+1) \in R} \frac{||\mathbf{I}_{comp}^{i,j+1} - \mathbf{I}_{comp}^{i,j}||_1}{N_{\mathbf{I}_{comp}}} + \sum_{(i,j) \in R, (i+1,j) \in R} \frac{||\mathbf{I}_{comp}^{i+1,j} - \mathbf{I}_{comp}^{i,j}||_1}{N_{\mathbf{I}_{comp}}}$$

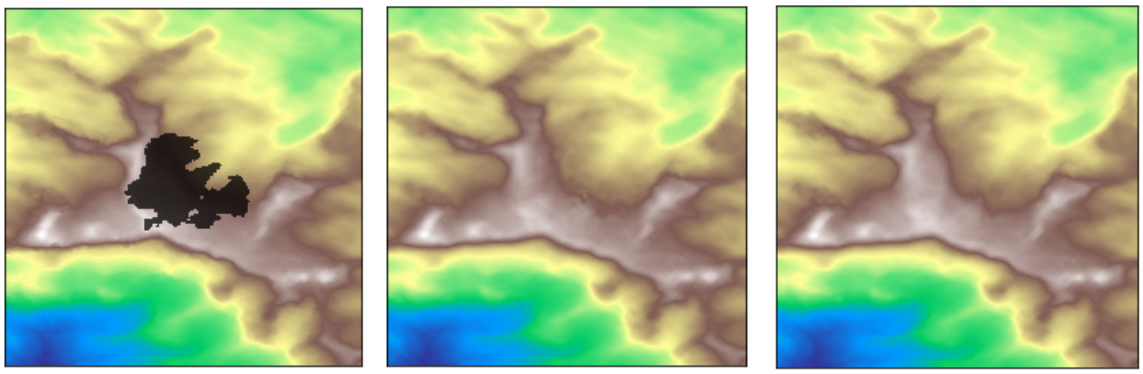
These four loss terms comes together into the composite loss function:

$$L_{total} = L_{valid} + 6L_{hole} + 0.05L_{perceptual} + 120(L_{style_{out}} + L_{style_{comp}}) + 0.1L_{tv}$$

The loss term weights were determined by the authors by performing hyper parameter search on 100 validation images. When re-purposing the model to perform inpainting on DEM patches, performing a new hyper parameter search could be a valid path of inquiry.

Model training: The partial convolutional network is trained in two stages; one initial stage and a fine-tuning stage. The reason two stage are necessary is, that the holes in the images presents a problem for batch normalization, since the holes causes problems for the mean and variance of the image. In order to still use batch normalization despite the presence of image holes is to first use batch normalization for the initial training with a learning rate of 0.0002, then afterwards the model is trained in a fine-tuning stage, where the batch normalization parameters in the down-sampling part of the u-net is frozen, but keep them enabled in the up-sampling part, since

the holes are likely to be filled it at this point in the network by the mask updating function. At the fine-tuning stage the learning rate is reduce further to 0.00005. The difference between the stages can be seen in figure 30. The training of the model is done with a batch size of 8, 70 epochs during the initial stage and 50 epochs during the fine-tuning stage with 1000 steps per epoch, this takes on average 18 hours on a consumer grade RTX3060 laptop GPU.



(a) (b) (c)

Figure 30: (a) The input image and mask, (b) initial stage of the model and (c) the fine-tuning stage of the model.

Gated Convolution Network

The second model, which is briefly visited in this thesis to perform inpainting on the DEM patches is the gated convolutional network. [Yu et al., 2018a] The framework for this model consists of two generator networks and two discriminator networks. The two generator networks are fully gated convoltional networks with diluted convolutions, which increases the receptive field of the model with the trade-off of skipping consecutive spacial locations. One generator is used for reconstructing a coarse result and one refines the coarse result. The discriminator used in this model is the SN-PatchGAN discriminator, which is a General Adversarial Network (GAN) discriminator, that uses diluted residual layers. Compared to the GAN, which looks at the image in it's entirety, the SN-PatchGAN looks at several local regions, and decides how real each region is independently. The architecture of the gated convolutional network can be seen in the figure 31 and a graph of the model is drawn in the appendix figure 51. This approach of having a network for coarse result and fine tuning is similar to the partial convolutional network. As seen in figure 31 the gated convolutional network uses a simple encoder-decoder framework instead of the u-net architecture in [Liu et al., 2018], the authors shows that with gated convolutional layers, diluted convolutions layers and contextual attention, the skip connections, which are common in the u-net architecture are not required,

that is because these layers are sufficient at carrying the information from the encoder over to the decoder.

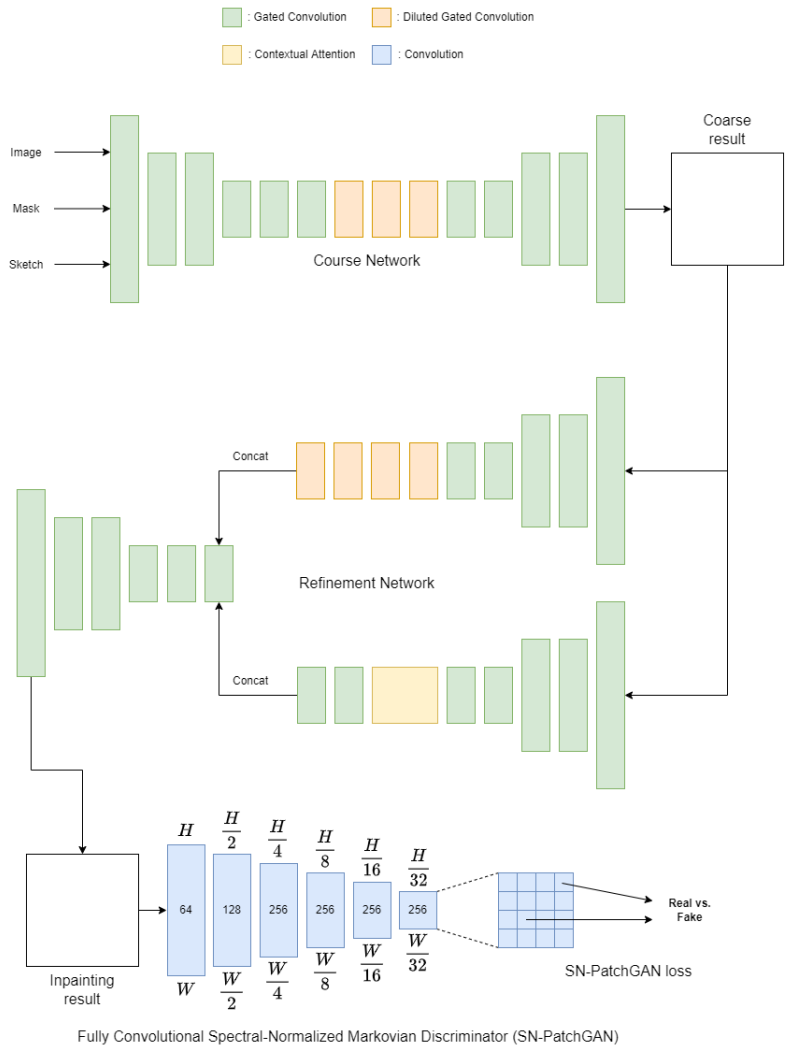


Figure 31: Complete overview of the gated convolutional network, source: [Yu et al., 2018a]

The gated convolutional network is in short an attempt to improve the partial convolutional network. To understand how gated convolution can improve the partial convolutional network, we must first identify, what part can be improved. In [Yu et al., 2018a] the authors points out, that the mask update function used as a core part of the partial convolutional network is un-learnable, hence a way to improve this is to make this update function learnable. This is done by creating a mask update formula, which is passed through a sigmoid activation function, allowing the network to learn the best rules to update the masks. The formula used for gated convolutions are

defined as:

$$\begin{aligned} \text{Gating}_{y,x} &= \sum\sum W_g \cdot I \\ \text{Feature}_{y,x} &= \sum\sum W_f \cdot I \\ O_{y,x} &= \phi(\text{Feature}_{y,x}) \odot \sigma(\text{Gating}_{y,x}) \end{aligned}$$

where ϕ can be any activation function, commonly ReLU, σ is a sigmoid activation function, which gates the mask values between 0 and 1 and the symbols W_g and W_f are convolutional filters.

Additionally one goal of the authors was to introduce user input, which can guide the result in certain directions, as such the partial convolutional layers where incompatible, since the rule based mask update, with emphasis on valid and invalid pixels, does not define how a user sketch should be treated. The difference between partial and gated convolution is illustrated in figure 32.

