1 Methodological framework

This section will first outline the theoretical background of convolutions from a deep learning point of view, as well as provide a brief overview of image segmentation as a computer vision task. Second, the methodological framework of the U-Net architecture, the deep neural network which is used in the present work, is outlined with a detailed description of its training loop and central algorithms. Finally, validation metrics which are used to asses the performance of the developed deep learning system will be described.

1.1 Convolutional layers

Convolutional layers incorporated into a deep neural network which are utilizing the backpropagation algorithm (Rumelhart et al., 1986) was initially proposed by LeCun et al. (1989) to classify handwritten numbers. The layer LeCun et al. (1989) presented consists of an arbitrary amount of filters, which are small two dimensional matrices (e.g. (3×3) pixels) designed to capture a certain structure in the image such as lines or edges . Each filter contains trainable weights, which are learned from the data during backpropagation (LeCun et al., 1989) and gradient descent. When a filter is convolved with all possible local neighborhoods from the input, it outputs a feature map which represent where the input image triggered a response from the filter (Zeiler et al., 2010). Moreover, inputting feature maps to a convolutional layer allows for the filters to respond to combinations of lower level structures, which trains the layer to detect more complicated patterns (Fukushima, 1980). Additionally, stacking convolutional layers in a network-architecture structure increases the field of view for each subsequent layer, which makes each layer observe an increasingly complex pattern of higher order feature maps at increasingly larger spatial scales (Fukushima, 1980). As a result, convolutional layers are able to discern between object and background as they perceive only a limited view of the scene. The convolutional layer is also invariant to the translation of the object, since the filter is constant when creating the feature map, i.e. the filter is detecting the same feature at all locations in the image, known as weight sharing (LeCun et al., 1989).

Sitere boka til Good-fel-low2016?

The number of trainable parameters for a convolutional layer is equal to the size of a filter times the number of filters. As a result, the number of trainable parameters is invariant to the spatial extent of the input images. Contrarily, fitting a fully connected layer to spatial gridded data consists of associating a separate trainable parameter to each pixel. As such, the size of a fully connected layer scales with the size of the image, which increases the risk of overfitting the network. In the case of the convolutional layer, LeCun et al. (1989) notes that reducing the number of trainable parameters through weight sharing constrains the solution space such that overfitting is avoided while still having enough trainable parameters to fit the layer to the data. Furthermore, the fully connected layer is not invariant to translation as each trainable parameter is exclusive to their respective pixel, hence no weight sharing. As such, the layer is unable to detect a similar object at a different position, reducing their usefulness for image-based prediction tasks.

Finally, Ciresan et al. (2012b) showed that the processing time of a convolutional layer is significantly shortened by utilizing a graphics processing unit (GPU), due to their large amount of compute cores compared to traditional Central Processing Units (CPUs). Furthermore, the authors of Krizhevsky et al. (2012) provided the first publicly available implementation of a CNN running on a GPU by utilizing the Nvidia Compute Unified Device Architecture (CUDA) api. Krizhevsky et al. (2012) also demonstrated that their results are tied to the performance of the GPU in terms of available memory as well as the rate of floating point operations per second, with the implication that a better GPU as well as larger datasets would improve their results. As such, both larger convolutional layers as well as deeper convolutional architectures written in a machine learning library which interacts with the GPU through CUDA (Jia et al., 2014; Abadi et al., 2015; Paszke et al., 2019) can be initiated. This is due to the speed up induced by the GPU, which allows for larger datasets to be processed, thus satisfying the increased parameter-count of the architecture.

since they can process greater datasets consisting of larger samples due to their GPU implementation.

The convolutional layer can be described mathematically by utilizing the previously described principle of allowing the filter to only perceive a local neighborhood of the input. Consider the value of a single point $y_{i,j} \subset Y \in \mathbb{R}^2$ where i, j denote the position in the x and y direction as a single output from a convolution. Let $X \in \mathbb{R}^3$ be an input image of size $(A \times B \times D)$ consisting of a single channel, and $W \in \mathbb{R}^3$ be a symmetric filter of size $(r \times r \times D)$. Then, the value at a single point $y_{i,j}$ is given as follows,

$$y_{i,j} = \sum_{a=1-\frac{r}{2}}^{\frac{r}{2}} \sum_{b=1-\frac{r}{2}}^{\frac{r}{2}} \sum_{d=1}^{D} W_{a+\frac{r}{2},b+\frac{r}{2},d} X_{i+a,j+b,d} \tag{1}$$

Where the subscript notation is used in W and X to denote indexes similar to Y. Equation (1) is described graphically in figure (1)

Repeating equation (1) across all points $x \subset X$ by applying a sliding window technique returns the convolution of X with filter W, which results in an output Y with size $(A-r+1) \times (B-r+1)$. Note that the above definition only applies for $X_{1 \leq i+a \leq A, 1 \leq j+b \leq B}$. The size of the output can be adjusted by padding the input X by a size P in each direction or increasing the stride S of the sliding window, which reformulates the output size of Y in a single dimension as a function

