Chapter 1
Data assimilation by neural networks on ocean circulation models

Olmo Zavala-Romero[1,2], Dr. Steven Cocke[2], Eric P. Chassignet[2], Alexandra Bozec[1], Jose R. Miranda[1,2]

[1] Department of Scientific Computing, Florida State University, Tallahassee, FL 32306
[2] Center for Ocean-Atmospheric Prediction Studies, Florida State University, Tallahassee, FL 32306

Contents

Abstract Deep learning models have excelled in language processing and computer vision, enabling real-time translation, image classification, and anomaly detection. In ocean sciences, machine learning has shown promise in data assimilation (DA), emulating dynamical models, accelerating processes, or serving as hybrid surrogate models. However, transitioning to full-scale operational ocean models poses challenges not considered in existing works with intermediate complexity. This chapter introduces Convolutional Neural Networks (CNNs) for assimilating sea surface height and temperature observations in the HYbrid Coordinate Ocean Model (HYCOM) for the Gulf of Mexico. Five experiments analyze CNN performance, providing insights for applying CNNs to primitive equations ocean models with real observations and complex topographies. The experiments evaluate CNN architecture, input and output choices, window size impact, and the role of coastlines. Additionally, the speed improvement gained by using CNNs for DA is estimated. This research advances data assimilation techniques in ocean modeling by offering guidelines for employing CNNs in full-scale operational models. It sheds light on optimizing CNN configurations and choices regarding inputs, outputs, window size, and coastal considerations. The chapter highlights the potential benefits and trade-offs of integrating CNNs into the DA process, enhancing ocean forecasting capabilities.

1.1 Introduction

The problem of partial and noisy observations is a long-standing challenge in the field of oceanography. Despite the significant strides made in addressing this issue, there remains a need for more efficient and accurate methods. These advancements are crucial not only for enhancing forecasts but also for representing the attractor of the system more effectively [1]. The utilization of these methods encompasses improved parametrization of unresolved processes in numerical models and enhanced spatio-temporal interpolation of ocean dynamics [2]. This chapter explores the application of machine learning, specifically Convolutional Neural Networks (CNNs), in data assimilation for oceanography, with a focus on the HYbrid Coordinate Ocean Model (HYCOM) in the Gulf of Mexico.

The concept of CNNs, introduced by Fukushima in 1980 [3], was inspired by the human ability for pattern recognition. This idea evolved with the work of Homma et al., who introduced convolutions, and Yann LeCun in 1989, who utilized back-propagation to learn the kernel coefficients. CNNs were primarily developed to analyze image data, where each pixel in the image has a spatial relationship with its neighbors, a feature that is also present in oceanographic data.

Recent works in ocean sciences show that machine learning can be used to emulate the dynamical model [4], substitute the data assimilation step to

speed up the process, or as a hybrid surrogate model to improve a forecast. However, most of these works use ocean models of intermediate complexity with significant simplifications. Overcoming these simplifications when moving to full-size operational ocean models presents a new set of challenges.

This chapter investigates the use of CNNs to assimilate sea surface height and sea surface temperature observations with HYCOM. The CNNs are trained to correct the model error from a 1/25° resolution two-year-long data assimilated HYCOM run with the Tendral Statistical Interpolation System (T-SIS) as the assimilation package. The performance of the CNNs is studied through five controlled experiments that provide intuition on how to apply them in settings with full primitive equations, real observations, and complex topographies.

Our experiments evaluate the architecture and complexity of the CNN, the type and number of observations, the type and number of assimilated fields, the response to the training window size, and the effects of the coastline. Our results show strong correlations between the window size selected to train the CNN, which is not commonly evaluated, and the ability of the CNN to assimilate the observations. Similarly, we found a clear relationship between the complexity of the chosen CNN and its overall performance.

The following sections will provide a detailed introduction to machine learning and its specific models such as the Perceptron, Multi-layer Perceptron (MLP), CNNs, and U-net (Encoder-Decoder and Skip-Connections), as well as an overview of HYCOM and T-SIS. The aim is to provide a comprehensive understanding of these models and their potential applications in oceanography, particularly in data assimilation with the HYCOM model.

1.2 An Introduction to the Hybrid Coordinate System Ocean Model (HYCOM)

The HYbrid Coordinate Ocean Model (HYCOM) is state-of-the-art multi-layer ocean model [5][6][7][8][9]. A key feature of HYCOM is the use of a hybrid vertical coordinate. While the horizontal coordinates are typically Cartesian, the vertical coordinate need not be restricted to represent the vertical distance from a specified origin, the so-called "z-coordinate". In various parts of an ocean basin, the layer flow may be driven more strongly by different processes, which in turn gives preference to the use of a more suitable vertical coordinate. In the open stratified ocean, for example, in stable conditions ocean flow typically follows along layers of constant potential density (isopycnals). For shallow coastal regions, terrain-following coordinates may be more suitable to characterize the flow subject to the kinematic constraint provided by the bathymetry. In the surface mixed layer or where the ocean is un-stratified, fixed pressure level coordinates may better represent the flow. The choices for vertical coordinates for HYCOM is discussed in [6].

The primitive equations of the HYCOM are detailed in [5]:

$$\frac{\partial \mathbf{v}}{\partial t_s} + \nabla_s \frac{\mathbf{v}^2}{2} + (\zeta + f)\mathbf{k} \times \mathbf{v} + \left(\dot{s}\frac{\partial p}{\partial s}\right)\frac{\partial \mathbf{v}}{\partial p} \nabla_s M - p\nabla_s \alpha$$
$$= -g\frac{\partial \boldsymbol{\tau}}{\partial p} + \left(\frac{\partial p}{\partial s}\right)^2 \nabla_s \cdot \left(\frac{\partial p}{\partial s}\nabla_s \mathbf{v}\right) \tag{1.1}$$

$$\frac{\partial}{\partial t_s}\left(\frac{\partial p}{\partial s}\right) + \nabla_s \cdot \left(\mathbf{v}\frac{\partial p}{\partial s}\right) + \frac{\partial}{\partial s}\left(\dot{s}\frac{\partial p}{\partial s}\right) = 0 \tag{1.2}$$

$$\frac{\partial}{\partial t_s}\left(\frac{\partial p}{\partial s}\theta\right) + \nabla_s \cdot \left(\mathbf{v}\frac{\partial p}{\partial s}\theta\right) + \frac{\partial}{\partial s}\left(\dot{s}\frac{\partial p}{\partial s}\theta\right) = \nabla_s \cdot \left(\nu\frac{\partial p}{\partial s}\nabla_s \theta\right) + H_\theta \tag{1.3}$$

where $\mathbf{v}$ is the horizontal velocity vector, s is the vertical coordinate, ζ is the relative vorticity, f is the Coriolis parameter, $\mathbf{k}$ is the vertical unit vector, p is pressure, $M = gz + p\alpha$ is the Montgomery potential, α is the potential specific volume, $\boldsymbol{\tau}$ is the horizontal wind stress at the surface or drag at the ocean bottom, θ are one of two thermodynamic variables, either temperature or salinity, and ν is the eddy viscosity coefficient.

The first equation (1.1) is the momentum equation for the components of $\mathbf{v}$, yielding two scalar equations. The second equation (1.2) is the mass continuity equation. The third equation (1.3) represents two scalar thermodynamic equations, one for each thermodynamic variable. Thus, there are a total of five equations that are being solved.

A unique feature of HYCOM is that the vertical coordinate system can be modified at any given time step during model integration as flow conditions change. This is done through the use of a grid generator.

In addition to the equations above, HYCOM includes parameterizations that take into account other physical processes, such as vertical mixing (possibly due to turbulence), convection and sea ice.

