Universiteit Leiden
The Netherlands

Computer Science

Automated Machine Learning
for Sea Ice Concentration Charting

Sven van Collenburg

Supervisors:
dr. J.N. van Rijn (Leiden University) &
prof.dr. H.H. Hoos (RWTH Aachen University, Leiden University)
External Supervisor:
A. Stokholm (Technical University of Denmark)

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

22/01/2024

Abstract

To further automate the process of sea ice charting, convolutional neural networks can be used. These networks take satellite images as input and convert them into charts showing the sea cce concentration. While machine learning models have the potential to aid the domain expert, training these networks accurately is a complex task and tuning the hyperparameters of the network is still done by hand. This is a labour-intensive job that can be replaced by an automated machine learning process. In this work, we use a Bayesian optimisation process to find the best set of hyperparameters for a U-Net convolutional neural network. A parallel Bayesian optimisation scheduler is created consisting of a chief optimisation process and workers that train an individual network. Additionally, we change the evaluation method to ignore ambiguities within the peprocessed satellite data. We analyse the results of the best configurations obtained by our Bayesian optimisation procedure and find that running an experiment consisting of multiple smaller optimisation processes has the highest chance of finding a good set of hyperparameters. This is because a single run of Bayesian optimization on this process can get stuck in local optima. Overall, this experiment achieves the same results as previous publications that utilize manual hyperparameter tuning, while having a simpler network architecture. The best-found set of hyperparameters achieves a R^2 score of 86.01% with an accuracy of 72.30%. To the best of our knowledge, this is the first work to apply automated machine learning to the problem of sea ice concentration charting.

Contents

1 Introduction

In the following subsections, we will first explain the importance of sea ice concentration charting and then introduce the process of automating machine learning.

1.1 Importance of sea ice concentration charting

The amount of sea ice in the Arctic oceans is rapidly decreasing due to global warming [PMT+20], allowing both political and economic activities to increase. New trading routes can be opened, connecting the north of America to Europe and Asia across the Arctic, leading to a large increase in shipping activities due to lower shipping times and costs compared to existing routes [BFRR17]. Simultaneously, geopolitical activity is increasing as Arctic nations want to gain control over the Arctic waters and the resources they contain [Fun09]. This increasing interest in the Arctic waters requires up-to-date and high-resolution Sea Ice Concentration (SIC) charts to guarantee safe and efficient travel of these waters.
Professional sea ice analysts have been manually drawing SIC charts for several decades. To do so these professional analysts use Synthetic Aperture Radar (SAR) images due to their high resolution and the ability to obtain these images independent of clouds and sun illumination [SKBH+20]. Even for trained analysts manually inspecting these SAR images is a time-consuming and labor-intensive job. In addition, the analysts have only a small time window to analyse the SAR images due to the dynamic nature of sea ice. Automating the task of creating SIC charts may increase the detail of the images along with increasing the speed at which they are delivered, making it highly desirable.

1.2 Automation of machine learning

Professionals who create SIC charts can be aided by data-driven systems, such as Convolutional Neural Networks (CNN). However, the process of tuning the hyperparameters of the network can, just like manually interpreting SAR images, be a very time-consuming and labor-intensive job. Automated Machine Learning (AutoML) will automatically search for an optimal set of hyperparameters and hence save work and time.
This paper will improve upon the convolutional neural networks used by creating an automated machine learning network to tune the hyperparameters. Therefore the focus will be on the question:

To which degree can automated machine learning be used to generate better Sea Ice Concentration images?

To achieve this, we utilize a parallel Bayesian optimization algorithm. This algorithm generates sets of hyperparameters that are used to train neural networks. These networks are trained with the satellite image data and afterward evaluated with a more appropriate evaluation method. This evaluation method has been created to deal with an ambiguity within the dataset. In every evaluation, the data is scanned for areas with ambiguity and these areas are removed from the data before final calculation of any performance measurements. More information about what this ambiguity is and how it is handled will be discussed in Section 4.5.

This optimisation algorithm tries to find the set of hyperparameters that will result in the highest scoring network. This network is then used to make predictions on an evaluation set that is not used for training. This is compared against baselines, such as a mean prediction and the results earlier reported by Stokholm et al. [SWK+22]. Our best network achieves a R^2 score of 86.01% with an accuracy of 72.30%. This is better than the mean prediction baseline with a R^2 score of -194% with an accuracy of 0.03% and is comparable to the earlier reported network results by Stokholm et al. [SWK+22] with a R^2 score of 86.34% (accuracy not reported).

2 Related work

In 2016 the first paper was published that uses convolutional neural networks together with SAR data to predict the sea ice concentration [WSXC16]. It is shown that convolutional neural networks can produce SIC maps with a higher amount of detail than when these maps are produced operationally. The results suggest that a CNN is robust enough to deal with the noise within the SAR data, deal with the effect of incidence angles, and deal with the effect that wind can have on water. The experiment is executed on a limited amount of data and hence the authors call for a more extensive study with a larger dataset containing a large diversity of locations.
The initial model was further developed by using deeper and larger CNNs to better deal with noisy data. Also, different regions are looked at [WSC17] [WSCX17]. Newer advancements have been made by the Automatic Sea Ice Products (ASIP) project which combines SAR data with Passive Microwave Radiometer (PMR) data [MHPN+20] [MHPK+21].

More recently the first paper to use a U-Net convolutional neural network architecture [Ron17] to predict SIC values using SAR data and ice chart labels was published [DGCL21]. Their model can handle a much larger field of view and produce higher resolution SIC maps. Their best performance was obtained when correcting the data for image burst miscalibration, the incidence angle, and the wind speed. This was followed closely by a paper using SAR data and ice values from PMR data while applying curriculum learning [RSC21]. First, open water regions are learned together with regions that contain large areas of sea ice. Only after this the model starts to learn more complicated regions. The results are obtained through a tenfold cross-validation experiment. This work shows that U-Net architectures are significantly faster and better than more traditional convolutional neural networks.

One of the most recent publications [SWK+22] shows that the SIC prediction improves when increasing the receptive field of the U-Net. Additionally, the influence of different noise correction schemes is investigated. The authors conclude that the NERSC noise correction scheme is better than the ESA noise correction scheme. We use this work as the basis for our research and will compare our obtained results with the results that are reported. The most relevant results will be shown in more detail in Section 6.1.2. Additional work suggests four different loss functions [KS23], namely: the mean squared error, regression based binary cross-entropy, categorical cross-entropy, and the squared Earth mover's distance. They report unique advantages and disadvantages to each of the aforementioned loss functions. Classification models tend to be better at predicting 0% sea ice and 100% sea ice, while regression models tend to be better at predicting the intermediate values.

3 Data

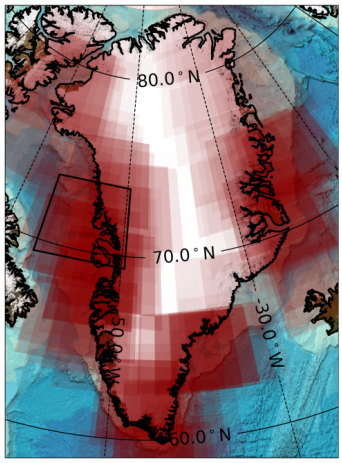


Figure 1: Geographical locations of training SAR scenes are shown as a red semi-transparent silhouette. The black frame highlights the ice chart location of the scene that will be used in this paper to show predictions. Figure and description are taken from [SWK+22].

The data used for the experiments is from the European Space Agency's (ESA) AI Ready Earth Observation (AIREO) sea ice dataset, AI4Arctic/ASIP v2 (ASID-v2) [SKBH+20]. The size of the dataset is 310GB. This dataset contains 452 scenes. Each scene is a 2D map containing two values for every pixel, namely Sentinel-1 dual polarised HH (transmitted and received in Horizontal polarisation) and HV (transmitted in Horizontal and received in Vertical polarisation) SAR images. Every scene has a corresponding SIC chart created by sea ice experts. This will serve as the ground truth data to be predicted. In Figure 1 the geographical locations of these scenes can be seen. The black square highlights one particular scene that will be used to show the prediction of the models that we create.

