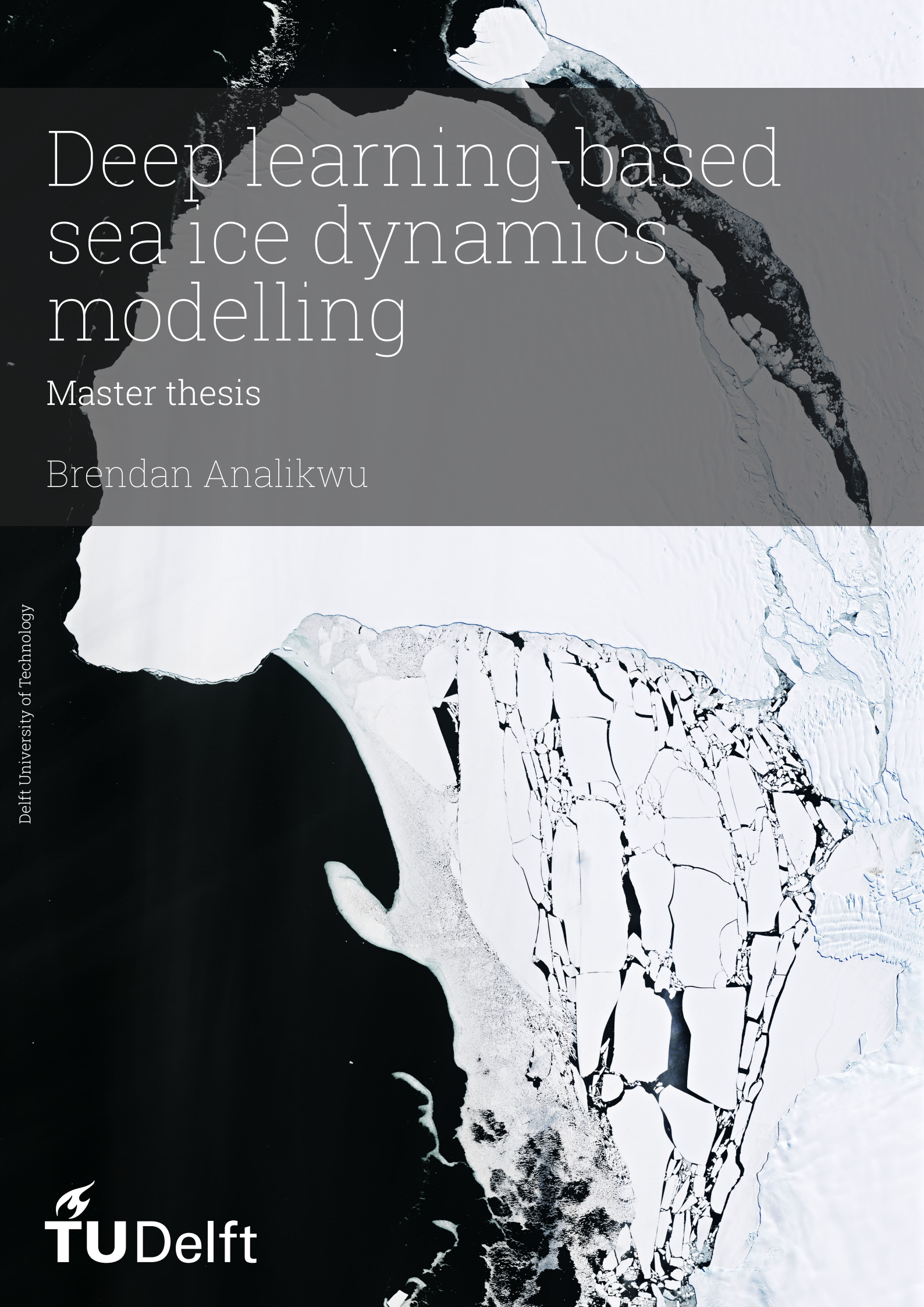
Deep learning-based sea ice dynamics modelling

Master thesis

Brendan Analikwu



Delft University of Technology

TU Delft

Deep learning-based sea ice dynamics modelling

Master thesis

by

Brendan Analikwu

Brendan Analikwu 4589076

Daily supervisors:	Dr. Alexander Heinlein, and Dr. Carolin Mehlmann
Committee chair:	Dr. Matthias Möller
Committee member:	Dr. Bernard Meulenbroek
Project Duration:	March 2023 - October 2024
Programme:	Master Applied Mathematics, Delft

TU Delft

Summary

In this thesis, a deep learning-based surrogate model for predicting sea ice dynamics is developed that is capable of predicting linear kinematic features in a high-resolution setting. Predicting sea ice dynamics at high resolutions is critical for understanding climate patterns and enabling safe navigation in Arctic regions. Traditional continuum models based on the viscous-plastic rheology are computationally intensive when employed at high resolutions to capture linear kinematic features (LKFs), which are narrow zones of deformation in sea ice.

A supervised learning approach was adopted, focusing on convolutional neural network architectures, specifically the U-Net and a classic bottleneck model. Various loss functions were explored, including traditional metrics like the MSE and novel domain-specific functions such as the strain rate error (SRE), which incorporates physical knowledge of sea ice behaviour. The models were trained on a dataset generated from numerical simulations of sea ice dynamics on a 2 km grid.

The key findings indicate that the U-Net architecture combined with the MSE+SRE loss function outperforms other models. This architecture's depth and use of skip connections allow it to capture complex, multi-scale patterns inherent in sea ice dynamics. The integration of the SRE into the loss function significantly enhances the model's ability to predict LKFs, demonstrating the benefit of incorporating domain knowledge into machine learning models.

While the surrogate model effectively predicts LKFs for short-term forecasts up to approximately 5.5 hours (10 time steps), errors accumulate over longer periods, leading to significant errors after about 11 hours (20 time steps). However, the efficiency gain of this surrogate model is striking: the computation of 10 time steps can be done within a second, compared to the 30 minutes that traditional numerical methods need. The study also underscores the critical importance of training data selection and preparation in influencing model performance and generalisation capabilities.

In conclusion, this work demonstrates that a deep learning-based surrogate model, particularly utilising the U-Net architecture and the MSE+SRE loss function, can effectively predict sea ice dynamics at high resolutions with significant computational efficiency. The model generates forecasts within seconds, offering a viable alternative to traditional numerical simulations that require hours of computation. Future work should focus on mitigating error accumulation to extend the forecasting horizon and exploring advanced learning methods to further integrate physical insights into the modelling process.

Acknowledgements

I would like to express my deepest gratitude to my supervisors, Alexander Heinlein and Carolin Mehlmann, for their guidance and support throughout this thesis. Their insights and constructive feedback during our meetings were invaluable in shaping the direction and quality of my research.

A heartfelt thank you goes to my parents, whose constant encouragement and support have been instrumental in my educational journey. Their belief in me has been a source of strength and motivation. I am also very grateful to my boyfriend Martien for being a great source of support and understanding during this journey.

Working on this thesis was both exciting and challenging. My strong interest in machine learning and its application to climate models motivated me throughout. Sea ice modeling with machine learning is a relatively new field, and much of my work involved exploring uncharted territory through trial and error. While I recognize the irony in addressing sustainability through AI - an area often criticized for its energy demands - I hope that this research contributes to more efficient modelling solutions. Balancing the demands of this individual research project with my responsibilities as a city council member was demanding at times, and I learned the importance of starting small and being patient with the process.

Brendan Analikwu
Delft, October 2024

Contents

1 Introduction

Sea ice dynamics play an important role in understanding climate patterns, predicting environmental changes and enabling safe navigation in the Arctic regions. Earth system models (ESMs), used in climate research to simulate long-term weather patterns, incorporate sea ice to account for its role in sunlight reflection (albedo) and interactions with the atmosphere [1, 2]. However, the complexity of sea ice dynamics modelling causes significant computational demands using current numerical methods. To address these challenges both new model formulations and alternative methods are currently being investigated [3].

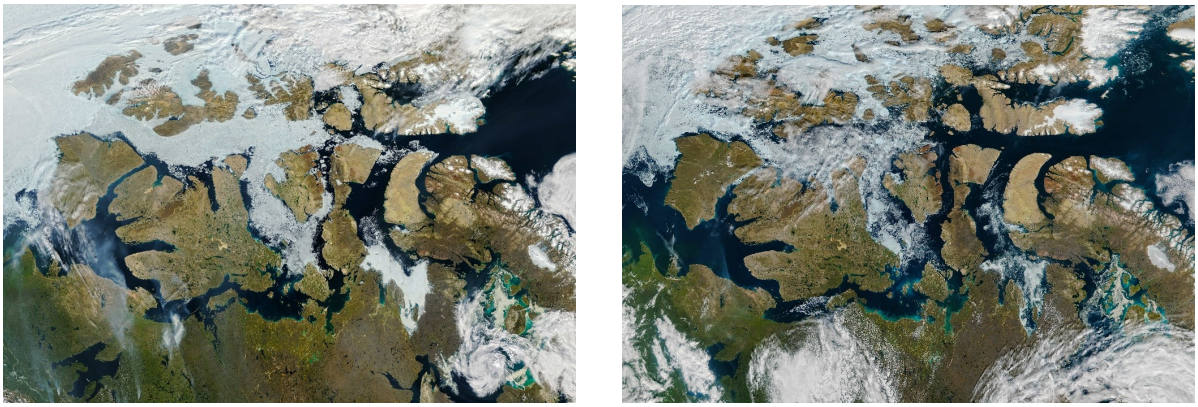
These studies primarily focus on improving the currently used solvers or applying numerical methods that were previously considered unsuitable for sea ice modelling. However, advancements in high performance computing, for example, have now made these methods feasible [3, p. 6]. Machine learning has gained traction as a powerful tool for scientific applications [4], helping them through hybrid approaches that integrate machine learning with traditional numerical methods [5, 6], or even replacing numerical methods completely [7, 8, 9]. In this thesis, we explore the potential of deep learning-based models applied to sea ice modelling, paving the way for improved understanding and prediction of sea ice behaviour.

1.1. Sea ice modelling

Due to its reflexivity and its role as a boundary layer between the atmosphere and the ocean, sea ice is an integral part of the Earth's system. Approximately 9 percent of the Earth's ocean surface is covered with ice during at least some part of the year [10]. Due to its reflexivity, changes in sea ice coverage have significant effects on the absorption rate of sunlight by the Earth, which therefore influence the planet's temperature. As sea ice coverage decreases, more heat is absorbed, contributing to global temperature rise and subsequent sea ice melting. This positive feedback loop is referred to as *ice-albedo feedback* and acts as an accelerator of global warming. In the scenario corresponding to 1.5 degrees warming, the disappearance of Arctic sea ice and the retreat of ice sheets across the world have been computed to be responsible for 29 percent of this increase [11, p. 3]. The importance of sea ice in the climate system is evident.

As the Arctic sea ice rapidly declines due to global warming, interest in the region as a navigation route has grown, with shorter shipping routes becoming increasingly accessible [12, p. 1631]. To illustrate, the Northwest Passage (which runs through the Canadian Arctic Archipelago) has been nearly impassible throughout recorded history [13]. Since 2007, however, the Northwest Passage has been fairly open at some point during the summer months [14] with the exception of 2013. A comparison of the ice extent in the Northwest Passage on August 9, 2013, and August 9, 2016, is shown in Figure 1.1.

Even though Arctic maritime routes are long desired, navigation through the Arctic does not come without controversy. On the one hand, these Arctic trade routes could cut greenhouse gas emissions due to their shorter lengths, but on the other hand, the increased risk of oil spills, for example, could prove more damaging in the Arctic than in other regions. These developments have also called for the use of sea ice models in a more 'weather-like' setting: predicting the navigability at high resolutions [15]. While these predictions have not been found to be very precise, simply predicting the pattern and



(a) (b)

Figure 1.1: Comparison of the ice extent in the Northwest Passage on August 9. Taken from [14]. (a) The ice extent in 2013 as observed by the Moderate Resolution Imaging Spectroradiometer (MODIS) on the Aqua satellite. (b) The image captured by the Visible Infrared Imaging Radiometer Suite (VIIRS) on the Suomi NPP satellite in 2016.

(a) (b)

Figure 1.2: Example of (a) a lead and (b) a pressure ridge from the MOSAiC expedition. Taken from [18]. The images were taken by Steven Fons from NASA's Goddard Space Flight Center.

direction of navigable routes might be enough [16, p. 17]. Ships can navigate through thin ice and cracks in the ice cover called *leads*. Phenomena as leads, but also pressure ridges, together referred to as linear kinematic features (LKFs), are only produced by models at resolution higher than 5 kilometres [17, p. 673]. This calls for sea ice models that operate at higher resolutions. Examples of a lead and a pressure ridge from the MOSAiC expedition are shown in Figure 1.2.

Even though ice on land plays an important role in the climate system, its physics differ from that of sea ice modelling. While both land and sea ice grow and shrink due to interactions with the atmosphere above, they are set apart by the physics of the interactions with the earth or ocean. Most important here is the greater mobility of sea ice. To illustrate, glaciers typically move by several metres per day, with some of the fastest in the world reaching roughly 40 metres per day [19, 20]. On the other hand, sea ice velocity modelling deals with typical speeds in the order of one metre per second. Their typical speeds therefore differ three orders of magnitude. As we will demonstrate, the key challenges in ice modelling emerge at higher rates of deformation, where sea ice exhibits plastic rather than fluid-like behaviour. Therefore, we focus on sea ice in this work, as land ice does not exhibit these higher rates of deformation due to the significantly lower speeds.

In sea ice modelling, two processes are captured: thermodynamics and dynamics. In this thesis, however, we only consider the latter. The study of sea ice dynamics is concerned with the drift of sea ice forced by, for example, winds and ocean currents. To capture this process, three variables are modelled: sea ice velocity, thickness and concentration. As sea ice dynamics models are traditionally used on larger scales, they are based on continuum mechanics. In this continuum mechanics approach,

averages over a sufficiently large number of floating pieces of ice, called ice floes, are taken. This leads to a scale of tens of kilometres, as the typical size of floes ranges from a few metres to several kilometres in diameter. The sea ice velocity is therefore the average speed of ice floes. Furthermore, the ice thickness is the average sea ice height in a given volume, corresponding to a measure of the ice mass in a cell. The ice concentration is then the fraction of the area in the cell covered by thick ice. The remainder is either open water or thin (or 'young') ice. According to Blockley et al. [3], continuum models are expected to remain in use for the foreseeable future, but their usage will depend on their performance at high resolutions.

Sea ice dynamics are influenced not only by interactions with wind and ocean currents, but also by internal stresses within the ice. These stresses are mostly modelled using a viscous-plastic (VP) rheology introduced by Hibler [21]. Presently, nearly all sea ice models use a formulation based on this viscous-plastic rheology [3]. The viscous-plastic sea ice model consists of two transport laws and a nonlinear momentum equation. Explicit time stepping schemes have been shown by Ip et al. [22] to be stable only for small time steps, negating their efficiency gain compared to implicit time discretisations with longer time steps. Therefore, Ip et al. [22] conclude implicit schemes to be the better choice. However, due to the nonlinearity of the momentum equation, iterative solvers must be used to solve problems with the VP rheology.

To allow for explicit time integration, Hunke and Dukowicz [23] introduced elasticity to the traditional viscous-plastic rheology. This resulted in the development of the elastic-viscous-plastic (EVP) model, which offers more computational efficiency compared to the explicit VP model, since it allows for larger time steps. However, the two models only converge to the same solution if the resolution is low enough and the subcycle timesteps of the EVP method are sufficiently small [24], resulting in an efficiency-accuracy trade-off. In conclusion, the VP model solved with an implicit scheme provides the most accurate results, despite its higher computational demands.

In sea-ice dynamics literature, the two common solution methods to solve the implicit scheme are Picard iterations, as first applied to sea ice modelling by Zhang and Hibler [25], and the inexact Newton method, introduced in this context by Lemieux et al. [26] and subsequently improved by Lemieux et al. [27]. The first method has been shown to be slow [28], while the second suffers from non-convergence for fine meshes.

In light of these considerations, several methods have been proposed to improve convergence. For example, Mehlmann and Richter [29] proposed a modified Newton solver designed for solving the VP model directly. Their approach involves splitting the Jacobian matrix resulting from the Newton method into a positive definite and a negative semidefinite part. To enhance convergence, Mehlmann and Richter implemented a damping mechanism to address situations where the method fails to converge to a solution with a sufficiently small residual. In those cases, the negative semidefinite part of the Jacobian is damped. Notably, their method demonstrated successful convergence on a 4km grid and achieved convergence on a 2km grid for almost all simulated time steps in their test case.

Moreover, in another study by Mehlmann and Richter [30], a multigrid-framework is proposed to solve the resulting linear system. The use of a multigrid method as a preconditioner to the generalized minimal risidual method (GMRES) was found to reduce the number of GMRES iterations by 80 percent compared to the incomplete LU decomposition (ILU) preconditioner on fine meshes ranging from 16km to 2km in resolution. Additionally, Shih et al. [31] proposed an alternative linearisation of the nonlinear momentum equation in combination with an algebraic multigrid preconditioner, which was shown to substantially reduce computational costs. Even with these recent advancements, solving the VP model remains computationally expensive.

1.2. Scientific machine learning

In recent years, the development and usage of Artificial Intelligence (AI) has been on the rise, with its applications now extending far beyond traditional statistical inference tasks such as image classification and recommendation systems. A notable trend in AI is its increasing application in scientific domains, where machine learning algorithms are combined with scientific computing [32]. This emerging field, known as Scientific Machine Learning (SciML), stands apart from conventional machine learning (ML) by integrating knowledge about the problem at hand (domain knowledge) such as physical laws or known symmetries, or by improving numerical algorithms replacing part of it by a ML technique. The combination of the two approaches can result in increased accuracy, robustness or efficiency. Fur-

thermore, when domain knowledge is employed, SciML can have the advantage of being more data-efficient than traditional machine learning algorithms.

To help understand the different modelling options, we explore different ways in which SciML methods are classified. In the following, we will first detail a classification based on the role and integration of the Machine Learning method within or alongside the numerical method. Secondly, the traditional three machine learning paradigms are described, which are based on how methods infer patterns from data.

SciML classification

Heinlein et al. [33] make the distinction between three classes of scientific machine learning applied to domain decomposition methods (DDMs), which involve dividing the computational domain into subdomains solved in parallel and then synchronised at their interfaces. The different classes are: ML integrated into the numerical method, ML used as a solver and ML methods aided by knowledge from the DDM. We will generalise these three paradigms to encompass the broader scope of scientific machine learning, applying them to various numerical methods beyond DDMs. As is the case with DDMs, these classes do overlap, and certain approaches can be classified under multiple categories.

The first class involves employing ML techniques within a traditional numerical framework to enhance either the convergence properties or the computational efficiency. This typically results in hybrid models, where a machine learning model replaces some part of a numerical algorithm. For example, Ray and Hesthaven [5] use an artificial neural network to identify grid cells where the solution obtained by a numerical integration scheme needs correcting. The correction is computationally expensive and causes loss in accuracy if applied to the wrong grid cells. Therefore, the correction should be applied to no more grid cells than necessary. Here, the machine learning technique is implemented to replace the total variation bounded (TVB) indicator as a troubled cell indicator, enhancing the computational efficiency.

Another example is given by Margenberg et al. [6]. In this paper, a ML technique is employed to predict the solution of the Navier-Stokes equation on a fine grid given the simulation result on a coarse grid. The fine solution is then used to compute the right hand side of the equation which is restricted back to the coarse grid. In this way, accuracies approaching those on the fine grid are obtained using only computations on a coarse grid, increasing efficiency. Again, a hybrid model is obtained that uses a machine learning technique within a numerical framework.

In the case where ML techniques are used to act as solvers of differential equations, we see for example finite element methods or domain decomposition methods replaced by neural networks. An example of this category is the class of physics-informed neural networks (PINNs) first introduced by Lagaris et al. [34] in 1998 and later used in combination with data by Raissi et al. [35, 36] in 2017. Since their introduction, the use of PINNs has grown exponentially [4, p. 4].When using PINNs, a network is trained as the solution function, with coordinates as inputs and the function value as an output, using only the physical equations and the boundary conditions in training.

In [8], a neural network was employed to find the solution to the one-dimensional Schrödinger equation for both the infinite square well problem and the harmonic oscillator problem. One of the network's weights is used in the loss function as the eigenvalue. In this way, not only the eigenfunction, but also the corresponding eigenvalue is found. By using a 'driving' term in the loss function that penalises an eigenvalue that is too low, the optimisation process yields all eigenvalue-eigenfunction pairs if the bound is increased during optimisation. This driving term is an adaptation to the classical PINN approach. It allows finding all eigenfunction-eigenvalue pairs. In [37], E and Yu introduce the Deep Ritz method, which is similar to the PINN approach, but uses the variational formulation of the partial differential equation instead.

In their work, Tompson et al. [9] propose a hybrid approach for solving the Navier-Stokes equation. They combine the use of a neural network, following a PINN-like approach, to solve the pressure component, while employing conventional numerical methods for the advection step. This example illustrates an instance of overlap between the first two paradigms discussed, since the ML method is used in place of a traditional solver, but is also integrated within a larger traditional numerical framework.

In their study, Zang et al. [38] use adversarial neural networks to solve PDEs in their weak formulation. This approach involves the use of two neural networks: a primal network serves as the solution function, while an adversarial network acts as the test function in the weak formulation. The training

process follows an opposing strategy for the two networks: the primal network aims to minimise the norm of the weak form operator, while the adversarial network is trained to maximise it. Consequently, the primal network discovers a solution that satisfies the weak form for all test functions proposed by the adversarial network.

In the last class of SciML approaches, domain knowledge is often used to speed up the machine learning algorithm, but the machine learning method is typically still dominant. In the context of domain decomposition methods, examples of this can be found in distributed learning or collaborative learning [33].

Another example of this paradigm can be found in the study of Eichinger et al. [39]. Here, a convolutional neural network with a bottleneck structure is employed as a surrogate model for fluid flow around objects in a channel. The bottleneck structure of the network results in an encoding to reduced order, which is similar to reduced order models [40], which are known for their reduction in computational costs and quicker evaluation. This neural network design is inspired by traditional numerical methods, illustrating the effective use of domain knowledge.

Machine learning paradigms

The various approaches discussed above do not only differ in terms of the role machine learning plays in relation to numerical methods or domain knowledge but can also be distinguished based on their data usage. In machine learning, three paradigms are commonly recognised: supervised, unsupervised and reinforcement learning. In supervised learning, algorithms learn patterns from labelled data, enabling tasks such as regression or classification. On the other hand, unsupervised learning involves algorithms discovering patterns in unlabelled data (e.g. clustering). While also not relying on labelled data, reinforcement learning is different from unsupervised learning since it learns what to do (mapping a state to an action) to maximise a reward signal.

The approaches of Ray and Hesthaven [5], Margenberg et al. [6], and Eichinger et al. [39] are typical examples of supervised learning: in each example, the algorithm learns a pattern from labelled data. In [5], for example, grid cells are labelled using the TVB indicator, while the labels in [6] consist of fine grid solutions and [39] use simulation results. These last two examples constitute regression task, while the former is a classification task.

On the other hand, the methods of Raissi et al. [35, 36], Jin et al. [8], E and Yu [37], Tompson et al. [9], and Zang et al. [38] only rely on unlabelled data and a cleverly chosen cost function and are thus considered unsupervised. As noted by Cuomo et al. [4], PINNs that only use differential equations and boundary conditions should be considered unsupervised, but when applied to inverse problems or if a limited amount of experimental data is used, they should be considered supervised or semi-supervised learning methods, respectively. In contrast, reinforcement learning is not as pronounced in SciML.

The choice between a supervised and an unsupervised method has serious implications for the data requirements of the resulting algorithm. For instance, the surrogate model of Eichinger et al. [39] requires data that consist of both the geometry of the object and simulation results. If those simulations are computationally expensive, as is often the case when a surrogate model is needed, the generation of such data becomes relatively costly. Consequently, the computational effort shifts from the online phase to the offline phase: in [39], the training process reportedly spans from several hours to days, while the evaluation time is significantly reduced to $\mathcal{O}(0.1)$ s from $\mathcal{O}(10)$ s.

In contrast, unsupervised learning approaches do not require labelled data. Since physics-informed neural networks, for instance, have no clear separation of the offline and online phase, the computational burden occurs at the same time as for traditional methods. However, this does not apply universally to all unsupervised learning or even PINN methods. In the case of Tompson et al. [9], the model is trained using an unsupervised and offline approach, enabling efficient online evaluation. In their study, a neural network is employed to predict fluid flow around various geometries using a PINN-like loss function during training.

Furthermore, the data usage of offline-heavy methods does not always pose significant challenges. For example, Margenberg et al. [6] employ a relatively small neural network with only 8,634 trainable parameters due to their patch-wise application of the network. In contrast, Eichinger et al. [39] use networks with parameter counts ranging from 34 to 87 million due to a global approach. A smaller network reduces the time required for training since the dataset size should depend on the number of trainable parameters [41] and thus fewer computations are necessary for optimisation. Additionally, the

method of Ray and Hesthaven [5] demonstrates a less data-intensive approach by using the network to replace the TVB indicator. The approach of Margenberg et al. is considered a local approach, as the network is only applied to a small region of the domain. Conversely, the approach by Eichinger et al. is a global approach, since their surrogate model predicts a solution on the entire domain.

SciML applied to sea ice modelling

While the prediction of sea ice thickness, concentration and/or momentum using neural networks is not new, most studies involve a conventional machine learning approach. In their study, Belchansky et al. [42] perform a regression analysis to predict the sea ice thickness using four shortwave and long-wave radiative fluxes, surface air temperature, ice drift velocity, and ice divergenve. Similarly, Herbert et al. [43] predict the sea ice thickness using altimetry data and brightness temperatures. In another study, Chi and Kim [44] employ a neural network as a forecasting model for sea ice concentration based on historic sea ice concentration data. Furthermore, Kim et al. [45] propose a one month sea ice concentration forecasting model using a convolutional neural network based on the historic sea ice concentration, sea surface temperature, 2m air temperature, forecast albedo, and wind. While all of these examples can be considered surrogate models, none of these supervised learning approaches uses domain knowledge and are therefore not examples of scientific machine learning.

In the field of sea ice modelling, while traditional machine learning approaches have been applied occasionally, there is a very limited body of research that has explored the application of scientific machine learning. One notable example is the work conducted by Finn et al. [46], where a hybrid model is proposed that employs a neural network to address subgrid-scale dynamics. Their approach bears similarities to the methodology Margenberg et al. [6]. Initially, the study involves conducting simulations at both 4km and 8km resolutions. Subsequently, the solution on the low-resolution grid is interpolated to the high-resolution grid, after which a convolutional neural network is applied to predict features on this grid. These features are then projected back onto the 4km grid and a second part of the network predicts the difference between the low and high-resolution solutions. The model is trained on these differences, and the predicted error is added to the low-resolution solution as a correction. In this approach, a neural network is used to improve classical methods, making it an example of the first SciML paradigm.

In another recent work by Finn et al. [47], a method is developed for the generation of sea ice velocity, height and concentration data. In their approach, a latent diffusion model is employed alongside a variational autoencoder, where the latter is used to learn a mapping from data space to an encoding in latent space and vice versa. The diffusion model then learns to randomly generate new latent encodings from noise. In this way, new samples of sea ice states can be created randomly by first generating a new encoding in latent space and transforming that back to data space. Domain knowledge is used in the training process, making it an example of the third SciML paradigm.

While the aforementioned studies show significant advancements in predicting sea ice dynamics using machine learning, it remains an open question how domain knowledge, defined above, can be integrated into machine learning techniques for sea ice dynamics modelling. Addressing this question is essential for developing alternative methods that accurately capture the complexities of sea ice dynamics.

1.3. Research aim

Having established that the integration of domain-specific knowledge into machine learning models for sea ice dynamics remains underexplored, it is imperative to address this gap to meet the demand for high-resolution predictions, for example for use in Arctic maritime navigation. Traditional continuum models, grounded in viscous-plastic rheology, are computationally intensive and often operate at scales too large to capture linear kinematic features. These features require models with resolutions finer than 5 kilometres, but achieving such detail with conventional methods is impractical for timely predictions. Therefore, the main research aim of this work is to develop a deep learning-based surrogate model that can predict linear kinematic features.

Considering the previously discussed SciML and ML paradigms, developing a surrogate model emerges as a compelling approach for enhancing sea ice dynamics modelling with SciML techniques. Surrogate models are typical examples of the third SciML paradigm, where machine learning techniques are central and domain knowledge is used to enhance the learning phase. This integration is

advantageous because it improves data-efficiency during learning while retaining the benefits of rapid evaluation times typical for classical machine learning techniques.

Additionally, surrogate models can be trained in a supervised manner when labelled data is abundant (e.g. Eichinger et al. [39]), unsupervised with only a physics-informed loss (e.g. Tompson et al. [9]) or even semi-supervised combining both approaches. In this work, a supervised learning approach is adopted for training the surrogate model. By focusing on a supervised, data-driven method, we aim to delve into the intricacies of modelling sea ice behaviour with neural networks, particularly at the high resolutions for linear kinematic feature formation. This approach allows us to lay a solid foundation for future work that may explore more complex techniques, such as unsupervised or semi-supervised learning methods.

To address the demand for alternative methods for high-resolution predictions, we develop a deep learning-based surrogate model, aided by the following supporting research questions:

1. Which neural network architecture is most effective for predicting sea ice dynamics at high resolutions?
2. How do different loss functions, particularly those incorporating domain knowledge, impact the performance of the model?
3. How does error propagate and accumulate over time?
4. To what extent can the model accurately predict linear kinematic features?
5. Is the choice of training data critical for the surrogate model's performance?

1.4. Thesis outline

This thesis is structured into the following chapters. Chapter 2 introduces the sea ice dynamics model based on the viscous-plastic rheology that forms the foundation of this study. Following this, Chapter 3 details the numerical framework used to benchmark the developed deep learning-based surrogate model against traditional methods. Chapter 4 delves into the relevant machine learning techniques employed, while Chapter 5 discusses the training data and test cases. The design and architecture of the surrogate model are covered in Chapter 6. Subsequently, Chapter 7 presents the results, including a comparison of different neural network architectures and loss functions, an analysis of the network's ability to predict linear kinematic features, and an examination of error accumulation in time series predictions. Finally, Chapter 8 offers conclusions, discussions, and recommendations for future research based on the findings of this study.

2

Sea ice model

A mathematical description of the sea ice model including the viscous-plastic (VP) rheology is given in this chapter. The governing equation are presented first, followed by the rheology and finally a dimensionless version of the model is given. The description of the sea ice model follows Mehlmann [48], which is also used in [29, 30, 49].

2.1. Model description

The ice velocity $\boldsymbol{v}$, the ice concentration A and the ice thickness H are the three modelled quantities. The viscous-plastic sea ice model consists of three equations: the ice momentum equation for the velocity and two balance equations for the ice height and ice concentration. The system of equations is as follows:

$$\rho_{\text{ice}} H \left(\partial_t \boldsymbol{v} + f_c \boldsymbol{e}_z \times \boldsymbol{v}\right) = \nabla \cdot \boldsymbol{\sigma}(\boldsymbol{v}) + \boldsymbol{\tau}(\boldsymbol{v}) - \rho_{\text{ice}} H g \nabla \tilde{H}_g, \tag{2.1}$$

$$\partial_t H + \nabla \cdot (H\boldsymbol{v}) = 0, \quad 0 \leq H, \tag{2.2}$$

$$\partial_t A + \nabla \cdot (A\boldsymbol{v}) = 0, \quad 0 \leq A \leq 1, \tag{2.3}$$

with a no-slip boundary condition on the ice velocity: $\boldsymbol{v} = 0$ on $\partial\Omega$. Here, ρ_{ice} is the ice density, $\boldsymbol{\sigma}$ is the stress tensor, f_c is the Coriolis parameter, $\boldsymbol{e}_z$ is the unit vector in the z-direction, $\boldsymbol{\tau}$ is the surface stress, g is the gravitational acceleration, and $\tilde{H}_g$ is the sea surface height. The forcing is caused by the surface stress $\boldsymbol{\tau}$, which is the sum of ocean and atmospheric surface stresses, due to ocean current and wind. The total surface stress is given by

$$\boldsymbol{\tau}(\boldsymbol{v}) = C_{\text{ocean}} \rho_{\text{ocean}} \left\|\boldsymbol{v}_{\text{ocean}} - \boldsymbol{v}\right\|_2 \left(\boldsymbol{v}_{\text{ocean}} - \boldsymbol{v}\right) + C_{\text{atm}} \rho_{\text{atm}} \|\boldsymbol{v}_{\text{atm}}\|_2 \boldsymbol{v}_{\text{atm}}, \tag{2.4}$$

where C_{ocean} is the water drag coefficient, ρ_{ocean} is the water density, $\boldsymbol{v}_{\text{ocean}}$ is the ocean current velocity, C_{atm} is the air drag coefficient, ρ_{atm} is the air density and $\boldsymbol{v}_{\text{atm}}$ is wind velocity. The values of the physical constants are listed in Table 2.1.

The last term in (2.1) models the force due to a changing sea surface tilt, with g the gravitational acceleration and H_s the sea surface height. Following Tremblay and Mysak [50, p. 2352], we approximate this term by

$$g \nabla \tilde{H}_g \approx -f_c \boldsymbol{e}_z \times \boldsymbol{v}_{\text{ocean}}.$$

The internal stresses in the ice are represented by $\boldsymbol{\sigma}$. The stresses are modelled following the viscous-plastic ice rheology as described in the following section.

