NOVA IMS Information Management School

MGI

Master Degree Program in
Information Management

DYNAMIC MODELING OF 3-DIMENSIONAL MARINE FIELDS

Unleashing the Power of Convolutional Transformers
for Advanced Environmental Analysis

Inês Ferreira Rodrigues

Dissertation

presented as partial requirement for obtaining the Master Degree Program in Information Management

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação

Universidade Nova de Lisboa

DYNAMIC MODELING OF 3-DIMENSIONAL MARINE FIELDS
Unleashing the Power of Convolutional Transformers for Advanced Environmental Analysis

by

Inês Ferreira Rodrigues

Master Thesis presented as a partial requirement for obtaining the Master's degree in Information Management, with a specialization in Knowledge Management and Business Intelligence.

Supervisor: Mauro Castelli

Co-Supervisors: Gloria Pietropolli and Luca Manzoni

November 2023

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration. I further declare that I have fully acknowledged the Rules of Conduct and Code of Honor from the NOVA Information Management School.

Lisbon, November 2023

In God we trust. All others must bring data.

—W. Edwards Deming

ACKNOWLEDGEMENTS

The completion of this thesis has been a journey filled with challenges and growth, and I owe its success to the support, encouragement, and guidance of numerous individuals. I want to take this opportunity to express my heartfelt gratitude to all those who have played a significant role in this endeavor.

First and foremost, I would like to extend my appreciation to my thesis advisor, Professor Mauro Castelli. Your expertise and insightful guidance have been instrumental in shaping this research. Your patience and dedication to fostering my intellectual growth are deeply appreciated.

I owe a debt of gratitude to my dearest friends, namely Luis Maia, Sofia Pereira, Tomás Carvalho, Matilde Costa, Tomás Carrondo, Dinis Rocha, Beatriz Mendes, Gonçalo Câmara, Daniel Sørensen, António Sousa, Inês Borges, Mariana Gonçalves and Maria Rodrigues. They have been an incredible source of inspiration and camaraderie throughout this endeavor. From the late-night study sessions, and brainstorming sessions to the celebratory gatherings after milestones achieved and shared laughter during the most stressful times, you have been my companions on this academic voyage. Their understanding, patience, support, and the countless times they lent an empathetic ear when I needed it most, have been a constant source of strength and their presence and the moments shared both inside and outside of academia, have made this journey more meaningful.

I want to express my deepest gratitude to my parents and sister, Rosália, Nelson, and Carolina. Your boundless love, sacrifices, and unwavering faith in my abilities have been my driving force. Your guidance and the values you instilled in me have shaped not only this research but also the person I have become. Your encouragement during the long nights of study and research will forever be etched in my memory.

This journey was not without its challenges, and it was your unwavering support that carried me through the difficult times. Your belief in my dreams and aspirations has been the driving force behind this accomplishment.

This work is dedicated to all those who have supported me on this academic journey, both seen and unseen. Your contributions have left an indelible mark on this thesis and my personal and professional growth.

Thank you,
Inês Rodrigues

DYNAMIC MODELING OF 3-DIMENSIONAL MARINE FIELDS

Unleashing the Power of Convolutional Transformers for Advanced Environmental Analysis

ABSTRACT

Observations obtained through deterministic models play pivotal roles in monitoring marine ecosystems. However, they possess distinct drawbacks. Deterministic models, offer broad coverage but may suffer from inaccuracies. This research aims to address these limitations by developing a deep learning model that seamlessly integrates the output of an existing deterministic model, with a specific focus on biogeochemical variables in the Mediterranean Sea. A novel deep learning architecture is proposed and rigorously tested based on a Hierarchical Multi-Head Self-Attention Transformer and Convolutional Neural Networks. Our findings illustrate the effectiveness of the proposed Convolutional Transformer in successfully replicating the output of the deterministic model. This study sheds light on the model's commendable proficiency, particularly in simulating key variables such as Phosphate and Nitrate, across varying depths. Despite challenges related to variables that tend to zero as we go deeper into the sea, the model's adaptability and precision at surface levels underscore its potential for advancing marine ecosystem monitoring. Quantitatively, the mean squared error values, ranging from 0.0026 to 0.047, affirm the model's consistently low errors across various variables, with Phosphate exhibiting the lowest mean squared error of 0.0026. This outstanding precision in simulating Phosphate highlights the model's capability to perform exceptionally well in capturing the intricate biogeochemical dynamics of the Mediterranean Sea. This research provides valuable insights into the integration of deep learning techniques with existing monitoring methods, paving the way for more accurate and expansive assessments of the Mediterranean Sea's biogeochemical dynamics. The code and results are available here.

KEYWORDS

Deep Learning; Machine Learning; Vision Transformer; Convolutional Neural Networks; Attention; Hierarchical Multi-Head Self Attention.

Sustainable Development Goals

MODELAÇÃO DINÂMICA DE VARIÁVEIS MARÍTIMAS 3-DIMENSIONAIS

Liberando o Poder dos Transformers Convolucionais para uma Análise Ambiental Avançada

RESUMO

Observações obtidas por meio de modelos determinísticos desempenham papéis fundamentais no monitoramento dos ecossistemas marinhos. No entanto, estas observações possuem desvantagens distintas, visto que oferecem cobertura ampla, mas podem ter imprecisões. Esta pesquisa tem como objetivo abordar essa limitação desenvolvendo um modelo de aprendizagem profunda que integra de forma eficaz um modelo determinístico existente, com foco específico nas variáveis biogeoquímicas do Mar Mediterrâneo. Uma nova arquitetura de aprendizagem profunda, baseada num Transformador Hierárquico de Auto-Atenção Multi-Cabeça e Redes Neuronais Convolucionais, é proposta e rigorosamente testada. As nossas descobertas ilustram a eficácia do proposto Transformer Convolucional em replicar com sucesso os resultados do modelo determinístico. Este estudo desca-se pela notável competência do modelo, especialmente na simulação de variáveis-chave como o Fosfato e o Nitrato, em diferentes profundidades. Apesar dos desafios relacionados com variáveis que tendem para zero com a profundida, a adaptabilidade e a precisão do modelo em níveis superficiais destacam o seu potencial para monitorizar ecossistemas marinhos. Quantitativamente, os valores do erro quadrático médio, que variam de 0.0026 a 0.047, confirmam os erros consistentemente baixos do modelo em várias variáveis, sendo que o Fosfato é o que apresenta um menor erro quadrático médio de 0.0026. Essa notável precisão na simulação do Fosfato destaca a capacidade do modelo na captura das intricadas dinâmicas biogeoquímicas do Mar Mediterrâneo. Esta pesquisa oferece *insights* valiosos sobre a integração de técnicas de aprendizagem profunda com métodos de monitorização existentes, abrindo caminho para avaliações mais precisas e abrangentes da dinâmica biogeoquímica do Mar Mediterrâneo. O código e os resultados encontram-se disponíveis aqui.

PALAVRAS-CHAVE

Aprendizagem Profunda; Aprendizagem Automática; Transformador de Visão; Redes Neuronais Convolucionais; Atenção; Transformador Hierárquico de Auto-Atenção Multi-Cabeça.

Objetivos de Desenvolvimento Sustentável

Contents

List of Figures

List of Tables

List of Abbreviations

ANN Artificial Neural Network 10

ARE Absolute Relative Error 31

CHLA Chlorophyll-a 4

CNN Convolutional Neural Networks 7

CV Computer Vision 7

DBN Deep Belief Network 10

DDR4 Double Data Rate Fourth Generation 42

DL Deep Learning 3

DNN Deep Neural Network 11

GELU Gaussian Error Unit 23

GPU Graphical Processing Unit 18

H-MHSA Hierarchical Multi-Head Self-Attention 4

K Key 16

LR Learning Rate 18

LRG Linear Regression 10

MAE Mean Absolute Error 31

MHSA Multi-Head Self-Attention 4

ML Machine Learning 3

MLP Multilayer Perceptron 10

MSE Mean Squared Error 5

N1P Phosphate 4

N3N Nitrate 4

NAR Nonlinear Autoregressive 10

NLP Natural Language Processing 8

NN Neural Network 10

NPP Net Primary Production 4

PSU Practical Salinity Unit 5

Q Query 16

R6C Medium Particulate Carbon 4

RAM Random-Access Memory 42

RMSE Root Mean Squared Error 11

SDA Stacked Denoising Autoencoder 10

SiLU Sigmoid Linear Unit 23

V Value 16

VT Vision Transformer 17

Chapter 1

Introduction

1.1 Motivation

Planet Earth is a blue planet. We can not deny that the ocean covers over two-thirds of our world. Almost 40% of the world's population lives within 100 kilometers of the shore, and the majority of the world's major cities were founded on the ocean's edge. But, the relationship between humanity and the water goes far deeper than mere geographic closeness. There are numerous and intricate links between the seas and human health (Euzen et al., 2017).

The ocean is an important part of the climate system. It transfers large amounts of energy and water to the environment. It moves energy from the tropics to higher latitudes, warming the poles and cooling the tropics. It also contributes to the establishment of regional climates by influencing air temperature and rainfall levels and helps to reduce the greenhouse effect by absorbing one-quarter of atmospheric carbon dioxide and producing a considerable fraction of the oxygen in our atmosphere. With 230 000 marine species reported and over a million to be discovered, the ocean is also a tremendous repository of biodiversity and a site of life.

It is also a crucial component in a wide range of human activities because it contains vast amounts of food, serving as the primary source of proteins for over a billion people; a source of energy, such as renewable energies; minerals; genetics, within which genes and molecules of marine sources are now being applied in the medical and pharmaceutical industries, and there is a tremendous potential for many other sectors such as biofuels and cosmetics; and other resources. These ocean-related activities contribute significantly to the global economy, with the "blue economy" sector accounting for 2.5% of worldwide gross value added ($1,500 billion) (Euzen et al., 2017).

Yet, the ocean changes, transforming as a result of human perturbations of many kinds, such as climate change, acidification, pollution, exploitation, and overexploitation of a variety of resources. These environmental changes have an impact on the seas by causing sea levels to rise, as well as their temperature and acidity. These phenomena, in turn, have an impact

1

on human health, either directly (new opportunistic illnesses) or indirectly (modification of the species distribution and therefore of fishing zones, salinization of soils, the disappearance of arable land) (Euzen et al., 2017).

For that reason, it is critical to increase the capability of monitoring and forecasting the state of the marine ecosystem. This can provide us with crucial information for understanding the dynamics and evolution of this ecosystem.

Originally, marine variable assessments were carried out by dedicated cruises that collected water samples and then examined them in the laboratory. This is still the most precise and dependable method of collecting maritime data today. However, there are important constraints to take into account: the cost of maritime transportation, as well as the spatial and temporal undersampling of these observations (Claustre et al., 2020). This undersampling considerably affects our capacity to quantify important processes in the marine carbon, nitrogen, and oxygen cycles, as well as overall ecosystem change (Pietropolli, Cossarini, and Manzoni, 2022 and Pietropolli, Manzoni, and Cossarini, 2022).

Throughout the last thirty years, new oceanographic devices have been launched to collect subsurface observations, resulting in an unparalleled improvement in ocean monitoring. Satellite sensors appeared in the 1990s, while in-situ autonomous oceanographic instrumentation, such as floats, appeared in the 2000s as part of the Global Ocean Observing System (GOOS) (Pietropolli, Cossarini, and Manzoni, 2022 and Pietropolli, Manzoni, and Cossarini, 2022).

The floats (Roemmich, 2009) are two-meter-long robotic devices that gather marine variable data while diving in the water and changing depth due to buoyancy changes. The main advantage of these devices is that they do not require human intervention and can offer profiles until the battery runs out (typically after 4 or 5 years). Nevertheless, they are costly and hence perform very few measurements in comparison to the entire area to cover (Figure 1.1), which cannot be modeled solely on these observations. Those measures are also less accurate than cruise measurements since sensors degrade with time, rendering the recorded data imprecise or wrong (Pietropolli, Cossarini, and Manzoni, 2022 and Pietropolli, Manzoni, and Cossarini, 2022).

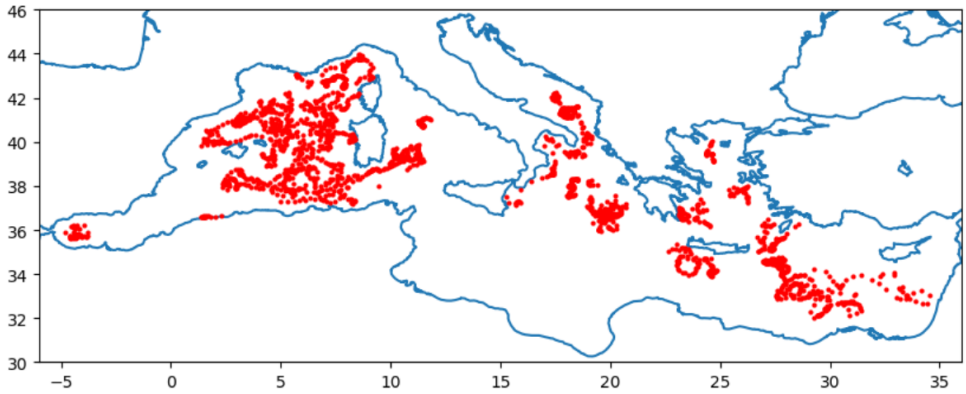


Figure 1.1: Map showing the distribution of the float measurements obtained across the Mediterranean Sea in 2015 (Pietropolli, Cossarini, and Manzoni, 2022).

According to current experience, around 80% of the raw profile data provided from the floats meets established accuracy criteria, and the remaining 20% is routinely rectified inside quality control procedures (Bittig et al., 2018). Temperature, salinity, and pressure are all measured using standard float sensors. Moreover, floats can be outfitted with BCG sensors (which monitor chlorophyll, oxygen, nitrate, pH, and optical variables like bbp700), but the cost increases dramatically. Even if floats boost our comprehension of the ocean, the problem of

undersampling persists since many marine characteristics are measured less often (Pietropolli, Cossarini, and Manzoni, 2022 and Pietropolli, Manzoni, and Cossarini, 2022).

Satellites scan the whole territorial waters with high resolution, but only at the surface, and they are affected by cloud cover. As a result, accessible observational data are typically geographically sparse, with a shortage of series covering over than a few decades (Pietropolli, Cossarini, and Manzoni, 2022 and Pietropolli, Manzoni, and Cossarini, 2022).

Since they can offer reanalyses and forecasts for the whole 3D domain, deterministic models have been employed to mimic the maritime environment. Uncertainties in parameterization and input data, as well as significant computing costs, can, nevertheless, influence their reliability and application (Pietropolli, Cossarini, and Manzoni, 2022 and Pietropolli, Manzoni, and Cossarini, 2022).

The implementation of Machine Learning (ML) techniques provides different and stimulating prospects for improving the ability to merge theory, knowledge, and observations to model the marine environment (Sonnewald et al., 2021, Pietropolli, Cossarini, and Manzoni, 2022 and Pietropolli, Manzoni, and Cossarini, 2022).

1.2 Objectives

The current study seeks to provide a unique Deep Learning (DL) technique to simulate biogeochemical variables in marine areas by combining information from deterministic models with *in-situ* and satellite data.

We aim to utilize a DL framework instead of a deterministic model since a DL structure allows us to use more than one channel inside a tensor, enabling us to model the framework using all of the variables, which a deterministic model does not allow. Employing a DL approach also yields faster results since we do not need to run the entire simulation to generate a forecast.

With this in mind, the proposed DL approach in this dissertation is based on a Convolutional Vision Transformer with Hierarchical Attention (Y. Liu et al., 2022). Since this model was designed specifically to compute relationships between pixels in numerous small sections of an image, we plan to use its architecture to build a model that, given some biogeochemical variables, can simulate a different biogeochemical variable, thanks to the high correlation of this variable to the others, and thus through the relationships between the variables.