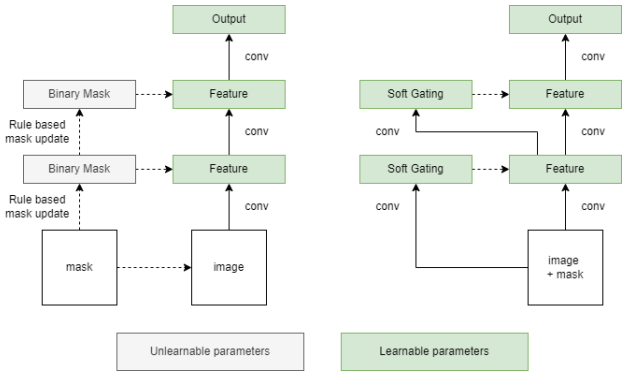


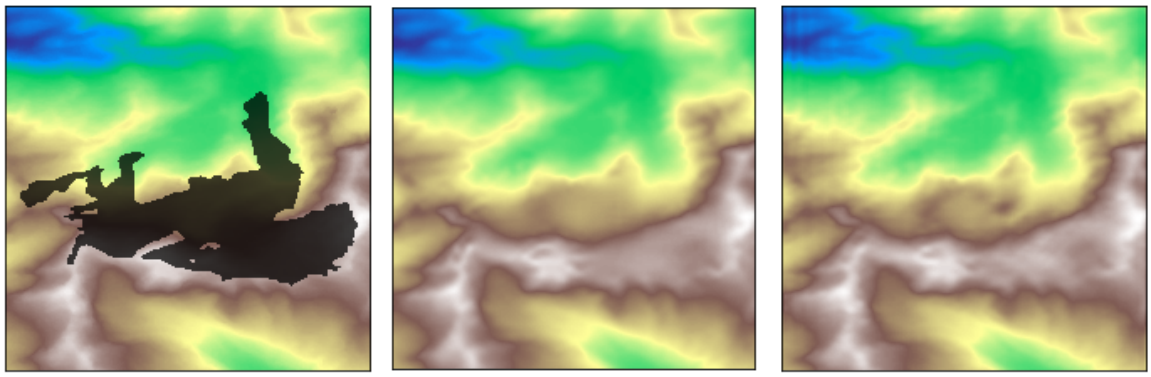
Figure 32: Difference betwwen the partial convolutional layer on the left and the gated convolutional network in the right, source: [Yu et al., 2018a]

The **loss function** used for the gated convolution network are two terms weighted equally. The first term is the pixel-wise L1 reconstruction accuracy and the second term is SN-PatchGAN loss. The first term, pixel-wise accuracy governs the accuracy of every pixel compared to the original and the second term is used to evaluate the resulting inpainted image, by attempting to spot the inpainted regions of the image, hence it works as a semantic quality check. This means if the inpainted image is indistinguishable from a real image the SN-PatchGAN loss will be low. The SN-PatchGAN loss is defined as the negative mean of the output of the discriminator, which is also called hinge loss.

Model training: The gated convolutional network is trained fully in one go, since the architecture contains two networks, compared to the partial convolutional network, which reuses the same network for two separate training processes. Training the model to a point of

diminishing returns takes on average 18 hours on a consumer grade RTX3060 laptop GPU.

This network, with the implementation utilized for this thesis, requires that the input image and mask have three channels and are scaled to 255. This is not a problem, since the one channel DEM can be broadcast into three identical channels, which causes the network to see the image as a normal image with gray colors, and the scaling is inconsequential, since we can move between different normalization schemes seamlessly.



(a) (b) (c)

Figure 33: (a) The input image and mask, (b) the first stage of the model producing a coarse inpainting result and (c) the second stage of the model refining the coarse inpainting result.

Contextual Attention

Contextual attention is a method used in the above gated convolutional network and is described in the paper [Yu et al., 2018b] as shown in figure 31. Contextual attention is a method, where the network learns, from where in the image background, it should borrow or copy features. Contextual attention is differentiable, which means it can be trained in the deep neural networks and fully-convolutional, which means the operations can be performed at different resolutions. Contextual attention uses two key operations to function; **Match and attend** and **Attention propagation**.

Match and attend is a solution to the problem, where we want to find and match a feature to a missing pixel. This is done by first extracting 3x3 patches from the image, that are not overlapped by the mask and reshape them as convolutional filters. We can then match the patches $f_{x,y}$ to the missing pixels $b_{x',y'}$, we then measure the normalized inner product to find the best match:

$$s_{x,y,x',y'} = \left\langle \frac{f_{x,y}}{||f_{x,y}||}, \frac{b_{x',y'}}{||b_{x',y'}||} \right\rangle$$

where $s_{x,y,x',y'}$ is the similarity of the patch centered in the back-

ground (x', y') and the foreground (x, y). In order to then compute the attention score a softmax activation function is applied to the similarity as: $\sigma_{x',y'}(\lambda s_{x,y,x',y'})$, where σ is the softmax activation function and λ is a constant value. Based on the resulting attention score the best extracted patches $b_{x',y'}$ is used as deconvolutional filters to reconstruct the missing pixels. [Yu et al., 2018b]

Figure 34: The colorwheel of the contextual attention layer, source: [Yu et al., 2018b]

Attention propagation is a method, that the authors claim can be used to improve the overall inpainting results and enrich the gradients during network training. The method is usually regarded as a fine-tuning tool for attention feature maps. The idea is that the neighboring pixel usually has a value close to each other. This means we can consider the neighboring pixels when computing the attention score:

$$\hat{s}_{x,y,x',y'} = \sum_{i \in \{-k,\dots,k\}} s^*_{x+i,y,x'+i,y'}$$

This kind of left to right propagation can be implemented as convolution with an identity matrix.

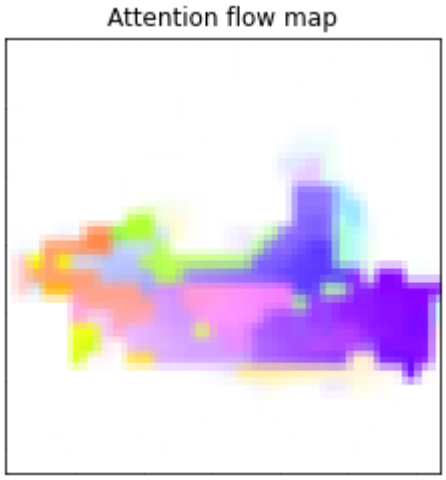


Figure 35: Visualization of the contextual attention layer when inpainting the glacier RGI60-11.01144.

Using the networks colorwheel, shown in figure 34, we can inspect the contents of the contextual attention layer and visualize where in the image the network borrows features in order to reconstruct the masked section of the image. As seen in figure 35 the network borrows features from all over the input image, but in particular the network uses features close to the mask edges, and a significant section of the mountain (purple and pink), which is used to fill in the mountain under the mask. The white section of the colorwheel implies that the pixel focused on itself, which means that the pixel under the mask is determined to be the best match.

Post-processing

In order to extract the estimated heights of the glacier from the inpainted DEM patches, we subtract the prediction from the ground truth image, this will result in a new image, that is zero outside of the masked region, and if the inpainting predicted a height lower than the ground truth, then we get positive values inside of the masked region. In order to convert this height map of the glacier into ice volume, the size of each individual pixel in the image is determined based on the longitude and latitude of the glacier in the image. This is because in the EPSG:4326 projection the pixels are not perfect 30x30m squares, the longitude is an almost constant 30m, but the latitude changed based on the longitude as visualized in figure 20, the conversion to meters is based on the great circle distance, which as previously mentioned is calculated as $\texttt{cos}(\pi/180 \cdot \texttt{longitude}) \cdot 30$.

Evaluation methods and metrics

During training: the performance of the network and intermediate inpainting result is evaluated using primarily the loss function of the network; in the case of the partial convolutional network, this is the loss L_{total} and for the gated convolutional network, this is the pixel-wise L1 loss and SN-PatchGAN loss, which are weighted equally.

The secondary metrics used to monitor the performance are the Structural Similarity Index Measure (SSIM) and Peak Signal to Noise Ratio (PSNR), these two metrics allows us to stop a network from training, if they do not increase as the loss decreases. SSIM is a popular quality metric, which measures the similarity between two images f and g, where 0 denotes that the images are entirely dissimilar to 1, which denotes that the images are entirely identical. SSIM is considered to be correlated the quality perceived by the human eye, which means a SSIM value close to 1 means the human eye could find it difficult to measure a reduction in quality between two images. SSIM accounts for three factors, which are, loss of correlation, luminance distortion and contrast distortion and is defined as:

$$\texttt{SSIM}(f,g) = l(f,g)c(f,g)s(f,g)$$

where

$$l(f,g) = \frac{2\mu_f\mu_g + C_1}{\mu_f^2 + \mu_g^2 + C_1}$$
$$c(f,g) = \frac{2\sigma_f\sigma_g + C_2}{\sigma_f^2 + \sigma_g^2 + C_2}$$
$$s(f,g) = \frac{\sigma_{fg} + C_3}{\sigma_f\sigma_g + C_3}$$

here $l(f,g)$ is the luminance distortion, which measures the closeness in mean luminance μ_f and μ_g between the two images, this is equal to 1 if $\mu_f = \mu_g$. $c(f,g)$ is the contrast distortion, which measures the closeness in contrast between the two images. The contrast is measured by the standard deviation σ_f and σ_g, this term is equal to 1 if $\sigma_f = \sigma_g$. Lastly $s(f,g)$ is the structure comaprison betwwen the two images, here σ_{fg} is the covariance between f and g, the constants C_1, C_2 and C_3 are used to avoid a null denominator in case of complete dissimilarity between f and g. [Horé and Ziou, 2010]

The second image quality term used is the PSNR, this term admittedly performs worse than other quality metrics [Q. Huynh-Thu, 2008], however the metrics ability to describe the presence of noise in an image is desirable in order to measure the amount of perceived noise in the reconstruction. The PSNR between two images f and g is defined as:

$$\mathtt{PSNR}(f,g) = 10 \cdot log_{10}\left(\frac{\mathtt{MAX}(f)^2}{\mathtt{MSE}(f,g)}\right)$$

where

$$\mathtt{MSE}(f,g) = \frac{1}{MN}\sum_{i=1}^{M}\sum_{j=1}^{N}(f_{ij} - g_{ij})^2$$

where MN is the size of both images f and g, and $\mathtt{MAX}(f)$ is the maximum possible value of the image. [Horé and Ziou, 2010] For example the max possible value of the DEM is 1, because the network normalizes every DEM locally to the range $[0, 1]$.