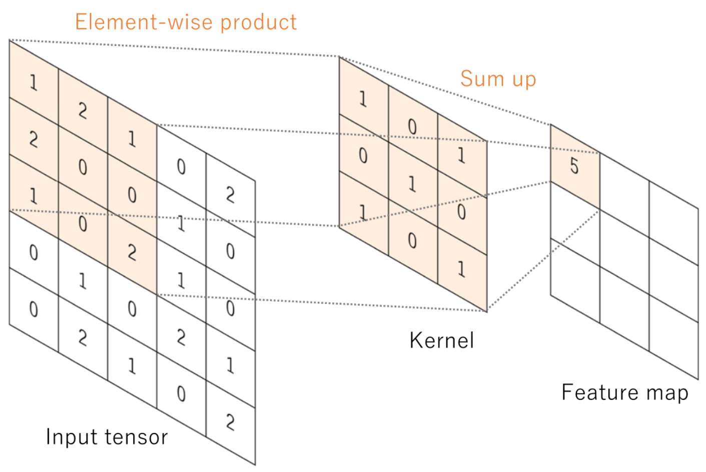


Figure 1: Convolution applied at a single point given a two dimensional input. Figure adapted from (Yamashita et al., 2018).

$$Y_{\text{dim}} = \lfloor \frac{A - r + 2P}{S} + 1 \rfloor \tag{2}$$

The convolutional layer adds the convolution described in equation (1) with a bias term $B \in \mathbb{R}^2$ of the same spatial shape as Y, as well as applying an activation function g to each $y_{i,j}$ which introduces nonlinearity. In summary, the output of a convolutional layer can be described as

$$Y' = g(Y + B) = g(W^T X + B) \tag{3}$$

If the number of filters increases from 1 to N, equation (3) is repeated for all filters, resulting in an output $Y \in \mathbb{R}^3$ of size $(Y_{\text{dim1}}, Y_{\text{dim2}}, N)$.

1.2 Image segmentation

Image segmentation is a computer vision task where pixels are assigned labels according to some predetermined rules. It is common to define an image segmentation task either as a study of countable *things* (Instance segmentation), or recognizing similarly textured *stuff* (Semantic segmentation) (Kirillov et al., 2018). The task for this thesis, which is labeling sea ice concentration according to its predicted concentration class, falls into the latter category following the definition of *stuff* in Adelson (2001). I.e. the current task is to assign each pixel in a predicted scene a single class label.

Network architectures based on the Convolutional Neural Network (CNN) (LeCun et al., 1989; Ciresan et al., 2012b; Krizhevsky et al., 2012; Simonyan and Zisserman, 2014; Szegedy et al., 2014; He et al., 2015b; Huang et al., 2016) can be used to perform pixelwise semantic segmentation, however the CNN architectures listed have been developed for image classification i.e. predicting a single label for the entire image. Ciresan et al. (2012a) presented an approach where a CNN (see the architecture of Ciresan et al. (2012b)) was used to predict a label for all pixels in an image. Instead of processing the entire image at once, Ciresan et al. (2012a) applied a sliding window technique which predicted each pixel by using their surrounding neighborhood as input. However, due to only processing parts of the image at once, the segmentation algorithm in Ciresan et al. (2012a) is computationally expensive as the CNN must be run for all possible neighborhoods. Additionally, the context for each CNN is limited to the local neighborhood surrounding the pixel (Ronneberger et al., 2015).

To capture the global context of a scene, network architectures such as Long et al. (2015); Noh et al. (2015); Ronneberger et al. (2015); Badrinarayanan et al. (2017); Chen et al. (2018) implement the Encoder-Decoder architecture, where the entire input scene is first processed by a CNN-like architecture referred to as the Encoder to produce a signal. The signal is then used as input to a subsequent network which reconstructs the encoded signal to match the resolution of the original image through upsampling. Long et al. (2015); Ronneberger et al. (2015); Badrinarayanan et al. (2017) all apply the Deconvolution architecture proposed by Zeiler et al. (2010) to upsample the encoded signal through the use of a trainable deconvolutional layer, which will be described in greater detail in Subsection (1.3.3). However, other upsampling techniques exists, such as unpooling used in Noh et al. (2015) which performs a upsampling by performing the opposite operation of a maxpool layer (maxpooling is described in Subsection (1.3.2)).

This thesis will utilize the U-Net architecture proposed by Ronneberger et al. (2015). The U-Net was initially developed for medical image segmentation, however the architecture has shown promising results for both pan-arctic seasonal (Andersson et al., 2021) and regional short term (Grigoryev et al., 2022) sea ice concentration forecasting amongst other applications. Another aspect which makes the U-Net more suitable to the current task, compared to other previously mentioned image-to-image architectures is that the network converges quickly, which is ideal when working with a dataset consisting of few samples (Ronneberger et al., 2015).

1.3 Describing the U-Net architecture

Figure (2) shows the U-Net architecture. This section intends to describe the different components constituting the architecture from a technical point of view.