We note that using ML to simulate marine environment variables involves numerous obstacles. To begin with, marine datasets include four dimensions (temporal, vertical, and two horizontal) with varying sizes and units (e.g., kilometer and meter respectively for horizontal and vertical spatial dimensions). Furthermore, unlike many ML applications, we cannot rely on ground-truth data in geosciences. Thus, the deterministic model is simply an approximation of the marine ecosystems, with measurements only offering a minimal and scant image of it (Pietropolli, Cossarini, and Manzoni, 2022).

These considerations lead us to use a Convolutional-based architecture, which is inherently suited to handling spatial data. The fundamental concept was to regard horizontal maps of the area under consideration as pictures that depict the maritime environment as if they were images, with the traditional RGB channels replaced with channels expressing marine variables. Likewise, the three color channels are highly interconnected and reliant on one another since they must work together to generate a wide spectrum of hues. Similarly, the variables of the marine environment are also inevitably correlated (Pietropolli, Cossarini, and Manzoni, 2022).

1

We also chose a Transformer-based architecture because it features self-attention, a mechanism developed to collect long-term dependencies and relevant information in input data. At the same time, it enables the model to prioritize relevant input features, allowing it to capture more meaningful representations of the input data (Vaswani et al., 2017).

There are several reasons for using Hierarchical Multi-Head Self-Attention (H-MHSA) instead of the standard Multi-Head Self-Attention (MHSA) inside the Transformer. First, while computing the MHSA, there is a significant computational/space complexity, which is lowered by employing H-MHSA. Moreover, H-MHSA can learn both global relationship modeling and fine-grained information. Lastly, when compared to other exceptional Vision Transformers, one with H-MHSA outperforms the others (Vaswani et al., 2017).

In this thesis, we intend to simulate biogeochemical variables from the Mediterranean Sea using Convolutional Transformers. While this sea is not an ocean, it is regarded as an ocean in miniature because of its unique biogeochemical features, particularly in the eastern basin, which are produced by differences in nutrient supplies in terms of quantity and quality (Bethoux et al., 1999). It is also defined by elevated salinity, oligotrophy, and significant spatial gradients (Schneider et al., 2010 and Pietropolli, Manzoni, and Cossarini, 2022).

The variables that we desire to simulate (the ones that will be the output of our model) are Chlorophyll-a, Net Primary Production, Phosphate, Nitrate, and Medium Particulate Carbon, and the variables we will use to train our model (the ones that will be input variables) are Temperature, Salinity, and Oxygen.

These variables are key indicators used in marine and oceanographic studies to assess various aspects of the ocean's health, productivity, and environmental conditions. Here is a brief explanation of each:

- Chlorophyll-a (CHLA): Chlorophyll-a is a pigment found in algae and plants, including phytoplankton in the ocean. It is a crucial indicator of primary productivity, as it plays a central role in photosynthesis, the process by which organisms convert sunlight into energy;

- Net Primary Production (NPP): Net Primary Production represents the amount of organic matter, primarily in the form of carbon, that is produced by photosynthetic organisms (mainly phytoplankton) in an ecosystem after subtracting the amount used in cellular respiration. It is a measure of the ocean's overall productivity;

- Phosphate (N1P): Nitrate is a form of nitrogen present in seawater. It serves as a nutrient for marine plants, particularly phytoplankton. Monitoring nitrate levels is crucial for understanding nutrient cycles and the factors influencing marine primary productivity;

- Nitrate (N3N): Phosphate is another essential nutrient for marine organisms, particularly in the formation of deoxyribonucleic acid (DNA), ribonucleic acid (RNA) and adenosine triphosphate (ATP). Like nitrate, phosphate levels are monitored to understand nutrient dynamics in the ocean;

- Medium Particulate Carbon (R6C): This represents the amount of carbon contained in medium-sized particles in the ocean. Carbon is a fundamental element in organic matter, and monitoring its distribution helps in understanding carbon cycling in marine ecosystems;

- Temperature: Ocean temperature is a key environmental variable influencing various physical and biological processes. It affects the distribution of marine species, influences ocean currents, and plays a role in the ocean's capacity to absorb and store heat;

- Salinity: Salinity is a measure of the concentration of dissolved salts in seawater, typically expressed in Practical Salinity Unit (PSU). It is a critical parameter influencing the physical properties of seawater, such as its density, which, in turn, affects ocean circulation patterns. Practical Salinity Units are a dimensionless quantity used to express salinity, and they provide a standardized measure that accounts for the variations in the chemical composition of seawater;
- Oxygen: Dissolved oxygen in seawater is vital for marine life, especially for the survival of fish and other aerobic organisms. Monitoring oxygen levels is essential to assess the health of marine ecosystems and identify areas prone to hypoxia or oxygen depletion.

In summary, these variables provide insights into the biological, chemical, and physical characteristics of the ocean, helping scientists and researchers understand its complex dynamics and responses to environmental changes.

Our model demonstrated proficiency, especially in simulating N1P, and showcased adaptability and precision at surface levels. Quantitative evaluation using Mean Squared Error (MSE) indicated consistently low errors across various variables, emphasizing the model's accuracy.

1.3 Structure of the Thesis

This dissertation is divided into six chapters.

The current chapter introduces the study, explaining the reason for this thesis as well as its objectives.

The second chapter delves into the concepts and advances of Convolutional Neural Networks, Self-Attention, and Vision Transformers. We also discuss, in a technical way, several of the hyperparameters that will be utilized in our model, such as the batch size, number of epochs, and learning rate.

The third chapter goes through the basic concept of Hierarchical Multi-Head Self-attention and the modifications that have been made to it. The model proposed in this thesis work is then described, as well as the alterations made to it to match our data.

The fourth chapter starts with an explanation of the data source that will be utilized and how the upcoming dataset was constructed and then moves on to a description of the hyperparameters utilized to train our model.

The fifth chapter presents the results and the performance of what was planned, conceived, and programmed in the previous chapter, as well as an analysis and discussion of it.

The sixth and final chapter discusses the work's conclusion as well as potential future works.

Chapter 2

Literature Review

In this chapter, we will first look at breakthroughs in Convolutional Neural Networks, Self-Attention, and Vision Transformers, before delving into their concepts. We also dive into some DL and ML models used to analyze the sea and we also describe activation functions and introduce some of the hyperparameters that will be used in our model, such as batch size, number of epochs, and learning rate, in detail.

2.1 Related Work

2.1.1 Convolutional Neural Network

In this section, we will mention some Convolutional Neural Networks (CNN) designs that paved the way for CNNs to make significant strides in the field of computer vision.

CNNs arose two decades ago, with the first model built called LeNet-5. This CNN was created to help the recognition of handwritten characters (LeCun et al., 1998). More than ten years later, Alex Krizhevsky et al. made AlexNet to classify photos from the ImageNet dataset and was the first victor of the ImageNet challenge (Krizhevsky et al., 2012). After that, CNNs became a heavily researched subject in the field of Computer Vision (CV), and two years later, VGGNet was developed. This design was developed under the presumption that, up to a given depth, increasing the CNN's depth will reduce the error rate for image classification (Simonyan and Zisserman, 2015). The GoogLeNet network, sometimes referred to as the inception architecture, was introduced by Google in the same year. It was designed to avoid overfitting in networks with multiple deep layers. They apply a variety of ways to overcome this challenge, including 1×1 convolutions and global average pooling. Thanks to this new technique, the network becomes wider rather than deeper (Szegedy et al., 2015). ResNet, one of the most popular CNNs of the time, was later developed. This network was built on the premise that the deeper the CNN, the harder it is to train because the gradient is exploding, so Kaiming He and colleagues invented residual blocks with skip connections to address this issue. These blocks were designed to connect some layers to subsequent levels while skipping the layers in between (He et al., 2016). In 2017, MobileNet, a network made to run on a mobile device, was introduced. This CNN is based on depthwise separable convolutions, that incorporate depthwise

and pointwise convolutions. This architecture significantly reduces the number of parameters (Howard et al., 2017 and Sandler et al., 2018).

2.1.2 Self-Attention

The following models are closest to the one we will realize, which consists of using self-attention to learn feature representations. The primary distinction is that we will employ a system that will learn global associations, using all the inputs, as opposed to a straightforward feature recalibration employing spatial or channel attention, which will be introduced more ahead. In 2015, Jaderberg et al. created STN (Spatial Transformer Network) that explicitly allows the spatial manipulation of data within the network, e.g. scale, rotation, and translation. Together with the introduction of this, they also defined the so-called spatial attention mechanism, which chooses the most relevant regions of a picture (Jaderberg et al., 2015). One year later, SCA-CNN was defined and achieved remarkable results in the task of image captioning compared to what had already been achieved in this area. Chen et al. came with channel-wise attention, which can be viewed as the process of selecting semantic attributes on the demand of the sentence context. SCA-CNN was based on spatial and channel-wise attention, which learns to pay attention to every feature entry in the multi-layer 3D feature maps (Chen et al., 2017). In the same year, Fei Wang and colleagues introduced the attention mechanism in residual networks. To produce this model - Residual Attention Network - they pile up attention modules that will develop attention-aware features (F. Wang et al., 2017). Then, Hu et al., 2020, explored the channel component, since when thinking about the spatial and channel-wise duo of a CNN, the spatial component has already been widely studied in previous works. For this purpose, authors created a new architecture named SENet (Squeeze-and-Excitation Networks), which has as its fundamental unit an SE block that adaptively recalibrates channel-wise feature responses by explicitly modeling interdependencies between channels. This model outperformed the winner of the 2016 ImageNet challenge by around 25% of accuracy. CBAM (Convolutional Block Attention Module) was then built for adaptive feature refinement in feed-forward CNNs. To do that Woo et al. use spatial and channel attention maps that will be then multiplied by the input (Woo et al., 2018). Based again on spatial and channel attention, BAM (Bottleneck Attention Module), which can be applied in any CNN, was developed. This module is placed at each bottleneck of models where the downsampling of feature maps occurs. BAM creates a 3D attention map from a 3D feature map to highlight crucial details (Park et al., 2018). The receptive fields of the neurons in each layer of the CNN are made to be the same size, which is not what happens compared to our brain. To bring CNNs even closer to our neurons, SKNets (Selective Kernel Networks) were invented. Receptive fields are recursively modified by the input in each neuron, just like our brain is modified by a stimulus. Following this idea, the Selective Kernel (SK) block is created, which uses channel attention to fuse numerous branches with various kernel sizes. When gathering these SK units we form a SKNet (Li et al., 2019). Two years ago Zhang et al. came up with ResNeSt which uses channel attention on several network branches to record cross-feature interactions and learn various representations (Zhang et al., 2022).

2.1.3 Vision Transformer

The first Transformers were introduced in 2017, and they are currently one of the most popular Natural Language Processing (NLP) architectures (Vaswani et al., 2017). Initially, they were mainly used in the NLP field e.g. translation and text summarization tasks. They did this by handling sequence-to-sequence problems while easily addressing long-range dependencies. Later, in 2020, Dosovitskiy et al. introduced them to CV by dividing the input, which consisted of

images, into patches that would later be translated into vectors, allowing a transformer to interpret those images as words. When compared to the results produced by some CNN models, this model - ViT (Vision Transformer) - obtained results that were more accurate while also requiring less computational power (Dosovitskiy et al., 2021). After the introduction of the ViT, researchers tried to improve this model and came up with exceptional architectures like T2TViT (Tokens-to-Token ViT) (Yuan et al., 2021), CaiT (Class-Attention in Image Transformers) (Touvron, Cord, Sablayrolles, et al., 2021), DeepViT (Zhou et al., 2021), DeiT (Data-efficient image Transformers) (Touvron, Cord, Douze, et al., 2021), BoTNet (Bottleneck Transformers) (Srinivas et al., 2021), PVT (Pyramid Vision Transformer) (W. Wang et al., 2021), MViT (Multiscale Vision Transformers) (Fan et al., 2021), Swin Transformer (Z. Liu et al., 2021), CoaT (Co-scale conv-attentional image Transformers) (Xu et al., 2021) and HAT-Net (Y. Liu et al., 2022). Yuan et al. realized that ViT, when trained from scratch, produces inferior results, compared to CNNs. To address this issue, they created T2TViT. The big difference concerning ViT is that this model has a layerwise Tokens-to-Token (T2T) transformation to progressively structurize the image to tokens by recursively aggregating neighboring Tokens into one Token (Tokens-to-Token), such that the local structure represented by surrounding tokens can be modeled and tokens length can be reduced. With this, T2TViT not only reached a higher accuracy than ViT but was also able to reduce the computational power. It also achieved a higher accuracy than the CNN model ResNet and accomplished similar results as MobileNets, when trained on the ImageNet dataset (Yuan et al., 2021). Afterward, CaiT (Class-Attention in Image Transformers) was developed and here authors introduced and improved deeper transformer networks. This new model is characterized by two new fundamental concepts: LayerScale and Class-Attention Layers. LayerScale is a layer that assists in controlling the contribution of the residual branches. To enable the branches to behave like identity functions and subsequently allow them to determine the degrees of interactions during training, LayerScale's learnable parameters are first initialized to a modest value. As it is applied per channel, the diagonal matrix also aids in controlling the contributions of the various residual input dimensions. In the Class-Attention Layer, they explicitly separate the transformer layers involving self-attention between patches, from class-attention layers that are devoted to extracting the content of the processed patches into a single vector so that it can be fed to a linear classifier (Touvron, Cord, Sablayrolles, et al., 2021). While studying ViT, Zhou et al. discovered that the performance of ViTs saturates quickly when scaled to be deeper, unlike CNNs. To surpass this issue they came with DeepVit which uses Re-attention. This latest re-generates the attention maps to increase their diversity at different layers (Zhou et al., 2021). To enhance the ViT model, Touvron et al. developed DeiT, which employed a teacher-student approach (Knowledge Distillation introduced by Hinton et al., 2015) unique to transformers. It depends on a distillation token to make sure that the student (smaller model), through attention, learns from the teacher (large model or set of models) (Touvron, Cord, Douze, et al., 2021). To add the bottleneck architecture to ViT, BoTNet was built. In the remaining three bottleneck blocks of a ResNet, Srinivas et al. merely substituted global self-attention for the spatial convolutions (Srinivas et al., 2021). ViT was mainly designed for image classification tasks. PVT, which incorporates the pyramid structure within the transformer architecture, was developed so that Transformers may be used for prediction tasks. In this model, as the Transformer gets deeper, the sequence length gets shorter, allowing for the usage of more fine-grained inputs while also lowering the computing cost. A spatial-reduction attention layer is additionally employed to cut down on the number of resources needed when learning high-resolution features (W. Wang et al., 2021). MViT was designed to recognize video and images. The MViT architecture builds on the core concept of stages. Each stage consists of multiple transformer blocks with specific space-time resolution and channel dimensions. The main idea of Multiscale Transformers is to progressively expand the channel capacity while pooling the resolution from input to output

of the network (Fan et al., 2021). To address the challenges in adapting Transformers from language to vision, the Swin Transformer was built. This hierarchical Transformer uses Shifted Windows. This consists of having a layer that is separated into windows. In each window, Self-Attention is then computed. The windows are shifted in the following layer, resulting in the development of new windows. The calculation of the Self-Attention in the new windows spans the boundaries of the preceding windows in the layer, creating links between them (Z. Liu et al., 2021). Coat introduced the co-scale and conv-attention mechanisms. First, the co-scale mechanism maintains the integrity of Transformers' encoder branches at individual scales, while allowing representations learned at different scales to effectively communicate with each other; we design a series of serial and parallel blocks to realize the co-scale mechanism. Second, we devise a conv-attentional mechanism by realizing a relative position embedding formulation in the factorized attention module with an efficient convolution-like implementation (Xu et al., 2021). In the HAT-Net model, H-MHSA was built to address the low-efficiency defect of the ViT due to the high computational complexity in the MHSA. We will dig deeper into this model ahead (Y. Liu et al., 2022).

2.1.4 Machine Learning and Deep Learning Techniques Used to Analyze the Sea

In recent years, there has been a lot of interest in using ML and DL techniques to analyze the sea, with applications spanning from oceanography and marine biology to environmental monitoring and maritime security. I will discuss some significant methods in this sector further on.