Post training: here we move away from metrics, that asserts the quality of the reconstruction to metrics that asserts the predicted volume extracted from the inpainting result both qualitatively and quantitatively.

First, we can inspect and compare the real world observations [Welty et al., 2020], which are georeferenced, such that we can compare the observed ice thickness from radar instruments and the thickness extracted from the inpainting result. With these real thickness observations and the predicted volumes, we can calculate the Root

Mean Squared Error (RMSE) in order to get a sense of the predicted volume when compared to real thickness observations. Since we only have real world observations from 68 glaciers, the extend of this evaluation is limited to those 68 glaciers. We can then choose to either analyse the glaciers separately or as a sum of all the glaciers.

Since an inpainting model, when not modified to behave otherwise, is capable of inpainting a result, which creates a negative volume. This thesis proposes that a metric for evaluating volume prediction by inpainting, would be the ratio between negative and positive volumes. If this ratio is higher that 0.5 it can imply one of two things: Either the model training has resulted in a model, which has learned to create mountains rather than overall mountainous terrain, the cause of which could be small masks, which primarily overlaps mountains tops. The other reason for a 1:1 ratio between negative and positive volumes, could be that the model has learned to reconstruct the terrain almost perfectly, but the glaciers blends into this terrain making it indistinguishable from the de-glacierized region, which can be caused by a low resolution DEM.

From the paper [Farinotti et al., 2019], which summarizes five other volume prediction models, we gain a large amount of insight into what we can call a consensus about the glacier thickness of all glaciers in the Alps (RGI60-11). From this we find that the average thickness of the ice across all glaciers in RGI60 region 11 is $61m$, this means we compare the average height of all glaciers after inpainting, and check if the thickness is close to the consensus. Furthermore we can compare the volume of every glacier with the volume of four of the five[14] consensus models.

[14] The fifth model does not have volume predictions for a majority of the glaciers in RGI60 region 11.

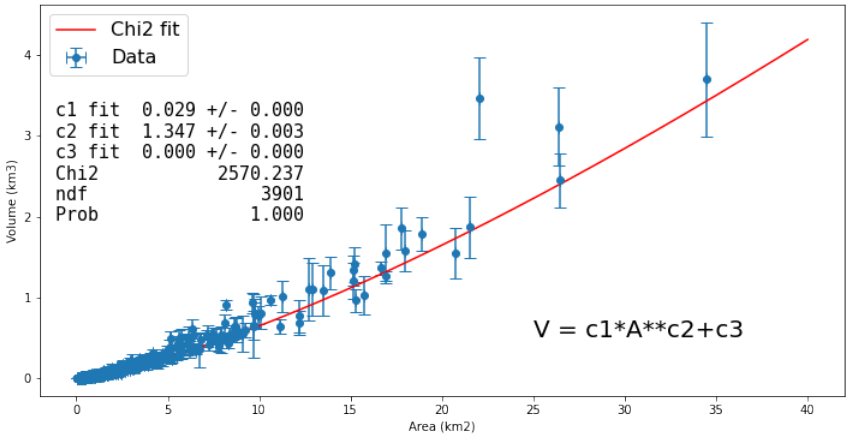


Figure 36: Chi2 fit performed on the relationship between the volume and area of the consensus models. This shows a scaled area scale formula $V = 0.029A^{1.347} + .0$ fits the data.

Lastly we find that a relationship between the area in km^2 and volume in km^3 exists between the consensus models, which is similar to the area scale formula, which is defined as:

$$V = A^{(3/2)}$$

Notice that a scaled version of the area scale formula for glacier volume estimation can be fitted to the consensus models with a high degree of explain-ability. This can be seen in figure 36.

For this thesis the models post training was evaluated on every method and metric above. Using the 68 glaciers, that we have known measurements from, we first check every point in the DEM against real observations, this will be the average, since many measurements have been taking within most of the 1/3600 degree by 1/3600 degree DEM pixels. The real observed ice thickness versus the predicted ice thickness can in this context be described by RMSE, and due to the presence of height uncertainty in the source DEM, another metric we can use are the percentage of points that are equal to their real world data point $\pm$10m.

After this we can compute the predicted ice volume of all 68 glaciers and check the number of negative volumes as well as the mean ice thickness, which is compared to the consensus models as well as the area scale formula.

Experiments

In the experimental section of this thesis, we will inspect the performance and capabilities of the two inpainting models in regards to height accuracy, in which we compared the inpainted results to observations made with radar devices. furthermore we inspect the ability of the networks to model the volume of glaciers in a DEM, when compared to the predictions of previous work in the field, which does not rely on inpainting. The training process of these models can be can be seen in figure 37 and 38 in which we can see the primary optimization metrics used by the models and their loss.

First we deploy the models with randomly generated box and brushstroke masks as shown in figure 25. These masks cover large sections of the DEM, which teaches the model of to generally reconstruct the mountain terrain. In figure 39 and 40 we can see five glaciers and the ice thickness behind a histogram, which shows the mean, std and a probability density function, constructed from this information.

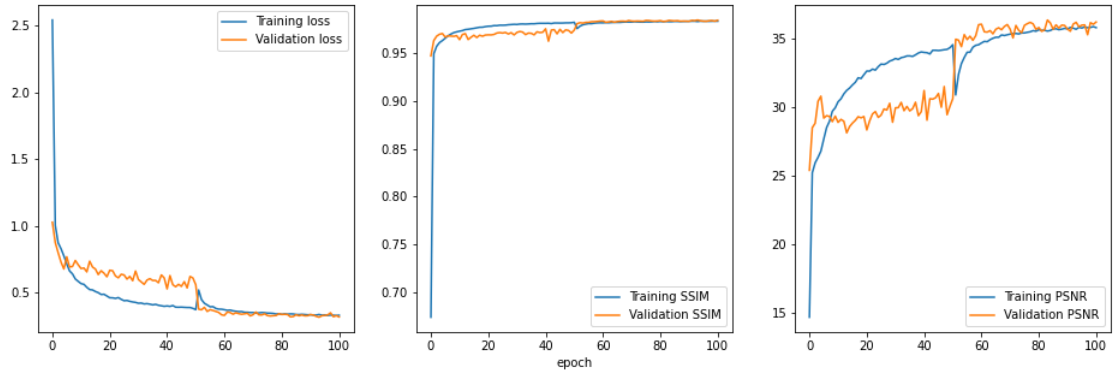


Figure 37: L_{total}, SSIM and PSNR during 120 epochs of training, the clear

ss

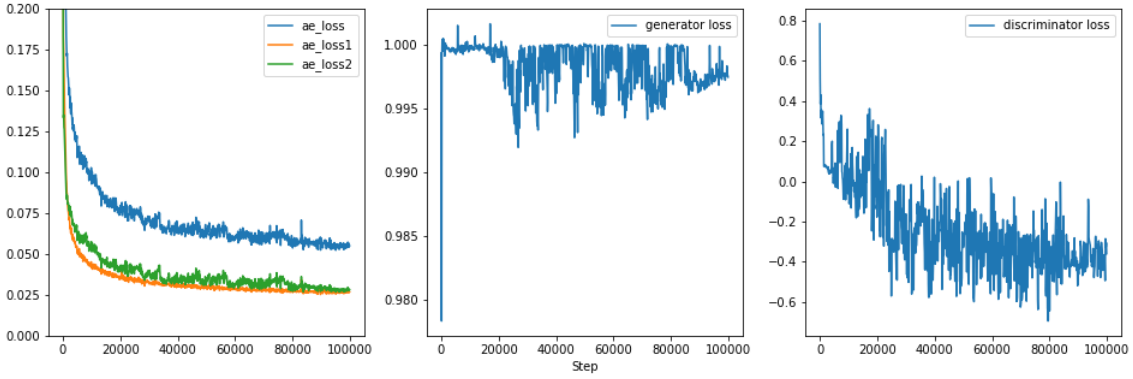


Figure 38: Loss functions during training of deepfillv2, which trains for

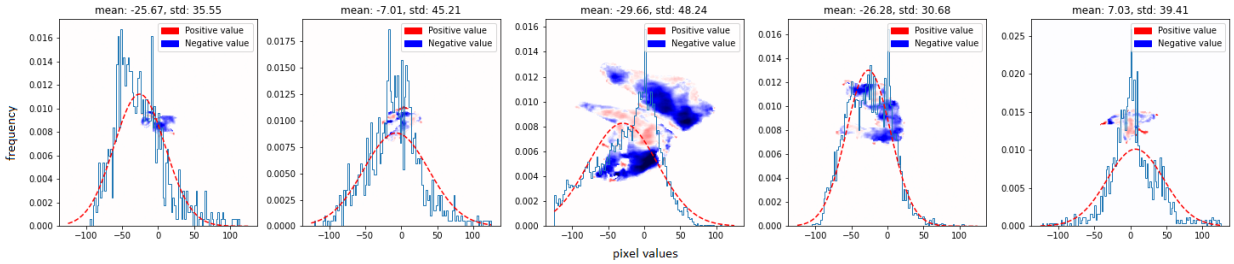


Figure 39: Height predictions from partial convolutional model.