The HYCOM model is a highly configurable model that can be run at a wide range of horizontal resolutions, vertical levels and can be driven using readily available lateral and boundary conditions (e.g., surface wind-forcing, tidal forcing and bathymetry).

1.2.1 HYCOM Data Assimilation System

The HYCOM modeling system in this study utilizes the T-SIS data assimilaton system [10]. In the earliest version of this system, T-SIS followed the classical Kalman filter approach for optimal interpolation. In this approach, it is assumed that the model forecast follows a Markov process, and observations can improve the estimate of the model state, in a least squares sense, taking into account the modeled and observed error covariances as follows:

$$\mathbf{x}_t^f = \mathbf{f}_t\left(\mathbf{x}_{t-1}^a\right) \tag{1.4}$$

$$\mathbf{x}_t^a = \mathbf{x}_t^f + \mathbf{K}_t\left(\mathbf{y}_t - \mathbf{H}_t\mathbf{x}_t^f\right) \tag{1.5}$$

where $\mathbf{x}$ is the model state, f refers to the forecast after application of the forcast operator, $\mathbf{f}$ ("the model"), while a refers to the analysis after observations are assimilated. The matrix $\mathbf{K}$ is commonly known as the *Kalman Gain*, and determines the relative weight given to observations versus forecast state taking into account model and observation error covariances. $\mathbf{H}$ is an observation operator that maps the modeled state variables to the observation variables. In the simple case where the observations are of the complete model state, $\mathbf{H}$ is just the identity operator. In most cases, observations will sample only part of the model state, hence $\mathbf{H}$ will then select the corresponding element(s) of the model state and perform any other transformation that may be needed. The Kalman gain is computed as

$$\mathbf{K}_t = \mathbf{P}_t^f\mathbf{H}_t^T\left(\mathbf{H}_t\mathbf{P}_t\mathbf{H}_t^T + \mathbf{R}_t\right)^{-1} \tag{1.6}$$

where $\mathbf{P}^f$ is the forecast model state error covariance matrix, and $\mathbf{R}$ is the observation error covariance matrix. When the observation errors are high ($\mathbf{R}$ is large), $\mathbf{K}$ gives low weight in the second term in Eq. 1.5, giving the forecast state more weight. On the other hand, when observations are near perfect, $\mathbf{R} \approx \mathbf{0}$, then most weight is given to the observations.

In general, a covariance matrix can be decomposed into a correlation matrix and a diagonal variance matrix:

$$\mathbf{P}^f = \mathbf{D}^{1/2}\mathbf{C}\mathbf{D}^{1/2} \tag{1.7}$$

where $\mathbf{C}$ is a matrix of correlations and $\mathbf{D}$ is a diagonal matrix of variances. The correlations in general are computationally challenging to compute and may be non-stationary, hence requiring frequent re-computation, so an analytical expression based on a second order auto-regressive (SOAR) function is used instead as an approximation [11]. The correlations (elements of $\mathbf{C}$) between gridpoints and observations are computed as the product of C_hC_v where

$$C_h = (1 + s_h)\exp(-s_h) \tag{1.8}$$

$$C_v = (1 + s_v)\exp(-s_v) \tag{1.9}$$

where s_h and s_v are the distances between grid points and observations, scaled by the geometric mean of the horizontal and vertical correlation length

scales, respectively, based on an auto-regression. These expressions could also be used to compute the observation error covariances ($\mathbf{R}$), but currently the observations are considered to be independent, leading $\mathbf{R}$ to be diagonal.

In the newer version of T-SIS, version 2.0 [10], an alternative approach to calculate the Kalman gain is used

$$\mathbf{K}_t = \left(\mathbf{P}_t^{-1} + \mathbf{H}_t^T \mathbf{R}_t^{-1} \mathbf{H}_t\right)^{-1} \mathbf{H}_t^T \mathbf{R}_t^{-1} \tag{1.10}$$

The Kalman gain is computed by first defining the information matrix as $\mathbf{L} \equiv \mathbf{P}^{-1}$ and then the information matrix is modeled by a Gaussian Markov Random Field (GMRF). Each element is conditionally specified based on a set of neighbors. Via spatial regression [9], the neighbors can be determined in a manner that can lead to a sparse matrix for $\mathbf{L}$. This approximation of the inverse error covariance matrix results in a significant reduction in computational expense when used implicitly to solve Eq. (1.10). Speed-ups of an order of magnitude have been reported [10].

After each assimilation step, there are further adjustments to the data in order to accommodate certain HYCOM constraints, such as model layer thickness adjustments, min/max thresholds, hydrostatic checks, and geostrophic balance. The data used in the assimilation have wide temporal and spatial availability. Commonly used data are satellite altimetry and surface temperature estimates as well as in-situ observations from fixed and floating platforms (e.g. floaters and buoys). Table (1.1) shows the data types, source, frequency and spatial characteristics used in this study.

Type	Provider/Source	Frequency	Spatial Variability
Sea Level Anomalies	CLS: https://www.aviso.altimetry.fr/	Daily	Along Track
Sea Surface Temp	NAVOCEANO: https://podaac.jpl.nasa.gov/GHRSST	Daily	Gridded

Table 1.1 Overview of observations in the Gulf of Mexico assimilated by T-SIS.

1.2.2 An introduction to Machine Learning

The term "machine learning" was originally coined by Arthur Samuel in 1959, who developed a checker-playing program that improved its performance with experience [12]. In the 1960s and 1970s, the focus of machine learning research was primarily on symbolic methods, also known as "rule-based" systems [13] [14]. These systems were designed to mimic human problem-solving skills, but were limited in handling uncertainty and learning from data. The 1980s and

1990s saw a shift towards statistical methods and neural networks, inspired by the structure and function of the human brain. The development of the backpropagation algorithm in the 1980s, used to train neural networks, was a significant milestone [14]. However, due to computational limitations at the time, the practical applications of these methods were constrained. The resurgence of neural networks in the 2000s was driven by the availability of large datasets and powerful computational resources [15].

In a nutshell, Machine learning gives computers the ability to learn without being explicitly programmed. It involves the development of algorithms that can learn from data and make predictions or decisions without being explicitly told how to do so. Classically, there are three main types of machine learning: **supervised learning, unsupervised learning,** and **reinforcement learning**.

Supervised learning is the most common type of machine learning, where the algorithms are trained on a set of input data (features) and known outputs (labels). The programs learns to predict the outputs for new, unlabeled data.

Unsupervised learning the algorithm is tasked with finding patterns in the data (e.g. clustering), and the data usually does not have labels.

Reinforcement learning is a type of machine learning where the agent learns to behave in an environment by trial and error. The agent is rewarded for taking actions that lead to desired outcomes and penalized for taking actions that lead to undesired outcomes [16] [17].

1.2.3 Perceptron

The perceptron is one of the simplest forms of a neural network and serves as the foundation for more complex neural architectures. It was first introduced by Frank Rosenblatt in 1958 [18], inspired by earlier work on the McCulloch-Pitts neuron model. The perceptron is essentially a binary classifier that makes its decisions based on a linear predictor function combining a set of weights with the feature vector [19].

The mathematical operation of a perceptron can be described as follows: Given an input vector $x = (x_1, x_2, ..., x_n)$ and a weight vector $w = (w_1, w_2, ..., w_n)$, the output y of the perceptron is determined by the weighted sum of the inputs and a bias b. This can be represented as:

$$y = w \cdot x + b$$

where $w \cdot x$ is the dot product of the weight and input vectors, and b is the bias term which adjusts the threshold [20].

The perceptron learning algorithm involves iteratively adjusting the weights and bias based on the difference between the desired and actual output for training examples. This is done until the perceptron can not improve the

evaluation metric in all training examples, or a maximum number of iterations is reached [21].