2.2. Rheology

The internal stress tensor is related to the strain rate tensor through the nonlinear viscous-plastic rheology. The strain rate tensor $\dot{\boldsymbol{\epsilon}}$ and the trace free strain rate tensor $\dot{\boldsymbol{\epsilon}}'$ are given by

$$\dot{\boldsymbol{\epsilon}} = \frac{1}{2}\left(\nabla \boldsymbol{v} + \nabla \boldsymbol{v}^T\right), \tag{2.5}$$

$$\dot{\boldsymbol{\epsilon}}' = \dot{\boldsymbol{\epsilon}} - \frac{1}{2}\text{tr}(\dot{\boldsymbol{\epsilon}}) I. \tag{2.6}$$

Table 2.1: List of physical constants in the sea ice dynamics system of equations taken from [48].

Parameter	Definition	Value
ρ_{ice}	sea ice density	900 kg/m^3
ρ_{atm}	air density	1.3 kg/m^3
ρ_{ocean}	sea water density	1026 kg/m^3
C_{atm}	air drag coefficient	$1.2 \cdot 10^{-3}$
C_{ocean}	water drag coefficient	$5.5 \cdot 10^{-3}$
f_c	Coriolis parameter	$1.46 \cdot 10^{-4}$ s^{-1}
P^*	ice strength parameter	$27.5 \cdot 10^3$ N/m^2
C	ice concentration parameter	20
e	ellipse ratio	2
Δ_{min}		$2 \cdot 10^{-9}$ s^{-1}

The stress tensor is then as follows:

$$\boldsymbol{\sigma} = 2\eta\dot{\boldsymbol{\epsilon}}' + \zeta \text{tr}(\dot{\boldsymbol{\epsilon}})I - \frac{P}{2}I. \tag{2.7}$$

Here, η and ζ are the viscosities and are given by

$$\eta = e^{-2}\zeta, \tag{2.8}$$

$$\zeta = \frac{P}{2\Delta(\dot{\boldsymbol{\epsilon}})}, \tag{2.9}$$

where the compressive ice strength P and the parameter Δ are given by

$$P(H, A) = P^* H \exp(-C(1 - A)), \tag{2.10}$$

$$\Delta(\dot{\boldsymbol{\epsilon}}) = \sqrt{2e^{-2}\dot{\boldsymbol{\epsilon}}' : \dot{\boldsymbol{\epsilon}}' + \text{tr}(\dot{\boldsymbol{\epsilon}})^2 + \Delta_{\text{min}}^2}. \tag{2.11}$$

The constants P^* and C determine the ice's strength. Furthermore, for the purpose of numerical stability and to introduce viscosity in the limiting case where $\dot{\boldsymbol{\epsilon}} \to \boldsymbol{0}$, the small parameter Δ_{min} is introduced. In this way, ice behaves like a viscous fluid at very low rates of deformation and as a plastic material at high rates.

A useful quantity to investigate the formation of linear kinematic features is the shear deformation $\dot{\boldsymbol{\epsilon}}_{II} \in \mathbb{R}_+$. This strain invariant describes the relative motion between different parts of the sea ice, which is critical for understanding the forces that lead to the formation of LKFs. Therefore, by studying shear deformation, we can identify and localise these linear kinematic features. The shear deformation is given by

$$\dot{\boldsymbol{\epsilon}}_{II} = 2\sqrt{-\det(\dot{\boldsymbol{\epsilon}})} = \sqrt{\left(\dot{\boldsymbol{\epsilon}}_{x,x} - \dot{\boldsymbol{\epsilon}}_{y,y}\right)^2 + 4\dot{\boldsymbol{\epsilon}}_{x,y}^2}. \tag{2.12}$$

2.3. Dimensionless model

To aid the simulation and in line with the analysis by [48], we derive a dimensionless form for the equations. We do this by rescaling the time t, position $\boldsymbol{x}$ and ice thickness H by characteristic time T, characteristic length L and typical ice thickness G:

$$\bar{t} = \frac{t}{T}, \quad \bar{\boldsymbol{x}} = \frac{\boldsymbol{x}}{L}, \quad \bar{H} = \frac{H}{G}.$$

Then, the dimensionless velocity and strain rates are

$$\boldsymbol{v} = \frac{L}{T}\bar{\boldsymbol{v}}, \quad \dot{\boldsymbol{\epsilon}} = T^{-1}\bar{\dot{\boldsymbol{\epsilon}}}, \quad \dot{\boldsymbol{\epsilon}}' = T^{-1}\bar{\dot{\boldsymbol{\epsilon}}}'.$$

If we rescale the nonlinearity $\Delta_{\text{min}} = T^{-1}\bar{\Delta}_{\text{min}}$, we get

$$\Delta(\dot{\boldsymbol{\epsilon}}) = T^{-1}\sqrt{2e^{-2}\bar{\dot{\boldsymbol{\epsilon}}}' : \bar{\dot{\boldsymbol{\epsilon}}} + \text{tr}(\bar{\dot{\boldsymbol{\epsilon}}})^2 + \bar{\Delta}_{\text{min}}^2} = T^{-1}\bar{\Delta}(\bar{\dot{\boldsymbol{\epsilon}}}).$$

Furthermore, the ice strength becomes

$$\bar{P}(\bar{H}, A) = P^* G \bar{H} \exp(-C(1-A))$$

and so $P = G\bar{P}$. The dimensionless viscosities become $\zeta = GT\bar{\zeta}$ and $\eta = GT\bar{\eta}$. The internal stress tensor $\boldsymbol{\sigma}$ is then rescaled to

$$\boldsymbol{\sigma} = \frac{P^* G}{2} \bar{\boldsymbol{\sigma}},$$

with

$$\bar{\boldsymbol{\sigma}} = \frac{1}{\bar{\Delta}(\bar{\dot{\boldsymbol{\epsilon}}})} \bar{H} \exp(-C(1-A)) \left(\mathrm{tr}(\bar{\dot{\boldsymbol{\epsilon}}}) I + 2e^{-2} \bar{\dot{\boldsymbol{\epsilon}}}' - \bar{\Delta}(\bar{\dot{\boldsymbol{\epsilon}}}) I \right)$$

The total surface stress is transformed to

$$\begin{aligned}
\boldsymbol{\tau}(\boldsymbol{v}) &= C_{\text{ocean}} \rho_{\text{ocean}} \left\| \boldsymbol{v}_{\text{ocean}} - \boldsymbol{v} \right\|_2 (\boldsymbol{v}_{\text{ocean}} - \boldsymbol{v}) + C_{\text{atm}} \rho_{\text{atm}} \| \boldsymbol{v}_{\text{atm}} \|_2 \boldsymbol{v}_{\text{atm}} \\
&= \frac{C_{\text{ocean}} \rho_{\text{ocean}} L^2}{T^2} \left\| \bar{\boldsymbol{v}}_{\text{ocean}} - \bar{\boldsymbol{v}} \right\|_2 (\bar{\boldsymbol{v}}_{\text{ocean}} - \bar{\boldsymbol{v}}) + \frac{C_{\text{atm}} \rho_{\text{atm}} L^2}{T^2} \| \bar{\boldsymbol{v}}_{\text{atm}} \|_2 \bar{\boldsymbol{v}}_{\text{atm}} \\
&= \frac{C_{\text{ocean}} \rho_{\text{ocean}} L^2}{T^2} \bar{\boldsymbol{\tau}}_{\text{ocean}}(\bar{\boldsymbol{v}}) + \frac{C_{\text{atm}} \rho_{\text{atm}} L^2}{T^2} \bar{\boldsymbol{\tau}}_{\text{atm}}
\end{aligned}$$

where $\boldsymbol{v}_{\text{ocean}} = \frac{L}{T} \bar{\boldsymbol{v}}_{\text{ocean}}$ and $\boldsymbol{v}_{\text{atm}} = \frac{L}{T} \bar{\boldsymbol{v}}_{\text{atm}}$.

Using the dimensionless quantities and denoting the $\bar{\nabla}$ as the divergence operator with respect to coordinates $\boldsymbol{x} = (x, y)$, we can rewrite the momentum equation as follows:

$$\frac{\rho_{\text{ice}} GL}{T} \bar{H} \partial_t \bar{\boldsymbol{v}} + \frac{\rho_{\text{ice}} GL f_c}{T} \boldsymbol{e}_z \times (\bar{\boldsymbol{v}} - \bar{\boldsymbol{v}}_{\text{ocean}}) = \frac{P^* G}{2} \nabla \cdot \bar{\boldsymbol{\sigma}} + \frac{C_{\text{ocean}} \rho_{\text{ocean}} L^2}{T^2} \bar{\boldsymbol{\tau}}_{\text{ocean}}(\bar{\boldsymbol{v}}) + \frac{C_{\text{atm}} \rho_{\text{atm}} L^2}{T^2} \bar{\boldsymbol{\tau}}_{\text{atm}},$$

becomes

$$\frac{\rho_{\text{ice}} GL}{T^2} \bar{H} \partial_{\bar{t}} \bar{\boldsymbol{v}} + \frac{\rho_{\text{ice}} GL f_c}{T} \boldsymbol{e}_z \times (\bar{\boldsymbol{v}} - \bar{\boldsymbol{v}}_{\text{ocean}}) = \frac{P^* G}{2L} \bar{\nabla} \cdot \bar{\boldsymbol{\sigma}} + \frac{C_{\text{ocean}} \rho_{\text{ocean}} L^2}{T^2} \bar{\boldsymbol{\tau}}_{\text{ocean}}(\bar{\boldsymbol{v}}) + \frac{C_{\text{atm}} \rho_{\text{atm}} L^2}{T^2} \bar{\boldsymbol{\tau}}_{\text{atm}},$$

which can be simplified to

$$\bar{H} \partial_{\bar{t}} \bar{\boldsymbol{v}} + T f_c \bar{H} \boldsymbol{e}_z \times (\bar{\boldsymbol{v}} - \bar{\boldsymbol{v}}_{\text{ocean}}) = C_r \bar{\nabla} \cdot \bar{\boldsymbol{\sigma}} + C_{\text{ocean}} \bar{\boldsymbol{\tau}}_{\text{ocean}}(\bar{\boldsymbol{v}}) + C_{\text{atm}} \bar{\boldsymbol{\tau}}_{\text{atm}},$$

with the dimensionless parameters

$$C_r = \frac{P^* T^2}{2 \rho_{\text{ice}} L^2}, \quad C_{\text{ocean}} = \frac{C_{\text{ocean}} \rho_{\text{ocean}} L}{\rho_{\text{ice}} G}, \quad C_{\text{atm}} = \frac{C_{\text{atm}} \rho_{\text{atm}} L}{\rho_{\text{ice}} G}.$$

Next, we transform the balance equations. The equation for the ice thickness becomes:

$$\begin{aligned}
0 &= G \partial_t \bar{H} + \frac{GL}{T} \nabla \cdot (\bar{H} \bar{\boldsymbol{v}}) \\
&= \frac{G}{T} \left(\partial_{\bar{t}} \bar{H} + \bar{\nabla} \cdot (\bar{H} \bar{\boldsymbol{v}}) \right) \\
&= \partial_{\bar{t}} \bar{H} + \bar{\nabla} \cdot (\bar{H} \bar{\boldsymbol{v}})
\end{aligned}$$

The conservation law for the ice concentration in dimensionless form is then:

$$\begin{aligned}
&= \partial_t A + \frac{L}{T} \nabla \cdot (A \bar{\boldsymbol{v}}) \\
&= \frac{1}{T} \left(\partial_{\bar{t}} A + \bar{\nabla} \cdot (A \bar{\boldsymbol{v}}) \right) \\
&= \partial_{\bar{t}} A + \bar{\nabla} \cdot (A \bar{\boldsymbol{v}})
\end{aligned}$$

In the following, we drop the bar and assume that all quantities are dimensionless to ease notation. The complete dimensionless viscous-plastic sea ice model is summarised in the frame below.

Complete dimensionless viscous-plastic sea ice model

On the interior of domain Ω, the sea ice velocity $\boldsymbol{v}$, height H and concentration A are governed by

$$H\partial_t \boldsymbol{v} + T f_c H \boldsymbol{e}_z \times (\boldsymbol{v} - \boldsymbol{v}_{\text{ocean}}) = C_r \nabla \cdot \boldsymbol{\sigma} + C_{\text{ocean}} \boldsymbol{\tau}_{\text{ocean}}(\boldsymbol{v}) + C_{\text{atm}} \boldsymbol{\tau}_{\text{atm}}, \tag{2.13}$$

$$\partial_t H + \nabla \cdot (H\boldsymbol{v}) = 0, \quad 0 \leq H \tag{2.14}$$

$$\partial_t A + \nabla \cdot (A\boldsymbol{v}) = 0, \quad 0 \leq A \leq 1, \tag{2.15}$$

while we have

$$\boldsymbol{v} = \boldsymbol{0}$$

on the boundary $\partial\Omega$. The scaling parameters are chosen the same as in [48]:

$$T = 10^3 \text{ s}, \quad L = 10^6 \text{ m}, \quad H = 1 \text{ m}.$$

By using the values listed in Table 2.1, the dimensionless parameters are given by

$$C_r \approx 1.52778 \cdot 10^{-5}, \quad C_{\text{ocean}} = 6270, \quad C_{\text{atm}} \approx 1.73333,$$

the surface stresses are given by

$$\boldsymbol{\tau}_{\text{ocean}}(\boldsymbol{v}) = \|\boldsymbol{v}_{\text{ocean}} - \boldsymbol{v}\|_2 (\boldsymbol{v}_{\text{ocean}} - \boldsymbol{v}), \quad \boldsymbol{\tau}_{\text{atm}} = \|\boldsymbol{v}_{\text{atm}}\|_2 \boldsymbol{v}_{\text{atm}}$$

and the internal stress is given by

$$\boldsymbol{\sigma} = \frac{1}{\Delta(\dot{\boldsymbol{\epsilon}})} H \exp(-C(1-A)) \left(\text{tr}(\dot{\boldsymbol{\epsilon}})I + 2e^{-2}\dot{\boldsymbol{\epsilon}}' - \Delta(\dot{\boldsymbol{\epsilon}})I\right), \tag{2.16}$$

$$\Delta(\dot{\boldsymbol{\epsilon}}) = \sqrt{2e^{-2}\dot{\boldsymbol{\epsilon}}' : \dot{\boldsymbol{\epsilon}}' + \text{tr}(\dot{\boldsymbol{\epsilon}})^2 + \Delta_{\text{min}}^2}, \quad \Delta_{\text{min}} = 2 \cdot 10^{-6}. \tag{2.17}$$

3

Numerical methods

In this chapter, the numerical framework from [48], or more specifically [49], is detailed. The numerical methods described in this chapter will form the benchmark to compare our deep learning approach against. The sea ice model is implemented using the finite element library Gascoigne 3D [51].

The system of equations of the sea ice dynamics model as presented in the previous section is solved using a partitioned approach. First, the balance equations for sea ice thickness H and concentration A are solved to find their new values H^n and A^n using the ice velocity $\boldsymbol{v}^{n-1}$ from the previous time step. Next, the momentum equation is solved for $\boldsymbol{v}^n$ using these new values of H^n and A^n. For the balance equations, an upwind finite volume method is used. On the other hand, the momentum equation is solved using a finite element method. An overview of the steps is given in Algorithm 1. After describing the spatial discretisation of choice in Section 3.1, the finite volume method and the time discretisation are detailed in Section 3.2 and Section 3.3, respectively. Next, the finite element method is broken up by first describing the weak formulation in Section 3.4, followed by the Newton method used to solve the resulting nonlinear equation in Section 3.5 and finally the method to solve the system of equations that follows in Section 3.6.

Algorithm 1 Partitioned solution approach

Initial conditions H^0, A^0 and $\boldsymbol{v}^0$.
for $n = 1, \ldots, N$ **do**
1. Solve balance equations using finite volumes

$$H^{n-1}, A^{n-1}, \boldsymbol{v}^{n-1} \mapsto H^n, A^n$$

2. Solve momentum equation using finite elements

$$H^n, A^n, \boldsymbol{v}^{n-1} \mapsto \boldsymbol{v}^n$$

3.1. Spatial discretisation

In their work, Mehlmann et al. [49] present the effect of different spatial discretisations on the formation of linear kinematic features. Even though the CD discretisation yields more linear kinematic features, the Arakawa B-grid is better suited for use in a machine learning context since the CD-grid would require interpolation to be transformed to a tensor format. Moreover, the B-grid performed better in terms of the number of linear kinematic features compared to the third discretisation, the A-grid.

On the Arakawa B-grid, the sea ice velocity nodes are placed of the vertices of the grid cells, and the sea ice thickness and concentration nodes are placed in the midpoints of the cells. The grid is visualised in Figure 3.1. In each grid cell, the quantities are approximated using a linear combination of basis functions. The ice height H and concentration A in a grid cell ω in the domain Ω are approximated

with piecewise constant basis functions:

$$\varphi_\omega(\boldsymbol{x}) = \begin{cases} 1, & \text{if } \boldsymbol{x} \in \omega \\ 0, & \text{otherwise} \end{cases}$$
$$H(\boldsymbol{x}) = \sum_{\omega \in \Omega} H_\omega \varphi_\omega(\boldsymbol{x}),$$
$$A(\boldsymbol{x}) = \sum_{\omega \in \Omega} A_\omega \varphi_\omega(\boldsymbol{x}).$$

The sea ice velocity $\boldsymbol{v}$ is approximated using bilinear quadrilateral basis functions φ_i that are 1 on vertex i and 0 on all other vertices:

$$\boldsymbol{v}(\boldsymbol{x}) = \sum_{i \in V} \begin{pmatrix} v_{x,i} \\ v_{y,i} \end{pmatrix} \varphi_i(\boldsymbol{x}), \tag{3.1}$$

where V is the set of all vertices and $v_{x,i}$ and $v_{y,i}$ are the velocities in the x- and y-directions respectively at vertex i. Following the numbering of vertices in Figure 3.1, there are four nonzero basis functions:

$$\varphi_1(x,y) = \frac{1}{h^2}(1-x)(1-y),$$
$$\varphi_2(x,y) = \frac{1}{h^2}x(1-y),$$
$$\varphi_3(x,y) = \frac{1}{h^2}(1-x)y,$$
$$\varphi_4(x,y) = \frac{1}{h^2}xy,$$

where $\boldsymbol{x} = (x, y)$ is scaled such that $[0, h]^2$ corresponds to the grid cell and h is the grid spacing on a constant square grid.

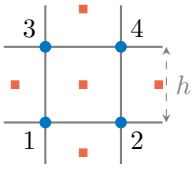


Figure 3.1: A representation of the Arakawa B-grid. The numbered blue circles are the grid nodes for the sea ice velocity $\boldsymbol{v}$. The orange squares are the nodes for the sea ice thickness H and concentration A. The grid spacing h is shown.

3.2. Finite volume method

Having discussed the spatial discretisation, we now describe the discretisation of the balance equations. Following the approach of Mehlmann et al. [49, p. 18], the balance equations are solved using an upwind finite volume method [52]. Here, the change in H and A in each grid cell, which is computed by the total flux over the cell's edges, is approximated using the discretised velocities, ice height and ice concentration.

Since we use elements with velocity nodes on the grid vertices, the flux over the cell edges is computed using the average between the velocities on the vertices of the edge:

$$\boldsymbol{v}_e^{n-1} = \frac{\boldsymbol{v}_i^{n-1} + \boldsymbol{v}_j^{n-1}}{2}. \tag{3.2}$$

We then compute the flux over the cell edge $e = (i, j)$ and update the quantity in the cell accordingly:

$$H^n = H^{n-1} + \begin{cases} H^{n-1}\boldsymbol{v}_e^{n-1} \cdot \hat{\boldsymbol{n}} & \text{, if } \boldsymbol{v}_e^{n-1} \cdot \hat{\boldsymbol{n}} \geq 0 \\ \hat{H}^{n-1}\boldsymbol{v}_e^{n-1} \cdot \hat{\boldsymbol{n}} & \text{, if } \boldsymbol{v}_e^{n-1} \cdot \hat{\boldsymbol{n}} < 0 \end{cases} \tag{3.3}$$

Here, H is the sea ice thickness in the cell that is updated, $\hat{H}$ is the thickness of the cell on the other side of the edge we consider, $\boldsymbol{v}_i$ is the ice velocity at vertex i and $\hat{\boldsymbol{n}}$ is the normal vector pointing away

from the cell perpendicular to the edge. Therefore, if $\boldsymbol{v}_e \cdot \hat{\boldsymbol{n}} \geq 0$, the flow is outward, in which case we use the ice thickness H from the cell we consider. Conversely, we use the ice thickness $\hat{H}$ from the adjacent cell if the flow is inward. The flux over each edge is therefore computed using the 'upwind' value. The ice concentration A is updated analogously.

3.3. Time discretisation for the sea ice problem

The discrete derivative with respect to time is

$$\partial_t \boldsymbol{v} \approx \frac{\boldsymbol{v}^n - \boldsymbol{v}^{n-1}}{k},$$

where $\boldsymbol{v}^n$ indicates the solution at time $t = t_n$ and k is the step size. For the time discretisation, the implicit Euler method is chosen. In this approach, the unknown value $\boldsymbol{v}^n$ is used in the momentum equation (2.13) where the time derivative has been replaced by the discrete derivative:

$$H^n \frac{\boldsymbol{v}^n - \boldsymbol{v}^{n-1}}{k} + T f_c H^n \boldsymbol{e}_z \times (\boldsymbol{v}^n - \boldsymbol{v}_{\text{ocean}}) = C_r \nabla \cdot \boldsymbol{\sigma}\left(\boldsymbol{v}^n, H^n, A^n\right) + C_{\text{ocean}} \boldsymbol{\tau}_{\text{ocean}}(\boldsymbol{v}^n) + C_{\text{atm}} \boldsymbol{\tau}_{\text{atm}}.$$

Due to the nonlinearity in the stress term with respect to the unknown $\boldsymbol{v}^n$, we need a nonlinear solver to find $\boldsymbol{v}^n$. In Section 3.5, the Newton method is detailed.

3.4. Weak formulation of the momentum equation

In this section, we derive the weak formulation of the momentum equation. This is done by first multiplying the equation by a test function $\boldsymbol{\varphi}$ in $H_0^1(\Omega)$, which is L^2-integrable on Ω, has first weak derivatives and is zero on the boundary $\partial\Omega$, followed by integrating over the domain Ω. For ease of notation, this operation is written as the inner product:

$$(\boldsymbol{u}, \boldsymbol{\varphi}) = \int_\Omega u \cdot \boldsymbol{\varphi} \, d\Omega. \tag{3.4}$$

The equation is then written as

$$A(\boldsymbol{v}^n, \boldsymbol{\varphi}) = F(\boldsymbol{\varphi}),$$

where

$$A(\boldsymbol{v}^n, \boldsymbol{\varphi}) = \left(\frac{H^n}{k}\boldsymbol{v}^n, \boldsymbol{\varphi}\right) + (T f_c H^n \boldsymbol{e}_z \times (\boldsymbol{v}^n - \boldsymbol{v}_{\text{ocean}}), \boldsymbol{\varphi}) - (C_r \nabla \cdot \boldsymbol{\sigma}^n, \boldsymbol{\varphi}) - (C_{\text{ocean}} \boldsymbol{\tau}_{\text{ocean}}(\boldsymbol{v}^n), \boldsymbol{\varphi})$$

and

$$F(\boldsymbol{\varphi}) = \left(\frac{H^n}{k}\boldsymbol{v}^{n-1}, \boldsymbol{\varphi}\right) + (C_{\text{atm}} \boldsymbol{\tau}_{\text{atm}}, \boldsymbol{\varphi}). \tag{3.5}$$

The stress term $\boldsymbol{\sigma}\left(\boldsymbol{v}^n, H^n, A^n\right)$ was written as $\boldsymbol{\sigma}$ to ease notation. We now simplify A by integrating by parts and applying Gauss' divergence theorem:

$$\begin{aligned}
(C_r \nabla \cdot \boldsymbol{\sigma}^n, \boldsymbol{\varphi}) &= \int_\Omega C_r (\nabla \cdot \boldsymbol{\sigma}^n) \cdot \boldsymbol{\varphi} \, d\Omega \\
&= \int_\Omega C_r \nabla \cdot (\boldsymbol{\sigma}^n \boldsymbol{\varphi}) - C_r \boldsymbol{\sigma}^n : \nabla \varphi_1 \, d\Omega, && \text{by partial integration} \\
&= -\int_\Omega C_r \boldsymbol{\sigma}^n : \nabla \boldsymbol{\varphi} \, d\Omega + \int_{\partial\Omega} C_r (\boldsymbol{\sigma}^n \boldsymbol{\varphi}) \cdot \hat{\boldsymbol{n}} \, d\Gamma, && \text{by Gauss' divergence theorem} \\
&= -\int_\Omega C_r \boldsymbol{\sigma}^n : \nabla \boldsymbol{\varphi} \, d\Omega, && \text{since } \boldsymbol{\varphi} = 0 \text{ on } \partial\Omega \\
&= -(C_r \boldsymbol{\sigma}^n, \nabla \boldsymbol{\varphi}),
\end{aligned}$$

and obtain

$$A(\boldsymbol{v}^n, \boldsymbol{\varphi}) = \left(\frac{H^n}{k}\boldsymbol{v}^n, \boldsymbol{\varphi}\right) + (T f_c H^n \boldsymbol{e}_z \times (\boldsymbol{v}^n - \boldsymbol{v}_{\text{ocean}}), \boldsymbol{\varphi}) + (C_r \boldsymbol{\sigma}^n, \nabla \boldsymbol{\varphi}) - (C_{\text{ocean}} \boldsymbol{\tau}_{\text{ocean}}(\boldsymbol{v}^n), \boldsymbol{\varphi})$$

As described in Section 3.1, $\boldsymbol{v}^n$ is approximated using a linear combination of bilinear quadrilateral basis functions φ_i, which is equal to 1 on grid vertex i, but 0 on all other vertices. Each basis function is then multiplied with a vector of coefficients $\boldsymbol{v}_i^n \in \mathbb{R}^2$. We now see that there are $2m$ unknowns, with m the number of nodes in the grid. Therefore, we choose the test functions $\hat{x}\varphi_i$ and $\hat{y}\varphi_i$, where $\hat{x}$ and $\hat{y}$ are the unit vectors. This gives a system of $2m$ equations which we need to solve to find the coefficients $v_{x,i}^n$ and $v_{y,i}^n$ in (3.1):

$$\begin{aligned} A(\boldsymbol{v}^n, \hat{x}\varphi_i) &= F(\hat{x}\varphi_i) \\ A(\boldsymbol{v}^n, \hat{y}\varphi_i) &= F(\hat{y}\varphi_i) \end{aligned} \qquad i = 1, \dots, m \tag{3.6}$$

In the following, we simply write $\mathsf{A}(\boldsymbol{v}^n) = \mathsf{F}$ for this nonlinear system of equations.

3.5. Newton method

Since the left hand side of the system of equations in (3.6) is nonlinear with respect to $\boldsymbol{v}^n$, we require an iterative method to find the new solution $\boldsymbol{v}^n$. We use Newton's method for this. Since the objective is to find a value $\boldsymbol{v}^n$ that solves the system, we write

$$f(\boldsymbol{v}^n) = \mathsf{F} - \mathsf{A}(\boldsymbol{v}^n) = 0.$$

Starting with an initial guess $\boldsymbol{v}^{n,(0)}$, the equation is linearised around the previous value $\boldsymbol{v}^{n,(l-1)}$ and solved for $\boldsymbol{w}$:

$$f\left(\boldsymbol{v}^{n,(l-1)}\right) = -f'\left(\boldsymbol{v}^{n,(l-1)}\right)\boldsymbol{w},$$

and we then update

$$\boldsymbol{v}^{n,(l)} = \boldsymbol{v}^{n,(l-1)} + \boldsymbol{w}.$$

Following Mehlmann et al. [49, p. 18], this process is terminated when the L^∞ norm of $\boldsymbol{w}$ is smaller than 10^{-13}.

The derivative $f'\left(\boldsymbol{v}^{n,(l-1)}\right)$ is the Jacobian of $-\mathsf{A}$ around $\boldsymbol{v}^{n,(l-1)}$, which is obtained with the Gâteaux derivative $A'(\boldsymbol{v})(\boldsymbol{w}, \varphi_i) = \frac{d}{ds}A(\boldsymbol{v} + s\boldsymbol{w}, \varphi_i)\big|_{s=0}$. Since $A'(\boldsymbol{v})(\boldsymbol{w}, \varphi_i)$ is linear in $\boldsymbol{w}$, we obtain a linear system of equations

$$\mathcal{A}\left(\boldsymbol{v}^{n,(l-1)}\right)\boldsymbol{w} = \boldsymbol{b}, \tag{3.7}$$

where $\mathcal{A}$ is the Jacobian and $\boldsymbol{b} = \mathsf{F} - \mathsf{A}(\boldsymbol{v}^{n,(l-1)})$.

Since $\boldsymbol{b}$ can be constructed following the previous section, it rests to find $\mathcal{A}\left(\boldsymbol{v}^{n,(l-1)}\right)$, of which the matrix entries $\mathcal{A}_{2i,j}$ and $\mathcal{A}_{2i+1,j}$ are equal to $A'(\boldsymbol{v}^{n,(l-1)})(\hat{e}_j, \hat{x}\varphi_i)$ and $A'(\boldsymbol{v}^{n,(l-1)})(\hat{e}_j, \hat{y}\varphi_i)$, where $\hat{e}_j$ is the j-th unit vector in $\mathbb{R}^{2m}$ and the operator is given by

$$\begin{aligned} A'(\boldsymbol{v})(\boldsymbol{w}, \boldsymbol{\varphi}) &= \frac{d}{ds}A(\boldsymbol{v} + s\boldsymbol{w}, \boldsymbol{\varphi})\bigg|_{s=0} \\ &= \frac{d}{ds}\Bigg(\left(\frac{H}{k}(\boldsymbol{v} + s\boldsymbol{w}), \boldsymbol{\varphi}\right) + (Tf_cH\boldsymbol{e}_z \times (\boldsymbol{v} + s\boldsymbol{w}), \boldsymbol{\varphi}) \\ &\qquad - (C_r\boldsymbol{\sigma}(\boldsymbol{v} + s\boldsymbol{w}), \nabla\boldsymbol{\varphi}) - (C_{\mathsf{ocean}}A\boldsymbol{\tau}_{\mathsf{ocean}}(\boldsymbol{v} + s\boldsymbol{w}), \boldsymbol{\varphi})\ \Bigg)\Bigg|_{s=0} \\ &= \left(\frac{H}{k}\boldsymbol{w}, \boldsymbol{\varphi}\right) + (Tf_cH\boldsymbol{e}_z \times \boldsymbol{w}, \boldsymbol{\varphi}) - (C_r\boldsymbol{\sigma}'(\boldsymbol{v})(\boldsymbol{w}), \nabla\boldsymbol{\varphi}) - (C_{\mathsf{ocean}}A\boldsymbol{\tau}'_{\mathsf{ocean}}(\boldsymbol{v})(\boldsymbol{w}), \boldsymbol{\varphi}), \end{aligned}$$

where

$$\begin{aligned} \boldsymbol{\sigma}'(\boldsymbol{v})(\boldsymbol{w}) =& \frac{H\exp(-C(1-A))}{\Delta(\dot{\boldsymbol{\epsilon}})}\left(\mathsf{tr}(\dot{\boldsymbol{\epsilon}}(\boldsymbol{w}))I + 2e^{-2}\dot{\boldsymbol{\epsilon}}'(\boldsymbol{w})\right) \\ &- \frac{H\exp(-C(1-A))}{\Delta^2(\dot{\boldsymbol{\epsilon}})}\left(\mathsf{tr}(\dot{\boldsymbol{\epsilon}})I + 2e^{-2}\dot{\boldsymbol{\epsilon}}'\right)\left(2e^{-2}\dot{\boldsymbol{\epsilon}}' : \dot{\boldsymbol{\epsilon}}'(\boldsymbol{w}) + \mathsf{tr}(\dot{\boldsymbol{\epsilon}})\mathsf{tr}(\dot{\boldsymbol{\epsilon}}(\boldsymbol{w}))\right) \end{aligned}$$

and

$$\boldsymbol{\tau}'_{\mathsf{ocean}}(\boldsymbol{v})(\boldsymbol{w}) = -\|\boldsymbol{v}_{\mathsf{ocean}} - \boldsymbol{v}\|_2\boldsymbol{w} + \frac{(\boldsymbol{v}_{\mathsf{ocean}} - \boldsymbol{v}) \cdot \boldsymbol{w}}{\|\boldsymbol{v}_{\mathsf{ocean}} - \boldsymbol{v}\|_2}(\boldsymbol{v}_{\mathsf{ocean}} - \boldsymbol{v}).$$

An overview of the Newton method applied to the sea ice momentum equation is given in Algorithm 2. In the next section, we describe how the remaining linear system is solved.