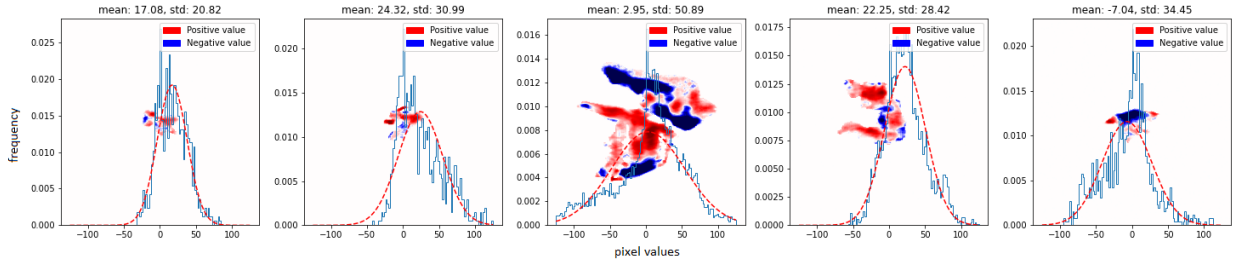


Figure 40: Height predictions from deepfillv2.

In these figures we can see, that the partial convolutional model creates ice thickness and volumes that are biased towards being negative, which is contrary to the deepfillv2 model, which is biased towards positive thickness and volume. However the two models does not inpaint the same glaciers as negative, hence a combination of the two, might be able to entirely eliminate negative volume predictions.

In table 2 we can see that the deepfillv2 model clearly outperforms the partial convolutional model both in terms of real radar observation accuracy and volumes. An ensemble of both models using majority vote could be used to maximize the number of positive pixels. One option, that we can use to completely eliminate negative volumes is to set all negative pixel prediction to zero, this is shown in the lower part of the table, in which we achieve the best results is regards to the consensus models and real radar observations, but the predictions are still far below the average of the consensus models. Notice that the increase in RMSE and correct observations caused by removing negative ice thickness from the thickness distributions is because the negative predictions are set to zero and some measurements are put into the 10m graze range.

	Volume			*Glathida*	
	Mean (m)	**NR[3]**	**Pred (km^3)**	**Correct[4]**	**RMSE**
Consensus	63.514	-	13.508	-	-
Deepfillv2	19.577	22.058%	3.542	15.426%	56.931
PConv	-12.732	70.588%	-3.025	9.476%	70.774
Ensemble[1]	3.422	41.176%	0.259	14.634%	67.313
Ensemble[2]	30.765	5.882%	5.753	17.757%	57.149
Exclude negative ice thickness:					
Deepfillv2	29.497	-	6.427	16.739%	54.165
PConv	13.159	-	2.264	13.469%	55.970
Ensemble[1]	16.241	-	3.414	16.957%	51.027
Ensemble[2]	35.934	-	7.300	18.673%	52.921

Table 2: Results from the two base models: Partial convolutional and Deepfillv2, as well as ensambles of the two compared to the consensus models.

[1] Ensemble using the mean of the two models.
[2] Ensemble using the majority vote for every pixel in the DEM.
[3] Negative ratio (NR) percentage of glaciers with a predicted negative volume.
[4] Number of observation, which are within 10 meters of the Glathida data points..

From the results of the partial convolutional model in particular, we can infer that the model has become biased towards creating mountains inside of the masked region, compared to valleys, which ideally we would prefer, since we can predict that the bedrock under a glacier must have a valley-like shape in order to contain the amount of ice, which has been previously estimated. As such, we can try and limit the models exposure to mountain ridges and top by avoiding these features in the DEM. The easiest method for avoiding these fea-

tures, would be to use masks, that mimic the shapes and placements of real glaciers. These masks can be made using a segmentation model trained on the real glaciers in the alps and the result of this can be seen in figure 41, which looks identical to any other glacier.

As clearly visible in table 3 the models trained on realistic glacier masks are noticeably worse than the base training masks. The primary factor behind this degradation is believed to be the size of the masks, which are significantly smaller than the box and brush stroke masks, and obscure consistently 25-30% of the image, compared to the segmented masks, which obscure $> 5\%$.

Figure 41: Example of a mask generated by the glacier segmentation model, showing a highly realistic glacier mask. Duplicate of figure 26

	Volume			*Glathida*	
	Mean (m)	**NR**	**Pred (km^3)**	**Correct**	**RMSE**
Consensus	63.514	-	13.508	-	-
PConv	-18.460	77.941%	-6.229	8.847%	65.548
Deepfillv2	-14-587	85.294%	-2.069	7.449%	71.745
Ensemble[1]	-16.524	85.294%	-4.151	8.117%	98.736
Ensemble[2]	-0.005	52.941%	0.289	12.669%	65.031
Exclude negative ice thickness:					
PConv	9.947	-	1.769	12.964%	61.927
Deepfillv2	8.857	-	3.249	11.993%	60.114
Ensemble[1]	7.517	-	1.951	12.343%	60.669
Ensemble[2]	15.927	-	4.191	15.955%	58.819

Table 3: Results from the two models trained in smaller masks, which are supposed to mimic the shapes and placements of real glaciers.

Adjust loss function

Since we are still seeing negative volumes, another option is to create a loss function term, which either penalises negative thickness predictions, or a term, which encourages, that the predicted volume during training is equal to the area scale formula $V = A^{(3/2)}$. The strongest response between the two models is the response from the partial convolutional network. From table 4 we can see that the more simple loss function adjustment, which creates a positive thickness bias (negative height bias) performs better than the more complex area scale formula. However as we can see, applying a penalty term to the loss function of the models is not a silver bullet.

PConv		***Volume***		***Glathida***	
	Mean (m)	**NR**	**Pred (km^3)**	**Correct**	**RMSE**
Consensus	63.514	-	13.508	-	-
Penalty **-box mask**	-6.763	63.235%	-1.059	12.809%	63.610
Area volume **-box mask**	-7.934	69.117%	-1.994	10.781%	66.460
Penalty **-segmented mask**	12.750	35.294%	1.675	12.793%	62.892
Area volume **-segmented mask**	-4.110	57.352%	-1.456	9.220%	69.854
Exclude negative ice thickness:					
Penalty **-box mask**	15.267	-	3.098	15.442%	53.362
Area volume **-box mask**	15.302	-	2.932	13.997%	53.564
Penalty **-segmented mask**	24.996	-	5.006	15.372%	60.239
Area volume **-segmented mask**	15.233	-	3.178	13.446%	60.934

Table 4: Experimenting with loss function penalties to steer the direction of the image inpainting performed by the partial convolutional network.

Guided image inpainting

Given that we have information about the real ice thickness at coordinates for our 68 glaciers, one avenue, that we could pursue, is the use of real data points to guide the image inpainting. This method relies on the real data, but the training of the models does not use any additional information, since we want the models to infer the bedrock, when given a few points inside the mask to guide the direction of the inpainting. Using base models, trained with box and brushstroke masks and models trained with segmented masks, we achieve the results as shown in figures 42 and 43 for the box and brushstroke mask and in tables 5 and 6. An illustration of the position of the points used to guide the inpainting in this experiment can be seen in the appendix figure 53

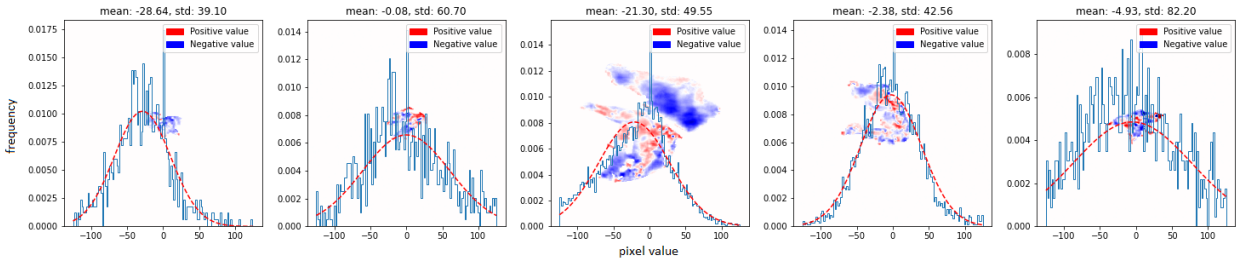


Figure 42: Height predictions from partial convolutional model with guided inpainting

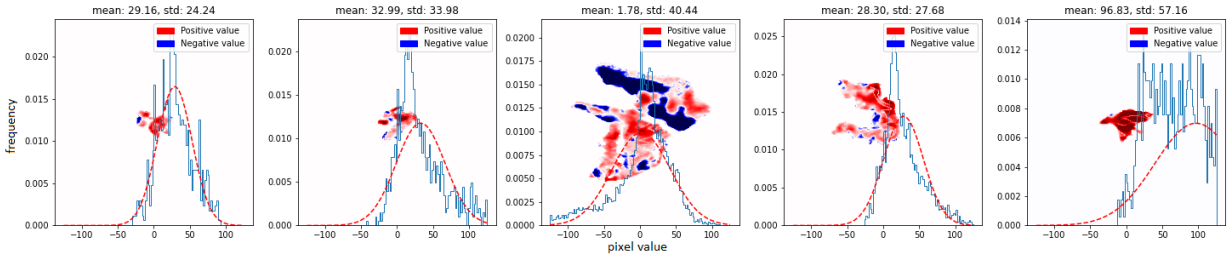


Figure 43: Height predictions from deepfillv2 with guided inpainting.