1.2.4 Multi-layer Perceptron (MLP)

A Multi-layer Perceptron (MLP) is a class of feedforward artificial neural network [22]. Unlike the simple perceptron, which consists of a single layer and neuron or node, the MLP has one or more layers of hidden nodes between its input and output nodes. The nodes in these layers are fully connected to the nodes in the preceding and succeeding layers [20].

The MLP can be expressed as a function $f : R^d \rightarrow R^o$, where d is the dimension of the input vector and o is the dimension of the output vector. The function f is defined by a series of transformations:

1. An affine transformation $z = Wx + b$, where W is a weight matrix, x is the input vector, b is a bias vector, and z is the output vector.
2. An activation function $h : R \rightarrow R$, which is applied element-wise to the output vector z. The activation function introduces a non-linearity into the model, allowing it to learn complex patterns. Common choices for h include the sigmoid function, the hyperbolic tangent function, and the rectified linear unit (ReLU) function.
3. These steps are repeated for each layer in the MLP, with the output of one layer serving as the input to the next.

The MLP learns by adjusting the weights and biases in the affine transformations based on the error of its predictions. This is done using a process called backpropagation, which is a method for calculating the gradient of the loss function with respect to the weights and biases [21].

The ability of MLPs to approximate any continuous function, given enough neurons and layers, makes them a versatile tool for emulating more computationally expensive data assimilation techniques, such as the Kalman filter and the ensemble Kalman filter. For example, a study by Cintra and Campos Velho (2018) [23] used MLPs to emulate the local ensemble transform Kalman filter (LETKF) for data assimilation in an atmospheric general circulation model. The MLP was trained offline using synthetic observational data, and the resulting MLP-based data assimilation system was able to produce analyses that were very close to those produced by the LETKF, but with reduced computational cost.

1.2.5 Convolutional Neural Networks

Fully-connected neural networks have approximately $m * n + n$ number of parameters for every layer whith m previous nodes and n current nodes. The number of parameters grows rapidily by incorporating additional intermediate layers, rendering them impractical for training large-scale problems encountered in domains like computer vision. However, this limitation is overcome by the introduction of Convolutional Neural Networks (CNNs).

CNNs are able to reduce the number of parameters by sharing weights across different locations in the input data. This is done by using a convolution operation, which takes a small window of the input data and applies a filter to it. The output of the convolution operation is a feature map, which represents the different features that are present in the input data [24].

The convolutional layers extract different sets of features from the input data. The features extracted by the first layer are typically low-level features (such as edges and corners for vision tasks), and the features extracted by the subsequent layers are typically higher-level features (such as objects and faces, also for vision tasks).

Convolutional Neural Networks (CNNs) are a type of NNs designed to process data with a grid-like topology (e.g. images). The name "convolutional" indicates that the network employs a mathematical operation called convolution, which is a specialized kind of linear operation. CNNs are neural networks that use convolution in place of general matrix multiplication in at least one of their layers [24].

The convolution operation is a fundamental building block of CNNs. Let's consider two sequences, $\boldsymbol{a} = (a_i)_{i\in\mathbb{Z}}$ and $\boldsymbol{k} = (k_i)_{i\in\mathbb{Z}}$. Their convolution is the sequence:

$$\boldsymbol{b} = (b_i)_{i\in\mathbb{Z}}\,, \quad b_i := \sum_{j\in\mathbb{Z}} a_j k_{i-j}$$

Here, the sequence $\boldsymbol{k}$ is called the kernel of the convolution. The sequence $\boldsymbol{a}$ is the input and the output $\boldsymbol{b}$ is known as the feature map.

Typically, the kernel has finite length, that is, $k_i = 0$ unless $-n \leq i \leq m$. Hence, the convolution operation can be written as:

$$b_i = \sum_{j=-n}^{m} a_j k_{i-j}$$

This operation operates locally: each b_i is a weighted sum of nearby values of $\boldsymbol{a}$. We often call the vector $(k_{-n}, k_{1-n}, \ldots, k_m)$ a filter of length $n + m + 1$.

The stride of a convolution operation can be changed by replacing the previous equation with:

$$b_i = \sum_{j=-n}^{m} a_{tj} k_{i-j}$$

where $t \in \mathbb{N}$ is the stride. This stride operation essentially skips the feature map over the input.

CNNs are composed of one or more convolutional layers, often followed by a pooling or down-sampling operation, which is then followed by one or more fully connected layers, as in a typical image classification problems.

Pooling or down-sampling operations are used to reduce the spatial size of feature maps while preserving their most important features. This has the effect of reducing the number of parameters in the network and making it more efficient to train [24].

1.2.6 U-net

U-net is a type of convolutional neural network (CNN) that was initially developed for biomedical image segmentation at the Computer Science Department of the University of Freiburg, Germany [25]. The architecture of U-net is symmetric and it consists of two main parts: an encoder (contracting path) and a decoder (expanding path), which gives it a U-shaped architecture.

The encoder part of the U-net architecture is a typical CNN that consists of repeated application of two 3x3 convolutions (unpadded), each followed by a rectified linear unit (ReLU) and a 2x2 max pooling operation with stride 2 for downsampling. At each downsampling step, the number of feature channels is doubled.

The decoder part of the U-net architecture consists of an upsampling of the feature map followed by a 2x2 up-convolution that halves the number of feature channels, a concatenation with the correspondingly cropped feature map from the encoder path, and two 3x3 convolutions, each followed by a ReLU. The cropping is necessary due to the loss of border pixels in every convolution.

The unique feature of U-net is the skip connections between the encoder and decoder parts. These connections provide a path to propagate local information by bypassing the encoder-decoder bottleneck, which is crucial for recovering the fine-grained details that are lost during the encoding process.

The final layer of the U-net is a 1x1 convolution used to map each 64-component feature vector to the desired number of classes.

The operation of a U-net can be represented as follows. Let f_i denote the feature maps at the i-th layer of the network. In the encoder, each layer performs two convolutions with ReLU activations, followed by max pooling:

$$f_i = \max(0, W_i * f_{i-1} + b_i)$$

where W_i and b_i are the weights and biases of the i-th layer, and $*$ denotes the convolution operation. The max pooling operation is applied to the result.

In the decoder, each layer performs upsampling of the feature map from the previous layer, concatenation with the feature map from the corresponding encoder layer, followed by two convolutions with ReLU activations:

$$f_i = \max(0, W_i * (f_{i-1} \oplus f_{i-encoder}) + b_i)$$

where $\oplus$ denotes the concatenation operation.

1.3 Data assimilation with Convolutional Neural Networks

1.3.1 Methods

In this section, we explore the application of Convolutional Neural Networks (CNNs) as a data assimilation technique for ocean models. We evaluate the performance of CNNs through five experiments involving multiple CNN models. These models assimilate both sea surface temperature (SST) and sea surface height (SSH) data, and their results are compared with those obtained through the optimal interpolation method as implemented in T-SIS. The Gulf of Mexico serves as our test case for the numerical ocean model, with a domain extent spanning from 18.09° to 31.96° in latitude and −98.0° to −77.04° in longitude, as depicted in Figure 1.1.

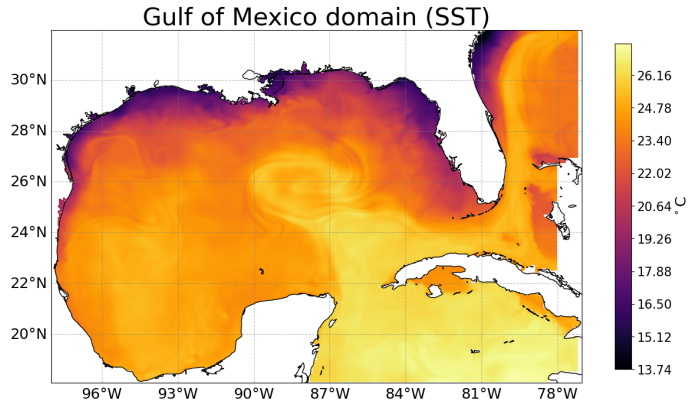

Fig. 1.1 Gulf of Mexico domain used as test case.