Algorithm 2 Newton iteration for time step t_n

Set $\boldsymbol{v}^{n,(0)} = \boldsymbol{v}^{n-1}$ and $l = 0$.
Compute vector F
while $\|\mathsf{F} - \mathsf{A}(\boldsymbol{v}^{n,(l)})\|_\infty \geq 10^{-13}$ **do**
    l = l+1
    Compute matrix $\mathcal{A}\left(\boldsymbol{v}^{n,(l-1)}\right)$
    Compute vector $\mathsf{A}(\boldsymbol{v}^{n,(l-1)})$ and let $\boldsymbol{b} = \mathsf{F} - \mathsf{A}(\boldsymbol{v}^{n,(l-1)})$
    Solve $\mathcal{A}\left(\boldsymbol{v}^{n,(l-1)}\right)\boldsymbol{w} = \boldsymbol{b}$
    Update $\boldsymbol{v}^{n,(l)} = \boldsymbol{v}^{n,(l-1)} + \boldsymbol{w}$
Let $\boldsymbol{v}^n = \boldsymbol{v}^{n,(l)}$

3.6. Solving of linear system

In the previous sections, we have reduced solving the linear system of equations in (3.6) to solving for $\boldsymbol{w}$ in (3.7) in each iteration of the Newton method for each time step. Solving this equation is done using using the GMRES method preconditioned by the multigrid method [30] with an ILU decomposition as smoothing operator.

4

Machine learning techniques

In this chapter, the machine learning methods are presented. We first describe the concept of artificial neural networks in Section 4.1, followed by an explanation of the training process in Section 4.2. Different optimisation algorithms and techniques for further enhancement of the training process are illustrated in Section 4.3 and Section 4.4, respectively. Lastly, Section 4.5 described a type of neural network specifically tailored towards image data, so-called convolutional neural networks.

Artificial Intelligence (AI) encompasses a wide range of technologies that aim to create 'intelligent' algorithms capable of performing tasks that typically require human intelligence. Within the realm of AI, Machine Learning (ML) serves as a crucial subclass, defined by Mitchell [53, p. 2] as a program's ability to improve its performance on a specific class of tasks through experience, measured by a particular performance metric. To illustrate this definition, consider a weather prediction algorithm that employs today's weather data to forecast tomorrow's conditions. A dataset consisting of weather data on consecutive days acts as the experience from which the algorithm learns, with the accuracy of temperature predictions assessed using metrics such as the mean squared error. This definition effectively differentiates ML from AI, as AI includes various approaches that may not necessarily involve learning from data or experience. Such examples can be found in symbolic reasoning, which includes rule based systems.

Machine Learning algorithms are employed on a diverse array of tasks. Some of the most common include classification, regression, clustering, generation, and dimension reduction. Furthermore, the ML techniques employed for these tasks come in many forms. Some of the simpler methods include linear regression, decision trees, and support vector machines, while more complex approaches can involve artificial neural networks. In this work, we essentially perform a regression analysis using an artificial neural network. The surrogate model we aim to develop performs predictions based on statistical inference, which is in essence a regression task.

4.1. Artificial neural networks

Artificial neural networks are as universal function approximators [54], capable of approximating any function in a given function space, provided they have sufficiently complexity. This versatility is the reason for their widespread application across various fields. The concept of artificial neural networks is inspired by the biological neural networks, from which they derive their name. Like their biological counterparts, they learn to recognise complex patterns from diverse data sources. In the realm of biology, neurons communicate through synaptic connections, whereas artificial neural networks use linear algebra and continuous optimisation techniques to process.

An artificial neural network comprises input nodes, hidden nodes, and output nodes. The input nodes receive external data and pass it to the hidden nodes. Each hidden node calculates a linear combination of the inputs it receives and applies an activation function to introduce nonlinearity. The output nodes then compute weighted linear combinations of the hidden nodes' outputs to generate the final predictions. Since we can organise every layer as a vector, a neural network is, in essence, a series of matrix-vector multiplications and activation functions. The input layer consists of a vector x_0 with input data that is multiplied with a matrix W_0 with coefficients referred to as weights and results

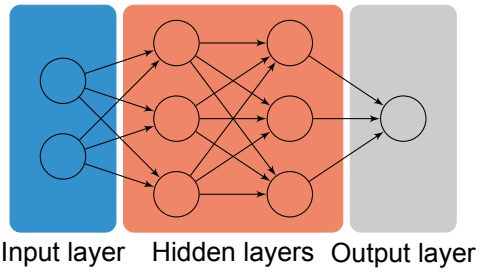


Figure 4.1: Visualisation of a multilayer perceptron (MLP) having two hidden layers.

in the first hidden layer $\boldsymbol{h}_1$. This vector $\boldsymbol{h}_1$ is then again multiplied with another matrix W_1 to either give us the output layer $\boldsymbol{y}$ or the next hidden layer $\boldsymbol{h}_2$, depending on the architecture of the network. By introducing a nonlinear function ϕ after each matrix-vector multiplication, the resulting function becomes nonlinear. This function ϕ is called the activation function. [55]

Multilayer Perceptron

The most simple neural network is the multilayer perceptron (MLP). It consists of at least three layers and uses non-linear activations. For example, suppose we have feature vector $\boldsymbol{x}_0 \in \mathbb{R}^2$, weight matrices $W_0 \in \mathbb{R}^{3\times 2}$, $W_1 \in \mathbb{R}^{3\times 3}$, and $W_2 \in \mathbb{R}^{1\times 3}$, and activation functions ϕ_0, ϕ_1, and ϕ_2, this gives us the following neural network:

$$\begin{aligned} \boldsymbol{h}_1 &= \phi_0\left(W_0\boldsymbol{x}_0 + \boldsymbol{b}_0\right) \\ \boldsymbol{h}_2 &= \phi_1\left(W_1\boldsymbol{h}_1 + \boldsymbol{b}_1\right) \\ y &= \phi_2\left(W_2\boldsymbol{h}_2 + b_2\right). \end{aligned} \tag{4.1}$$

Here, we have introduced an offset term $\boldsymbol{b}_0, \boldsymbol{b}_1 \in \mathbb{R}^3$ and $b_2 \in \mathbb{R}$, often referred to as the bias. This simple network is visualised in Figure 4.1.

Activation functions

An activation function ϕ can be any element-wise function. However, by the universal approximation theorem by Pinkus [56, p. 153], non-polynomial functions are needed to approximate any arbitrary function. Typical activation functions include the hyperbolic tangent, sigmoid, and rectified linear unit (ReLU), which are shown in Figure 4.2. They are given as follows:

$$\begin{aligned} \tanh(x) &= \frac{e^x - e^{-x}}{e^x + e^{-x}}, \\ \sigma(x) &= \frac{1}{1 - e^{-x}}, \\ \mathrm{ReLU}(x) &= \begin{cases} 0 & \text{if } x < 0, \\ x & \text{otherwise.} \end{cases} \end{aligned}$$

The choice for activation functions can depend on a range of factors. First of all, the image of the function is important. The activation used in the output layer dictates the image of the network. For example, if the ReLU function is used, the network's image is restricted to all positive values, but if the sigmoid function is used the image is contained in $[0, 1]$. In some applications it might be desirable to have no limitations on the networks image, in which cases the identity is often used.

The choice of activation function is not only important for the final layer, but it also effects the training behaviour of the network itself. As we will see in Section 4.2, neural networks are trained using the gradient of the output with respect to the network's weights, so understanding the effects of derivatives of activation functions is essential for understanding training dynamics. One typical issue relates to derivatives approaching zero, which will cause slow learning. Suppose that the derivative of an activation function is close to or equal to zero for a given input, by the chain rule, the updates of the preceding weight matrix will also be small. This is referred to as the *vanishing gradient problem* [55, p. 398] and is caused by gradients in subsequent layers approaching zero, essentially causing slow learning for

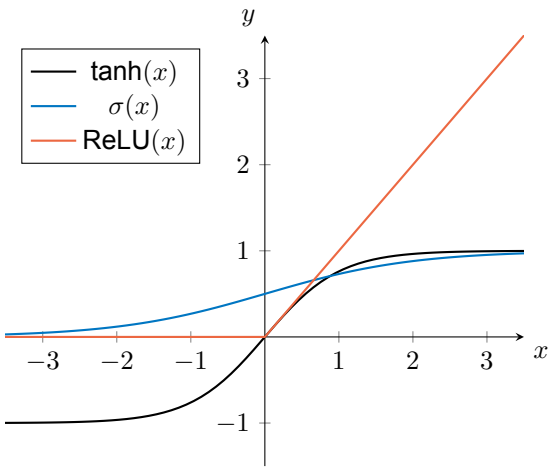


Figure 4.2: Three typical activation functions: hyperbolic tangent, sigmoid, and ReLU.

preceding layers. Since the gradients of the hyperbolic tangent and the sigmoid function are always smaller than 1, deep networks with these activations are prone to experiencing the vanishing gradient problem.

On the other hand, networks with a ReLU activation function can suffer from the *dead ReLU problem*, where nodes before a ReLU activation become inactive ('die') during training if all outputs lie in the negative domain. In that case, the ReLU function's gradient is equal to 0 for all inputs and the node's preceding weights are never updated again.

The identification of these problems caused a range of other activations to become more widespread adopted. For example, the Leaky ReLU function is given by:

$$\text{Leaky ReLU}(x) = \begin{cases} ax & \text{if } x < 0, \\ x & \text{otherwise,} \end{cases} \tag{4.2}$$

with $0 < a \ll 1$. With for example $a = 0.01$, the small gradient allows for an escape from this non-active state.

4.2. Neural network training

Having established the basic principles of artificial neural networks, this section will detail two fundamental components of neural network optimisation: the loss function and backpropagation. In the context of neural networks, this optimisation is often referred to as *training*. This involves adjusting the network's parameters (weights and biases) to improve the model's performance.

Loss functions

To score the performance of the network on a specific task, we employ a performance measure. In the case of a regression task, this can, for example, be the mean squared error (MSE), which is the L^2-norm of the difference between the network's output and the correct value. The latter is also called the *ground truth* or *label*. The loss function is a measure of how close the network's predictions are to the ground truth. The network's performance is improved through the use of a loss function that, via backpropagation and optimization algorithms, guides the adjustment of the network's weights.

Since a surrogate model essentially performs a regression, natural choices for loss functions include the mean squared error (MSE), mean absolute error (MAE) and the mean relative error (MRE). They

are defined as follows:

$$\text{MSE}_B(\boldsymbol{w}) = \frac{1}{|B|}\sum_{i\in B}\|\boldsymbol{y}_i - \hat{\boldsymbol{y}}_i\|_2^2\,, \tag{4.3}$$

$$\text{MAE}_B(\boldsymbol{w}) = \frac{1}{|B|}\sum_{i\in B}\|\boldsymbol{y}_i - \hat{\boldsymbol{y}}_i\|_1\,, \tag{4.4}$$

$$\text{MRE}_B(\boldsymbol{w}) = \frac{1}{|B|}\sum_{i\in B}\frac{\|\boldsymbol{y}_i - \hat{\boldsymbol{y}}_i\|_2}{\|\hat{\boldsymbol{y}}_i\|_2}, \tag{4.5}$$

where $\boldsymbol{w}$ represents all coefficients in the network, and B is the training dataset, such that for $i \in B$, $\hat{\boldsymbol{y}}_i$ is the i-th label and $\boldsymbol{y}_i$ is the i-th prediction by the network.

The choice of loss function influences the performance of the network on different aspects. For example the mean squared error penalises large outliers more, while the mean absolute error applies a linear penality which is less sentive to large outliers. Depending on the application, one might be preferred over the other.

Additionally, in some applications, it might be useful to compute the loss of a derived quantity. In the case of sea ice dynamics predictions, the error in the strain rate or deformation can be used instead of the predicted velocity, since strain is what causes linear kinematic features to form. This strain rate error (SRE) is defined as follows:

$$\text{SRE}_B(\boldsymbol{w}) = \frac{1}{|B|}\sum_{i\in B}\left\|\dot{\boldsymbol{\epsilon}}_i - \hat{\dot{\boldsymbol{\epsilon}}}_i\right\|_2^2 = \frac{1}{4|B|}\sum_{i\in B}\left\|\nabla(\boldsymbol{y}_i - \hat{\boldsymbol{y}}_i) + \nabla(\boldsymbol{y}_i - \hat{\boldsymbol{y}}_i)^T\right\|_2^2, \tag{4.6}$$

where $\dot{\boldsymbol{\epsilon}}$ is the strain rate introduced in (2.5). Alternatively, the sea ice concentrations after advection using the predicted velocity may be compared. The employment of these loss functions is further discussed in Chapter 6.

Furthermore, less data-driven and more physics-informed loss functions can be considered. To illustrate, the L^2-norm of the error in the differential equation may be taken. In this way, the model is trained directly on the underlying physics. The possibilities are endless, but more complex loss function can introduce complex behaviours in training. However, they can serve to focus the network's training on interesting features.

Backpropagation

Having previously discussed how loss functions serve as measures of performance of neural networks, it becomes evident that training these networks involves solving an optimisation problem. For instance, if a network's performance is measured by the MSE, the loss function's minimum is attained if the network's predictions coincide with the ground truth. This optimisation, commonly referred to as *training*, typically employs gradient based methods.

In the training process, effectively adjusting the network's parameters is done through calculating the gradient of the loss function with respect to these parameters, and performing gradient descent, which is detailed in the next section. This gradient is calculated through a systematic application of the chain rule across each layer, a method known as *backpropagation* [57]. Backpropagation provides the mechanism by which errors detected at the output are used to iteratively adjust the weights throughout the network, thereby minimising the loss function and improving the model's predictions.

The mathematical representation of a simple multilayer perceptron with three layers is given in (4.1). An arbitrary layer can be represented by

$$\boldsymbol{y} = \phi\left(W\boldsymbol{x} + \boldsymbol{b}\right), \tag{4.7}$$

where $\boldsymbol{y} \in \mathbb{R}^n$ is the output of the layer, $\boldsymbol{x} \in \mathbb{R}^m$ is the input, $W \in \mathbb{R}^{n\times m}$ is the weight matrix, $\boldsymbol{b} \in \mathbb{R}^n$ is the bias and ϕ is the activation function. Furthermore, we are given the loss value L on a specific instance. To update the network's parameters W and $\boldsymbol{b}$, we need to compute the partial derivatives $\frac{\partial L}{\partial W}$ and $\frac{\partial L}{\partial \boldsymbol{b}}$. Suppose we are given $\frac{\partial L}{\partial \boldsymbol{y}}$, which is a vector in $\mathbb{R}^n$. By the chain rule, the partial derivative

with respect to weight $W_{i,j}$ is given by

$$\begin{aligned}\frac{\partial L}{\partial W_{i,j}} &= \frac{\partial L}{\partial y_i}\frac{\partial y_i}{\partial W_{i,j}} \\ &= \frac{\partial L}{\partial y_i}\frac{d\phi}{z_i}\frac{\partial z_i}{\partial W_{i,j}} \\ &= \frac{\partial L}{\partial y_i}\frac{d\phi}{dz_i}x_j,\end{aligned} \tag{4.8}$$

where we've have introduced $\boldsymbol{z} = W\boldsymbol{x} + \boldsymbol{b}$. Similarly, the derivative of L with respect to bias b_j is

$$\begin{aligned}\frac{\partial L}{\partial b_i} &= \frac{\partial L}{\partial y_i}\frac{\partial y_i}{\partial b_i} \\ &= \frac{\partial L}{\partial y_i}\frac{d\phi}{z_i}\frac{\partial z_i}{\partial b_i} \\ &= \frac{\partial L}{\partial y_i}\frac{d\phi}{dz_i}.\end{aligned} \tag{4.9}$$

For the ReLU function, for example, the derivative is given by

$$\frac{d\phi}{dz} = \begin{cases}0, & \text{if } x \leq 0, \\ 1, & \text{otherwise.}\end{cases} \tag{4.10}$$

It only remains to obtain $\frac{\partial L}{\partial \boldsymbol{y}}$. If $\boldsymbol{y}$ is the output of the final layer, then this derivative is simply the derivative of the loss function with respect to the prediction. For the mean squared error, we get

$$\frac{\partial L_{\text{MSE}}}{\partial \boldsymbol{y}} = 2\left(\boldsymbol{y} - \hat{\boldsymbol{y}}\right)^T, \tag{4.11}$$

where $\hat{\boldsymbol{y}}$ is the ground truth. If, however, the layer in question is either a hidden layer or the input layer, the derivative of loss L with respect to the output of the layer is equal to the $\frac{\partial L}{\partial \boldsymbol{x}}$ of the next layer, since the output of the current layer is the input of the next layer. We, therefore, obtain

$$\frac{\partial L}{\partial \boldsymbol{y}} = \frac{\partial L}{\partial \boldsymbol{x}^+}, \tag{4.12}$$

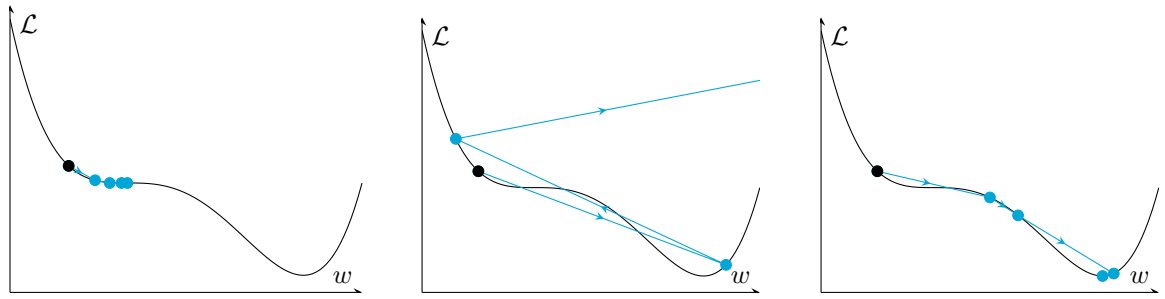
where $\boldsymbol{x}^+$ denotes the input of the next layer. For every layer, we therefore also need to calculate the derivative with respect to the input:

$$\begin{aligned}\frac{\partial L}{\partial x_j} &= \left(\frac{\partial L}{\partial \boldsymbol{y}}\right)^T \frac{\partial \boldsymbol{y}}{\partial x_j} \\ &= \left(\frac{\partial L}{\partial \boldsymbol{y}}\right)^T \frac{d\phi}{d\boldsymbol{z}}\frac{\partial \boldsymbol{z}}{\partial x_j} \\ &= \left(\frac{\partial L}{\partial \boldsymbol{y}}\right)^T \frac{d\phi}{d\boldsymbol{z}}\boldsymbol{W}_j,\end{aligned} \tag{4.13}$$

where $\boldsymbol{W}_j$ is the j-column of W, and the Jacobian $\frac{d\phi}{d\boldsymbol{z}}$ is given by

$$\frac{d\phi}{d\boldsymbol{z}} = \text{Diag}\left(\frac{d\phi}{dz}(z_1), \ldots, \frac{d\phi}{dz}(z_n)\right). \tag{4.14}$$

In summary, to obtain the gradients for every coefficient in the neural network, the gradient of the loss with respect to the output of the network needs to be computed, followed by computations of $\frac{\partial L}{\partial W}$, $\frac{\partial L}{\partial \boldsymbol{b}}$, and $\frac{\partial L}{\partial \boldsymbol{x}}$. This process starts from the last layer and works backwards through the network layer-by-layer, hence the name backpropagation.



(a) $\lambda < \lambda_0$ **(b)** $\lambda > \lambda_0$ **(c)** $\lambda = \lambda_0$

Figure 4.3: Three examples of the effect of different learning rates λ compared to the optimal learning rate λ_0. The trajectory is shown with the starting point marked in black.

4.3. Optimisers

With gradients computed via backpropagation, the next step is optimisation - navigating the loss landscape to find a global minimum. The optimisation algorithms, simply called *optimisers*, play a crucial role in neural network training: they iteratively adjust parameters to minimise the loss function and improve the model's performance. The choice of optimiser impacts the efficiency, susceptibility to local minima, and stability of the optimisation path.

Gradient descent

Gradient descent is an optimisation algorithm that operates by incrementally adjusting parameters in the direction of the negative gradient of the loss function. The optimisation of an arbitrary network coefficient w is given by:

$$w_{t+1} = w_t - \lambda \frac{\partial L_B}{\partial w}(w_t), \tag{4.15}$$

where the subscripts indicate the step number and $\frac{\partial L_B}{\partial w}$ is the derivative of the loss L on training set B with respect to the coefficient, as determined by backpropagation [58].

In gradient decent, the loss is computed for the entire training set. This means that the partial derivatives of individual instances computed through backpropagation are aggregated either through summation or averaging, depending on the loss function itself.

The step size $\lambda \in \mathbb{R}_+$, referred to as the *learning rate*, controls the magnitude of parameter updates, thereby determining the speed at which the network learns. Whether the learning rate is too small or too large depends on factors such as the complexity of the network, characteristics of the training data, the curvature of the loss landscape, and the specific optimisation algorithm being used. A very small λ may result in excessively slow convergence, since it prolongs the training duration unnecessarily. Conversely, an overly large λ can cause the updates to overshoot the optimal value, repeatedly missing the minimum and potentially even diverging. To illustrate the importance of choosing the right learning rate, the effect of different learning rates is visualised in Figure 4.3.

Stochastic gradient descent

While gradient descent uses the gradient of the loss for the entire training dataset, *stochastic gradient descent* (SGD) updates the model parameters based on the gradient of the loss from a randomly selected single data point or a small subset of the data at each iteration [55, p. 275]. This latter method is referred to as *minibatch gradient descent*, named for the subsets of the dataset, known as *minibatches*, used in each iteration. However, it is commonly referred to as SGD.

The main motivations for stochastic gradient descent are the computational expense of calculating gradients over the entire dataset and the introduction of randomness through minibatches, which has a regularizing effect on the training process [55, p. 276]. By calculating gradients on randomly sampled subsets, an unbiased estimate of the gradient on the whole dataset is obtained. This approach introduces a trade-off between computational efficiency and accuracy of the gradient estimates, influenced by the the chosen size of the minibatches.

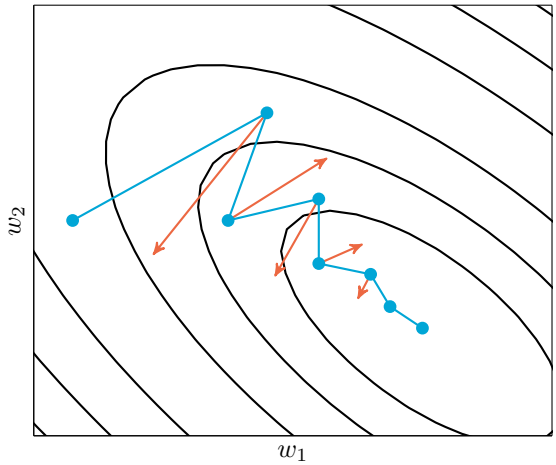


Figure 4.4: Effect of momentum on the trajectory across the loss landscape for two weights w_1 and w_2. The gradients are shown with orange arrows, while the path of gradient descent with momentum is shown in blue. Based on [55, Figure 8.5, p. 293]. Without momentum, the trajectory would follow the gradients marked by orange arrows.

Momentum

Even though stochastic gradient descent has proven effective, its convergence can be hindered by the instability inherent in gradient approximation. In response, Polyak developed the *momentum* technique [59] to provide more stability and increase convergence speed. This is achieved by replacing the partial derivative in (4.15) with a momentum term m_t that combines the current gradient and the previous momentum term m_{t-1}. Specifically, the update is given by:

$$m_t = \beta m_{t-1} - \lambda \frac{\partial L_B}{\partial w}(w_{t-1}) \tag{4.16}$$

$$w_t = w_{t-1} + m_t, \tag{4.17}$$

where β is the momentum parameter, tuning the influence of previous gradients on the current update. By incorporating a portion of the previous gradients, the system gains a 'memory' of past updates, which helps to stabilise the direction of the descent and often leads to faster convergence by smoothing out the trajectory. This momentum technique is illustrated in Figure 4.4.

Adaptive learning rates

Stochastic gradient descent and momentum form the basis for more complex optimisers. One such algorithm is Adam, introduced by Kingma and Ba [60], which enhances SGD by integrating momentum and *adaptive learning rates*.

Adam extends the concept of momentum, referred to as the *first moment*, by also computing what is known as the *second moment*. This second moment is an exponentially decaying average of the square of the gradient:

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2,$$

where the gradient is

$$g_t = \frac{\partial L_{B_t}}{\partial w}(w_{t-1}).$$

It serves as a measure of the magnitude of recent updates a coefficient has received, providing a gauge for the variability or noise in the gradient information.

The adaptive learning rate in Adam adjusts how each parameter is updated by dividing the first moment by the square root of the second moment:

$$w_t = w_{t-1} - \lambda \frac{m_t}{\sqrt{v_t} + \epsilon},$$

where ϵ is small parameter for numerical stability. This ensures that parameters with smaller updates in the recent past are adjusted more sensitively, allowing for finer control over the optimisation process.

This mechanism particularly aids in rapid convergence when the algorithm encounters convex regions in the loss landscape.

Furthermore, Adam incorporates a bias correction step to counteract the initial zero-bias in the first and second moments, ensuring that the moments are more accurate during the initial phase of training:

$$m_t = \frac{m_t}{1-\beta_1^t},$$
$$v_t = \frac{v_t}{1-\beta_2^t}.$$

The Adam algorithm is shown in Algorithm 3.

Algorithm 3 Adam algorithm for network $\mathcal{NN}$

Given learning rate $\lambda \in \mathbb{R}_+$, momentum parameters $\beta_1, \beta_2 \in [0, 1)$, and small ϵ for numerical stabilisation (typically $\epsilon = 10^{-8}$).
Initialise $m_t, v_0, t = 0$.
while not converged **do**
  $t \leftarrow t + 1$
  Sample minibatch B_t from dataset
  Compute predictions $\boldsymbol{y}_i$ for $i \in B_t$ with $\mathcal{NN}$ (forward pass)
  Compute gradients $g_t = \frac{\partial L_{B_t}}{\partial w}(w_{t-1})$ for each coefficient w (backpropagation)
  Compute first moment $m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$
  Compute second moment $v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$
  Perform bias correction $m_t \leftarrow m_t/(1-\beta_1^t)$ and $v_t \leftarrow v_t/(1-\beta_2^t)$
  Update coefficients $w_t = w_{t-1} - \lambda m_t / \left(\sqrt{v_t} + \epsilon\right)$
return Network $\mathcal{NN}$

4.4. Enhancing training convergence

Having detailed different optimisation algorithms, this section explores several advanced techniques designed to optimise convergence of model training. These methods, including gradient clipping, learning rate scheduling, explicit regularisation, and early stopping, play essential roles in ensuring that the model reaches optimal performance efficiently.

Central in this is the concept of *overfitting*, which is the phenomenon where a model learns the detail and noise in the training data to an extent that it negatively impacts the performance on new data. Prechelt [61] states that overfitting occurs as the error on the training data decreases, while the error on the validation dataset starts to increase. A sketch of this process is presented in Figure 4.5. As a result of overfitting, specifically in our case, a model has not inferred the physical laws governing sea ice dynamics, but instead simply memorises the training data by directly associating the specific inputs with their corresponding outputs. This means that the model captures patterns unique to the training set without understanding the underlying principles, leading to poor generalisation to unseen data.

Preventing overfitting is easier said than done, however, since many factors can cause a model to overfit. Firstly, the size of the dataset that is used to train the model on is an important factor, since there must be enough data for the model to actually infer the physical laws. Another typical factor is the size of the model in terms of the number of trainable parameters. A model with too many parameters relative to the amount of data can fit noise and spurious correlations that do not represent the true dynamics of sea ice. To address these challenges, we have implemented various techniques to balance model complexity and enhance generalisation to unseen data

Conversely, *underfitting* occurs when a model is too simple to represent the underlying patters in the data, resulting in poor performance on both training and validation datasets. Underfitting can occur if the model is not complex enough or if the training process is inadequate, failing to reach the loss function's global minimum. The latter can, for example, occur if the learning rate is too high, disallowing the optimisation algorithm from settling into deeper, more accurate minima of the loss and instead causing it to overshoot without converging.

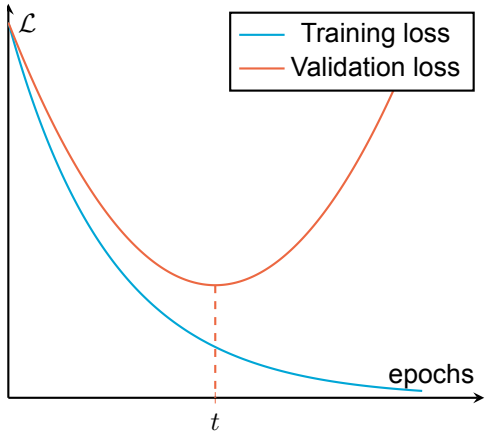


Figure 4.5: Sketch of the idealised training and validation loss. A dashed line marks the epoch t where the model attains the global minimum of the validation loss. The model is said to be overfit after t, and underfit before t. Based on [61, Figure 2.1, p. 56].

Learning rate scheduling

The first technique employed for improved model convergence is *learning rate scheduling*. This method involves dynamically adjusting the learning rate, rather than keepin it constant throughout the training proccess. Common examples of learning rate schedules include linear and step-wise schedules. In a linear schedule, the learning rate decreases uniformly each epoch:

$$\lambda_t = \lambda_0 - \delta\lambda \cdot t, \tag{4.18}$$

where λ_0 is the initial learning rate, reduced by $\delta\lambda$ at each epoch. Alternatively, a step-wise schedule reduces the learning rate in discrete intervals, such as halving it every predetermined number of steps.

Furthermore, the learning rate can be adapted based on the progress of the training process. For instance, if there is no significant improvement in the validation loss after several training steps - indicating that training might have plateaued - the learning rate can be decreased. This more dynamic approach balances the speed of convergence in early training with the effective convergence to a minimum as training progresses.

Gradient clipping

Another obstacle to reaching the global minimum is the phenomenon known as *exploding gradients*. This occurs when abrupt changes in the loss function produce excessively large gradient values. These large gradients can lead to disproportionately large updates to the model's weights, thereby hindering the optimisation process and preventing the model from settling into the desired global minimum.

These exploding gradients can be mitigated by applying *gradient clipping*. This method was first introduced by Mikolov [62, p. 37] to combat exploding gradients in recurrent neural networks, but is used widely to stabilise neural training. In short, gradient clipping restricts the total norm of the gradients. To illustrate, given the partial derivative of the loss L with respect to an arbitrary network coefficient w, the the gradients can be restricted by setting

$$\frac{\partial L}{\partial w}(\boldsymbol{x}) = \frac{\partial L}{\partial w}(\boldsymbol{x})/G \quad \text{if } G > 1, \tag{4.19}$$

where G is the norm of the gradients of all weights w of the network:

$$G = \sqrt{\sum_w \left(\frac{\partial L}{\partial w}(\boldsymbol{x})\right)^2}. \tag{4.20}$$

Since the gradients are not clipped, but scaled, gradient scaling might be a better suited term.

Explicit regularisation

While the previous techniques primarily focus on improving convergence towards the global minimum to mitigate underfitting, *explicit regularisation* aims to prevent overfitting. Regularisation refers to any technique that aims to address the ill-posedness of a problem, such as when a model's excessive complexity allows for multiple potential solutions. In turn, this can result in overfitting since the model has the capacity to learn the noise in the data. Examples of regularisation techniques include early stopping, which is detailed in the following section, and the addition of random noise to the inputs during training, as is described in Section 5.3. These are referred to as *implicit regularisation* methods, which modify the training process to reduce overfitting risks. In contrast, explicit regularisation directly incorporates a penalty term into the loss function, effectively constraining the model to ensure a unique and generalised solution.

This regularisation term imposes a penalty on the complexity of the model. It encourages simpler solutions by either reducing the magnitude of the model parameters or eliminating non-essential parameters, thereby enhancing the model's ability to generalise to new data. Two common explicit regularisation techniques are LASSO (Least Absolute Shrinkage and Selection Operator) [63] and Ridge [64], which respectively impose an L^1 and L^2 norm on the network's coefficients:

$$R_1(\mathcal{NN}) = \sum_{w \in \mathcal{NN}} |w|, \tag{4.21}$$

$$R_2(\mathcal{NN}) = \sum_{w \in \mathcal{NN}} (w)^2. \tag{4.22}$$

These two terms differ in how they penalise the coefficients of the network: LASSO typically results in a sparse model by driving coefficients to zero, whereas Ridge encourages smaller, more uniformly distributed coefficients [55, p. 232]. Thus, choosing between LASSO and Ridge regularisation depends largely on the specific characteristics of the dataset.

Early stopping

Another method to improve model performance is the implementation of an early stopping mechanism. This approach monitors the model's performance on a validation dataset during training. When the validation error stops decreasing or begins to increase, it signals a decline in the model's ability to generalise, warranting an end to further training.

A stopping criterion is used to decide to terminate training, after which the result of training is the best network in terms of the validation loss. In the idealised form, the validation loss is smooth and only has a global minimum (without additional local minima), which case is visualised in Figure 4.5.

However, in practice, the validation loss is often not smooth and has many local minima. Terminating training too soon in a local minimum that is significantly worse than the global minimum, results in an underfit model. Terminating too late, however, wastes computational power. This trade-off between performance and efficiency, makes choosing a stopping criterion more art than science.

Prechelt [61] introduces three classes of stopping criteria: those based on the ratio between the training and validation loss, based on the improvement of the validation loss, and based on the ratio between the two losses corrected for recent improvements. The simplest of these, the second, is implemented in this work (see Section 6.3). More specifically, the criterion that is used terminates training after a predetermined number of epochs where the validation loss has increased.