From figures 42 and 43 we can again see that the models behave differently. One characteristics, that seperates the models is that the partial convolutional model is allowed to change to values of the radar observations used to guide the inpainting, which causes a much more shallow volume prediction compared to the deepfillv2 model. This is also reflected in the RMSE and correct fields of tables 5 and 6.

Box and brushstroke mask	*Volume*			*Glathida*	
	Mean (m)	**NR**	**Pred (km^3)**	**Correct**	**RMSE**
Consensus	63.514	-	13.508	-	-
PConv	17.913	30.882%	4.088	16.786%	53.904
Deepfillv2	38.524	2.941%	7.949	100%	4.862
Ensemble^1	24.850	0.0%	5.567	30.332%	31.311
Ensemble^2	57.562	0.0%	10.468	52.447%	39.793
Exclude negative ice thickness:					
PConv	37.292	-	7.039	18.852%	46.962
Deepfillv2	42.143	-	8.936	100%	4.862
Ensemble^1	35.139	-	7.355	31.863%	26.304
Ensemble^2	59.424	-	10.985	52.447%	39.793

Table 5: Results of both models with guided inpainting from real radar points and models trained on box masks.

Segmented mask	*Volume*			*Glathida*	
	Mean (m)	**NR**	**Pred (km^3)**	**Correct**	**RMSE**
Consensus	63.514	-	13.508	-	-
PConv	13.349	29.411%	2.982	31.272%	27.609
Deepfillv2	6.049	44.117%	2.147	100%	4.862
Ensemble^1	9.699	35.294%	2.558	55.430%	14.419
Ensemble^2	25.480	11.764%	6.385	59.453%	22.492
Exclude negative ice thickness:					
PConv	24.850	-	5.567	32.189%	27.093
Deepfillv2	18.628	-	5.128	100%	4.862
Ensemble^1	20.003	-	2.558	55.694%	14.382
Ensemble^2	31.808	-	7.805	100%	4.862

Table 6: Results of both models with guided inpainting from real radar points and models trained on segmented masks.

Augment target

From the guided inpainting results, we can see that giving the inpainting process some information about the terrain under the glacier improves the overall volume predictions. The drawback of this procedure, is that we only have radar observations on the ice thickness for 68 glaciers, hence it will not work on the majority of the glaciers in the Alps. For this reason, we can attempt to generalise the information and apply the information to every glacier in the hope that it will improve the volume predictions for all glaciers, not just the once we have measurements for. Given the glacier masks, and the Glathida observations, we can craft a function, which describes the relationship between the distance to the nearest edge and the thickness of the ice. This relationship can be seen in figure 44. This figure also shows that correlation between the two, is not perfect, and any function we produce will be largely a generalization, because of the large variation at the same distances. Introducing more factors, such as the area of the glacier, does not increase the correlation.

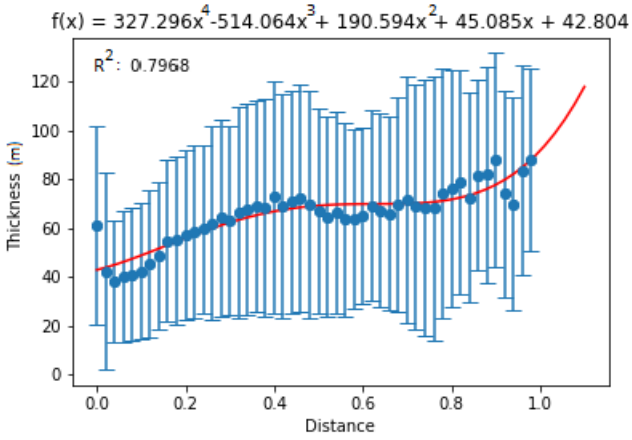


Figure 44: relationship between the distance to the nearest edge and the thickness of the ice.

In order to then apply this function to our models we augment the target of the models, which means the new target for the models becomes real_height - distance_function instead of the height in the DEMs. This augmentation can be applied directly to the training images. This augmentation also creates a clear and distinct feature in the DEMs, which the models can attempt to remove.

With the ensemble model[1], which takes the average ice thickness of both models, we very close to the consensus model average in terms of mean ice thickness and predicted total volume for the 68 glaciers. However, we can also see, that the inpainted result does not match with many of the Glathida observations. This could be interpreted as; the inpainted bedrock level is very wrong and the

	Volume			*Glathida*	
	Mean (m)	**NR**	**Pred (km^3)**	**Correct***	**RMSE**
Consensus	63.514	-	13.508	-	-
PConv	38.195	11.764%	7.190	13.220%	59.476
Deepfillv2	93.529	2.941%	22.395	8.342%	65.306
Ensemble[1]	65.862	1.470%	14.794	13.314%	75.657
Ensemble[2]	106.20	0.0%	24.718	8.676%	63.834
Exclude negative ice thickness:					
PConv	43.906	-	8.738	14.859%	58.752
Deepfillv2	98.737	-	23.640	9.189%	119.94
Ensemble[1]	68.833	-	15.521	14.098%	72.662
Ensemble[2]	107.30	-	24.983	9.135%	120.87

Table 7: Result of both models, run on the augmented dataset with segmented masks.

predicted mean ice thickness and total volume is entirely incidental. In figure 45 we can see five of the glacier ice thickness distribution produced by the ensemble model, and in figure 46 we can see the predicted volumes against the consensus average.

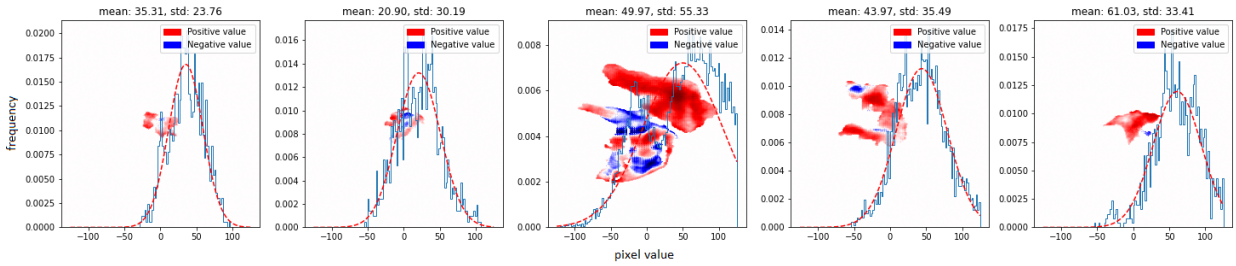


Figure 45: Ensemble ice thickness distribution.

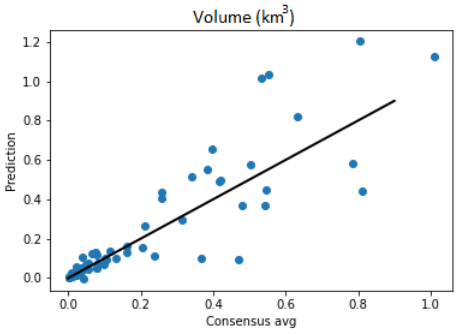


Figure 46: Consensus average volume vs. ensemble predicted volume.

Given that we now known how the models perform on the 68 glathida test glaciers, we can apply the models to the entire RGI60-11 region. We can expect that the generalised distance to thickness function might be too general for the 3000+ glaciers in the region, but

RGI60-11 glaciers	*Volume*		
	Mean (m)	**NR**	**Pred (km^3)**
Consensus	44.715	-	75.869
PConv	33.438	4.165%	55.086
Deepfillv2	66.133	6.248%	135.87
Ensemble^1	49.785	3.853%	95.480
Ensemble^2	75.559	0.520%	154.12
Exclude negative ice thickness:			
PConv	35.413	-	63.403
Deepfillv2	70.109	-	144.56
Ensemble^1	51.573	-	100.33
Ensemble^2	76.117	-	155.84

Table 8: Augmented target inpainting performed by both models on all glaciers in the RGI60-11 region.

we should still get into a reasonable range of ice thickness average and volumes. The results of this can be seen in table 8.

Finally we can visualise the predicted volume against the consensus average in figure 47. In table 8 we can see that again the ensemble model, which performs averaging between the two models performs reasonably well, but when run on all glaciers, we can also see that the partial convolutional network on it's own in producing reasonable results, when compared to the consensus models. When evaluation these results we should also consider, that the standard deviation between the volumes of the four consensus models is 11.112km^3. And the relative error inherent in the DEM tiles are around $\pm$10m.