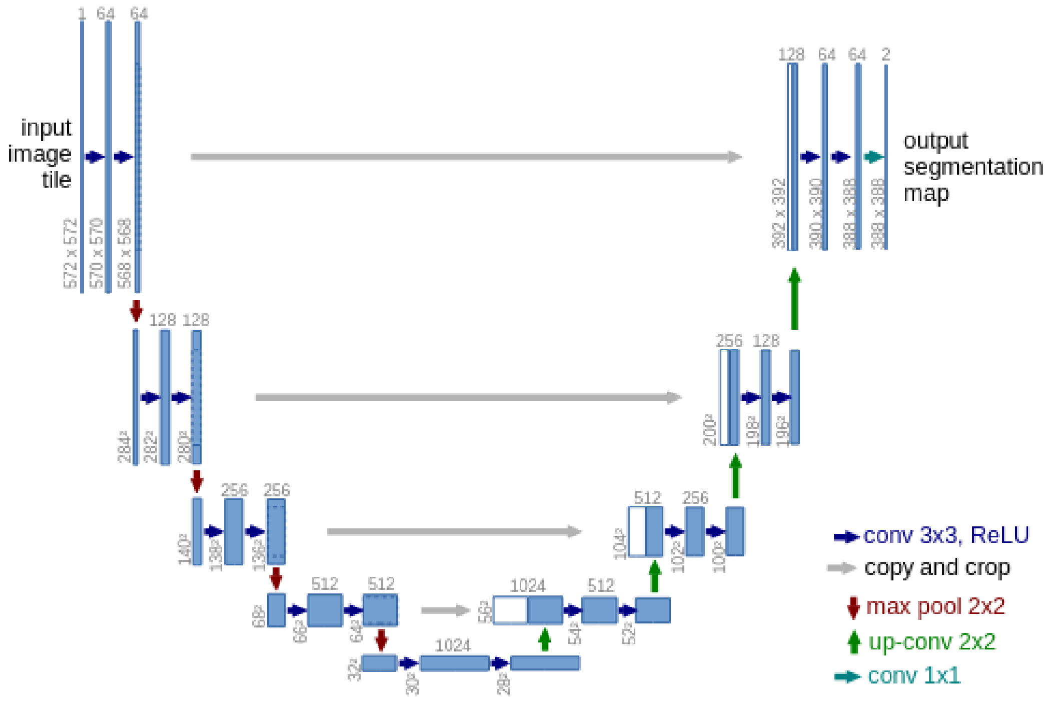
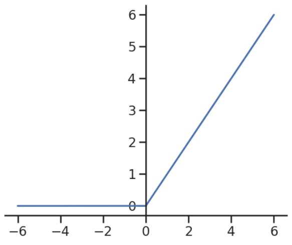


Figure 2: The U-Net architecture. The blue boxes represent feature maps, with the lower left numbers determining the spatial resolution and the top number the amount of feature maps. White boxes in the expansive path (right side / decoder) are the copied feature maps from the contractive path (left side / encoder). Arrows denote the different operations. Note that the original U-Net only performs *valid* convolutions, i.e. convolution without padding to match the input. This causes a convolutional layer to slightly decrease the spatial extent. As a result, the copied features from the contracting path are also cropped to match the dimensionality in the expansive path. Figure extracted from Ronneberger et al. (2015).

1.3.1 The convolutional block

A single convolutional block consists of two repeat convolutional layers, each followed by the Rectified Linear Unit (ReLU) (Nair and Hinton, 2010) nonlinear activation function. The ReLU activation function is defined as follows

$$f(x) = \max(0, x) \tag{4}$$



The ReLU function, similar to other activation functions used in deep neural networks, introduce non-linearities to the connections in the network. Thus the network is able to learn non-linear connections in the data.

Each convolution is performed using a 3×3 window. The original formulation of the U-Net also does not apply padding to the input, resulting the convolutional filter only being applied to the entries of the input where the filter is never out of bounds. With a stride $S = 1$, this results in each convolutional layer reducing the spatial extent by two pixels in each direction following equation (2). It is also noted that the number of feature maps is doubled after each downsampling step, which is performed by the pooling layers.

1.3.2 Maxpooling

Pooling operations are used to reduce the spatial extent of the current feature maps, by downsampling the data in the spatial dimensions. As seen in Figure (2), the U-Net downsamples the data in the contracting path through 2×2 maximum pool layers with a stride of 2. This specific configuration causes the spatial resolution to be halved. In the max-pool layer, a filter runs through each input channel and and chooses the maximum value inside the neighborhood of the filter. As such, the extreme values in each feature map is retained at the expense of rejecting the rest of the data. Since the maxpooling operation is rejecting some parts of the data, it may be regarded as a regularizer for the network which aid in keeping the model generalized, which is a topic further explored in the final paragraph of section 1.4. See Figure (3) for a graphical description.

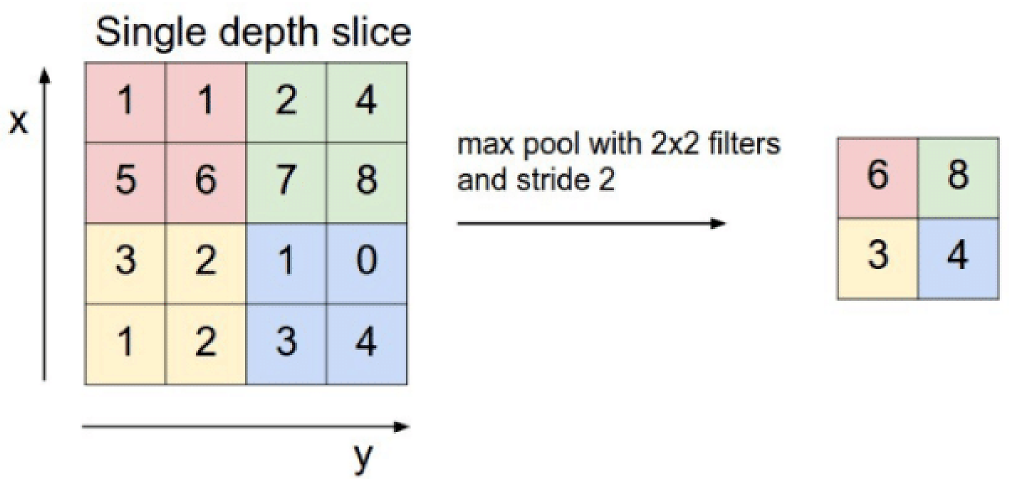
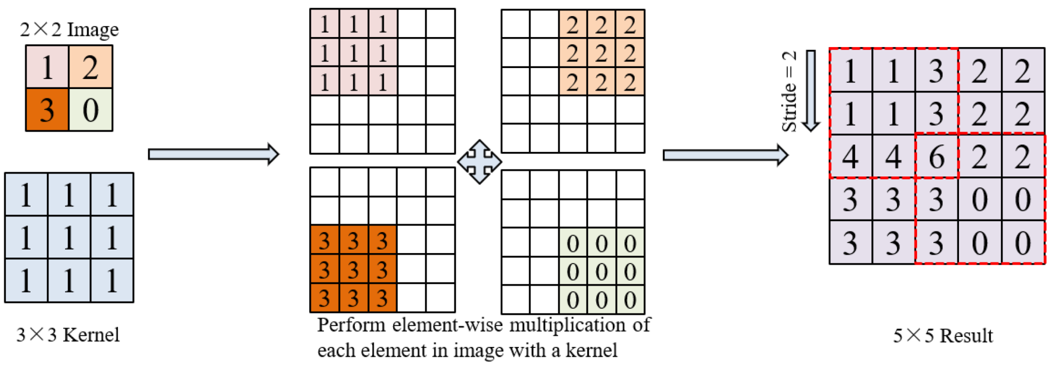


