Physics-Informed Neural Networks in Probabilistic Spatio-Temporal Modelling

Tuukka Himanka

School of Science

Thesis submitted for examination for the degree of Master of Science in Technology.

Helsinki 14.2.2024

Supervisor

D.Sc. (Tech.) Harri Hakula

Advisors

Prof. Marko Laine

Ph.D. Olle Räty

Aalto University
School of Science

Aalto University
School of Science

Aalto University, P.O. BOX 11000, 00076 AALTO
www.aalto.fi
Abstract of the master's thesis

Author Tuukka Himanka

Title Physics-Informed Neural Networks in Probabilistic Spatio-Temporal Modelling

Degree programme Mathematics and Operations Research

Major Mathematics **Code of major** SCI3054

Supervisor D.Sc. (Tech.) Harri Hakula

Advisors Prof. Marko Laine, Ph.D. Olle Räty

Date 14.2.2024 **Number of pages** 52+6 **Language** English

Abstract

Physics-informed neural networks have proved themselves as valuable supplements to classical mathematical models. Compared to regular physics-free neural networks, boundary conditions can be better enforced in the physics-informed variants. The computational efficiency of neural networks makes them feasible for many applications not possible with classical methods.

Most real-world observations and mathematical models are subject to uncertainties. The Kalman filter is a solution to the uncertainty quantification to problems of uncertain observations and models. The ensemble Kalman filter provides an efficient Monte Carlo approximation when dealing with large data vectors.

In this thesis, a neural network is constructed to infer velocity fields within the advection equation. The model is trained with meteorological cloud data. Despite the intrinsic difficulties of the training data, the neural network model is shown to learn the dynamics of the advection equation even with simpler synthetic data. In addition, the neural network presents comparable results compared to a commonly used method for optical flow estimation.

The network is used as a state process operator in the ensemble Kalman filter. A simulated setting of partial observations with an attempt to produce complete state ensembles is tested. The network functions expectedly within the Kalman filter model.

Keywords Physics-informed neural networks, Uncertainty quantification, Bayesian filtering, Weather modelling

Aalto University
School of Science

Aalto-yliopisto, PL 11000, 00076 AALTO
www.aalto.fi
Diplomityön tiivistelmä

Tekijä Tuukka Himanka

Työn nimi Fysiikkainformoidut neuroverkot todennäköisyyspohjaisessa spatiotemporaalisessa mallinnuksessa

Koulutusohjelma Mathematics and Operations Researach

Pääaine Mathematics **Pääaineen koodi** SCI3054

Työn valvoja TkT Harri Hakula

Työn ohjaaja Prof. Marko Laine, FT Olle Räty

Päivämäärä 14.2.2024 **Sivumäärä** 52+6 **Kieli** Englanti

Tiivistelmä

Fysiikkainformoidut neuroverkot ovat osoittautuneet hyödyllisiksi työkaluiksi klassisten matemaattisten mallien rinnalle. Tavallisiin neuroverkkoihin verrattuna, fysiikkainformoitujen neuroverkkojen ennusteita pystyy rajoittamaan paremmin reunaehdoilla. Neuroverkkomallien laskennallinen tehokkuus mahdollistaa uusia nopeutta vaativia sovelluskohteita.

Havaintoihin ja malleihin liittyy usein epävarmuuksia. Kalman-suodin on yksi ratkaisu epävarmuuden arviointiin epävarmojen havaintojen ja mallien ongelmissa. Parven Kalman-suodin on tehokas Monte Carlo -tyyppinen approksimaatio perinteisestä Kalman-suotimesta, jota tarvitaan suurten datavektoreiden hallintaan.

Tässä diplomityössä rakennetaan neuroverkko, joka tunnistaa advektioyhtälön liikekenttiä datasta. Neuroverkkomalli koulutetaan meterologisella pilvidatalla. Koulutusdatan vaikeuksista huolimatta, sillä koulutettu neuroverkko yleistyy odotetusti myös yksinkertaisempaan synteettiseen testidataan. Lisäksi neuroverkkomallin ennustuskyky on verrattavissa yleisesti käytettyyn optisen virtauksen menetelmään.

Neuroverkkoa testaan tilaoperaattorina parven Kalman-suotimessa. Testinä käytetään simuloitua tilannetta puuttuvien havaintojen täyttämiseksi. Neuroverkko toimii odotetusti myös Kalman-suodin -mallin osana.

Avainsanat Fysiikkainformoitu neuroverkko, Epävarmuuden määritys, Bayesilainen suodatus, Säämallinnus

Aalto University
School of Science

Aalto-universitetet, PB 11000, 00076 AALTO
www.aalto.fi
Sammandrag av diplomarbetet

Författare Tuukka Himanka

Titel Fysikinformerade Neuronnät i Probabilistisk Rumstemporal Modellering

Utbildningsprogram Mathematics and Operations Researach

Huvudämne Mathematics **Huvudämnets kod** SCI3054

Övervakare TkD Harri Hakula

Handledare Prof. Marko Laine, FD Olle Räty

Datum 14.2.2024 **Sidantal** 52+6 **Språk** Engelska

Sammandrag

Fysikinformerade neuronnät har visat sig vara värdefulla komplement till klassiska matematiska modeller. Jämfört med vanliga fysikfria neuronnät kan randvillkor bättre upprätthållas i de fysikinformerade varianterna. Neuronnätens beräkningsmässiga effektivitet gör dem genomförbara för många tillämpningar som inte är möjliga med klassiska metoder.

De flesta observationer i den verkliga världen och matematiska modeller är föremål för osäkerheter. Kalmanfiltret är en lösning för att kvantifiera osäkerhet i problem med osäkra observationer och modeller. Ensemble-Kalmanfiltret ger en effektiv Monte Carlo-approximation när man hanterar stora datavektorer.

I denna avhandling konstrueras ett neuronnät för att härleda hastighetsfält inom advektion ekvationen. Modellen tränas med meteorologiska molndata. Trots de inneboende svårigheterna med träningsdatan visar sig neuronnätsmodellen kunna lära sig dynamiken i advektion ekvationen även med enklare syntetiska data. Dessutom presenterar neuronnätet jämförbara resultat jämfört med en vanligt använd metod för skattning av optiskt flöde.

Nätverket används som en tillstånds processoperator i ensemble Kalman filtret. En simulerad inställning av partiella observationer med ett försök att producera kompletta tillståndsensembler testas. Nätverket fungerar som förväntat inom Kalman filter modellen.

Nyckelord Fysikinformerade neuronnät, Osäkerhetskvantifiering, Bayesisk filtrering, Vädermodellering

Contents

1 Introduction

This thesis studies the validity of physics-informed neural networks in a Bayesian filtering setting. Neural networks have proved themselves an important tool in the modelling of high-dimensional and complex data. Physics-informed neural networks that are constructed with prior knowledge of physical models provide a way to constrain a neural network's predictions. Placing such network in a Bayesian filtering scheme allows considering uncertain data as probability distributions.

Physics-informed neural networks cover a wide range of possible applications from solvers to partial differential equations [3] to inverse problem reconstructors [24], and physics-based simulation models such as the interest of this work. Constraining a neural network by physical models provides many benefits compared to regular networks such as the ability to capture the essential information quicker with less dependence on a specific set of data. Compared to many classical mathematical models, neural network models offer computational efficiency essential in many applications. On the flip side, it is often difficult to impose boundary conditions on neural networks and unexpected predictions are sometimes difficult to prevent.

In real-world problems, complete and direct measurements from a quantity of interest are not always available. But even then, uncertain partial measurements from one or more related quantities that are easier to measure may provide useful information. Bayesian filtering techniques provide a powerful method in such cases to estimate the state of the dynamical system in terms of the actual quantity of interest as probability distributions.

In this work, a neural network is constructed to predict motion from two-dimensional data, and it is used as a state propagator in an ensemble-based Kalman filter. The work is a part of a research project in the Meteorological Research Applications group at the Finnish Meteorological Institute. Consequently, the model is trained and tested on meteorological satellite data concerning the physical properties of clouds. However, the model is designed to learn the general advection model dynamics instead of the specific data. The two-dimensional advection equation that models the transportation of substance according to a vector field is chosen due to the model's simplicity and suitability for short-term modelling of cloud behaviour.

In meteorological short-term forecasting or nowcasting, the so-called physics-free neural networks have been extensively researched in, e.g. precipitation forecasts. [2, 31] While these methods have shown some efficiency in forecasting applications, the neural networks often lack interpretability and require large amounts of training data. As current physical numerical weather prediction models lack the ability to generate instantaneous high-resolution predictions for clouds, the currently used methods are based on crude mathematical models such as the advection equation. A hybrid approach of the physics-free neural networks and current advection-based models is taken to train a neural network to learn the temporal evolution of the advection equation. While well-behaving methods exist for this specific advection modelling problem, the neural network could be a more efficient and straightforward alternative for larger data sets. The successful development of a neural network for the advection model where good benchmarks exist could also pave the way for

models in such applications where well-performing methods do not exist yet. Similar work on physics-informed advection-based neural networks has previously been done in [37, 8] with sea surface temperature data. However, in this work, the objective is to generate smooth velocity fields to better meet the requirements of small-scale cloud data. As already experimented in [37], the neural network model is integrated with the Kalman filter model. However, instead of the probabilistic forecasting objective, the emphasis is on producing optimal state estimates of the system with uncertain and incomplete observations.

The main contributions of this thesis are designing a physics-informed neural network to produce smooth velocity fields and integrating it into a high-dimensional ensemble-based Kalman filtering scheme. The scope of this work is primarily on the exploration and validation of neural network-assisted physics-based modelling. Consequently, the neural network design is mainly motivated by general data-independent physical interpretability with less focus on performance-based tuning. The neural network trained with real-world satellite cloud data is compared against an established classical method used in nowcasting applications using a test partition of the dataset. While not being able to exceed the benchmark accuracy, the neural network still exhibits promising results. Moreover, the Kalman filter is set up in a simulated setting with missing observations to validate the proposed framework visually.

The thesis unfolds as follows. First, some background material on the theory behind the used methods is presented. The background section consists of three parts: the characteristics of spatio-temporal data, theoretical background of neural networks, and Bayesian filtering theory. Then the proposed model is presented as a general model for partial and uncertain advective spatio-temporal data. Finally, numerical experiments and verification of the trained model is presented with both the cloud data and synthetic test images separately for the neural network model and the Kalman filter application. In the very last section, an overall summary of the work is discussed, along with possible directions for further research stemming from the findings of this work.

2 Theoretical Background

Some underlying principles of the theoretical topics behind this thesis are presented in this section, which is divided into three parts.

The first part covers the principles and handling of spatio-temporal data, with an emphasis on modelling it through the advection-diffusion model. Subsequently, an introduction to neural network theory is presented. Finally, the concepts of Bayesian filtering are covered, and the classical Kalman filter is presented and extended to an ensemble-based version.

By reviewing these concepts, this chapter aims to provide a theoretical foundation to understanding the proposed model's underlying mechanisms presented in subsequent sections.

2.1 Spatio-temporal Data

This subsection covers some basic principles of spatio-temporal data and the dynamic modelling of such data with the advection-diffusion equation. Spatio-temporal data analysis is an integral part of research across many disciplines. Spatio-temporal data is characterised by having both spatial and temporal components that carry information across space and over time. Modelling these datasets requires thus observing and understanding how the phenomenon of interest evolves over time in many locations.

2.1.1 Defining Spatio-Temporal Data

A real-valued spatio-temporal domain $\Omega \times I$ is the Cartesian product of the spatial domain $\Omega \subset \mathbb{R}^n$ and the temporal domain $I \subset \mathbb{R}^1$. An arbitrary spatio-temporal process Y over the spatio-temporal domain can be written as

$$\{Y(x,t) \,|\, x \in \Omega,\, t \in I\}\,.$$

Naturally, each of the domains can also be discrete by the nature of the domains or due to discrete measurements.

This formulation only requires the spatial and temporal domains to be independent and is thus general enough for most temporally evolving data. Dynamic models are often used to explain the evolution of variables of spatio-temporal data in time. This information is usually described with some partial differential equations, such as the advection-diffusion equation.

2.1.2 Advection-Diffusion Model

At the core of the model of this thesis is the advection-diffusion equation, which describes how the quantity of interest is transported and dispersed in a medium. A concrete example of spatio-temporal data that could modelled with the advection-diffusion model is air pollution data that tracks the concentration of pollutants across different areas over time. The pollutants are advected by winds and diffused

outwards of high concentrations. A simplified example of data evolving through the advection-diffusion process is shown in Figure 2.1.

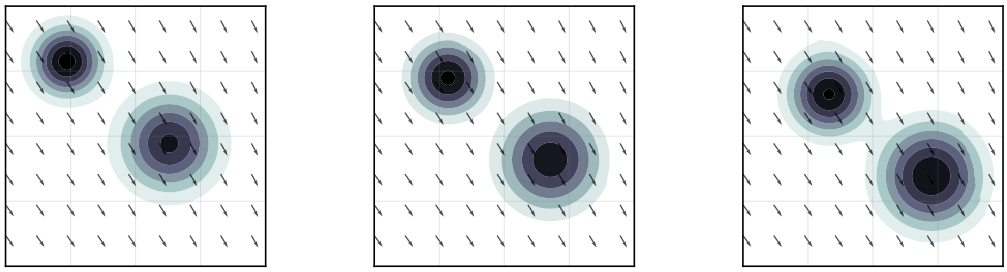

Figure 2.1: Advection-diffusion process with two Gaussian kernels. The objects are transported according to the velocity field and diffused outwards with a constant diffusion parameter.

The advection-diffusion equation consists of an advection term that describes the transportation of property and a diffusion term that describes the diffusion of property. Consider a spatio-temporal function $u : \Omega \times I \to \mathbb{R}$ representing some quantity on a spatial domain $\Omega \subset \mathbb{R}^n$, and temporal domain $I = (0, T)$. The plain advection equation is defined as

$$\frac{\partial u}{\partial t} = -\mathbf{F}\nabla u, \tag{2.1}$$

where $\mathbf{F} = \mathbf{F}(x, t)$ is the vector-valued velocity field that transports u forward in time.

The diffusion equation is

$$\frac{\partial u}{\partial t} = \nabla \cdot (D\nabla u), \tag{2.2}$$

with $D = D(x, t)$ being the diffusion coefficient. In practice, D is often assumed to be spatially constant, simplifying the diffusion term $\nabla \cdot (D\nabla u)$ to $D\Delta u$ and then diffusion equation (2.2) recovers to the heat equation.

Combining (2.1) and (2.2) gives the full advection-diffusion equation

$$\frac{\partial u}{\partial t} = \nabla \cdot (D\nabla u) - \mathbf{F}\nabla u. \tag{2.3}$$

If the velocity field $\mathbf{F}$ and the diffusion coefficient D were known, and the model was perfect, u could be solved given some initial condition $u(\cdot, 0) = u_0(\cdot)$ and some appropriate boundary conditions. However, measuring advection and diffusion parameters is often not possible and estimates $\overline{\mathbf{F}}$ and $\overline{D}$ need to be estimated, e.g. from data. Many uncertainties are often involved in the parameter estimation, but also in the model selection. The approximate advection-diffusion model

$$\frac{\partial u}{\partial t} = \nabla \cdot (\overline{D}\nabla u) - \overline{\mathbf{F}}\nabla u + e \tag{2.4}$$

needs to be then considered with the (hopefully small) modelling error $e = e(x, t)$. In applications, the model also needs to be discretised in time and space so that each time step represents one measurement point, and in space, so that Ω becomes a grid in which the measurements are taken at some resolution.

In the air pollution example the motivation for computing the estimates $\overline{\mathbf{F}}$ and $\overline{D}$ would be that assuming the underlying parameters change little in time, i.e. $\frac{\partial \mathbf{F}(x,t)}{\partial t} \approx 0$ and $\frac{\partial D(x,t)}{\partial t} \approx 0$, the parameters can be utilised to transport air pollution images to future, thus producing forecasts.

2.2 Neural Networks

This subsection covers the basics of neural networks, starting from the McCulloch-Pitts Neuron and training it to backpropagation to the modern convolutional neural networks. As the material of this subsection also covers the elementary underlying principles of neural networks, it is beyond the bare minimum needed to understand the results of this work, but still provides some context for later discussions.

Artificial neural networks draw inspiration from the neural structures in the human brain and are a prominent part of modern machine learning methods. As computing capabilities have seen a dramatic increase, more complex neurals have become feasible and thus more useful in more applications. Substantial success in using neural networks has been found in domains such as image classification, speech and text recognition, object detection, and automation of diverse tasks. In this work, a neural network is adapted to learn the dynamics of the advection-diffusion equation (2.3).

Over the history of neural networks since their inception in the 1940s as a mathematical model for nervous activity [22, 14], they have undergone many transformations in their architectures and capabilities. In the 1950s to 1970s, the first concepts were modelled with a single neuron. Then the application of backpropagation enabled optimising networks of multiple layers of neurons, laying the foundations for the modern research of neural networks. The most recent wave of neural network research from the 2000s was spurred by powerful parallel computing performance that has enabled the training of networks capable of learning increasingly complex phenomena.

2.2.1 Neurons and Activation Functions

The McCulloch-Pitts neuron from [22] is the first mathematical model of neurons and is the fundamental building block for any neural network. It consists of two principal components: an aggregation unit that processes an input vector, and an activation function that generates an output based on the aggregation. The operational structure of the McCulloch-Pitts Neuron is illustrated in Figure 2.2.

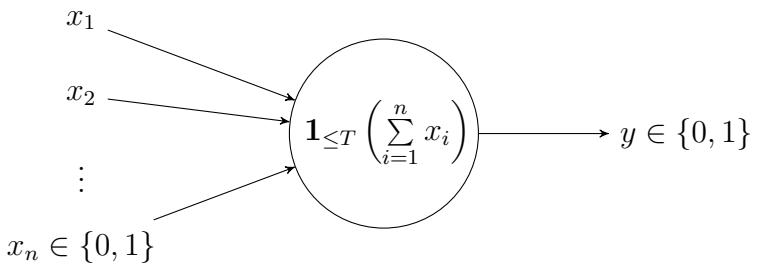

Figure 2.2: McCulloch-Pitts Neuron takes a binary input vector, evaluates its sum and determines the binary output by comparing the sum to a given threshold.