In 2015, Solanki et al. used four techniques - Stacked Denoising Autoencoder (SDA), Deep Belief Network (DBN), Linear Regression (LRG), and Multilayer Perceptron (MLP) - to do predictive analysis of the quality of the water on a dataset from Chaskaman reservoir, Nasik. The variables that they try to predict are turbidity, pH, and dissolved oxygen. The first technique that they use is SDA which is a type of artificial Neural Network (NN) used for unsupervised feature learning and data denoising. It is a variant of the traditional autoencoder, which is a neural network architecture designed to learn efficient representations of data in an unsupervised manner. The key idea behind a denoising autoencoder is to train the network to recover clean data from noisy or corrupted input. The second technique is a DBN is a type of Artificial Neural Network (ANN) that combines the principles of unsupervised and supervised learning to model complex patterns and representations in data. DBNs are composed of multiple layers of stochastic, latent variables, and they have shown significant success in a variety of ML tasks, including dimensionality reduction, feature learning, and classification. The third and fourth techniques are LR and MLP. Their results showed that DL approaches can handle the under and overfitting of predictions as compared to the classical approaches such as MLP and LR (Solanki et al., 2015). The following year, Khan and See used an ANN with Nonlinear Autoregressive (NAR) time series model to forecast the values of water quality metrics based on their current values. In their investigation, they used the following variables: chlorophyll, specific conductance, dissolved oxygen, and turbidity. The study area of their research lay in Island Park village, situated in the state of New York. Their ANN-NAR model had an MSE of 3.7×10^{-4} for turbidity and the best regression value for specific conductance (0.99) (Khan and See, 2016). Prasad et al. created an AutoDL model to estimate water quality in 2022. An AutoDL model finds appropriate highly efficient models automatically spread across various predictive modelling tasks. Total Dissolved Salts, pH, Phosphate, Turbidity, Iron, Nitrate, Chloride, Sodium, and Chemical Oxygen Demand from Korattur Lake in Chennai were among the parameters studied. They concluded that the AutoDL model is as competent as traditional DL models, with enhancement opportunities highlighted (Prasad et al., 2022). In the same year, Chahar et al. tried to predict if the quality of

the surface water in the Ebinur Lake Watershed was good or not using two neural networks - an ANN and a Deep Neural Network (DNN) - and also tried to find which performs better. Their dataset contained the following features: pH, hardness, solids, chloramines, sulfate, conductivity, organic carbon, trihalomethanes, and turbidity. They achieved a startling accuracy of 92% in their ANN model and an accuracy of 89% in the DNN model, concluding that the ANN model worked better (Chahar et al., 2022). In 2023, Rodríguez-López et al. studied the variable chlorophyll-a to estimate the level of water quality in Lake Llanquihue in southern Chile. A 31-year time series of data acquired in situ (1989 to 2020) was used to determine the evolution of limnological parameters at eight spaced stations encompassing all of the lake's key locations, with the year, month, day, and hour time intervals chosen. They used a variety of ML techniques for this task, with XGBoost, LightGBM, and AdaBoost producing the best results, with coefficients of determination ranging from 0.81 to 0.99, Root Mean Squared Error (RMSE) ranging from 0.03 ug/L to 0.46 ug/L, and a mean bias error ranging from 0.01 ug/L to 0.27 ug/L (Rodríguez-López et al., 2023). In the same year, Han et al. introduced the Convolution Recurrent Basis Expansion Analysis Architecture, an upgraded DL approach based on spatiotemporal feature correlation, to forecast the quality of water in the Tanghe Reservoir. This model outperformed the original because it combines the capacity of the CNN structure to extract spatial data with the continuous memory ability of the recurrent neural network structure. Their findings showed that this model is adequate for predicting water quality and has various benefits over typical neural networks (Han et al., 2023).

2.2 Technical Background

2.2.1 Convolutional Neural Networks

Right now, one of the most widely used architectures for image analysis is convolutional neural networks (LeCun et al., 1998). Since they maintain the spatial links between the pixels, these kinds of networks are particularly beneficial.

A CNN can have three types of layers: convolutional, pooling, and fully connected layers (Figure 2.1).

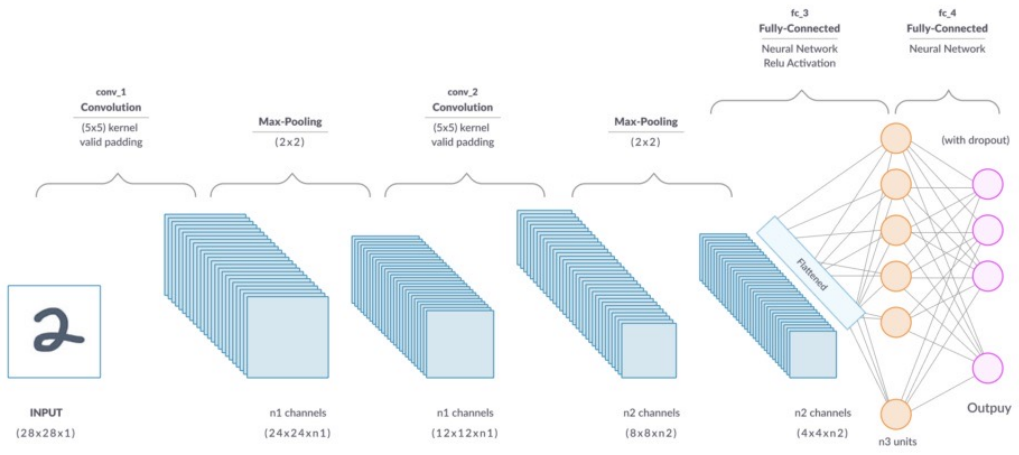


Figure 2.1: CNN Architecture (Source).

In the convolution layer, we have a filter, also known as kernel, that is a matrix of learnable parameters - weights - that will be used to perform a convolution:

$$g(x,y) = \omega * f(x,y) = \sum_{dx=-a}^{a} \sum_{dy=-b}^{b} w(dx,dy) f(x-dx, y-dy) \tag{2.1}$$

where $g(x,y)$ is the output of the convolution (feature map), $f(x,y)$ is the input, and ω is the filter. The elements of the kernel are inside the following range $-a < dx < a$ and $-b < dy < b$. When considering matrices, we have:

$$\begin{bmatrix} x_{11} & x_{12} & \dots & x_{1n} \\ x_{21} & x_{22} & \dots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & \dots & x_{mn} \end{bmatrix} * \begin{bmatrix} y_{11} & y_{12} & \dots & y_{1n} \\ y_{21} & y_{22} & \dots & y_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ y_{m1} & y_{m2} & \dots & y_{mn} \end{bmatrix} = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} x_{(m-i)(n-j)} y_{(1+i)(1+j)} \tag{2.2}$$

where the first matrix is the input and the second the filter (Figure 2.2). Some parameters that can be useful when using Convolutional Layers, are the stride and padding. The first one is a parameter that controls the filter movement along the image, e.g. The filter can then move one pixel or more at a time, skipping the intermediate positions. The number of rows and columns traversed per slide is referred to as stride. Regarding padding, we use this parameter to add extra pixels around the edge of our input image. This is useful since we tend to lose pixels at the edges of our images when applying a convolution. Typically, the excess pixels' values are set to zero. (Figure 2.3)

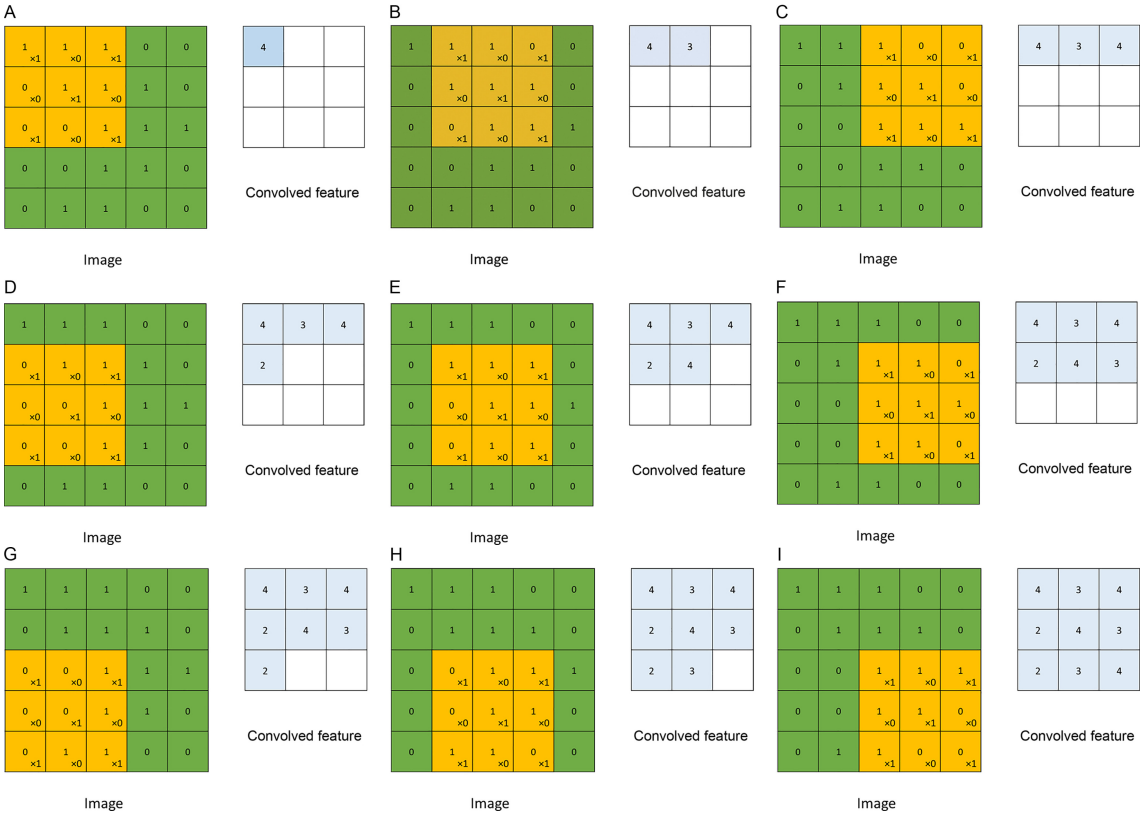


Figure 2.2: 2D Convolution (Source).
The yellow square is a 3×3 filter and their weights can be seen in the right lower corner of each pixel. The green square is a 5×5 input. Here we can see a 2D convolution being made, where in each stage (A, B, C,...) the sum of the result of the filter weight multiplied by the input can be seen in the convolved feature square.

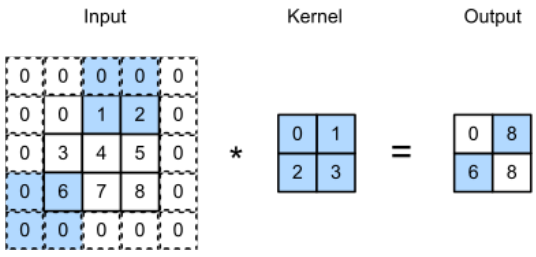


Figure 2.3: 2D Convolution with Padding and Strides of 3 and 2 for height and width (Source).
A 3×3 input can be seen with a padding of 1 (one row of 0's at the top and bottom of the input and one column of 0's at the right and left of the input. Then there is a 2×2 kernel that first will slide two columns to the right until the end of the row. Then, when the row ends, it will slide three rows down and do the same thing as before. If the kernel does not fit in the input it will move to the next row. The result of this 2D convolution can be seen in the output square.

The pooling layer (downsampling) is used to reduce the size of the feature map before it is passed to the FC layer. That is achieved by sliding a filter and then applying a pooling operation. The two most commonly used are max pooling, which uses the maximum value, and average pooling, which uses the average from all the values (Figure 2.4). This layer makes the network less computationally expensive and speeds up the training process.

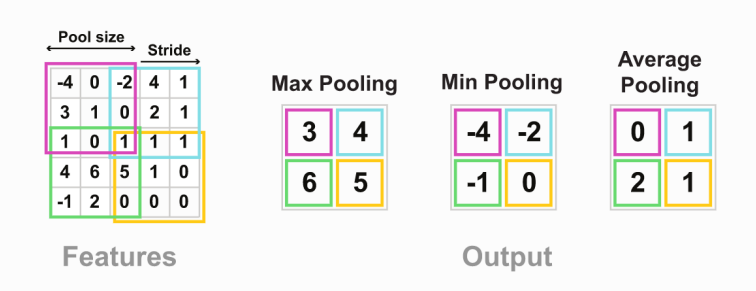


Figure 2.4: Pooling Operations (Source).
In this case, for the pool size, we have a 3×3 filter that has a stride of 2. The numbers inside the filter will have a pooling operation applied to them (max, min, or average). The results of the operations can be seen in the output squares.

The fully connected layers are applied before the output layer and are mainly used for the classification part, i.e. predict the output. This layer in a CNN is very similar to a multilayer perceptron, the only difference being that the input will be a single vector. This vector is achieved by flattening the feature map. After the FC layer, we may additionally add a dropout layer where some neurons will be deleted, which will minimize the complexity of the network and reduce overfitting.

2.2.2 Self-Attention

One of the most important discoveries in DL research during the past ten years is the attention mechanism (Bahdanau et al., 2015). Recent advances in natural language processing have been made possible by it, including the Transformer architecture, which will be discussed in detail later. This mechanism was introduced to focus on a small number of essential aspects, while neglecting others, in deep neural networks.

Regarding self-attention, it is similar to attention. The only difference is that in self-attention the inputs of a sequence interact with one another after deciding which one deserves more attention. In attention, different sequences can interact with each other but the inputs inside of each one of them can not (Figure 2.5).

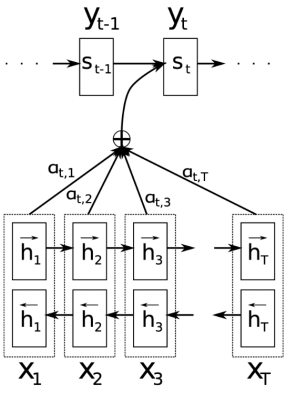
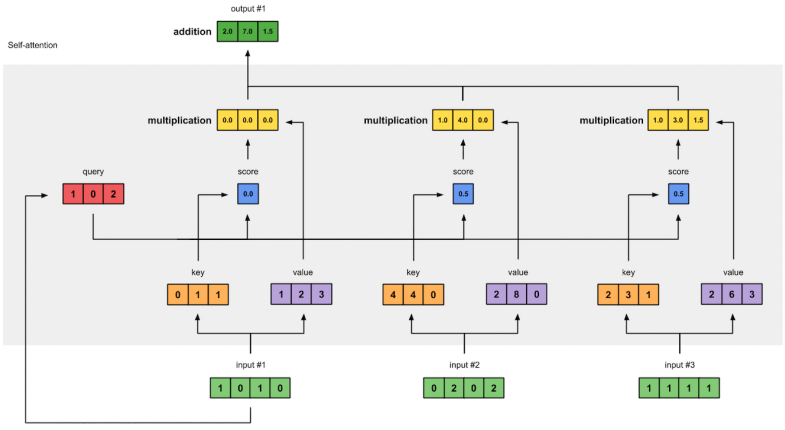
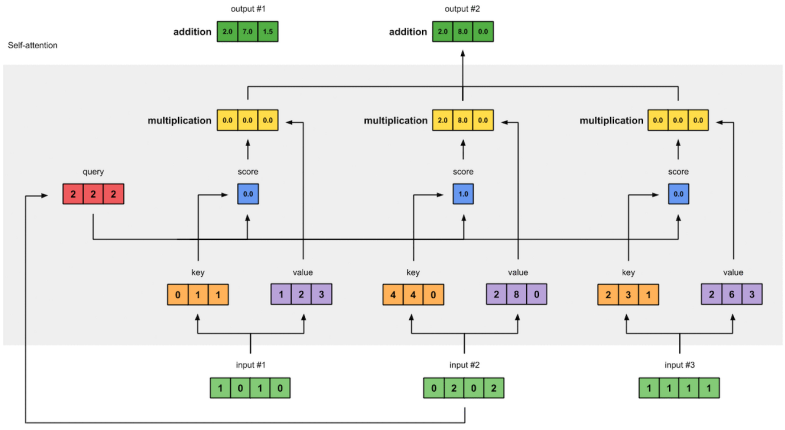


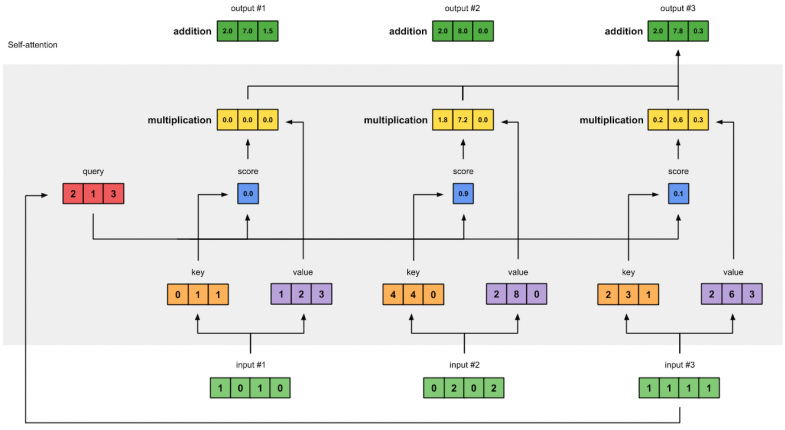
Figure 2.5: Attention Mechanism (Bahdanau et al., 2015).
Generating the t-th target word y_t given a source sentence $(x_1, x_2, ..., x_T)$.