Figure 3: The max-pool operation for a 2×2 filter with a stride of 2. Figure taken from (RADU et al., 2020)

1.3.3 Transposed convolutions

Transposed convolution was proposed by Zeiler et al. (2010) (note the incorrect use of deconvolution, this is not the mathematical inverse of a convolution) to increase the resolution of a feature map. The method was first utilized by Long et al. (2015) to connect the coarse output of an encoder with the image resolution of the target (it is referred to as both *backwards convolution* and *deconvolution* in the proceedings paper). Similar to the convolutional layer, the transposed convolutional layer involves striding a convolutional filter with trainable parameters across a feature map. However, the transposed convolutional layer projects a singular entry from the input through the convolutional kernel to produce an output that is larger than the input. Figure (4) shows a graphical description of transposed convolutions.

In the Encoder architecture, lower level feature maps provide spatial information regarding where stuff is located in a scene, whereas higher level feature maps contain information regarding what is in the scene at the expense of losing spatial information (Long et al., 2015). To circumvent this, Ronneberger et al. (2015) concatenate the features from the contracting path with the output from the transposed convolution at the same level of depth, i.e. where the number of feature maps are equal at the end of the convolutional block. The concatenation operation is possible in Ronneberger et al. (2015) since they crop the feature maps in the encoder in their spatial dimensions to match the spatial dimensionality of the feature maps in the decoder. The operation can be seen in Figure(2) denoted by the gray arrow. The resulting convolutional layer is then trained to make a more precise prediction due to the concatenated input (Ronneberger et al., 2015).



2×2 Image
3×3 Kernel
Perform element-wise multiplication of each element in image with a kernel
Stride = 2
5×5 Result

Figure 4: Figure demonstrating the computations performed by a transposed convolutional layer. Figure adapted from (Wu et al., 2022)

1.3.4 Outputs

The output layer of the U-Net is denoted by the turquoise arrow at the right side of Figure (2). The arrow denote that the input is processed by a convolutional layer which have as many filters as there are output classes. Each filter is of size (1×1) with stride $S = 1$ and maps each layer in the input feature map to their respective class probability map of equal spatial shape (Ronneberger et al., 2015). By inspecting Figure (2), the U-Net outputs two feature maps, and from each feature map the pixelwise probability of belonging to the associated class can be computed.

1.4 Training procedure for the U-Net

This subsection aims to demonstrate how Ronneberger et al. (2015) trained the U-Net, and will consequently highlight some different hyperparameters and exemplify some functions and operations which are used in the training. Hyperparameters refer to model parameters which are not updated during training (Yu and Zhu, 2020), and may directly influence the model architecture or the training procedure. This section will not describe how samples were preprocessed and loaded, and modifications made which reflects concerns regarding medical images may be noted but not explained.

Training the U-Net starts by assigning random values to the weights of the network. Since the U-Net utilizes the ReLU activation function after each convolutional layer in the convolutional blocks (Ronneberger et al., 2015), it is standard for each layer to draw the weights from a normal distribution with $\mu = 0$ and standard deviation $\sigma = \sqrt{\frac{2}{n_l}}$, where n_l is the number of inputs to the layer He et al. (2015a). This weight initialization scheme ensures that variance of the feature maps are approximately equal, i.e. avoids varying the activation of input signals between layers He et al. (2015a); Ronneberger et al. (2015).

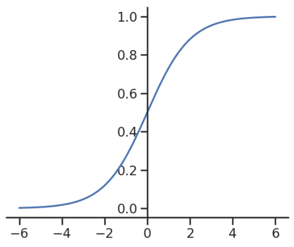
The process of training the U-Net involves making predictions on all training data. For each sample, the prediction is compared against a ground truth label. For the U-Net, a pixelwise prediction map is created by computing the pixelwise softmax which is an extension of the softmax function (Bridle, 1990) defined as

$$p_{k,i,j} = \frac{e^{a_{k,i,j}}}{\sum_{k'=1}^{K} e^{a_{k',i,j}}} \tag{5}$$

where a_k is the feature map for feature channel k of input x and K is the number of output classes and $p \in [0,1]$. i, j are the spatial coordinates. Similarly to the standard softmax function (Bridle, 1990), equation (5) is ≈ 1 for the class that has maximum $a_{k,i,j}$ and ≈ 0 for all other classes, albeit depthwise in the channel dimension for all pixels (Ronneberger et al., 2015). The sum of the depthwise output from the pixelwise softmax is 1, hence the function maps each pixel with the probability of that pixel belonging to each class.