The aggregation function of the McCulloch-Pitts Neuron $f_{\text{McCulloch-Pitts}} : \{0,1\}^n \to \mathbb{N}$,

$$f_{\text{McCulloch-Pitts}}(x) = \sum_{i=1}^{n} x_i$$

computes the sum of an n-dimensional binary input vector x. The activation function ϕ subsequently determines the neuron's output based on a threshold value T,

$$y = \phi(f(x)) = \mathbf{1}_{\geq T}(f(x)) = \begin{cases} 1 & , \quad \text{if } f(x) \geq T \\ 0 & , \quad \text{if } f(x) < T. \end{cases}$$

While this is an attractive model in its simplicity, it has a major limitation. As the input vectors and output values are binary, the range of possible applications is narrow.

A few years later, Donald Hebb [14] proposed that the proximity between inputs and other neurons should have an influence in the output. This concept was applied to the neuron model by Frank Rosenblatt [29], who developed the much more flexible perceptron model. The perceptron model extends the McCulloch-Pitts Neuron by allowing any input values to the aggregation unit. The perceptron aggregation function $f_{\text{perceptron}} : \mathbb{R}^n \to \mathbb{R}$ is

$$f_{\text{perceptron}}(x) = \sum_{i=1}^{n} w_i x_i + b,$$

where the input vector is weighted by the weight vector $w \in \mathbb{R}^n$ and bias term $b \in \mathbb{R}$ is added to the output. The structure of the perceptron is depicted in Figure 2.3.

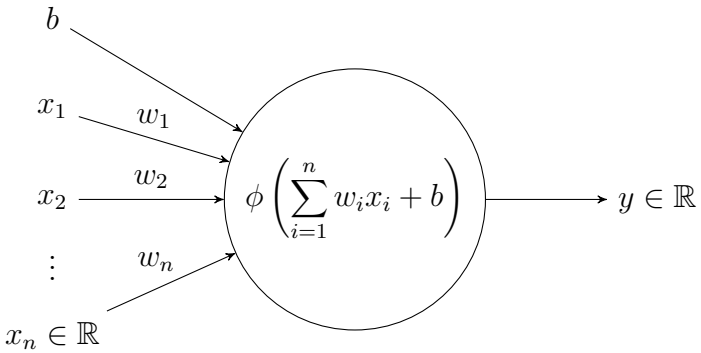

Figure 2.3: Perceptron allows any input values that are weighted with w and adds a bias term β. With a more sophisticated activation function ϕ, the output values can also be anything.

With a more advanced activation function, nonlinearity can be introduced to the otherwise linear networks more efficiently. Commonly used activation functions include the Rectified Linear Unit (ReLU), Leaky ReLU, Sigmoid, and Softmax functions. These activation functions are visualised in Figure 2.4. The choice of activation function comes with technicalities crucial for training a neural network, such as the range of possible values and behaviour of their derivatives.

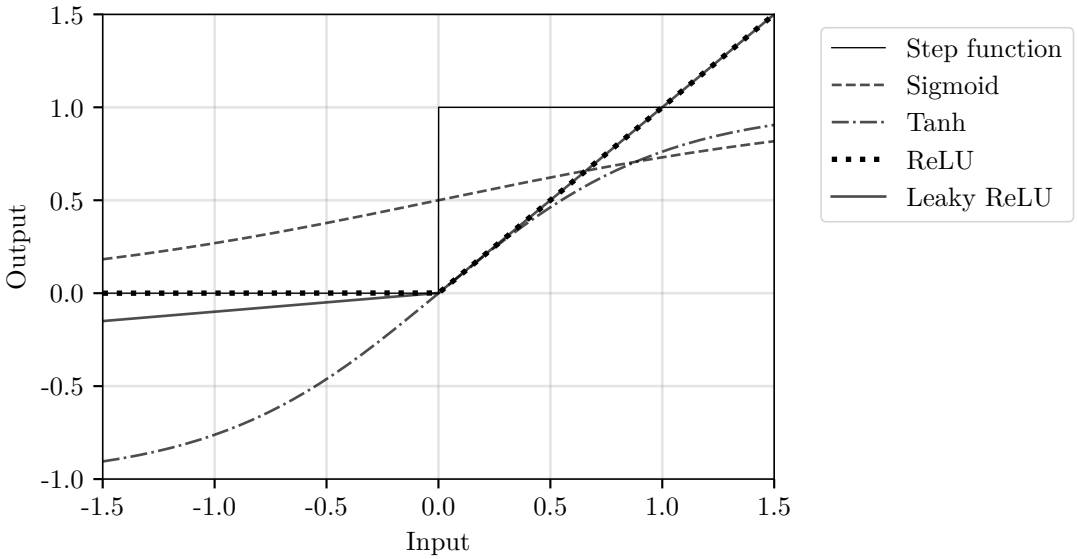

Figure 2.4: Different commonly used activation functions in neural networks. The activation functions control the range of values going through a network, and introduce nonlinearity to the otherwise linear networks.

The single-neuron model is only applicable to extremely simple tasks. Therefore, the so-called multilayer perceptron (MLP) was developed in which several perceptrons are combined as layers and applied sequentially. The MLP structure is shown in Figure 2.5.

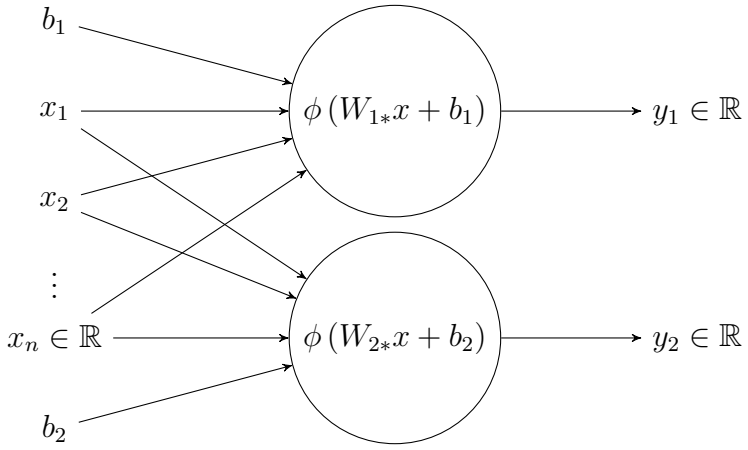

Figure 2.5: A schematic illustration of multiplayer perceptron. The model has one hidden layer with two perceptrons, and the output is two-dimensional. By composing multiple perceptrons together, a neural network uses more parameters to learn more complex patterns from data

Formally, the MLP aggregation function $f^{(l)}_{\text{MLP}} : \mathbb{R}^n \to \mathbb{R}^m$ for layer l is

$$f^{(l)}_{\text{MLP}}(x; W, b) = Wx + b.$$

Rows of the weight matrix $W \in \mathbb{R}^{n \times m}$ contain the weights and the bias vector $b \in \mathbb{R}^m$ has the bias terms of the neurons in the layer. The activation function is then applied element-wise for the output vector of $f^{(l)}_{\text{MLP}}$. The entire neural network is then a composition of all the individual layers.

MLPs are related to the well-known Universal Approximation Theorem presented at the turn of the 1990s. [7, 17, 16] The theorem states that an MLP network with at least one hidden layer and any squashing activation function can approximate any Borel measurable function between finite-dimensional spaces. While the theorem might sound compelling, the number of neurons is generally not guaranteed to be bounded. Therefore, this may lead to unfeasibly large networks. Furthermore, the theorem does not state that any suitable function can be learnt, only that it can be represented.

2.2.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) have emerged as an instrumental class of models, especially in tasks involving image data, e.g. image classification. CNNs take advantage of the convolution operations's ability to identify critical features and structures from high-dimensional data effectively.

The convolution operation is the fundamental building block of CNNs. The n-dimensional convolution of two compactly supported functions $f, g : \mathbb{R}^n \to \mathbb{R}$ is defined by

$$(f * g)(x) = \int_{\mathbb{R}^n} f(y)g(x - y)\, dy.$$

For practical neural network applications, the operation needs to be discretised. The discrete convolution for compactly supported $f, g : \mathbb{Z}^n \to \mathbb{R}$ is given by

$$(f * g)[x] = \sum_{y \in \mathbb{N}^n} f[y]g[x - y].$$

Neural networks usually treat f as some input data and g as convolution kernel that is the learnable model parameter.

Practical neural network libraries implement convolution layers as a similar cross-correlation function [12]

$$(f * g)[x] = \sum_{y \in \mathbb{N}^n} f[x]g[y + x].$$

The cross-correlation differs from the convolution by not flipping the kernel. Consequently, the cross-correlation is not a commutative operation unlike the convolution as can be seen with a change of variables. As neural network filters are learnt from data, flipping or not flipping the kernel does not have an effect on the results. For practical and historical reasons, the convolution term is anyways used in the neural networks field.

Due to dimensionality of real-world data and computational reasons, the of the neural network convolution dimension seldom exceeds 3. Particularly in the case of image data, two-dimensional convolutions are considered. Therefore, only the two-dimensional case is considered from here on.

CNNs employ many small convolutions as a replacement for the dot products performed in the MLP neurons. With image inputs, the convolution kernels often have sizes in the range of 3×3 to 9×9. In Figure 2.6, a convolution of a larger matrix with a 3×3 kernel is visualised. Smaller kernels are usually preferred due to the reduced number of model parameters and lower computational cost while retaining the capacity to learn patterns from data. The input is convolved with a large number of these small kernels individually and hence increasing the depth of the input. A typical number of convolution filters in the initial layer could be 32 or 64, and then increasing number of filters for the subsequent layers to allow the model to learn more complex representations.

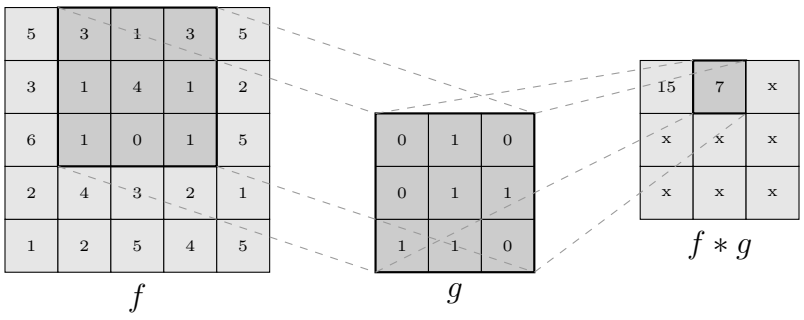

Figure 2.6: Two-dimensional convolution with a 3×3 kernel. The convolution operation can be used to reduce the dimensions of the input while retaining essential features with a suitable convolution kernel.

CNNs typically employ pooling layers for downsampling the spatial dimensions of the input by combining multiple values into one, usually by taking the average (average pooling) or maximum (max pooling) values. Additionally, activation functions are applied after convolution layers, along with some normalisations for the data in preparation for the next layer. Convolutions with pooling and normalisations are rarely the sole type of layers in CNN architectures. Typically, at least the very last layers are similar to the fully connected dot-product layers found in the MLP. These output layers are primarily for final classification and re-shaping the output of the network.

2.2.3 Optimising Neural Network Parameters

Computing many of the introduced numerical operations with arbitrary parameters has little use in practice. Therefore, a neural network has to undergo a training process to optimise the parameters over a training dataset.

Learning tasks are classically classified into three categories: supervised, unsupervised, and reinforcement learning. In supervised learning, data is labelled, and the model learns by comparing its output to the correct label. Conversely, in unsupervised learning, the model has to learn patterns in data without correct labels. Combinations of supervised and unsupervised learning also exist, called semi-supervised learning. An example would be various forms of clustering tasks. In reinforcement learning, a model is trained to take some actions based on reward feedback, and it learns to perform actions that maximise the cumulative reward.

All neural network learning tasks ultimately lead to an optimisation problem. Optimising neural network parameters bears resemblance to other numerical optimisation methods. But due to the huge number of parameters and the non-convex nature of the optimisation problem, training a neural network is a lengthy and complex task. The optimisation problem of a supervised learning task can be written out as a minimisation of a real-valued cost functional J, which represents the expected value of a loss function, with respect to the network parameters θ:

$$\min_{\theta} J(\theta) = \mathbb{E}_{(x,y)\sim\hat{\pi}_{\text{data}}} L(f(x;\theta), y) = \frac{1}{n}\sum_{i=1}^{n} L(f(x^{(i)};\theta), y^{(i)}).$$

Here, f denotes the neural network function that generates predictions for some input data x, drawn n times from the empirical training data distribution $\hat{\pi}_{\text{data}}$ given the network parameters. L is the loss function which compares the network prediction to the true label y. This is easily modified to unsupervised learning by excluding y from the equation, in which case the loss function has to use another method to compare correctness.

Choosing an appropriate loss function is a crucial part of model training. The family of loss functions suitable for neural networks is large and dependent on the type of data. While the definitions of different loss functions look very different, the choice is typically made based on how large prediction errors should be valued compared to small ones. Perhaps the most commonly used and well-known loss function is the mean squared error (MSE), which computes the mean of squared

prediction errors between an n-dimensional target value y and predicted value $\hat{y}$:

$$L_{\text{MSE}}(\hat{y}, y) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 .$$

The standard loss functions are sometimes augmented with application-specific quantities. For instance, in the realm of physics-informed neural networks, it is customary to add some physical quantities with some weights to the loss function. Theoretically, these quantities are then minimised in the training process, but enforcing it is difficult, and choosing appropriate weights is non-trivial. Consequently, hard boundary conditions are hard to meet in this manner.

Many optimisation algorithms with neural networks are based on Stochastic gradient descent (SGD). SGD is an adaptation of the classical gradient descent algorithm. The basic concept of adjusting the decision variable in the direction of the loss function's gradient, scaled by a constant step size, is the same. The stochastic version primarily differs from the original one in that it estimates gradients from a subset of the data, thereby reducing the computational cost.

For a true stochastic optimisation method, only one sample would be taken, but it is often more practical to take a small number of samples, called mini-batching. The SGD algorithm with mini-batching is detailed in Algorithm 1.

Algorithm 1 Stochastic gradient descent

initialise step size $\eta > 0$, initial parameter θ
while stopping criterion not met **do** ▷ E.g. non-decreasing losses
  Sample a mini-batch $\{(x^{(i)}, y^{(i)})\}_{i=1}^{m}$
  $d = -\frac{1}{m} \nabla_\theta \sum_{i=1}^{m} L(f(x^{(i)}; \theta), y^{(i)})$ ▷ Compute search direction
  $\theta = \theta + \eta d$ ▷ Update parameters
end while
return θ

The stochastic nature of the SGD algorithm introduces noise into the computed losses also in an optimum. Consequently, the algorithm would never stabilise at the optimum but keep altering parameters. A usual mitigation for this is to decrease the step size during training according to some schedule. As the technical details related to step size scheduling techniques are outside the scope of this work. E.g. [12] provides an in-depth review.

While SGD serves as a standard first-order algorithm, more advanced algorithms are also often employed. SGD with momentum, for instance, supplements the current search direction with a fraction of the previous one to robustify the search directions. Some other popular optimisation algorithms used for neural network training include RMSProp, AdaDelta, and Adam, with the latter combining the benefits of RMSProp and AdaGrad. Again, [12] provides detailed explanations.

The different optimisation algorithm options each offer distinct advantages, and there is no one that would beat all others. For instance, some have adaptive step sizing, eliminating the need for step size scheduling. However, the common theme

in all of them is the iterative update of the model parameters by moving in the direction of the gradient of the loss function. Hence, the next challenge is to find an efficient method for estimating the gradients.

2.2.4 Backpropagation

Training a neural network begins with computing the network's predictions using inputs from the training dataset. This part is often called forward propagation. However, in all optimisation algorithms, the gradient of the loss function with respect to the model parameters is needed at some point. Conventional approaches for numerical gradient estimation are computationally infeasible for neural networks consisting of many trainable parameters. Therefore, a technique called backpropagation is needed for efficient gradient estimation.

Reverse accumulation as an automatic differentiation technique for MLP networks was first introduced in the 1970s and 1980s and became later known as backpropagation. It was first introduced by Paul Werbos in their 1974 PhD thesis [36] but gained popularity a decade later through the works of David Rumelhart and others [30].

These works facilitated the efficient training of more complex networks due to the precise gradient estimation of neural network parameters. However, reverse accumulating gradients were studied independently by multiple others even earlier. Perhaps the most notable example is the work of the Finnish mathematician Seppo Linnainmaa in their MSc thesis from 1970 [20]. However, in Linnainmaa's work, the concept of reverse accumulation was studied in the context of cumulative floating point errors.

The core principle behind the backpropagation algorithm is the recursive application of the chain rule of calculus. The chain rule states that for $g : \mathbb{R}^m \to \mathbb{R}^n$ and $f : \mathbb{R}^n \to \mathbb{R}$ with $f(g(x))$ the partial derivatives of f are given by

$$\frac{\partial f}{\partial x_i} = \sum_{j=1}^{n} \frac{\partial f}{\partial (g(x)_j)} \frac{\partial (g(x)_j)}{\partial x_i}.$$

The objective function J of the optimisation problem can be considered a composition of all the individual functions in the network. Subsequently, the chain rule can be used to obtain the partial derivatives with respect to the trainable network parameters.