For each SAR image, two noise corrections are available; the ordinary ESA Instrument Processing Facility (IPF) SAR data and an alternative noise correction developed by the Nansen Environmental and Remote Sensing Center (NERSC) [PWK+19].
We preprocess the ASID-v2 scenes further according to the following procedure, following Stokholm et al. [SWK+22]. During this preprocessing 14 different SIC classes are converted into 11 SIC classes (class 0 - class 10) ranging from 0% to 100% in increments of 10%, where 0% is open-water and 100% is fully-covered sea ice. In addition, the SAR and SIC images are downsampled from 40 m to 80 m pixel spacing by applying a 2 x 2 averaging kernel to the SAR data and a maximizing kernel to the SIC data. The advantages of this are there is a positive impact of increasing the geospatial receptive field and speckle noise is reduced [HMZ21]. The disadvantages are having fewer pixels to train on and a poorer resolution.
After this, the SAR is normalized to the range [-1, 1]. For the SAR images, NaN values are replaced with 0, and for the SIC images, a new class 11 is created to represent non-data pixels, in most cases being land values. After preprocessing the dataset size is reduced to 196GB.

Additionally, all files are converted to NumPy array files. This increases the speed at which the neural network models can open these files, making the overall training of the network faster by reducing the reading time from the data directory. This has the additional benefit of further reducing the data directory size from 196GB to 104GB.

4 Methods

The problem of predicting the SIC can be seen as a regression task, where the absolute sea ice percentage is predicted, or a discrete classification task, where the SIC is divided into classes ranging from water to sea ice. In this study, we formulate the prediction of the SIC as a classification problem with a (weighted) categorical cross-entropy loss function.
The implementation of the following methods is available as code on the following GitHub page [vC24].

4.1 Performance measurements

The performance measurements that will be looked at are accuracy and the R^2 score. For each experiment one of these metrics will be used for optimizing the network. The unused metric will also be reported to better compare the models. Additionally, the Root Mean Squared Error(RMSE) values will be reported to provide a more detailed analysis of the performance. Landmasses are excluded from evaluation.

The accuracy is defined as:

$$Accuracy = \frac{\sum_{i=1}^{N_{pixel}} g(y_i^{true}, y_i^{pred})}{N_{pixel}}, \text{ where } g(y^{true}, y^{pred}) = \begin{cases} 1 & \text{if } y^{true} = y^{pred} \\ 0 & \text{if } y^{true} \neq y^{pred} \end{cases} \tag{1}$$

Here y_i^{true} is the true value of the i^{th} pixel, y_i^{pred} is the predicted value of the i^{th} pixel. N_{pixel} is the total number of pixels, flattened from 2D to 1D.

The statistical R^2 coefficient is defined as:

$$R^2 = 1 - \frac{\sum_{i=1}^{N_{pixel}} (y_i^{true} - y_i^{pred})^2}{\sum_{i=1}^{N_{pixel}} (y_i^{true} - \hat{y}^{true})^2} \tag{2}$$

y_i^{true} is the true value of the i^{th} pixel, y_i^{pred} is the predicted value of the i^{th} pixel, and $\hat{y}^{true}$ is the true mean.
There are two advantages of this metric compared to accuracy. Firstly, the R^2 score does not only differentiate between a correct prediction and an incorrect prediction but also takes into account how incorrect a certain prediction is. For example: predicting 20% or 40% when the true value is 30% is much better than predicting 100%. Secondly, the data imbalance is not well reflected by accuracy. The hardest classes to predict, the intermediate classes between 0% and 100% sea ice, only make up a small percentage of the total area.

The RMSE is very closely related to the R^2 score. Both measurements are based on the squared error. The reason RMSE will also be reported is that it is a more intuitive measurement compared to the R^2 score. Together with the accuracy it can give a good impression of how often the network is correct, what the average error of a prediction is, and from this how large the error is when the model is incorrect.

The RMSE is defined as:

$$RMSE = \sqrt{\sum_{i=1}^{N_{pixel}} \frac{(y_i^{true} - y_i^{pred})^2}{N_{pixel}}} \tag{3}$$

y_i^{true} is the true value of the i^{th} pixel, y_i^{pred} is the predicted value of the i^{th} pixel, and N_{pixel} is the total number of pixels.

4.2 Baseline methods

We use two baseline methods. The first method is a mean prediction. This method is chosen because it has two advantages: Ease of implementation and speed of calculation. Other methods that are often used like a random forest network do not perform well as they can not handle semantic segmentation tasks well. This makes a lot of methods unfit as a baseline.

We implemented the mean prediction baseline as follows. Firstly the mean of the training files is calculated. Then this mean is used to predict the value of every pixel. Because these are all matrix operations it does not take long to calculate the performance measurements.
Some interesting facts about the prediction can already be made. Because the dataset has an imbalance that looks like a valley parabola, the mean will be a class that is underrepresented. This will make the accuracy very low. In the best case, the training file mean is equal to the true mean of the validation files. This would result in $R^2 = 0$. When the mean of the training files is unequal to the true mean, it will always be a worse prediction than the true mean when comparing R^2 scores. Hence by definition $R^2 \leq 0$ for the mean prediction method.

The second baseline method that we will use is a U-Net CNN from a previous publication about SIC charting [SWK+22]. The U-Net CNN closely resembles the architecture we use to build our models. The hyperparameters are however tuned by hand. This makes it interesting to see how the automated machine learning process will compare to a closely resembling manually tuned network.

4.3 U-Net model

A U-Net model is as the name implies a U-shaped network architecture which consists of downsample and upsample blocks with in the middle a double convolutional block. This can be seen in Figure 2, where a schematic overview of a 4-level U-Net with a maximum level size of 32 is shown. This is the simplest network possible with the layer sizes: 16, 32, 32, and 32.
A level is defined as a downsample block with its corresponding upsample block connected via a skip connection. Each level is associated with a given number of filters. Max pooling is used, which

means that each level performs a 1:2 downsample. In other words, each input dimension is reduced by a factor of two. This makes it so that each downsample block groups certain pixels (effectively lowering the resolution) but adds filters, and each upsampling block reduces the number of filters but increases the resolution in pixels.

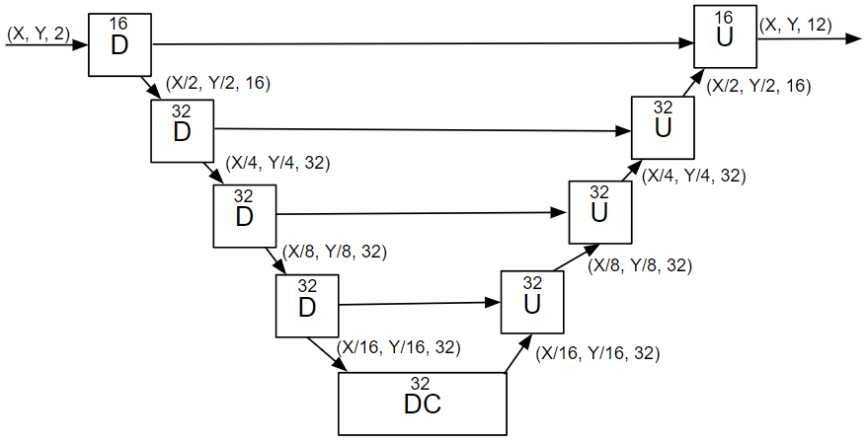


Figure 2: A schematic overview of a 4-level U-Net architecture with a maximum level size of 32. Each level consists of a downsample block with a corresponding upsample block connected via a skip connection. A D represents a downsampling block, a U represents an upsampling block, and a DC represents a double connected layer. The numbers at the top represent the filter sizes for each level.

The left side of the U consists of downsampling blocks, represented by D, wherein in each layer multiple pixels are represented by a single pixel in the layer below. The first downsample block transforms a two attribute layer, or the input size, into a downsampled layer with 16 filters, while simultaneously reducing each input dimension by a factor of two.
The bottom of the U consists of a double convolutional block, represented by DC, with the maximum layer size. Here no downsampling or upsampling happens.
The right size of the U consists of upsampling blocks, represented by U, where for each layer a single tensor represents multiple tensors in the layer above. The upsampling happens in reverse order as the downsampling. Hence each downsample block has a corresponding upsampling block at the other side of the U. These blocks are directly connected via a skip connection. An exception is the last upsample block where the filter of size 16 turns into 12 output values, instead of an upsample to the original input size. This is because the model output is a tensor with 12 attributes, each attribute representing a single class.