The ocean model used is the HYbrid Coordinate Ocean Model (HYCOM) with a spatial resolution of 1/25°. The GOMb0.04 domain is set up with the

high resolution 1km bathymetry of the Gulf of Mexico [26] over a domain going from 98°E to 77°E in longitude and from 18°N to 32°N in latitude. With 41-hybrid layers in the vertical, the latest version of the HYCOM model (2.3.01: https://github.com/HYCOM/HYCOM-src) is forced at the surface with the CFSR/CFSv2 hourly atmospheric forcing. The lateral open boundaries are relaxed to daily means of the global HYCOM GOFS3.1 reanalysis https://www.hycom.org/dataserver/gofs-3pt1/reanalysis. The initial conditions are taken from a 20-year reanalysis created with the same configuration.

The Tendral Statistical Interpolation (T-SIS) package www.tendral.com/tsis, described in section 1.2.1, is used with HYCOM to produce the hindcast. The basic functionality of the package is multivariate linear statistical estimation given a predicted ocean state and observations. To optimize the system's performance for the HYCOM Lagrangian vertical coordinate system, subsurface profile observations are first layerized (re-mapped onto the model hybrid isopycnic-sigma-z vertical coordinate system) prior to assimilation. The analysis procedure then updates each layer separately in a vertically decoupled manner. A layerized version of the Cooper and Haines (1996) procedure is used to adjust model layer thicknesses in the isopycnic-coordinate interior in response to SSH anomaly innovations. Before calculating SSH innovations, the mean dynamic topography (MDT) is added to the altimetry observations. A MDT derived from a 20-year freerun of the GOMb0.04 configuration is used for converting SLA to SSH. The multiscale sequential assimilation scheme based on a simplified ensemble Kalman Filter [27, 28] will be used to combine the observations and the model to produce best estimates of the ocean state at analysis time.

This assimilative ocean model configuration is run initially for two years, 2009 and 2010, yielding a total of 730 daily outputs. These outputs serve to train and validate the proposed CNNs. For each day, the increment fields of SSH and SST, $\mathbf{K}_t(\mathbf{y}_t - \mathbf{H}_t\mathbf{x}_t^f)$ from equation 1.5, the background state $\mathbf{x}_t^f$, and the observations $\mathbf{y}_t$ are utilized to train the CNN models.

The CNNs' performance is assessed through five controlled experiments designed to reveal their expected behavior in practical operational settings with full primitive equations, real observations, and complex topographies. These experiments investigate the CNNs' response relative to the size of the spatial windows used for model training, the complexity of the CNN architecture, the number and types of ocean fields used as input and output fields, and the permissible ocean percentage in the training examples.

Window size. The first experiment investigates the performance of the CNNs concerning the size of the spatial dimensions and the number of training examples. Training a CNN within a fixed domain does not guarantee that the model will generalize well to other domains. Even with translational invariant convolutional layers, the models may learn specific relationships with the masked land areas in the full domain, making it challenging to generalize to domains with different coastlines and ocean dynamics. Conversely, training

our models with the whole domain means each training example corresponds to one day from our model run, totaling only 730 days with our two-year daily run. Alternatively, selecting sub-windows within our model as training examples may increase model generalization and the number of examples during training, but possibly at the cost of hindering the models from learning specific relationships found in the entire domain.

Training a model with the full domain provides just one training example per day. When training with smaller window sizes, the total number of training examples is determined by how many sub-windows can fit within our domain. For each dimension, the total amount of sub-windows that can be selected is given by:

$$\text{Number of training examples} = D_s - W_s + 1 \tag{1.11}$$

Here, D_s represents the dimension size, and W_s denotes the window size. For instance, if the domain size is 10×10 and the window size is 5×5, we can fit a total of 6 windows in each dimension, or a total of 36 different examples. In our experiments, when training the networks with a window size smaller than the full domain, 10 random windows are selected for a given day, and each epoch is completed after 1000 of these randomly selected windows are generated. The experiment investigates four different window sizes: 384×520 (whole domain), 160×160, 120×120, and 80×80.

CNN complexity. The second experiment evaluates the performance of the CNNs for data assimilation concerning the complexity of the CNN architecture. Five different models are evaluated using two CNN architectures. The first four models follow a simple CNN architecture, which we refer to as *SimpleCNN*. Models from this architecture are built by stacking convolutional layers with an increasing number of filters. Each hidden convolutional layer employs a ReLu activation function, and the last two convolutional layers contain a single filter and a linear activation function. The four models using this architecture vary in the number of hidden convolutional layers with 2, 4, 8, and 16 layers. The second architecture tested follows the encoder-decoder architecture with skip connections from the U-Net [29]. For this architecture, one model with three levels and 18 CNN layers is evaluated. Figure 1.2 presents detailed information on this model, where all CNN layers except the last one use the ReLu activation function.
Table 1.2[h] shows the names, number of hidden layers, and number of filters used at each hidden layer for the five model architectures tested.

Inputs. This experiment investigates the use of multiple ocean fields as inputs in our models. It would be advantageous to have a single deep learning model capable of assimilating multiple observations at the same time. In traditional data assimilation methods, that use an approximation of the model's error covariance matrix for the assimilation, incorporating correlations with

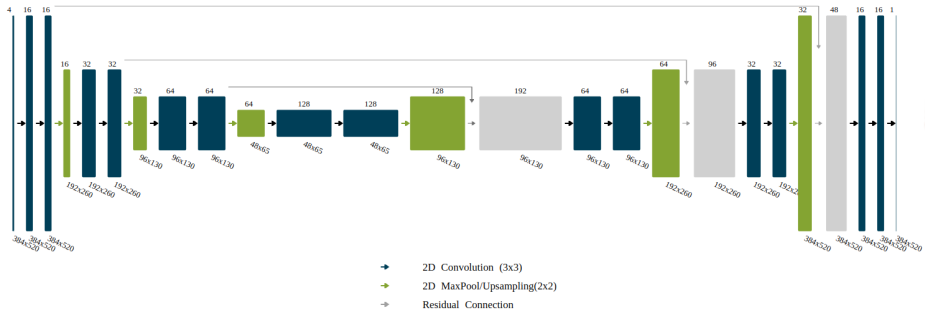

Fig. 1.2 Encoder

Name	CNN Hidden layers	Filter size
SimpleCNN02	2	32, 64
SimpleCNN04	4	32, 64x3
SimpleCNN08	8	32, 64x7
SimpleCNN16	16	32, 64x15
U-Net	14	16x2, 32x2, 64x2, 128x2, 64x2, 32x2, 16x2

Table 1.2 Summary of the number of hidden layers and filters tested in each of the proposed CNN models.

additional fields exponentially increases the size of the error covariance matrix. In this experiment, we vary the inputs by adding new input fields such as SST and the SSH observation errors.

Outputs This experiment evaluates the network's performance concerning the type and number of output fields. As in the previous experiment, having a single model that can assimilate observations into multiple fields is desired. The two fields considered in this experiment are SSH and SST, tested individually and jointly. The primary objective is to investigate whether a moderately complex CNN model can assimilate observations from multiple fields as effectively as from a single field.

Table 1.3 summarizes all the options tested in each experiment. Each tested model is trained five times to gather statistics on the training's consistency and allow a more accurate comparison between the models' performances. A total of 75 CNN models are evaluated in these experiments.