As the shape and size of the trainable parameters differ for each specific network architecture, the exact formulation of the gradients differs a bit. For the simple MLP case, the formulas are easy to write to see the idea behind the backpropagation algorithm. Recall that the network parameters can be expressed for each layer ℓ as a matrix $\theta^{(\ell)} = \begin{bmatrix} W & b \end{bmatrix}^{(\ell)}$. Then, denote the output of layer ℓ as $z^{(\ell)} = W^{(\ell)} a^{(\ell-1)} + b$ where $a^{(\ell-1)} = \phi^{\ell-1}(z^{(\ell-1)})$ is the output from the activation of the previous layer. The chain rule thus gives for the partial derivative of the objective function w.r.t.

parameter $\theta_{ij}^{(\ell)}$

$$\frac{\partial J}{\partial \theta_{ij}^{(\ell)}} = \sum_k \frac{\partial J}{\partial z_k^{(\ell)}} \frac{\partial z_k^{(\ell)}}{\partial \theta_{ij}^{(\ell)}}. \tag{2.5}$$

For the weight derivatives this is evaluated as

$$\frac{\partial J}{\partial W_{ij}^{(\ell)}} = \frac{\partial J}{\partial z_j^{(\ell)}} a_i^{(\ell-1)},$$

and the bias derives as

$$\frac{\partial J}{\partial b_j^{(\ell)}} = \frac{\partial J}{\partial z_j^{(\ell)}} 1.$$

Thus, (2.5) can be written as

$$\nabla_{\theta^{(\ell)}} J = \begin{bmatrix} z^{(\ell)} \\ 1 \end{bmatrix} (\nabla_{z^{(\ell)}} J)^\top. \tag{2.6}$$

The only thing left then is to find an equation for the gradient term of (2.6) that is often called the local gradient or error. Denote it by $\delta^{(\ell)}$, and apply the chain rule once again,

$$\begin{aligned} \delta_j^{(\ell)} = \frac{\partial J}{\partial z_j^{(\ell)}} &= \sum_k \frac{\partial J}{\partial z_k^{(\ell+1)}} \frac{\partial z_k^{(\ell+1)}}{\partial z_j^{(\ell)}} \\ &= \sum_k \delta_k^{(\ell+1)} \frac{\partial z_k^{(\ell+1)}}{\partial z_j^{(\ell)}}, \end{aligned}$$

from which

$$\frac{\partial z_k^{(\ell+1)}}{\partial z_j^{(\ell)}} = \frac{\partial}{\partial z_j^{(\ell)}} \sum_i W_{ki}^{(\ell+1)} \phi^{(\ell)}(z_i^{(\ell)}) + b_k^{(\ell+1)} = W_{kj}^{(\ell+1)} \phi^{(\ell)\prime}(z_j^{(\ell)}).$$

Thus, the error is recursively

$$\delta_j^{(\ell)} = \sum_k \delta_k^{(\ell+1)} W_{kj}^{(\ell+1)} \phi^{(\ell)\prime}(z_j^{(\ell)}).$$

For the last layer L, the error can be given more simply,

$$\begin{aligned} \delta_j^{(L)} = \frac{\partial J}{\partial z_j^{(L)}} &= \sum_k \frac{\partial J}{\partial a_k^L} \underbrace{\frac{\partial a_k^L}{\partial z_j^L}}_{=0 \text{ when } k \neq j} \\ &= \frac{\partial J}{\partial a_j^L} \frac{\partial a_j^L}{\partial z_j^L} \\ &= \frac{\partial J}{\partial a_j^L} \phi^{(L)\prime}(z_j^{(L)}). \end{aligned}$$

Now that the neural network construction is defined, there is a way to find the optimal network parameters by computing the gradients, and a complete neural network can be trained. Things need not be this complicated for users of neural networks, as the computations presented are typically encapsulated inside the widely used numerical neural network libraries.

2.3 Bayesian Filtering

This subsection covers the mathematical theory of Bayesian filtering. The Kalman filter is presented as an optimal state estimation algorithm under certain assumptions. Then, it is extended into an ensemble-based version that efficiently handles larger observation and state vectors.

Bayesian filtering refers to the Bayesian way of deriving an optimal filtering problem of estimating the state of a nonstationary system through some observations. This approach becomes valuable when dealing with real-world phenomena where a mathematical model can approximate the temporal evolution of some process, and measurements of the system may be uncertain.

A key application of Bayesian filtering is state estimation, which emerges in many disciplines. It involves estimating the hidden state of a system based on some available indirect and noisy measurements. Bayesian filtering provides a robust mathematical framework to combine prior knowledge about a system's state with new measurements to enhance state estimates over time.

The Bayesian filtering theory emerged at the turn of the 1960s. Perhaps the most famous example being the works of Rudolf E. Kálmán, who published a recursive solution to a linear special case of the optimal state estimation problem in [19]. Meanwhile, in the Soviet Union, Ruslan Stratonovich studied the more general nonlinear Bayesian filtering problem, see e.g. [32, 33]. The simultaneous advances in digital computers allowed the theory to quickly prove itself valuable in many applications such as in the guidance system of the Apollo spacecraft [23].

In the context of this work, Bayesian filtering is employed to estimate uncertainties related to deterministic estimates and for completing incomplete observations.

2.3.1 Bayesian Inference

Bayesian inference refers to the process of updating prior probability estimates in light of more information. The basis of Bayesian inference lies in the Bayes' Theorem.

Theorem 2.1 (Bayes' Theorem)**.** *Assume a known prior probability density function $\pi(x)$ from a random variable X and an observed probability $\pi(y)$ from another random variable Y. Then, the posterior distribution of X given the observed data is given by*

$$\pi(x \,|\, y) = \frac{\pi(x)\pi(y \,|\, x)}{\pi(y)}.$$

In Bayesian inference, the prior distribution is thought to contain the information of some quantity before observing the data. The posterior distribution then contains the updated information after the observation. Bayesian inference revolves around using data to transform the prior distributions into the posterior distributions.

2.3.2 Probabilistic state-space models

The setting in which measurements are assimilated and processed using a model can be formalised as the state-space model. Given inherent uncertainties linked with

measurements and modelling, it is substantially more fruitful to consider probability distributions instead of relying solely on point estimates.

Consider two stochastic processes, $\{X_k\}_{k=0}^{\infty}$ and $\{Y_k\}_{k=0}^{\infty}$, representing the state and measurements of a system, respectively. Figure 2.7 presents the dependency scheme of the processes needed to formulate the state-space model.

$$\begin{array}{rccccccccc} \text{State:} & X_0 & \to & X_1 & \to & X_2 & \to \cdots \to & X_k & \to \cdots \\ & & & \downarrow & & \downarrow & & \downarrow & \\ \text{Observations:} & & & Y_1 & & Y_2 & & Y_k & \end{array}$$

Figure 2.7: Dependency scheme of the state-space model.

In order for the dependencies of the figure to materialise, some formal assumptions about the processes are required:

1. $\{X_k\}_{k=0}^{\infty}$ satisfies the Markov property with respect to its own history and history of the observations:

$$\pi(x_{k+1} \,|\, x_{0:k}, y_{1:k}) = \pi(x_{k+1} \,|\, x_k). \tag{2.7}$$

2. $\{Y_k\}_{k=0}^{\infty}$ satisfies the Markov property with respect to the history of the true state:

$$\pi(y_k \,|\, x_{0:k}) = \pi(y_k \,|\, x_k). \tag{2.8}$$

With these assumptions, the probabilistic state-space model can be defined.

Definition 2.1 (Probabilistic state-space model)**.** *Assume stochastic processes* $\{X_k\}_{k=0}^{\infty}$ *and* $\{Y_k\}_{k=0}^{\infty}$ *meet the previous assumptions. Then, the following conditional probability distributions define the probabilistic state-space model:*

$$X_0 \sim \pi(x_0) \tag{2.9a}$$
$$X_{k+1} \sim \pi(x_{k+1} \,|\, x_k), \quad k = 0, 1, 2, \ldots \tag{2.9b}$$
$$Y_k \sim \pi(y_k \,|\, x_k), \quad k = 1, 2, \ldots. \tag{2.9c}$$

In Definition 2.1, (2.9b) is the model that describes the evolution of the state over time, and (2.9c) is the observation model that describes the distribution of measurements conditioned on the state.

If functions for the state evolution and observations are known, the same can be alternatively written as

$$X_{k+1} = M_{k+1}(X_k, W_{k+1}), \quad k = 0, 1, \ldots \tag{2.10a}$$
$$Y_k = H_k(X_k, V_k), \quad k = 1, 2, \ldots. \tag{2.10b}$$

Now (2.10a) is the state evolution equation where function M_{k+1} gives the next state X_{k+1} given the previous state X_k and the state noise W_{k+1}. The observation

equation (2.10b) describes the observations Y_k as being the output of the observation function H_k with the true state X_k and the observation noise V_k.

A few conditional probability distributions are usually of interest with regards to the probabilistic state-space models, outlined in the following definition.

Definition 2.2. *Assume two stochastic processes $\{X_k\}_{k=0}^{\infty}$ and $\{Y_k\}_{k=0}^{\infty}$ form a probabilistic state-space model. Then, the conditional probability distribution*

- $\pi(x_{k+n} \mid y_{1:k})$, $n > 0$ *is called a prediction distribution.*
- $\pi(x_k \mid y_{1:k})$ *is called a filtering distribution.*
- $\pi(x_k \mid y_{1:})$ *is called a smoothing distribution.*

By estimating these distributions, different kinds of questions can be answered. The prediction distribution is an estimate of future states where observations are yet not available. When the observations arrive, the filtering distribution corrects the prediction distribution according to the observations. After observations of future time steps are available, a state may be estimated given observations at all time steps as with, e.g. historical time series data.

With the state-space model of Definition 2.1, the Bayesian filtering problems can be discussed. The idea behind Bayesian filtering is to recursively update the predictions and states of the model given latest observations. The following theorem presents the two update equations for the Bayesian filtering scheme.

Theorem 2.2 (Bayesian filtering equations)**.** *Let $\{X_k\}_{k=0}^{\infty}$ be a state process and $\{Y_k\}_{k=0}^{\infty}$ an observation process in a state-space model. Then, the following recursive equations for Bayesian filtering steps apply:*

1. *Prediction step:*

$$\pi(x_{k+1} \mid y_{1:k}) = \int \pi(x_{k+1} \mid x_k)\pi(x_k \mid y_{1:k})\, dx_k. \tag{2.11}$$

2. *Observation update step:*

$$\pi(x_{k+1} \mid y_{1:k+1}) = \frac{\pi(y_{k+1} \mid x_{k+1})\pi(x_{k+1}|y_{1:k})}{\pi(y_{k+1} \mid y_{1:k})}, \tag{2.12}$$

with

$$\pi(y_{k+1} \mid y_{1:k}) = \int \pi(y_k \mid x_k)\pi(x_k \mid y_{1:k-1})\, dx_{k+1}.$$

Proof. In Appendix A. □

Theorem 2.2 provides a framework for theoretically estimating probability distributions conditioned on observations. The prediction step equation 2.11, which describes the joint probability distribution of the current and next state conditioned on the latest observations, facilitates probabilistic state estimates. Then the observation update equation 2.12 is just the Bayes formula where $\pi(x_{k+1} \mid y_{1:k})$ is interpreted as the prior information of the state for statistical inversion of a new observation.

2.3.3 Kalman Filter

The Kalman filter provides an elegant closed-form solution for the update equations of Theorem 2.2 in the special case of a linear state-space model with Gaussian additive noise. In this case, the state-space model corresponding to (2.10) becomes

$$X_{k+1} = M_{k+1}X_k + W_{k+1}, \quad W_{k+1} \sim \mathcal{N}(0, Q_{k+1}) \tag{2.13a}$$
$$Y_k = H_k X_k + V_k, \quad V_t \sim \mathcal{N}(0, R_k) \tag{2.13b}$$

with a Gaussian distribution for the initial state X_0.

Under these Gaussian assumptions, probability densities can be given by mean and covariance. The following theorem gives the Kalman filter equations corresponding to Theorem 2.2.

Theorem 2.3 (Kalman filter equations)**.** *Assume a linear Gaussian state-space model (2.13). Let $x_{k\,|\,\ell}$ and $P_{k\,|\,\ell}$ represent the posterior mean and covariance of the state at time k given observations $y_{1:\ell}$. Then, the filtering equations of Theorem 2.2 take closed-form solutions:*

1. *Prediction step: Given*

$$\pi(x_k \,|\, y_{1:k}) \sim \mathcal{N}(x_{k\,|\,k}, P_{k\,|\,k}),$$

a prior distribution of the next state is

$$\pi(x_{k+1} \,|\, y_{1:k}) \sim \mathcal{N}(x_{k+1\,|\,k}, P_{k+1\,|\,k}),$$

where parameters of the distribution are given by

$$x_{k+1\,|\,k} = M_{k+1}x_{k\,|\,k} \tag{2.14a}$$
$$P_{k+1\,|\,k} = M_{k+1}P_{k\,|\,k}M_{k+1}^\top + Q_{k+1}. \tag{2.14b}$$

2. *Observation update step: Given*

$$\pi(x_{k+1} \,|\, y_{1:k}) \sim \mathcal{N}(x_{k+1\,|\,k}, P_{k+1\,|\,k}),$$

a posterior distribution of the next state is

$$\pi(x_{k+1} \,|\, y_{1:k+1}) \sim \mathcal{N}(x_{k+1\,|\,k+1}, P_{k+1\,|\,k+1}).$$

With

$$v_{k+1} = y_{k+1} - H_{k+1}x_{k+1\,|\,k}, \tag{2.15a}$$
$$S_{k+1} = H_{k+1}P_{k+1\,|\,k}H_{k+1}^\top + R_{k+1}, \tag{2.15b}$$
$$K_{k+1} = P_{k+1\,|\,k}H_{k+1}^\top S_{k+1}^{-1}, \tag{2.15c}$$
$$x_{k+1\,|\,k+1} = x_{k+1\,|\,k} + K_{k+1}v_{k+1}, \tag{2.15d}$$
$$P_{k+1\,|\,k+1} = (I - K_{k+1}H_{k+1})P_{k+1\,|\,k}. \tag{2.15e}$$

Proof. In Appendix A. □

While the theorem assumes zero means for noises, the Gaussian distributions can be shifted to accommodate non-zero means. The Kalman filter leverages the property of Gaussian distributions retaining their Gaussian form under affine transformations. With nonlinear state-space models, linear linear approximations can be used to preserve the convenience of Gaussian densities. This version is known as the extended Kalman filter.

Even when the assumptions for the Kalman filter apply, with large state and observation spaces, problems will arise when computing the Kalman filter steps. The matrix update formulas of Theorem 2.3 quickly cross the limits of computational feasibility. One way of tackling the problem is to consider ensembles as representations of the Gaussian densities.

2.3.4 Ensemble Kalman Filter

Although the original Kalman filter offers a robust solution to optimal filtering problems, it fails to efficiently scale for large state and observation spaces. This limitation paved the way for the development of the Ensemble Kalman Filter (EnKF). It was first introduced by Geir Evensen in [10] as a Monte Carlo approach to the Kalman filter. EnKF circumvents the need for large covariance matrices by representing both a state and its uncertainty via an ensemble of state samples.

Formally the filtering distribution at time k is represented as an ensemble $x_{k\,|\,k}^{(1)}, \ldots, x_{k\,|\,k}^{(N)}$ where each $x_{k\,|\,k}^{(i)} \sim \mathcal{N}(x_{k\,|\,k}, P_{k\,|\,k})$. The prior distribution for the next state in the prediction step is obtained by simply propagating the ensemble using the state evolution operator,

$$x_{k+1\,|\,k}^{(i)} = M_{k+1} x_{k\,|\,k}^{(i)} + W_{k+1}, \quad W_{k+1} \sim \mathcal{N}(0, Q_{k+1}), \quad i = 1, \ldots, N.$$

The observation update step of the EnKF has multiple formulations typically categorised into stochastic and deterministic versions. Both versions are introduced here following [1] with a focus on the deterministic EnKF. All EnKF variants derive an estimate of the Kalman gain matrix

$$\hat{K}_{k+1} = \hat{P}_{k+1\,|\,k} H_{k+1}^\top (H_{k+1} \hat{P}_{k+1\,|\,k} H_{k+1}^\top + R_{k+1})^{-1}$$

from the ensemble statistics. $\hat{P}_{k+1\,|\,k}$ estimates the state forecast error covariance $P_{k+1\,|\,k}$ that can be obtained from the ensemble forecast error as

$$\begin{aligned} \hat{P}_{k+1\,|\,k} &= \frac{1}{N-1} \sum_{i=1}^{N} (x_{k+1\,|\,k}^{(i)} - \overline{x}_{k+1\,|\,k})(x_{k+1\,|\,k}^{(i)} - \overline{x}_{k+1\,|\,k})^\top \\ &= X_{k+1\,|\,k} X_{k+1\,|\,k}^\top, \end{aligned}$$

where $\overline{x}$ represents the ensemble mean

$$\overline{x} = \frac{1}{N} \sum_{i=1}^{N} x^{(i)},$$

and columns of $X_{k+1\,|\,k}$ are normalised ensemble anomalies to the mean. Thus, $\hat{K}$ may be expressed as

$$\begin{aligned}\hat{K}_{k+1} &= X_{k+1\,|\,k}X_{k+1\,|\,k}^{\top}H_{k+1}^{\top}(H_{k+1}X_{k+1\,|\,k}X_{k+1\,|\,k}^{\top}H_{k+1}^{\top}+R_{k+1})^{-1}\\ &= X_{k+1\,|\,k}Y_{k+1\,|\,k}^{\top}(Y_{k+1}Y_{k+1}^{\top}+R_{k+1})^{-1}.\end{aligned}$$

The stochastic variant produces a simulated ensemble of observations $\left\{y_{k+1}^{(i)}\right\}_{i=1}^{N}$ by introducing Gaussian noise to form the updated filtering distribution. Specifically, the observation update step equations (2.15) of Theorem 2.3 for each ensemble member become

$$v_{k+1}^{(i)} = (H_{k+1}x_{k+1\,|\,k}^{(i)} - u_k^{(i)}), \quad u_k^{(i)} \sim \mathcal{N}(0, R_{k+1}), \tag{2.17a}$$

$$x_{k+1\,|\,k+1}^{(i)} = x_{k+1\,|\,k}^{(i)} + \hat{K}_{k+1}(y_{k+1} - v_{k+1}^{(i)}). \tag{2.17b}$$

In the deterministic EnKF versions, the update step works by deterministically shifting prior ensemble forecasts, avoiding stochastic noise. Instead of performing the observation update for each ensemble member, a posterior mean is created on which a new ensemble is created. For deriving the deterministic EnKF equations, the following matrix inversion lemma is needed.