The downsampling and upsampling have to happen in a specific order. As mentioned before the dimension downsampling or upsampling is 1:2 for each downsample or upsample block respectively. The amount of filters in the first layer is always 16. After that, the amount of filters increases by a factor of two in each layer until the maximum layer size is reached.
This results in the following filters:

- 16, 32 x (n_levels - 1)
- 16, 32, 64 x (n_levels - 2)
- 16, 32, 64, 128 x (n_levels - 3)

4.3.1 Loss function

A weighted loss function is used as it may help the model learn underrepresented classes in the class distribution. Three weight distributions will be used, namely: no weights, excluding land masses, and a weight distribution based on the data imbalance, which can be set as a hyperparameter of the learning process. To calculate the loss a weighted sparse categorical cross-entropy loss function is used. For each pixel, the loss is defined as:

$$loss(\mathbf{x}, class) = -1 * w_{class} * log(x_{class}) \tag{4}$$

Where $\mathbf{x} = (x_j)_{j=0}^{N_{class}-1}$ is the vector of predicted class probabilities. *class* is the index of the true class. The total loss is simply defined as the sum of the loss values of all pixels:

$$loss = \sum_{i=1}^{N_{pixels}} loss(\mathbf{x}_i, class_i) \tag{5}$$

4.3.2 Sample probability and data loading

We have implemented two data loaders that allow for lazy loading (obtaining a patch at the moment it is needed) during the hyperparameter optimization loop. These data loaders have different strategies when it comes to presenting samples to the network. Note that for testing the final model (after the hyperparameter optimization loop has ended), we have one fixed data loader that works consistent with previous work [SWK+22].

The first method tries to load an entire scene. This is not possible due to memory limitations but is mimicked by dividing the scene into patches that have minimal overlap. This way every epoch the same patches are loaded from a single scene and hence there is more control over the patches the network receives. The order in which the scenes are loaded is random by implementing a shuffle method.
Every scene is loaded. In other words: the sample probability of a scene is one. When the scenes are divided into patches 425 batches of 16 patches can be extracted.

The second method first calculates the amount of patches that will be taken from a scene beforehand based on the amount of patches extracted from method one. When dividing the scenes into patches with minimal overlap we get a certain amount of patches. The same number of patches is extracted

from each scene during random loading as if one would extract patches like in method one. This way the amount of patches is based upon the size of a scene. The resulting distribution resembles the chance distribution from [SWK+22]. The difference is that during random loading the patches may come from anywhere in the scene, compared to the first method where the patches have a fixed location.

4.4 Bayesian optimisation

We have chosen to implement a Bayesian optimisation algorithm because of its proven effectiveness [SLA12]. An additional benefit is the sample efficiency of Bayesian optimisation, where relatively few evaluations are needed compared to other optimisation methods. Bayesian optimisation uses a probabilistic model to model an unknown objective function. This way the behaviour of a function along with its uncertainty can be estimated. An acquisition function guides the optimisation process by trading off exploration and exploitation. This is done in an iterative process by choosing a set of hyperparameters and evaluating the corresponding model. When new evaluation results become available, the probabilistic model is updated. The process stops when a stopping criterion is met or when the computational resources have been exhausted. For a more detailed explanation of Bayesian optimisation, please refer to [Gar23]
Most Bayesian optimisation algorithms work sequentially, as this is the most optimal for performance. To reduce wall clock time, we have chosen to trade some efficiency for a parallelised speedup. This will be explained in further detail in Section 4.4.2. The Bayesian optimisation implementation is taken directly from KerasTuner. The choice of library will be explained in the next section.

4.4.1 Selected implementation

A large part of our model architecture is based on one of the baseline methods we use, namely a U-Net CNN model implemented in PyTorch [SWK+22]. By using automated machine learning libraries for PyTorch a large part of the network code can be reused. This turned out to be impossible as all Auto-PyTorch libraries require a dataset to be loaded into memory at once, as lazy loading was not supported at the time of experimenting. Even with a reduced dataset size of 104GB, it is not possible to load all the data at once into memory, hence lazy loading is a necessity. For this reason, AutoML libraries for PyTorch like Auto-PyTorch cannot be used.

TensorFlow libraries can support lazy loading. In Figure 3 some of the most well known TensorFlow automated machine learning frameworks can be seen. These range from 'very configurable but hard to implement' to 'simple but not very configurable'. TensorFlow contains hardly any framework for the automation of machine learning, while AutoKeras handles almost the entire automated machine learning process.
To make the implementation of the AutoML network as easy as possible using AutoKeras would be preferred. However, AutoKeras does at the time of experimentation not support semantic segmentation [JCSH23]. This is exactly the task we seek to complete. Because of this, a more configurable approach to this AutoML process is needed. In Figure 3 can be seen that the next best library for keeping the automated machine learning process as simple as possible, excluding AutoKeras, is KerasTuner. Because of this KerasTuner will be the library of choice to implement the U-Net model and automated machine learning process.

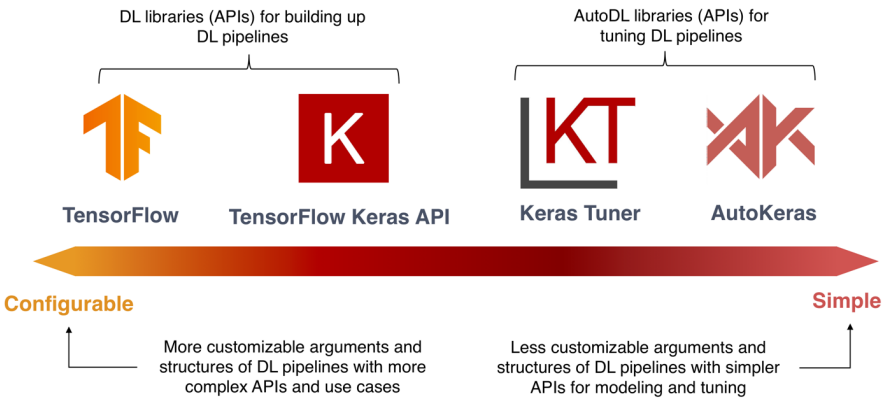


Figure 3: Graph showing how simple or configurable the libraries TensorFlow, TensorFlow Keras API, KerasTuner and AutoKeras are. This image is taken from [SJH22].

4.4.2 Parallel Bayesian optimisation scheduling

The cluster on which we will evaluate our method has a maximum job time of 7 days. In order to maximize the amount of resources we can use per optimisation run in this time, we distribute the individual runs of the Bayesian optimisation process on cluster-scheduler level.

The chief is a CPU job (that will be ran for the maximum amount of time) that controls the Bayesian optimisation process. During this time the chief waits for workers to report themselves. Once a worker reports to the chief, the chief sends a set of hyperparameters to the worker, and the worker's status is set to running. It will then wait for a response from that given worker, either a successful complete or unsuccessful complete. Because the chief does not have to wait for any specific worker to finish, the Bayesian optimisation process can be executed in parallel with a large amount of workers. There is however a trade-off: a new set of hyperparameters is selected based on the workers that have successfully finished. If plenty of workers are executed at the same time, then the set of hyperparameters given to these workers will be much less informed compared to sequentially executing workers. For example: If ten workers are executed at the same time, they all have zero previous knowledge, while running two groups of five workers in sequence results in the second group having five results to base their hyperparameters on. The chief saves information about its optimisation process in a directory for later analysis.

Each worker is a GPU job that trains a single convolutional neural network for a single set of hyperparameters. The moment a worker is started it searches for a chief and reports to it when found. Then the worker waits to receive a set of hyperparameters to begin creating the convolutional neural network. While training, metadata about the training process is written to a directory such that it can be analysed at a later time. The moment the training process has been completed the worker reports back to the chief. The worker reports how well the model performed and sets the worker status to completed. The chief can then use this information to create a new set of hyperparameters.

After all workers have finished their jobs, the chief process can gather all results, and determine which model performed best. This model model can be used to make predictions for the validation set and these can be evaluated.