All models are trained using the Adam optimizer [30] with a learning rate of 10^{-3}. The loss function used is the Root Mean Square Error (RMSE) between the increment provided by the CNN and the one generated by the T-SIS model. The RMSE loss is only evaluated in the grid cells where there is ocean, and the CNN models' outputs are always masked by land areas, which are irrelevant for data assimilation in the ocean. All trainings are halted when

Table 1.3 Models tested

Window size	CNN Complexity	Ocean Perc.	Inputs	Output
384x520	SimpleCNN_02	90%	SSH	SSH
160x160	SimpleCNN_04	60%	SSH, SST	SST
120x120	SimpleCNN_08	30%	SSH, SST, SSH Err, SST-Error	SSH and SST
80x80	SimpleCNN_16	0%		
	U-Net			

the error in the loss function has not decreased for 20 epochs, as evaluated in the validation set.

1.3.2 Results

Figure 1.3 shows a performance comparison with respect to the window size used to train the networks. In this experiment, all other parameters remain fixed, with U-Net serving as the default architecture. The SSH increment is used as the target output, and the SSH background state x_t^f and satellite altimeter observations y_t are used as inputs. Furthermore, a mask delimiting areas in the GoM deeper than 200 meters is included as input because T-SIS does not generate any SSH increment for shallow areas. To enable the CNN to learn this restriction, we provided this mask as an additional input channel.

Fig. 1.3 RMSE comparison of CNN models by window size, evaluated in the test dataset.

This experiment reveals a clear relationship between the model's performance and the size of the window used for training. Larger windows yield better performance, and using the entire domain for training achieves the best results.

Figure 1.4 presents the results of the comparison of the CNN architecture and complexity. As before, all other parameters remain fixed. In this case, we used the full domain to train the models, with SSH increment used as the target output, and SSH background state and satellite altimeter observations serving as input.

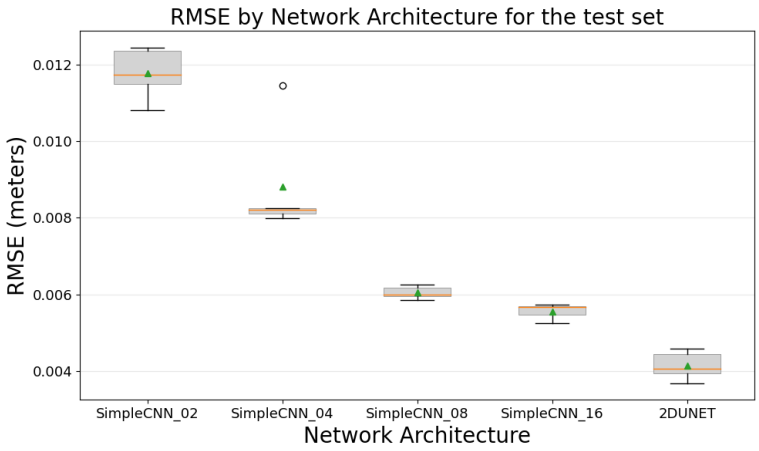

Fig. 1.4 RMSE comparison of CNN models by architecture complexity, evaluated in the test dataset.

The results illustrate that, for the problem of data assimilation in ocean models mimicking the optimal interpolation method, the CNNs' performance improves with increased complexity in their architecture. Two key observations include: the exponential decay observed in the RMSE of the loss function relative to the complexity for the *SimpleCNN* architectures (as the number of hidden layers increases), and how the more advanced U-Net architecture, incorporating batch normalization, skip connections, and an encoder-decoder design, yields the best performance. It's worth noting from this experiment that although there's a clear relationship between the complexity of the CNNs' architectures and the performance obtained, the difference between them is not too big. The *SimpleCNN* architecture with only four hidden CNN layers already approximates the T-SIS data assimilation package with a RMSE of just 8 mm.

Figure 1.5 presents the results of the experiment that compares the percentage of ocean required in the training windows. Recall that the goal of this experiment is to investigate how grid cells with land areas can affect the

training of the CNNs—an problem that is not intrinsic in typical computer vision problems.. For this experiment, the window size is fixed at 160×160, the network architecture is the U-Net, the SSH increment is used as the target output, and the SSH background state and satellite altimeter observations are used as inputs.

Interestingly, we do not identify a clear trend between the performance of the CNNs and the percentage of ocean specified in the training examples. These results suggest that CNNs are not significantly affected by land grid cells when addressing the problem of data assimilation in ocean models.

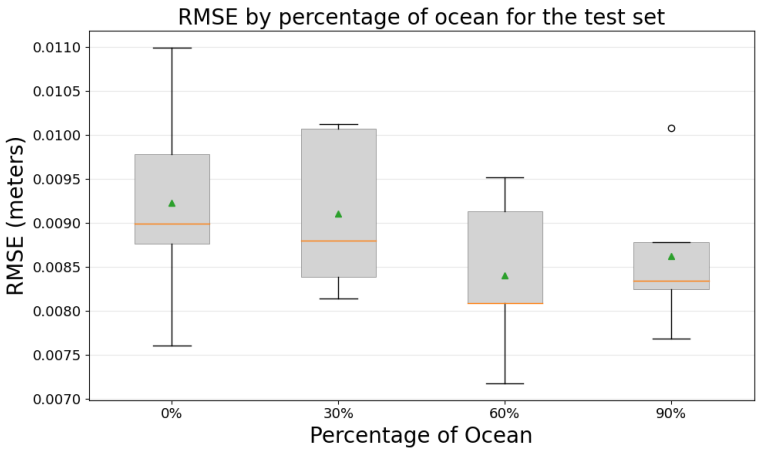

Fig. 1.5 Comparison of the RMSE loss in the test dataset by the percentage of ocean required in the training examples.

Figure 1.6 presents the results of including additional observations as input into the models. For this experiment, the rest of the parameters are as follows: U-Net is used as the network architecture, the entire domain is used to train the models, and the SSH increment serves as the target output. The three tested input observations are the satellite altimeter tracks (SSH), the altimeter tracks combined with SSH and their corresponding observational errors (SSH, SSH-ERR, SST, SST-ERR), and the altimeter tracks combined with SST, but without the error information (SSH, SST).

This experiment reveals how the CNN models might benefit from additional observations as inputs. The performance improves when the error of the observations is included as an input (as an extra channel in the input layer), and the variance of the trained models improves when including SST observation as an input variable. It is expected that the performance improves by including the observational error because T-SIS uses it to compute the increment—the error covariance matrix of the observations, R_t^{-1} in equation 1.6, contains this information. However, it's noteworthy to show that including

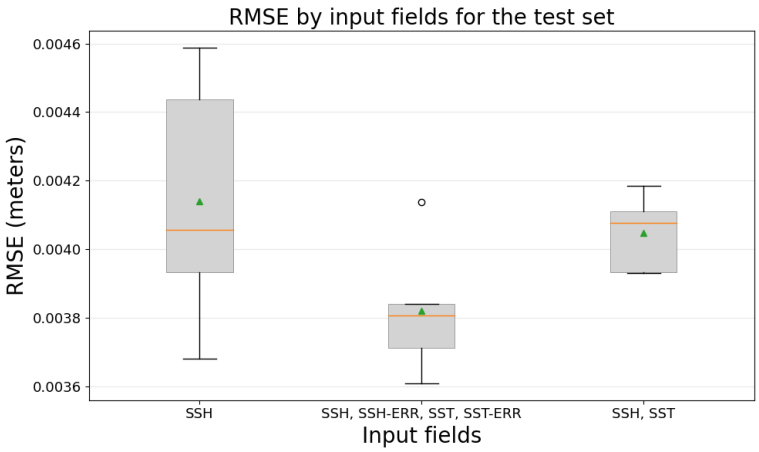

Fig. 1.6 Comparison of the test RMSE loss by the number and types of input fields.

additional SST observations does not affect the model's performance, even though we know that SST is not used by T-SIS to generate the SSH increment.