Lemma 2.1 (Woodbury matrix identity)**.** *Assume* $A \in \mathbb{R}^{n\times n}, C \in \mathbb{R}^{k\times k}, U \in \mathbb{R}^{n\times k}, V \in \mathbb{R}^{k\times n}$. *Then*

$$(A+UCV)^{-1} = A^{-1} - A^{-1}U(C^{-1}+VA^{-1}U)^{-1}VA^{-1}.$$

Proof. In Appendix A. □

Now, the posterior mean is created using prior ensemble forecast means:

$$\overline{x}_{k+1\,|\,k+1} = \overline{x}_{k+1\,|\,k} + \hat{K}_{k+1}\left(y_{k+1} - H_{k+1}\left(\overline{x}_{k+1\,|\,k}\right)\right).$$

The same may now be formulated with some vector w as

$$\overline{x}_{k+1\,|\,k+1} = \overline{x}_{k+1\,|\,k} + X_{k+1\,|\,k}w_{k+1}.$$

From these with denoting $v_{k+1} = \left(y_{k+1} - H_{k+1}\left(\overline{x}_{k+1\,|\,k}\right)\right)$ and using Lemma 2.1 on line 4,

$$\begin{aligned}w_{k+1} &= X_{k+1\,|\,k}^{-1}\hat{K}_{k+1}\\ &= X_{k+1\,|\,k}^{-1}\hat{P}_{k+1\,|\,k}H_{k+1}^{\top}(H_{k+1}\hat{P}_{k+1\,|\,k}H_{k+1}^{\top}+R_{k+1})^{-1}\\ &= Y_{k+1\,|\,k}^{\top}(Y_{k+1}Y_{k+1}^{\top}+R_{k+1})^{-1}v_{k+1}\\ &= (I+Y_{k+1\,|\,k}^{\top}R_{k+1}^{-1}Y_{k+1\,|\,k})^{-1}Y_{k+1\,|\,k}^{\top}R_{k+1}^{-1}v_{k+1}.\end{aligned}$$

With this, the mean can be computed in the ensemble space dimensions, reducing computational cost significantly when the ensemble is smaller than the number of observations.

Multiple algorithms exist to create an ensemble around the posterior mean, but the Ensemble transform Kalman filter (ETKF) variant is presented and used in this work. A posterior ensemble is created by perturbing the posterior mean with an estimate of the posterior uncertainty $P_{k+1\,|\,k+1}$.

Starting from the original Kalman filter posterior covariance (2.15e) with the ensemble Kalman gain estimate and again using Lemma 2.1,

$$\begin{aligned}
\hat{P}_{k+1\,|\,k+1} &= (I - \hat{K}_{k+1}H_{k+1})\hat{P}_{k+1\,|\,k} \\
&= \left(I - X_{k+1\,|\,k}Y_{k+1\,|\,k}^\top (Y_{k+1}Y_{k+1}^\top + R_{k+1})^{-1}H_{k+1}\right) X_{k+1\,|\,k}X_{k+1\,|\,k}^\top \\
&= X_{k+1\,|\,k}\left(I - Y_{k+1\,|\,k}^\top (Y_{k+1}Y_{k+1}^\top + R_{k+1})^{-1}Y_{k+1\,|\,k}\right) X_{k+1\,|\,k}^\top \\
&= X_{k+1\,|\,k}\left(I - (I + Y_{k+1\,|\,k}^\top R_{k+1}^{-1}Y_{k+1})^{-1}Y_{k+1}^\top R_{k+1}^{-1}Y_{k+1}\right) X_{k+1\,|\,k}^\top \\
&= X_{k+1\,|\,k}(I + Y_{k+1\,|\,k}^\top R_{k+1}^{-1}Y_{k+1})^{-1}X_{k+1\,|\,k}^\top.
\end{aligned}$$

Writing $T = (I + Y_{k+1\,|\,k}^\top R_{k+1}^{-1}Y_{k+1\,|\,k})$, the expression can be written as

$$\hat{P}_{k+1\,|\,k+1} = (X_{k+1\,|\,k}T^{\frac{1}{2}}U)(X_{k+1\,|\,k}T^{\frac{1}{2}}U)^\top$$

for any unitary U. And now, the posterior ensemble may be given as

$$\begin{aligned}
x_{k+1\,|\,k+1}^{(i)} &= \overline{x}_{k+1\,|\,k+1} + \sqrt{m-1}X_{k+1\,|\,k}(T^{\frac{1}{2}}U)_i \\
&= \overline{x}_{k+1\,|\,k} + X_{k+1\,|\,k}w_{k+1} + \sqrt{m-1}X_{k+1\,|\,k}(T^{\frac{1}{2}}U)_i \\
&= \overline{x}_{k+1\,|\,k} + X_{k+1\,|\,k}\left(w_{k+1} + \sqrt{m-1}(T^{\frac{1}{2}}U)_i\right).
\end{aligned}$$

This proceeding is presented in Algorithm 2 with one final step being the propagation of the updated state estimate to the next time step. The algorithm may thus be executed in a loop to create a complete filtering distribution for some time series.

Algorithm 2 Ensemble transform Kalman filter (ETKF)

1: **initialise** observation and model operators H and M, forecast ensemble $X_{ens} = \begin{bmatrix} x^{(1)} & \cdots & x^{(N)} \end{bmatrix}$, observation y, orthogonal matrix U, error covariance matrix R
2: $\overline{X}_{ens} = \frac{1}{N}X_{ens}\mathbf{1}^\top$
3: $X = (X_{ens} - \overline{X}_{ens})/\sqrt{N-1}$
4: $Y = H(X)$
5: $\overline{Y} = \frac{1}{N}Y\mathbf{1}^\top$
6: $C = R^{-\frac{1}{2}}(Y - \overline{Y})/\sqrt{N-1}$
7: $v = R^{-\frac{1}{2}}(y - \overline{Y})$
8: $T = (I + C^\top C)^{-1}$
9: $w = TC^\top v$
10: $W = w + \sqrt{N-1}T^{\frac{1}{2}}$
11: $X_{ens} = \overline{X}_{ens} + XW$
12: **return** $M(X_{ens})$

While the EnKF methods are computationally tempting, free lunches are not guaranteed. As full high-dimensional covariance matrices are represented by small ensembles, the ensemble covariance is not a full representation the full covariance. A common issue is spurious correlations over distant observations caused by sampling errors. Techniques called localisation and inflation are used to overcome this issue.

Localisation is a commonly used approach to mitigate the spurious correlations issue by reducing the interdependence of distant observations during analysis. Localisation can be performed either by domain localisation or covariance localisation. Domain localisation works by performing the EnKF analysis in smaller subdomains separately. This way, spatially distant observations have no effect in the analysis. As an added benefit, domain localisation can be used to speed up the computation by dividing the problem into smaller parallel subproblems. Covariance localisation works by filtering the ensemble covariance from spurious correlations. If some distant observations are known to have little correlation, the ensemble covariance matrix values can be adjusted accordingly.

Spurious correlations and small ensembles also easily result in underestimating ensemble variances. By inflating ensemble spreads, overly optimistic variances can be widened. Inflation is simplest to achieve by multiplying the individual ensemble member deviations from the mean by some scalar larger than 1.

The usage of Kalman filter methods is not without complexity. It involves tuning multiple parameters that significantly impact the filtering results. As the parameter tuning is not trivial and depends on the used data, experimentation is required to yield optimal results.

3 Model

In this section, the hybrid model incorporating a physics-informed neural network with the Bayesian state-space modelling methods is proposed. The objective is to create a framework where possibly multiple partial observations could be used to generate probabilistic estimates of some quantity of interest that evolves according to the advection equation. The developed model is compromised of two main components:

- A deterministic neural network-based model that learns the patterns and dynamics of the advection equation from spatio-temporal data.
- An ensemble Kalman filter that leverages the neural network as a state evolution model to provide probabilistic state estimates.

These components are next presented separately, starting with the neural network model.

3.1 Deterministic Neural Network Model

This subsection describes the neural network model predicting motion from two-dimensional spatio-temporal data.

The data that the network is designed to work with is assumed to approximately follow the discrete advection equation in a square spatial domain:

$$\frac{\partial I_t[x,y]}{\partial t} \approx -\mathbf{F}_u[x,y]\frac{\partial I_t[x,y]}{\partial x} - \mathbf{F}_v[x,y]\frac{\partial I_t[x,y]}{\partial y}, \quad t = 0, 1, \ldots,$$

where $I_t \in \mathbb{R}^{n\times n}$ is an image at time t, and $\mathbf{F}$ a vector field of the two-dimensional motion vectors. The specific assumption for the images I is that they consist of smooth objects moving uniformly across the two spatial dimensions over time. Furthermore, as necessitated by the used data, the model was also created to be robust to even large deviations from the pure advection process.

The network predicts a velocity field representing the advection process across a sequence of spatio-temporal image data. Formally, the objective is to train a neural network that maps a sequence of images to a vector field. Then the output may be interpreted as a warping operator $W : \mathbb{R}^{n\times n} \to \mathbb{R}^{n'\times n''}$ that maps an image forward according to the predicted vector field $\mathbf{F}$,

$$W(I_t\,;\,\mathbf{F})[x,y] = I_t[x - \mathbf{F}_u[x,y],\, j - \mathbf{F}_v[x,y]], \quad \text{if } x, y \in [1,n]. \tag{3.1}$$

The model is unaware of image values outside the domain, and thus the output domain of the warping operator is smaller, $n', n'' \leq n$. However, for practical purposes, the operator was modified to copy boundary values that can then be disregarded. Figure 3.1 depicts how the warping operator works for an interior pixel in a 5×5 image.

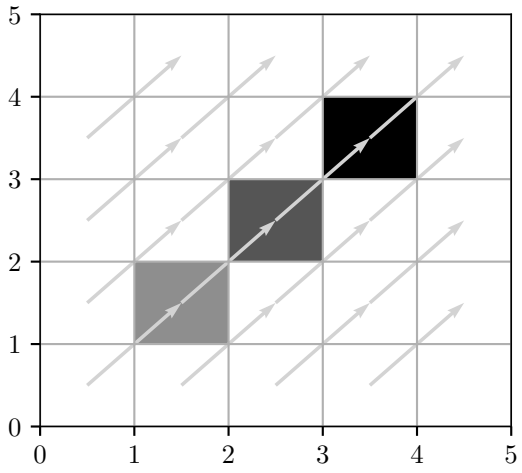

Figure 3.1: The warping operator takes the pixel values and moves them to new pixels corresponding to the velocity field. The velocity field moves pixel values one step up and one to the right at each time step.

Motion estimation has been studied in the context of optical flow problems. Classical optical flow techniques are also often used in meteorology in nowcasting problems [26]. The neural network approach for the optical flow problems has mostly been studied in a different setting. They are often trained with ground-truth flows and with datasets where motion only applies to small objects in images, most notably with the KITTI [11], and Sintel [5] datasets. Contrary to this, this work considers an unsupervised approach where ground-truth vector fields are not available. Moreover, the data used differs from the typical ones used in the neural network optical flow context.

A similar unsupervised approach, from which much of the inspiration to the model in this work can be attributed, is explored in [8] and [37]. In these works, a vector field neural network was used with much more complex sea surface temperature data with an attempt to generate pixel-wise motion vector estimates. Moreover, these works also modelled the diffusion process, by a hard-coded diffusion coefficient [8] and by a separate neural network [37]. The decision to dismiss diffusion from this model primarily stems from the fact that it is unlikely a meaningful physical model for spreading of clouds in the used cloud data. Also, in using a neural network for diffusion modelling, specific care must be given to the loss function selection, as diffusion is easily overestimated if average prediction errors are minimised. Nevertheless, a diffusion term could be implemented without changing the advection model.

In order to tackle the needs of the much simpler data similar to the one described in Section 2.1.2 and in Figure 2.1 but without diffusion, a different approach to the neural network structure is required. As a consequence, the underlying neural network architecture is also designed differently. Basis vectors are utilised here akin to the method used in [37] but with two key differences. A simple second-degree polynomial basis is opted for instead of a high-dimensional radial basis for the vector

field, which guarantees smooth motion. Furthermore, instead of directly predicting the basis vector coefficients, the neural network predicts motion at fixed nodes from which a polynomial can be interpolated throughout the domain. With the chosen approach, small-scale motion in the data can be disregarded while focusing on the larger-scale uniform motion that affects the entire domain. Even though the small-scale motion is important for individual samples, it is often random and might be affected by measurement uncertainties or data processing.

3.1.1 Model Architecture

The proposed model architecture is similar to the well-known Residual Networks (ResNet). To create two-dimensional motion vectors, two separate ResNet models are employed, each trained to predict one dimension of the velocity fields.

Residual Networks are characterised by skip connections, which allow the gradient to be directly backpropagated to earlier layers. The 'residual' term comes from the learning process of these networks, where they learn the residual mappings relative to the layer inputs. The residual mappings alleviate the common vanishing gradient problem of deep neural networks. While residual connections are utilised in a wide array of neural network architectures, they were first formally developed for image recognition tasks in [13]. Learning advection equation parameters at fixed nodes in this work resembles the image recognition tasks in that the inputs are images and outputs vectors. However, instead of outputting classes from multi-channel images, this thesis interprets the network as outputting vector field values at nodes from spatio-temporal data.

A key component of the ResNet architecture is the residual block. Residual blocks have the general form

$$h(x) = f(x) + x, \tag{3.2}$$

where $f(x)$ represents the trainable layers in a block, and x is the block's original input, representing the skip connection. An important aspect here is the behaviour of the residual blocks in backpropagation. Deep neural networks often suffer from a problem called diminishing gradients. This is because the gradient computation involves long chains of multiplications of small values. The derivative of the residual block $h(x)$

$$\frac{d}{dx}h(x) = \frac{d}{dx}f(x) + 1, \tag{3.3}$$

ensures, due to the $+1$ term that gradients don't vanish.

This technique is applied in a wide array of neural network architectures, but perhaps the most common use is in convolutional neural networks. In convolutional networks, the residual blocks consist of several convolution layers, each with their own batch normalisations and activations. In this thesis, a 26-layer convolutional ResNet was deployed. The specific layer architecture of the used network is shown in Figure 3.2.

$$
\begin{gathered}
t \times n \times n \\
\downarrow \\
5 \times 5 \text{ conv2d, } 64, /1 \\
\downarrow \\
3 \times 3 \text{ maxpool, } /2 \\
\downarrow \\
3 \times \begin{bmatrix} 3 \times 3 \text{ conv2d, } 64, /1 \\ 3 \times 3 \text{ conv2d, } 64, /1 \end{bmatrix} \\
\downarrow \\
\begin{bmatrix} 3 \times 3 \text{ conv2d, } 128, /2 \\ 3 \times 3 \text{ conv2d, } 128, /1 \end{bmatrix} \\
\downarrow \\
2 \times \begin{bmatrix} 3 \times 3 \text{ conv2d, } 128, /1 \\ 3 \times 3 \text{ conv2d, } 128, /1 \end{bmatrix} \\
\downarrow \\
\begin{bmatrix} 3 \times 3 \text{ conv2d, } 256, /2 \\ 3 \times 3 \text{ conv2d, } 256, /1 \end{bmatrix} \\
\downarrow \\
2 \times \begin{bmatrix} 3 \times 3 \text{ conv2d, } 256, /1 \\ 3 \times 3 \text{ conv2d, } 256, /1 \end{bmatrix} \\
\downarrow \\
\begin{bmatrix} 3 \times 3 \text{ conv2d, } 512, /2 \\ 3 \times 3 \text{ conv2d, } 512, /1 \end{bmatrix} \\
\downarrow \\
3 \times \begin{bmatrix} 3 \times 3 \text{ conv2d, } 512, /1 \\ 3 \times 3 \text{ conv2d, } 512, /1 \end{bmatrix} \\
\downarrow \\
\text{adaptive average pool} \\
\downarrow \\
\text{linear, } n_{\text{basis}} \\
\downarrow \\
n_{\text{basis}} \times 1
\end{gathered}
$$

Figure 3.2: The structure of the used 26-layer ResNet architecture. The network takes as an input spatio-temporal data with temporal dimension t, and spatial dimension $n \times n$, and outputs a vector of length n_{basis}. Here, the residual blocks that sum the block input to its output are the groups of layers in the brackets. The layer parameters are the used kernel size, followed by the layer type, number of output channels, and stride. Striding divides the spatial dimension by the used parameter by skipping that number of pixels when computing the results. The maxpool layer outputs the maximum value inside the used kernel size, and the average pool takes the average value. In the last layers, the adaptive average pool adjusts the kernel size such that the output shape of the layer is $512 \times 1 \times 1$, and the linear layer is just a matrix multiplication to produce an n_{basis}-vector.

3.1.2 Producing Vector Fields and Forecasts

While the described architecture generates low-dimensional velocity fiels only a few points, the primary objective is to produce the complete vector fields. To this end, Lagrange polynomial interpolation is used to extend the predicted vectors to form a vector field across the entire domain. The method of predicting motion vector values from which the polynomial coefficients are then computed gives stability to the optimisation process. This way, small changes to the network output result with predictable small changes in the complete vector fields and thus in the loss values.

Second-degree polynomials are chosen as a compromise between capturing a majority of the underlying process and maintaining sufficient flexibility to span the same vector field over multiple time steps. Since the basis vectors are constant, the model complexity is effectively controlled by the dimensionality of the basis. Consequently, the basis selection represents a trade-off between the smoothness of the vector fields and the ability to fit more complex patterns in the vector fields.