4.4.3 Configuration space

In this section, all hyperparameters that are used will be explained. The set of hyperparameters that is being optimised differs for each experiment and will be discussed in the setup sections for that experiment. The hyperparameters that will be discussed in this report are:

- Patch and batch size
- Learning rate
- Noise reduction
- Dropout
- Number of levels
- Maximum level size
- Class weights
- Random loading

The patch size is the size of a patch extracted from a scene. The batch size is the amount of patches in a batch that is loaded into the convolutional neural network before updating the model parameters during the training phase.
The learning rate determines the pace at which the internal parameters of the convolutional neural network are updated.
The ESA and NERSC noise reduction methods will be looked at, as mentioned in Section 3.
The dropout is the probability that a node within the neural network is dropped. Dropping a node means that all forward and backward connections are temporarily removed, thus disconnecting it from the network. A temporary new architecture is created from a parent network. This helps against overfitting by preventing nodes from fixing the mistakes of other nodes.
The number of levels and maximum level size determine the architecture of the U-Net neural network. The architectures possible are discussed in Section 4.3

Three different class weight distributions will be tested as hyperparameters. The first distribution is one where all classes have an equal weight, namely a weight of one.

$$w_{0,1,\ldots,11} = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1] \tag{6}$$

In the second weight distribution, the classes 0-10 all have a weight of one, however, class 11 has a weight of zero and is thus excluded.

$$w_{0,1,\ldots,11} = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0] \tag{7}$$

The last weight distribution used is the weight distribution based on the class imbalance in the data. This weight distribution is directly taken from the following paper [SWK+22].

$$w_{0,1,\ldots,11} = [0.039, 1.413, 0.907, 0.925, 1.089, 1.401, 1.233, 1.154, 0.702, 0.369, 0.099, 0] \tag{8}$$

Two data loading methods have been implemented that will also be tested during the hyperparameter optimisation. The first method is non-random data loading based on loading entire scenes. The second method is more random by loading a patch randomly located in a scene. In this method, a scene is selected with a chance distribution based on the size of the scenes.

4.5 Evaluation procedure

While evaluating the U-Net neural networks we found that some areas were hard to predict. These regions could most often be found within fjords in scenes, although not exclusively. On closer inspection of the data, we found that there exists some inconsistency within the data that makes these areas hard to predict. The attribute data from the satellite would be homogeneous and contain all zeroes, while the target would be different classes. For example: two pixels next to each other both have the attribute values 0 and 0, while the class of the first pixel is 4 and the class of the second pixel is 7. In Figure 4 it can be seen that the fjords have not been predicted well. Mostly the fjords in the north have not been predicted correctly as a large range of colours can be found while the correct prediction should only contain 100% sea ice.

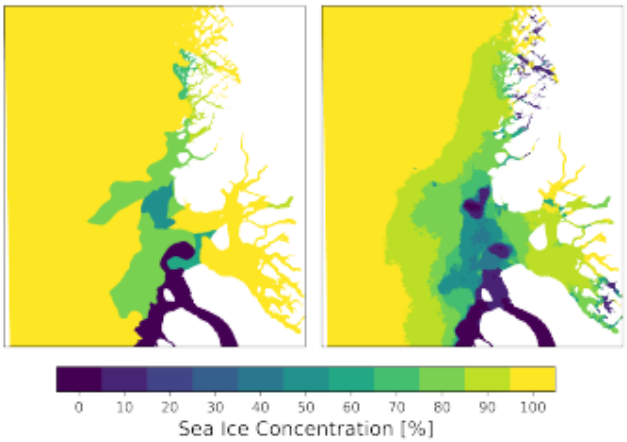


Figure 4: The northern fjords are hard to predict because of homogeneous data. The data within these fjords is all zeroes. Figure taken from [SWK+22].

In the new evaluation method areas with this homogeneous data are excluded before calculating the performance measurements. This happens both during the training of the network and during the final statistical calculations. The difference in area can be seen in Figure 5. Here we can see that a large part of the northern fjord contains the homogeneous data and has hence been removed.

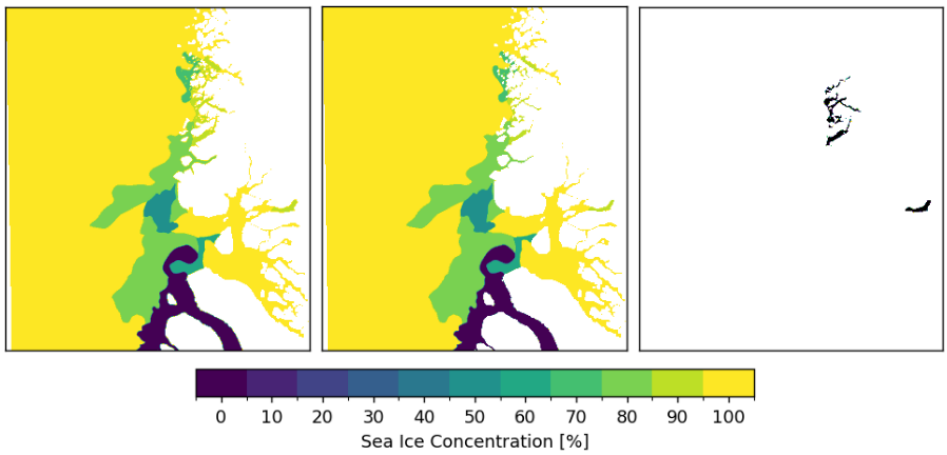
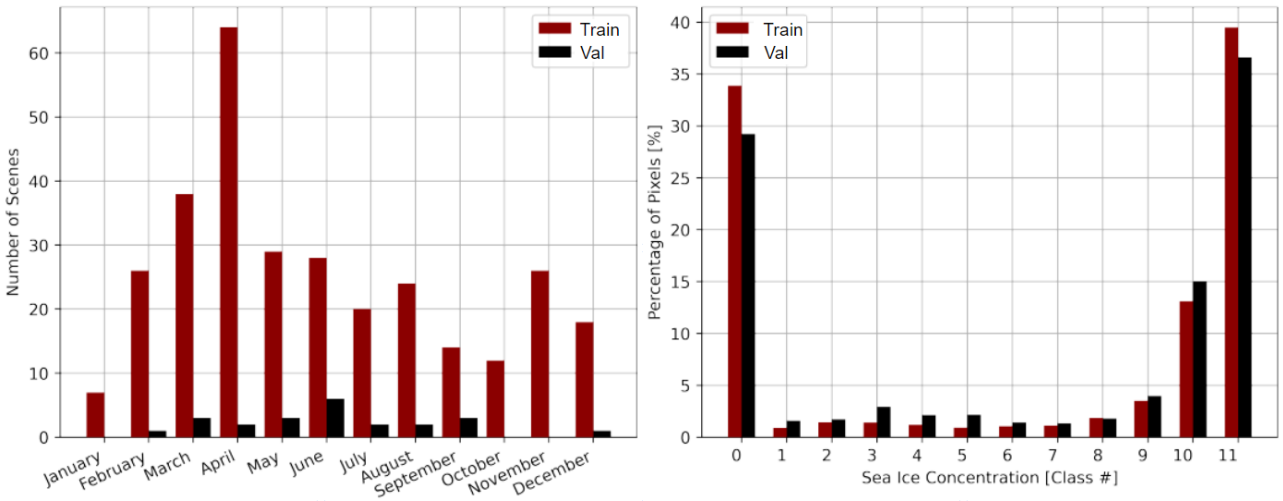


Figure 5: The different areas used in the evaluation. The old evaluation area is on the left and the new evaluation area with excluded regions is on the right.

5 Experimental setup

For all experiments the data as described in Section 3 is used.



(a) (b)

Figure 6: **a**: seasonal distribution of train and test scenes. **b**: percentage of pixels belonging to each class with respect to the total pixels in the train and validation split. Class 0-10 refers to 0%-100% sea ice, and class 11 is masked pixels. Figure and description taken from [SWK+22].

A train validation split is made based on a representative distribution of the data. 306 scenes are selected for training and 23 scenes are selected for validating. These validating scenes have been chosen in collaboration with the Danish Meteorological Institute (DMI) because of their difficulty. The seasonal distribution for the train and validating set can be seen in Figure 6a. The

SIC distribution of the train and validation set can be seen in Figure 6b. It is interesting to note that classes 1-9 occur far less than classes 0 (open water), 10 (100% sea ice), and 11 (land). Though between themselves classes 1-9 are roughly equally present. The validation scenes are fairly well distributed across the seasons, although there are relatively fewer scenes in the spring compared to the train set. For the SIC values the validation set follows the imbalance of the entire data set.

Five sets of experiments have been run. Each experiment improves upon the results of the previous experiments. First, the mean prediction is run to give the baseline for the other experiments. After that two experiments optimized on accuracy will be run and two experiments optimized on R^2 score.

6 Experiments

All experiments, except for the convolutional U-Net baseline, have been run on the ALICE High Performance Computing facility of Leiden University. The scheduler in this environment can be unpredictable as to how much run time gets reserved for experiments. Additionally, trials being run in this environment may fail due to several reasons, like timeouts or out of memory errors. Hence, the number of trials requested may differ from the number of trials executed or the number of trials completed successfully.

6.1 Baselines

6.1.1 Mean prediction

The mean of the training set is 4.22. This value is used as a predictor for all pixels. Table 1 shows the performance statistics.

TABLE 1: This table contains the R^2 score and RMSE of the mean prediction of 4.22 and the rounded prediction of 4. For the rounded prediction, the accuracy is also reported.