Finally, figure 1.7 presents the results of testing CNNs to simultaneously generate multiple data assimilation increments, in this case, SSH and SST. The rest of the parameters are as follows: U-Net is used as the network architecture, the entire domain is used to train the models, and the SSH increment serves as the target output. The three output increments tested are the satellite altimeter tracks (SSH), sea surface temperature (SST), and both together (SSH, SST).

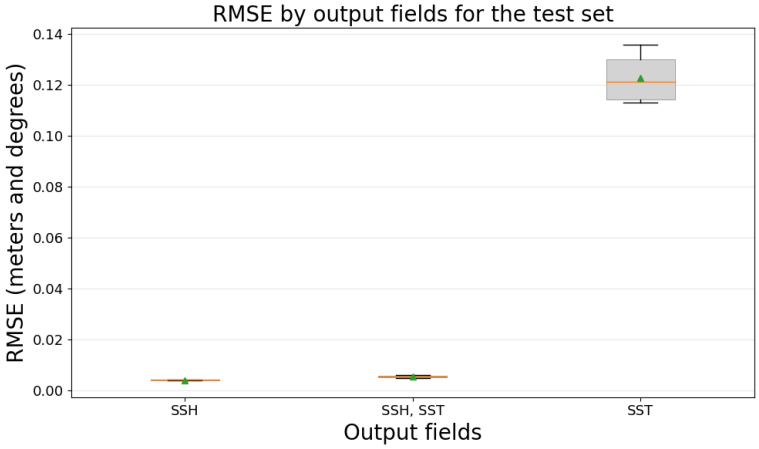

Fig. 1.7 RMSE comparison of CNN models by the number and types of output fields, evaluated in the test dataset.

For this final experiment, it's important to note that the Y-axis is in meters for the first two models, SSH and SSH, SST, but it is in degrees for the last case of SST. The key takeaway from this experiment is that the performance in predicting the SSH increment is not affected when the model is tasked with generating both increments (SSH and SST) simultaneously. This indicates the ability of the CNNs to manage multiple outputs without a significant drop in performance for individual tasks.

1.3.3 Generalization Tests

Following the series of experiments in the previous section, which provide insights in the performance of CNNs in an operational ocean model setting with data assimilation, the best model was selected based on optimal parameters. This model utilized the U-Net architecture, was trained using the entire domain of the Gulf of Mexico (GoM) for training examples, and incorporated the SSH observations, the SSH observation errors, the SSH background state, and a binary mask indicating depths greater than 200 meters as inputs. The desired output was the increment of SSH, essentially the corrections to be made to this field in the model on a daily basis.

Figure 1.8 illustrates a comparison between the SSH increment as predicted by T-SIS and the increment predicted by the CNN model for a specific day, October 27$^{\text{th}}$, 2010, from the test dataset. Generally, the overall predictions are similar, with the RMSE across the entire domain in this example being 3.2 mm.

However, the figure also reveals some discrepancies, primarily at the peripheries of areas where there is an increment. This could potentially be attributed to a hard threshold within T-SIS that doesn't provide any increments beyond a certain distance from the observation. An expected pattern from Figure 1.8 is that the corrections to the model are made predominantly close to the locations of the satellite tracks.

Figure 1.11 depicts RMSE for the entire test set, ranging from October 19$^{\text{th}}$ to December 31$^{\text{st}}$ of 2010, as well as the initial days of the year used for training. The mean RMSE for all the test set is 3.72 mm, while for the days used for training it is 3.51 mm. This suggests that the CNN model is effectively generalizing to unseen examples. However, two points need to be considered in this analysis:

1. The Gulf of Mexico's dynamics do not change rapidly over time. Hence, the dynamical state of the GoM for the test set might be quite similar to the state used for training the model.
2. There is a slight discrepancy in the mean RMSE between the training and validation sets, indicating that a more comprehensive experiment is necessary to understand how effectively the model generalizes to unseen

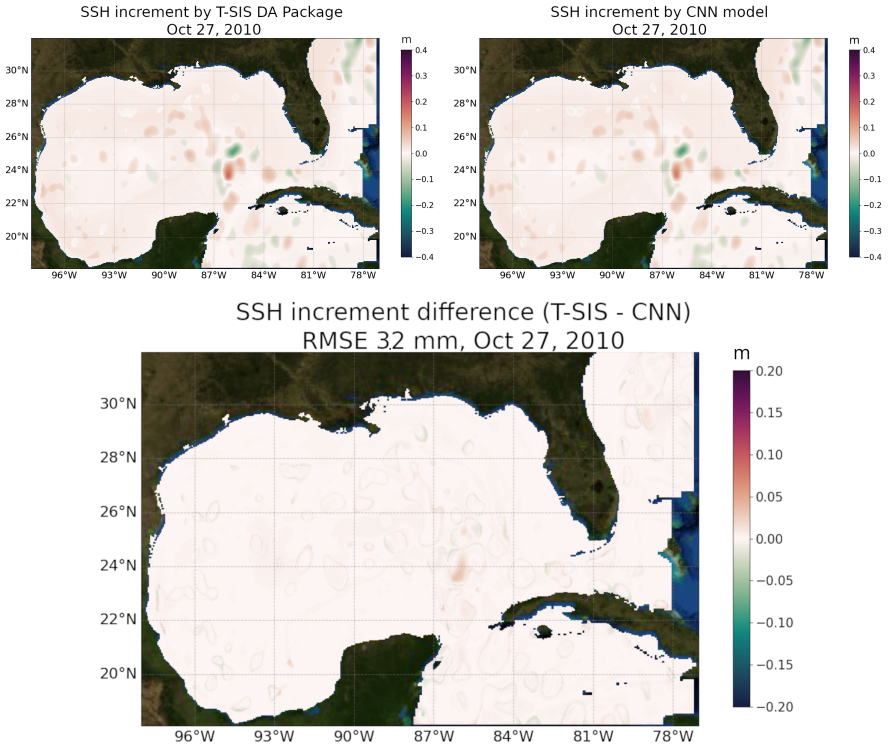

Fig. 1.8 Comparison of the predictecd HYCOM model error (increment) for SSH from T-SIS in the top left panel and the proposed CNN model in the top right panel. The bottom panel in the middle shows the difference between both predictions.

examples where the GoM's dynamical state differs from that in the training set.

To scrutinize the model's ability to generalize across different dynamical states of the GoM, two contrasting years were chosen based on the states of the Loop Current (LC), the key driver of ocean dynamics in the GoM. Notably, it's challenging to confidently predict how a trained model will perform on unseen data. In experiments that use synthetic data, it is simpler to identify examples that fall outside of the training distribution, but in this scenario, the process is not as straightforward. The assumption is that the CNN model will learn to assimilate observations in the GoM akin to the optimal interpolation method and will generalize correctly to data from different years, regardless of the GoM's dynamical state.