The second-degree polynomial form for the vector fields is

$$\mathbf{F}(x,y) = \begin{bmatrix} \mathbf{F}_u(x,y) \\ \mathbf{F}_v(x,y) \end{bmatrix} = \begin{bmatrix} a_{u_1} + a_{u_2}x + a_{u_3}y + a_{u_4}x^2 + a_{u_5}xy + a_{u_6}y^2 \\ a_{v_1} + a_{v_2}x + a_{v_3}y + a_{v_4}x^2 + a_{v_5}xy + a_{v_6}y^2 \end{bmatrix}.$$

Employing two separate ResNet models configured to output the motion vectors at six different nodes $\{(x_i, y_i)\}_{i=1}^6$, a Vandermonde matrix can be constructed at the same set nodes by

$$V((x_1,y_1),(x_2,y_2),\ldots,(x_6,y_6)) = \begin{bmatrix} 1 & x_1 & y_1 & x_1^2 & x_1y_1 & y_1^2 \\ 1 & x_2 & y_2 & x_2^2 & x_2y_2 & y_2^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_6 & y_6 & x_6^2 & x_6y_6 & y_6^2 \end{bmatrix}.$$

Given this, the polynomial coefficients a_u can be solved uniquely by inverting the Vandermonde matrix

$$\begin{bmatrix} 1 & x_1 & y_1 & x_1^2 & x_1y_1 & y_1^2 \\ 1 & x_2 & y_2 & x_2^2 & x_2y_2 & y_2^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_6 & y_6 & x_6^2 & x_6y_6 & y_6^2 \end{bmatrix} \begin{bmatrix} a_{u_1} \\ a_{u_2} \\ \vdots \\ a_{u_6} \end{bmatrix} = \begin{bmatrix} \mathbf{F}_u(x_1,y_1) \\ \mathbf{F}_u(x_2,y_2) \\ \vdots \\ \mathbf{F}_u(x_6,y_6) \end{bmatrix}$$

$$\begin{bmatrix} a_{u_1} \\ a_{u_2} \\ \vdots \\ a_{u_6} \end{bmatrix} = \begin{bmatrix} 1 & x_1 & y_1 & x_1^2 & x_1y_1 & y_1^2 \\ 1 & x_2 & y_2 & x_2^2 & x_2y_2 & y_2^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_6 & y_6 & x_6^2 & x_6y_6 & y_6^2 \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{F}_u(x_1,y_1) \\ \mathbf{F}_u(x_2,y_2) \\ \vdots \\ \mathbf{F}_u(x_6,y_6) \end{bmatrix},$$

and similarly for a_v. As the Vandermonde matrix is notorious for having a large condition number, the nodes need to be selected carefully to ensure stable invertibility. Hence, the image domain is set to be $[-1,1]\times[-1,1]$, and one node is set at the origin, and five at non-symmetric points on the unit circle as depicted in Figure 3.3.

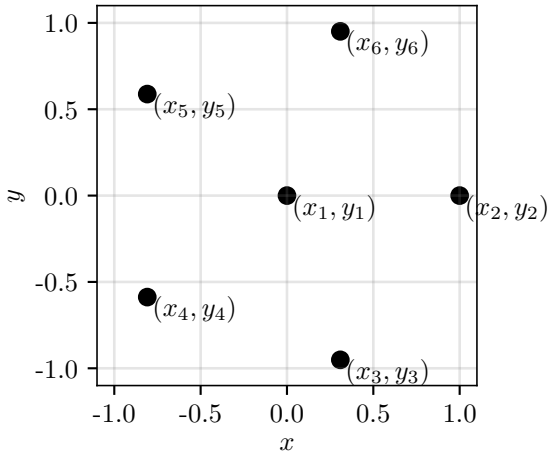

Figure 3.3: The six nodes at which the neural network approximates velocity fields. In order to ensure stable invertibility of the two-dimensional Vandermonde matrix, the location of the nodes must be selected carefully.

After solving the polynomial coefficients, the motion vectors may be interpolated over the entire domain by evaluating the Vandermonde matrix at every pixel and right-multiplying it by the coefficient vectors.

Now that the velocity field can be estimated, forecasts can be produced by applying the warping operator $W(\cdot\,; \mathbf{F})$ from Equation (3.1) to the latest image repeatedly. In principle, the velocity field could be estimated again by adding the predicted image to the sequence and feeding through the model to produce a new estimate. However, the predicted images do not contain any new information from the underlying process and thus, velocity fields predicted from observed data are used.

3.1.3 Training the Neural Network

Due to the general unavailability of ground-truth velocity fields, an unsupervised training approach must be taken. Therefore, the loss metric computation has to be done by warping the latest data sample according to the predicted velocity field and comparing it to the true data samples of corresponding time steps.

Bilinear interpolation was applied in training instead of a nearest-point warping interpolation of Equation (3.1) for differentiability reasons. This method calculates the new value at time $t + 1$ after warping as a weighted average of the values from the nearest four pixels at time t, ensuring differentiability for backpropagation. However, this method smooths out sharp edges in images, and consequently, the simpler nearest-point interpolation is used in interference.

The network is trained by minimising the average loss between warped images and the target images over a number of time steps. For each iteration, the network predicts the velocity fields from a batch of consecutive data samples, and the latest image from this batch is warped using the aforementioned warping scheme. The

warping is done over several subsequent time steps to compute an average loss between actual and warped images. As the warping operator is undefined beyond the latest observed image boundaries, it is modified to copy the edge values to obtain full images and the loss metric is only computed in the interior pixels of the predicted images. The selection of the input sequence length and the number of forecasted time steps for loss computation in training depend on the temporal characteristics of the training data. Suppose the underlying process adheres to the simplified advection model accurately. In that case, longer image sequences can be used for more accurate training convergence. However, if the advection model is generally unusable after some number of time steps, the loss would only be random noise.

Adam is employed as the optimisation algorithm for the model parameters. It is a variant of the SGD discussed in Section 2.2.3 but benefits from an adaptive step size. It also simplifies the model tuning, as the step size and step size schedulers do not play such an important role in the proper convergence of the model.

Some fine-tuning was applied to the model with respect to the actual data. This is discussed in the results section in addition to the specific loss function selection.

3.2 Integrating the Neural Network into ETKF

This subsection describes the ensemble-based Kalman filter set-up of the later experiments.

The neural network is trained to produce velocity fields from spatio-temporal data but can only work with complete data and produce deterministic forecasts. The neural network is integrated into the Kalman filter model to widen the class of possible applications. The motivation for this is two-fold. Firstly, approximating the filtering distribution provides a way to assimilate multiple indirect partial observations of varying quality to the state estimates as they arrive. Secondly, approximating the prediction distribution yields probabilistic forecasts over multiple time steps. While this work's primary focus has been filtering distribution, the prediction distribution case is also discussed, as it comes with some intricacies of its own. A third closely related problem, but not considered in this work's scope, is the smoothing distribution estimation with a Kalman smoother to update previous state estimates given future observations.

Due to the high dimensionality of image data, the ensemble transform Kalman filter presented in Algorithm 2 is used. Therefore, the distributions are only ensemble approximations instead of the analytical solutions of the standard Kalman filter. Using the neural network model as a state evolution operator in an EnKF is influenced by [37]. However, a slightly different approach is taken here to accommodate even larger state vectors. Furthermore, here, the interest is in the filtering distribution instead of probabilistic forecasting, with the aim of providing an estimate of the entire state from incomplete observations.

3.2.1 State-Space Model Operators

For analysis of the state-space model (2.10), observation and state evolution operators H and M are needed. While the nonstationary inverse problem related to the observation operator represents a significant part of the power of the Kalman filter, the main contribution and interest of this work is in incorporating the neural network in the state evolution operator. Therefore, the observation operator is assumed to be a trivial identity mapping but leaves out missing observations from the state vectors. Despite this somewhat artificial treatment of the observation process, challenges related to missing observations are essential in many real-world applications. Missing observations present a significant challenge also in the data considered in this work's numerical experiments.

As the neural network models data using the advection equation, the network output can be interpreted as the linear warping function that transforms a state vector to the next time step already discussed with the neural network model. The state operator is consequently a time-dependent warping operator that operates with the state vectors in the state-space model, i.e.

$$M_{k+1}x_k[i,j] = x_k[i - \mathbf{F}_u[i,j],\, j - \mathbf{F}_v[i,j]],$$

when indices $i - \mathbf{F}_u[i,j]$ and $j - \mathbf{F}_v[i,j]$ are inside the image domain. As discussed earlier, if the source point is outside the image domain, the border values are used to retain the domain size in the filter. The filtering can be performed over a larger domain and focus on the interior of the filtered states, so that observations would be assimilated multiple times for the interior pixels. Also, wrong border values are corrected over time with observations by the Kalman filter.

Due to the use of the ensemble filter, the same model operator is applied for each ensemble member separately according to the Algorithm 2. As the neural network is intended to estimate motion from single point estimates, the ensemble means are used to produce new velocity fields to update M when required. When computing the filtering distributions, incoming observations give new information on the system's state. Therefore, an updated model operator is computed at each step to give the most up-to-date velocity field estimate. However, as observations are not available for the prediction case, the best guess for the model operator is the one computed from the latest filtering distribution means.

3.2.2 ETKF settings

While the ETKF model in Algorithm 2 provides a computationally efficient Kalman filter formulation, some tuning and parameters are needed for optimal results. Particularly in the case of the data considered in this work, ensemble generation is crucial to get realistic state estimates.

Ensemble Kalman filters require an initial ensemble of the state vector before running the algorithm. For the neural network's case, even an entire input sequence is needed to obtain the first model operator. The ensemble size is an important parameter representing a trade-off between accurately representing the true theoretical state distribution and computational efficiency with larger state vectors. Even

though the initial ensemble of the state vector must be computed without the help of the Kalman filter, the assimilation of observations in subsequent steps corrects the first estimate over time. The initial ensemble can be constructed by artifically perturbing a single initial estimate of the state vector. To help with this, many data sources have some theoretical uncertainty estimates. Kalman filters provide an efficient way to optimally assimilate observations to the state estimates under the assumption that the uncertainties are additive and Gaussian. In the ensemble case these assumptions translate to that the state ensemble should follow a Gaussian distribution. This assumption yields an efficient formulation, but the validity of the assumption depends on the target application and data used.

It is also imporant to note that the assumptions are only needed for the optimality of the observation update step. The prediction distribution case with the ETKF works essentially by just propagating the latest state ensemble estimate forward. Thus, the ensemble need not be limited by the Gaussian distribution assumptions, and can be generated with more freedom. In the prediction case the ensemble can then be generated and inflated on a case-by-case basis depending on the exact data source used and on the accuracy of previous predictions.

But also for the filtering distribution case, a non-religious following of the assumptions is needed to get valid state ensembles. Given the data considered here, the uncertainties are known to be related to the location of spatial objects in data and their values. Therefore, while not theoretically entirely consistent with the Kalman filter assumptions, the initial ensemble is created by applying Gaussian noise of small intensity and applying uniform random warps to the first state estimate.

As the same model operator given by the neural network is applied to all ensemble members, the state's borders become broadly similar between ensemble members. Consequently, uncertainty in the border areas is easily underestimated. Inflation is thus needed to to increase the variation between the ensemble members. Similar techniques as with creating the initial ensemble are used, i.e. perturbing the ensemble by Gaussian noise and applying small uniform warping operations to each member of the ensemble.

Another issue arising from the nature of the used data is that that a small ensemble size and the assimilation of large observations easily break down the spatial structures. Therefore, localisation is performed by dividing the state into smaller patches where the ETKF algorithm is performed separately. As the trivial observation operator does not consider values outside the domain, localisation behaves well on the borders of the patches. Another benefit of localisation is the possibility of speeding up computation significantly by performing the algorithm in parallel for each of the smaller blocks.

4 Numerical Experiments

In this section, the described model is trained and tested on meteorological cloud optical thickness data. This section is divided into three parts. Firstly, the used cloud optical thickness data is described. Then, the deterministic neural network model is trained with the data, and its performance is evaluated. And finally, the neural network is tested as a state operator in the ensemble Kalman filter.

4.1 Cloud Optical Thickness Data

While the described model is designed to be data-independent, cloud optical thickness (COT) data retrieved from satellites was used in its development. COT observations provide a consistent data source suitable for the model. The choice of using COT data is motivated by two key factors: (1) it is available at regular intervals, which makes it suitable for short-term forecasting applications, and (2) the nature of the COT data makes it suitable for the advection equation modelling.

Optical thickness is a quantity that is related to the Beer's law. It describes the exponential attenuation of monochromatic light by

$$T = \frac{I(s_1)}{I(s_0)} = e^{-\tau}.$$

In this equation, T is called transmittance, $I(s_0)$ is the intensity of incident light and $I(s_1)$ the intensity of transmitted light after traveling through a medium. Optical thickness τ at height z is defined by the vertical integral

$$\tau(z) = \int_z^\infty \beta_e(z')\, dz',$$

where $\beta_e(z')$ is the extinction coefficient at elevation z'. A higher cloud optical thickness then means that less light passes through it. A more detailed description of clouds' optical properties can be found from, e.g. [21].

The data used for this work is computed with the EUMETSAT Nowcasting and Very Short Range Forecasting Satellite Application Facility (NWC SAF) cloud microphysical properties postprocessing within the software package for geostationary satellites (NWC/GEO v2018) at the Finnish Meteorological Institute. In practice, the algorithm computes the COT values using measured and simulated reflectances at 0.6 µm and 1.6 µm channels. For a deeper understanding of the algorithmic principles behind the computation of the COT data, readers are referred to [9] and [25].

The data is captured with geostationary satellites that orbit 36000 km above the equator at a point where they remain at a constant position relative to the Earth. The stationarity enables continuous observations of the same geographical region which is indispensable for coherent data retrieval. However, due to the geostationary orbit above the equator, the spatial resolution of the data varies due to the Earth's curvature and the satellites' angle. Right under the satellite, the spatial resolution is about 3 km. This dataset has a temporal resolution of 15 minutes, enabling reasonably accurate tracking of the evolution of clouds.

4.1.1 COT Data Preprocessing

In order to make the COT data usable for the neural network's purposes, extensive preprocessing was required. Advection modelling within the presented neural network necessitates complete spatio-temporal data in Euclidean coordinates. As preprocessing the data is computationally demanding, a relatively small dataset was preprocessed. For preprocessing, the data was spatially constrained to an area around southern Finland and temporally to the months from April to October of 2021.

COT images were resampled from the native satellite projection to the metric Universal Transverse Mercator (UTM) projection using the pyresample Python package [15]. The spatial resolution of the data was artificially increased to have a pixel cover of approximately $2\,\mathrm{km}^2$. The larger pixel size allows clouds to occupy a larger image area and increases the observable velocities for the network. The data was segmented into four 128×128 image regions to accommodate the neural network's square image inputs. Figure 4.1 illustrates the image area and the four segments used for training the network.

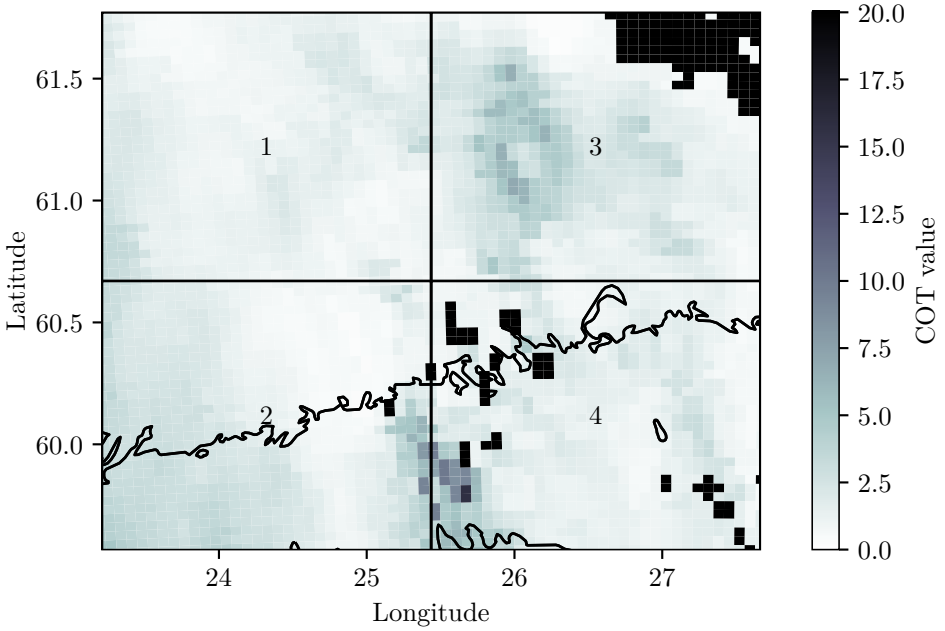

Figure 4.1: The area selected for neural network training over southern Finland and the Gulf of Finland. The entire image area is 512×512 pixels from which four 128×128 sub-images were taken. One pixel covers roughly 2 km in longitude and 1 km in latitude. The missing measurements, represented as black pixels, present a major complication in satellite data usage.

Satellite data often contain a considerable amount of missing values, as seen in Figure 4.1. The frequency and location of the missing values depends on multiple factors as outlined in [9]. For use of the data within the advection equation model,

missing values are problematic. Interpolation of missing values was needed to obtain sufficient training data for the neural network. Given the computational cost of more advanced interpolation, a simple spatial nearest neighbour value approach was employed. While this crude method results in interpolation artefacts for the purposes of the model training, this is not necessarily a problem as the network is robust against poor and noisy data by its design. As the sun's elevation angle heavily influences the dataset's quality, interpolation was restricted to daytime images when the elevation angle exceeded 15°. As a last filter for the data, images with low mean COT values, indicating an absence of clouds, were subsequently filtered out from the dataset. In this stage, the data values were not yet normalised in any way to allow for quick experimentation within the neural network itself. The finalised dataset was temporally segmented into distinct training, validation, and test sets, laying the groundwork for subsequent neural network training.

A major difficulty posed by the COT data with regard to the advection equation modelling is the varying nature of its behaviour. Large chunks of the finalised data consist of data that hardly evolves according to the simple advection equation. While this could, in principle, have been bypassed by selecting only sequences of the data that evolve according to the advection equation model, it would have been computationally restrictive and difficult to achieve in practice. However, the mixed quality of the training data tests the neural network's ability to capture and learn the relevant information.