For the case of binary classification, i.e. when the number of classes $k = 2$, the softmax function in equation (5) is reduced to the Sigmoid function which is defined as,

$$p_{i,j} = \frac{e^{a_{i,j}}}{e^{a_{i,j}} + 1} \tag{6}$$



To quantify the prediction error, a loss function is defined. The overall goal of training a neural network is to minimize the loss function with respect to the trainable weights. For the U-Net, a weighted variation of the cross entropy loss function is proposed (Ronneberger et al., 2015).

$$L(p) = \sum_{i,j \subset \mathbb{Z}^2} w_{i,j} log(p_{l,i,j}) \tag{7}$$

where w is a predefined weight map and $p_{l,i,j}$ is the prediction made at pixel i, j at the true label l.

The error computed by the loss function is then sent backwards throughout the network according to the backpropagation algorithm (Rumelhart et al., 1986), which effectively computes the gradient of the loss function with regards to the trainable parameters

$$\frac{\partial L}{\partial w_l} = \frac{\partial L}{\partial p}\frac{\partial p}{\partial w_l} \tag{8}$$

where w_l is the trainable parameters associated with the l-th layer. The gradient of the loss for a weight at a given layer shown in equation (8) is used by an optimizer to adjust the weights such that the loss is minimized with respect to the weights (gradient descent).

Ronneberger et al. (2015) uses the stochastic gradient descent with momentum optimizer implemented in the Machine Learning library Caffe (Jia et al., 2014), where the optimizer is defined as follows,

$$w_l^{t+1} = \gamma(w_l^t - w_l^{t-1}) - \mu\frac{\partial L}{\partial w_l^t} \tag{9}$$

In equation (9), the superscript t was added to w and refers to training step, which is defined as a prediction and subsequent backpropagation of a batch of samples, where the size of a batch is a pre-determined hyperparameter. γ and μ are the momentum and learning rate hyperparameters respectively. Note that γ is introduced by momentum stochastic gradient descent, whereas the learning rate μ is a hyperparameter common for all deep learning models and determines the rate of weight adjustment as seen in equation (9).

When all training samples have been inspected once by the U-Net, the training data is shuffled and the above outlined training procedure is repeated. The process of going through all the training data once is defined as an epoch. The number of epochs is a hyperparameter which can be adjusted, and is tied to the bias-variance tradeoff dilemma (Geman et al., 1992). Moreover, the number of epochs determines the duration of training time, and is influenced by the available computing resources.

Geman et al. (1992) states that the cost of low bias in a model is high variance. A model with high bias and low variance is assumed to not have underwent much, if any training, and is thus underfitted to the data. Consequently, a model with low bias but high variance has been trained for a high number of epochs, and is overfitted towards the training-data. An overfitted model is, due to its high variance, ideal at explaining the training data, but lacks the ability to generalize to external datasets. For the training procedure described above, the optimum model has been trained for a sufficient amount of epochs, where it is neither underfitted nor overfitted.

1.5 Forecast verification metrics

Verification schemes provide insight into how a forecasting system performs. For this thesis, verification metrics serve a dual purpose. From a model development point of view, verification metrics will be used to increase the skill of the model. However, the same metrics will also be utilized to assess the quality of a prediction as well as explain the physical interpretation of the model (Casati et al., 2008). The model developed for this thesis predicts a scene consisting of labelled pixels, as described in section (1.2). It was mentioned in section (1) that the developed model is aimed towards operational end users, which is partly achieved by validating the model against metrics of end user relevance. Furthermore, it can be assumed that the model and target observations will not differ much outside of the marginal ice zone (Fritzner et al., 2020). Thus, this section will introduce metrics which are relevant for evaluating the sea-ice edge position, as the sea ice edge is important information for maritime operators in the Arctic (Melsom et al., 2019). The following subsections will describe how to determine the position of the sea ice edge, as well as its length according to Melsom et al. (2019), and derive the Integrated Ice Edge Error (Goessling et al., 2016), with regards to a spatially gridded dataset of deterministic sea ice concentration values.

The Integrated Ice Edge Error is chosen among similar sea ice edge metrics (Melsom et al., 2019; Dukhovskoy et al., 2015) as it has been shown to be less sensitive to isolated ice patches (Palerme et al., 2019). Furthermore, the work of Melsom et al. (2019) recommends the Integrated Ice Edge Error amongst other metrics for its intuitive interpretation as well as for the possibility to provide the spatial distribution of IIEE areas.

1.6 Root Mean Square Error

The Root mean square error (rmse) is a commonly used metric which measures the difference between two points in a sample. Dukhovskoy et al. (2015) presents the rmse for sea ice applications as a measure of the pixelwise difference between model data and control data. Moreover, Dukhovskoy et al. (2015) defines rmse in two dimensions as follows,

$$D_{\mathrm{RMSE}}(A, B) = \sqrt{\frac{\sum_{i=1}^{n} \left[(a_i, b_i)\right]^2}{n}} \tag{10}$$

In equation (10), A and B refer to the model data and control data sets containing n samples. a, b $\subset$ A, B subscripted by i.