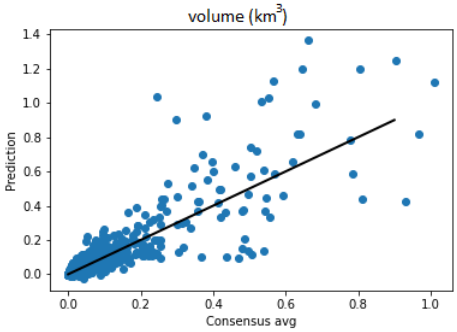


Figure 47: Consensus average volume vs. ensemble predicted volume.

Conclusion and Future work

In this thesis we have found, that the use of deep learning based state of the art image inpainting shows promise in the task of reconstructing the underlying glacier bedrock, which can be used to compute the total ice volume in a glacier. First we have shown, that the base

version of the partial convolutional network and deepfillv2 network is ill suited for inferring the bedrock underneath the glaciers. The reason behind this is believed to be because the glaciers are indistinguishable from the de-glacierized area of the mountainous terrain, meaning there is not a clear difference between the foreground and background of the DEM, from which the inpainting models can infer the bedrock. It is also unlikely that increasing the resolution of the DEM tiles, will resolve this problem, and increasing the DEM tiles to a 1-5m resolution would make the images of glaciers much too large for the models to perform consistent inpainting.[15]

[15] For perspective the image patches would have to be 8192x8192 to fit most RGI60-11 glaciers at a 1m resolution, and around 2048x2048 at 5m resolution.

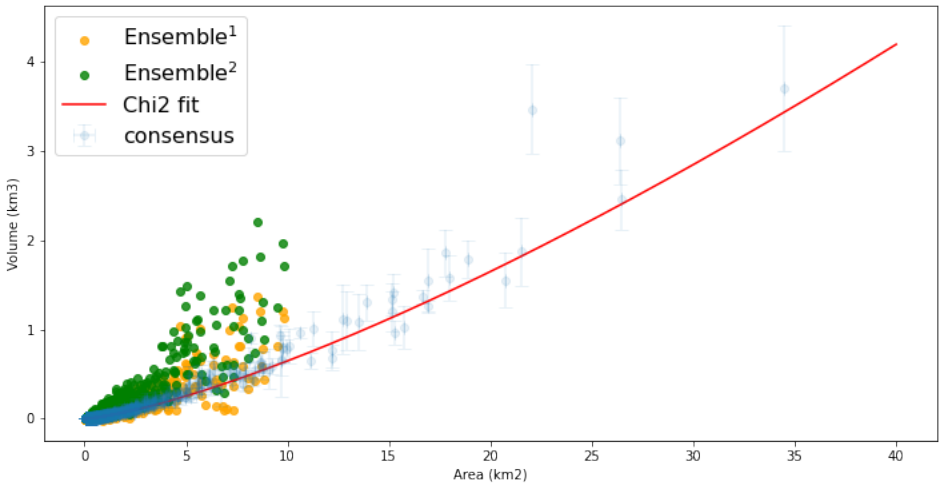


Figure 48: Chi2 fit of consensus models, with ensemble models using augmented targets.

We have further discovered, that introducing additional data to the models, such as the real world radar observations, measuring the ice thickness at different coordinates, can improve the capabilities of the models and nudge the inpainting in a direction, which starts to imitate the bedrock level under the glaciers. With ice thickness observations, we can alter the input data and masks without altering the base model. This means the model can function both with and without the additional ice thickness measurements. This can then be generalised and applied to all glaciers in the region, which also achieves a reasonable result when compared against the consensus models. The result of the two ensemble models using augmented targets compared to the consensus models and their correlation between volume and area can be seen in figure 48. As also seen in the figure, the majority vote ensemble model deviates more from the area to volume fit than the average ensemble model, but due to the exclusion of

very large glaciers, that would not fit inside a 256x256 patch, we do not get a real sense about how closely the models follow this chi2 fit. In general, it is safe to say that the volumes are overestimated, when compared to the consensus models. This overestimation could be resolved by a better "distance from edge to thickness" function with more radar observations, possibly from more glaciers.

Since the addition of real data seemingly improves the performance of the inpainting models, an avenue for further work would be to introduce meta data into the model, meta data could be a number of data points from the Randolph Glacier Inventory such as glacier elevation or information about glacier velocity and flowlines from OGGM. In this case, we should pick metrics, which we suspect are casually or directly linked to glacier ice thickness, for example we have an expectation that ice velocity is casually linked to the thickness of the ice, or maybe the elevation of the glacier can be linked to the thickness of the glacier ice in general.

References

Michael Abrams, Robert Crippen, and Hiroyuki Fujisada. Aster global digital elevation model (gdem) and aster global water body dataset (astwbd). *Remote Sensing*, 12(7), 2020. ISSN 2072-4292. DOI: 10.3390/rs12071156. URL https://www.mdpi.com/2072-4292/12/7/1156.

Marcelo Bertalmio, Guillermo Sapiro, Vincent Caselles, and Coloma Ballester. Image inpainting. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '00, page 417–424, USA, 2000. ACM Press/Addison-Wesley Publishing Co. ISBN 1581132085. DOI: 10.1145/344779.344972. URL https://doi.org/10.1145/344779.344972.

RGI Consortium. Randolph glacier inventory - a dataset of global glacier outlines, version 6. boulder, colorado usa. nsidc: National snow and ice data center. 2017. DOI: https://doi.org/10.7265/4m1f-gd79.

Ugur Demir and Gozde Unal. Patch-based image inpainting with generative adversarial networks, 2018. URL https://arxiv.org/abs/1803.07422.

Julia Eis, Fabien Maussion, and Ben Marzeion. Initialization of a global glacier model based on present-day glacier geometry and past climate information: an ensemble approach. *The Cryosphere*, 13:3317–3335, 12 2019. DOI: 10.5194/tc-13-3317-2019.

Daniel Farinotti, Matthias Huss, Johannes J. Fürst, Johannes Landmann, Horst Machguth, Fabien Maussion, and Ankur Pandit. A consensus estimate for the ice thickness distribution of all glaciers on earth. *Nature Geoscience*, 12(3):168–173, February 2019. DOI: 10.1038/s41561-019-0300-3. URL https://doi.org/10.1038/s41561-019-0300-3.

Tom G. Farr, Paul A. Rosen, Edward Caro, Robert Crippen, Riley Duren, Scott Hensley, Michael Kobrick, Mimi Paller, Ernesto Rodriguez, Ladislav Roth, David Seal, Scott Shaffer, Joanne Shimada, Jeffrey Umland, Marian Werner, Michael Oskin, Douglas Burbank, and Douglas Alsdorf. The shuttle radar topography mission. *Reviews of Geophysics*, 45(2), 2007. DOI: https://doi.org/10.1029/2005RG000183. URL https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2005RG000183.

Holger Frey, Horst Machguth, Matthias Huss, Christian Huggel, Samjwal Bajracharya, Tobias Bolch, A. Kulkarni, Andreas Linsbauer, and Nadine Salzmann. Estimating the volume of glaciers

in the himalayan–karakoram region using different methods. *The Cryosphere*, 8:2313–2333, 12 2014. DOI: 10.5194/tc-8-2313-2014.

J. J. Fürst, F. Gillet-Chaulet, T. J. Benham, J. A. Dowdeswell, M. Grabiec, F. Navarro, R. Pettersson, G. Moholdt, C. Nuth, B. Sass, K. Aas, X. Fettweis, C. Lang, T. Seehaus, and M. Braun. Application of a two-step approach for mapping ice thickness to various glacier types on svalbard. *The Cryosphere*, 11(5): 2003–2032, 2017. DOI: 10.5194/tc-11-2003-2017. URL https://tc.copernicus.org/articles/11/2003/2017/.

Dean Gesch, M Oimoen, Jeffrey Danielson, and David Meyer. Validation of the aster global digital elevation model version 3 over the conterminous united states. volume XLI-B4, 07 2016. DOI: 10.5194/isprsarchives-XLI-B4-143-2016.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL http://arxiv.org/abs/1512.03385.

Alain Horé and Djemel Ziou. Image quality metrics: Psnr vs. ssim. pages 2366–2369, 08 2010. DOI: 10.1109/ICPR.2010.579.

Matthias Huss and Daniel Farinotti. Distributed ice thickness and volume of all glaciers around the globe. *Journal of Geophysical Research*, 117:F04010, 10 2012. DOI: 10.1029/2012JF002523.

W.W. Immerzeel, A.F. Lutz, M. Andrade, A. Bahl, H. Biemans, T. Bolch, S. Hyde, S. Brumby, B.J. Davies, A.C. Elmore, A. Emmer, M. Feng, A. Fernández, U. Haritashya, J.S. Kargel, M. Koppes, P.D.A. Kraaijenbrink, A.V. Kulkarni, P.A. Mayewski, S. Nepal, P. Pacheco, T.H. Painter, F. Pellicciotti, H. Rajaram, S. Rupper, A. Sinisalo, A.B. Shrestha, D. Viviroli, Y. Wada, C. Xiao, T. Yao, and J.E.M. Baillie. Importance and vulnerability of the world's water towers. *Nature*, 577(7790):364–369, January 2020. ISSN 0028-0836. DOI: 10.1038/s41586-019-1822-y.