Batch normalisation

To conclude the description of techniques that enhance model convergence, *batch normalisation* is discussed. This method addresses the issue of internal covariate shift, wherein the distribution of inputs of each layer shifts as the parameters of preceding layers evolve during training. This phenomenon slows down the training process as it needs to compensate for the shifting distributions. Batch normalisation, introduced by Ioffe and Szegedy [65], was developed to tackle internal covariate shift by stabilising the input distributions.

In short, batch normalisation works by standardising the inputs or outputs of activation functions, collectively referred to as activations. The process involves replacing any given activation x_j with:

$$\hat{x}_j = \frac{x_j - \mu_j}{\sigma_j}, \tag{4.23}$$

where μ_j represents the mean and σ_j the standard deviation of the activations of the j-th feature in a hidden layer.

Ideally, μ_j and σ_j would be computed over the entire dataset, but if a stochastic gradient descent algorithm is employed, only the activations from the current mini-batch B are available during training. To account for this limitation, batch normalisation employs a running average of the mean and standard deviation, updated by:

$$\mu_j \leftarrow (1-\gamma)\mu_j + \gamma \frac{1}{|B|} \sum_{i \in B} x_i, \tag{4.24}$$

$$\sigma_j \leftarrow (1-\gamma)\sigma_j + \gamma \sqrt{\frac{1}{|B|} \sum_{i \in B} \left(x_i - \sum_{n \in B} x_n \right)^2}, \tag{4.25}$$

where γ is the momentum parameter of the running average, typically set to 0.1. This approach helps stabilise the input distributions to each layer during training by ensuring that the normalisation is based on consistent, representative statistics. When training has finished, however, only the result of the running averages is used, such that the statistics are not influenced by new data during validation or testing.

For multilayer perceptrons, which deal with vector data, the mean and standard deviation are computed separately for each feature. However, in convolutional networks dealing with image data, which will be introduced in the following section, the statistics are computed for each *channel*, meaning that the mean and standard deviation are computed over the entire images. In this way, the standardisation is invariant under spatial transformations of the image.

In their work, Ioffe and Szegedy [65, p. 3] applied batch normalisation directly before the activation function of hidden layers. They reason that this is superior since the outputs of activation function are more non-Gaussian, meaning that they are less suitable for standardisation by a linear transformation. In other words, the results of the matrix-vector multiplication $W\boldsymbol{x} + \boldsymbol{b}$ (before the activation function) is more likely to be symmetric, and is therefore more likely to eliminate covariate shift. Other works, however, argue for the opposite and place batch normalisation after activation functions [66].

4.5. Convolutional neural networks

When selecting a neural network architecture, computational efficiency is an important consideration. As we have seen previously, the size of the multilayer perceptrons (in terms of the number of weights) scales with the size of the input data. In the context of sea ice dynamics, we work with multidimensional data. For example, Mehlmann et al. [49] work on grids up to 257×257. Traditional MLPs tend to become inefficient for such tasks. This inefficiency arises because MLPs operate with fully connected layers, presuming interactions between every input feature, thereby generating very large weight matrices. For instance, a 257×257 image under the MLP architecture would require an impractical number of weights, scaling up to $66,049$ for each output pixel, making the model computationally slow when the output size lies in the same order of magnitude as the input dimensions.

Convolutional neural networks (CNNs), introduced by LeCun et al. [67], on the other hand, offer a more suitable approach for handling spatial data, such as in this work. CNNs exploit three concepts: sparse local interactions, parameter sharing, and equivariant representations [55, p. 330]. These principles facilitate efficient data processing, reducing computational demands while enhancing model performance.

The multilayer perceptron employs dense weight matrices, assuming that each input unit interacts with every output unit. In contrast, CNNs make use of sparse interactions through the use of smaller, localised filters (or kernels) that interact only with a portion of the input at a time. This approach limits interactions to local regions, significantly reducing the dimensionality and computational complexity of the problem. For instance, rather than mapping every pixel-to-pixel connection separately, a kernel of size $P \times Q$ moves across the input image. This not only cuts down on the number of computations, but also drastically reduces the number of weights required.

Furthermore, besides reducing the dimensionality of the problem, the use of a convolution operation also has the benefit of parameter sharing. This means that the same set of weights in a kernel is applied repetitively across the entire input field. This technique is particularly effective because features (e.g. ridges or leads in sea ice) tend to appear in various locations across the domain. By reusing the same

parameters regardless of position, CNNs utilise data more efficient and learn effectively by recognising that the same types of features can appear throughout the input space. This not only streamlines the learning process, but also significantly enhances the network's ability to generalise from fewer examples.

Lastly, as a result of applying kernels across the input image, convolutional networks create feature maps that capture important information from the data. This process is inherently equivariant, meaning that if the input data is transformed through, for example, translation, the feature maps adapt correspondingly. This characteristic ensures that CNNs can generalise across the spatial domain, ensuring that once a feature is learned in one part, the network can recognise it anywhere.

Convolutional layers

Moving from the conceptual framework to the practical implementation of convolutional networks, we explore the mechanics of convolutional layers. Here, data such as velocity $\boldsymbol{v}$ is represented by a matrix in $\mathbb{R}^{2\times M\times N}$. In this case, there are two features - in the context of CNNs called channels - one for each velocity component and two spatial dimensions of sizes M and N. Transitioning from C_i to C_o channels necessitates $C_i \cdot C_o$ kernels of shape $P \times Q$, which results in a matrix in $\mathbb{R}^{C_i\times C_o\times P\times Q}$ with trainable parameters. Since kernels are typically 3×3 or 5×5, this kernel matrix is much smaller than the weight matrix of a fully-connected layer.

Even though the operation is often called a convolution, widely used libraries actually implement cross-correlation instead [68, 69] and the two are used interchangeably in the literature [55, p. 329]. Cross-correlation and convolutions are equivalent by flipping the kernel in both direction. The two-dimensional input $x \in \mathbb{R}^{N\times M}$ cross-correlated with kernel $K \in \mathbb{R}^{P\times Q}$ is given by

$$y_{i,j} = (x * K)_{i,j} = \sum_{p=0}^{P-1}\sum_{q=0}^{Q-1} K_{p,q}x_{i+p,j+q}, \tag{4.26}$$

where $*$ denotes the cross-correlation operation and the result is given by $y \in \mathbb{R}^{(M-P+1)\times(N-Q+1)}$. In the following, this operation will be referred to as convolution. Moreover, in convolutional layers, every input channel x^m is convolved with kernel $K^{m,n}$, which yields output channel y^n:

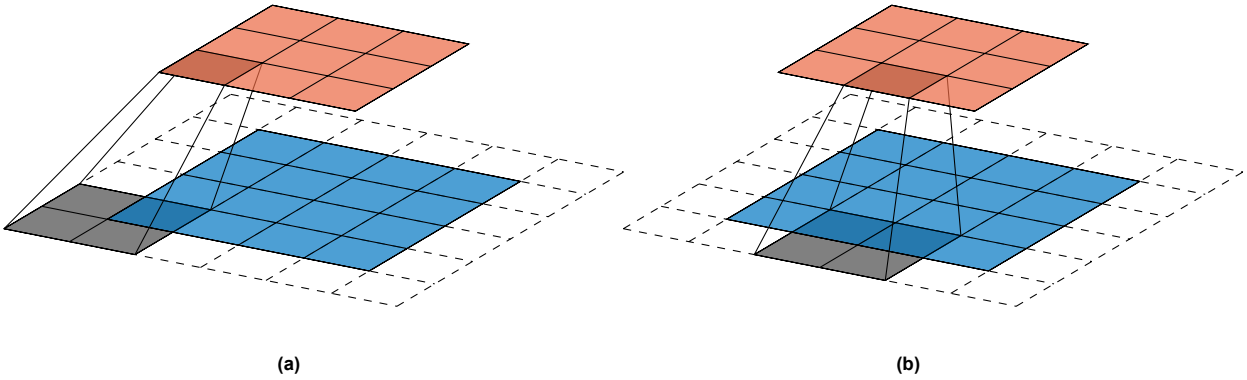
$$y^n = \sum_{m=1}^{C_i} x^m * K^{m,n}. \tag{4.27}$$

Stride and padding

Beyond the kernel shape and the number of output channels, hyperparameters such as stride and padding significantly influence the architecture of convolutional neural networks. The stride, which specifies the step size with which the kernel is moved across the input, effectively dictates the spatial dimensions of the output feature map. For instance, employing a stride of 2 approximately reduces the spatial size by half in each dimension, thus increasing the receptive field of the next layer and impacting the overall network efficiency by decreasing the computational load. Conversely, padding involves adding margins of pixels around the input image. Typically, the pixels take on the value of zero, but other values such as the value of the adjacent pixel can be applied. While padding can be used to maintain the original input size after convolution, it also ensures that pixels along the edges contribute fully to the output. If not, pixels at the borders appear only once in the convolution, thus enhancing the model's ability to capture features at the image borders. Figure 4.6 shows a visualisation of striding and padding in a convolutional layer.

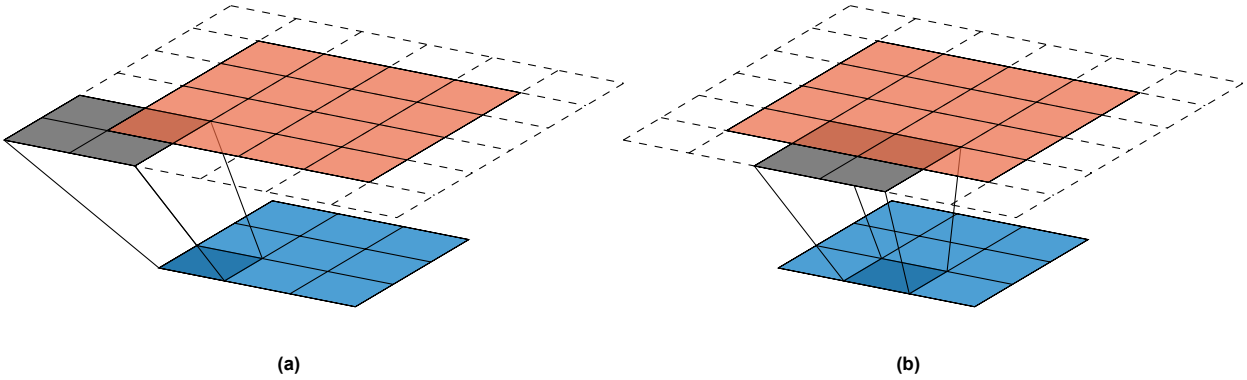
Deconvolutional layers

Where convolutions are useful to decrease spatial dimensions and compress information, its inverse may be used to decompress information into larger spatial dimensions. This operation is called deconvolution, inverse convolution or transposed convolution. Similar to convolutions, deconvolutional layers introduced by Zeiler et al. [71] use a kernel to map pixel-to-pixel relations. Here, however, each input pixel is multiplied by the entire kernel, and the resulting matrix is placed into the output feature map in a way that spreads the pixel's influence over a larger area. This operation involves overlaying the expanded kernel outputs such that they overlap and sum together, allowing the network to learn



(a) (b)

Figure 4.6: Example of convolution with a 2×2 kernel, padding of 1, and stride 2. The input image is shown in blue and the output is shown in orange. The dashed grid shows the padding. The receptive field and the corresponding output pixel are marked in grey for two iterations. Adapted from [70].



(a) (b)

Figure 4.7: Example of deconvolution again with a 2×2 kernel, padding of 1, and stride 2. The input is shown in blue and the output in orange. The dashed grid shows the padding. Adapted from [70].

how to reconstruct higher-resolution representations from lower-resolution inputs. If we write $\hat{x}$ as the padded input, then the deconvolution operation denoted by $/$ using kernel K is given by

$$y_{i,j} = (x/K)_{i,j} = \sum_{p=0}^{P-1} \sum_{q=0}^{Q-1} K_{P-1-p,Q-1-q} \hat{x}_{i+p,j+q}. \tag{4.28}$$

Furthermore, without any padding, the pixels on the edges are influenced only by one input pixel. To ensure that border pixels in the output are computed using more than one value in the input, a padding may be specified. In this context, however, the padding specifies a number of layers of border pixels to be trimmed from the output. In the equivalent formulation, this parameter specifies how many layers of border pixels should be trimmed off from the padded input. A visualisation of a deconvolution is shown in Figure 4.7. Furthermore, a striding may again be specified. This striding determines the step size of the kernel being overlayed on the output. A stride of two, roughly doubles the output image's dimensions.

Deconvolutional layers, however, may lead to artefacts such as chequerboard patterns due to a fragile learning process [72]. Especially features that lie farther from the average pixel value - linear kinematic features for example - tend to suffer from these artefacts. To combat this, Odena et al. [72] propose a combination of upsampling and convolution. In the first step, the input image is resized using nearest neighbour interpolation or bilinear interpolation, after which a convolutional layer is applied.

Bottleneck networks

Convolutional layers - especially those that apply striding - can be used to decrease the spatial dimensions of the output image significantly. Such layers are called *down-convolutions*. On the other hand, deconvolutions - or a combination of upsampling and convolutions - are referred to as *up-convolutions*

if they increase the spatial dimensions significantly. In network architectures that are employed to transform image data to again produce images, a combination of down-convolutions and up-convolutions can be used in a *bottleneck* structure. Here, a series of down-convolutions is first used to transform the input image into a vector, the encoding. Next, a series of up-convolutions is applied on this encoding to produce image data. This is a form of dimensionality reduction if the vector size is (much) smaller than the input and output sizes.

5 Data generation

Since the goal is to find a surrogate model for the sea ice momentum equation, the data should consist of the different variables in the equation. These variables are the initial velocity v, the sea ice height H, sea ice concentration A, wind velocity v_{atm} and finally the ocean velocity v_{ocean}.

Since we are pursuing a data-driven method, where we want to predict the velocity field in the next step from these variables, we also need the velocity of the next step, which will act as the 'label' in the training. Gascoigne simulations will be employed for this data generation, where the physical constants from Table 2.1 are used. These constants will not be fed explicitly into the neural networks since they are taken as constants. The code used to implement the problems described in this chapter is made available on GitHub [73].

5.1. Benchmark problem

Mehlmann et al. [49] compared the performance of various discretisations on a so-called 'benchmark problem'. This problem represents sea ice deformations caused by a moving anticyclone on a square domain Ω that is 512 km by 512 km. The simulation is run to cover a period of approximately 2 days. The initial conditions on the whole domain are as follows:

$$H = 0.3, \quad A = 1.0, \quad \boldsymbol{v} = \boldsymbol{0}. \tag{5.1}$$

In the benchmark problem, the ocean current velocity $\boldsymbol{v}_{\text{ocean}}$ is circular around the centre of the domain $(256 \text{ km}, 256 \text{ km})$. The ocean current is given by

$$\boldsymbol{v}_{\text{ocean}} = \frac{v_{\text{ocean}}^{\text{max}}}{256} \begin{pmatrix} y - 256 \\ -(x - 256) \end{pmatrix}, \tag{5.2}$$

where x and y are in kilometres and $v_{\text{ocean}}^{\text{max}} = 0.01$ m/s. The oceanic velocity field on the domain Ω is visualised in Figure 5.1.

The wind field is given by

$$\boldsymbol{v}_{\text{atm}}(x, y, t) = -v_{\text{atm}}^{\text{max}} \boldsymbol{s}(x, y, t) \begin{pmatrix} \cos(\alpha) & \sin(\alpha) \\ -\sin(\alpha) & \cos(\alpha) \end{pmatrix}, \tag{5.3}$$

where

$$\boldsymbol{s}(x, y, t) = \frac{1}{r_0 \exp(-1)} \begin{pmatrix} x - m_x(t) \\ y - m_y(t) \end{pmatrix} \exp\left(-\frac{\sqrt{(x - m_x(t))^2 + (y - m_y(t))^2}}{r_0} \right). \tag{5.4}$$

The normalised wind velocity $s(x, y, t)$ reaches its maximum at

$$r(x, y, t) = \sqrt{(x - m_x(t))^2 + (y - m_y(t))^2} = r_0,$$

where $\boldsymbol{s}(r_0) = 1$. The cyclone's midpoint starts in the centre of the domain at time $t = 0$ and moves towards the upper right:

$$m_x(t) = m_y(t) = 256 + 50t, \tag{5.5}$$

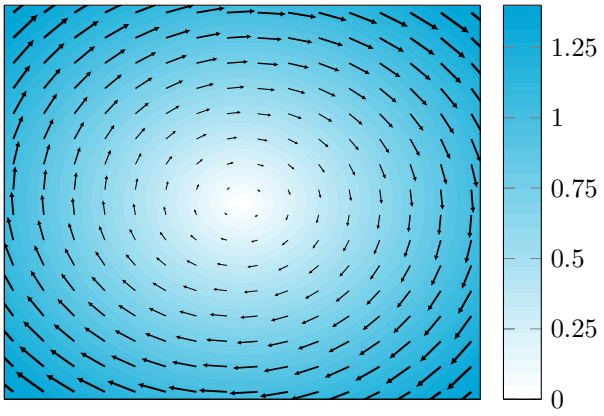


Figure 5.1: Ocean velocity of the benchmark problem. The magnitude of the ocean velocity relative to $v_{\text{ocean}}^{\max}$ is visualised by a contour plot. The direction of the current is shown by quivers.

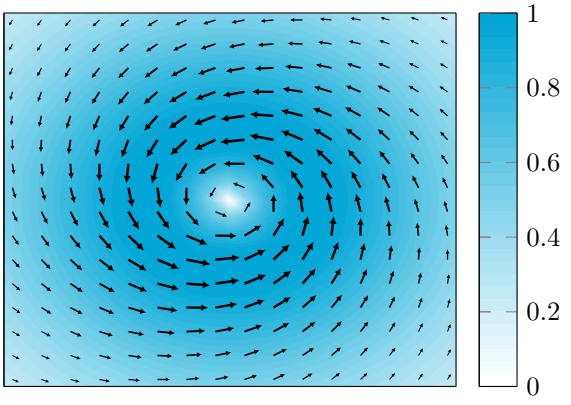


Figure 5.2: Visualisation of a cyclonic wind field centred in the middle of the domain and with $\alpha = \frac{2}{5}\pi$. The magnitude of the velocity is visualised by shading the background in terms of $v_a^{\max}$

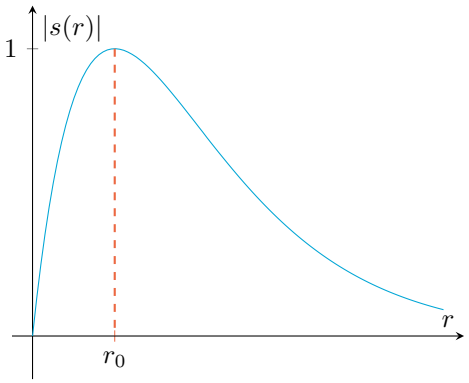


Figure 5.3: Sketch of the magnitude of the normalised wind velocity $s(r)$ given in (5.4). The maximum velocity is reached for $r = r_0$, which is marked by a dashed line.

where m_x and m_y are in kilometres and t is in days. In this benchmark problem,[1] the maximum wind velocity is given by

$$v_{\text{atm}}^{\max} = 11 \text{ m/s}. \tag{5.6}$$

Lastly, the cyclonic wind is characterised by the convergence angle α. Perfect circular motion is obtained for $\alpha = \frac{1}{2}\pi$, and thus $\frac{1}{2}\pi - \alpha$ is the angle with which the the wind velocity converges with respect to this circular motion. The wind velocity field on the domain Ω at time $t = 0$ is shown in Figure 5.2. Moreover, a plot of the normalised wind velocity magnitude $|s|$ is given in Figure 5.3.

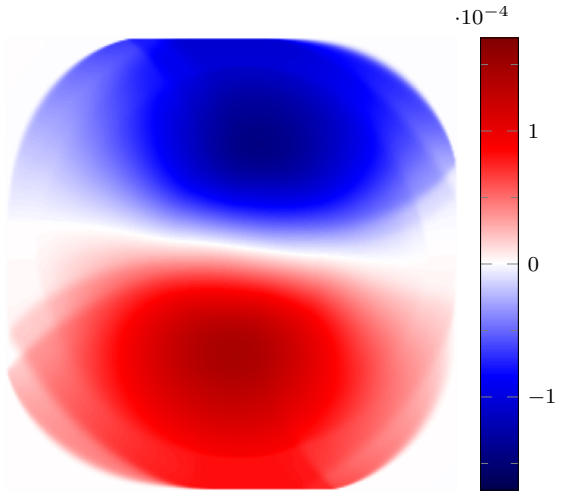
At the boundary of the domain, the a no-slip boundary condition is enforced, meaning that

$$\boldsymbol{v} = \boldsymbol{0} \text{ on } \partial\Omega. \tag{5.7}$$

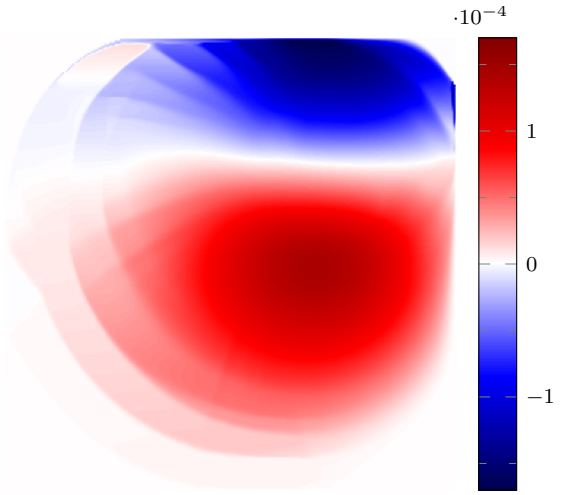
This condition mimics a land border, resulting in a square ocean.

To illustrate, the solution of the benchmark problem using the Gascoigne implementation is visualised in Figure 5.4. The sea ice velocities for two different time steps are shown. Furthermore, the sea ice concentration and shear deformation are shown for the last time step.

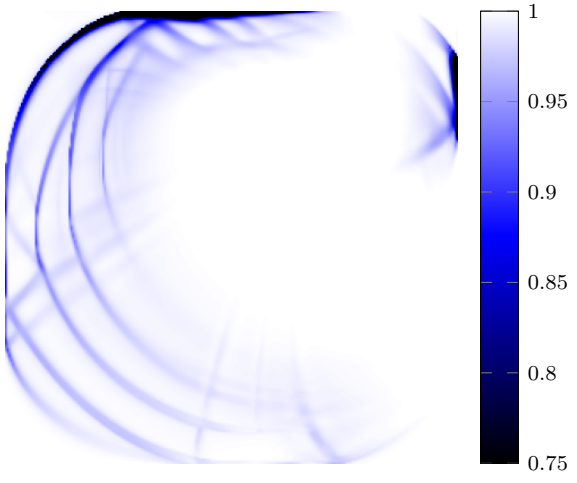
[1] Mehlmann et al. [49] set $v_a^{\max} = 15$ m/s. Their function s is scaled slightly differently, causing it to reach its maximum of e^{-1}. In this current work, this function is scaled such that its maximum is equal to 1, and thus $v_{\text{atm}}^{\max}$ must be scaled down to obtain the same wind velocity field.



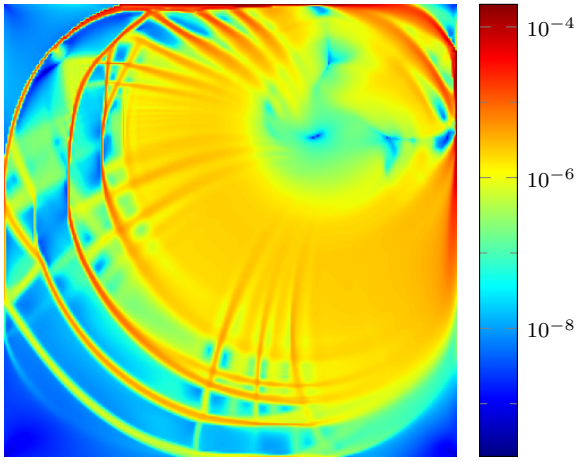
(a) Sea ice velocity in the x-direction after 10 time steps.



(b) Sea ice velocity in the x-direction after 90 time steps.



(c) Sea ice concentration after 90 time steps.



(d) Shear deformation after 90 time steps.

Figure 5.4: Solution of the benchmark problem using the B-grid discretisation and the Gascoigne library for the numerical computations.

5.2. Training data

In order to train a model that is able to predict the velocity in the benchmark problem, we need training data from which the model is able to infer the sea ice dynamics needed for this problem. In order to gauge the performance of the model on the benchmark problem, the training data should be similar, but not too similar since the model should generalise instead of overfit. The choice is made to vary only the wind field and the initial sea ice height in the dataset. The initial sea ice velocity and concentration, as well as the oceanic current is thus the same for all test cases.

The main reason for limiting the number of variations is to simplify the training process, since we can only generate a limited number of samples. Another factor, however, is that we are most interested in cases that resemble the benchmark problem. From experience, we see that linear kinematic features are most clearly formed when the initial concentration is 1.0, and the oceanic current is much less influential than the wind in forcing the sea ice movements.

Linear kinematic features are typically generated by two types of wind: (anti)cyclonic winds and winds pushing sea ice against a solid border. Other wind patterns are less interesting because the sea ice movements they cause are relatively easy to compute. Since the goal is to predict linear kinematic features, the wind velocity fields in the dataset will consist of cyclonic and anticyclonic winds. The case where wind pushes the ice against a solid border is given for free when the (anti)cyclone's centre is close to the border.

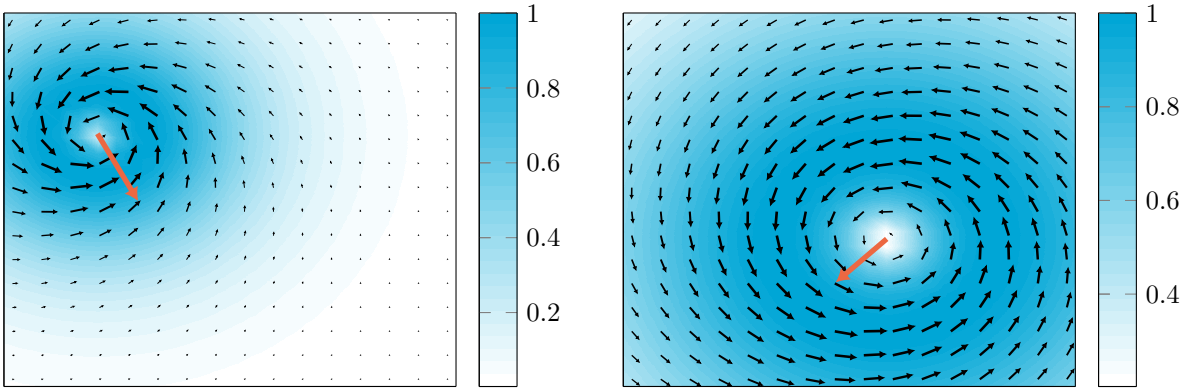
The (anti)cyclonic winds are generated randomly: their maximum wind velocity $v_a^{\max}$, eyewall radius r_0, initial centre $(m_x(0), m_y(0))$, centre velocity $\partial_t(m_x(t), m_y(t))$, and convergence angle α are drawn from random distributions. Moreover, the variable 'cyclone' corresponds to whether the wind field represents a cyclone or anticyclone. If the wind field represents an anticyclone, the wind direction is reversed by multiplying (5.3) by -1. The distributions are chosen in such a way that the variable's value corresponding to the benchmark problem is contained in the distribution. The distributions are all uniform and their domains are shown in Table 5.1. Moreover, two examples of wind velocity fields generated with different parameters are shown in Figure 5.5.

Table 5.1: Summary of the parameter settings used to generate the training dataset. For every variable both the domain of the uniform distributions from which random values are drawn and the corresponding value from the benchmark problem are given. The benchmark values are taken from [49].

Parameter	Domain	Benchmark value
H_0	$[0.1, 0.5]$ m	0.3 m
$m_x(0)$, $m_y(0)$	$[100, 400]$ m	256 m
$\partial_t m_x(t)$, $\partial_t m_y(t)$	$[-0.8, -0.4] \cup [0.4, 0.8]$ m/s	0.579 m/s
$v_a^{\max}$	$[6, 15]$ m/s	11 m
α	$[\pi/3, \pi/2]$	$\frac{2}{5}\pi$
r_0	$[60, 150]$ m	100 m
cyclone	$\{-1, 1\}$	1

In total, 800 parameter combinations are drawn at random. Each of these parameter combinations is used to generate a time series of sea ice dynamics. Starting from the initial conditions for ice concentration A and the sea ice velocity, defined in (5.1) respectively, the Gascoigne implementation is used to simulate the sea ice dynamics with time steps of 2000 s. The forcing terms are given by (5.2), where $v_{\text{ocean}}^{\max} = 0.01$ m/s is the same as in the benchmark problem, and by (5.3) with its parameters chosen randomly. Lastly, the initial sea ice height H_0 is chosen randomly.

For each test case, 30 time steps are generated. It was found that there is a clear difference in the performance of models for roughly the first ten time steps compared to the other steps. The first several steps from these initial conditions are not typically physical, since they do not represent states that are realistic. In these initial steps, the wind - and to a lesser degree the ocean - forcing is very dominant and the change in velocity is typically very correlated to the the wind field. In later steps, however, the forcing terms are counterbalanced to a certain extend by the internal stress. Since we are most interested in replacing the expensive stress computation, we want to focus the model's attention to this last category, where the wind is less dominant. The first 10 time steps of each time series are, therefore, excluded from the dataset, which results in 20 time steps for each parameter combination.



(a) $\boldsymbol{m}(0) = (100, 350)$ m, $\alpha = \frac{1}{3}\pi$, $r_0 = 60$ m, $\partial_t \boldsymbol{m}(t) = (0.4, -0.8)$ m/s

(b) $\boldsymbol{m}(0) = (300, 200)$ m, $\alpha = \frac{3}{7}\pi$, $r_0 = 130$ m, $\partial_t \boldsymbol{m}(t) = (-0.5, -0.6)$ m/s

Figure 5.5: Two examples of cyclonic wind fields with different parameters. Both are shown at $t = 0$ and the cyclone's velocity is visualised by an orange arrow from the centre of the cyclone in the direction of its movement.

Some examples of the training data are presented in Appendix A.

5.3. Data augmentation

In order to synthetically add data points to our dataset, we can augment the randomly generated test cases. Typical data augmentation techniques include cropping, translation, rotation, reflection, intensity changes, and noise addition. However, since we are training the model to represent a physical process, the transformations should conserve this. Therefore, only two augmentations are applied: rotation and reflection. This selection of augmentations allows for straightforward transformations that ensure that the result satisfies the governing physical laws.

In the simulation of sea ice dynamics, the underlying physical processes are invariant under rotation. Consequently, any model provided with rotated data should produce an outcome that is correspondingly rotated. Since we are working with a square domain, rotations are restricted to $0°$, $90°$, $180°$, and $270°$. While the physics remains consistent at any rotational degree, the use of the square grid necessitates a complex re-mapping of data points back onto the grid for rotations other than multiples of $90°$, potentially introducing errors. Moreover, such rotations require the loss of data points that lie within the domain, specifically those in the corners. To avoid these complications, our model exclusively employs rotations by these orthogonal increments. This approach not only maintains the integrity of the grid, but also serves as an effective data augmentation technique, inflating the dataset size synthetically by a factor of four.

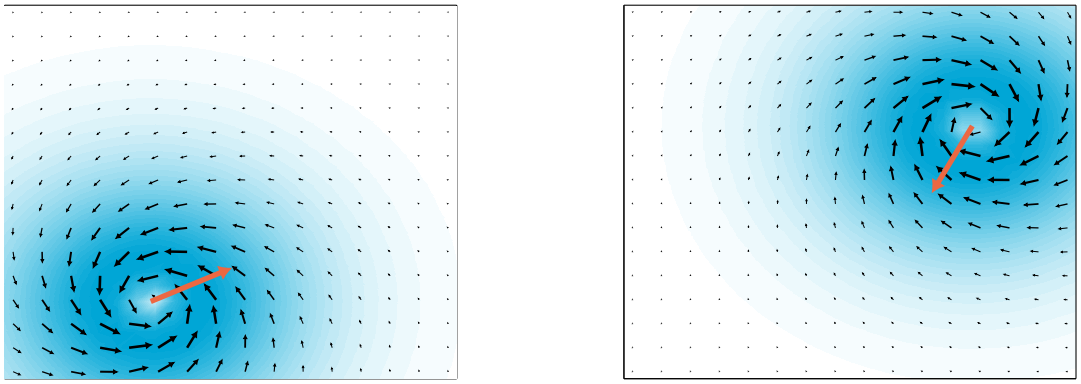
For rotation of scalar fields, the rotation operation is a mapping between indices. In the case of vector fields, the rotation operation also entails a multiplication with a rotation matrix. The rotation operations on an $N \times N$-grid are given as follows:

$$\mathsf{R}_{90°}: \begin{cases} \boldsymbol{v}_{(i,j)} \mapsto \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \boldsymbol{v}_{(N-j,i)} \\ H_{(i,j)} \mapsto H_{(N-j,i)}, \end{cases}$$

$$\mathsf{R}_{180°}: \begin{cases} \boldsymbol{v}_{(i,j)} \mapsto \begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix} \boldsymbol{v}_{(N-i,N-j)} \\ H_{(i,j)} \mapsto H_{(N-i,N-j)}, \end{cases}$$

$$\mathsf{R}_{270°}: \begin{cases} \boldsymbol{v}_{(i,j)} \mapsto \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \boldsymbol{v}_{(j,N-i)} \\ H_{(i,j)} \mapsto H_{(j,N-i)}. \end{cases}$$

The other augmentation technique we employ is reflection, which, like rotation does not alter the phys-



(a) Rotation by $90°$.