The years 2002 and 2006 were selected for this test. In 2002, the LC is primarily in a contracted state, while in 2006, it is predominantly in an extended state, with some eddies being shed throughout the year. New assimilated runs of HYCOM and T-SIS were created for these two years

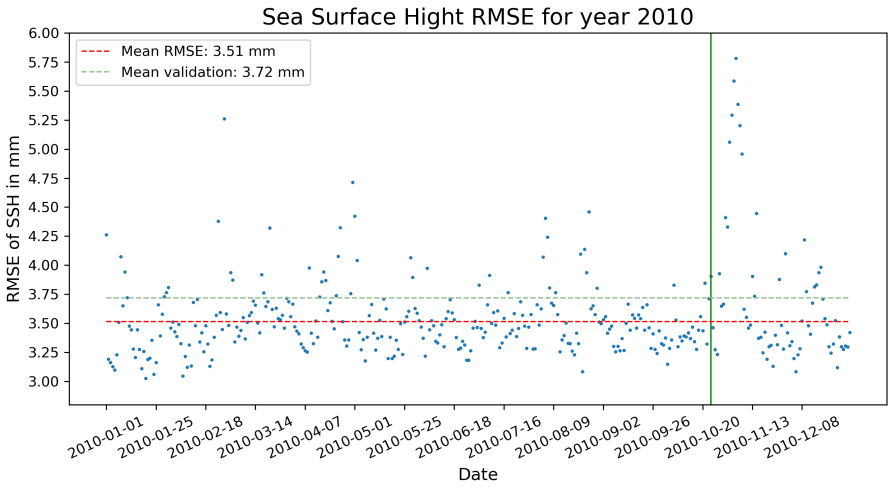

Fig. 1.9 RMSE of the proposed CNN model for the year 2010. The vertical green line indicates the date where the validation dataset starts. The two dashed lines indicate the RMSE of the training and validation sets.

as described earlier, featuring a 1/25° spatial resolution and using NCEP CFSR/CFSv2 as the atmospheric forcings.

Figure 1.10 showcases a day from 2002 and 2006, emphasizing the different dynamical states of the GoM for these two years.

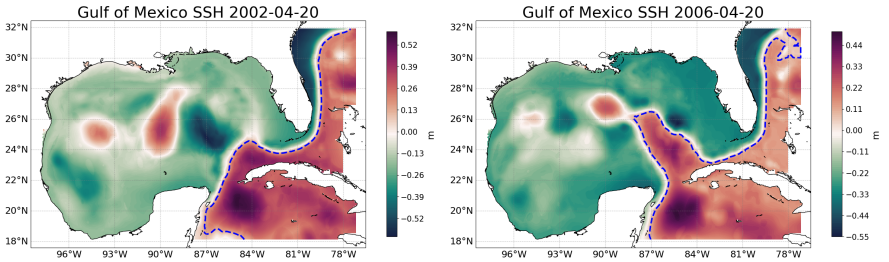

Fig. 1.10 Contrasting dynamical states of the Gulf of Mexico for years 2002 and 2006. The left panel illustrates the retracted Loop Current on April 20$^{\text{th}}$, 2002, while the right panel depicts the extended Loop Current on April 20$^{\text{th}}$, 2006. Both cases are representative of the mean dynamical state of the GoM for that respective year.

The RMSE of the proposed model, trained with data from 2009 and 2010, is 4.39 mm for 2002 and 4.22 mm for 2006. This demonstrates how effectively the model is generalizing to new data and varying states of the GoM. It is anticipated that the model will yield similar results, with an RMSE around 4 mm, for any other timeframe of the GoM. Figure 1.11 presents the RMSE

obtained for every day in 2002 and 2006, along with the mean for the two years. The RMSE has increased from 3.7 mm in the validation set to 4.2 mm in this new generalization test. This underscores the importance of identifying appropriate scenarios to test the generalization of our models. Specifically, in the context of ocean models, it is crucial to evaluate the model in different dynamical scenarios to avoid being satisfied with metrics that may not hold up when using the model operationally.

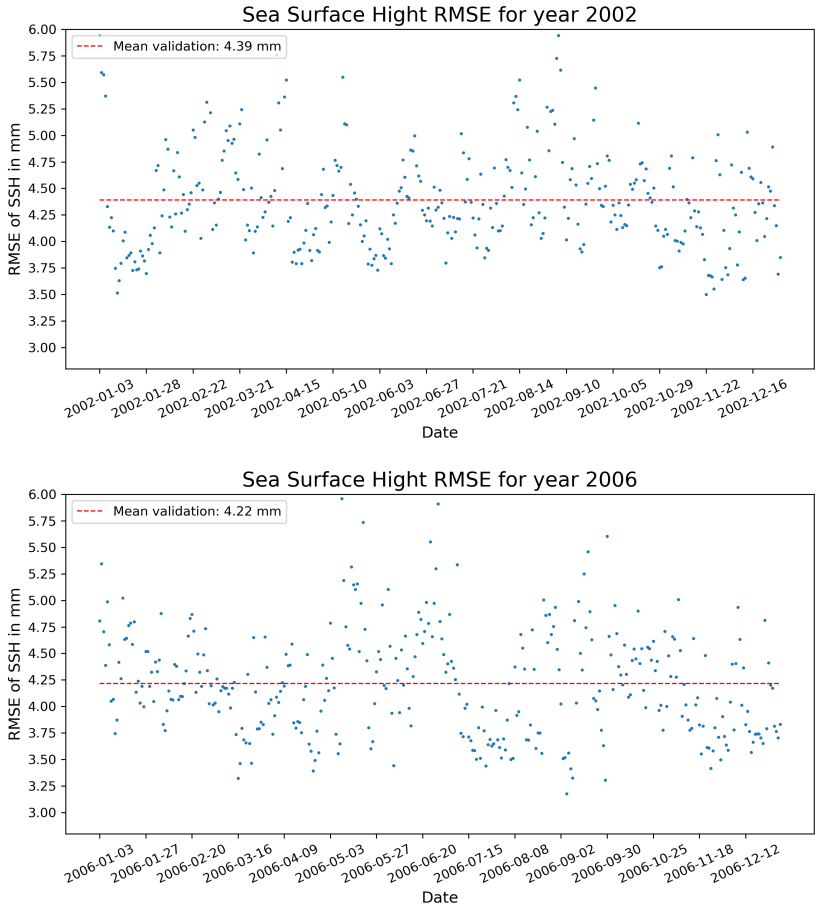

Fig. 1.11 RMSE of the proposed CNN model for the years 2002 on the top plot and 2006 in the bottom. The vertical dashed lines indicate the mean error for the whole year.

1.3.4 Performance Comparison

The primary objective of this study is to investigate the use of CNNs as a more efficient alternative to traditional data assimilation methods in ocean models. However, comparing the performance between the proposed CNN model and the traditional T-SIS optimal interpolation method is not straightforward.

The proposed CNN model is a prototype that only assimilates surface data for a single field at a time (two in some of the experiments). In contrast, the T-SIS data assimilation software is an operational package that simultaneously assimilates all the HYCOM fields, namely temperature, sea surface height, velocity fields U and V, salinity, and the 41 vertical layers of the model.

Furthermore, like most operational data assimilation systems, the T-SIS package is implemented in FORTRAN and runs on tens to hundreds of CPUs stored in clusters at High Performance Computing (HPC) centers. Meanwhile, the proposed model is implemented in Tensorflow with Python and runs on GPUs, which may contain thousands of small processors.

In this performance analysis, the times taken by T-SIS to assimilate a single day of observations in two different settings are compared. The first setting is when it is executed on a cluster with 32 processors at the HPC center in Florida State University, and the second when it is executed at the Narwhal Navy Super Computer with 96 processors.

Assimilating a single surface field using the proposed CNN model takes 0.054 ± 0.005 seconds with the an NVIDIA Quadro RTX 4500 GPU.

To simulate the performance when executed in full 3D (5 fields and 41 vertical-layers), we have two scenarios. The first scenario assumes we will not be able to parallelize our code further (CNN Simulated 3D Sequential), where we need to multiply our times by the number of fields and the number of vertical levels (41*5). The second scenario assumes we will be able to parallelize everything in 3D (CNN Simulated 3D Parallel).

These are the two extremes we can have, and the speed up of the proposed CNN, compared to the 32 T-SIS processors, ranges between 1.8 and 388. Compared with the 96 processors T-SIS version, the speedup ranges between 0.73 (slower) and 150.