R^2	RMSE	rounded R^2	rounded RMSE	Accuracy
-1.94	4.24	-1.80	4.21	0.03%

These statistics are supported by the data distribution mentioned in Figure 6b. This data distribution has the following percentages for each class:

[0.29295652, 0.01417391, 0.01713043, 0.02734783, 0.02552174, 0.02304348,
0.0266087, 0.01182609, 0.02182609, 0.03830435, 0.13808696, 0.36317391]

Here we can see that a prediction of 4.22 (or 4) for all classes does not perform well, although not for all performance measurements. The accuracy is low because class 4 is underrepresented. The RMSE is quite high as the most present classes are 0 and 10, so, logically, the RMSE will be between four and five. The R^2 score is also less than 0, implying that the prediction is worse than predicting the (true) mean. This can be explained by the fact that the mean of the training files does not equal the mean of the validation files. The mean prediction of the training files will always be worse than the mean prediction of the validation files when they are not equal.

Not only should the R^2 score be looked at, but accuracy and the RMSE as well. This is illustrated in Figure 7. On the left is the target. The middle prediction has an R^2 score of -9.21% while the prediction on the right has an R^2 score of -0.0068%. The prediction on the right is worse, even though the R^2 score is higher, because the accuracy is zero. Class 4 is predicted even though this class is not present in the scene.

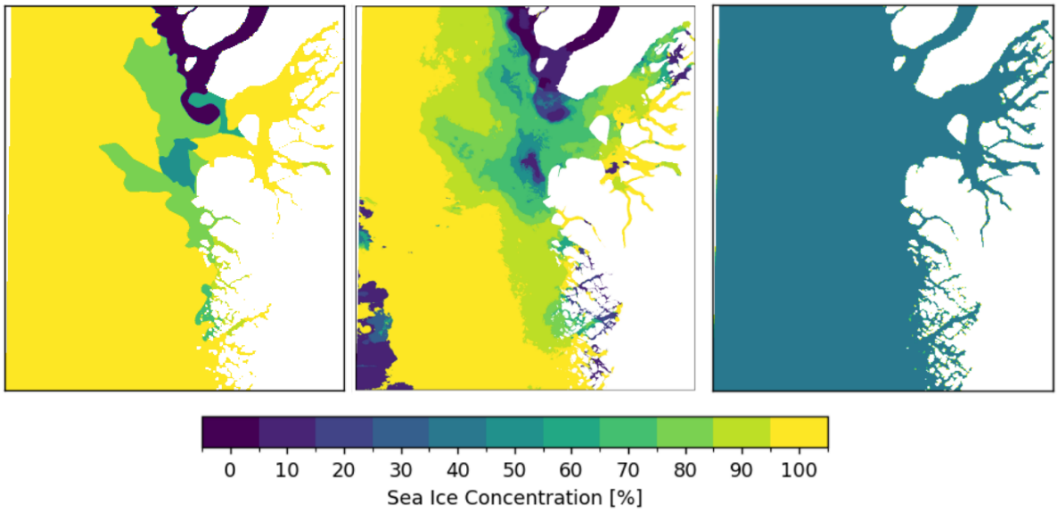


Figure 7: A higher R^2 score is not always better. The target is on the left. The middle image has an R^2 score of -9.21%. The right image has an R^2 score of -0.0068%. The middle image is taken from [SWK+22].

6.1.2 Convolutional U-Net

To give an idea of how a more complex model performs, the results of the work by Stokholm et al. [SWK+22] will be mentioned below in Table 2. This method was not run locally, instead the statistics mentioned in their work have been listed. Additionally, the evaluation method still includes ambiguous areas. The best performing model, highlighted in bold, has an R^2 score of 86.34%. This model has a high patch size, low batch size, and a large number of levels and uses the NERSC noise correction.

TABLE 2: Summary of trained models and performance. The column names refer to the following: reference number, training patch size, training batch size, *receptive field (RF), number of levels in U-Net architecture, number of convolutional filters in each level, SAR noise correction, and finally the test R^2 score. Table and description taken from Stokholm et al. [SWK+22].

#	patch	batch	RF*	levels	layers	filters	correction	R^2
1	512	64	92	3	15	16, 2×32	ESA	69.61%
2	512	64	92	3	15	16, 2×32	NERSC	76.72%
3	512	64	188	4	19	16, 3×32	ESA	77.90%
4	512	64	188	4	19	16, 3×32	NERSC	79.90%
5	512	64	380	5	23	16, 4×32	ESA	82.65%
6	512	64	380	5	23	16, 4×32	NERSC	84.11%
7	512	64	764	6	27	16, 5×32	ESA	84.78%
8	512	64	764	6	27	16, 5×32	NERSC	85.61%
9	768	32	1,532	7	31	16, 6×32	NERSC	85.33%
10	**768**	**32**	**3,068**	**8**	**35**	**16, 7×32**	**NERSC**	**86.34%**

6.2 Accuracy optimisation small-scale

For the first experiment, we run a small experiment that uses accuracy as a validation score to optimize the network.

6.2.1 Setup

The model search space for this experiment is defined as follows:

- patch_size: [512, 768]
- batch_size: [16, 32]
- num_levels : [4, 5, 6, 7, 8]
- learning_rate: [0.0001, 0.0002, 0.0005]
- dropout: [0]
- max_level_size: [32]
- load_randomly: [False]
- noise_reduction: [NERSC]
- weights: [None]

In the first experiment batch and patch sizes 512, 768, and 16, 32, respectively, are tested as hyperparameters. Further increasing the patch size as suggested in the conclusion and future work section of [SWK+22] could not be done due to memory limitations. The number of levels is also taken directly from this paper. The learning rate is the standard implementation from KerasTuner.

This experiment uses a smaller set of hyperparameters to quickly get back and check the first experiment results.
A total of 50 trials will be requested for this experiment. This number is based on getting results back in roughly 3 days, making it not take too long before seeing results to improve further experiments on.

6.2.2 Results

A total of 50 trials were executed, 39 of which completed successfully. Of these 39 trials, the top 5 trials and their hyperparameters and result metrics are shown in Table 3.

TABLE 3: Summary of trained models and their performance for the small-scale accuracy experiment. The column names refer to: patch size, batch size, learning rate, number of levels, the test accuracy score, the test RMSE value, and the test R^2 score.

#	patch	batch	learning	levels	Accuracy	RMSE	R^2
1	**768**	**16**	**0.0001**	**6**	**79.01%**	**1.71**	**84.39%**
2	768	16	0.0001	8	78.91%	1.91	80.62%
3	768	16	0.0002	4	78.66%	2.04	77.89%
4	768	16	0.0005	6	78.58%	1.95	79.64%
5	512	16	0.0005	7	78.39%	1.77	83.31%

The best network, namely network number one, is used to generate prediction images of the scenes. A scene that is representative of the test set is shown in Figure 8. It is a good prediction, although the network has a hard time within the fjords and a misprediction is made on the bottom left side of the scene.

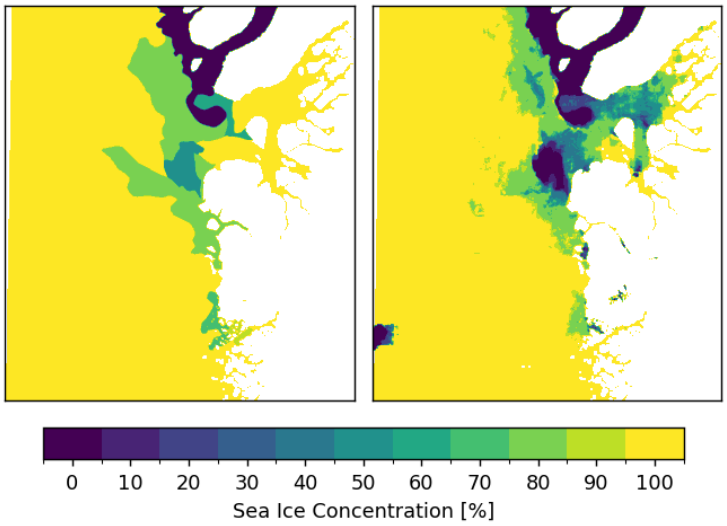


Figure 8: The prediction for the small-scale accuracy experiment. The target is on the left and the prediction is on the right. The R^2 score for this scene is 54.84%.

6.2.3 Discussion

In this results table, we can see that the network seems to prefer a high patch size and a low batch size. This is in line with the results found in [SWK+22] mentioned above. To reduce the number of hyperparameter combinations that need to be checked the patch and batch size will be removed from further experiments.