(b) Reflection over the horizontal axis.

Figure 5.6: Two examples of transformations of the cyclone in Figure 5.5a.

ical process we simulate. Reflections can be applied either horizontally or vertically. However, the effect of vertical reflection can be reproduced by a horizontal reflection followed by a rotation of $180°$. Therefore, to prevent redundancy, only horizontal reflection is used as an augmentation technique. All combinations of reflection and rotation are applied to each test case, thereby synthetically increasing the dataset size eightfold. The resulting dataset consists of $128,000$ unique, but highly correlated samples.

On an $N \times N$-grid, the horizontal reflection operation is given as follows:

$$\mathsf{F}_x : \begin{cases} \boldsymbol{v}_{(i,j)} \mapsto \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix} \boldsymbol{v}_{(N-i,j)} \\ H_{(i,j)} \mapsto H_{(N-i,j)}. \end{cases} \tag{5.8}$$

Again, for scalar fields, we only require a mapping between indices, whereas reflection of vector fields also requires an inversion along the direction of reflection. Two visual examples of these transformations are given in Figure 5.6

Other options for data augmentation to synthetically enlarge the dataset size either fail to maintain the integrity of the physical processes governing sea ice dynamics, or are impractical. For instance, while limited changes in the intensity of image data are permissible in more typical applications of convolutional neural networks, in our context, alterations of a single pixel can influence dynamics across the entire domain. This is due to the highly non-linear nature of the equations governing sea ice dynamics, where minor modifications can lead to complex outcomes. Therefore, introducing noise to the inputs cannot be offset by a simple, predetermined noise pattern on the outputs whilst remaining physically accurate. However, adding noise to the data has proven to be effective in combatting overfitting [74, 75], especially for small dataset sizes [76]. The addition of noise to the data will be discussed further in Chapter 6.

Moreover, the idea of cropping the data is central to an alternative approach, in which the neural network is applied to smaller patches of the domain. This approach is briefly touched upon in the next chapter, but in short, it was found not to work, and thus cropping the data is not included as an augmentation technique here.

In summary, the chosen data augmentation techniques of rotation and reflection effectively expand the dataset - by a factor of eight - while maintaining the integrity of the physical processes governing sea ice dynamics, enhancing model robustness without additional computational complexity.

6

Network design

In this chapter, we explore our approach to develop a neural network surrogate model that is able to predict sea ice dynamics. Fundamental to our approach is that the entire domain is predicted at once by the network. This contrasts the patch-wise approach by Margenberg et al. [6] for solving the Navier-Stokes equations. A similar patch-wise approach was unsuccessfully tried for this work. While a patch-wise approach allows for more efficient parameter sharing and thus leads to a smaller network, a network that is applied to the whole domain offers the advantage of using global information. An analogy may be made with an implicit method, in which the solution at a certain point 'knows' about the solution at other points in the same time step. In their paper, Ip et al. [22] conclude that implicit methods are the better choice for sea ice dynamics problems, allowing for larger time steps.

The goal here is to find a neural network design that is able to predict the sea ice velocity field. The main components of this design are the architecture of the network, as well as the loss function to train on. The network architecture and its data requirements are detailed in the first section. This is followed by a description of the experiment setup, which includes details about the loss functions that are compared and the Bayesian optimisation algorithm used for hyperparameter optimisation.

6.1. Network architecture

In this approach, a bottleneck structure similar to that of Eichinger et al. [39] is chosen. In their paper, from a comparison between a classical bottleneck convolutional neural network (Bottleneck CNN introduced by [7]) and a U-Net, the latter appeared to be superior. Since we are dealing with a different physical problems and network architectures can be better suited for different problems, both architectures are tested and compared in this work.

Both architectures are based on the same bottleneck structure, which is described in Section 4.5.4. The data is first passed through the encoder, which, through applications of down-convolutions, yields a vector with features that in theory describe the entire domain, the encoding. Next, the encoding is input into the decoder, which then produces the resulting sea ice velocity field. This bottleneck structure gives rise to an analogy with reduced order models, with which an approximation of the solution is found using a reduced order basis. The decoder learns this basis, while the encoder learns a mapping from the input space to an encoding for this basis. Reduced order models are typically employed because of their quicker evaluations due to their approximations.

While both approaches force information to be aggregated into a single vector (the bottleneck), the U-Net employs skip connections which allow local information in the encoder to be passed to their counterpart in the decoder. The strict analogy with reduced order models is lost, but one can now be made with multigrid methods in which a problem is solved on an increasingly coarser grid [77]. Similar to how multigrid methods address processes on different spatial scales by solving coarse approximations before refining solutions on finer grids, the skip connections facilitate representation of information at different resolutions.

Data structures

Before delving into the details of the neural network architectures, it is important to note that both architectures are designed to process the same data structures. The data is given on a 512×512 km grid with 2 km grid spacing. Since the B-grid is used, velocities are specified on the vertices of the grid, while the ice height and ice concentration are given in the cell. This means that the ice, wind and ocean velocity fields are given on a 257×257 grid. In terms of image data, this means $6 \times 257 \times 257$ pixels, since each velocity field is specified by two components, resulting in 6 channels in total. The ice height and concentration are given on a 256×256 grid and can be represented by $2 \times 256 \times 256$ pixels.

The resulting network acts as the mapping

$$\mathcal{NN} : (\boldsymbol{v}, \boldsymbol{v}_a, \boldsymbol{v}_w), (H, A) \mapsto \boldsymbol{y}, \tag{6.1}$$

where $\boldsymbol{v}$ is the initial sea ice velocity, $\boldsymbol{v}_{\text{atm}}$ is the wind velocity, $\boldsymbol{v}_{\text{ocean}}$ is the ocean velocity, H is the sea ice height, and A is the sea ice concentration. The network prediction is $\boldsymbol{y} \in \mathbb{R}^{2\times257\times257}$. Since the change in velocity between time steps typically is two orders of magnitude smaller than the velocity, the network is trained to predict the change in velocity. This allows the network to focus on modelling the small variations in velocity more precisely. To this end, the labels are computed by

$$\hat{\boldsymbol{y}}_t = \boldsymbol{v}_t - \boldsymbol{v}_{t-1}, \tag{6.2}$$

where the subscript t indicates the time step.

Moreover, all inputs are scaled between -1 and 1. To ensure that 0 values are mapped to 0, the new values are computed by

$$\bar{x} = \frac{x}{\max_{i,j,k} |x^k_{i,j}|}, \tag{6.3}$$

where i and j iterate over pixels and k over instances in the training set. The labels, however, are scaled by

$$\bar{y} = \tanh\left(10 \frac{y}{\max_{i,j,k} |y^k_{i,j}|}\right). \tag{6.4}$$

The labels contain regions in which the spatial derivatives become quite large. Without an additional transformation, these regions will dominate the dataset. Moreover, beyond a certain magnitude, the precise velocity is less important for the creation of linear kinematic features (LKFs). Therefore, the largest outliers are scaled back using the hyperbolic tangent function. Only 0.006% of pixels in the training dataset have values large enough to be severely affected by this transformation.[1] This amounts to, on average, 4 pixels in each image.

Classic Bottleneck

In their work, Eichinger et al. [39] use the architecture from Guo et al. [7], which employs two consecutive down-convolutional layers to reduce the $1 \times 256 \times 128$ image to $512 \times 4 \times 4$, after which a fully connected layer is used which yields a vector with 1024 elements. Next, four deconvolutional layers are used as up-convolutions to get back to an image of shape $2 \times 256 \times 128$. This architecture serves as the foundation for the bottleneck design in this work.

Due to the different grid sizes of the input data, the input layer of the network is split into two separate paths: one for the velocities and another for the ice height and concentration. The outputs of these paths are summed and passed through an activation function, forming a single hidden layer. Since the output channels of a convolutional layer consists of the sum of convolutions of inputs and kernels, summing the results of two separate layers yields the behaviour of a single layer, as if the inputs live on the same grid size. For the remainder of the network, the bottleneck structure from Guo et al. [7] is followed more closely.

One other alteration is made: an extra down-convolutional layer is added to transform the $512\times4\times4$ result to a $1024\times1\times1$ image, before the fully connected linear layer is applied. Furthermore, except for the first up-convolution, which is a deconvolutional layer, the other up-convolutions are combinations of upsampling and convolutions as proposed by Odena et al. [72].

[1] The main effect of this transformation is that the influence of errors in the largest pixel values are devalued. The use of the hyperbolic tangent function as the output activation mitigates the loss of precision.

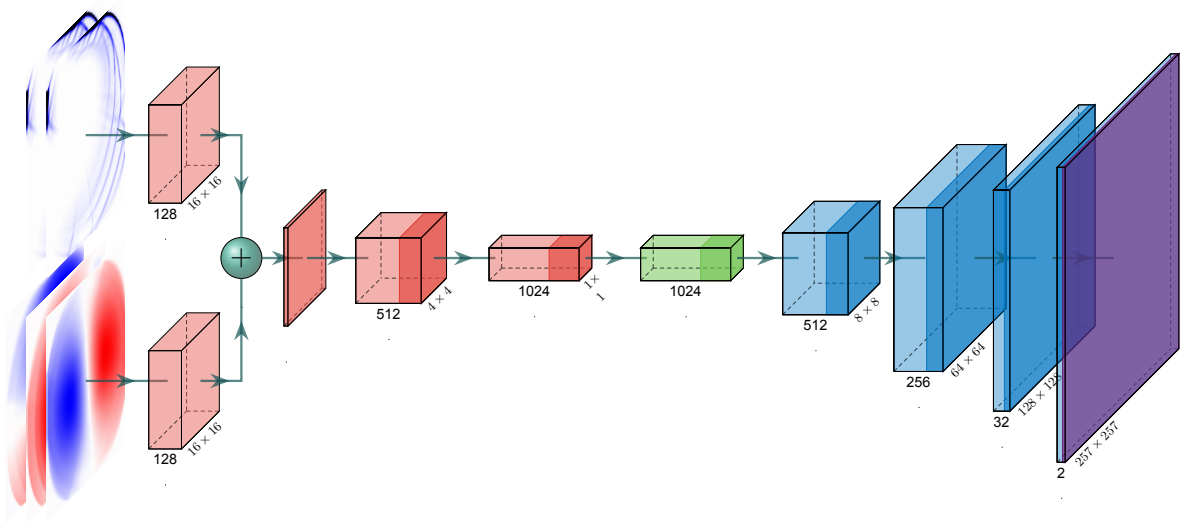


Figure 6.1: Architecture of the bottleneck network for sea ice dynamics. Visualisation using [79]. The network has 46 million trainable parameters.

After each layer, the Gaussian Error Linear Unit (GELU) is used as the activation function in combination with batch normalisation, which has been found to yield performance improvements for convolutional neural networks [78]. After the final layer, however, the hyperbolic tangent is used as activation to ensure that the image space of the network is $[-1, 1]^{2\times257\times257}$. The full bottleneck network architecture is presented in Figure 6.1.

U-Net

The other network architecture studied in this work is the U-Net, first introduced by Ronneberger et al. [80]. The two main differences with the bottleneck architecture of Guo et al. [7] are the use of skip connections and of additional size-conserving convolutional layers following up- and down-convolutional layers.

The U-Net implementation used in this work follows the design by Eichinger et al. [39] with some modifications. Again, the input layer is composed of two different paths: one for the velocities and another for the ice height and concentration. Including the input layer, the encoder of this bottleneck structure consists of six down-convolutional layers. Between each pair of down-convolutions, except for the last, two convolutional layers are added that conserve the image size. The last layers yields a vector with 1024 elements, the encoding.

The second half of the network, the decoder, mirrors the encoder. It consists of six up-convolutional layers. Again, with the exception of the first, every pair of up-convolutional layers is separated by two convolutional layers that conserve the image size. Similar to the classical bottleneck architecture, the first up-convolution in this U-Net design is a deconvolutional layer, while the others consist of upsampling and a convolution as proposed by Odena et al. [72]. In contrast to the encoder, two convolutional layers are added to the end of the decoder, the first of which conserves the image size, while the second reduces the number of channels to two.

Additionally, there are four skip connections in the design. The outputs of layers in the encoder on four different levels are concatenated to the outputs of up-convolutions in the decoder. In this way, a strict analogy with reduced order models is lost, since not all information is contained in the encoding. However, local information, on the four different levels, is passed through the skip connections, which allows for strictly global information to be stored in the encoding, while more local processes are represented on finer grids.

Lastly, the GELU is used as the activation function in combination with batch normalisation. Again, the hyperbolic tangent function is used as the activation function of the output layer to guarantee an image space of $[-1, 1]^{2\times257\times257}$ A visualisation of the U-Net architecture is given in Figure 6.2.

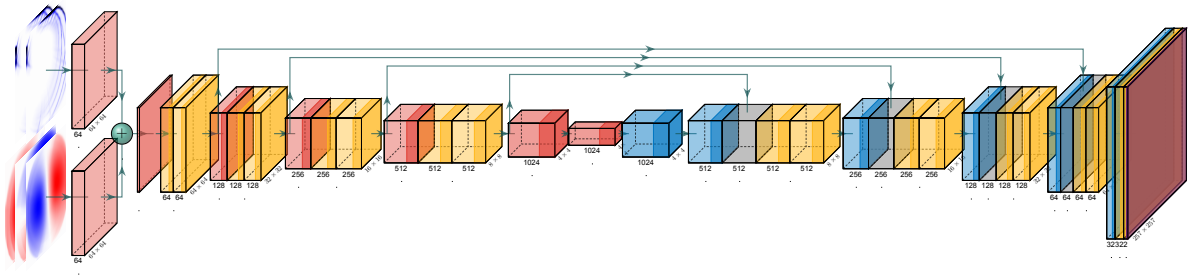


Figure 6.2: Architecture of the U-Net network for sea ice dynamics. Visualisation using [79]. The network has 67 million trainable parameters.

6.2. Loss functions

To find an optimal design for the network, not only the two different architectures detailed above are compared, but also the effect of different loss functions is studied. Since different loss functions emphasise different aspects of the data, their effect on the model performance must be studied. Four different loss functions are used to train the neural networks: the mean squared error (MSE) of the velocities, the strain rate error (SRE), the weighted sum of the MSE and the mean relative error (MSE+MRE) and the weighted sum of the MSE and the SRE (MSE+SRE). These loss functions were introduced in Section 4.2.1:

$$\text{MSE}(\boldsymbol{y}_i, \hat{\boldsymbol{y}}_i) = \frac{1}{N}\sum_{i=1}^{N} \|\boldsymbol{y}_i - \hat{\boldsymbol{y}}_i\|_2^2\,, \tag{4.3}$$

$$\text{MRE}(\boldsymbol{y}_i, \hat{\boldsymbol{y}}_i) = \frac{1}{N}\sum_{i=1}^{N} \frac{\|\boldsymbol{y}_i - \hat{\boldsymbol{y}}_i\|_2}{\|\hat{\boldsymbol{y}}_i\|_2}, \tag{4.5}$$

$$\text{SRE}(\boldsymbol{y}_i, \hat{\boldsymbol{y}}_i) = \frac{1}{4N}\sum_{i=1}^{N} \left\|\nabla(\boldsymbol{y}_i - \hat{\boldsymbol{y}}_i) + \nabla(\boldsymbol{y}_i - \hat{\boldsymbol{y}}_i)^T\right\|_2^2 \tag{4.6}$$

where $\hat{\boldsymbol{y}}_i$ is the label corresponding to the instance i and $\boldsymbol{y}_i$ is the prediction by the network of instance i. The strain rate error is based on the strain rate $\dot{\epsilon}$ introduced in (2.5). Since the network's prediction is given pixelwise on a grid, the gradient must be approximated using a finite difference method. The central difference method is applied to obtain the gradient in the cell's centre using the velocities on the cell's vertices.

The performance of these loss functions, however, must be compared using a relatively independent metric. If, for example, their performance was compared using the mean squared error (MSE), the model trained using this metric would have an advantage.

Since we are interested in the formation of Linear Kinematic Features (LKFs), the absolute error in the shear deformation given in (2.12) is used as the main comparative metric. The error in the sea ice concentration and the MSE of the velocities is also studied, to get a full understanding of the behaviour.

The labels and network outputs remain scaled to the $[-1, 1]$ domain for the computation of the MSE, SRE and MRE. Specifically for the the SRE and MRE, it was found that using the unscaled labels and the rescaled network output caused instability in the training process (e.g., exploding gradients), occasionally resulting in computational errors during training. Most likely, this is caused by the inverse of the hyperbolic tangent used in the rescaling. For the error in the shear and in the sea ice concentration, however, the unscaled values are used to conserve their physical meaning.

6.3. Training phase

For a fair evaluation of the performance of the model, a part of the dataset is reserved to ensure an unbiased evaluation of the model. As described in Section 5.1, the goal of this work is to train a model that is able to perform well on the benchmark problem used in the comparative work by Mehlmann et al. [49]. This data pertaining to this benchmark problem is, therefore, used as the test dataset. Moreover, an additional set of data is needed to allow unbiased evaluation of model performance when comparing different models. This validation requires a subset of the data that is independent of both the training

Table 6.1: List of hyperparameter ranges and values for hyperparameters that are respectively optimised and fixed.

Hyperparameter	Value
b	8
β_1	0.67
β_2	0.95
α	$\in [10^{-3}, 10^{2}]$
ν	$\in [10^{-3}, -0.3]$
λ	$\in [10^{-4}, 10^{-1}]$
w_{SRE}	$\in [10^{-1}, 10^{3}]$
w_{MRE}	$\in [10^{-6}, 10^{-2}]$
ε_{MRE}	$\in [10^{-6}, 10^{-1}]$

set and the test set. To this end, the training dataset generated as described in Section 5.2 is divided such that 80% is used for training and 20% is used for validation. Finally, we are left with three datasets: the training, validation and test datasets.

As described in Section 5.3, noise may be added to the input data to combat overfitting. However, as mentioned, the addition of noise on the inputs, without any alternations to the outputs, makes the instance unphysical. Since the strain rates are important for sea ice dynamics modelling, noise on the inputs interferes significantly with the predictions. To mitigate this slightly, multiplicative noise is chosen:

$$x = x \cdot (1 + \mathcal{N}(0, \nu^2)), \tag{6.5}$$

where ν is the standard deviation of the normal distribution (i.e. noise level) and x represents any input pixel. In this way, the distortion is relative to the value of the input, which limits the effect of the noise on the strain rates.

Furthermore, the Adam optimiser is chosen and has momentum parameters β_1 and β_2. Additionally, gradient clipping is performed where the gradients are constrained to $[-1, 1]$.

Moreover, a stopping criterion based on the improvement of the validation loss is implemented. The criterion is met if the validation loss has not improved in the last 100 epochs. To ensure that overfitting has really occurred, training is only stopped if, additionally, the validation loss is larger than the training loss. Furthermore, to give the network sufficient time and to ensure that training is not terminated too soon, the condition that the network has trained for at least 300 epochs is applied as well. To summarise, the three conditions for early stopping:

1. the validation loss has not improved in the last 100 epochs, and
2. the validation loss is larger than the training loss, and
3. the network has trained for at least 300 epochs.

6.4. Hyperparameter optimisation

An important part of training is the hyperparameter optimisation. These are the parameters that describe the training process and include the regularisation parameter α, the batch size b, the noise level added to the data ν, the learning rate λ, and the momentum parameters (β_1, β_2). Furthermore, when training using the sum of two loss functions, the weight of the second loss function w is considered a hyperparameter. Lastly, the MRE has a small value ε in the denominator originally just used to prevent division by zero, but this parameter can be used to tune the behaviour of the loss function. To reduce the search space of the hyperparameter optimisation, the batch size and momentum are not included in the search. The other variables, however, are included in the optimisation. The values and ranges of the respectively constant and optimised hyperparameters are listed in Table 6.1.

For each combination of architecture and loss function, a Bayesian optimisation is performed to find the optimal hyperparameters for the pair. The hyperparameter optimisation is performed for each combination separately since each pair might respond differently to the hyperparameter values. Bayesian optimisation, while more difficult to implement than a grid search, is a more efficient search method. To illustrate, a grid search with four candidates for each of the five parameters would require a total of 1024 models to be trained, compared to the 20 to 40 candidates we will test through Bayesian optimisation.

Bayesian optimisation, however, uses previous training results to find a better hyperparameter set for the next evaluation. This approach was first introduced by Mockus et al. [81] and is typically used to find the extremum of a function that is costly to evaluate, which in our case is the mapping of a given set of hyperparameters to the shear error on the validation dataset, which requires training a neural network.

In short, the algorithm works by constructing a prior probability distribution of the approximated function based on a set of data points. Initially, this set can be obtained by randomly selecting points from the parameter space and training a network for those values. In this work, the prior distribution is then constructed using a Gaussian process with a radial basis function (RBF) kernel $K_\gamma(\cdot,\cdot)$. The covariance of two sets of hyperparameters $\boldsymbol{h}$ and $\boldsymbol{h}'$ is then assumed

$$K_\gamma(\boldsymbol{h},\boldsymbol{h}') = \exp\left(-\gamma\|\log_{10}(\boldsymbol{h}) - \log_{10}(\boldsymbol{h}')\|_2^2\right), \tag{6.6}$$

where γ is the scaling parameter. Hyperparameters that lie close to each other are, therefore, expected to give similar results. Conversely, in a region where no known values are found, the uncertainty in the prior is larger. In this way, a probability distribution is built, and confidence bounds can be constructed for each point in the hyperparameter space, between which the corresponding validation shear error value is expected to be found.

Subsequently, the next set of hyperparameters is determined using an acquisition function, which is chosen as the lower confidence bound of the prior, since it incorporates both the knowledge of previously evaluated points, as well as the uncertainty of new points. After training the network with the new set of hyperparameters, the result is used to update the prior and the acquisition function, after which a next point can be chosen, and so on.

Since the order of magnitude of the hyperparameters is often the most important, the optimisation is performed using the logarithm of their values. Moreover, the first 10 points in the hyperparameter space are drawn randomly, after which the acquisition function is used to find the next set of hyperparameters. Furthermore, to increase efficiency, at most eight models are trained in parallel, depending on the availability of the high performance computer on which the models are trained.

This parallelism introduces a problem in the selection of the next hyperparameter combination: without any alternations to the optimisation algorithm, many of the models will be trained with the same set of hyperparameters. To illustrate, suppose we have a set of hyperparameter combinations $\boldsymbol{h}_1,\ldots,\boldsymbol{h}_n$ with corresponding validation shear error values $l_1,\ldots,l_n$. The optimisation algorithm yields, after computing and minimising the acquisition function, a next combination of hyperparameters $\boldsymbol{h}'$, which is used to train a new model. During the training of this model, the computational capacity allows for another instance of the network to be trained, so the optimisation algorithm is run again, and, since the process is deterministic, it yields the same $\boldsymbol{h}'$.

To fully use the efficiency that the parallelism offers, this second Bayesian optimisation should yield a different set of parameters. Therefore, a new prior is constructed for which $\boldsymbol{h}'$ and the expected validation loss according to the previous prior $\mathbb{E}(\boldsymbol{h}')$ have been added to the data. Thus, an updated acquisition function is used to obtain different candidate hyperparameters.

6.5. Implementation

The neural networks presented in this chapter were implemented in Python [82] using the machine learning framework Pytorch [83]. The code is made publicly available on Github [84]. DelftBlue supercomputer at Delft High-Performance Computing Center [85], which employs NVIDIA A100 GPUs with 80 GB video RAM each, was used for training the neural networks. As a student, at most eight jobs can be run simultaneously which can run for at most 24 hours. However, most networks could be trained within 10 hours before satisfying the early stopping criterion.

7 Results

This chapter presents the results from the development of a neural network surrogate model aimed at predicting sea ice dynamics. We evaluate the performance of the Bottleneck and U-Net architectures when trained with various loss functions. Initially, the results of the hyperparameter optimisation are discussed, identifying the most effective model configuration for each architecture-loss pair in Table 7.1. This optimal model is then applied to the benchmark problem, for both one-step-ahead predictions and time series generation. Additionally, a qualitative analysis is conducted of to distinguish the performance nuances between the architectures and loss functions. This chapter concludes with an investigation of the generalisation behaviour.

7.1. Hyperparameter optimisation

The results of the hyperparameter optimisation are presented in Table 7.1. For each combination of loss function (MSE, SRE, MSE+SRE, MSE+MRE) and architecture (Bottleneck, U-Net), the hyperparameter combination resulting in the lowest shear error is shown. Additionally, the shear error, mean squarred error (MSE), strain rate error (SRE), mean relative error (MRE), mean concentration error (MCE) are presented in Table 7.2. For comparison, the error values corresponding to a baseline prediction consisting of only zeros is also presented.

To enable a fair comparison of MRE values across all models, we fix the parameter ε used in the MRE computation (see (4.5)) to 2.3×10^{-3}. This values corresponds to the optimal ε found during the Bayesian optimisation for the U-Net architecture trained with the MSE+MRE loss function. For models not trained with the MRE loss (and therefore without a tuned ε), we use this fixed value in the MRE calculation. It should be noted that the Bottleneck architecture trained with the MSE+MRE loss has its own optimal ε, which would yield a different MRE value of 24.9%. However, to maintain consistency in our comparisons, we report all MRE values using $\varepsilon = 2.3 \times 10^{-3}$.

Comparing the shear error across the different architecture-loss combinations, we see that the U-Net

Table 7.1: Optimal hyperparameters as found by the hyperparameter optimisation for each combination of architecture (Bottleneck, U-Net) and loss function (MSE, SRE, MSE+SRE, MSE+MRE).

Architecture	Loss function	α	ν	λ	w	ε
Bottleneck	MSE	5.7×10^{1}	3.0×10^{-1}	1.6×10^{-3}	-	-
	SRE	1.4	2.4×10^{-3}	1.7×10^{-4}	-	-
	MSE+SRE	5.8×10^{-3}	1.6×10^{-2}	1.6×10^{-3}	2.6	-
	MSE+MRE	8.6×10^{-5}	1.3×10^{-2}	2.2×10^{-4}	9.2×10^{-3}	1.2×10^{-3}
U-Net	MSE	1.4×10^{-2}	8.0×10^{-3}	3.7×10^{-4}	-	-
	SRE	2.7×10^{-3}	3.4×10^{-3}	2.4×10^{-4}	-	-
	MSE+SRE	7.0×10^{-2}	1.2×10^{-3}	1.0×10^{-4}	2.5×10^{0}	-
	MSE+MRE	2.8×10^{-3}	1.1×10^{-2}	4.6×10^{-3}	8.7×10^{-5}	2.3×10^{-3}

Table 7.2: Error rates on the validation set for the best hyperparameter combination for each pair of architecture (Bottleneck, U-Net) and loss function (MSE, SRE, MSE+SRE, MSE+MRE). The shear error, MSE, SRE, MRE and MCE on the validation dataset are shown. The best error values in each column are marked in bold. The errors corresponding to a baseline prediction of only zeros are also shown.

Architecture	Loss function	shear error $\times 10^{-8}$	MSE $\times 10^{-4}$	SRE $\times 10^{-4}$	MRE	MCE $\times 10^{-8}$
Baseline		15.3	49.4	46.0	34.1%	33.1
Bottleneck	MSE	15.3	49.2	31.3	45.3%	51.2
	SRE	11.7	44.8	14.0	75.8%	27.4
	MSE+SRE	15.0	30.0	30.9	39.7%	46.2
	MSE+MRE	15.2	26.7	31.0	**8.39%**	51.5
U-Net	MSE	7.87	14.3	5.44	28.3%	11.1
	SRE	7.20	41.2	**3.19**	94.6%	6.83
	MSE+SRE	**6.41**	**3.63**	3.32	24.0%	**6.32**
	MSE+MRE	8.77	12.8	6.49	28.7%	13.8

architecture, trained with the MSE+SRE loss function, achieved the lowest shear error, MSE and MCE. Since this model resulted in the lowest shear error, we choose it for further comparison in Section 7.3. Notably, the U-Net consistently outperformed the Bottleneck architecture on not only the shear error, but also the MSE, SRE and MCE. The SRE was lowest for the U-Net only trained with the SRE, while the Bottleneck architecture trained with the MSE+MRE achieved the lowest relative error on the validation set. An explanation for this can be found in the fact that the weight of the MRE was much larger for the best Bottleneck model than for the best U-Net model.

Moreover, not only did the U-Net outperform the Bottleneck architecture, but trained with only the MSE, the Bottleneck architecture with the lowest shear error was even severely underfit, producing only values close to zero for all test cases. It must be noted that other hyperparameter combinations lead to lower MSE values on the validation loss, but these yielded higher errors in the shear and thus are not shown.

Furthermore, the MSE+SRE loss resulted in the lowest shear loss for both architectures. It appears that the combination of the two loss functions is the decisive factor. Even though the SRE U-Net model had the lowest strain rate error, its mean squared error was relatively large. This is also visible in the visualisations presented and discussen in Section 7.2. Interestingly, the MSE+SRE model even had a lower mean squared error than the model trained with only the MSE, indicating that the additional SRE helps capture sea ice dynamics better, regardless of the chosen loss function.

To better understand the optimised hyperparameters, we now examine the Gaussian process model from the Bayesian optimisation for the MSE+SRE U-Net. To assess the effect of a single hyperparameter, we fix the other three at their optimal values as determined by the optimisation algorithm, and plot the regression curve for the selected hyperparameter. In addition to the regression curve, we also show individual hyperparameter candidates and their corresponding shear error in a scatter plot. To include a sense of closeness in the other three dimensions, we adjust the transparency of the points based on their proximity to the fixed hyperparameter values. Here, the radial basis function, also used for the Bayesian optimisation, is employed as a distance measure. Points closer to the fixed values are displayed with higher opacity, indicating a greater contribution to the regression at that point, while points further away are rendered more transparently.

Figure 7.1 shows the expected shear error for a U-Net trained with the MSE+SRE loss function versus the four hyperparameters. From these plots, we see that there is an optimum for α and w, while ν and λ resemble increasing functions. The optimum in the choice α shows the sensitivity of the network to overfitting. Some penalisation of the size of weights in the network is needed to prevent overfitting, while a penalisation that is too large results in underfitting. On the other hand, lower values of ν resulted in a lower error, suggesting that regularisation through added noise to the inputs is not effective. Since the noise added to the inputs is multiplicative, even lower noise levels would have minimal impact on the error, indicating that decreasing ν further would not enhance the model's performance. This is supported by the minimal difference between a noise level of 10^{-2} and 10^{-3}.

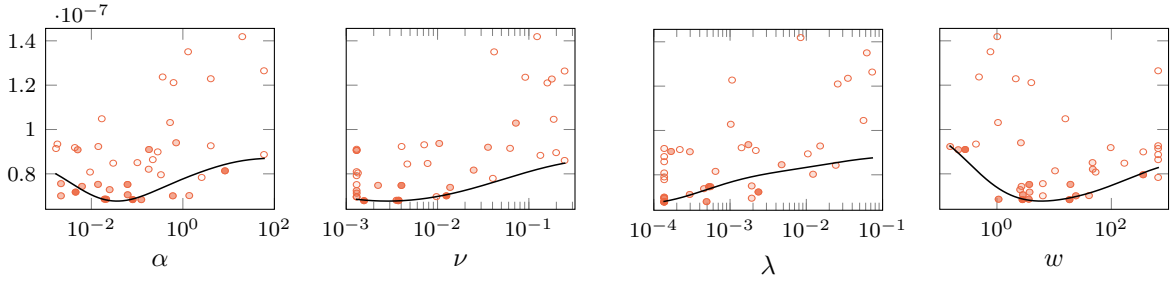


Figure 7.1: Plot of the shear error versus the four hyperparameters for the MSE+SRE U-Net (α, ν, λ, w). The Gaussian process regression from the Bayesian optimisation is shown with one parameter varying and the other hyperparameters fixed at their optimal values. The scatter plot shows the error for each individual hyperparameter combination, where the opacity reflects their proximity to the fixed hyperparameters.

Furthermore, the plots of the learning rate λ and SRE weight w also demonstrate notable trends in the performance. Specifically, models trained with lower learning rates consistently achieved lower shear errors. This observation, combined with the fact that also the other SRE and MSE+SRE models were found to be performing best with a low learning rate, indicates that the spatial derivative necessitates smaller steps in the optimisation algorithm. Additionally, the shear error reaches its minimum when the weights assigned to the MSE and SRE are of the same order of magnitude. Together with the fact that the MSE+SRE model outperformed the models trained with only one of the loss functions, we see that the combination of the two is valuable.

Even though the search domain for the explicit regularisation parameter α was initially set to $[10^{-3}, 10^{2}]$, this was expanded down to 10^{-5} for the MSE+MRE Bottleneck models since the Bayesian optimisation algorithm consistently yielded candidate hyperparameter combinations on the edge of the interval. This was the only case where the initial search interval was expanded during the search.