(a) Calculations for output 1.



(b) Calculations for output 2.



(c) Calculations for output 3.

Figure 2.6: Self-Attention Mechanism (Source).

First, to apply the Self-Attention mechanism, it is necessary to introduce three quantities that are defined according to the input under consideration: Key (K), Value (V), and Query (Q). In our example (Figure 2.6), we start from an input of dimension 4 and we want to obtain a representation of it of dimension 3. We will multiply each input by 3 distinct matrices - weights - to obtain these representations. The weight matrix for the K and Q will have a $4 \times D_{kq}$ dimension, while the weight matrix for the V will have dimension $4 \times D_v$, where D_{kq} is the desired dimension for the K and Q, and D_v is the desired dimension for the V. More ahead, we are going to see that the dimension of the V will be the dimension of the output. After computing the K, V, and Q for each input, we will calculate the Attention for each Q. To do that, in this case, we are going to use the following formula:

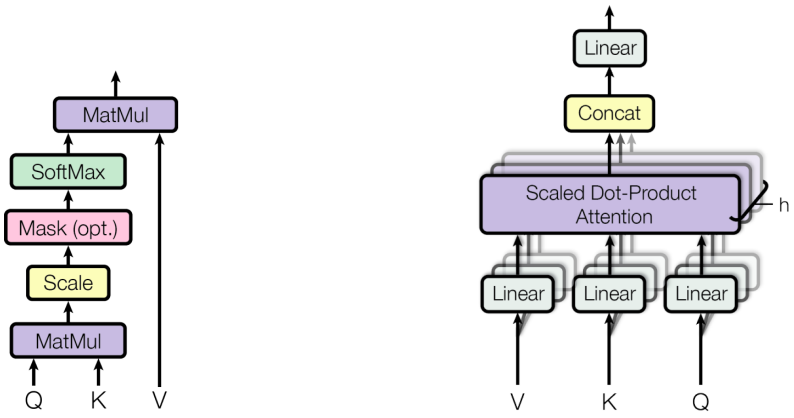
$$Attention(Q, K, V) = softmax(QK^T)V \tag{2.3}$$

The softmax function (equation 2.4) transforms a vector of K real values into a vector of K real values that add up to 1, implying that each K value represents a probability.

$$softmax(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{k_j}} \tag{2.4}$$

for $i = 1, ..., K$ and $\vec{z} = (z_1, ..., z_K)$.

The result of equation 2.3 will be the output.



(a) Scaled Dot-Product Attention.

(b) Multi-Head Attention.

Figure 2.7: Scaled Dot-Product Attention and Multi-Head Attention (Vaswani et al., 2017).

There are various Attention functions. In this example, we use the Dot-Product Attention, but in the Transformer architecture, e.g., it used the Scaled Dot-Product Attention (Figure 2.7(a)):

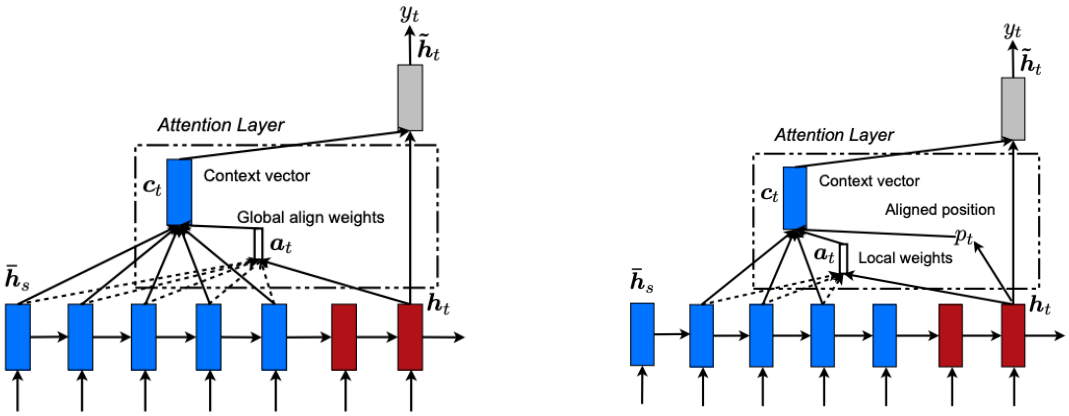
$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{D_{kq}}})V \tag{2.5}$$

Multi-Head Attention is made up of numerous Scaled Dot-Product Attention layers that work in tandem (Figure 2.7(b)).

Another useful concept, discovered by Luong et al., 2015, is the distinction between Global and Local Attention. The first always attends to all inputs, whereas the second only looks at a subset of them (Figure 2.8).

When formulating a prediction for a particular time step, Global Attention is a sort of Attention mechanism that evaluates the complete input sequence. It computes the encoder's hidden states as a weighted sum, where the weights are learned during training, and indicates the relevance of each hidden state for the current time step. This enables the decoder to concentrate on different sections of the input sequence as necessary to produce correct predictions (Figure 2.8(a)).

Local Attention, on the other hand, is a sort of attention mechanism that only evaluates a portion of the input sequence when predicting a time step. This subset is defined using a location-based alignment model, which predicts a set of places in the input sequence that must be addressed (Figure 2.8(b)). Local Attention is especially beneficial for extended sequences where Global Attention is computationally costly and difficult to train.



(a) Global Attention. (b) Local Attention.

Figure 2.8: Global Attention vs. Local Attention (Luong et al., 2015).

Overall, Local Attention is a powerful mechanism for performing sequence-to-sequence tasks that may be integrated with other attention mechanisms, such as Global Attention, to construct a more flexible and robust model.

2.2.3 Vision Transformer

A Vision Transformer (VT) is a widely used model in the CV field to deal with image recognition. It is based on the Transformer built by Vaswani et al., 2017, which was designed initially for NLP activities like machine translation.

A VT receives an image as input and processes it via a succession of layers or blocks, each of which has numerous attention mechanisms. These attention methods enable the model to selectively focus on different sections of the input image, much like we humans selectively focus on specific regions of an image when trying to recognize objects.

While training, the model learns to detect characteristics and patterns in an input picture by optimizing a loss function that evaluates the difference between the predicted and accurate output for a specified task, such as object classification. After finishing training, the model can be used in unseen images.

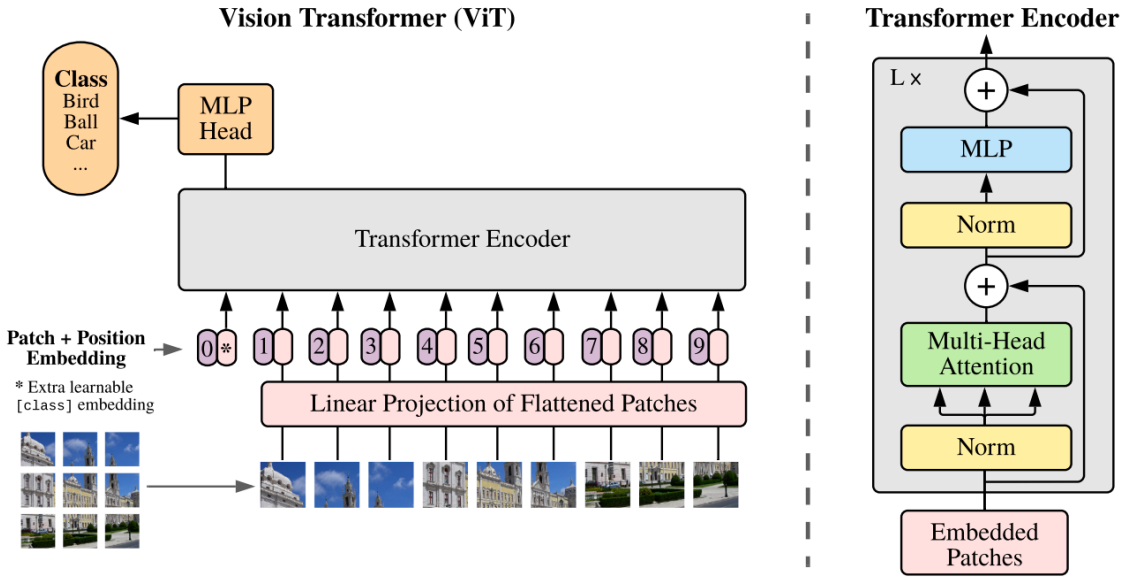


Figure 2.9: ViT architecture (Dosovitskiy et al., 2021).

A 1D sequence of token embeddings is fed into the Transformers described by Vaswani et al., 2017. In a ViT, to use 2D images as an input, these have to be reshaped into a sequence of flattened 2D patches. Assuming that $x \in \mathbb{R}^{H \times W \times C}$ is the input and H, W and C are the height, width, and number of channels of the image, respectively. Then we have the following:

$$x \in \mathbb{R}^{H \times W \times C} \rightarrow x \in \mathbb{R}^{(H \times W) \times C} \tag{2.6}$$

After reshaping the input, the K, V, and Q will be defined just like mentioned in Section 2.2.1, where the weight matrices for the K, V, and Q will have a $C \times C$ dimension. Then, after applying Multi-Head Attention (Figure 2.7(b)), a residual connection will be added. Finally, an MLP with a residual connection will be placed in the end (Figure 2.9).

2.2.4 Hyperparameters

Hyperparameters are configuration settings that are not learned from the data but are set before training the model. There are several key hyperparameters that researchers and practitioners must carefully tune to achieve the desired performance. Some of them are the following:

- Learning Rate (LR): This is a hyperparameter that determines how much we alter the weights of a model. In simple terms, it is how swiftly the model discards previous notions that he learned in favor of new ones. Setting an appropriate LR is critical for training stability and convergence;
- Batch Size: The batch size is the number of samples used in each forward and backward pass during training and it is one of the most significant hyperparameters in DL. Bigger batch size is frequently utilized to train the model since it offers computational speedups from Graphical Processing Unit (GPU) parallelism, but it will result in poor generalization. However, employing a batch corresponding to the whole dataset ensures that the model will eventually reach the global optimal, albeit slowly. Smaller batch sizes result in faster

convergence to excellent solutions. This is because it helps the model to begin learning before seeing all of the data. However, will not converge to the global optimal, but will get close to it. The difference between the batch size and the number of epochs can be seen in Figure 2.10;

- Number of epochs: It is the number of times that the full training dataset will be sent to the model. The choice of this hyperparameter influences how long the model is trained and can impact overfitting;
- Dropout Rate: A regularization technique that randomly drops out a fraction of neurons during training to prevent overfitting;

The values used for these hyperparameters will be mentioned more ahead.

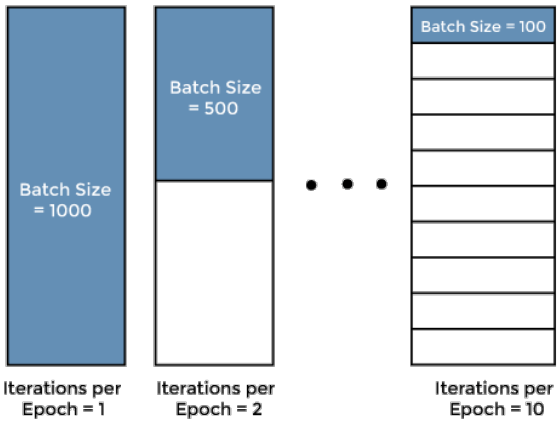


Figure 2.10: Epoch and Batch Size (Source).

Another fundamental component in the training of ML models, including NN and DL models is the Optimizer. It plays a crucial role in the iterative process of adjusting a model's hyperparameters, since it finds the optimal values for these (weights and biases), minimizing the loss function and improving the performance of the model on a given task.

An optimizer is used to implement the gradient descent process efficiently. A gradient descent works by iteratively adjusting the model's parameters in the direction of the steepest decrease in the loss function. The size of each update is determined by the LR. A loss function, also known as a cost function or objective function, quantifies the error or discrepancy between the model's predictions and the actual target values in the training data. So the optimizer calculates the gradients (derivatives) of the loss function concerning each model parameter and based on these gradients, the optimizer updates the model's parameters by subtracting a fraction of the gradient (scaled by the LR) from the current values. This update brings the parameters closer to the optimal values that minimize the loss, as a lower loss indicates better model performance.

There are various optimizers available, each with its approach to updating parameters. Some common optimizers include:

- Stochastic Gradient Descent (SGD): The basic gradient descent algorithm that computes gradients on a subset (mini-batch) of training data in each iteration;
- Adam: A popular optimizer that combines techniques from both momentum and RMSprop to adaptively adjust the LRs for each parameter;

- AdamW: This is an extension of the popular Adam optimizer, primarily used for training deep neural networks, especially in NLP tasks. It incorporates weight decay regularization directly into the optimization process;
- RMSprop: An optimizer that maintains a moving average of squared gradients to adjust LRs, helping with convergence on non-convex loss surfaces;
- Adagrad: An optimizer that adapts the LRs for each parameter based on their historical gradient information.

CHAPTER 3

Methodology

The first section of this chapter discusses the basic notion of H-MHSA and the changes that have been made to it. The second section describes the model suggested in this thesis work as well as the modifications made to it to fit our data.

3.1 Hierarchical Multi-Head Self-Attention

The model chosen to address our problem is the HAT-Net (Y. Liu et al., 2022). As previously mentioned, to reduce the computational complexity Liu et al. came with H-MHSA. In this mechanism, just like the name says, rather than feeding the whole input to the Attention mechanism, we compute it in small amounts of tokens of the input, in a hierarchical way.