Furthermore, it should be noted that when optimizing for accuracy, the R^2 score tends to fluctuate. The highest accuracy scoring network does not need to have the highest R^2 score, although, in this particular experiment, it does. The second highest R^2 score is obtained by network number five. The best performing network has an accuracy of 79.01% and a R^2 score of 84.39%. This is already quite a high R^2 score and comes close to the results obtained by others mentioned in Section 6.1.2. After the first experiment, a batch size of 768 and a patch size of 16 are used as these seem to give the best results. This reduces the number of hyperparameter combinations by 4, and such creating room for focusing on finding optimal values for different hyperparameters.

6.3 Accuracy optimisation large-scale

For the second experiment, we run a larger experiment that uses accuracy as a validation score to optimize the network.

6.3.1 Setup

The model search space for this experiment is defined as follows:

- num_levels : [4, 5, 6, 7, 8]
- learning_rate: [0.000001, 0.00001, 0.0001, 0.001, 0.01, 0.1]
- dropout: [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6]
- max_level_size: [32, 64, 128]
- load_randomly: [True, False]
- noise_reduction: [ESA, NERSC]
- weights: [None, Ignore_11, Custom]

The learning rate is chosen in a range of 0.1-0.000001[1]. Six values are chosen based on a logarithmic scale resulting in the values: [0.000001, 0.00001, 0.0001, 0.001, 0.01, 0.1]. A dropout was added to deal with overfitting. Values are chosen in a range of 0-0.6. The recommended range is 0-0.5[2]. The range is slightly larger to give the network some more options to work with for preventing overfitting. Seven values are chosen resulting in the following values: [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6]. Both noise correction methods are now given as a hyperparameter option. Additionally, the weights are added as mentioned in Section 4.4.3, and the maximum level size is added as an option as mentioned in Section 4.3.
The amount of trials run will be the maximum amount allowed in 7 days by the scheduler. This number is not known beforehand. Ideally, 100 trials will be run, double the number of trials of the small-scale accuracy experiment, as the time given to the experiment is also roughly double that of the small-scale accuracy experiment.

[1]Based on: https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/

[2]Based on: https://machinelearningmastery.com/dropout-regularization-deep-learning-models-keras/

6.3.2 Results

A total of 91 trials were executed, 80 of which completed successfully. Of these 80 trials, the top 5 trials and their hyperparameters and result metrics are shown in Table 4.

TABLE 4: Summary of trained models and their performance for the large-scale accuracy experiment. The column names refer to: the number of levels (lvl), learning rate, dropout percentage (drop), the size of a level, is the data randomly loaded, the noise correction method, weights used, test accuracy score, test RMSE value, and test R^2 score.

#	lvl	learning	drop	size	random	noise	weights	Accuracy	RMSE	R^2
1	**7**	**0.0001**	**0.1**	**64**	**False**	**NERSC**	**None**	**79.01%**	**1.97**	**79.36%**
2	8	0.0001	0.2	64	False	NERSC	None	78.80%	1.95	79.73%
3	7	0.0001	0.2	64	False	NERSC	None	78.67%	1.84	82.04%
4	8	0.0001	0.2	64	False	NERSC	None	78.52%	1.90	80.82%
5	8	0.0001	0.2	64	False	NERSC	None	78.48%	1.95	79.68%

The network with the highest accuracy is no longer the network with the highest R^2 score. However, because this experiment focuses on accuracy optimisation, network number one is used to generate prediction images of the scenes. The scene prediction can be seen in Figure 9. The largest fjord area is predicted better than the small-scale model and the bottom left is now correctly predicted as 100% sea ice. However, one large fjord is predicted as ice instead of water, and large pieces of intermediate values in the middle of the scene are predicted as 100% sea ice.

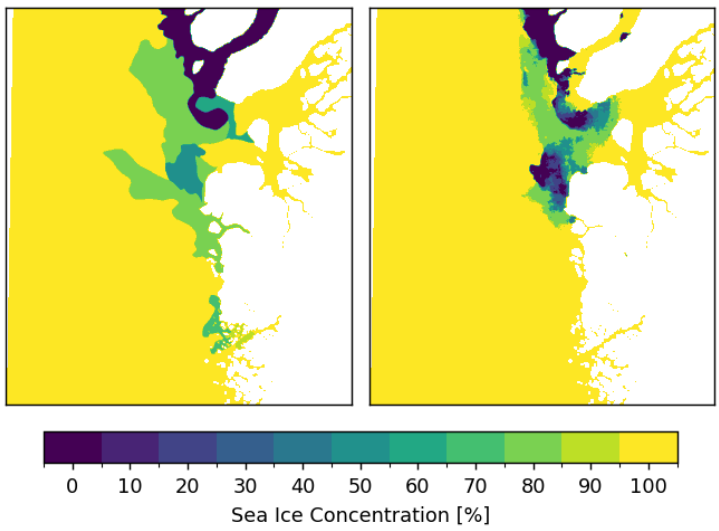


Figure 9: The prediction for the large-scale accuracy experiment. The target is on the left and the prediction is on the right. The R^2 score for this scene is 44.20%.

6.3.3 Discussion

The best model found by the optimisation process has an accuracy of 79.01% and an R^2 score of 79.36%. Interestingly, the best network of the small-scale accuracy experiment has found a network

with the same accuracy. What does differ is the R^2 score. The small-scale experiment network has an R^2 score of 84.39%. This is significantly higher and can be explained by the fact that the network is not told to pay attention to the R^2 score, so it does not matter if the score is high or low.

Even though the networks have the same accuracy, the network that was obtained from the large-scale experiment is better at predicting values in the fjords and does not make a misprediction at the bottom left corner, while the network from the small-scale experiment is better at predicting intermediate values.

6.4 R^2 optimisation large-scale

For the third experiment, we again run a large experiment. However, this time we use the R^2 score as the validation score to optimize the network.

6.4.1 Setup

The model search space for this experiment is defined as follows:

- num_levels : [4, 5, 6, 7, 8]
- learning_rate: [0.000001, 0.00001, 0.0001, 0.001, 0.01, 0.1]
- dropout: [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6]
- max_level_size: [32, 64, 128]
- load_randomly: [True, False]
- noise_reduction: [ESA, NERSC]
- weights: [None, Ignore_11, Custom]

These are the same values as the large-scale accuracy experiment, such that the results found can be optimally compared. The amount of trials run will be the maximum amount allowed in 7 days by the scheduler. This number is not known beforehand. Ideally, at least 80 trials will be completed successfully, the same target as the large-scale accuracy experiment.

6.4.2 Results

A total of 108 trials were executed, 91 of which were completed successfully. Of these 91 trials, the top 5 trials and their hyperparameters and result metrics are shown in Table 5.

The network with the highest R^2 score also has the highest accuracy. Both scores are however significantly lower than the previous two experiments. Network number one is used to generate prediction images of the scenes. Even though the global accuracy and R^2 scores are lower than the previous experiments, the scene R^2 score is higher. So the network performs worse generally but is particularly good in predicting this scene, with an R^2 score of 74.50% compared to 54.84% and 44.20% for the small-scale and large-scale accuracy experiments respectively. The scene prediction can be seen in Figure 10. The fjords are predicted without noticeable error, yet the intermediate values in the middle of the scene remain challenging.

TABLE 5: Summary of trained models and their performance for the large-scale R^2 experiment. The column names refer to: the number of levels (lvl), learning rate, dropout percentage (drop), is the data randomly loaded, the noise correction method, weights used, test accuracy score, test RMSE value, and test R^2 score.