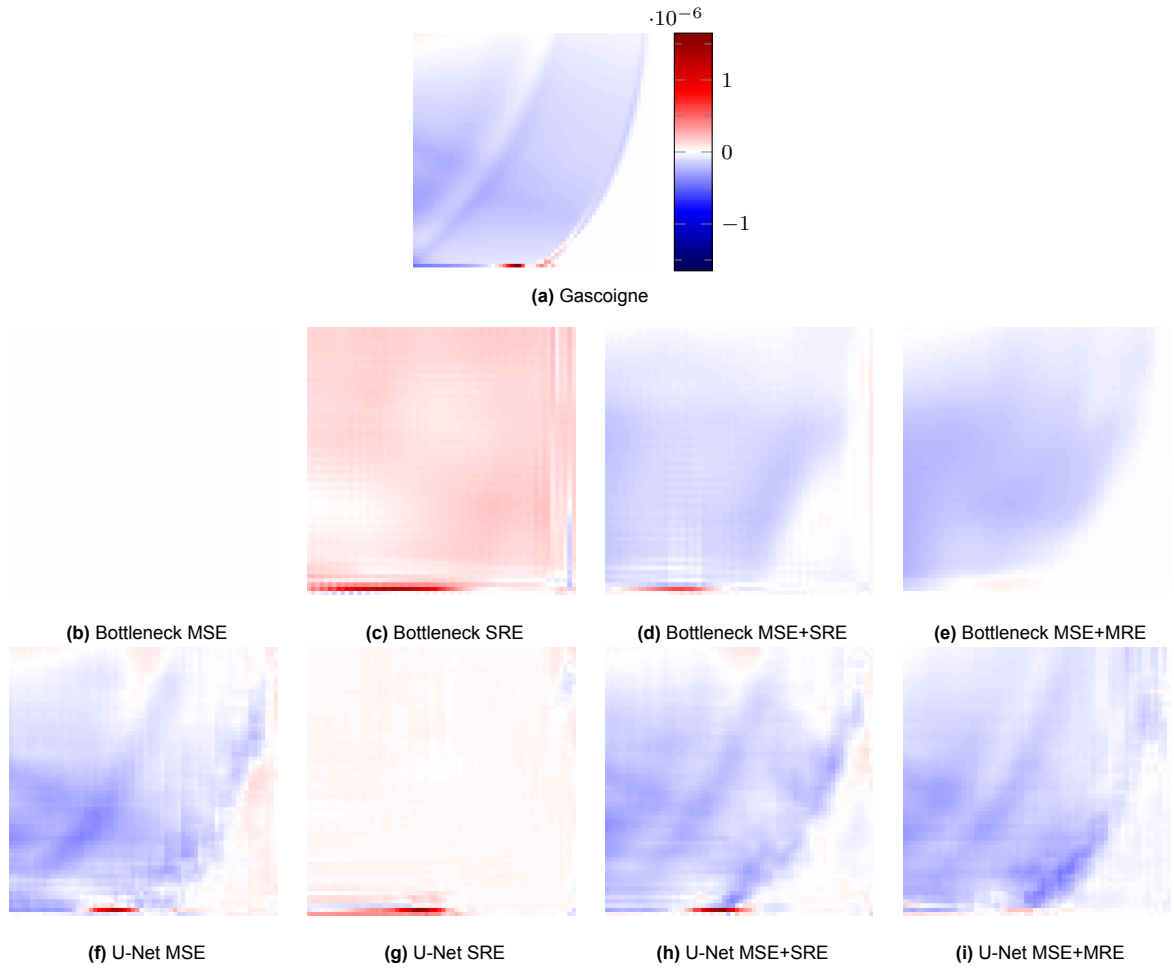
7.2. Qualitative comparison

This section provides a qualitative comparison of the eight optimised models derived from the combinations of the two architectures and four loss functions. First, it is investigated how the choice of loss function influences the model's output, exploring how different loss functions emphasise various features in the image data relevant to sea ice dynamics. Second, we examine the differences between the two architectures, analysing how the structural distinctions between the Bottleneck and U-Net architectures affect the models' ability to capture and represent key characteristics.

For this visual comparison, the models' outputs are presented in Figure 7.2 for one of the randomly generated test cases. The velocity's x-component as computed by each of the optimised networks as well as the Gascoigne computation is shown. Specifically, we have zoomed in on the bottom right corner of the results to get a close-up view of the results. The shown area is 128×128 km, so it is one sixteenth of the whole domain.

We have chosen this specific region as it contains low, high and very high rates of deformation. At the bottom boundary of the domain, we see a small region where the velocity changes rapidly, indicating a region with very high strain rates. More moderate strain rates can be observed in two diagonal linear across the visualised domain. Linear kinematic features are present along these lines. Lastly, areas of low strain can be observed in the rest of the image, specifically in the bottom right where the ice velocity is also close to zero, and more towards the centre where the velocity is relatively constant. We will examine the influence of the loss functions and architectures on networks' performance on these different regions.

The first thing we notice is that while some of the models produce relatively good outputs, others exhibit significant deficiencies, failing to capture patterns in the data and resulting in poor predictions. For instance, the Bottleneck model trained with only the MSE loss function fails to capture any patterns in the data and shows signs of being severely underfit. Not only is the velocity field produced by this model nearly zero everywhere, but its error metrics on the validation set, presented in Table 7.2, reflect significant deviations from the true values, highlighting the model's severe underfitting.



(a) Gascoigne

(b) Bottleneck MSE **(c)** Bottleneck SRE **(d)** Bottleneck MSE+SRE **(e)** Bottleneck MSE+MRE

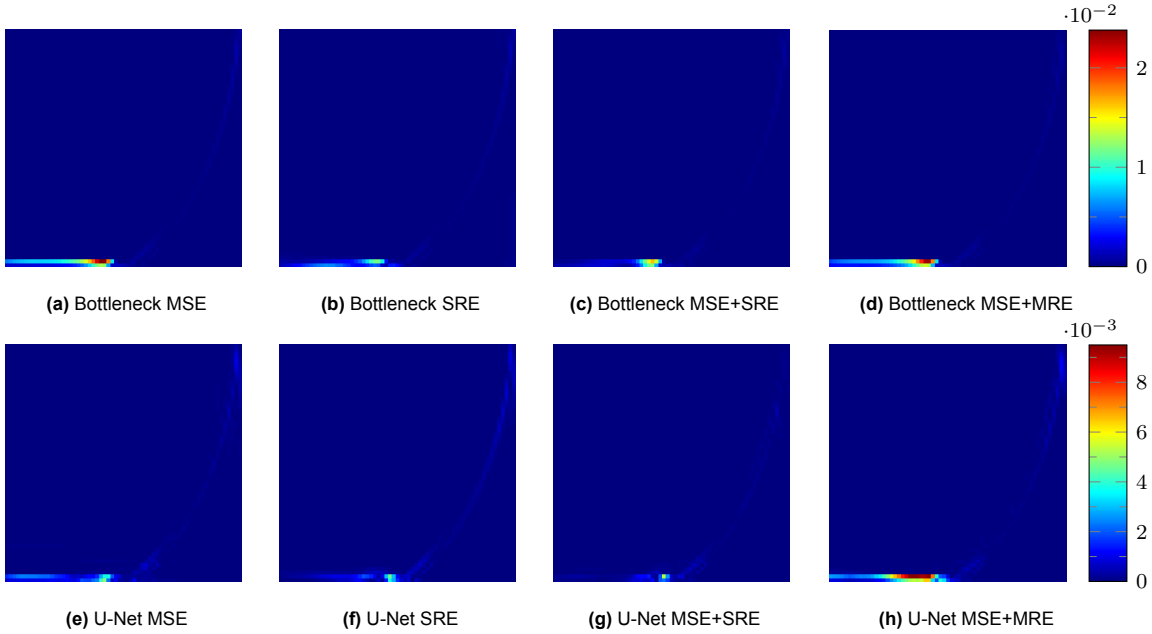
(f) U-Net MSE **(g)** U-Net SRE **(h)** U-Net MSE+SRE **(i)** U-Net MSE+MRE

Figure 7.2: Bottom right corner of both the Gascoigne simulation result and the predictions by each optimised model of the test case in Figure A.1. The x-component of the change in velocity is presented.

The models trained exclusively with the strain rate error (SRE) loss function exhibit outputs that do not resemble the ground truth computed by Gascoigne. Interestingly, however, the strain rate errors presented in Table 7.2 are comparatively low for these models, with the SRE U-Net even outperforming all other models on this metric. This is supported by the visualisation of the SRE presented in Figure 7.3. We see that the models trained with the SRE loss attain good results in the area with large strain rates. This suggests that the SRE loss function effectively captures the strain rates present in the data by penalising errors related to the spatial derivatives of the output. However, the mean squared errors (MSE) of the SRE models are significantly higher than those of the other models, providing additional evidence for the lack of visual resemblance observed in Figure 7.2. This indicates that while the SRE loss function emphasises performance on regions with very high strain rates, it neglects the overall velocity field and areas with a more moderate strain rate, leading to outputs that diverge substantially from the actual data.

In contrast, the models utilising the mean squared error (MSE) loss function demonstrate good results, particularly with the U-Net architecture. The two diagonal lines corresponding to the linear kinematic features are discernible in all three plots, indicating that the MSE loss function effectively captures the prominent features of the velocity field. This outcome is expected, as the MSE penalises larger deviations more heavily, encouraging the network to minimise significant errors across the entire output. The MSE effectively shapes the loss landscape to guide the model towards solutions that balance performance over both high and low deformation regions.

The addition of the mean relative error (MRE) to the loss function introduces a bias towards regions with low velocity, as these areas are weighted more heavily due to the denominator in the MRE computation. This effect is visible in the triangular region in the bottom right of the visualisations, where



(a) Bottleneck MSE **(b)** Bottleneck SRE **(c)** Bottleneck MSE+SRE **(d)** Bottleneck MSE+MRE

(e) U-Net MSE **(f)** U-Net SRE **(g)** U-Net MSE+SRE **(h)** U-Net MSE+MRE

Figure 7.3: Strain rate error of the predictions by each optimised model of the test case in Figure A.1, also shown in Figure 7.2. Note the different colorbar scalings.

the ice velocity is low: models trained with the MSE+MRE loss function perform best here. While this enhances the model's accuracy in low-strain areas, it may lead to underestimation of larger velocity values elsewhere, as the network prioritises minimising errors where velocities are small.

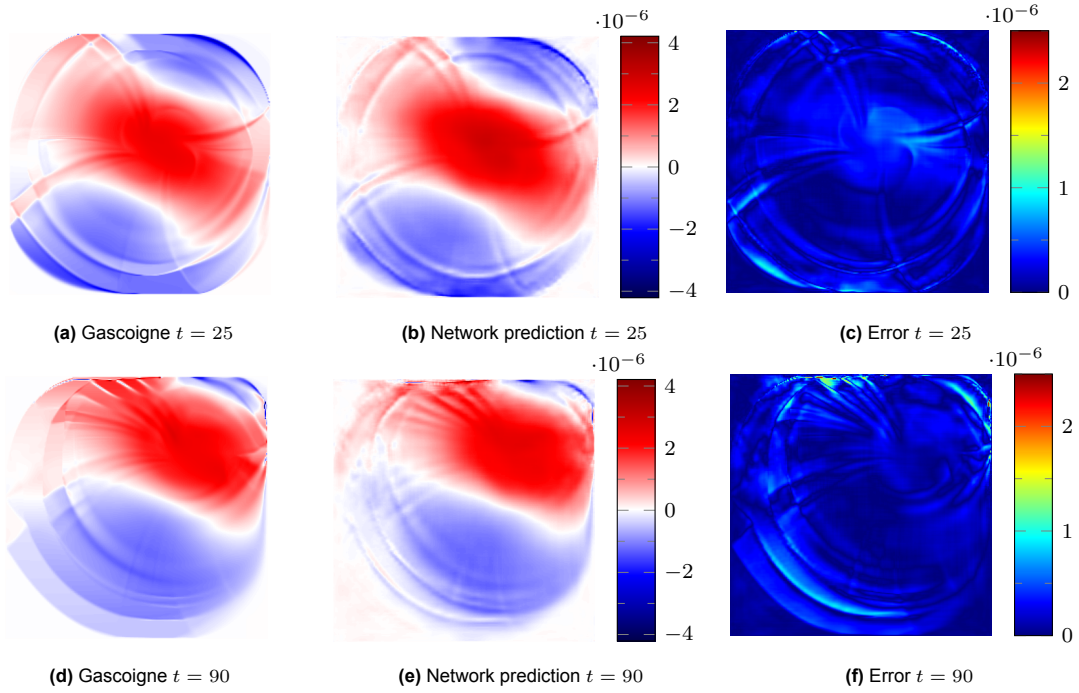
Combining the MSE and SRE loss functions results in the best overall performance for both the Bottleneck and U-Net architectures, as evidenced by the evaluation metrics in Table 7.2. The addition of the SRE term enhances the model's accuracy in areas with higher strain rates, effectively capturing the critical features associated with sea ice dynamics. This improvement is achieved while maintaining the good overall results provided by the MSE loss function. The inclusion of SRE appears to shape the loss landscape in a way that guides the network toward solutions that better represent the complex patterns of sea ice deformation, resulting in both the lowest MSE and shear error among all models. Thus, the MSE+SRE loss function effectively balances the emphasis on overall accuracy and the detailed representation of high-strain regions, leading to superior predictive performance.

In addition to the variations introduced by different loss functions, distinct visual differences are also apparent between the outputs of the two network architectures. The U-Net architecture consistently demonstrates superior capability in representing finer details within the velocity field, owing to its skip connections which facilitate the preservation and accurate reconstruction of small-scale features. In contrast, the classic Bottleneck architecture tends to produce smoother results that emphasize global patterns, thereby neglecting the intricate details necessary for accurately capturing regions with significant strain rates. This limitation in representing small-scale features by the Bottleneck architecture contributes to the higher shear error rates observed, as shear deformation predominantly occurs at the scale of a few grid cells. Consequently, the U-Net's architectural design proves more effective in modeling the complex, localized dynamics of sea ice deformation, resulting in more accurate and detailed predictions.

7.3. Performance on benchmark problem

Previously, the analyses have been made on the validation dataset, which is a randomly sampled subset of the generated dataset used for training. Since the goal of this work is to find a model that is able to predict linear kinematic features (LKFs) on the benchmark problem, we now apply the best performing model according to the shear loss to the benchmark problem. First, a comparison will be made for the one-step-ahead predictions, after which the network is applied recurrently, feeding its own outputs back as inputs to generate a time series of predictions. This evaluation enables us to assess

the model's ability to predict linear kinematic features on the benchmark problem and to understand how errors propagate over time for this model.



(a) Gascoigne $t = 25$ **(b)** Network prediction $t = 25$ **(c)** Error $t = 25$

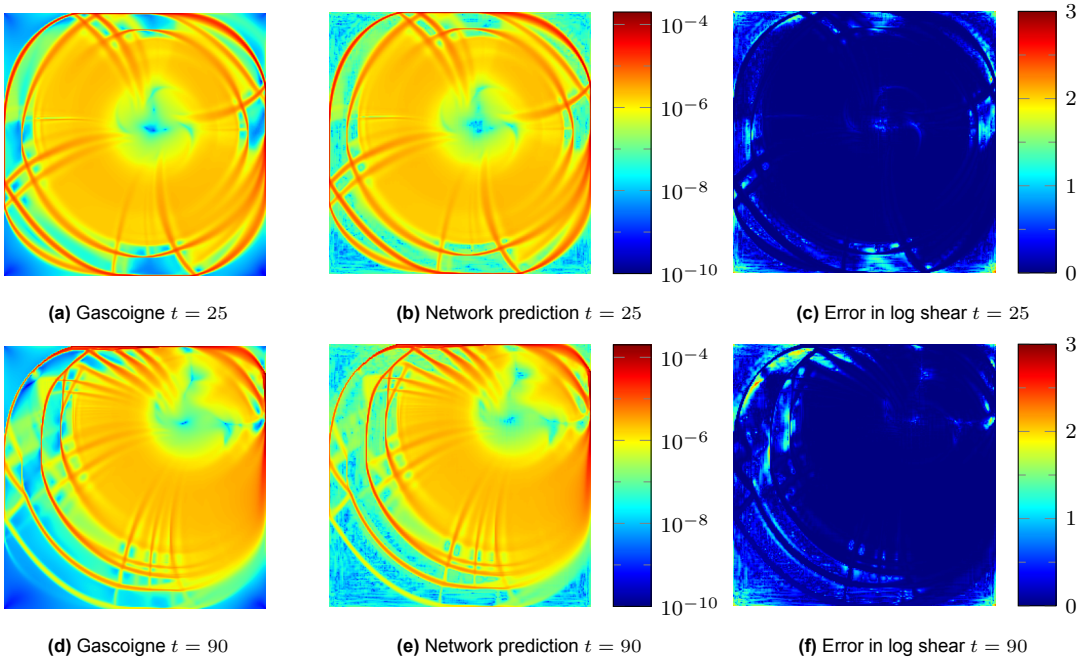
(d) Gascoigne $t = 90$ **(e)** Network prediction $t = 90$ **(f)** Error $t = 90$

Figure 7.4: Gascoigne simulation result, network prediction and error for two time steps of the benchmark problem. Only the x-component is shown. The errors are the absolute errors.

To illustrate the model's performance in velocity prediction, Figure 7.4 presents a comparison between the Gascoigne computation, which is used as ground truth, the model's output, and the corresponding error. The model's one-step-ahead prediction for time steps 25 and 90 are visualised.

From the figures, it is evident that the network's predictions lack the sharpness observed in the Gascoigne simulation that is used as ground truth, especially in areas with abrupt changes in velocity. The error plots reinforce this observation, revealing that the largest errors are concentrated around the linear kinematic features. This indicated that while the model can capture the general patterns of the velocity field, it struggles to accurately represent the fine-scale details associated with LKFs.

Furthermore, Figure 7.5 illustrates the shear deformation computed from the velocity fields of both the Gascoigne ground truth and the network's predictions for the same two time steps. The plots employ a logarithmic color scale to highlight variations across several orders of magnitude. Additionally, error plots showing the absolute difference between the logarithms of the shear deformation are presented. Consistent with previous findings, these error plots show that the errors are concentrated around the LKFs, indicating that the model struggles to accurately capture shear deformation in these critical regions. Since the network predicts only the change in velocity between time steps, its influence on the shear is limited, resulting in shear patterns that closely resemble the ground truth. Therefore, we must critically assess these images and remain aware that issues in sharpness can accumulate over time.



(a) Gascoigne $t = 25$ **(b)** Network prediction $t = 25$ **(c)** Error in log shear $t = 25$

(d) Gascoigne $t = 90$ **(e)** Network prediction $t = 90$ **(f)** Error in log shear $t = 90$

Figure 7.5: Shear deformation of the Gascoigne simulation result and network prediction for two time steps of the benchmark problem. The shear is shown using a logarithmic colour scaling. The errors shown are the absolute value of the difference between the logarithms of the shear deformation.

To asses the long-term predictive capabilities of the model, we apply the MSE+SRE U-Net recurrently, feeding its own outputs back as inputs to generate a time series of predictions. The sea ice height and concentrations are obtained by employing the upwind advection scheme detailed in Section 3.2. This approach enables us to observe how errors propagate over multiple time steps, simulating the model's performance in a sequential forecasting context.

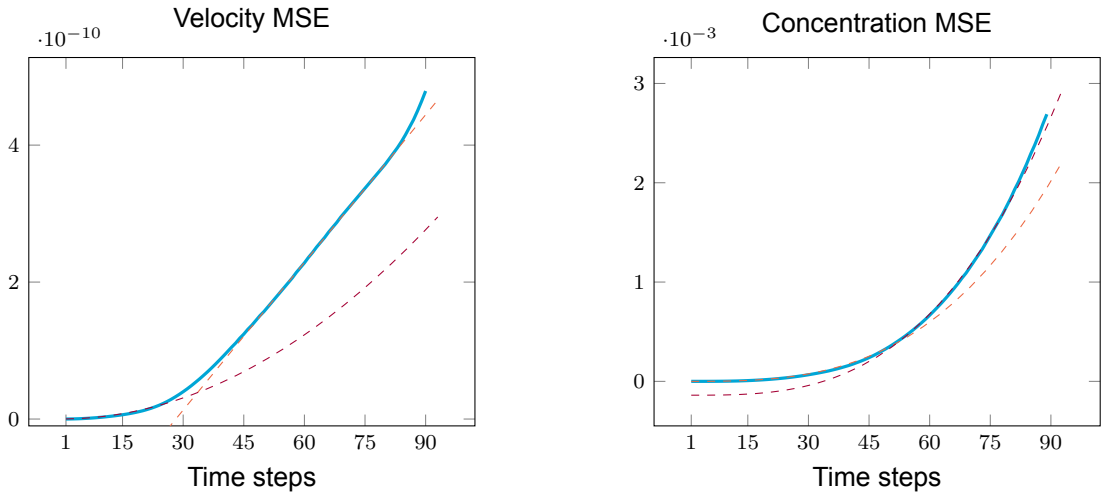


Figure 7.6: MSE of the velocity and concentration for a recurrently applied model on the benchmark problem. The dashed lines show a quadratic and linear for the velocity error and two cubic fits for the concentration error.

Figure 7.6 shows the MSE of the velocity field and sea ice concentration over time. Upon examination of the velocity MSE, we find that initially the errors increase quadratically, after which a linear growth of the errors is found. In the final time steps, the errors increase more rapidly, but we cannot assess the growth behaviour here due to the limited number of time steps. On the other hand, the

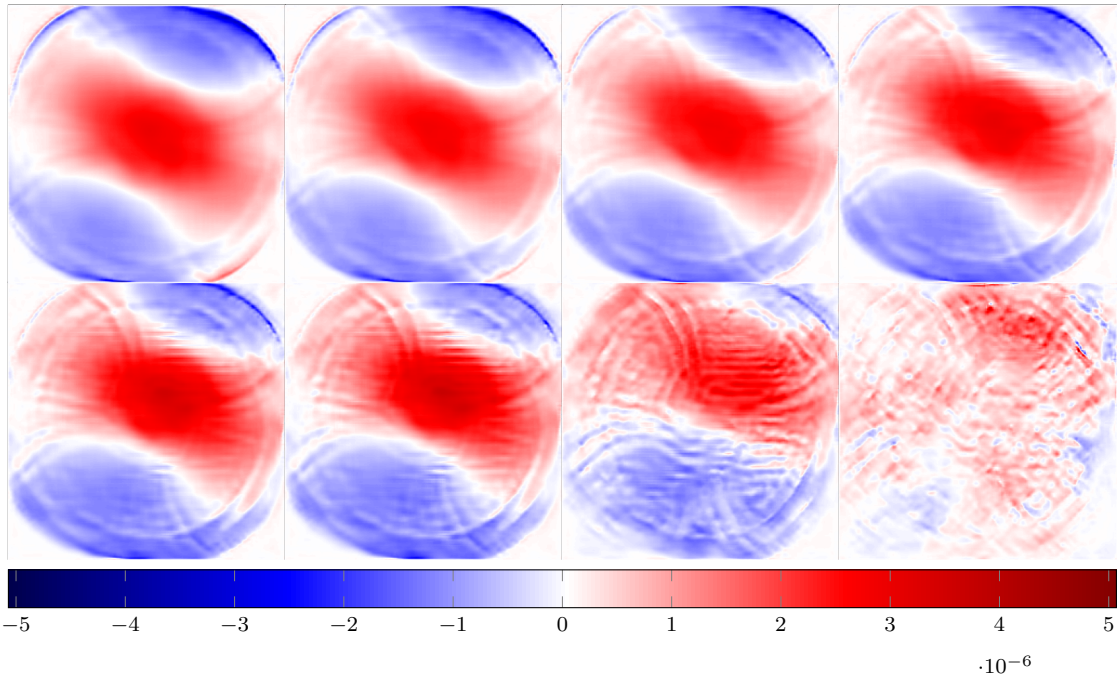


Figure 7.7: Change in velocities since the last time step, for time steps $t = 5, 10, 15, 20, 25, 30, 50, 90$.

error in the sea ice concentration grows cubicly. For reference, curves corresponding to the growth behaviour are shown in Figure 7.6.

Additionally, the change in velocity predicted by the network for time steps $t = 5, 10, 15, 20, 25, 30, 50, 90$ is presented in Figure 7.7, allowing for visual assessment of the accumulation of errors over time. The velocity, concentration and shear deformation for these time steps are presented in Appendix B. From these visualisations, we can see that the results initially loose their sharpness, after which the model's predictions still resemble the expected outputs. Finally, once the distortions have grown significantly, the outputs diverge drastically from the Gascoigne simulation. Visually, the velocity field is recognisable until $t = 30$, but to draw conclusions about the time until which the predictions are feasible, we evaluate the ability to produce LKFs below.

These findings are consistent with earlier findings, where we see that the model is able to capture the large scale patterns, while finer details are not represented well. This causes the initial quadratic growth of the errors, after which the large scale features are still represented relatively well, causing only linear growth of the errors. The different error growth behaviour can be attributed to the different physical regimes in the viscous-plastic sea ice model. If the errors have accumulated substantially, and the ice has shifted from the viscous regime to the plastic regime in some places due to these errors, the model will treat such places accordingly. As a result, we can expect different error growth depending on the physical regime.

This shift in regime occurs when the strain rate becomes of order Δ_{min}. Strain rates that below this parameter result in the viscous regime, while significantly larger strain rates cause plastic behaviour. To investigate when this shift occurs, we have plot the mean strain rate error in Figure 7.8. A horizontal line marks Δ^2_{min}, which is crossed at $t = 27$.

Finally, to evaluate the model's ability to reproduce LKFs, we employ a LKF detection algorithm introduced by Hutter et al. [86], following the adaptations for the idealised experiments by Mehlmann et al. [49, p. 8]. This algorithm detects the LKFs in both the network predictions' shear deformation field, as well as in the Gascoigne simulation. In this way, we can compare the number of LKFs formed.

The number of linear kinematic features formed by the Gascoigne simulation and by the network for different time steps is presented in Figure 7.9. We see that the U-Net underestimates the number of LKFs for the first 25 steps, after which the number rapidly increases. This increase is attributed to the accumulation of errors resulting from the recurrent application of the network. However, it is important to note that during the initial time steps, the difference in the number of detected LKFs between the

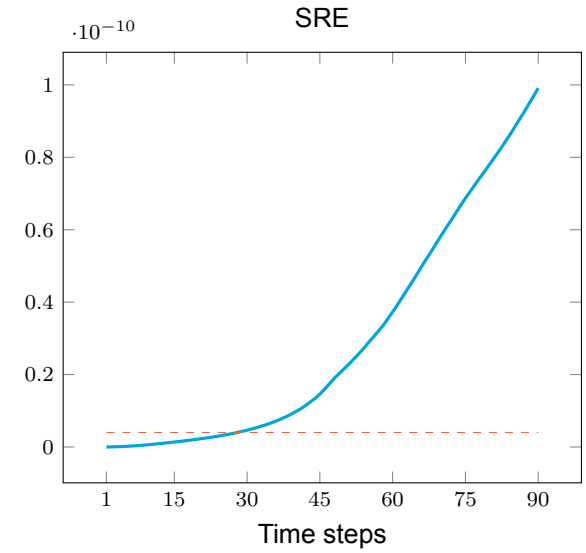


Figure 7.8: SRE for a recurrently applied model on the benchmark problem. A horizontal dashed line marks Δ^2_{min}

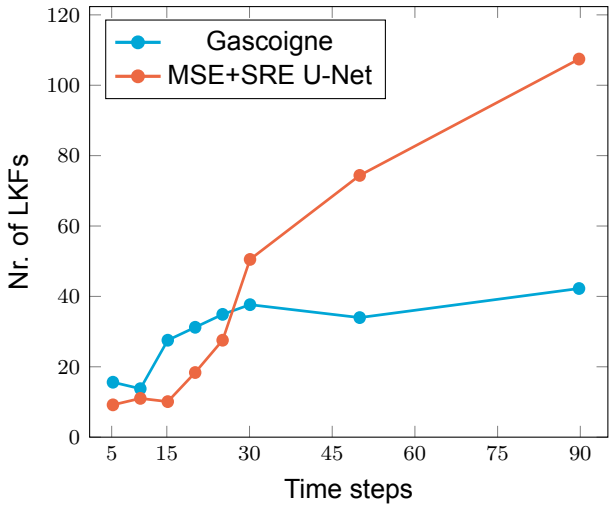
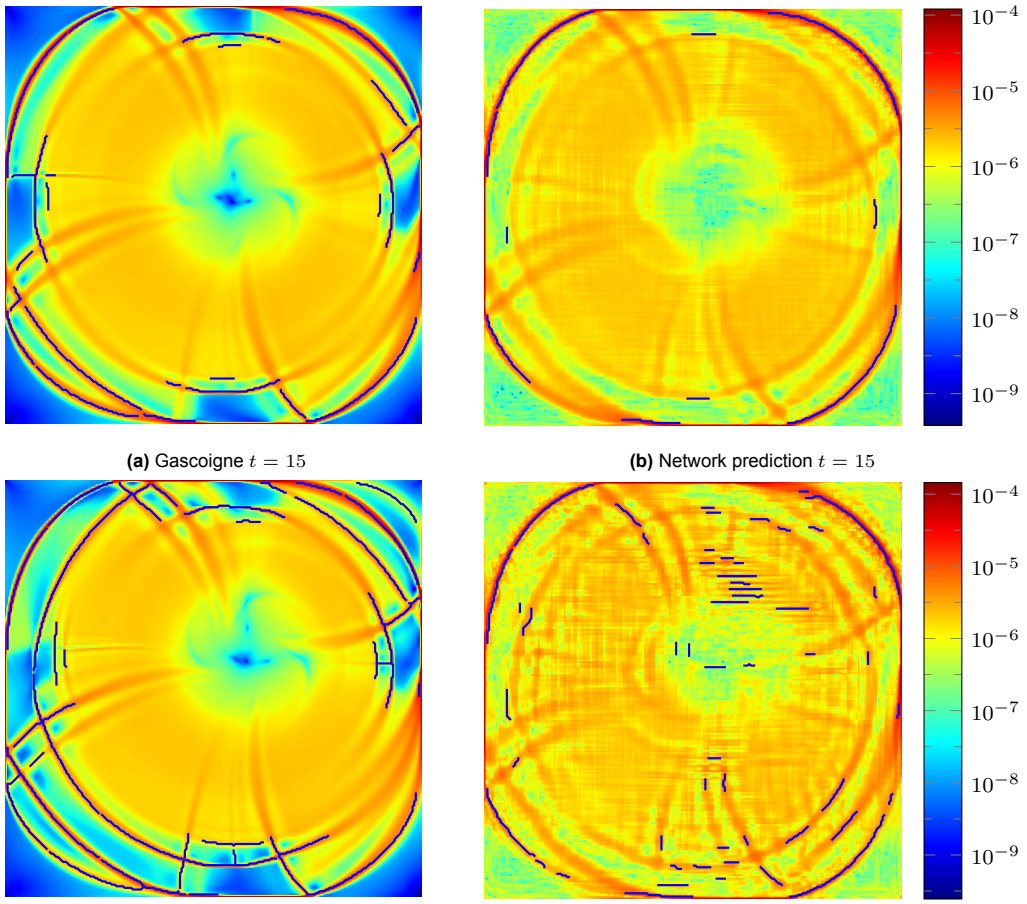


Figure 7.9: Number of linear kinematic features detected by the algorithm by Hutter et al. [86] both the Gascoigne simulation and the network's time series prediction.

network's predictions and the Gascoigne simulation is not substantial. This suggests that the model is able to predict LKFs for several time steps, effectively capturing the essential features of the sea ice dynamics in the short term.

Furthermore, as the model's predictions are fed back as inputs over multiple time steps, small inaccuracies in the velocity field compound, leading to the formation of artificial patterns. These false features are erroneously detected by the algorithm as LKFs, causing the rapid rise in the counted number of LKFs. This suggests that while the model initially struggles to capture newly generated LKFs, over time it generates noise that is indistinguishable from linear kinematic features. This can be seen in Figure 7.10 which shows the detected LKFs in the shear deformation field at $t = 15$ and $t = 30$. The parameters of the detection algorithm could be adjusted to be more sensitive towards finding LKFs, but this would also increase the number of falsely generated and detected LKFs.



(a) Gascoigne $t = 15$

(b) Network prediction $t = 15$

(c) Gascoigne $t = 15$

(d) Network prediction $t = 15$

Figure 7.10: Detected LKFs in the shear deformation field at $t = 15$ and $t = 30$.

7.4. Generalisation

We investigate the data requirements and generalisation behaviour of our neural network models to understand how training data size affects performance and overfitting. Two key analyses were conducted: one examining the impact of varying training dataset sizes on shear error and generalisation loss, and another assessing the model's performance on initial time steps previously excluded from the training data.