Since in our case we have 3-dimensional marine fields instead of 2-dimensional ones - $x \in \mathbb{R}^{H\times W\times L\times C}$, where H is the height, W is the width, L is the length and C is the number of channels - some changes had to be made to the H-MHSA to make it suitable for our input type. First, we want to break down our input into smaller tokens - grids. To do that we will transform our feature map into

$$\begin{aligned} x \in \mathbb{R}^{H\times W\times L\times C} &\longrightarrow x_1 \in \mathbb{R}^{(\frac{H}{G_1}\times G_1)\times(\frac{W}{G_1}\times G_1)\times(\frac{L}{G_1}\times G_1)\times C} \\ &\longrightarrow x_1 \in \mathbb{R}^{(\frac{H}{G_1}\times\frac{W}{G_1}\times\frac{L}{G_1})\times(G_1\times G_1\times G_1)\times C}, \end{aligned} \tag{3.1}$$

where G_1 is the size of the grid. The K, V, and Q will be calculated just like previously mentioned. After having these three representations for our input, Local Attention will be calculated through equation 2.3 - A_1. Then, A_1 will be transformed to the shape of our initial input - x

$$\begin{aligned} A_1 \in \mathbb{R}^{(\frac{H}{G_1}\times\frac{W}{G_1}\times\frac{L}{G_1})\times(G_1\times G_1\times G_1)\times C} &\longrightarrow A_1 \in \mathbb{R}^{(\frac{H}{G_1}\times G_1)\times(\frac{W}{G_1}\times G_1)\times(\frac{L}{G_1}\times G_1)\times C} \\ &\longrightarrow A_1 \in \mathbb{R}^{H\times W\times L\times C}, \end{aligned} \tag{3.2}$$

and after that a residual connection will be added to A_1

$$A_1 = A_1 + x. \tag{3.3}$$

Next, A_1 will be downsampled by G_2 times employing average pooling where the kernel size and the stride are G_2.

$$\hat{A}_1 = AvePool_{G_2}(A_1) \tag{3.4}$$

$\hat{A}_1 \in \mathbb{R}^{\frac{H}{G_2} \times \frac{W}{G_2} \times \frac{L}{G_2} \times C}$ will be used to calculate the K and V, so we can do an efficient calculation of the Global Attention. Then, A_1 and $\hat{A}_1$ will be reshaped to

$$A_1 \in \mathbb{R}^{H \times W \times L \times C} \longrightarrow A_1 \in \mathbb{R}^{(H \times W \times L) \times C} \tag{3.5}$$

$$\hat{A}_1 \in \mathbb{R}^{\frac{H}{G_2} \times \frac{W}{G_2} \times \frac{L}{G_2} \times C} \longrightarrow \hat{A}_1 \in \mathbb{R}^{(\frac{H}{G_2} \times \frac{W}{G_2} \times \frac{W}{G_2}) \times C}. \tag{3.6}$$

This time, the K, V, and Q will be calculated as

$$K_2 = \hat{A}_1 W_2^k, V_2 = \hat{A}_1 W_2^v, Q_2 = A_1 W_2^q, \tag{3.7}$$

where W_2^k, W_2^v and W_2^q are the weight matrices just like previously described. With this, the shape of K, Q and V will be $K_2 \in \mathbb{R}^{(H \times W \times L) \times C}$, $Q_2 \in \mathbb{R}^{(\frac{H}{G_2} \times \frac{W}{G_2} \times \frac{W}{G_2}) \times C}$ and $V_2 \in \mathbb{R}^{(\frac{H}{G_2} \times \frac{W}{G_2} \times \frac{W}{G_2}) \times C}$, respectively. Next, A_2 will be calculated through equation 2.3 again and then reshaped to $A_2 \in \mathbb{R}^{H \times W \times L \times C}$.

Finally, the output will be

$$H - MHSA = (A_1 + A_2)W^p + x, \tag{3.8}$$

where W^p is a weight matrix used for feature projection. With this approach, H-MHSA, like regular MHSA, may mimic both Local and Global Attention.

3.2 Network Architecture

Even though the suggested model is the ideal one, we had to modify the model due to computational constraints, which required us to reduce the input's size. Consequently, we will only be examining a portion of the Mediterranean Sea. As a result, the size of our input will be $30 \times 30 \times 30$ and the model that will be employed will be the one shown in table 3.1.

Regarding the architecture of this model (Y. Liu et al., 2022), we had to make some changes since the size of the input will not be $224 \times 224 \times 224$ and also because we have in our hands a prediction problem and not a classification one.

In terms of input size, ideally, it would be $31 \times 65 \times 75$, and the area investigated would be the entire Mediterranean Sea, but for computational reasons, the input size had to be lowered to $30 \times 30 \times 30$. As a result, two models will be provided, one for each of the input sizes, but only the one made for the smallest input will be employed in this thesis.

Figure 3.1 and 3.2 present an illustration of the architecture of both HAT-Net architectures with the changes that had to be made for both possible input sizes, where H, W and L, stand for the height, width, and length, respectively, of the input.

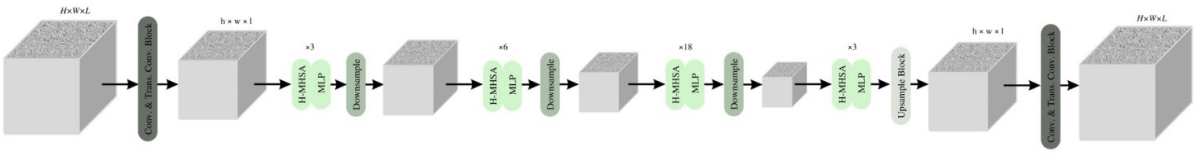


Figure 3.1: Illustration of the proposed HAT-Net for an input size of $61 \times 65 \times 75$.

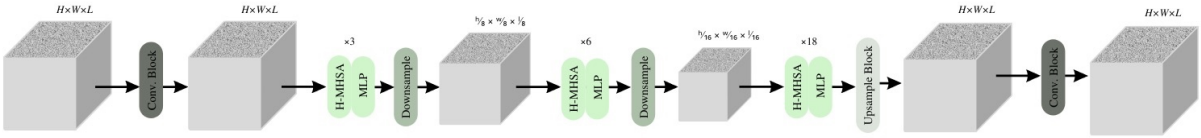


Figure 3.2: Illustration of the proposed HAT-Net for an input size of $30 \times 30 \times 30$.

For the architecture in Figure 3.1, first, a transposed convolution followed by two convolutions are employed to increase the number of channels and change the input size to $30 \times 64 \times 74$ - $h \times w \times l$, such that we have an easier time finding a convolution to downsample it. Next, a block of a H-MHSA and a MLP is applied multiple times during four stages, and between those first three blocks, the feature map is downsampled, with the help of a convolution, so that the input size - $h \times w \times l$ - is reduced. Finally, since the desired output is the prediction of a single variable for each $H \times W \times L$, where each variable is a channel, the input will be upsampled three times with a transposed convolution, and the next two transposed convolutions followed by a convolution are applied so that we have in the end one single channel and the original size of the input.

For the architecture in Figure 3.2, first two convolutions are employed with the main purpose of increasing the number of channels so that we can follow the proposed HAT-Net model by Y. Liu et al., 2022. Next, a block of an H-MHSA and an MLP is applied multiple times during one stage, and after that, the feature map is downsampled, with the help of a $2 \times 2 \times 2$ convolution with stride of 2 and padding of 1, so that the input - $h \times w \times l$ - is scaled to $\frac{1}{8}$ and $\frac{1}{16}$ of its size. Finally, since the desired output is the prediction of a single variable for each $H \times W \times L$, where each variable is a channel, the input will be upsampled two times with a $2 \times 2 \times 2$ transposed convolution of stride 2 and padding of 1 and next two transposed convolutions are applied so that we have in the end one single channel and the original size of the input.

In table 3.2 and 3.1, a summary of the setup details of the proposed HAT-Nets can be seen. Each $K \times K \times K$ convolution and $K \times K \times K$ transposed convolution, where K is the kernel size, has a padding of P, stride of S, and a C number of channels resulting from the convolution. The brackets represent a block that will be repeated a specified number of times and for the MLP, it is used a depthwise separable convolution (Howard et al., 2017) with an expansion ratio of E. In both the MLP and the H-MHSA, Sigmoid Linear Unit (SiLU) is the activation function used since it is more memory-friendly than Gaussian Error Unit (GELU) (Y. Liu et al., 2022), that is the usual activation function applied. For downsampling a convolution is used and for upsampling is used a transposed convolution.

Table 3.1: Network configurations of the proposed HAT-Net for an input size of $30 \times 30 \times 30$.

Stage	**Input Size**	**Operator**	**HAT-Net**
1	$30 \times 30 \times 30$	Conv.	$C = 16, K = 2, P = 0, S = 1$ $C = 64, K = 2, P = 0, S = 1$
2	$28 \times 28 \times 28$	H-MHSA MLP Downsample	$\begin{bmatrix} C = 64 \\ K = 5 \\ E = 8 \end{bmatrix} \times 3$ $C = 128, K = 2, P = 1, S = 2$
3	$15 \times 15 \times 15$	H-MHSA MLP Downsample	$\begin{bmatrix} C = 128 \\ K = 3 \\ E = 8 \end{bmatrix} \times 6$ $C = 320, K = 2, P = 1, S = 2$
4	$8 \times 8 \times 8$	H-MHSA MLP Upsample	$\begin{bmatrix} C = 320 \\ K = 5 \\ E = 4 \end{bmatrix} \times 18$ $C = 128, K = 2, P = 1, S = 2$ $C = 64, K = 2, P = 1, S = 2$
5	$30 \times 30 \times 30$	Trans. Conv.	$C = 16, K = 2, P = 0, S = 1$ $C = 1, K = 2, P = 0, S = 1$
	$1 \times 1 \times 1$	-	-
#Param			3.0M

Table 3.2: Network configurations of the proposed HAT-Net for an input size of $31 \times 65 \times 75$.

Stage	Input Size	Operator	HAT-Net
1	$31 \times 65 \times 75$	Trans. Conv. Conv. Conv.	$\begin{bmatrix} C=3 & P=2 \\ K=4 & S=2 \end{bmatrix} \times 2$ $C=16, K=4, P=2, S=2$ $C=64, K=2, P=0, S=2$
2	$30 \times 64 \times 74$	H-MHSA MLP Downsample	$\left[C=64, K=5, E=8\right] \times 3$ $C=128, K=5, P=0, S=1$
3	$26 \times 60 \times 70$	H-MHSA MLP Downsample	$\left[C=128, K=3, E=8\right] \times 6$ $C=320, K=3, P=0, S=1$
4	$24 \times 58 \times 68$	H-MHSA MLP Downsample	$\left[C=320, K=5, E=4\right] \times 18$ $C=512, K=2, P=0, S=2$
5	$12 \times 29 \times 34$	H-MHSA MLP	$\left[C=512, K=3, E=4\right] \times 3$
6	$12 \times 29 \times 34$	Upsample	$C=320, K=2, P=0, S=2$ $C=128, K=3, P=0, S=1$ $C=64, K=5, P=0, S=1$
7	$30 \times 64 \times 74$	Trans. Conv. Trans. Conv. Conv.	$C=16, K=2, P=0, S=2$ $C=3, K=4, P=2, S=2$ $\begin{bmatrix} C=3 & P=2 \\ K=4 & S=2 \end{bmatrix} \times 2$
	$1 \times 1 \times 1$	-	-
#Param			**FILL**

Chapter 4

Experimental Settings

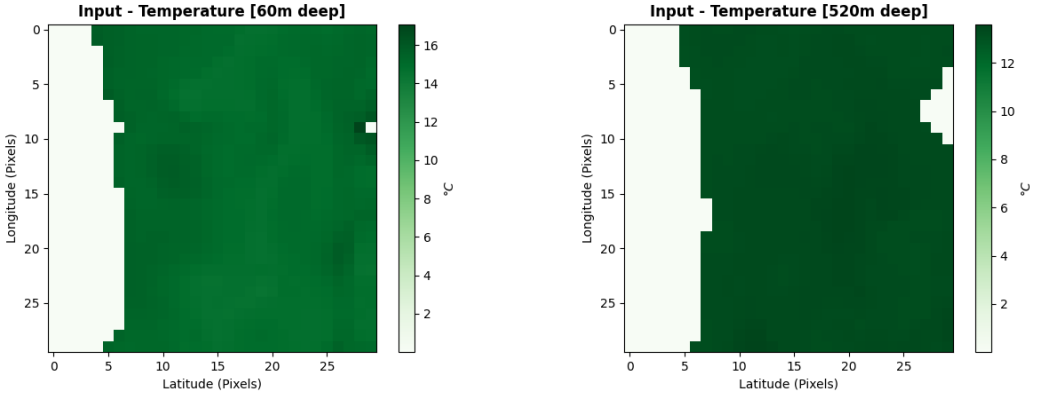
In the previous chapters, we have explored the architecture and components of Transformer models, which have revolutionized the field of NLP and ML. However, building an effective Transformer model goes beyond just understanding its architecture. Hyperparameter tuning plays a crucial role in achieving optimal performance for specific tasks. In this chapter, we begin with a description of the data source that will be used and how the future dataset was created, followed by a discussion of the hyperparameters that were used to train the model.

4.1 Dataset

MedBFM, which is run by OGS (Italy) (National Institute of Oceanography and Applied Geophysics), provides the short-term forecast and the analysis of the Mediterranean Sea biogeochemistry at 1/24 degree of horizontal resolution (ca. 4 km) since January 1999, for the Copernicus Marine Environment Monitoring Services (CMEMS6). In this framework, a ten-day forecast of biogeochemical variables (chlorophyll, nutrients (phosphate and nitrate) and dissolved oxygen concentrations, net primary production, phytoplankton biomass, ocean pH, and ocean pCO2) is released daily, and an analysis of the previous seven days is performed once a week with the assimilation of surface chlorophyll (i.e., CMEMS-OCTAC satellite data) and in situ profiles of chlorophyll and nitrate (i.e., BGC-Argo float data) (Bolzon et al., 2019). A reanalysis is also performed monthly and it provides means of the 3D fields of the biogeochemical variables mentioned previously (Teruzzi, Bolzon, et al., 2019). MedBFM was built over the coupling between the OGS transport model (OGSTM), the BFM (Lazzari et al., 2010; Lazzari et al., 2012; Lazzari et al., 2016; Cossarini et al., 2015 and Vichi et al., 2020), and the 3DVarBio variational scheme that assimilates surface chlorophyll concentrations (Teruzzi et al., 2014; Teruzzi et al., 2018 and Teruzzi, Cerboa, et al., 2019)(Salon et al., 2021 and Teruzzi, Cerbo, et al., 2019).

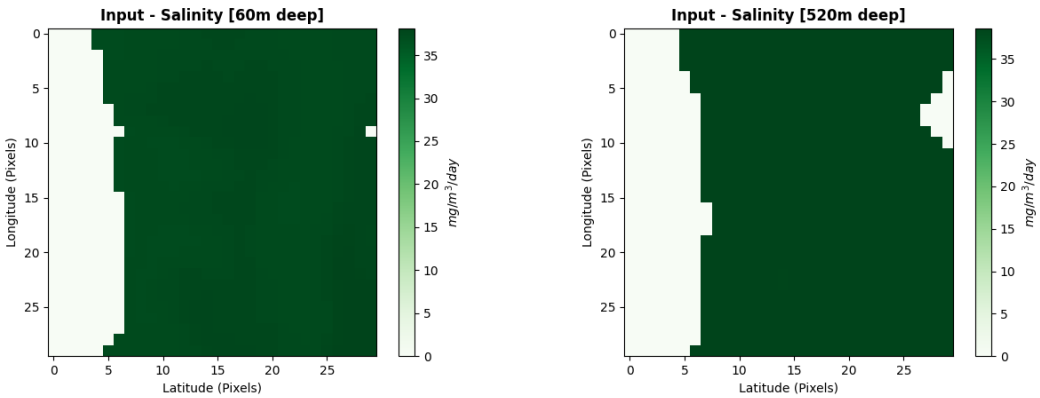
The dataset used in this work to train the DL architecture described in the previous chapter is based on the deterministic model MedBFM, as it is made up of tensors obtained by a discretization of the simulation from marine ecosystem variables. Temperature (Figure 4.1), salinity (Figure 4.2), and oxygen (Figure 4.3) are employed as input variables, while CHLA, NPP, N1P, N3N, and R6C are the output variables that we want to simulate. We only examined the 2015 year, which is discretized every week, therefore 52 weekly data are available. Our model's tensors are four-dimensional, with the channels serving as the marine ecosystem

variables (input and output tensors), and the height, width, and depth of the Mediterranean Sea region to be studied. This region is the western Mediterranean section, namely the one with latitudes extending from 36 to 44 and longitudes ranging from 2 to 9, with a vertical dimension (depth) ranging from 0 to 600 meters. It is the area between southern France and northern Africa, bordered on the east by Corsica and Sardinia and on the west by the Balearic Islands (Table 4.1). Although this would be the desired input, due to computational reasons, the size of the input had to be reduced. With this resize, the region that will be studied will be the one with latitudes extending from 36 to 39 and longitudes ranging from 2 to 5, with a vertical dimension (depth) ranging from 0 to 590 meters. This area is the one between northern Africa and the Balearic Islands - Mallorca and Menorca (Table 4.2).The spatial resolution is 12 kilometers along both the latitude and longitude axes and 20 meters along the vertical axis. These four marine ecosystem variables are related to each place in the 3D field. Because of their nature, each float gives a 1D profile in which the latitude and longitude are fixed and only the depth can fluctuate, but satellites can only monitor the water's surface and so provide 2D data (with holes due to cloud covers) (Pietropolli, Cossarini, and Manzoni, 2022).