4.2 Neural Network Model with COT Data

In this subsection, the neural network training procedure with the preprocessed COT dataset is described, and the performance is evaluated.

Even though the neural network presented in Section 3.1 is designed to be data-independent, it was developed, trained, and tested mostly for the COT data. On top of the presented architecture, some tuning of the neural network was done for the COT data.

4.2.1 Tuning the Neural Network With COT Data

The training process was tailored to the characteristics and challenges related to the COT data while keeping in mind the underlying physical interpretation of the advection equation. The previously described architecture was chosen through extensive experimentations guided by performance, interpretability, and model simplicity.

Within the ResNet model, increasing the network depth provided little benefits and, thus, kept the network simple. The velocity fields were produced using various basis types, including Fourier, wavelet, and polynomial basis, with different numbers of basis vectors. The described second-degree polynomials provided the best results in terms of desired velocity field properties.

Various loss functions were examined. Structural similarity (SSIM), multi-scale SSIM (MS-SSIM), [35] and logarithmic cosh loss functions were tested on top of the chosen MSE loss. Also, additional physics-based terms were tested accounting for

the velocity field properties such as gradient, curl, and divergence inspired by [8]. Due to the small effect of the loss function in the results, the simple MSE loss was used for training the final model.

As the input sequence length, three latest images proved to be the optimal choice in terms of the loss function values. The average loss was then computed over four future time steps in training. The highest possible mini-batch size of within the GPU memory constraints, 64, was chosen. A higher mini-batch size gave stability to convergence, given the noisy nature of the training data.

As the velocity fields are equivariant to rotations, the data was augmented by applying random rotations to the mini-batches. With this, overfitting to the training data was avoided while simultaneously permitting a smaller training dataset and shorter training times.

4.2.2 Training Results

The network was trained on the Finnish IT Center for Science (CSC) Puhti high-performance computing environment using a single Nvidia V100 GPU with 32 GB of VRAM. The network was implemented using the Python library PyTorch wrapped in the PyTorch Lightning interface.

The network was trained until the average MSE loss metric for the validation set reached a plateau. Typically, a plateau was reached in ten epochs and took 10 minutes with the described hardware resources. The efficiency of the training process can be attributed to the simplicity of the network architecture and the relatively small dataset required for good performance.

The trained model was tested with simulated synthetic data examples, not similar to the training data. Despite the differences to the cloud images, the model is able to successfully capture correct motion patterns, given similar data scaling to the training dataset. This promising result suggests that the model works as designed by learning the advection equation dynamics instead of the specifics of the particular training data. Moreover, this suggests that similar physics-informed models could also be trained even with simulated data. Hence, this approach could be used in problems with even more problematic and insufficient training data. Figure 4.2 shows model predictions for a Gaussian circle and squares.

To place the model's performance in a broader context, it was compared against two prediction methods with the test dataset. The first compared method is the well-established Lucas-Kanade technique for optical flow estimation. The second is a naive prediction using the most recent observation as the forecast. As predictability for the data, especially over multiple time steps, is known to be poor, the naive prediction serves as a rudimentary way to assess whether the motion models have any predictive value. The Lucas-Kanade method provides a good baseline for a motion-based model, as it is a commonly used computational method in similar meteorological applications. The general Lucas-Kanade algorithm assumes a locally constant velocity field that is solved using least squares approximation at each pixel using local spatial and temporal gradients. The used Lucas-Kanade version from the PySteps Python library [27] uses a pyramidal implementation presented in [4]

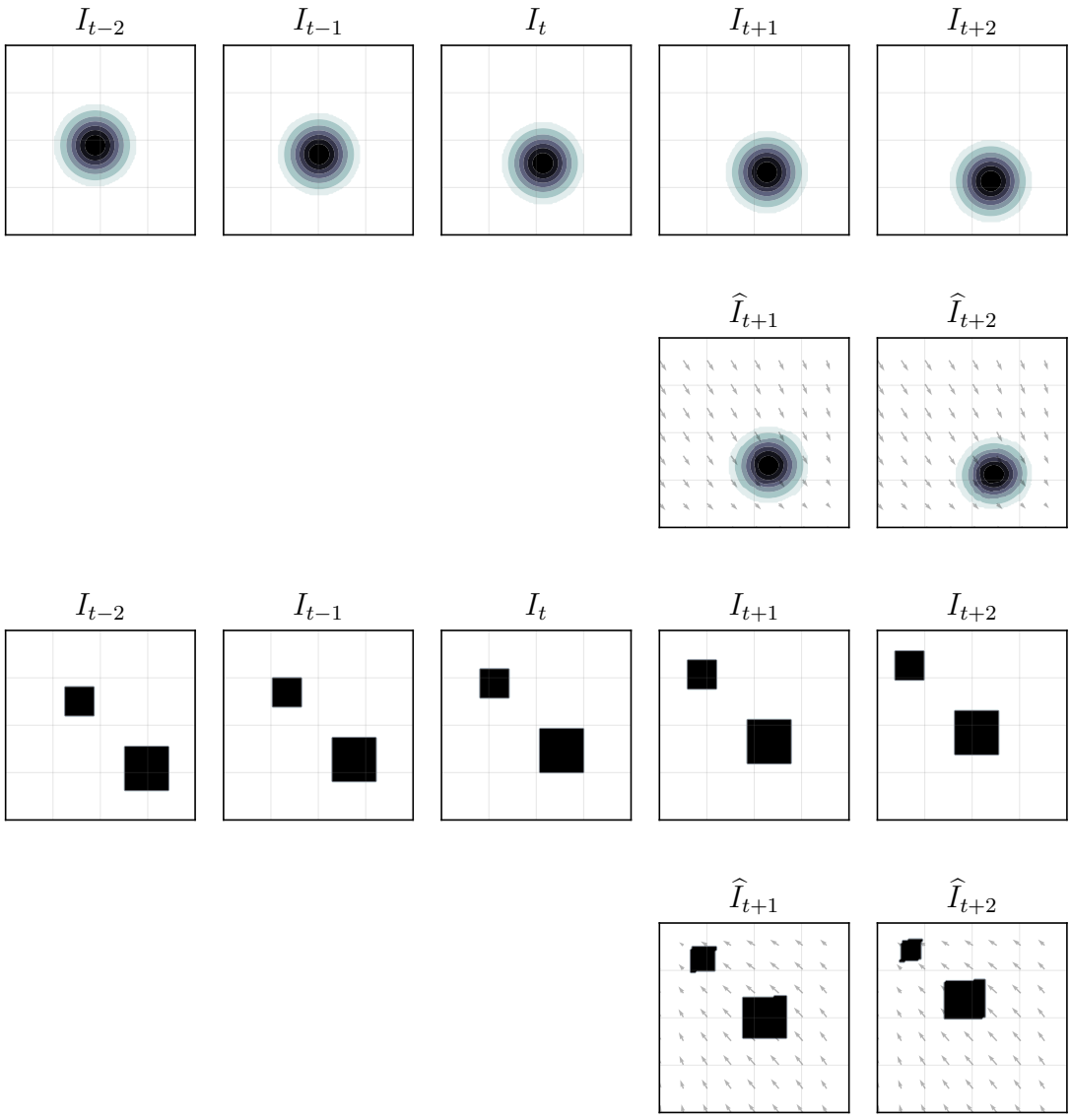

Figure 4.2: Predictions for simulated synthetic data using the neural network trained with the COT dataset. The first rows show the actual data, and the second rows predicted velocity fields and images for two time steps.

that captures the general motion of images by up-scaling the images to varying resolutions. Default parameters were used for the Lucas-Kanade test with the same data scaling and input sequence length of 3 as with the neural network.

In addition to visual estimations of results, qualitative evaluations were done based on three metrics: mean absolute error (MAE), root mean squared error (RMSE), and fractions skill score (FSS). MAE computes the mean of the absolute

deviations of the predicted value $\hat{y}$ from the target y:

$$\mathrm{MAE}(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i|.$$

RMSE is just the square root of the MSE error, restoring the value to the original units:

$$\mathrm{RMSE}(y, \hat{y}) = \sqrt{\mathrm{MSE}(y, \hat{y})} = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2}.$$

These two metrics give insight into the average predictive performance across the entire image domain with different emphasis on smaller and larger prediction errors. Due to the averaging, the one-dimensional indexing can be thought of here as flattening multidimensional data. The neural network was trained to minimise the MSE loss over the training dataset and the Lucas-Kanade is a least squares method for each image. Therefore, it is also important to use a metric not directly related to the average metrics to get a less biased perspective on the performance.

FSS is a spatial verification metric introduced in [28] that compares predictions and targets on a spatial area of some size above some threshold. FSS first computes binary fields $\mathbf{1}_{\geq q}(\hat{y})[i, j]$ and $\mathbf{1}_{\geq q}(y)$ for a two-dimensional prediction $\hat{y}$ and a target y given a threshold value q as

$$\mathbf{1}_{\geq q}(\hat{y})[i, j] = \begin{cases} 1, & \hat{y}_{i,j} \geq q \\ 0, & \hat{y}_{i,j} < q \end{cases}, \quad \mathbf{1}_{\geq q}(y)[i, j] = \begin{cases} 1, & y_{i,j} \geq q \\ 0, & y_{i,j} < q \end{cases}. \tag{4.1}$$

From the binary fields (4.1), spatial densities for a square window size of $n \times n$ are computed as

$$I_{\hat{y}}(n, q)[i, j] = \frac{1}{n^2} \sum_{k=1}^{n} \sum_{l=1}^{n} \mathbf{1}_{\geq q}(\hat{y}) \left[i + k - 1 - \frac{n-1}{2}, j + l - 1 - \frac{n-1}{2} \right], \tag{4.2a}$$

$$I_{y}(n, q)[i, j] = \frac{1}{n^2} \sum_{k=1}^{n} \sum_{l=1}^{n} \mathbf{1}_{\geq q}(y) \left[i + k - 1 - \frac{n-1}{2}, j + l - 1 - \frac{n-1}{2} \right]. \tag{4.2b}$$

In principle, the window does not need to be a square, but as discussed in [28], more sophisticated kernels would not necessarily alter the results enough to warrant the additional complexity. Finally, the FSS value given window length of n and threshold q is then computed from the spatial averages (4.2) by

$$\mathrm{FSS}(n, q) = 1 - \frac{\mathrm{MSE}\left(I_{\hat{y}}(n, q), I_y(n, q)\right)}{\mathrm{MSE}_{\mathrm{ref}}\left(I_{\hat{y}}(n, q), I_y(n, q)\right)},$$

where the reference MSE is the maximum difference

$$\mathrm{MSE}_{\mathrm{ref}}\left(I_{\hat{y}}(n, q), I_y(n, q)\right) = \sum_{i,j} I_{\hat{y}}(n, q)[i, j]^2 + \sum_{i,j} I_y(n, q)[i, j]^2.$$

The FSS value for a perfect skill, where the spatial densities for both binary fields agree, is then 1. For no skill, when the spatial densities disagree completely, the

value is 0. Due to the threshold and window size variables, computing multiple FSS scores is needed to get a greater understanding of the prediction skill.

In order to ensure meaningful and balanced comparisons, some issues need to be considered before presenting the results. Firstly, the metrics were computed using the normalised test dataset. This was essential to provide a more consistent interpretation between both strong and weak cloud patterns, given that the modelling focuses on the motion of cloud structures rather than evolution or formation. Furthermore, the evaluations were restricted to regions of the predicted images where the Lucas-Kanade and the neural network generated forecasts based on the latest available state, rather than extrapolating values from outside the image area. These strategies should ensure a more equitable comparison of the models' performance.

Figures 4.3, 4.4, and 4.5 visualise the computed averaged metrics of forecasts of the test dataset up to two hours. Across all the metrics, the neural network demonstrates performance close to the Lucas-Kanade method. Given that the Lucas-Kanade method is a refined and established algorithms, the neural network model's close performance is encouraging. It affirms the validity of these kinds of neural network applications.

The naive predictions, while not matching the performance of the motion models, are not significantly lagging. This relatively close performance can largely be attributed to the challenging nature of the noisy COT dataset. The situation is not helped by the high share of missing values that are interpolated in a rudimentary manner. This also underscores a need for more sophisticated yet efficient methods for handling missing observations, for which the Bayesian filtering approach is one method. Moreover, as already discussed, a significant portion of the dataset does not conform to the advection equation, thus not providing purposeful data for the modelling methods.

Despite the challenging dataset, what stands out is the resilience of the neural network model. This is partly demonstrated by the close performance of the Lucas-Kanade method but also by the ability to generate sensible velocity fields with simplified, synthetic examples far from the actual training dataset.

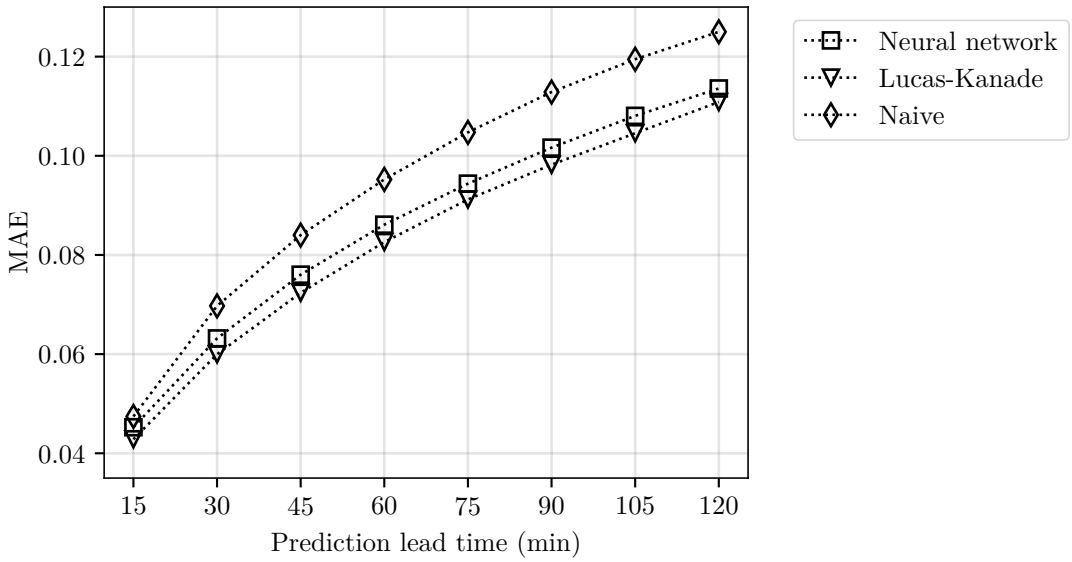

Figure 4.3: Mean absolute error (MAE) of predictions over two hours. Lower value indicates better performance. The Lucas-Kanade and neural network predictions have similar MAE values and naive higher errors.

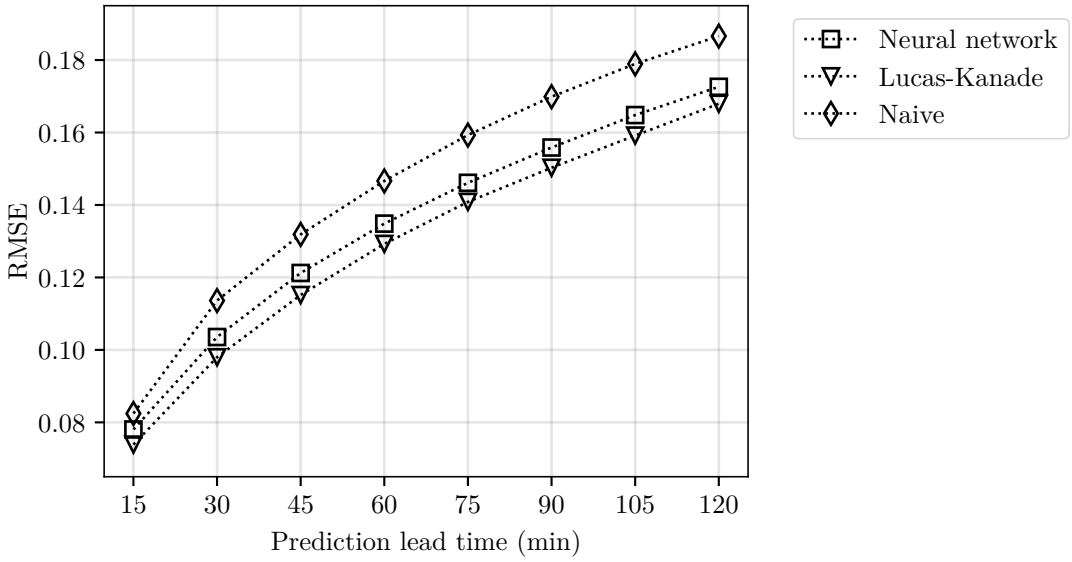

Figure 4.4: Root mean squared error (RMSE) of predictions over two hours. Lower RMSE value indicates better performance. The Lucas-Kanade and neural network have comparable RMSE values and naive worse values.