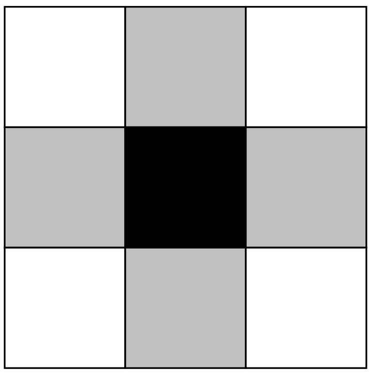


Figure 5: The gray pixels forms the 4-connected neighborhood of adjacent grid cells for the center pixel.

1.6.1 Defining the Sea Ice Edge

The sea ice edge for a given spatial distribution of sea ice concentration values is derived on a per pixel basis. Let $C \in \mathbb{R}^2$ be gridded sea ice concentration values. Then, the sea ice edge is defined as the entries in C which meets the following condition,

$$c_{i,j} \geq c_e \wedge \min(c_{i-1,j}, c_{i+1,j}, c_{i,j-1}, c_{i,j+1}) < c_e \tag{11}$$

In Condition (11), $c \subset C$ are sea ice concentration values, with i, j denoting indexes. c_e is a given concentration threshold.

Next, let $E \in \mathbb{R}^2$ be the set containing sea ice concentration pixels constituting the sea ice edge. It can be seen that the entries $c_{i,j}$ which adhere to condition (11) form the set E (Melsom et al., 2019).

Moreover, all the entries in E each contribute to the total length of the sea ice edge, with each entries' length contribution determined based on that entries' 4-connected adjacent grid points, (see figure 5). Using this formulation, the different combination of neighborhoods in E can result in three different length contributions. For the following contributions, s is the spatial resolution of the grid.

- A neighborless pixel is assumed to yield a contribution equal to the length of the diagonal of a grid cell ($l = \sqrt{2}s$). Here it is assumed that the grid cell only have diagonal neighbors ($\mathrm{e_a}$ in figure 6).
- A pixel with one of the four possible adjacent grid points contributes with the mean value between the length of the grid cell and length og the diagonal of the grid cell $l = \frac{s+\sqrt{2}s}{2}$. It is assumed that the grid cell also has a diagonal neighbor ($\mathrm{e_b}$ and $\mathrm{e_e}$ in figure 6).

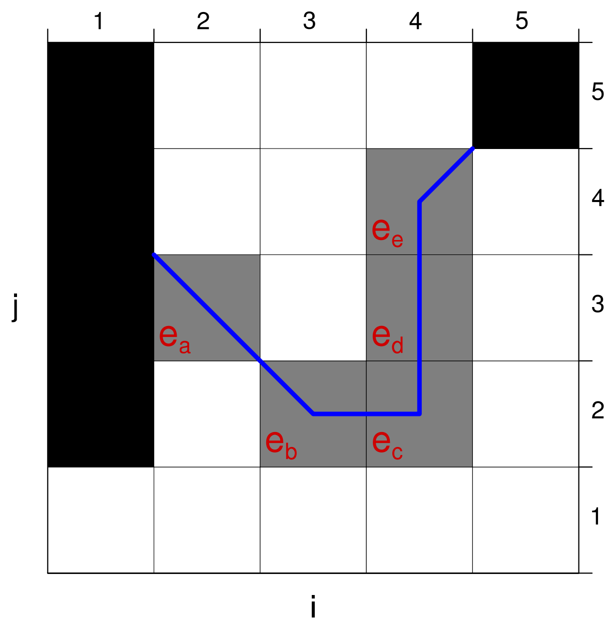


Figure 6: Sketch of an example gridded ice edge. The gray cells denote ice edge cells, which are labelled and illustrates the ice edge contained in the cell. The black cells denote land. Figure fetched from (Melsom et al., 2019)

- A pixel with two or more of the four adjacent grid points contributes with its spatial resolution (length of the grid cell) $l = s$ (e_c and e_d in figure 6).

The final length of the sea ice edge length then becomes

$$L = \sum_{\text{e in E}} l^e \tag{12}$$

where the superscript l^e denotes the length associated with the entry e according to the algorithm listed above, I.e. the sum of all contributions.

1.6.2 Integrated Ice Edge Error

The IIEE is an error metric which compares a forecast f to a predefined ground truth target t Goessling et al. (2016). The metric is defined as

$$\text{IIEE} = \text{O} + \text{U} \tag{13}$$

where

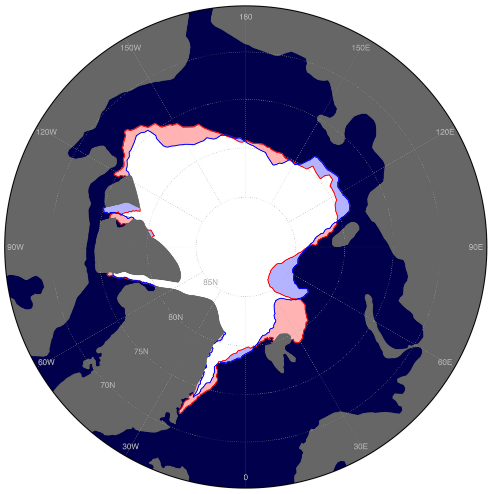


Figure 7: 15% sea ice concentration contours for a forecast (blue) and target (red) sea ice concentration product. The IIEE is the sum of the overestimated (O, blue) and underestimated (U, red). White denotes the union between the products. Figure fetched from (Goessling et al., 2016).