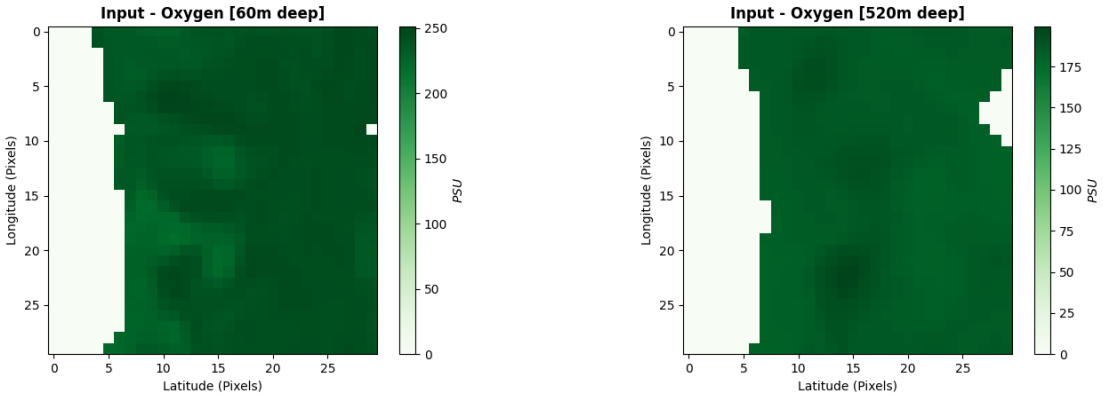
(a) 60m Deep (b) 520m Deep

Figure 4.1: Temperature



(a) 60m Deep (b) 520m Deep

Figure 4.2: Salinity



(a) 60m Deep

(b) 520m Deep

Figure 4.3: Oxygen

Table 4.1: Original input structure settings.

Parameter	Value
Time Interval	Weekly
Latitude Interval	36°- 44°
Longitude Interval	2°- 9°
Depth Interval	0 - 600 m
Time Resolution	Weekly
Latitude Resolution	12 km
Longitude Resolution	12 km
Depth Resolution	20 m

Table 4.2: Resized input structure settings.

Parameter	Value
Time Interval	Weekly
Latitude Interval	36°- 39°
Longitude Interval	2°- 5°
Depth Interval	0 - 590 m
Time Resolution	Weekly
Latitude Resolution	12 km
Longitude Resolution	12 km
Depth Resolution	20 m

4.2 Hyperparameters

We employed the following hyperparameters to train our HAT-Net, reported in Table 4.3. The network is trained for 2000 epochs with the AdamW optimizer, which automatically sets the LR for each weight in the network; the starting value for the LR is set to 0.000001, and the dropout rate is set to 0.1. The batch size was 5 since it was the largest we could use with the computational resources at our disposal.

Following suitable preliminary research, these hyperparameters were determined. For the number of epochs, we checked how many epochs the model converged after or, for some variables, how many were enough to reach a low error; the optimizer chosen was based on optimizers that are commonly used with Transformer techniques.

Table 4.3: The deep learning architecture's hyperparameters.

Parameter	Value
Input Size	$30 \times 30 \times 30$
Learning Rate	0.000001
Number of Epochs	2000
Batch Size	5
Optimizer	AdamW

Chapter 5

Results

This chapter presents some illustrative examples of the accuracy of the approach developed in Chapter 3. Numerical results are presented and discussed, having in mind the target values as well as an evaluation of the network's accuracy.

5.1 Results and Performance Evaluation

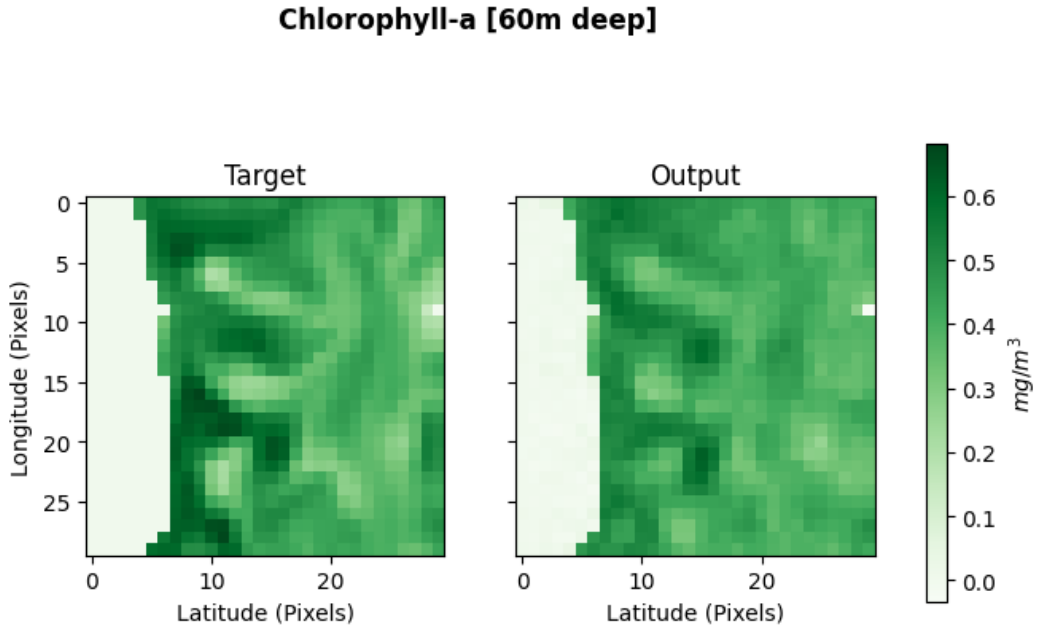
Figures 5.1 to 5.5 show a colormap of the target tensor on the left and the output tensor on the right, for the values we aim to simulate - CHLA, NPP, N1P, N3N, and R6C. Because we are simulating these variables over a depth range of 0 to 590 meters, plots for two profundity levels (60 and 520 meters) have been included so that it is possible to examine and evaluate the performance of our model for different depths.

To comprehensively examine the divergence between values provided by the deterministic model and those generated by our model for each variable, Figures 5.6 to 5.10 present colormaps illustrating the Absolute Relative Error (ARE) at both surface and deeper levels for N1P and N3N. In the case of CHLA, NPP, and R6C, the colormaps are exclusively displayed at the surface level. It is important to note that, in the calculation of the ARE, we assume zero values for land, thereby setting the ARE to zero in those regions. Additionally, Figure 5.11 features a line chart illustrating the Mean Absolute Error (MAE) for all variables across each epoch. It is noteworthy that, for the latter, the variables were normalized to ensure a precise comparison, in this way they all, share the same range of values.

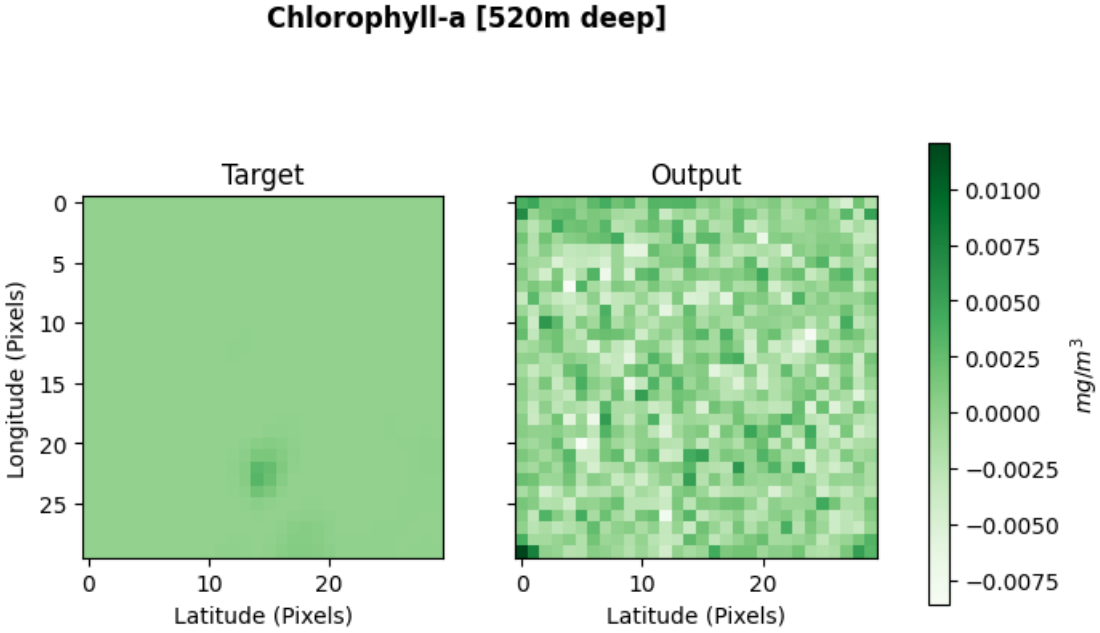
Figures 5.12 to 5.16 illustrate the loss of the train and test datasets for the previously listed variables so we can understand if there has been overfitting or underfitting. To evaluate the quality of the proposed model, we compute the MSE between the values given by the deterministic model and the ones generated by our model. We chose this metric, since squaring the errors lends greater weight to larger mistakes and also because it is in the same units as the variable being evaluated, which might aid in comprehension. [1]

For each variable, the values for the loss after 2000 epochs are displayed in Table 5.1.

[1] These plots are also available in an interactive format, which may be viewed using the links in the Figure descriptions.

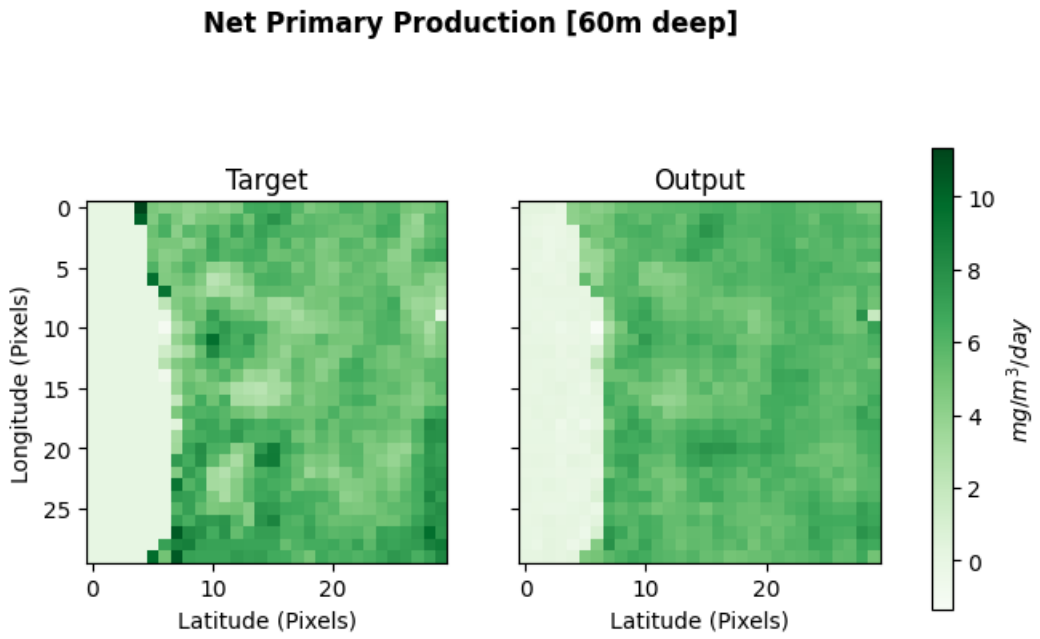


(a) 60m Deep

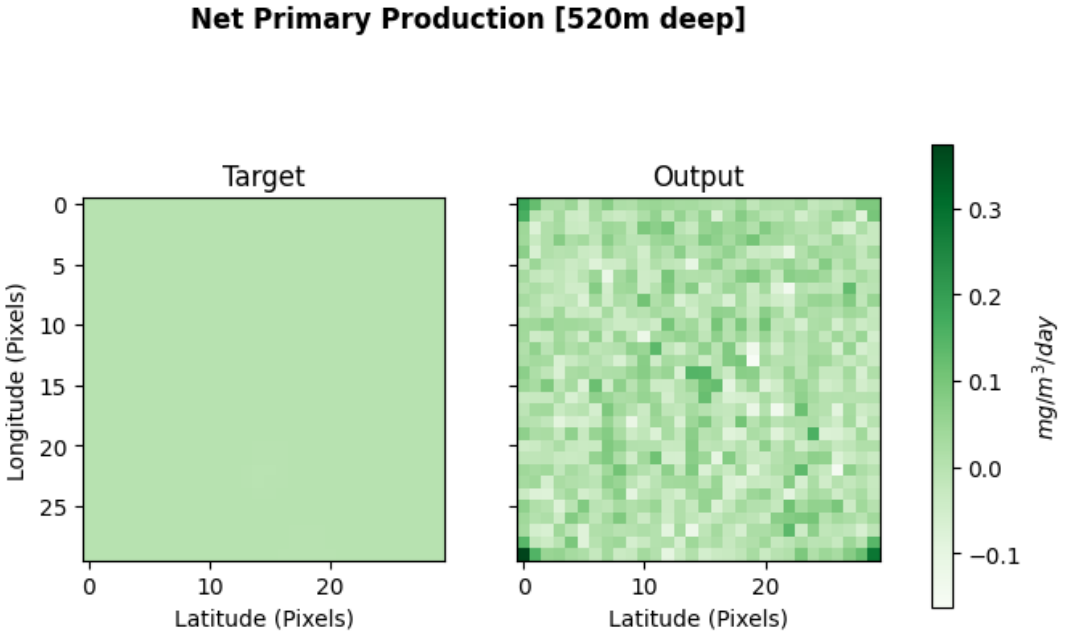


(b) 520m Deep

Figure 5.1: Chlorophyll-a - Target (Left) VS. Output (Right).