#	lvl	learning	drop	level_size	random	noise	weights	Accuracy	RMSE	R^2
1	**8**	**0.0001**	**0.2**	**128**	**True**	**SAR**	**None**	**72.87%**	**1.91**	**80.65%**
2	8	0.0001	0.2	128	True	SAR	None	72.65%	2.03	78.06%
3	8	0.0001	0.2	128	True	SAR	None	71.76%	2.11	77.49%
4	8	0.0001	0.2	128	True	SAR	None	71.04%	2.12	77.12%
5	8	0.0001	0.2	128	True	SAR	None	72.28%	2.09	77.01%

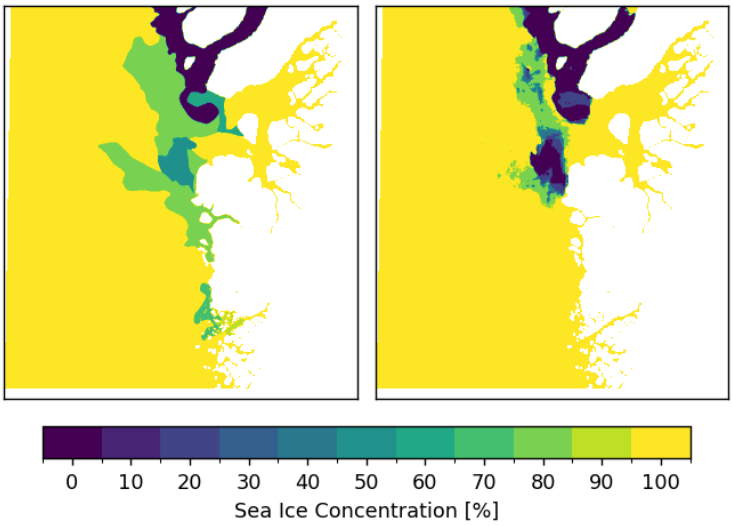


Figure 10: The prediction for the large-scale R^2 experiment. The target is on the left and the prediction is on the right. The R^2 score for this scene is 74.50%.

6.4.3 Discussion

We consider the network that was ranked first in the optimisation process. It is surprising to see that the R^2 score obtained in this experiment is lower than in the previous two experiments (cf. Section 6.2.2 and Section 6.3.2), while these have accuracy as optimisation target. The layer size is 128 instead of 64 and data is randomly loaded. This can be a difference preferred by R^2 optimisation instead of accuracy optimisation. However, it is strange that the SAR noise reduction method is used, as the NERSC noise reduction method has been found to be better for all possible network configurations. We hypothosise that the Bayesian optimisation procedure got stuck in a local optimum and was unable to find better optima. For this reason, we setup a different experiment that repeats the optimisation multiple times.

6.5 R^2 optimisation multiple small-scale

For the fourth experiment, we run multiple smaller experiments that use the R^2 score as the validation score to optimize the network.

6.5.1 Setup

The model search space for this experiment is defined as follows:

- num_levels : [4, 5, 6, 7, 8]
- learning_rate: {(0.00001, 0.01), log-uniform}
- dropout: [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6]
- max_level_size: [32, 64, 128]
- load_randomly: [True, False]
- noise_reduction: [ESA, NERSC]
- weights: [None, Ignore_11, Custom]

The hyperparameters are mostly the same as the large-scale R^2 experiment. One improvement has been made by allowing for a log distribution of learning rate values instead of six hard coded values on a logarithmic scale. Three sub-experiments will be started, with the amount of trials run being the maximum amount allowed in 7 days by the scheduler. Ideally, a minimum of 50 trials will be run for each of the sub-experiments, the same target as the small-scale accuracy experiment.

6.5.2 Results

Sub-experiment one ran 55 trials, 53 of which were completed succesfully, Sub-experiment two ran 39 trials, all of which were completed succesfully, Sub-experiment three ran 29 trials, all of which were completed succesfully. The hyperparameters and performance metrics of the top 3 networks of each sub-experiment are reported in the Table 6.
The best networks from all three sub-experiments have a high R^2 score compared to all other main experiments. The best networks from sub-experiments one and three outperform all other

TABLE 6: Summary of three trained models and their performance for the small-scale R^2 experiment. The column names refer to: the number of levels (lvl), learning rate, dropout percentage (drop), is the data randomly loaded (rand), the noise correction method, weights used, test accuracy score, test RMSE value, and test R^2 score.

#	lvl	learning	drop	size	rand	noise	weights	Accuracy	RMSE	R^2
1.1	**6**	**1.39e-4**	**0.0**	**64**	**False**	**NERSC**	**None**	**72.30%**	**1.62**	**86.01%**
1.2	6	1.71e-4	0.0	64	False	NERSC	None	71.51%	1.63	85.86%
1.3	5	1.89e-04	0.0	128	False	NERSC	None	73.13%	1.66	85.39%
2.1	7	5.17e-5	0.0	128	True	NERSC	Ignore_11	72.48%	1.72	84.23%
2.2	7	2.57e-4	0.3	128	True	SAR	None	72.33%	1.90	80.69%
2.3	6	6.74e-5	0.1	128	True	NERSC	Ignore_11	72.41%	1.91	80.64%
3.1	**6**	**1.09e-4**	**0.0**	**128**	**False**	**NERSC**	**None**	**67.06%**	**1.62**	**86.03%**
3.2	8	2.66e-4	0.1	64	False	NERSC	None	66.83%	1.87	81.32%
3.3	6	4.39e-4	0.1	128	False	SAR	None	72.43%	1.90	80.84%

networks from any of the experiments R^2 score wise and are even fairly similar to the U-Net baseline mentioned in Section 6.1.2. These two best performing networks have a high variance in performance, for example network 1.1 has a 5% higher accuracy compared to network 3.1. The difference in prediction can be seen in Figure 11. Based on this single scene, a visual inspection confirms this, as it suggests that the SIC produced by network 1.1 has less artifacts than the SIC produced by network 3.1.

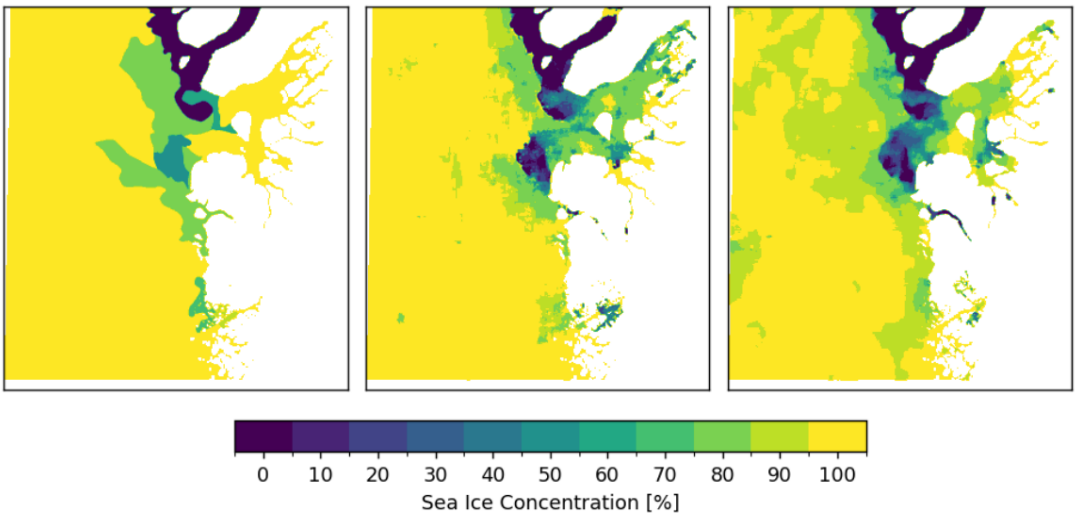


Figure 11: The predictions for the small-scale R^2 experiment. The target is on the left, the prediction of network 1.1 in the middle, and the prediction of network 3.1 on the right. The R^2 scores for network 1.1 and 3.1 are 63.36% and 69.35% respectively.

6.5.3 Discussion

To improve this experiment and gain a better spread of trials the maximum number of trials per sub-experiment can be set to 50. When additional computational time is available it would be better to run more sub-experiments with a maximum number of trials set to 50 than to run more trials per sub-experiment.
Due to the results obtained in this experiment, it can be concluded that the large-scale R^2 experiment from Section 6.4 found a local optimum. Network 1.1 has a negligibly worse R^2 score than network 3.1, but significantly beats network 3.1 when it comes to accuracy. Hence, network 1.1 generally outperforms network 3.1.
When comparing network 1.1 to the best network of the small-scale accuracy experiment from Section 6.2 it is harder to say which one is better as this depends on the optimisation criteria. The accuracy optimisation network has a significantly worse R^2 score of 84.39% compared to 86.01%, but a significantly higher accuracy of 79.01% compared to 72.30%. Here the best network can only be chosen based on a trade-off. A higher accuracy gives higher precision as more areas will be predicted precisely correct, while a higher R^2 score gives a prediction that is on average closer to the target, as every pixel is on average less far from the target.

7 Limitations

In this section, we will discuss three limitations in our work.

7.1 Result comparison

Because of the ambiguity in the data, some areas are excluded from the evaluation, as was explained in Section 4.5. This makes a direct comparison of results between this paper and previous work conducted not possible. The areas are not so large that it will drastically alter the results obtained, but it is best if previous results obtained by others are used as a guideline to indicate how well the networks in this paper perform.