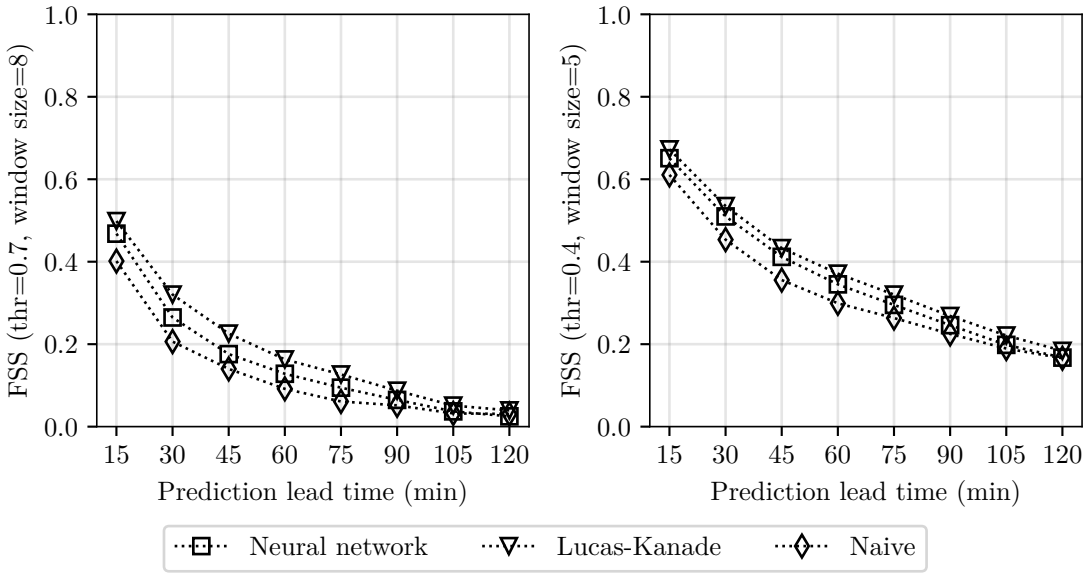

Figure 4.5: Fractions skill score (FSS) of predictions over two hours with two different window sizes and threshold settings. Higher value indicates better performance. The Lucas-Kanade and the neural network have similar skill while naive is worse.

4.3 Bayesian Filtering Results

As a final validation, the Kalman filter model with the neural network is tested.

Evaluating the neural network in the ensemble Kalman filter method presents a more intricate task. Due to the computationally involved ETKF algorithm, a quick computation of the error metrics for a larger dataset is not feasible. As the main focus of this work was on integrating the neural network as the state model operator, a simple observation model with simulated missing values was used. However, as already seen with the COT dataset, missing valued present a significant problem in many real data sources. Hence, the simulated exercise is not unrealistic, even though the model could be capable to more sophisticated inverse problems such as data fusion.

Similar COT images outside of the training dataset of 128×128 pixels and similar synthetic examples are used as with evaluations of the deterministic model. Despite using images of the same shape as the neural network works with, the model could work with different shape of images. Larger images could be downsampled for the neural network model to create a state operator to work with larger state vectors in the ETKF or vice versa. In order to visualise the performance of the motion models, the presented example images are selected such that they largely adhere to the advection equation with some random variation to demonstrate observation assimilation.

For fine-tuning the ETKF model presented in Subsection 3.2, a few assumptions were made related to the objectives of the verification. The ETKF parameters

were chosen to give a small uncertainty for the observations. The intention was to return the observation values where available and, otherwise, values warped from the previous state. This choice stems from the quickly evolving COT data not only within the model's capability but also to verify the results better. In real use of the model, these parameters are subject to case-by-case tuning depending on the used data. Particularly, the observation uncertainty should be a realistic uncertainty estimate of the measurements instead of the assumed constant of 0.23 used for these examples. The data was scaled to $[0, 1]$ range similarly as when training the network. For the initial ensemble and inflation, spatially varying additive noise was generated over the entire image area and then multiplied pixelwise with the image at hand. Localisation was performed in 16×16 pixel blocks. An ensemble size of 500 is used as it appears to give a good balance between results and computing time.

Simulated synthetic data provides a quick visual validation of the ETKF model. The simulated observed noisy data in Figure 4.6 has a noisy background with rectangular blocks moving according to a uniform velocity field. A vertical block of observations has been removed from the middle of the observation images. In this example, the ETKF model works as expected by filling in the missing values and removing fine noise from the observations. The neural network model appears to overestimate the velocity fields a little, but the kalman filter corrects the estimates where observations are available. As a new rectangle appears in the observations, it also appears in the filtered images as expected. This example thus suggests that the model works as intended.

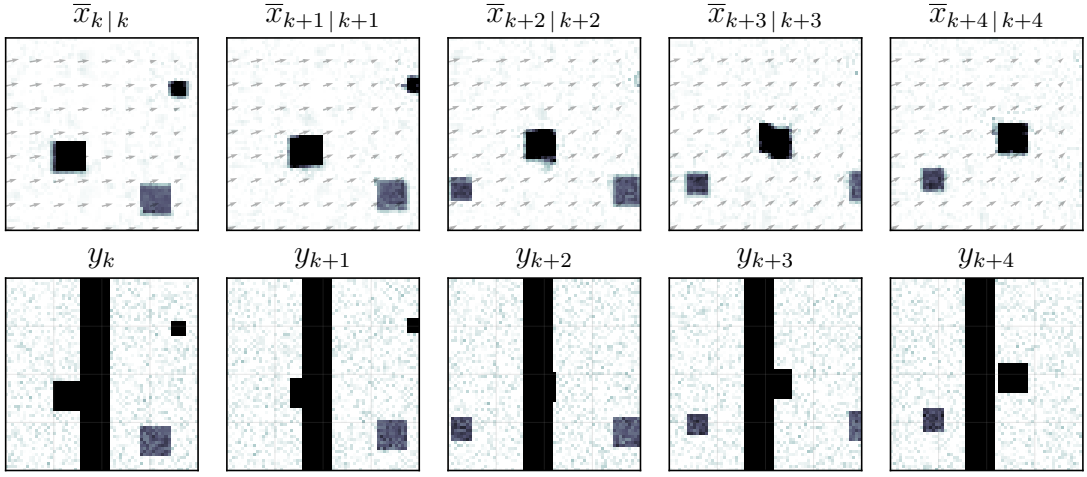

Figure 4.6: ETKF model example with synthetic data. The second row represents the observed data with a vertical black block of missing values, and the first row has the ensemble means of the estimated filtering distribution with the predicted velocity fields.

For verification with the COT data, a sequence of images was selected to compare the neural network model as the state operator against the Lucas-Kanade method and the identity operator. In Figure 4.7, the filtered images of the three different

state operators are presented. As expected, due to the small observation error assumption, the used state propagator shows relatively small differences in the areas of the image where observations are available. However, in the area of missing observations, the motion models provide a clear advantage, as the identity operator leaves the values to the initial state throughout the time steps.

As the Kalman filter produces probabilistic estimates, another way to interpret the results is through the uncertainty of the forecast ensembles, observation uncertainty, and posterior ensemble uncertainties. Figure 4.8 shows how the ETKF corrects the mean deviations with the observations for the neural network images of Figure 4.7. Figure 4.9, shows the state ensemble pixelwise standard deviation across the same time steps. The uncertainty estimates are mostly a result of the selected inflation technique and suitable observation standard deviation parameter. Particularly, the pixelwise uncertainties in Figure 4.9 show that there is larger uncertainty in the area of clouds which is a direct consequence of the added noise fields.

As seen in these examples, a significant difficulty in the uncertainty quantification with the Kalman filter is that the filter parameters effectively control the magnitude of uncertainty. Especially with the ensemble methods, where the ensemble essentially defines the state propagator model uncertainty, many unknown parameters must be guessed or estimated by hand. Particularly, the inflation method plays a crucial role accurate uncertainty estimates as seen with these examples. In wider use of the model, these parameters could be better estimated with some statistical methods instead of hand-tuning with the objective to obtain visually pleasing results.

The inflation and initial ensemble generation are even more critical in possible forecast usage, as no observations are available to support the forecasts. In addition, as the computation of a forecast distribution in the ensemble filter is not limited by the standard Kalman filter assumptions, the range of possibilities for ensemble generation and inflation is much broader. In a recent work related to the forecasting problem presented in [6], an ensemble is generated by multiplying the Fourier transforms of a noise field and a state estimate. In that work, an advection model is also combined with an autoregressive model that could potentially improve the predictive performance of the neural network of this work for modeling the COT data.

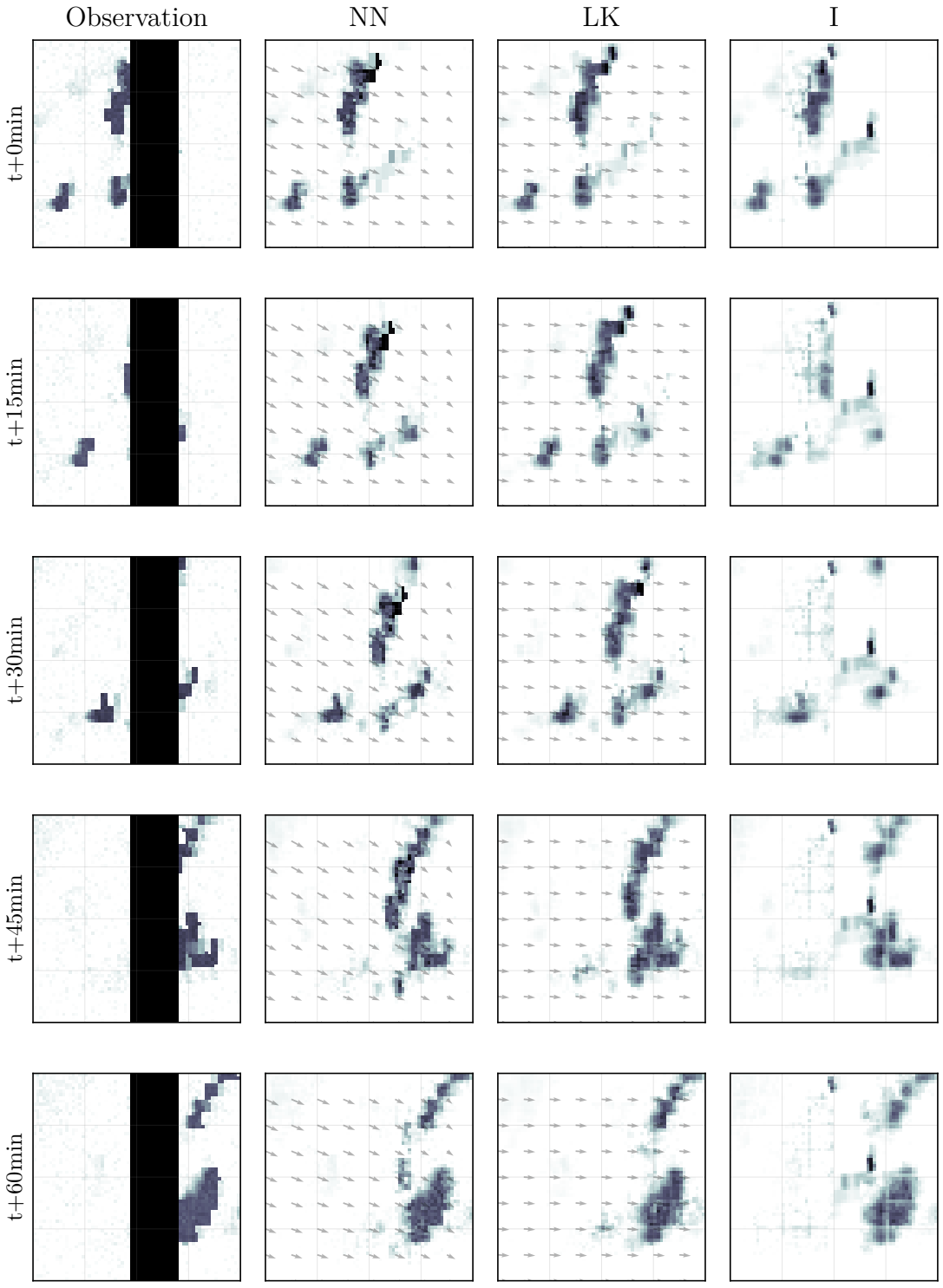

Figure 4.7: A comparison of different model operators for the ETKF. The leftmost column contains the observations with a vertical black block of simulated missing observations; the second column contains the filtering distribution with the neural network as the model operator, the third column with the Lucas-Kanade method, and the third with a trivial identity operator. Estimated velocity fields using the corresponding models in the filtered images show similar patterns with the Lucas-Kanade method and the neural network model.

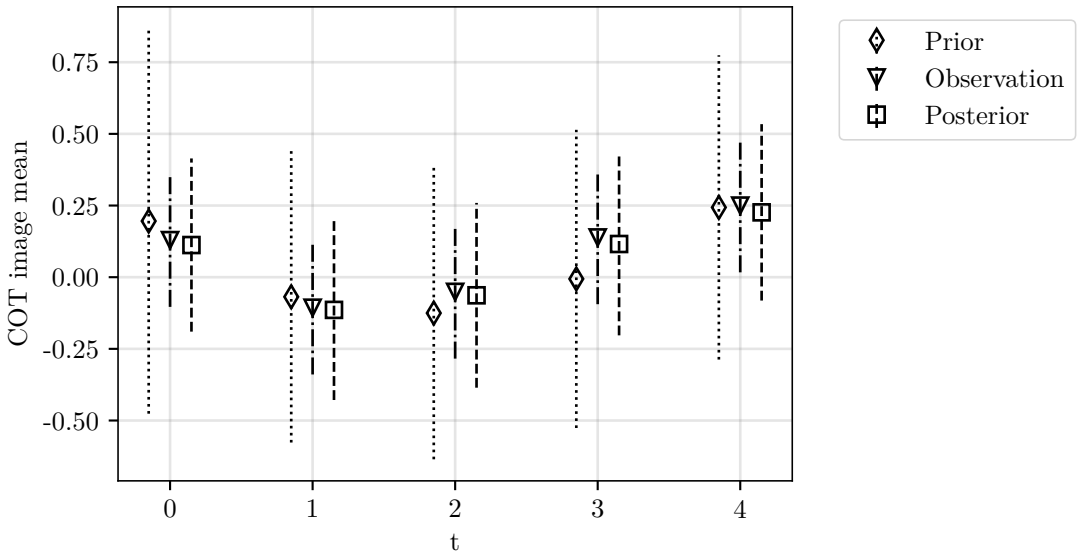

Figure 4.8: Mean uncertainty estimates of the neural network computed prior forecast, observations, and posterior estimates given by the ETKF of the example in Figure 4.7. The error bar widths correspond to two times the spatial mean of the standard deviations. For the observation, the standard deviation was assumed to be 0.23.

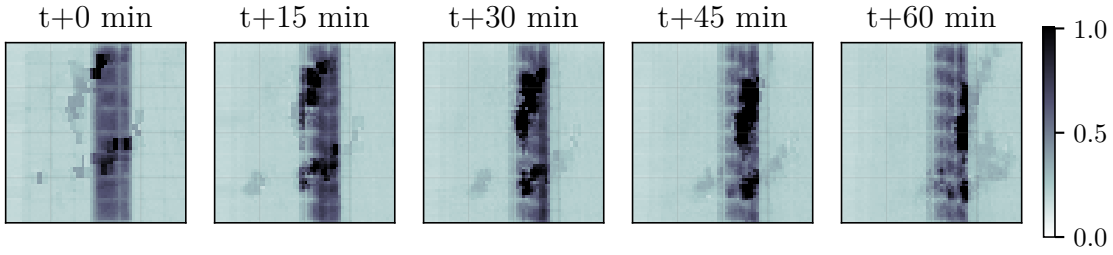

Figure 4.9: Two standard deviations of the neural network ETKF model of the images in Figure 4.7. Missing observations lead to higher uncertainty, and also, due to the use of inflation techniques, uncertainty is larger in the areas with more clouds.

5 Concluding Remarks

In this work, an existing neural network architecture was adapted for use in data-independent physics-based advection modelling of spatio-temporal data. The neural network was used as a state evolution operator in an ensemble-based Kalman filter model. The developed neural network trained with a small dataset of satellite measurements of clouds was shown to generalise well for general data evolving according to the advection equation. The neural network exhibited comparable performance to a classical optical flow method. Inside the Kalman filter, the neural network could model the state evolution and thus fill simulated missing observations. In these experiments, the concept of using neural networks in physical modelling was validated along with using it as a part of another modelling system. While the data used in numerical experiments was limited to the cloud optical thickness observations, similar modelling could have applications across multiple domains. The advection model was chosen for the generality and easy understanding. However, similar neural network frameworks could also be applied in other modelling of other phenomena where existing methods are not as well established.

While considerable work remains to follow up with what has been done for this thesis, the results were already encouraging. As the neural network was mostly a reused image recognition architecture with limited tuning, there could be considerable low-hanging fruits to improve its performance. For this work, the physics-based interpretability of the neural network model was a primary goal, so engineering the network further with performance in mind and with more sophisticated training could easily provide superior results. In application-specific tasks, neural networks predicting outputs directly without physics-based modelling have gained popularity. However, due to the lack of interpretability and for problems where mathematical models exist, similar neural network-assisted modelling approaches could be beneficial in generalising and constraining neural network models.

In many ways, the application of Kalman filtering in this work was limited and could not show its full potential. While the simulated examples demonstrated its ability to fill missing values in partial observations, more sophisticated treatment of nonstationary inverse problems related to the observation operator in the state-space model could be useful in more complex problems. For instance, data fusion of multiple indirect data sources, along with the neural network-assisted modelling studied here, could contribute to meaningful improvements in the accuracy of many applications. The concept of a data-independent neural network trained to learn a mathematical model instead of specific data would be advantageous in these cases. The training data could be selected or even simulated based on the mathematical model to account for potentially insufficient actual observations.

Even with the wide use and theoretical validity of the ensemble-based Kalman filters, major drawbacks remain to be considered. The accurate representation of the uncertainty of state estimates requires handcrafted engineering of the ensembles with multiple unknown parameters. Nevertheless, as larger state and observation spaces render the classical Kalman filter infeasible, the ensemble methods offer a valid workaround. An alternative for applications similar to the one presented here

could be to use appropriate dimension reduction techniques to continue using the classical Kalman filter for higher dimensional problems. In addition, the filtering problems could be reformulated to represent the actual task better. As the neural network state operator models the velocity field at a few control points, considerable uncertainties are related to the velocity fields. Thus, a potential reformulation would be incorporating the velocity field parameters into the probabilstic state vectors.

For meteorological applications, prediction problems related to the used cloud optical thickness observations could also be valuable, even though less focus was given to that. With reliable COT forecasts, the ever-increasing solar energy forecasts could be valuable information even in short time horizons given the variable power of renewable energy sources. For forecasting problems, the presented model framework could already be used to provide complete and reliable initial state estimates from which probabilistic forecasts could then be produced with suitable ensemble adjustments.