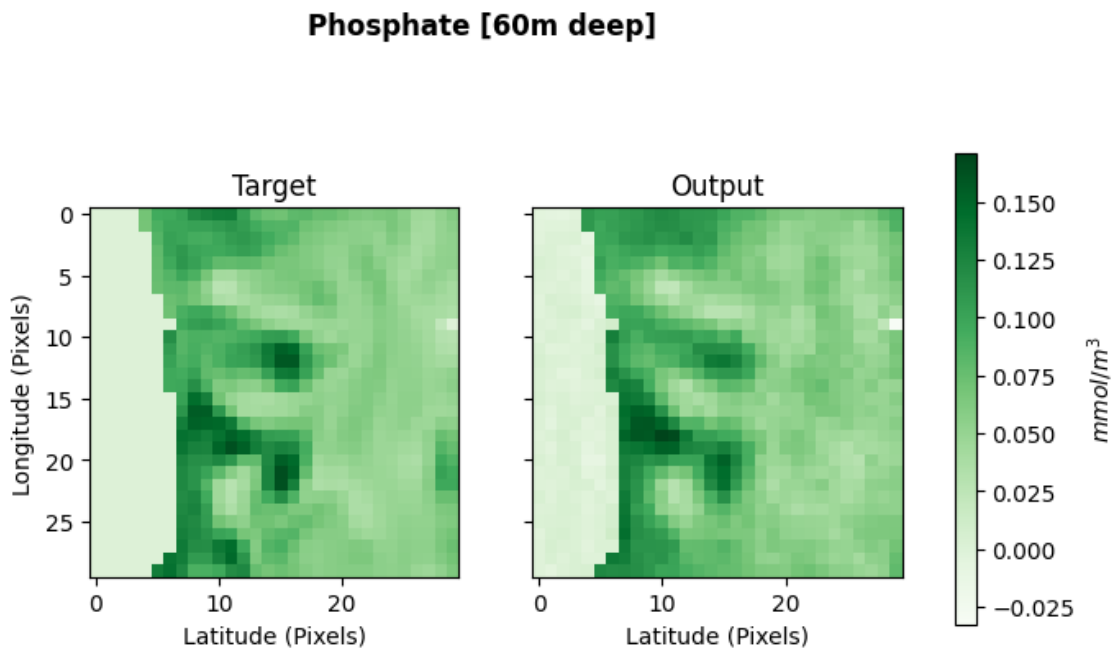


(a) 60m Deep

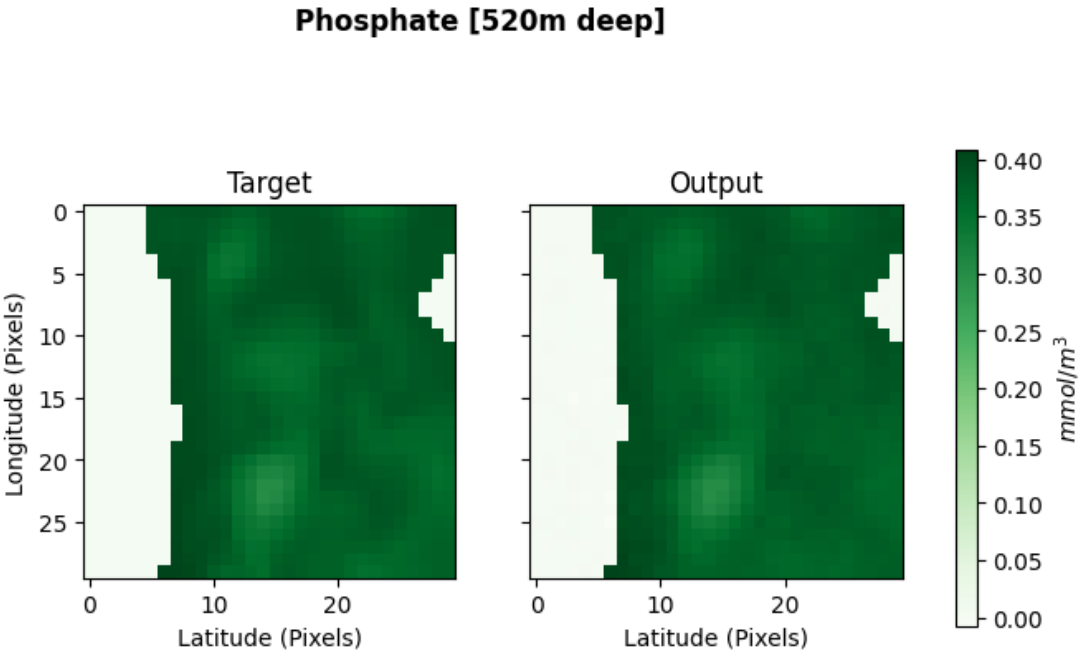


(b) 520m Deep

Figure 5.2: Net Primary Production - Target (Left) VS. Output (Right).

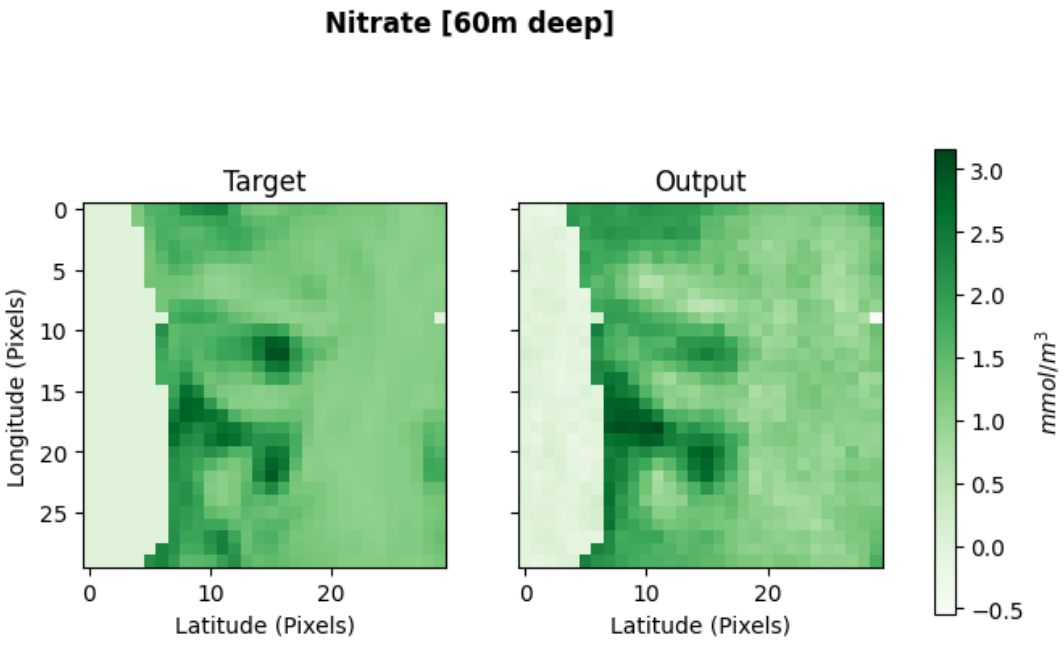


(a) 60m Deep

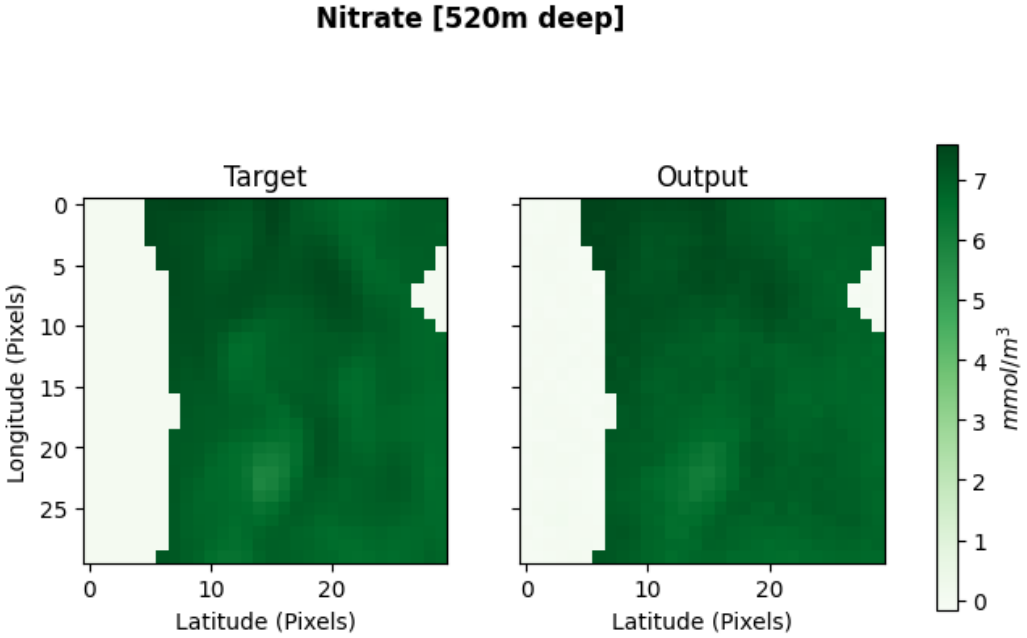


(b) 520m Deep

Figure 5.3: Phosphate - Target (Left) VS. Output (Right).

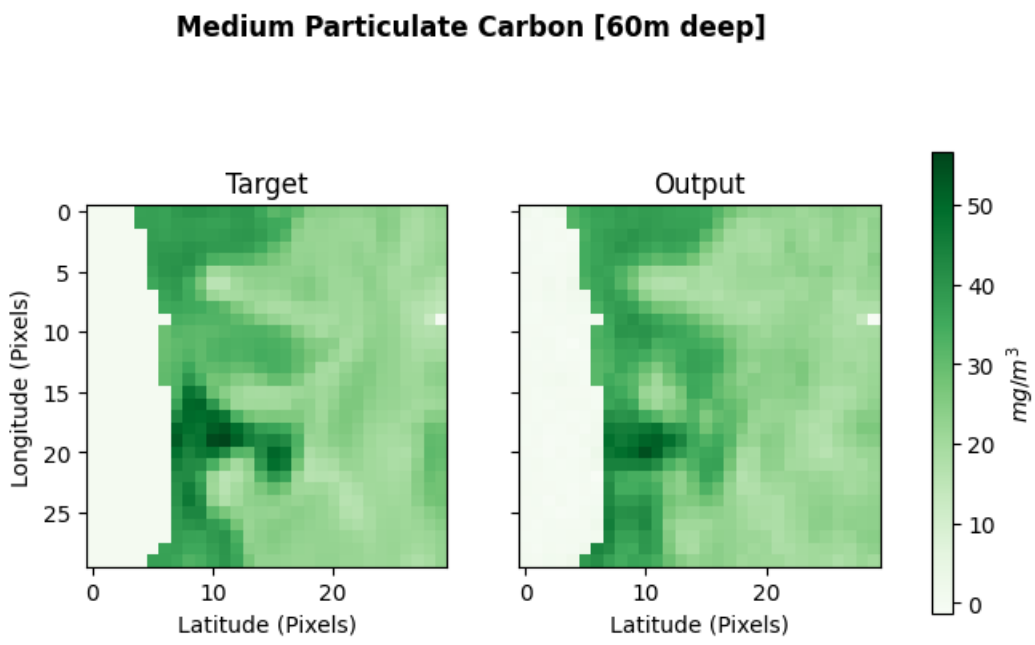


(a) 60m Deep

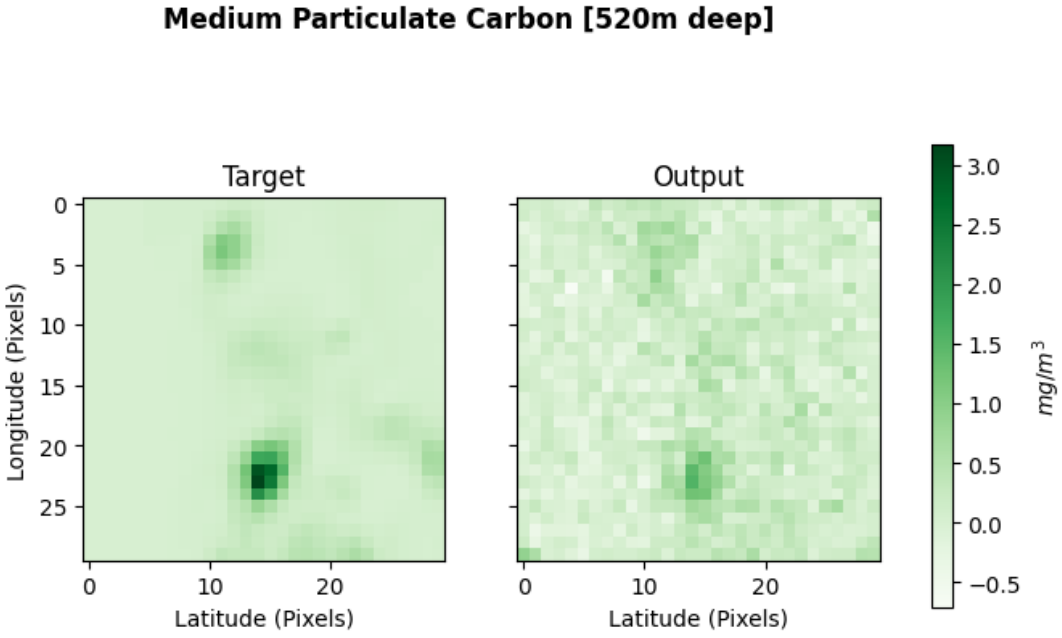


(b) 520m Deep

Figure 5.4: Nitrate - Target (Left) VS. Output (Right).

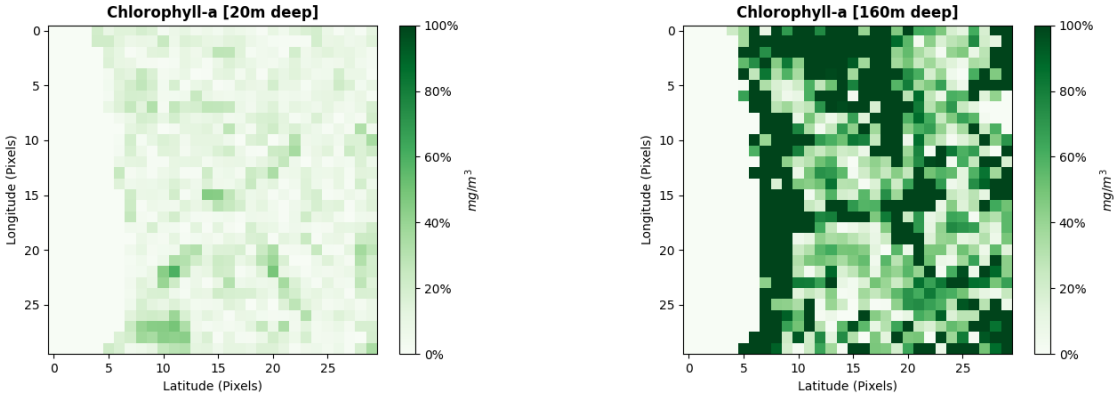


(a) 60m Deep



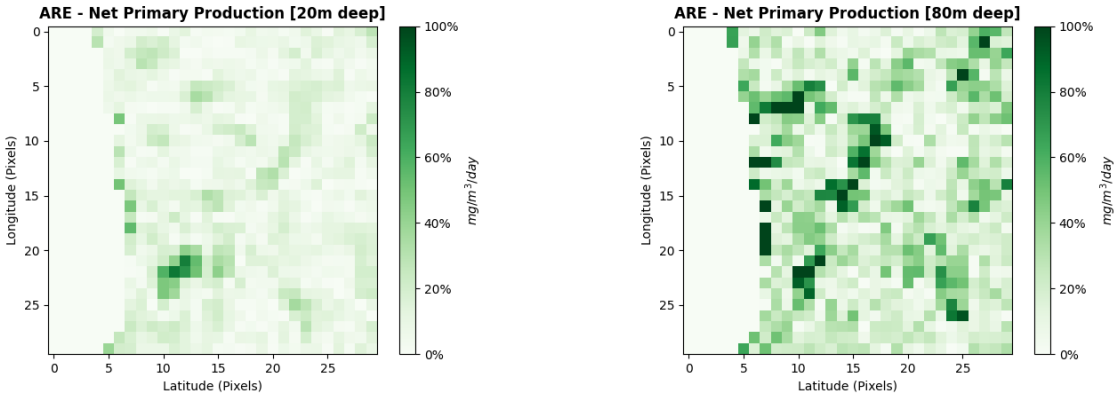
(b) 520m Deep

Figure 5.5: Medium Particulate Carbon - Target (Left) VS. Output (Right).