7.2 Training weights

All experiments in this paper show that having a weight of 1 for all classes is the best option to get the highest performing network. This is something that should be investigated further as it initially seems strange that excluding the land values, a class that is not counted towards performance measurements, results in a worse network. The same goes for the weight distribution based on the class distribution in the dataset. The influence of class weights should be tested in isolated scenarios, instead of the current environment where at the same time multiple hyperparameters are evaluated. Hopefully, this can explain the reason why using a weight of 1 for all classes is found to be the best hyperparameter option.

7.3 Same validation test split

We have followed the work by Stokholm et al. [SWK+22] in using the same scenes as both validation set as test set. This was motivated by the low number of scenes available, and the need for

scenes as training data. However, this has as a consequence that the validation set (used for the hyperparameter optimisation process) is also the test set (for objectively determining the performance of the final model). This is generally considered bad practice as it may lead to overfitting on the validation and test set and hence the model may handle new data much worse than the data the model has been trained on. This drawback is slightly reduced by handpicking the validation and test set to be representative of the data distribution. A solution should however be found to split the validation and test set into two different sets.

8 Conclusions and further research

In this work, we applied Bayesian optimization to automatically find the best hyperparameters of a U-Net predicting sea ice concentrations. For this, we created a parallel architecture consisting of a chief and workers to run the Bayesian optimisation algorithm with a balance of speed and precision. We analysed the best performing hyperparameter configurations and concluded from this that running multiple smaller Bayesian optimisation algorithms results in better performing networks. We raise the issue of ambiguous data and provide a new evaluation method that deals with this ambiguity within the data. Utilizing this automated machine learning approach lowers the time investment to find an appropriate neural network for this task.

Many improvements can however still be made. Additional ways to prevent over-fitting like data augmentation could be implemented. Mirroring and rotating scenes can help provide slightly different examples that may improve the network performance and increase the data size at the same time. This may be of use when creating a separate validation and test set.
Additionally, instead of using the newly implemented evaluation method the cause of the ambiguity in the data should be found. Because a large part of the ambiguity is located within fjords, by ignoring these areas one also ignores a large part of diversity within the data as the neural networks now have a lot fewer examples of fjord regions to base their predictions on.
As mentioned in the limitations section additional research should be conducted on the influence of class weights. This may further improve network performance or at least provide the knowledge necessary to explain why certain class weights are better than others.

Acknowledgment

The authors would like to thank Andreas Stokholm of the Technical University of Denmark (DTU) and Andrzej Kucik of the European Space Agency (ESA) for sharing their network code and providing answers to any questions that have arisen about their network code and the paper written about said network. Secondly, the authors would like to thank Julia Wasala of the Leiden Institute of Advanced Computer Science (LIACS) for helping set up a working environment within the ALICE High Performance Computing facility. In addition, the authors would like to thank Hanno van Maanen for intensive help with debugging and for meaningful input on design choices for the AutoML network.

References

[BFRR17] Eddy Bekkers, Joseph Francois, and Hugo Rojas-Romagosa. Melting ice caps and the economic impact of opening the Northern Sea route. *The Economic Journal*, 128(610):1095–1127, 2017.

[DGCL21] Iris De Gelis, Aurélien Colin, and Nicolas Longepe. Prediction of categorized sea ice concentration from sentinel-1 SAR images based on a fully convolutional network. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 14:5831–5841, 2021.

[Fun09] McKenzie Funk. Arctic landgrap, 2009. https://www.nationalgeographic.com/magazine/article/healy.

[Gar23] Roman Garnett. *Bayesian Optimization*. Cambridge University Press, 2023.

[HMZ21] Konrad Heidler, Lichao Mou, and Xiao Xiang Zhu. Seeing the Bigger Picture: Enabling Large Context Windows in Neural Networks by Combining Multiple Zoom Levels. *IEEE Transactions on Geoscience and Remote Sensing*, 2021.

[JCSH23] Haifeng Jin, François Chollet, Qingquan Song, and Xia Hu. Autokeras: An automl library for deep learning. *Journal of Machine Learning Research*, 24(6):1–6, 2023.

[KS23] Andrzej S. Kucik and Andreas Stokholm. AI4SEAICE: Selecting loss functions for automated SAR Sea Ice concentration charting. *Scientific Reports*, 13(1), 2023.

[MHPK+21] David Malmgren-Hansen, Leif Toudal Pedersen, Matilde Brandt Kreiner, Roberto Saldo, Henning Skriver, J. W. Lavelle, Jørgen Buus-Hinkler, and Klaus Harnvig Krane. A convolutional neural network architecture for Sentinel-1 and AMSR2 data fusion. *IEEE Transactions on Geoscience and Remote Sensing*, 59(3):1890–1902, 2021.

[MHPN+20] David Malmgren-Hansen, Leif Toudal Pedersen, Allan Aasbjerg Nielsen, Henning Skriver, Roberto Saldo, Matilde Brandt Kreiner, and Jørgen Buus-Hinkler. Asip sea ice dataset - version 1, 2020. https://data.dtu.dk/articles/dataset/ASIP_Sea_Ice_Dataset_-_version_1/11920416.

[PMT+20] Donald K. Perovich, W. Meier, M. A. Tschudi, Stefan Hendricks, Alek Petty, Dmitry Divine, S. L. Farrell, Sebastian Gerland, Christian Haas, Lars Kaleschke, Olga Pavlova, Robert Ricker, Xiangshan Tian-Kunze, Melinda Webster, and Kevin R. Wood. Arctic Report Card 2020: Sea Ice. *Arctic Report Card 2020*, 2020.

[PWK+19] Jeongwon Park, Joong-Sun Won, Anton Korosov, M. Babiker, and Nuno Miranda. Textural noise correction for Sentinel-1 TOPSAR Cross-Polarization Channel Images. *IEEE Transactions on Geoscience and Remote Sensing*, 57(6):4040–4049, 2019.

[Ron17] Olaf Ronneberger. Invited talk: U-net convolutional networks for biomedical image segmentation. In *Bildverarbeitung für die Medizin 2017 - Algorithmen - Systeme - Anwendungen. Proceedings des Workshops*, Informatik Aktuell, page 3. Springer, 2017.

[RSC21] Keerthijan Radhakrishnan, K. Andrea Scott, and David A. Clausi. Sea ice concentration estimation: using passive microwave and SAR data with a U-Net and curriculum learning. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 14:5339–5351, 2021.

[SJH22] Qingquan Song, Haifeng Jin, and Xia Hu. *Automated Machine Learning in Action.* Manning, 2022.

[SKBH+20] Roberto Saldo, Matilde Brandt Kreiner, Jørgen Buus-Hinkler, Leif Toudal Pedersen, David Malmgren-Hansen, Allan Aasbjerg Nielsen, and Henning Skriver. Ai4arctic/asip sea ice dataset - version 2, 2020. https://data.dtu.dk/articles/dataset/AI4Arctic_ASIP_Sea_Ice_Dataset_-_version_2/13011134.

[SLA12] Jasper Snoek, Hugo Larochelle, and Ryan Prescott Adams. Practical bayesian optimization of machine learning algorithms. *CoRR*, abs/1206.2944, 2012.

[SWK+22] Andreas Stokholm, Tore Wulf, Andrzej S. Kucik, Roberto Saldo, Jørgen Buus-Hinkler, and Sine Munk Hvidegaard. AI4SEAICE: Toward solving ambiguous SAR textures in convolutional neural networks for automatic sea Ice concentration charting. *IEEE Transactions on Geoscience and Remote Sensing*, 60:1–13, 2022.

[vC24] Sven van Collenburg. Automated machine learning for sea ice concentration charting code, 2024. https://github.com/svenvanc/BSc-AutoML4SeaIce.

[WSC17] Lei Wang, K. Andrea Scott, and David A. Clausi. Sea ice concentration estimation during Freeze-Up from SAR imagery using a convolutional neural network. *Remote Sensing*, 9(5):408, 2017.

[WSCX17] Lei Wang, K. Andrea Scott, David A. Clausi, and Yan Xu. Ice concentration estimation in the gulf of st. lawrence using fully convolutional neural network. In *IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, pages 4991–4994, 2017.

[WSXC16] Lei Wang, K. Andrea Scott, Linlin Xu, and David A. Clausi. Sea Ice Concentration Estimation During Melt From Dual-Pol SAR Scenes Using Deep Convolutional Neural Networks: A Case Study. *IEEE Transactions on Geoscience and Remote Sensing*, 54(8):4524–4533, 2016.