As these ranges are quite large, we simulate assimilating all the vertical layers by running our model in a batch of 41. This provides a metric closer to what we would expect by parallelizing our code. For this scenario (CNN Simulated 3D 41 Batch), the time it takes to perform a single day assimilation is 0.36 ± 0.01 seconds. In this case, the expected speed-ups are 58 versus running T-SIS with 32 processors and 22 versus running T-SIS with 96 processors.

Table 1.4 shows the time, in seconds, that the T-SIS package takes to assimilate one day of data when run on an HPC with 32 processors and 96 processors, together with the estimates of times our model would take in the case that we can only parallelize our model for assimilating the data in vertial

layers as a batch and when we assume we will be able to parallelize also the five fields.

T-SIS 32 Procs	T-SIS 96 Procs	CNN Simulated Sequential fields	CNN Simulated Parallel fields	CNN Simulated 41 Batch
21 ± 0.3 s	8.11 ± 0.5 s	11.07 ± 0.1 s	0.054 ± 0.005 s	0.36 ± 0.01 s

Table 1.4 A table comparing the times it takes to simulate a single day of observations using T-SIS, using different number of CPUs, and with the proposed CNN model, with multiple paralellization schemes.

1.4 Final Remarks

In this chapter, we've introduced the fundamental principles of numerical ocean models, supervised machine learning and deep learning to explore the application of CNNs as a DA scheme for ocean models.

A series of experiments were conducted to analyze the performance of CNNs when applied to data generated from ocean models. These experiments evaluated the architecture and complexity of the CNN, the type and number of observations (inputs), the type and number of assimilated fields (outputs), the response to the window size, and the effects of the coastline.

The results show a clear relationship between the window size used to train the CNNs and the achieved performance: larger window sizes generally yield better performance, especially when the full domain is used as the training window. In the context of assimilating surface fields in ocean models, there is also a discernible relationship between the complexity of the CNN network and its performance. Deeper networks obtained better results, and the U-net-based architecture outperformed the others.

We also found that even a shallow CNN with a straightforward architecture could assimilate SSH with an error of just 8 mm. Another critical experiment that assessed the effects of land in the ocean models showed that CNNs are robust to areas of no interest, such as land. By merely replacing the land areas with zeros, the CNNs' performance was not affected by the percentage of ocean used to train the models.

Our experiments also showed how CNNs could effectively utilize additional inputs without degrading performance and how they could assimilate multiple fields simultaneously.

Importantly, we highlight the need to identify a suitable test set to evaluate the generalization capabilities of deep learning models. When working with realistic ocean models on time scales shorter than weeks or months, it might be misleading to randomly select a portion of the training data to test our

models. In these cases, the ocean might not undergo sufficient change within those time scales, leading to overly favorable and misleading results.

We tested the generalization of our proposed model in two years that included different dynamical states of the GoM. Although the error was slightly higher than with the initial test data, an error of 4 mm was identified as the expected value when using our CNN DA method in operational systems.

Finally, we compared the time performance of a conventional DA method, implemented in FORTRAN and executed in a high-performance computing (HPC) cluster, with the proposed CNN method executed on a single GPU. These comparisons are challenging but provide some estimates on the potential time and cost savings achievable with these new technologies.

In our tests, we found that the proposed CNN model can approximate the DA optimal interpolation method implemented in T-SIS with less than a 4 mm error, for SSH, and potentially achieve speedups of 50 compared to running the DA system using a cluster with 32 processors.

References

1. Julien Brajard, Alberto Carrassi, Marc Bocquet, and Laurent Bertino. Combining data assimilation and machine learning to infer unresolved scale parametrization. *Philosophical Transactions of the Royal Society A*, 2021.
2. Maike Sonnewald, Redouane Lguensat, Daniel B. Jones, Peter Düben, Julien Brajard, and Venkatramani Balaji. Bridging observations, theory and numerical simulation of the ocean using machine learning. *Environmental Research Letters*, 2021.
3. Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202, 1980.
4. Chao Ma, Jianchun Wang, et al. Model reduction with memory and the machine learning of dynamical systems. *arXiv preprint arXiv:1808.04258*, 2018.
5. Rainer Bleck. An oceanic general circulation model framed in hybrid isopycnic-cartesian coordinates. *Ocean modelling*, 4(1):55–88, 2002.
6. Eric P Chassignet, Linda T Smith, George R Halliwell, and Rainer Bleck. North atlantic simulations with the hybrid coordinate ocean model (hycom): Impact of the vertical coordinate choice, reference pressure, and thermobaricity. *Journal of Physical Oceanography*, 33(12):2504–2526, 2003.
7. Eric P Chassignet, Harley E Hurlburt, Ole Martin Smedstad, George R Halliwell, Patrick J Hogan, Alan J Wallcraft, Remy Baraille, and Rainer Bleck. The hycom (hybrid coordinate ocean model) data assimilative system. *Journal of Marine Systems*, 65(1-4):60–83, 2007.
8. Eric P Chassignet, Harley E Hurlburt, E Joseph Metzger, Ole Martin Smedstad, James A Cummings, George R Halliwell, Rainer Bleck, Remy Baraille, Alan J Wallcraft, Carlos Lozano, et al. Us godae: global ocean prediction with the hybrid coordinate ocean model (hycom). *Oceanography*, 22(2):64–75, 2009.
9. Toshio M Chin, Arthur J Mariano, and Eric P Chassignet. Spatial regression and multiscale approximations for sequential data assimilation in ocean models. *Journal of Geophysical Research: Oceans*, 104(C4):7991–8014, 1999.

10. Ashwanth Srinivasan, TM Chin, EP Chassignet, M Iskandarani, and N Groves. A statistical interpolation code for ocean analysis and forecasting. *Journal of Atmospheric and Oceanic Technology*, 39(3):367–386, 2022.
11. James A Cummings. Operational multivariate ocean data assimilation. *Quarterly Journal of the Royal Meteorological Society: A journal of the atmospheric sciences, applied meteorology and physical oceanography*, 131(613):3583–3604, 2005.
12. Arthur L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–229, 1959.
13. Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.
14. David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning internal representations by error propagation. In *Parallel Distributed Processing*, volume 1, pages 318–362. MIT Press, 1986.
15. Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.
16. Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2016.
17. Trevor Hastie, Robert Tibshirani, and Jerome H. Friedman. *The Elements of Statistical Learning*. Springer, 2nd edition, 2005.
18. Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.
19. Marvin L. Minsky and Seymour A. Papert. *Perceptrons: An introduction to computational geometry*. MIT Press, 1969.
20. Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.
21. Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.
22. David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning internal representations by error propagation. In *Parallel distributed processing: Explorations in the microstructure of cognition*, volume 1, pages 318–362. MIT Press, 1986.
23. Rosangela Saher Cintra and Haroldo F de Campos Velho. Data assimilation by artificial neural networks for an atmospheric general circulation model. *Advanced applications for artificial neural networks*, 265, 2018.
24. Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks. *ar5iv.org*, 2015.
25. Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention – MICCAI*, pages 234–241, 2015.
26. V Panagiotis. Gulf of mexico high-resolution (0.01∘× 0.01∘) bathymetric grid-version 2.0, february 2013. *Distributed by: Gulf of Mexico Research Initiative Information and Data Cooperative (GRIIDC), Harte Research Institute, Texas A&M UniversityCorpus Christi https://doi. org/10.7266/N7X63JZ5*, 2014.
27. Geir Evensen. The ensemble kalman filter: Theoretical formulation and practical implementation. *Ocean dynamics*, 53:343–367, 2003.
28. Peter R Oke, John S Allen, Robert N Miller, Gary D Egbert, and P Michael Kosro. Assimilation of surface velocity data into a primitive equation coastal ocean model. *Journal of Geophysical Research: Oceans*, 107(C9):5–1, 2002.
29. Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.
30. Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.