Firstly, we evaluated the shear error on both the training and validation datasets for different training dataset sizes of 200, 400, 600, and 800 test cases, as shown in Figure 7.11. The training loss remained approximately constant across all dataset sizes, while the validation loss decreased as the dataset size increased. This indicates that the model's ability to generalise improves with more training data, as it

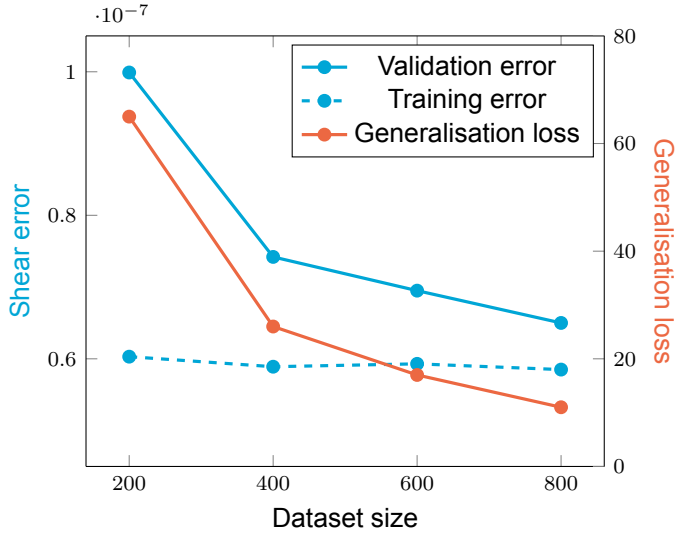


Figure 7.11: The shear error on both the validation (solid) and training dataset versus the dataset size in blue, and the Prechelt Generalisation loss versus the dataset size.

becomes better at predicting unseen data without overfitting. This is underscored by the decrease in Prechelt's generalisation loss:

$$\text{Generalisation loss} = 100 \cdot \left(\frac{L_{\text{val}}}{L_{\text{train}}} - 1 \right), \tag{7.1}$$

where L_{val} and L_{train} are the loss on the validation and training datasets, respectively [61]. Notably, there is no plateau observed near 800 samples, suggesting that further increases in training data could continue to enhance performance. Interestingly, the shear loss at 200 samples is comparable to the shear error obtained from the U-Net trained with the MSE and MSE+MRE loss function, indicating that incorporating the strain rate error into the loss function not only improves accuracy but also enhances data efficiency during training.

Secondly, we examined the model's performance on the full time series of the benchmark problem, which includes the initial 10 time steps that were previously excluded from the training dataset due to their non-physical nature dominated by wind forces, as discussed in Section 5.2. Figure 7.12 shows the mean squared error for single step predictions, contrasting the multi-step approach from the previous section. The results show an exceptionally high error during the first 10 time steps - initially approximately four orders of magnitude larger than the errors in subsequent time steps. Within these initial steps, the error decreases sharply to levels consistent with the rest of the dataset. This significant discrepancy supports the decision to exclude the initial time steps from the training data, as the model is unable to generalise to these non-physical conditions.

Furthermore, we see that the error is the lowest for the first 20 time steps after the initially removed steps, after which the error increases steadily before reaching a plateau. Since the model was trained on data from time steps 10 to 30, and since the error levels still correspond to the MSE of the model on the validation set as presented in Table 7.2, we can conclude that the model generalises well to data from longer simulations.

These results confirm that the model's performance is seriously impacted by the choice of data and the size of the dataset. An adequate size of the training dataset enables the model to generalise well to unseen data and accurately capture the complex dynamics of sea ice.

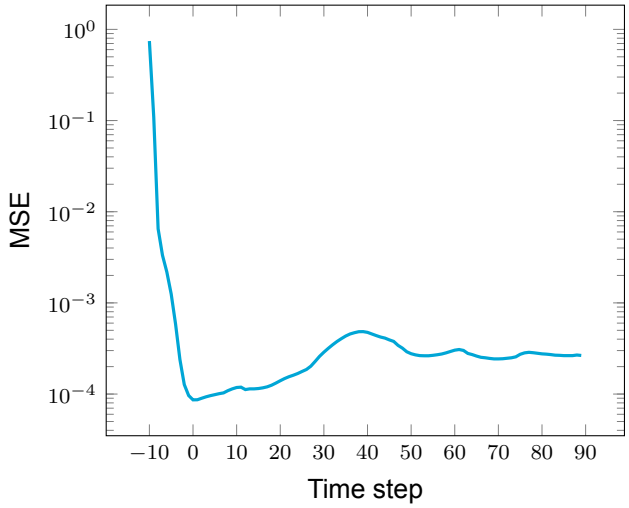


Figure 7.12: Mean squared error (MSE) of the predictions on the full benchmark problem. The first 10 steps were originally removed as they are considered to represent unphysical states and plotted as $t < 10$.

8

Discussion and conclusions

In this work, we have developed a deep learning-based surrogate model for sea ice dynamics with the goal of predicting linear kinematic features. Driven by climate change, there is an increasing demand for accurate sea ice models for both long term predictions to enhance our understanding of climate systems, and for short term predictions in a forecasting setting, for example, to support maritime navigation. The most commonly used models, based on continuum mechanics and the viscous-plastic rheology, encounter significant computational challenges at high resolutions, where linear kinematic features characterising sea ice fields begin to form. To address these limitations, an alternative approach leveraging machine learning techniques and the integration of domain knowledge has been explored. Our surrogate model aims to efficiently predict these LKFs. This chapter will discuss each research question and give conclusions, followed by recommendations for future research.

8.1. Discussion of research questions

Network architecture

To answer the first research question - 'Which neural network architectures are most effective for predicting sea ice dynamics at high resolutions?' - we have examined the performance of different neural network architectures. Specifically, we trained and compared two architectures - the U-Net and the classic bottleneck model - using four different loss functions and assessed their performance using the absolute error of the shear deformation on an unseen subset of the training dataset.

Our analysis revealed that the U-Net consistently outperformed the classic bottleneck, aligning with the findings of Eichinger et al. [39] in a fluid dynamics application. However, the performance difference in our study was more pronounced, indicating a greater advantage of the U-Net in the context of sea ice dynamics. Both architectures feature a bottleneck structure analogous to reduced order models, but the U-Net's increased complexity - with skip connections and 30 layers - contributes to its superior performance compared to the classic bottleneck's 11 layers.

The U-Net's additional layers and skip connections facilitate the representation of processes on different scales within the model, similar to multigrid methods where various scales are captured at different levels. In contrast, the classic bottleneck structure requires even small scale processes to be represented in the latent space at a global level, limiting its ability to accurately capture intricate features such as linear kinematic features.

To ensure a fair comparison, we employed Bayesian optimisation to identify the optimal hyperparameters for each architecture-loss combination. This approach efficiently converged with only 25 to 40 candidates needed for each pair, significantly reducing computational costs compared to traditional grid search methods, for which the number of combination scales with p^n, where p is the number of hyperparameters and n is the number of candidates per hyperparameter. For the MSE+SRE models, for example, having four candidates for each parameter, would require 256 combinations.

A significant challenge observed in our models is the issue of sharpness of the predictions, already visible in the single step predictions. When applied recurrently, errors in sharpness accumulate over time, leading to loss of LKFs. Similar smoothness issues were noted by Finn et al. [47] in their variational autoencoder, which also employs a bottleneck structure. The U-Net, originally designed for image

segmentation, relied on thresholding to generate sharp edges in that setting, but this approach may not fully address the sharpness requirements for sea ice predictions. Suggestions for improvements are made in Section 8.3.

In summary, our comparative analysis demonstrates that the U-Net architecture significantly outperforms the classic bottleneck model in predicting sea ice dynamics. This superior performance is primarily attributed to the U-Net's greater complexity, characterised by its deeper layer structure and incorporation of skip connections. However, despite its advantages, the U-Net still faces challenges related to sharpness of its predictions. These findings highlight the potential of advanced neural network architectures in modelling sea ice dynamics while also highlighting key areas for future improvement, such as improving the sharpness of predictions.

Loss functions

In addressing our research question, 'How do different loss functions, particularly those incorporating domain knowledge, impact the performance of the model?', we trained both the U-Net and classic bottleneck architectures using four distinct loss functions. These included the standard mean squared error (MSE), as well as the mean relative error (MRE) and the mean squared error in the strain rate, referred to as the strain rate error (SRE).

Among the various loss functions tested, the combination of mean squared error and strain rate error (MSE+SRE) resulted in the best representation of sea ice dynamics, as measured by the lowest shear error. The MSE loss function produced overall good results across the dataset, while the SRE specifically enhanced the model's focus on areas with high strain, improving the capture of critical features in those regions. Additionally, the combination of MSE with the mean relative error (MRE) led to better performance in areas with low velocity or low strain rate. The MSE+SRE combination, however, not only achieved the lowest shear error but also, interestingly, the best MSE, indicating that integrating both errors leads to a superior representation of sea ice dynamics regardless of the evaluation metric.

The superior performance of the MSE+SRE combination can be attributed to the intrinsic relationship between shear deformation and strain rate. Since shear deformation is a function of strain rate, training the model using the SRE directly targets the reduction of shear errors, leading to more accurate predictions. Furthermore, strain is essential in the formation of LKFs, making the focus on strain rates particularly relevant for capturing these critical features. This integration of domain knowledge through SRE exemplifies the principles of scientific machine learning, where incorporating physical insights enhances model performance beyond standard data-driven approaches.

The successful use of the strain rate into the loss function is a novel insight that has not been explored in existing literature, as the approaches by Belchansky et al. [42], Herbert et al. [43], Chi and Kim [44], Kim et al. [45], Finn et al. [46], and Finn et al. [47] all use (variations of) L_2 or L_1 losses. Our approach demonstrates the benefits of incorporating domain knowledge into machine learning models. By leveraging scientific machine learning principles, we have enhanced the model's ability to accurately and efficiently represent sea ice dynamics, paving the way for more advanced and reliable surrogate models in the future.

Error accumulation

To answer the research question 'How does error propagate and accumulate over time?', we investigated how errors propagate and accumulate when the network's outputs are used as inputs for subsequent predictions. While our model was trained using single time-step predictions, practical forecasting applications necessitate accurate predictions over multiple time steps.

From the inspection of single time-step errors, we anticipated that problems would primarily arise around linear kinematic features (LKFs), which are characterised by areas of high strain. The model exhibited difficulties in producing sharp results in these regions, leading us to expect that errors would accumulate over time, limiting accurate representation of LKFs. However, since the network predicts the change in velocity between time steps rather than the new velocity field itself, the influence of prediction errors is initially limited. The change in velocity is approximately 40 times smaller than the velocity itself, meaning that early errors have a reduced impact as long as there have not been many time steps for errors to accumulate significantly.

Our analysis of the model's multi-step predictions revealed three distinct phases of error accumulation. In the initial phase, errors remain limited and are predominantly localised around LKFs, as the overall velocity field is still largely influenced by the initial velocity due to its magnitude. As the errors

in small-scale features accumulate, affected regions of sea ice are erroneously modelled in the plastic instead of the viscous regime. As a result, different error growth behaviour can be observed: the internal stress of the ice is approximately independent of the strain rate as long as the ice is in the plastic regime. This independence causes linear growth of the error, due to a significant but constant error term. Eventually, the error lies in the same order of magnitude as the velocity field and we see a more rapid growth of errors.

The change in regime is caused by an error in the strain rate. To illustrate, the regime change is characterised by the parameter Δ_{min}. Strain rates that below this parameter result in the viscous regime, while significantly larger strain rates cause plastic behaviour. Therefore, once the error in the strain rate becomes of order Δ_{min}, the ice is typically modelled as a plastic material. The average error in the strain rate crosses this boundary around time step $t = 27$, but it happens earlier locally. Since the forecasting starts to diverge too much from the ground truth when the error in the strain rate becomes of order Δ_{min}, reducing the strain rate should be an key focus.

In summary, we have found that the model's predictive skills degenerates after approximately 20 time steps. Despite the challenges associated with error accumulation, it is noteworthy that a trained network can generate a forecast of 20 steps within a few seconds. In contrast, the Gascoigne simulation requires approximately an hour to compute the same number of time steps. This is significant difference in computational efficiency highlights the potential of the surrogate model for applications where rapid predictions are essential, provided that issues with long-term accuracy can be addressed.

Prediction of linear kinematic features

To address the fourth research question - 'To what extent can the model accurately predict linear kinematic features?' - we evaluated the model's performance in forecasting LKFs. Our findings indicate that the U-Net model trained with the combined mean squared error and strain rate error loss function is capable of accurately predicting LKFs for up to 10 time steps, corresponding to approximately 5.5 hours of forecasting time. Beyond this point, the forecast begins to degenerate, and the LKF detection algorithm by Hutter et al. identifies a divergence in the number of LKFs compared to the benchmark simulations.

To put these findings into perspective, we compared our model's performance with the results from Mehlmann et al. [49]. While our model predicts fewer LKFs than their B-grid model, which we used four our benchmark, the difference is acceptable. The discrepancy is similar to the differences observed between their A-grid and B-grid simulations or between simulations at different resolutions, such as 2 km versus 4 km. Specifically, after 10 time steps, our model produced 15 LKFs compared to 18 LKFs in the B-grid model, indicating a reasonable level of accuracy within this forecasting window.

Furthermore, our analysis demonstrates that the model trained using only the MSE loss function is unable to adequately predict LKFs, underscoring the necessity of incorporating the SRE into the loss function. The inclusion of the SRE enables the model to focus on capturing the strain rates critical for LKF formation, thereby enhancing its predictive capabilities within the initial 10 time steps. These results highlight the key finding of this study: the U-Net model trained with the MSE+SRE loss function effectively predicts LKFs for short-term forecasts. Recommendations for mitigating error accumulation to extend this predictive capacity to longer forecasting times are made below.

Data considerations

We now address the final research question, 'Is the choice of training data critical for the surrogate model's performance?'. To train a model for application to the benchmark problem, we generated 800 sample test cases with 30 time steps each. To enhance the training dataset, several data augmentation techniques were employed, including reflecting and rotating the data and adding multiplicative noise to the inputs. Furthermore, we removed the initial ten time steps from simulations due to their unphysical nature. These methods were implemented either to synthetically increase the size of the dataset or to improve the network's ability to learn and generalise from the data, thereby ensuring the model could robustly capture the complex dynamics of sea ice.

We have found that the dataset size and the selection of training data are important factors determining the performance of the model. Two primary analyses were conducted to support this claim. From our analysis of the network's performance for varying dataset sizes, we determined that an increased volume of training data improves the model's accuracy, suggesting that a larger dataset could further improve the network's predictions. The second analysis focused on evaluating the network's perfor-

mance on the initially removed unphysical time steps to assess the model's ability to generalise to data that differed significantly from the training set. We found that the model generalised poorly on these initial time steps, underscoring the importance of training data selection. Furthermore, it was found that the model generalises reasonably well to data from simulations that extend beyond the length of time series included in the dataset.

Other findings concerning the dataset design include that the addition of multiplicative noise did not enhance the performance of the MSE+SRE model, as evidenced by the low noise level of 0.1% found by the Bayesian optimisation. A larger noise level resulted in an increase in the shear error, which could be explained by the sensitivity of the strain rate error to noise.

8.2. Conclusions

This study has successfully developed a deep learning-based surrogate model for sea ice dynamics with the goal of predicting linear kinematic features (LKFs). The model effectively predicts LKFs for short-term forecasts of up to approximately 5.5 hours. Beyond this period, the forecast begins to deteriorate, and after 11 hours, the predictions are no longer reliable. Despite this limitation, the surrogate model offers significant computational efficiency, generating a forecast of 10 steps within a second compared to 30 minutes required by traditional numerical simulations.

Key findings of the research include the identification of the U-Net architecture as the most effective neural network model for predicting sea ice dynamics at high resolutions. Its superior performance is attributed to its depth in terms of layers and incorporation of skip connections, which enable the capture of complex patterns inherent in sea ice behaviour at different scales. Additionally, the novel integration of the strain rate error (SRE) into the loss function significantly enhances the model's ability to predict LKFs. This approach outperforms traditional loss functions like the mean squared error (MSE) alone, demonstrating the benefits of incorporating domain knowledge into machine learning models.

However, the study also found that errors accumulate over multiple time steps, leading to a degradation of predictive accuracy after approximately 20 time steps. This highlights the importance of addressing error propagation for long-term forecasting applications. Furthermore, the choice and preparation of training data were determined to be critical factors influencing the model's performance and generalisation capabilities.

In summary, by integrating domain-specific insights and employing advanced neural network architectures, the surrogate model shows promise for rapid, short-term forecasts of sea ice dynamics. To extend its reliability over longer forecasting periods, recommendations for improving prediction sharpness and reducing error accumulation are presented in the following section.

8.3. Recommendations

To enhance the performance and extend the forecasting capabilities of our deep learning-based surrogate model for sea ice dynamics, several key areas for improvement have been identified. Primarily, increasing the sharpness of the model's predictions and ensuring a low error in the strain rates over longer forecasting periods are essential for accurately capturing linear kinematic features (LKFs). The following recommendations outline potential strategies to address these challenges.

One promising approach is to explore alternative neural network architectures, such as recurrent neural networks (RNNs), which are inherently designed for time series predictions. Unlike feedforward networks, RNNs can retain information from previous time steps by passing the encoding or hidden states forward in time. This capability allows the model to capture temporal dependencies and mitigates the accumulation of errors introduced by sequential passes through the decoder and encoder in multi-step predictions. By leveraging RNN architectures, the model could achieve improved stability and accuracy over extended forecasting periods, effectively addressing the issue of error propagation in long-term predictions.

Another method to enhance the model's performance is the application of adversarial training techniques. In our study, we found that accurately representing strain rates is crucial for the prediction of LKFs, which was achieved through explicitly adding an error in the strain rate to the loss function. Adversarial training, however, can implicitly learn important aspects of sea ice dynamics, including strain rates, by training a second, adversarial network to distinguish between true and generated samples. Kemna et al. [87] demonstrated this approach by employing an adversarial network that learns visual differences, resulting in more visually convincing predictions. Although their method yielded less ac-

curate predictions compared to direct training on ground truth data, a hybrid approach that combines adversarial and conventional training showed promising results. Applying a similar strategy to sea ice dynamics could enhance the model's ability to maintain sharpness and accurately predict LKFs over longer time frames.

Furthermore, adversarial training could help suppress error accumulation by encouraging the model to produce outputs that are consistent with realistic sea ice dynamics. By implicitly learning which features are most important for LKF prediction, the model may better capture complex patterns and reduce the smoothing effects observed in multi-step predictions. This approach not only offers a structural alternative but also provides an opportunity to modify the loss function to better represent critical aspects of sea ice behaviour.

In addition to architectural and training method improvements, increasing the quantity and diversity of training data could significantly enhance the model's performance. A balance between longer time series and a greater number of test cases should be sought. Extending the length of training sequences would allow the model to learn sea ice dynamics characteristics that emerge over extended periods, improving its ability to generalise to longer forecasting horizons. Simultaneously, incorporating more test cases would introduce a wider variety of scenarios and conditions, providing the model with richer information, enhancing its robustness, and preventing overfitting.

Lastly, an approach that integrates the surrogate model into the numerical methods could be investigated. Such an approach would be a typical example of the first scientific machine learning paradigm detailed in Section 1.2. For example, the surrogate model could provide an initial guess for the Newton solver that is then refined by the traditional numerical methods, potentially reducing the number of iterations required for convergence.

In summary, to improve the surrogate model's predictive capabilities and extend its effective forecasting period, future work should focus on exploring recurrent neural network architectures, implementing adversarial training techniques, and expanding the training dataset. By addressing the issues of prediction sharpness and error accumulation through these strategies, the model's accuracy and reliability in predicting linear kinematic features over longer time frames can be improved, thereby increasing its practical use in applications such as Arctic navigation.

Bibliography

[1] M. Meredith et al. “Polar Regions”. In: *IPCC Special Report on the Ocean and Cryosphere in a Changing Climate*. Ed. by H.-O. Pörtner et al. Cambridge University Press, 2019, pp. 203–320.

[2] JGL Rae et al. “Development of the global sea ice 6.0 CICE configuration for the Met Office global coupled model”. In: *Geoscientific Model Development* 8.7 (2015), pp. 2221–2230.

[3] Ed Blockley et al. “The Future of Sea Ice Modeling: Where Do We Go from Here?” In: *Bulletin of the American Meteorological Society* 101.8 (2020), pp. E1304–E1311. ISSN: 00030007, 15200477. URL: https://www.jstor.org/stable/27110602.

[4] Salvatore Cuomo et al. “Scientific machine learning through physics–informed neural networks: where we are and what’s next”. In: *Journal of Scientific Computing* 92.3 (2022), p. 88.

[5] Deep Ray and Jan S Hesthaven. “An artificial neural network as a troubled-cell indicator”. In: *Journal of computational physics* 367 (2018), pp. 166–191.

[6] Nils Margenberg et al. “A neural network multigrid solver for the Navier-Stokes equations”. In: *Journal of Computational Physics* 460 (2022), p. 110983.

[7] Xiaoxiao Guo, Wei Li, and Francesco Iorio. “Convolutional neural networks for steady flow approximation”. In: *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 2016, pp. 481–490.

[8] Henry Jin, Marios Mattheakis, and Pavlos Protopapas. “Unsupervised neural networks for quantum eigenvalue problems”. In: *arXiv preprint arXiv:2010.05075* (2020).

[9] Jonathan Tompson et al. “Accelerating Eulerian Fluid Simulation With Convolutional Networks”. In: *Proceedings of the 34th International Conference on Machine Learning*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. PMLR, 2017, pp. 3424–3433. URL: https://proceedings.mlr.press/v70/tompson17a.html.

[10] Michon Scott and Kathryn Hansen. *Sea ice*. NASA Earth Observatory, 2016. URL: https://earthobservatory.nasa.gov/features/SeaIce (visited on 05/23/2023).

[11] Nico Wunderling et al. “Global warming due to loss of large ice masses and Arctic summer sea ice”. In: *Nature Communications* 11.1 (2020), p. 5177.

[12] Thomas Jung et al. “Advancing polar prediction capabilities on daily to seasonal time scales”. In: *Bulletin of the American Meteorological Society* 97.9 (2016), pp. 1631–1647.

[13] Mike Carlowicz. *Partial Opening of the Northwest Passage*. NASA Earth Observatory. Sept. 13, 2015. URL: https://earthobservatory.nasa.gov/images/86589/partial-opening-of-the-northwest-passage (visited on 06/08/2023).

[14] Kathryn Hansen. *A nearly ice-free Northwest Passage*. NASA Earth Observatory. Aug. 20, 2016. URL: https://earthobservatory.nasa.gov/images/88597/a-nearly-ice-free-northwest-passage (visited on 06/08/2023).

[15] Mahdi Mohammadi-Aragh et al. “Predictability of Arctic sea ice on weather time scales”. In: *Scientific reports* 8.1 (2018), p. 6514.

[16] R Kwok et al. “Variability of sea ice simulations assessed with RGPS kinematics”. In: *Journal of Geophysical Research: Oceans* 113.C11 (2008).

[17] Nils Hutter, Martin Losch, and Dimitris Menemenlis. “Scaling properties of arctic sea ice deformation in a high-resolution viscous-plastic sea ice model and in satellite observations”. In: *Journal of Geophysical Research: Oceans* 123.1 (2018), pp. 672–687.

[18] Kathryn Hansen. *Drifting With Broken Sea Ice*. NASA Earth Observatory. Mar. 31, 2020. URL: https://earthobservatory.nasa.gov/images/146508/drifting-with-broken-sea-ice (visited on 06/08/2023).

[19] Ian Joughin et al. “Brief communication: Further summer speedup of Jakobshavn Isbræ”. In: *The Cryosphere* 8.1 (2014), pp. 209–214.

[20] Magdalena Łukosz, Ryszard Hejmanowski, and Wojciech T Witkowski. “Analysis of the velocity changes of the Jakobshavn Glacier based on SAR imagery”. In: *Quaestiones Geographicae* 41.1 (2022), pp. 93–105.

[21] W D III Hibler. “A dynamic thermodynamic sea ice model”. In: *Journal of physical oceanography* 9.4 (1979), pp. 815–846.

[22] Chi F Ip, William D Hibler, and Gregory M Flato. “On the effect of rheology on seasonal sea-ice simulations”. In: *Annals of Glaciology* 15 (1991), pp. 17–25.

[23] Elizabeth C Hunke and John K Dukowicz. “An elastic–viscous–plastic model for sea ice dynamics”. In: *Journal of Physical Oceanography* 27.9 (1997), pp. 1849–1867.

[24] Madlen Kimmritz, Sergey Danilov, and Martin Losch. “On the convergence of the modified elastic–viscous–plastic method for solving the sea ice momentum equation”. In: *Journal of Computational Physics* 296 (2015), pp. 90–100.

[25] Jinlun Zhang and WD Hibler III. “On an efficient numerical method for modeling sea ice dynamics”. In: *Journal of Geophysical Research: Oceans* 102.C4 (1997), pp. 8691–8702.

[26] Jean-François Lemieux et al. “Improving the numerical convergence of viscous-plastic sea ice models with the Jacobian-free Newton–Krylov method”. In: *Journal of Computational Physics* 229.8 (2010), pp. 2840–2852.

[27] Jean-François Lemieux et al. “A second-order accurate in time IMplicit–EXplicit (IMEX) integration scheme for sea ice dynamics”. In: *Journal of Computational Physics* 263 (2014), pp. 375–392.

[28] Jean-François Lemieux and Bruno Tremblay. “Numerical convergence of viscous-plastic sea ice models”. In: *Journal of Geophysical Research: Oceans* 114.C5 (2009).

[29] C Mehlmann and T Richter. “A modified global Newton solver for viscous-plastic sea ice models”. In: *Ocean Modelling* 116 (2017), pp. 96–107.

[30] Carolin Mehlmann and Thomas Richter. “A finite element multigrid-framework to solve the sea ice momentum equation”. In: *Journal of Computational Physics* 348 (2017), pp. 847–861.

[31] Yu-hsuan Shih et al. “Robust and efficient primal-dual Newton-Krylov solvers for viscous-plastic sea-ice models”. In: *Journal of Computational Physics* 474 (2023), p. 111802. ISSN: 0021-9991. DOI: https://doi.org/10.1016/j.jcp.2022.111802. URL: https://www.sciencedirect.com/science/article/pii/S0021999122008658.

[32] Rick Stevens et al. *AI for Science: Report on the Department of Energy (DOE) Town Halls on Artificial Intelligence (AI) for Science*. Tech. rep. Department of Energy, Feb. 2020. DOI: 10.2172/1604756. URL: https://www.osti.gov/biblio/1604756.

[33] Alexander Heinlein et al. “Combining machine learning and domain decomposition methods for the solution of partial differential equations—A review”. In: *GAMM-Mitteilungen* 44.1 (2021), e202100001.

[34] Isaac E Lagaris, Aristidis Likas, and Dimitrios I Fotiadis. “Artificial neural networks for solving ordinary and partial differential equations”. In: *IEEE transactions on neural networks* 9.5 (1998), pp. 987–1000.

[35] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. “Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations”. In: *arXiv preprint arXiv:1711.10561* (2017).

[36] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. “Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations”. In: *arXiv preprint arXiv:1711.10561* (2017).

[37] Weinan E and Bing Yu. “The deep Ritz method: a deep learning-based numerical algorithm for solving variational problems”. In: *Communications in Mathematics and Statistics* 6.1 (2018), pp. 1–12.

[38] Yaohua Zang et al. “Weak adversarial networks for high-dimensional partial differential equations”. In: *Journal of Computational Physics* 411 (2020), p. 109409.

[39] Matthias Eichinger, Alexander Heinlein, and Axel Klawonn. “Surrogate convolutional neural network models for steady computational fluid dynamics simulations”. In: (2020).

[40] Mario Ohlberger and Stephan Rave. “Reduced basis methods: Success, limitations and future challenges”. In: *arXiv preprint arXiv:1511.02021* (2015).

[41] Chiyuan Zhang et al. “Understanding deep learning (still) requires rethinking generalization”. In: *Communications of the ACM* 64.3 (2021), pp. 107–115.

[42] GI Belchansky, David C Douglas, and Nikita G Platonov. “Fluctuating Arctic sea ice thickness changes estimated by an in situ learned and empirically forced neural network model”. In: *Journal of Climate* 21.4 (2008), pp. 716–729.

[43] Christoph Herbert et al. “Sea ice thickness estimation based on regression neural networks using L-band microwave radiometry data from the FSSCat mission”. In: *Remote Sensing* 13.7 (2021), p. 1366.

[44] Junhwa Chi and Hyun-choel Kim. “Prediction of arctic sea ice concentration using a fully data driven deep neural network”. In: *Remote Sensing* 9.12 (2017), p. 1305.

[45] Young Jun Kim et al. “Prediction of monthly Arctic sea ice concentrations using satellite and reanalysis data based on convolutional neural networks”. In: *The Cryosphere* 14.3 (2020), pp. 1083–1104.

[46] T. S. Finn et al. “Deep learning of subgrid-scale parametrisations for short-term forecasting of sea-ice dynamics with a Maxwell-Elasto-Brittle rheology”. In: *EGUsphere* 2023 (2023), pp. 1–39. DOI: 10.5194/egusphere-2022-1342. URL: https://egusphere.copernicus.org/preprints/2023/egusphere-2022-1342/.

[47] Tobias Sebastian Finn et al. “Towards diffusion models for large-scale sea-ice modelling”. In: *arXiv preprint arXiv:2406.18417* (2024).

[48] Carolin Mehlmann. “Efficient numerical methods to solve the viscous-plastic sea ice model at high spatial resolutions”. PhD thesis. Dissertation, Magdeburg, Otto-von-Guericke-Universität Magdeburg, 2019, 2019.

[49] Carolin Mehlmann et al. “Simulating linear kinematic features in viscous-plastic sea ice models on quadrilateral and triangular grids with different variable staggering”. In: *Journal of Advances in Modeling Earth Systems* 13.11 (2021), e2021MS002523.

[50] L Bruno Tremblay and LA Mysak. “Modeling sea ice as a granular material, including the dilatancy effect”. In: *Journal of Physical Oceanography* 27.11 (1997), pp. 2342–2360.

[51] Roland Becker et al. *Gascoigne 3D*. URL: https://gascoigne.math.uni-magdeburg.de/index.php.

[52] Randall J. LeVeque. *Finite Volume Methods for Hyperbolic Problems*. Cambridge Texts in Applied Mathematics. Cambridge University Press, 2002.

[53] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[54] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. “Multilayer feedforward networks are universal approximators”. In: *Neural networks* 2.5 (1989), pp. 359–366.

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

[56] Allan Pinkus. “Approximation theory of the MLP model in neural networks”. In: *Acta Numerica* 8 (1999), pp. 143–195. DOI: 10.1017/S0962492900002919.

[57] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. “Learning representations by back-propagating errors”. In: *nature* 323.6088 (1986), pp. 533–536.

[58] Léon Bottou. “Stochastic Gradient Descent Tricks”. In: *Neural Networks: Tricks of the Trade: Second Edition*. Ed. by Grégoire Montavon, Geneviève B. Orr, and Klaus-Robert Müller. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 421–436. ISBN: 978-3-642-35289-8. DOI: 10.1007/978-3-642-35289-8_25. URL: https://doi.org/10.1007/978-3-642-35289-8_25.

[59] B.T. Polyak. “Some methods of speeding up the convergence of iteration methods”. In: *USSR Computational Mathematics and Mathematical Physics* 4.5 (1964), pp. 1–17. ISSN: 0041-5553. DOI: https://doi.org/10.1016/0041-5553(64)90137-5. URL: https://www.sciencedirect.com/science/article/pii/0041555364901375.

[60] Diederik Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *International Conference on Learning Representations (ICLR)*. San Diega, CA, USA, 2015.

[61] Lutz Prechelt. “Early Stopping - But When?” In: *Neural Networks: Tricks of the Trade*. Ed. by Genevieve B. Orr and Klaus-Robert Müller. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 55–69. ISBN: 978-3-540-49430-0. DOI: 10.1007/3-540-49430-8_3. URL: https://doi.org/10.1007/3-540-49430-8_3.

[62] Tomáš Mikolov. “Statistical Language Models Based on Neural Networks”. PhD thesis. Brno University of Technology, Faculty of Information Technology, 2012.

[63] Robert Tibshirani. “Regression shrinkage and selection via the lasso”. In: *Journal of the Royal Statistical Society Series B: Statistical Methodology* 58.1 (1996), pp. 267–288.

[64] Arthur E Hoerl and Robert W Kennard. “Ridge regression: Biased estimation for nonorthogonal problems”. In: *Technometrics* 12.1 (1970), pp. 55–67.

[65] Sergey Ioffe and Christian Szegedy. “Batch normalization: Accelerating deep network training by reducing internal covariate shift”. In: *arXiv preprint arXiv:1502.03167* (2015).

[66] Moein Hasani and Hassan Khotanlou. “An Empirical Study on Position of the Batch Normalization Layer in Convolutional Neural Networks”. In: *2019 5th Iranian Conference on Signal Processing and Intelligent Systems (ICSPIS)*. 2019, pp. 1–4. DOI: 10.1109/ICSPIS48872.2019.9066113.

[67] Y. Le Cun et al. “Handwritten Digit Recognition: Applications of Neural Network Chips and Automatic Learning”. English (US). In: *IEEE Communications Magazine* 27.11 (Nov. 1989), pp. 41–46. ISSN: 0163-6804. DOI: 10.1109/35.41400.

[68] PyTorch. *torch.nn.Conv2d — PyTorch 2.3 documentation*. 2023. URL: https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html (visited on 07/16/2024).

[69] TensorFlow. *tf.nn.convolution — TensorFlow Core v2.16.1 documentation*. 2023. URL: https://www.tensorflow.org/api_docs/python/tf/nn/convolution (visited on 07/16/2024).