$$\mathrm{O} = \int_A \max(C_f - C_t, 0) dA \tag{14}$$

and

$$\mathrm{U} = \int_A \max(C_t - C_f, 0) dA \tag{15}$$

with $A \in \mathbb{R}^2$ being the area of interest, and is of similar size as C. Subscript f, t denotes whether C contains forecasted or target sea ice concentration values. In Equations 14 and 15, C is binary and is equal to 1 if its concentration value is above a predefined threshold, and 0 elsewhere (Goessling et al., 2016). From the definition of the metric, it can be seen that the IIEE is a sum of the forecast overshoot and undershoot compared to the ground truth target. For a graphical description, see figure (7)

Additionally, the IIEE can also be represented as a spatial metric by removing the integral with respect to A in equation (14 and 15). In this way, the metric is used to define the set of pixels which constitutes its area. To clearly distinguish between the area O (overestimation) and the set of pixels used to compute O, A^+ will be used to note the latter. Similarly, A^- will represent the set of pixels constituting U (underestimation). Finally, it can be seen that A^+ and A^- represent the spatial distribution of False Positive and False Negatives of the forecast respectively.

The length of the ice edge has a strong influence on the IIEE (Goessling and Jung, 2018; Palerme et al., 2019). Hence, to ensure that forecast errors are comparable across seasons, IIEE is normalized with the length of the ice edge, as mentioned in section (1.2.2). Furthermore, the normalized IIEE provides an estimate of the displacement error between the forecasted and target sea ice edge (Melsom et al., 2019).

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X.: TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, URL https://www.tensorflow.org/, software available from tensorflow.org, 2015.

Adelson, E. H.: On seeing stuff: the perception of materials by humans and machines, in: SPIE Proceedings, edited by Rogowitz, B. E. and Pappas, T. N., SPIE, https://doi.org/10.1117/12.429489, 2001.

Andersson, T. R., Hosking, J. S., Pérez-Ortiz, M., Paige, B., Elliott, A., Russell, C., Law, S., Jones, D. C., Wilkinson, J., Phillips, T., Byrne, J., Tietsche, S., Sarojini, B. B., Blanchard-Wrigglesworth, E., Aksenov, Y., Downie, R., and Shuckburgh, E.: Seasonal Arctic sea ice forecasting with probabilistic deep learning, Nature Communications, 12, https://doi.org/10.1038/s41467-021-25257-4, 2021.

Badrinarayanan, V., Kendall, A., and Cipolla, R.: SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation, IEEE Transactions on Pattern Analysis and Machine Intelligence, 39, 2481–2495, https://doi.org/10.1109/tpami.2016.2644615, 2017.

Bridle, J. S.: Probabilistic Interpretation of Feedforward Classification Network Outputs, with Relationships to Statistical Pattern Recognition, in: Neurocomputing, pp. 227–236, Springer Berlin Heidelberg, https://doi.org/10.1007/978-3-642-76153-9_28, 1990.

Casati, B., Wilson, L. J., Stephenson, D. B., Nurmi, P., Ghelli, A., Pocernich, M., Damrath, U., Ebert, E. E., Brown, B. G., and Mason, S.: Forecast verification: current status and future directions, Meteorological Applications, 15, 3–18, https://doi.org/10.1002/met.52, 2008.

Chen, L.-C., Papandreou, G., Kokkinos, I., Murphy, K., and Yuille, A. L.: DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs, IEEE Transactions on Pattern Analysis and Machine Intelligence, 40, 834–848, https://doi.org/10.1109/tpami.2017.2699184, 2018.

Ciresan, D., Giusti, A., Gambardella, L., and Schmidhuber, J.: Deep Neural Networks Segment Neuronal Membranes in Electron Microscopy Images, in: Advances in Neural Information Processing Systems, edited by Pereira, F., Burges, C., Bottou, L., and Weinberger, K., vol. 25, Curran Associates, Inc., URL https://proceedings.neurips.cc/paper/2012/file/459a4ddcb586f24efd9395aa7662bc7c-Paper.pdf, 2012a.

Ciresan, D., Meier, U., and Schmidhuber, J.: Multi-column deep neural networks for image classification, in: 2012 IEEE Conference on Computer Vision and Pattern Recognition, IEEE, https://doi.org/10.1109/cvpr.2012.6248110, 2012b.

Dukhovskoy, D. S., Ubnoske, J., Blanchard-Wrigglesworth, E., Hiester, H. R., and Proshutinsky, A.: Skill metrics for evaluation and comparison of sea ice models, Journal of Geophysical Research: Oceans, 120, 5910–5931, https://doi.org/10.1002/2015jc010989, 2015.

Fritzner, S., Graversen, R., and Christensen, K. H.: Assessment of High-Resolution Dynamical and Machine Learning Models for Prediction of Sea Ice Concentration in a Regional Application, 125, https://doi.org/10.1029/2020jc016277, neural Networks for predicting Sea-Ice concentration are only slightly more accurate than persistence forecasting for short-term predictions., 2020.

Fukushima, K.: Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position, Biological Cybernetics, 36, 193–202, https://doi.org/10.1007/bf00344251, 1980.

Geman, S., Bienenstock, E., and Doursat, R.: Neural Networks and the Bias/Variance Dilemma, Neural Computation, 4, 1–58, https://doi.org/10.1162/neco.1992.4.1.1, 1992.