IPCC. Climate change 2021: The physical science basis. contribution of working group i to the sixth assessmentreport of the intergovernmental panel on climate change. 2021.

Guilin Liu, Fitsum A. Reda, Kevin J. Shih, Ting-Chun Wang, Andrew Tao, and Bryan Catanzaro. Image inpainting for irregular holes using partial convolutions. In *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997. ISBN 9780070428072.

Kamyar Nazeri, Eric Ng, Tony Joseph, Faisal Z. Qureshi, and Mehran Ebrahimi. Edgeconnect: Generative image inpainting with adversarial edge learning. *ArXiv*, abs/1901.00212, 2019.

M. Ghanbari Q. Huynh-Thu. Scope of validity of psnr in image/video quality assessment. *Electronics Letters*, 44:800–801(1), June 2008. ISSN 0013-5194. URL https://digital-library.theiet.org/content/journals/10.1049/el_20080522.

Raaj Ramsankaran, Ankur Pandit, and Mohd. Farooq Azam. Spatially distributed ice-thickness modelling for chhota shigri glacier in western himalayas, india. *International Journal of Remote Sensing*, 39:3320 – 3343, 2018.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015. URL https://arxiv.org/abs/1505.04597.

Roman Suvorov, Elizaveta Logacheva, Anton Mashikhin, Anastasia Remizova, Arsenii Ashukha, Aleksei Silvestrov, Naejin Kong, Harshith Goka, Kiwoong Park, and Victor Lempitsky. Resolution-robust large mask inpainting with fourier convolutions, 2021. URL https://arxiv.org/abs/2109.07161.

A.C. Telea. An image inpainting technique based on the fast marching method. *Journal of Graphics Tools*, 9(1):23–34, 2004. ISSN 1086-7651. DOI: 10.1080/10867651.2004.10487596.

Lorraine Tighe and Drew Chamberlain. Accuracy comparison of the srtm, aster, ned, nextmap® usa digital terrain model over several usa study sites. 01 2009.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017. URL https://arxiv.org/abs/1706.03762.

E. Welty, M. Zemp, F. Navarro, M. Huss, J. J. Fürst, I. Gärtner-Roer, J. Landmann, H. Machguth, K. Naegeli, L. M. Andreassen, D. Farinotti, H. Li, and GlaThiDa Contributors. Worldwide version-controlled database of glacier thickness observations. *Earth System Science Data*, 12(4):3039–3055, 2020. DOI: 10.5194/essd-12-3039-2020. URL https://essd.copernicus.org/articles/12/3039/2020/.

Jiahui Yu, Zhe Lin, Jimei Yang, Xiaohui Shen, Xin Lu, and Thomas S Huang. Free-form image inpainting with gated convolution. *arXiv preprint arXiv:1806.03589*, 2018a.

Jiahui Yu, Zhe Lin, Jimei Yang, Xiaohui Shen, Xin Lu, and Thomas S Huang. Generative image inpainting with contextual attention. *arXiv preprint arXiv:1801.07892*, 2018b.

Appendix

Partial convolution network model graph

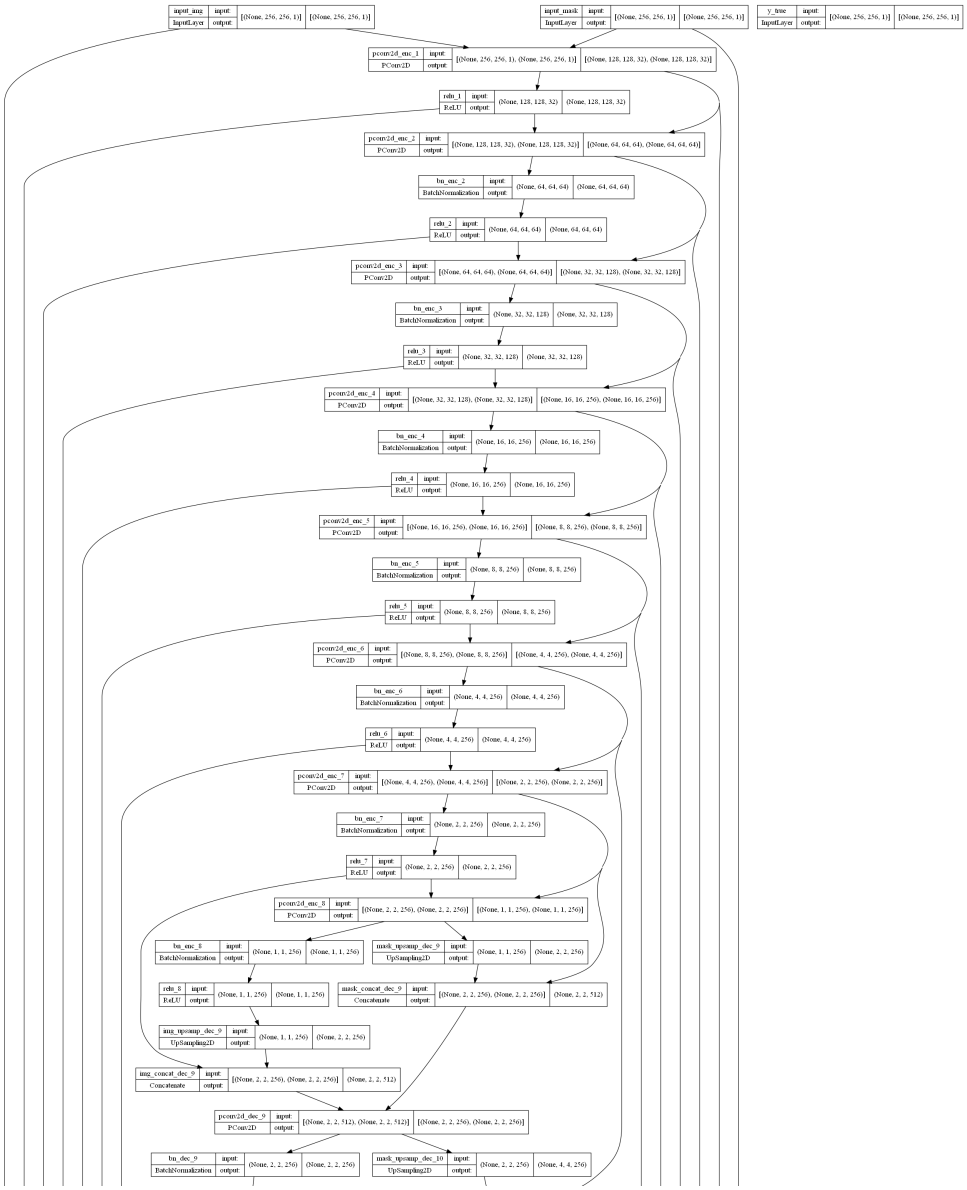


Figure 49: Downsampling part of the partial convolutional unet.

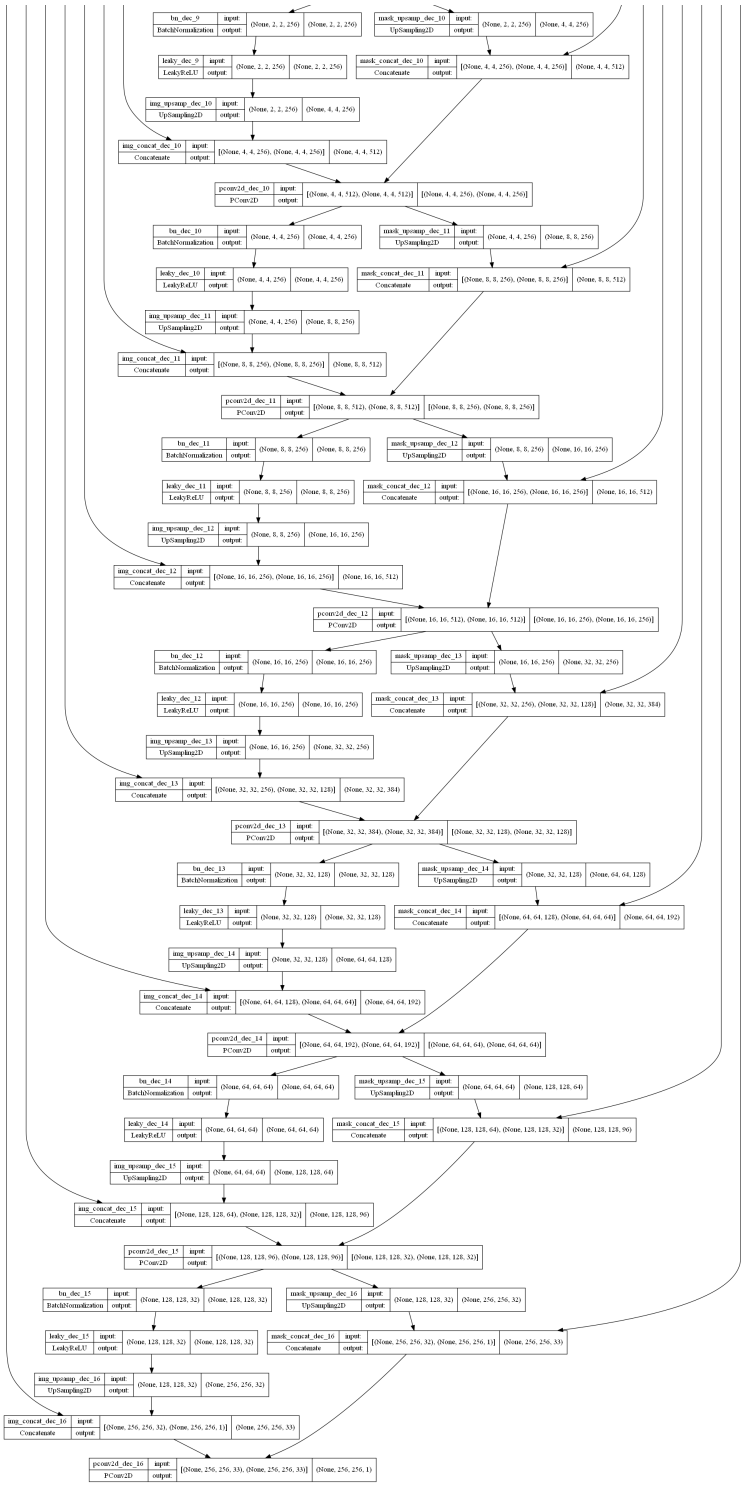


Figure 50: Upsampling part of the partial convolutional unet.