[70] Vincent Dumoulin and Francesco Visin. “A guide to convolution arithmetic for deep learning”. In: *ArXiv e-prints* (Mar. 2016). eprint: 1603.07285.

[71] Matthew D. Zeiler et al. “Deconvolutional networks”. In: *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 2010, pp. 2528–2535. DOI: 10.1109/CVPR.2010.5539957.

[72] Augustus Odena, Vincent Dumoulin, and Chris Olah. “Deconvolution and Checkerboard Artifacts”. In: *Distill* (2016). DOI: 10.23915/distill.00003. URL: http://distill.pub/2016/deconv-checkerboard.

[73] Brendan Analikwu. *Data generation for deep learning based sea ice dynamics*. URL: https://github.com/BrendanAnalikwu/MasterThesisGascoigne.

[74] M Eren Akbiyik. “Data augmentation in training CNNs: injecting noise to images”. In: *arXiv preprint arXiv:2307.06855* (2023).

[75] Peng Zhang et al. *Advancing Gamma-Ray Burst Identification through Transfer Learning with Convolutional Neural Networks*. 2024. arXiv: 2408.13598.

[76] Kartik Audhkhasi, Osonde Osoba, and Bart Kosko. “Noise-enhanced convolutional neural networks”. In: *Neural Networks* 78 (2016), pp. 15–23.

[77] Pieter Wesseling. *Introduction to multigrid methods*. John Wiley & Sons Ltd., 1992.

[78] Dan Hendrycks and Kevin Gimpel. “Gaussian error linear units (GELUs)”. In: *arXiv preprint arXiv:1606.08415* (2016).

[79] HarisIqbal88. *PlotNeuralNet GitHub repository*. URL: https://github.com/HarisIqbal88/PlotNeuralNet/ (visited on 08/12/2024).

[80] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. “U-Net: Convolutional Networks for Biomedical Image Segmentation”. In: *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*. Ed. by Nassir Navab et al. Cham: Springer International Publishing, 2015, pp. 234–241. ISBN: 978-3-319-24574-4.

[81] Jonas Mockus, Vytautas Tiesis, and Antanas Zilinskas. “The Application of Bayesian Methods for Seeking the Extremum”. In: *Towards Global Optimization* 2.117-129 (1978), p. 2.

[82] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009. ISBN: 1441412697.

[83] Jason Ansel et al. “PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation”. In: *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS ’24)*. ACM, Apr. 2024. DOI: 10.1145/3620665.3640366. URL: https://pytorch.org/assets/pytorch2-2.pdf.

[84] Brendan Analikwu. *Deep learning based sea ice dynamics*. URL: https://github.com/BrendanAnalikwu/MasterThesis.

[85] Delft High Performance Computing Centre (DHPC). *DelftBlue Supercomputer (Phase 2)*. https://www.tudelft.nl/dhpc/ark:/44463/DelftBluePhase2. 2024.

[86] N. Hutter, L. Zampieri, and M. Losch. “Leads and ridges in Arctic sea ice from RGPS data and a new tracking algorithm”. In: *The Cryosphere* 13.2 (2019), pp. 627–645. DOI: 10.5194/tc-13-627-2019. URL: https://tc.copernicus.org/articles/13/627/2019/.

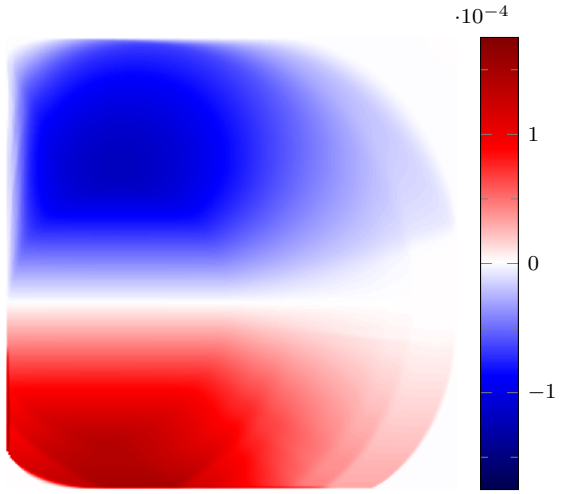
[87] Mirko Kemna, Alexander Heinlein, and Cornelis Vuik. “Reduced order fluid modeling with generative adversarial networks”. In: *Proc. Appl. Math. Mech.* Vol. 23. 1. 2023, e202200241. DOI: 10.1002/pamm.202200241. URL: https://onlinelibrary.wiley.com/doi/10.1002/pamm.202200241.

Appendices

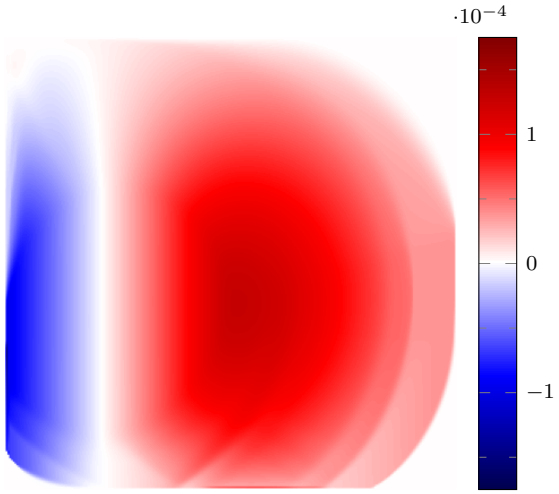
A

Examples of training data

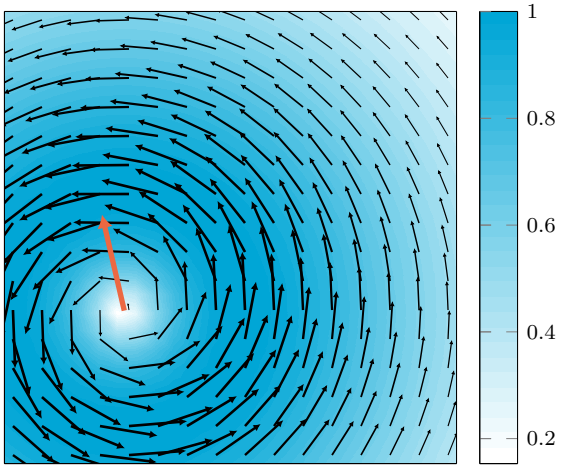
Here, we show some examples of randomly generated test cases that are used as training or validation data. The sea ice velocity in both the x- and y-direction is presented, as well as the wind velocity field, sea ice concentration and the shear deformation. The visualisations are made for $t = 20$ for all test cases.



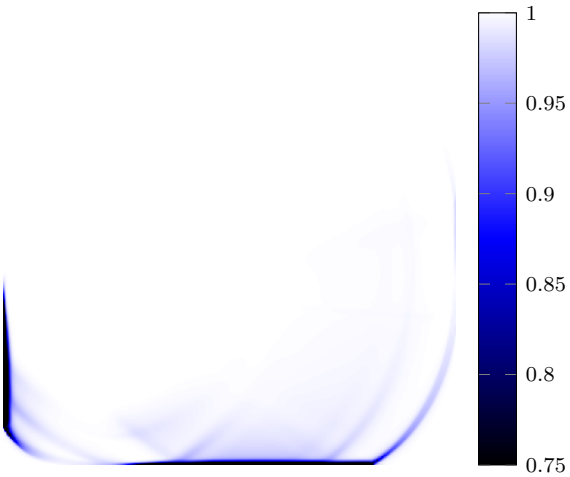
(a) Sea ice velocity in the x-direction after 20 time steps.



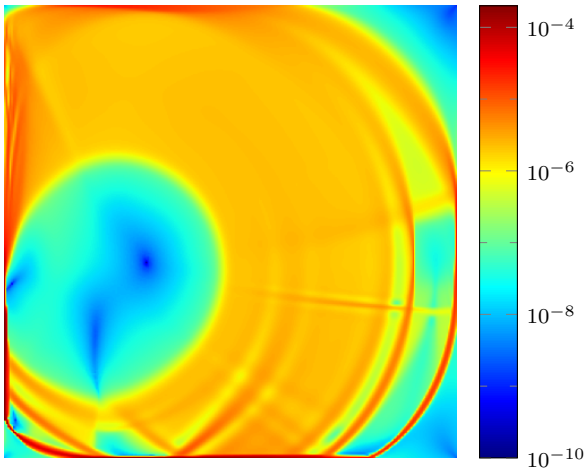
(b) Sea ice velocity in the y-direction after 20 time steps.



(c) Sketch of the Wind velocity field in terms of $v_a^{\max}$.

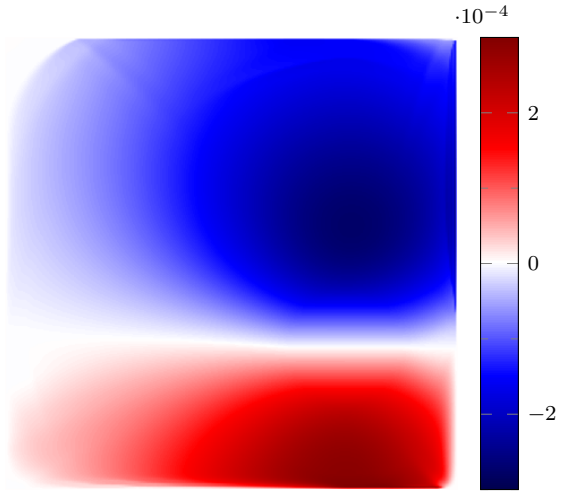


(d) Sea ice concentration after 20 time steps.

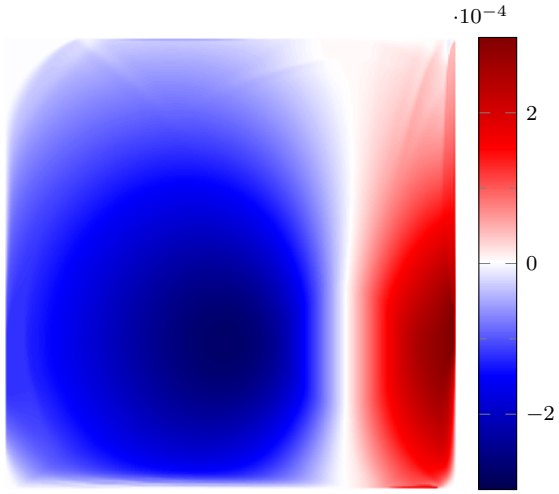


(e) Shear deformation after 20 time steps.

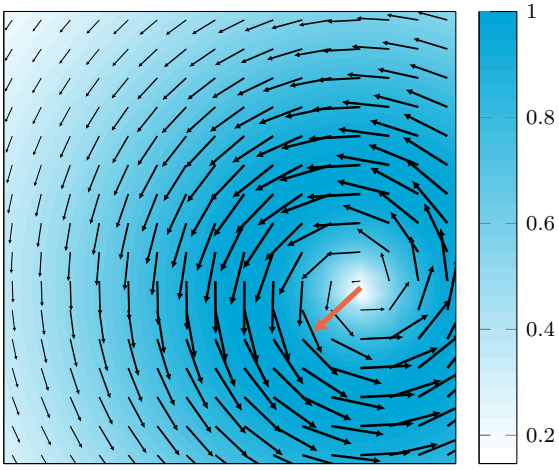
Figure A.1: Gascoigne solution from the training dataset. $H_0 = 0.414$ m and $v_a^{\max} = 10.0$ m/s.



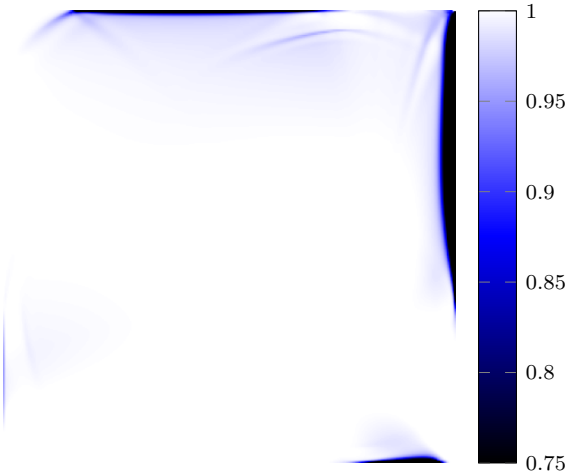
(a) Sea ice velocity in the x-direction after 20 time steps.



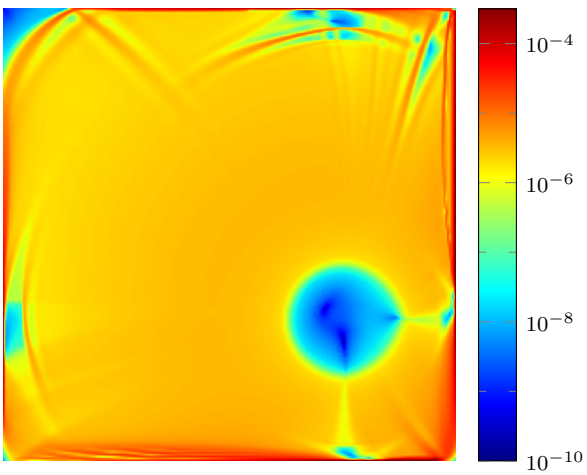
(b) Sea ice velocity in the y-direction after 20 time steps.



(c) Sketch of the Wind velocity field in terms of $v_a^{\max}$.

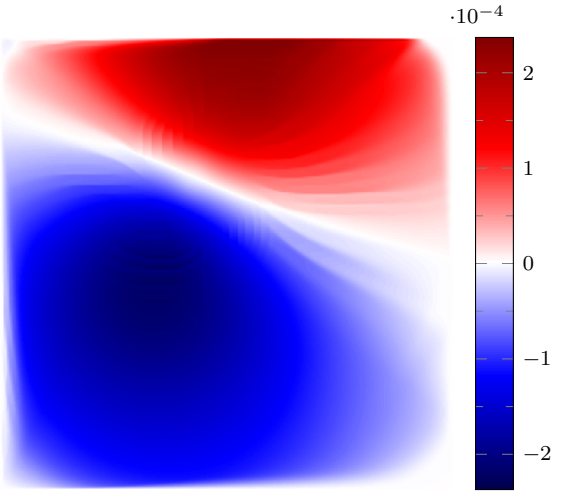


(d) Sea ice concentration after 20 time steps.

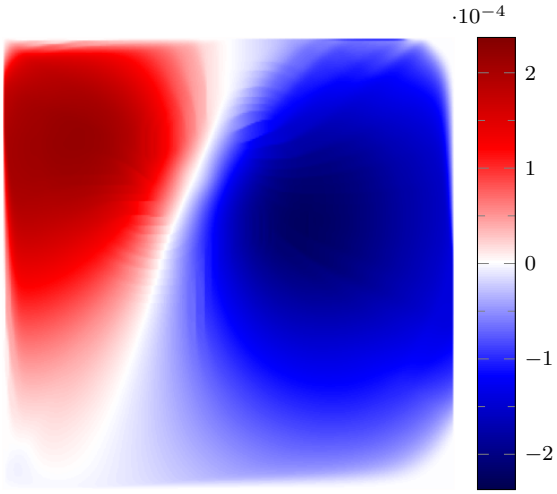


(e) Shear deformation after 20 time steps.

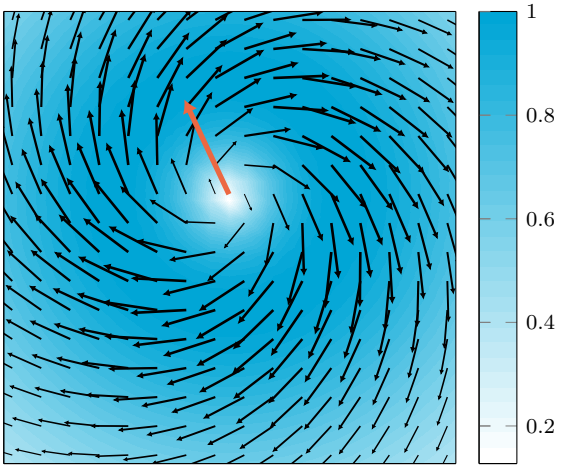
Figure A.2: Gascoigne solution from the training dataset. $H_0 = 0.174$ m and $v_a^{\max} = 17.4$ m/s.



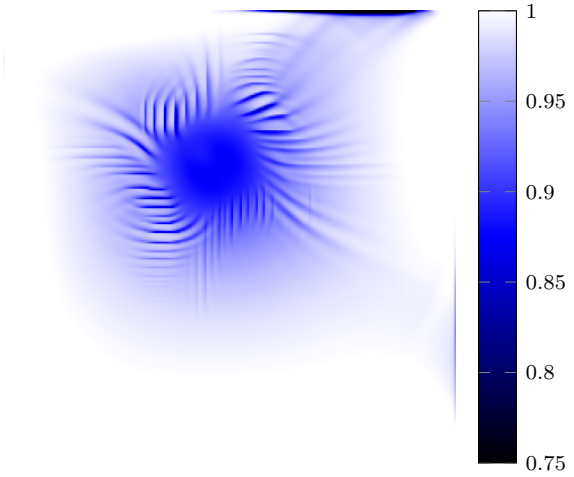
(a) Sea ice velocity in the x-direction after 20 time steps.



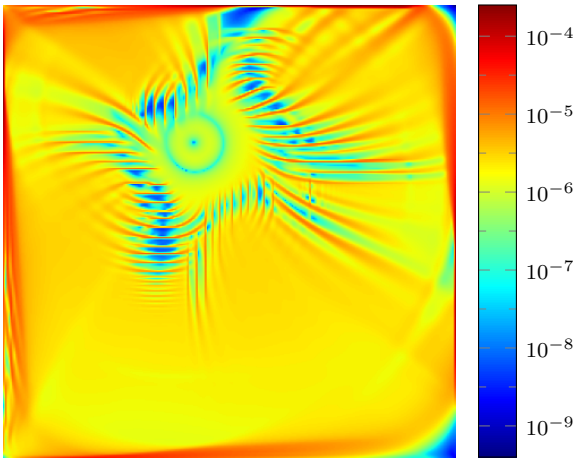
(b) Sea ice velocity in the y-direction after 20 time steps.



(c) Sketch of the Wind velocity field in terms of $v_a^{\max}$.

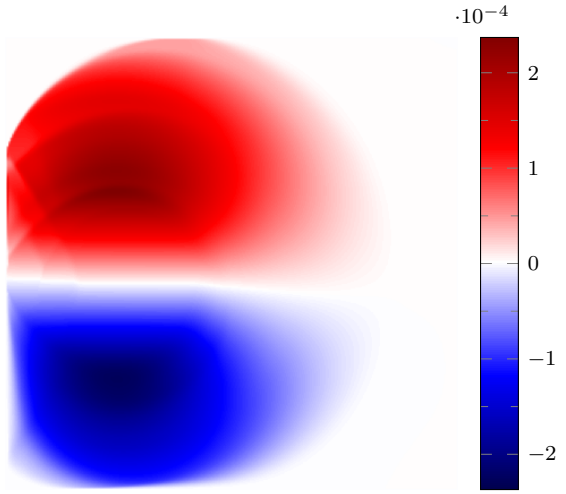


(d) Sea ice concentration after 20 time steps.

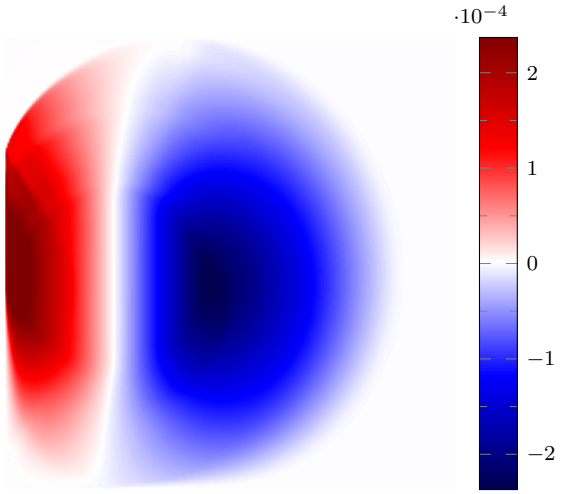


(e) Shear deformation after 20 time steps.

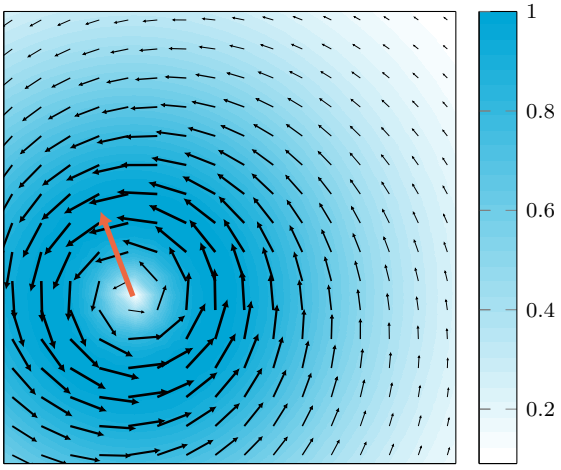
Figure A.3: Gascoigne solution from the training dataset. $H_0 = 0.226$ m and $v_a^{\max} = 14.5$ m/s.



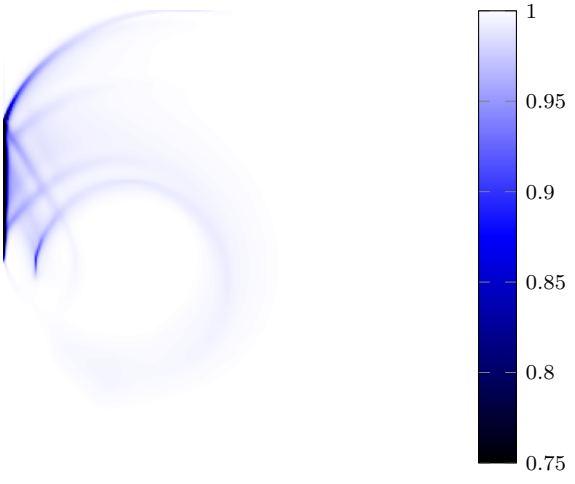
(a) Sea ice velocity in the x-direction after 20 time steps.



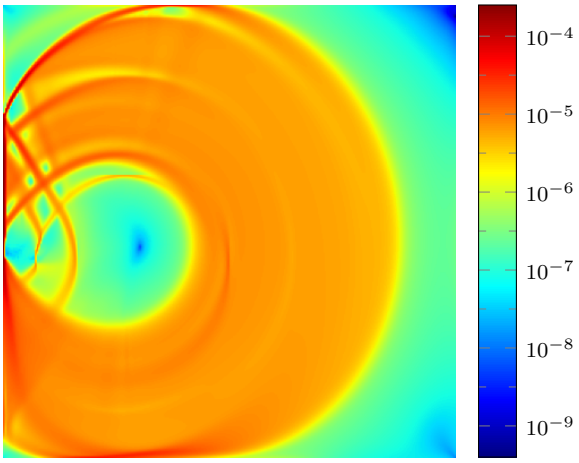
(b) Sea ice velocity in the y-direction after 20 time steps.



(c) Sketch of the Wind velocity field in terms of $v_a^{\max}$.



(d) Sea ice concentration after 20 time steps.



(e) Shear deformation after 20 time steps.

Figure A.4: Gascoigne solution from the training dataset. $H_0 = 0.399$ m and $v_a^{\max} = 10.8$ m/s.

B

Time series results

The results of the recurrent application of the optimised MSE+SRE U-Net. The figures show the results for time steps $t = 5, 10, 15, 20, 25, 30, 50, 90$. Figure B.1 shows the x-component of the velocities, Figure B.2 shows the change in velocity in the current time step, Figure B.3 shows the concentrations, and Figure B.4 shows the shear deformations.

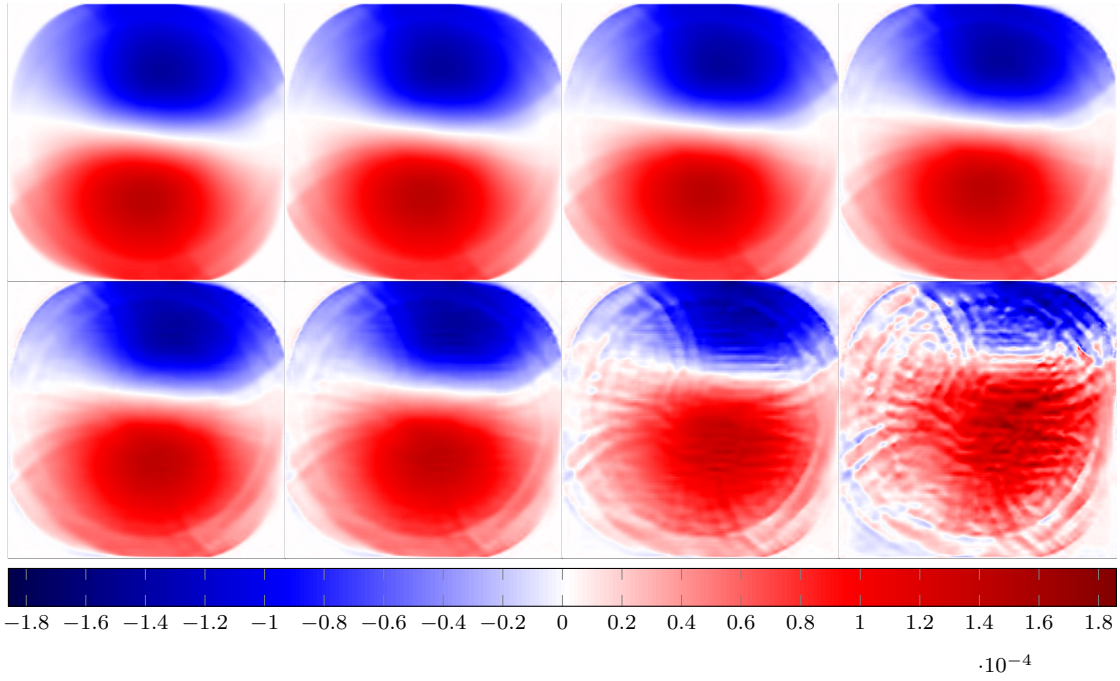


Figure B.1: x-component of the velocity for time steps $t = 5, 10, 15, 20, 25, 30, 50, 90$.

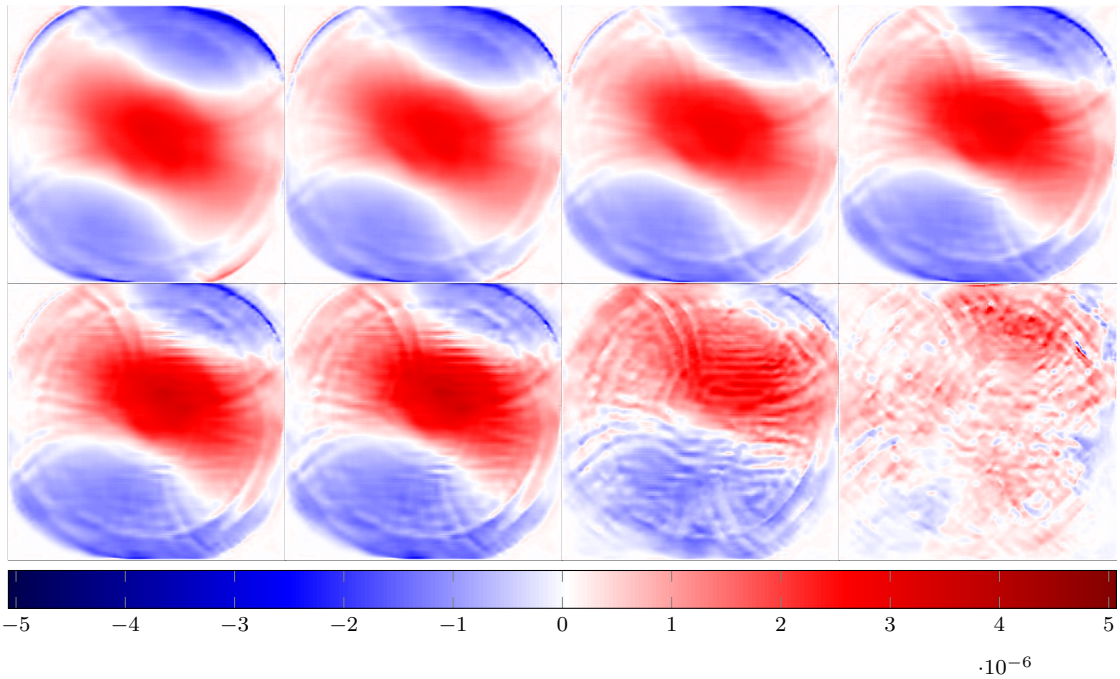


Figure B.2: Change in velocities since the last time step, for time steps $t = 5, 10, 15, 20, 25, 30, 50, 90$.

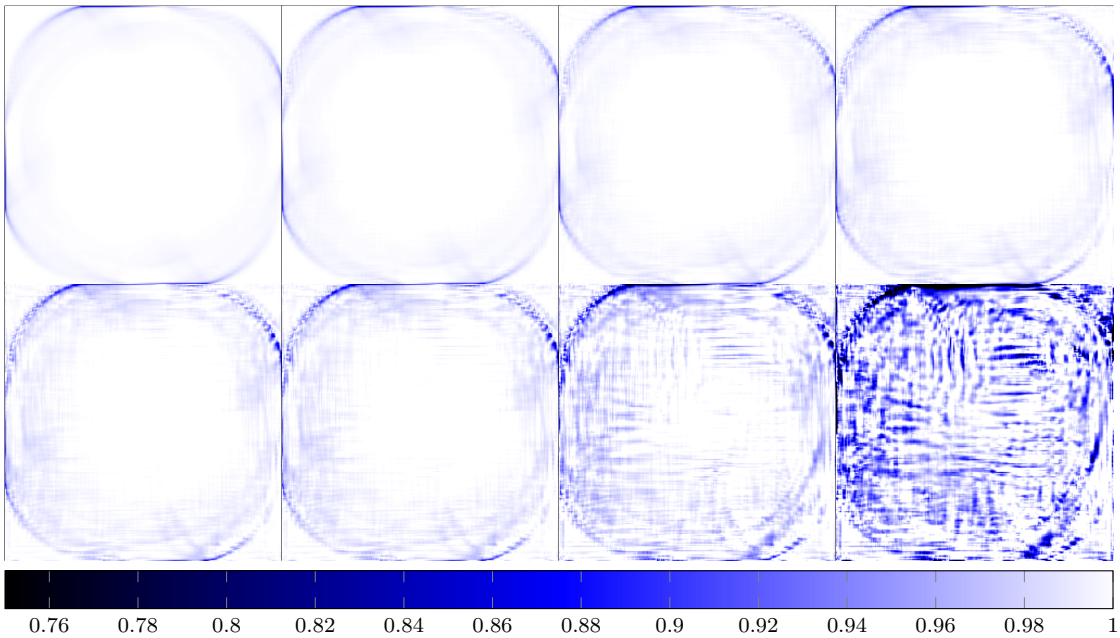


Figure B.3: Sea ice concentrations for time steps $t = 5, 10, 15, 20, 25, 30, 50, 90$.

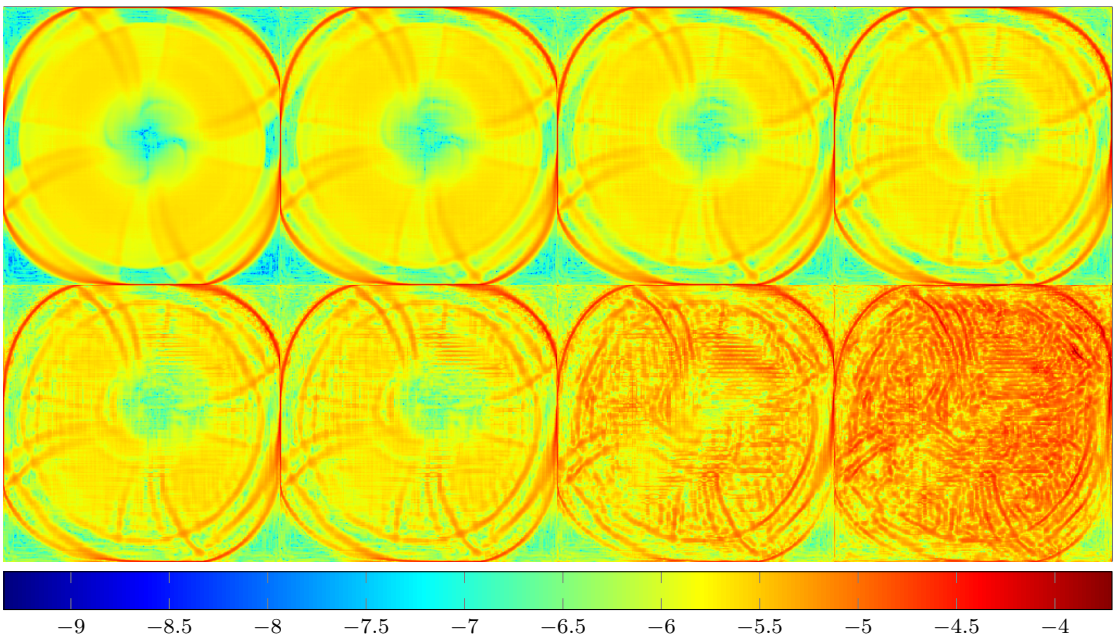


Figure B.4: Shear deformation for time steps $t = 5, 10, 15, 20, 25, 30, 50, 90$, plot using logarithmic color scales.