In conclusion, the findings of this work could guide further work in multiple directions. For example, in the study of neural network-assisted modelling, similar techniques could be researched for other physics-based problems, where existing methods are slow or difficult. For example, in more complicated problems with insufficient training data, a neural network could be trained to mathematical models using simulated or related data to yield speed benefits in modelling. On the other hand, this and similar methods would benefit from further work of verification and optimisation to specific tasks to yield better results. An extension to the work on combining the neural network with Bayesian filtering problems, completing and smoothing historical time series or incorporating more sophisticated observation models would be a possible research topic. As another continuation, future research could tackle prediction problems related to specific data and applications involving the advection-(diffusion) equation. The model of this work could both provide complete initial states and act as a prediction model in the modeling problem at hand.

A Proofs

Proof of Theorem 2.2 following [18]. For the prediction step (2.11), using the Markov property assumption (2.7) that x_{k+1} is dependent only from x_k,

$$\begin{aligned}\pi(x_{k+1}, x_k, y_{1:k}) &= \pi(x_{k+1} \,|\, x_k, y_{1:k})\pi(x_k, y_{1:k}) \\ &= \pi(x_{k+1} \,|\, x_k)\pi(x_k \,|\, y_{1:k})\pi(y_{1:k}).\end{aligned}$$

Substituting this to

$$\pi(x_{k+1}, y_{1:k}) = \int \pi(x_{k+1}, x_k, y_{1:k})\, dx_k,$$

and solving the conditional distribution $\pi(x_{k+1} \,|\, y_{1:k})$ gives the prediction step equation (2.11),

$$\pi(x_{k+1} \,|\, y_{1:k}) = \frac{\pi(x_{k+1}, y_{1:k})}{\pi(y_{1:k})} = \int \pi(x_{k+1} \,|\, x_k)\pi(x_k \,|\, y_{1:k})\, dx_k.$$

For the observation update step (2.12),

$$\begin{aligned}\pi(x_{k+1} \,|\, y_{1:k+1}) &= \frac{\pi(x_{k+1}, y_{1:k+1})}{\pi(y_{1:k+1})} \\ &= \frac{\pi(y_{k+1}, x_{k+1}, y_{1:k})}{\pi(y_{1:k+1})} \\ &= \frac{\pi(y_{k+1} \,|\, x_{k+1}, y_k)\pi(x_{k+1}, y_{1:k})}{\pi(y_{1:k+1})} \\ &= \frac{\pi(y_{k+1} \,|\, x_{k+1})\pi(x_{k+1}, y_{1:k})}{\pi(y_{1:k+1})} \qquad (*) \\ &= \frac{\pi(y_{k+1} \,|\, x_{k+1})\pi(x_{k+1} \,|\, y_{1:k})\pi(y_{1:k})}{\pi(y_{1:k}, y_{k+1})} \\ &= \frac{\pi(y_{k+1} \,|\, x_{k+1})\pi(x_{k+1} \,|\, y_{1:k})\pi(y_{1:k})}{\pi(y_{k+1}\, y_{1:k})\pi(y_{1:k})} \\ &= \frac{\pi(y_{k+1} \,|\, x_{k+1})\pi(x_{k+1} \,|\, y_{1:k})}{\pi(y_{k+1} \,|\, y_{1:k})}.\end{aligned}$$

Here $(*)$ is justified by the conditional independence of y_{k+1} from previous observations if x_{k+1} is known. □

Proof of Theorem 2.3 following [34]. The predicted state mean results directly from the assumption of the model operator being an unbiased estimator. The prior distribution covariance requires studying a joint distribution. As the state and observation vectors are assumed to be Gaussian variables, so is the joint distribution of a pair of them. The joint distribution of $\pi(x_k \,|\, y_{1:k})$ and $\pi(x_{k+1} \,|\, y_{1:k})$ is

$$\pi \begin{pmatrix} x_k \,|\, y_{1:k} \\ x_{k+1} \,|\, y_{1:k} \end{pmatrix} \sim \mathcal{N}\left(\begin{bmatrix} x_{k\,|\,k} \\ M_{k+1} x_{k\,|\,k} \end{bmatrix}, \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix} \right).$$

In the decomposed covariance matrix $\Sigma_{11} = P_{k\,|\,k}$ is the covariance of $\pi(x_k \,|\, y_{1:k})$. The covariance of $\pi(x_k \,|\, y_{1:k})$ and $\pi(x_{k+1} \,|\, y_{1:k})$ is Σ_{12}. By assumption, $x_{k+1} \,|\, y_{1:k} = M_{k+1}x_k \,|\, y_{1:k} + w_{k+1}$, where $w_{k+1} \sim \mathcal{N}(0, Q_{k+1})$. Therefore, Σ_{12} can be evaluated using the affine equivariance of covariance and the linearity of expectation by

$$\begin{aligned}
\Sigma_{12} &= \mathbb{E}[(x_k \,|\, y_{1:k} - \mathbb{E}[x_k \,|\, y_{1:k}])(x_{k+1} \,|\, y_{1:k} - \mathbb{E}[x_{k+1} \,|\, y_{1:k}])^\top] \\
&= \mathbb{E}[(x_k \,|\, y_{1:k} - \mathbb{E}[x_k \,|\, y_{1:k}])(M_{k+1}(x_k \,|\, y_{1:k} - \mathbb{E}[x_k \,|\, y_{1:k}]))^\top] \\
&= \mathbb{E}[(x_k \,|\, y_{1:k} - \mathbb{E}[x_k \,|\, y_{1:k}])(x_k \,|\, y_{1:k} - \mathbb{E}[x_k \,|\, y_{1:k}])^\top]M_{k+1}^\top \\
&= P_{k\,|\,k}M_{k+1}^\top.
\end{aligned}$$

By the symmetricity of the covariance matrix, Σ_{21} is then just the transpose of Σ_{12}, $\Sigma_{21} = M_{k+1}P_{k\,|\,k}^\top$. Again the affine equivariance of covariance, Σ_{22} which is the covariance of $\pi(x_{k+1} \,|\, y_{1:k})$ can be written as

$$\begin{aligned}
\Sigma_{22} &= \operatorname{Cov}(x_{k+1} \,|\, y_{1:k}, x_{k+1} \,|\, y_{1:k}) \\
&= \operatorname{Cov}(M_{k+1}x_k \,|\, y_{1:k} + w_{k+1}, M_{k+1}x_k \,|\, y_{1:k} + w_{k+1}) \\
&= M_{k+1} \operatorname{Cov}(x_k \,|\, y_{1:k}, x_k \,|\, y_{1:k})M_{k+1}^\top + \operatorname{Cov}(w_{k+1}) \\
&= M_{k+1}P_{k\,|\,k}M_{k+1}^\top + Q_{k+1} = P_{k+1\,|\,k}.
\end{aligned}$$

The distribution for $\pi(x_{k+1} \,|\, y_{1:k})$ can be given with the previous as

$$\pi(x_{k+1} \,|\, y_{1:k}) \sim \mathcal{N}(M_{k+1}x_{k\,|\,k}, M_{k+1}P_{k\,|\,k}M_{k+1}^\top + Q_{k+1},$$

which concludes the first part.

The observation update step is derived similarly through joint distributions. The joint distribution of the predicted state $\pi(x_{k+1} \,|\, y_{1:k})$ and the predicted observation $\pi(y_{k+1} \,|\, y_{1:k})$ is

$$\pi \begin{pmatrix} x_{k+1} \,|\, y_{1:k} \\ y_{k+1} \,|\, y_{1:k} \end{pmatrix} \sim \mathcal{N}\left(\begin{bmatrix} x_{k+1\,|\,k} \\ H_{k+1}x_{k+1\,|\,k} \end{bmatrix}, \begin{bmatrix} \Sigma'_{11} & \Sigma'_{12} \\ \Sigma'_{21} & \Sigma'_{22} \end{bmatrix} \right).$$

With similar reasoning as above the covariance matrix consists of,

$$\begin{aligned}
\Sigma'_{11} &= P_{k+1\,|\,k} \\
\Sigma'_{12} &= P_{k+1\,|\,k}H_{k+1}^\top \\
\Sigma'_{21} &= H_{k+1}P_{k+1\,|\,k}^\top = H_{k+1}P_{k+1\,|\,k} \\
\Sigma'_{22} &= H_{k+1}P_{k+1\,|\,k}H_{k+1}^\top + R_{k+1}.
\end{aligned}$$

The normal conditional distribution $\pi(x_{k+1} \,|\, y_{k+1}) \sim \mathcal{N}(x_{k+1\,|\,k+1}, P_{k+1\,|\,k+1})$ can be constructed from the above normal joint distribution (proof omitted). The expectation is given by

$$\begin{aligned}
x_{k+1\,|\,k+1} &= \mathbb{E}[x_{k+1} \,|\, y_{1:k}] + \Sigma'_{12}{\Sigma'_{22}}^{-1}(y_{k+1} - \mathbb{E}[y_{k+1} \,|\, y_{1:k}]) \\
&= x_{k+1\,|\,k} + P_{k+1\,|\,k}H_{k+1}^\top(H_{k+1}P_{k+1\,|\,k}H_{k+1}^\top + R_{k+1})^{-1}(y_{k+1} - H_{k+1}x_{k+1\,|\,k}) \\
&= x_{k+1\,|\,k} + K_{k+1}v_{k+1}.
\end{aligned}$$

And the covariance is given by

$$\begin{aligned} P_{k+1\,|\,k+1} &= \Sigma'_{12} - \Sigma'_{12}{\Sigma'_{22}}^{-1}\Sigma'_{21} \\ &= P_{k+1\,|\,k} - P_{k+1\,|\,k}H_{k+1}^{\top}(H_{k+1}P_{k+1\,|\,k}H_{k+1}^{\top} + R_{k+1})^{-1}H_{k+1}P_{k+1\,|\,k} \\ &= P_{k+1\,|\,k} - K_{k+1}H_{k+1}P_{k+1\,|\,k} \\ &= (I - K_{k+1}H_{k+1})P_{k+1\,|\,k}. \end{aligned}$$

This concludes the proof. □

Proof of Lemma 2.1. Proof directly by multiplying with the claimed inverse.

$$\begin{aligned} &(A+UCV)(A+UCV)^{-1} \\ &= (A+UCV)(A^{-1} - A^{-1}U(C^{-1}+VA^{-1}U)^{-1}VA^{-1}) \\ &= AA^{-1} - AA^{-1}U(C^{-1}+VA^{-1}U)^{-1}VA^{-1} + UCVA^{-1} - UCVA^{-1}U(C^{-1}+VA^{-1}U)^{-1}VA^{-1} \\ &= I - U((C^{-1}+VA^{-1}U)^{-1} + C - CVA^{-1}U(C^{-1}+VA^{-1}U)^{-1})VA^{-1} \\ &= I - U(C + (I - CVA^{-1}U)(C^{-1}+VA^{-1}U)^{-1})VA^{-1} \\ &= I - U(C - C(C^{-1} - VA^{-1}U)(C^{-1}+VA^{-1}U)^{-1})VA^{-1} \\ &= I - U(C-C)VA^{-1} \\ &= I \end{aligned}$$

□

B Code Availability

The code made for this work is available at https://github.com/tuukkahi/thesis.

References

[1] Mark Asch, Marc Bocquet, and Maëlle Nodet. *Data assimilation: methods, algorithms, and applications*. SIAM, 2016.

[2] Georgy Ayzel, Tobias Scheffer, and Maik Heistermann. Rainnet v1.0: a convolutional neural network for radar-based precipitation nowcasting. *Geoscientific Model Development*, 13(6):2631–2644, 2020.

[3] Jan Blechschmidt and Oliver G Ernst. Three ways to solve partial differential equations with neural networks—a review. *GAMM-Mitteilungen*, 44(2):e202100006, 2021.

[4] Jean-Yves Bouguet et al. Pyramidal implementation of the affine lucas kanade feature tracker description of the algorithm. *Intel corporation*, 5(1-10):4, 2001.

[5] Daniel J. Butler, Jonas Wulff, Garrett B. Stanley, and Michael J. Black. A naturalistic open source movie for optical flow evaluation. In Andrew Fitzgibbon, Svetlana Lazebnik, Pietro Perona, Yoichi Sato, and Cordelia Schmid, editors, *Computer Vision – ECCV 2012*, pages 611–625, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[6] Alberto Carpentieri, Doris Folini, Daniele Nerini, Seppo Pulkkinen, Martin Wild, and Angela Meyer. Intraday probabilistic forecasts of surface solar radiation with cloud scale-dependent autoregressive advection. *Applied Energy*, 351:121775, 2023.

[7] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[8] Emmanuel de Bézenac, Arthur Pajot, and Patrick Gallinari. Deep learning for physical processes: incorporating prior scientific knowledge. *Journal of Statistical Mechanics: Theory and Experiment*, 2019(12):124009, dec 2019.

[9] Météo-France / Centre de Météorologie Spatiale. Algorithm theoretical basis document for the cloud product processors of the nwc/geo. Technical Report NWC/CDOP2/GEO/MFL/SCI/ATBD/Cloud, Issue 2, Rev. 1, EUMETSAT NWC SAF, 2019.

[10] Geir Evensen. Sequential data assimilation with a nonlinear quasi-geostrophic model using monte carlo methods to forecast error statistics. *Journal of Geophysical Research: Oceans*, 99(C5):10143–10162, 1994.

[11] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3354–3361, 2012.

[12] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[14] Donald O. Hebb. The organization of behavior. *New York*, 1949.

[15] David Hoese, Martin Raspaud, Panu Lahtinen, William Roberts, Lavergne, Stickler Bot, Gerrit Holl, Stephan Finkensieper, Gionata Ghiggi, Adam Dybbroe, Xin Zhang, Mikhail Itkin, Andrea Meraner, BENR0, Antonio Valentino, Nina, Lars Ørum Rasmussen, lorenzo clementi, Martin Valgur, Denis Rykov, Alan Brammer, Bas Couwenberg, Brian Hawkins, Florian Pinault, storpipfugl, owenlittlejohns, Abel Aoun, and Andres Ricardo Pena Morena. pytroll/pyresample: Version 1.26.1 (2023/02/07), February 2023.

[16] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

[17] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[18] Jari Kaipio and Erkki Somersalo. *Statistical and computational inverse problems*, volume 160. Springer Science & Business Media, 2006.

[19] Rudolf. E. Kálmán. A New Approach to Linear Filtering and Prediction Problems. *Journal of Basic Engineering*, 82(1):35–45, 03 1960.

[20] Seppo Linnainmaa. The representation of the cumulative rounding error of an algorithm as a taylor expansion of the local rounding errors. Master's thesis, (in Finnish), Univ. Helsinki, 1970.

[21] Kuo-Nan Liou. *An introduction to atmospheric radiation*, volume 84. Elsevier, 2 edition, 2002.

[22] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5:115–133, 1943.

[23] Leonard A. McGee and Stanley F. Schmidt. Discovery of the kalman filter as a practical tool for aerospace and industry. Technical Report NASA Technical Memorandum 86847, National Aeronautics and Space Administration, 1985.

[24] Subhadip Mukherjee, Andreas Hauptmann, Ozan Öktem, Marcelo Pereyra, and Carola-Bibiane Schönlieb. Learned reconstruction methods with convergence guarantees: a survey of concepts and applications. *IEEE Signal Processing Magazine*, 40(1):164–182, 2023.

[25] Teruyuki Nakajima and Michael D King. Determination of the optical thickness and effective particle radius of clouds from reflected solar radiation measurements. part i: Theory. *Journal of Atmospheric Sciences*, 47(15):1878–1893, 1990.

[26] Seppo Pulkkinen, Daniele Nerini, Andrés A. Pérez Hortal, Carlos Velasco-Forero, Alan Seed, Urs Germann, and Loris Foresti. Pysteps: an open-source python library for probabilistic precipitation nowcasting (v1.0). *Geoscientific Model Development*, 12(10):4185–4219, 2019.

[27] PySteps. `pysteps.motion.lucaskanade.dense_lucaskanade`. https://pysteps.readthedocs.io/en/stable/generated/pysteps.motion.lucaskanade.dense_lucaskanade.html. Accessed: 2023-08-20.

[28] Nigel M Roberts and Humphrey W Lean. Scale-selective verification of rainfall accumulations from high-resolution forecasts of convective events. *Monthly Weather Review*, 136(1):78–97, 2008.

[29] Frank Rosenblatt. Principles of neurodynamics: perceptrons and the theory of brain mechanisms. Technical report, Cornell Aeronautical Lab Inc Buffalo NY, 1961.

[30] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[31] Casper Kaae Sønderby, Lasse Espeholt, Jonathan Heek, Mostafa Dehghani, Avital Oliver, Tim Salimans, Shreya Agrawal, Jason Hickey, and Nal Kalchbrenner. Metnet: A neural weather model for precipitation forecasting. *CoRR*, abs/2003.12140, 2020.

[32] Ruslan L Stratonovich. On the theory of optimal non-linear filtering of random functions. *Theory of Probability and its Applications*, 4:223–225, 1959.

[33] Ruslan L Stratonovich. Conditional markov processes. In *Non-linear transformations of stochastic processes*, pages 427–453. Elsevier, 1965.

[34] Simo Särkkä. *Bayesian Filtering and Smoothing*. Institute of Mathematical Statistics Textbooks. Cambridge University Press, 2013.

[35] Zhou Wang, Eero P. Simoncelli, and Alan C. Bovik. Multiscale structural similarity for image quality assessment. In *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers, 2003*, volume 2, pages 1398–1402 Vol.2, 2003.

[36] Paul Werbos. *Beyond regression: New tools for prediction and analysis in the behavioral sciences*. PhD thesis, Committee on Applied Mathematics, Harvard University, Cambridge, MA, 1974.

[37] Andrew Zammit-Mangion and Christopher K. Wikle. Deep integro-difference equation models for spatio-temporal forecasting. *Spatial Statistics*, 37:100408, 2020. Frontiers in Spatial and Spatio-temporal Research.