Gated convolution network model graph

Figure 51: The coarse stage of the gated convolutional network. Graph generated using Touchviz.

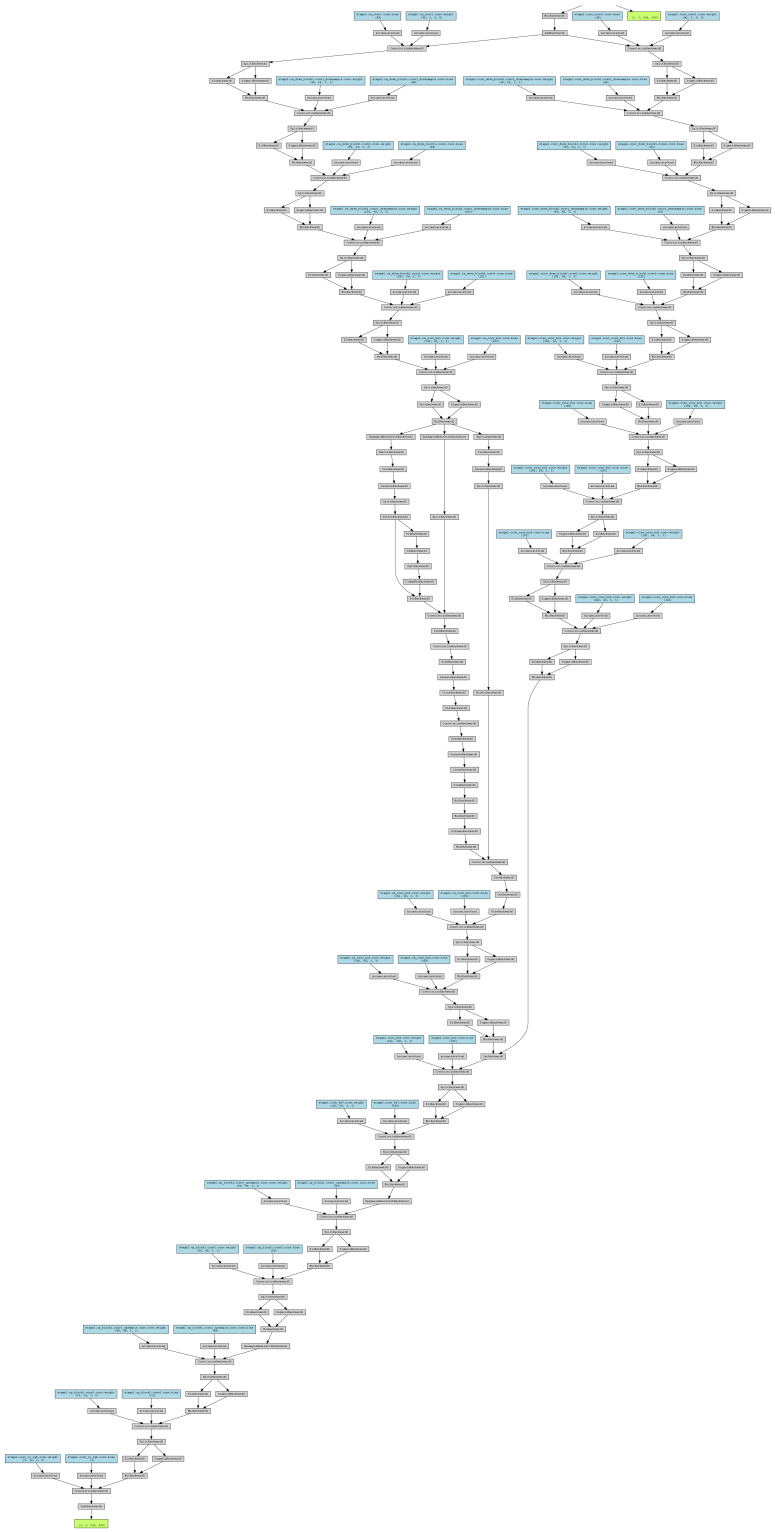


Figure 52: The refinement stage of the gated convolutional network. Graph generated using Touchviz.

Glathida observation locations on glacier masks

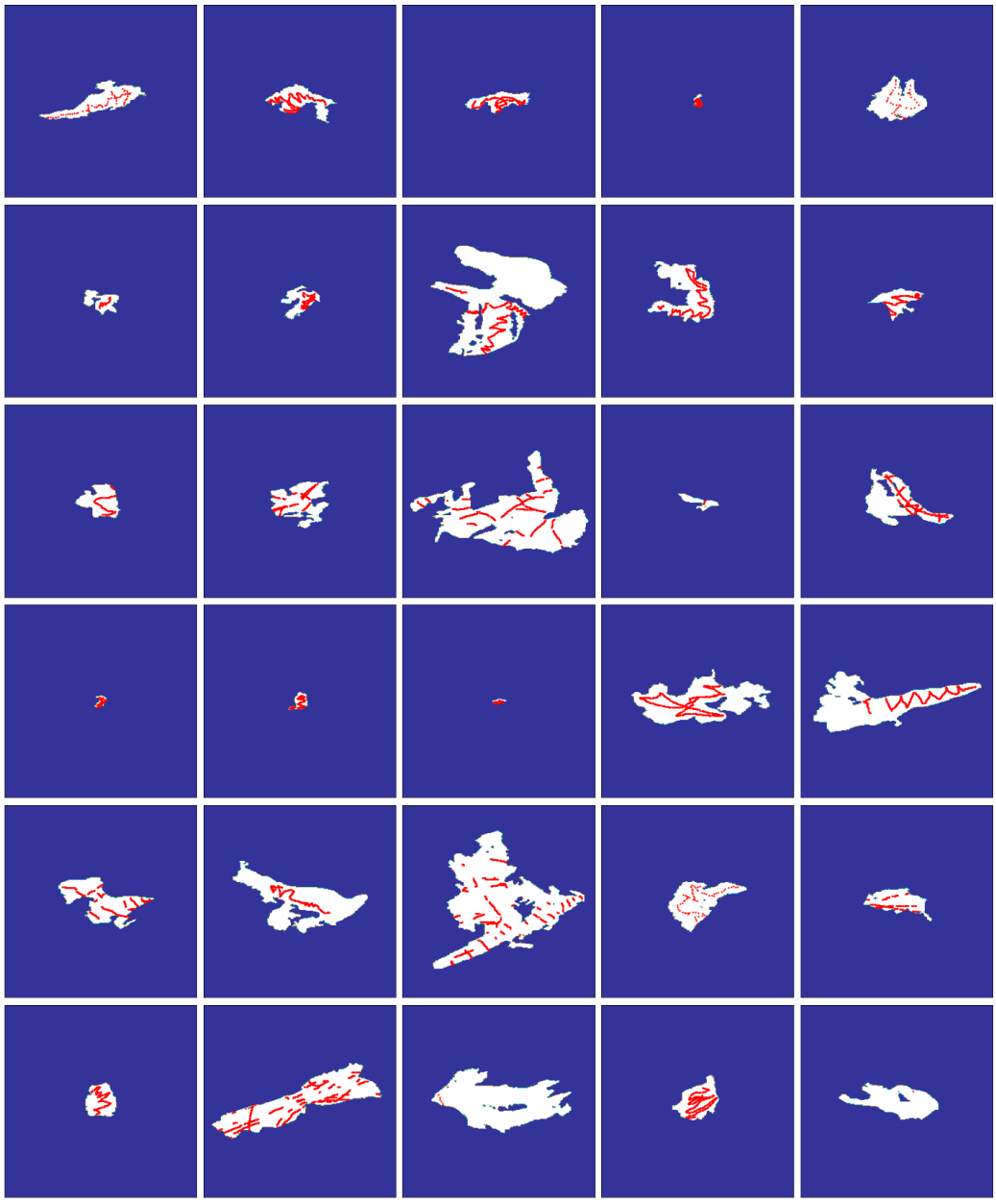


Figure 53: Glathida radar observation locations shown on 30 glacier masks.