Goessling, H. F. and Jung, T.: A probabilistic verification score for contours: Methodology and application to Arctic ice-edge forecasts, Quarterly Journal of the Royal Meteorological Society, 144, 735–743, https://doi.org/10.1002/qj.3242, 2018.

Goessling, H. F., Tietsche, S., Day, J. J., Hawkins, E., and Jung, T.: Predictability of the Arctic sea ice edge, Geophysical Research Letters, 43, 1642–1650, https://doi.org/10.1002/2015gl067232, 2016.

Grigoryev, T., Verezemskaya, P., Krinitskiy, M., Anikin, N., Gavrikov, A., Trofimov, I., Balabin, N., Shpilman, A., Eremchenko, A., Gulev, S., Burnaev, E., and Vanovskiy, V.: Data-Driven Short-Term Daily Operational Sea Ice Regional Forecasting, Remote Sensing, 14, https://doi.org/10.3390/rs14225837, URL https://www.mdpi.com/2072-4292/14/22/5837, 2022.

He, K., Zhang, X., Ren, S., and Sun, J.: Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification, https://doi.org/10.48550/ARXIV.1502.01852, 2015a.

He, K., Zhang, X., Ren, S., and Sun, J.: Deep Residual Learning for Image Recognition, 2015b.

Huang, G., Liu, Z., van der Maaten, L., and Weinberger, K. Q.: Densely Connected Convolutional Networks, https://doi.org/10.48550/ARXIV.1608.06993, 2016.

Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., and Darrell, T.: Caffe: Convolutional Architecture for Fast Feature Embedding, https://doi.org/10.48550/ARXIV.1408.5093, 2014.

Kirillov, A., He, K., Girshick, R., Rother, C., and Dollár, P.: Panoptic Segmentation, https://doi.org/10.48550/ARXIV.1801.00868, 2018.

Krizhevsky, A., Sutskever, I., and Hinton, G. E.: ImageNet Classification with Deep Convolutional Neural Networks, in: Advances in Neural Information Processing Systems, edited by Pereira, F., Burges, C., Bottou, L., and Weinberger, K., vol. 25, Curran Associates, Inc., URL https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf, 2012.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D.: Backpropagation Applied to Handwritten Zip Code Recognition, Neural Computation, 1, 541–551, https://doi.org/10.1162/neco.1989.1.4.541, 1989.

Long, J., Shelhamer, E., and Darrell, T.: Fully convolutional networks for semantic segmentation, in: 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, https://doi.org/10.1109/cvpr.2015.7298965, 2015.

Melsom, A., Palerme, C., and Müller, M.: Validation metrics for ice edge position forecasts, Ocean Science, 15, 615–630, https://doi.org/10.5194/os-15-615-2019, 2019.

Nair, V. and Hinton, G.: Rectified Linear Units Improve Restricted Boltzmann Machines Vinod Nair, vol. 27, pp. 807–814, 2010.

Noh, H., Hong, S., and Han, B.: Learning Deconvolution Network for Semantic Segmentation, in: 2015 IEEE International Conference on Computer Vision (ICCV), IEEE, https://doi.org/10.1109/iccv.2015.178, 2015.

Palerme, C., Müller, M., and Melsom, A.: An Intercomparison of Verification Scores for Evaluating the Sea Ice Edge Position in Seasonal Forecasts, Geophysical Research Letters, 46, 4757–4763, https://doi.org/10.1029/2019gl082482, 2019.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S.: PyTorch: An Imperative Style, High-Performance Deep Learning Library, https://doi.org/10.48550/ARXIV.1912.01703, 2019.

RADU, M. D., COSTEA, I. M., and STAN, V. A.: Automatic Traffic Sign Recognition Artificial Inteligence - Deep Learning Algorithm, in: 2020 12th International Conference on Electronics, Computers and Artificial Intelligence (ECAI), IEEE, https://doi.org/10.1109/ecai50035.2020.9223186, 2020.

Ronneberger, O., Fischer, P., and Brox, T.: U-Net: Convolutional Networks for Biomedical Image Segmentation, in: Lecture Notes in Computer Science, pp. 234–241, Springer International Publishing, https://doi.org/10.1007/978-3-319-24574-4_28, 2015.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J.: Learning representations by back-propagating errors, Nature, 323, 533–536, https://doi.org/10.1038/323533a0, 1986.

Simonyan, K. and Zisserman, A.: Very Deep Convolutional Networks for Large-Scale Image Recognition, https://doi.org/10.48550/ARXIV.1409.1556, 2014.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A.: Going Deeper with Convolutions, https://doi.org/10.48550/ARXIV.1409.4842, 2014.

Wu, M.-Y., Wu, Y., Yuan, X.-Y., Chen, Z.-H., Wu, W.-T., and Aubry, N.: Fast Prediction of Flow Field around Airfoils Based on Deep Convolutional Neural Network, Applied Sciences, 12, 12 075, https://doi.org/10.3390/app122312075, 2022.

Yamashita, R., Nishio, M., Do, R. K. G., and Togashi, K.: Convolutional neural networks: an overview and application in radiology, Insights into Imaging, 9, 611–629, https://doi.org/10.1007/s13244-018-0639-9, 2018.

Yu, T. and Zhu, H.: Hyper-Parameter Optimization: A Review of Algorithms and Applications, https://doi.org/10.48550/ARXIV.2003.05689, 2020.

Zeiler, M. D., Krishnan, D., Taylor, G. W., and Fergus, R.: Deconvolutional networks, in: 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, IEEE, https://doi.org/10.1109/cvpr.2010.5539957, 2010.