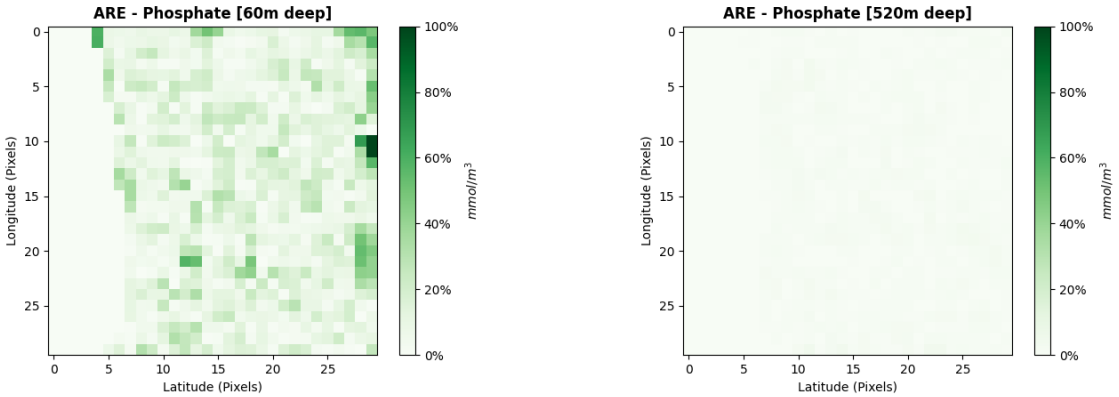
(a) 20m Deep (b) 160m Deep

Figure 5.6: Chlorophyll-a - Absolute Relative Error.



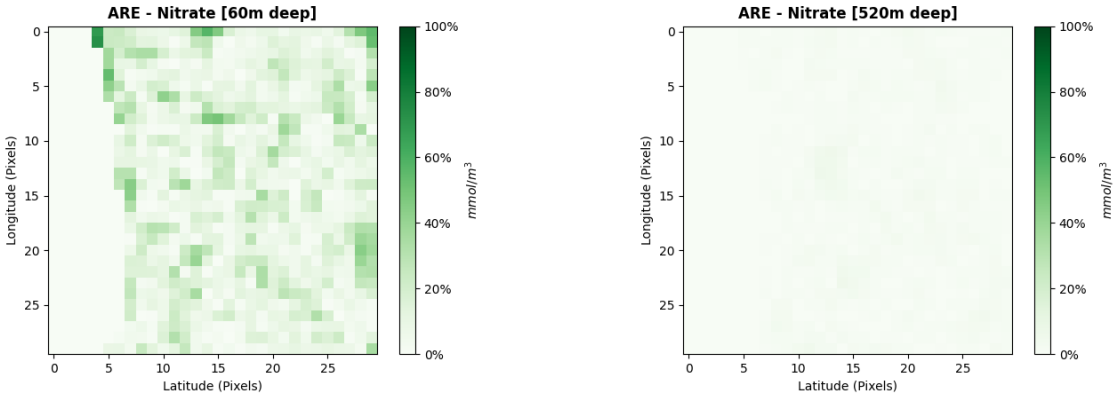
(a) 20m Deep (b) 80m Deep

Figure 5.7: Net Primary Production - Absolute Relative Error.



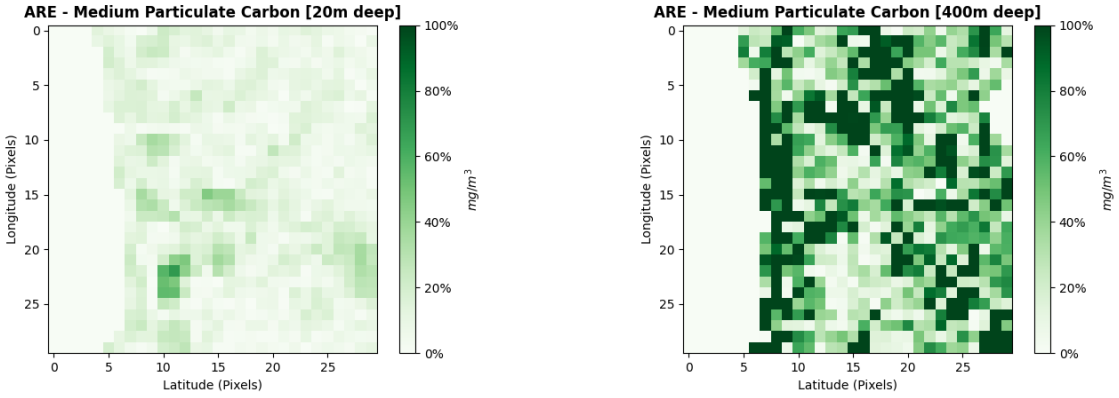
(a) 60m Deep (b) 520m Deep

Figure 5.8: Phosphate - Absolute Relative Error.



(a) 60m Deep (b) 520m Deep

Figure 5.9: Nitrate - Absolute Relative Error.



(a) 20m Deep (b) 200m Deep

Figure 5.10: Medium Particulate Carbon - Absolute Relative Error.

Table 5.1: Loss of the Test and Train Dataset for each variable.

Dataset	**Variable**				
	CHLA	PPN	N1P	N3N	R6C
Train	0.0120	0.0358	0.0022	0.0026	0.0186
Test	0.0407	0.0468	0.0026	0.0035	0.0451

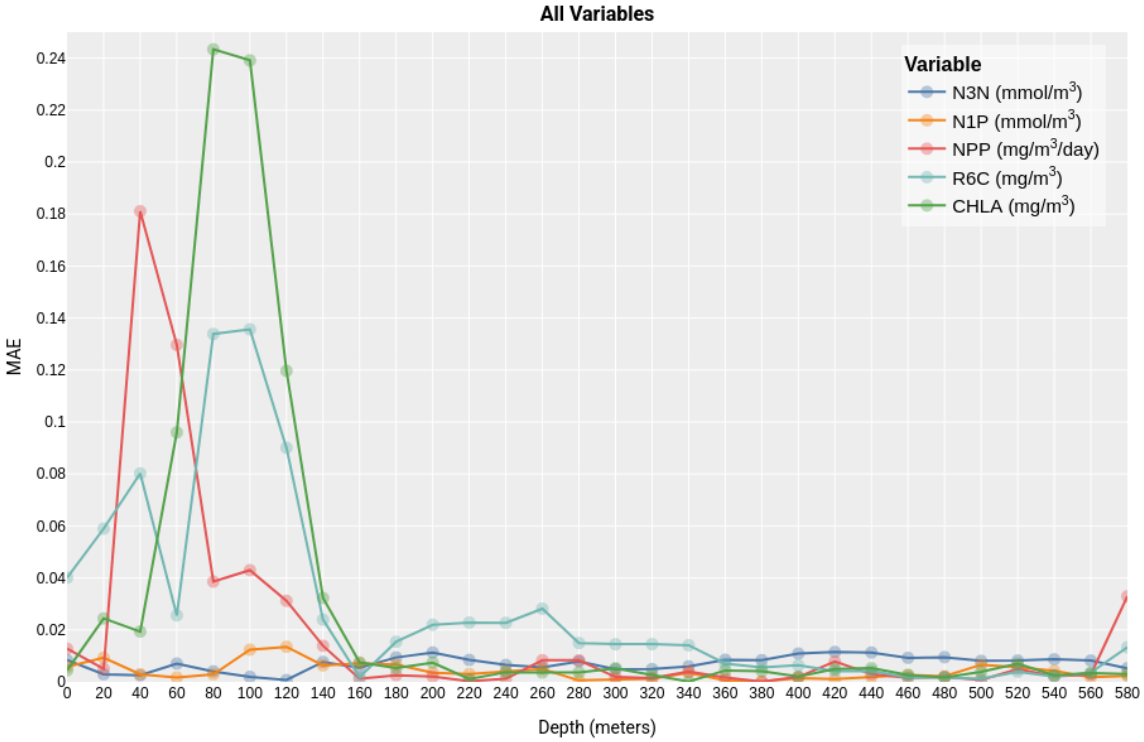


Figure 5.11: Mean Absolute Error of all the variables.
(See in more detail)

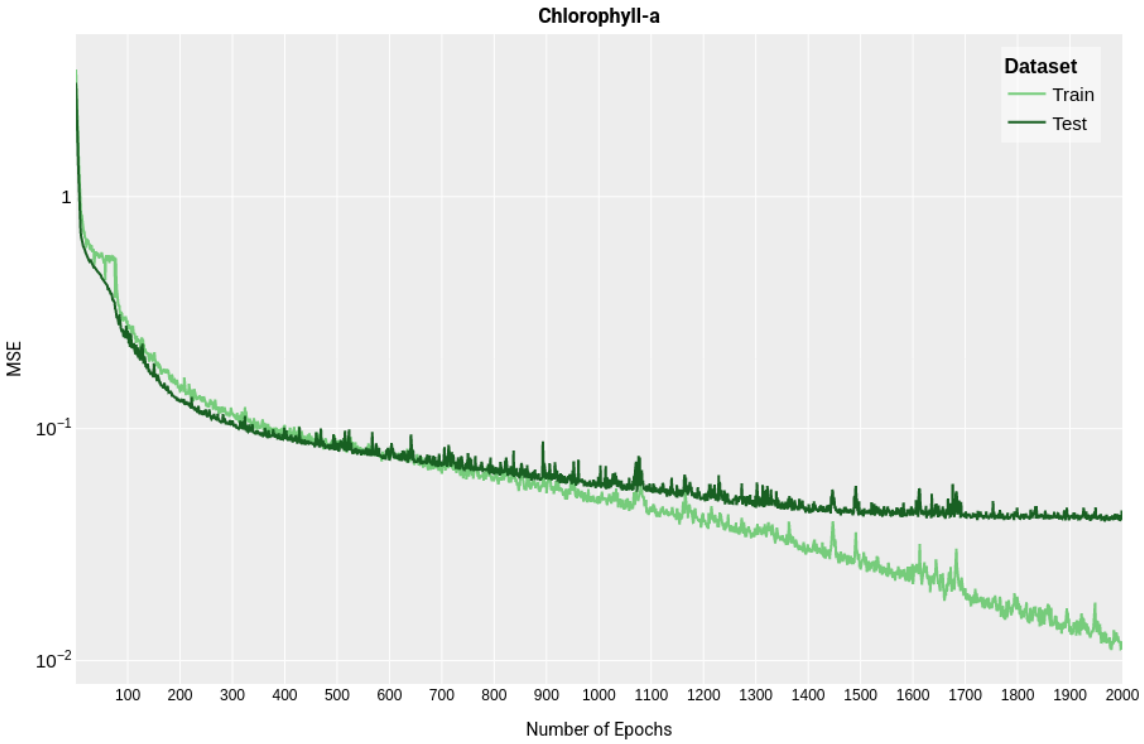


Figure 5.12: Chlorophyll-a - Loss of the Train and Test Datasets.
(See in more detail)

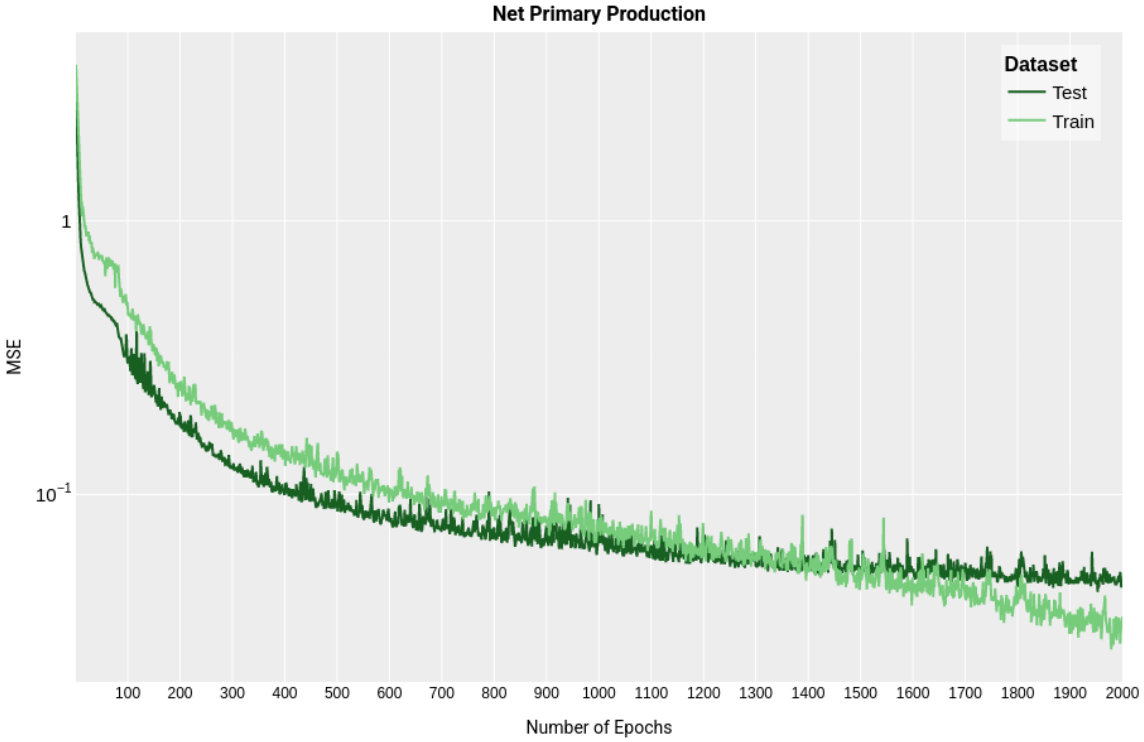


Figure 5.13: Net Primary Production - Loss of the Train and Test Datasets.
(See in more detail)

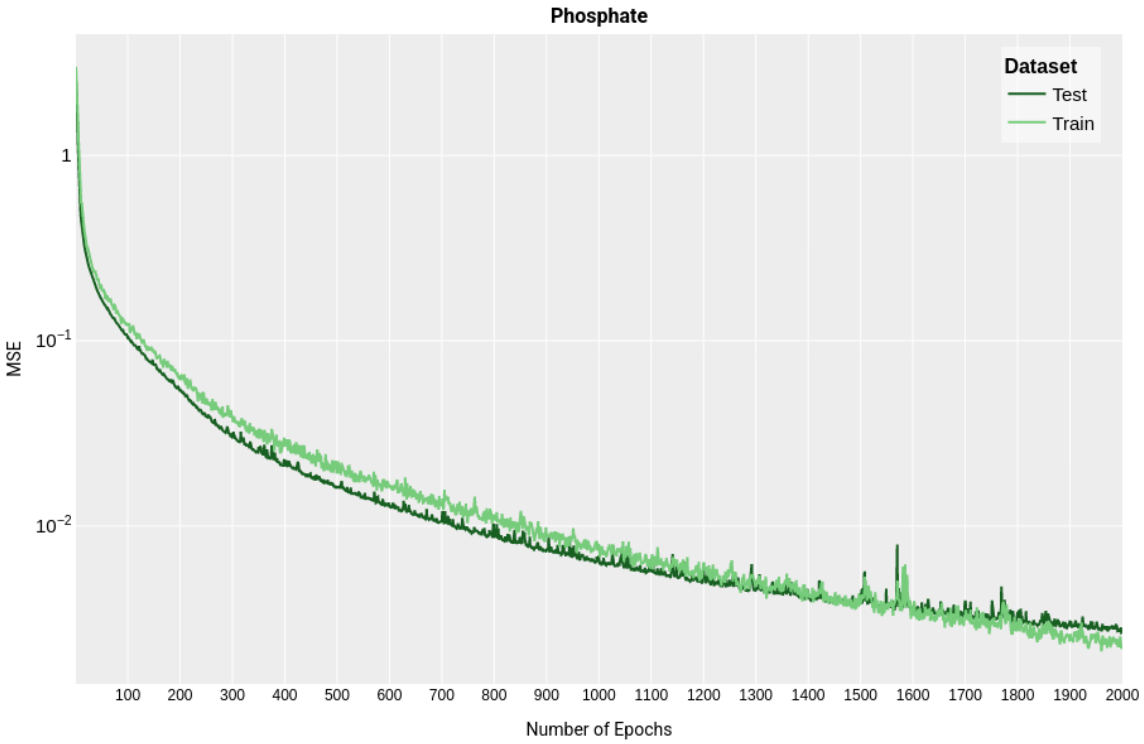


Figure 5.14: Phosphate - Loss of the Train and Test Datasets.
(See in more detail)

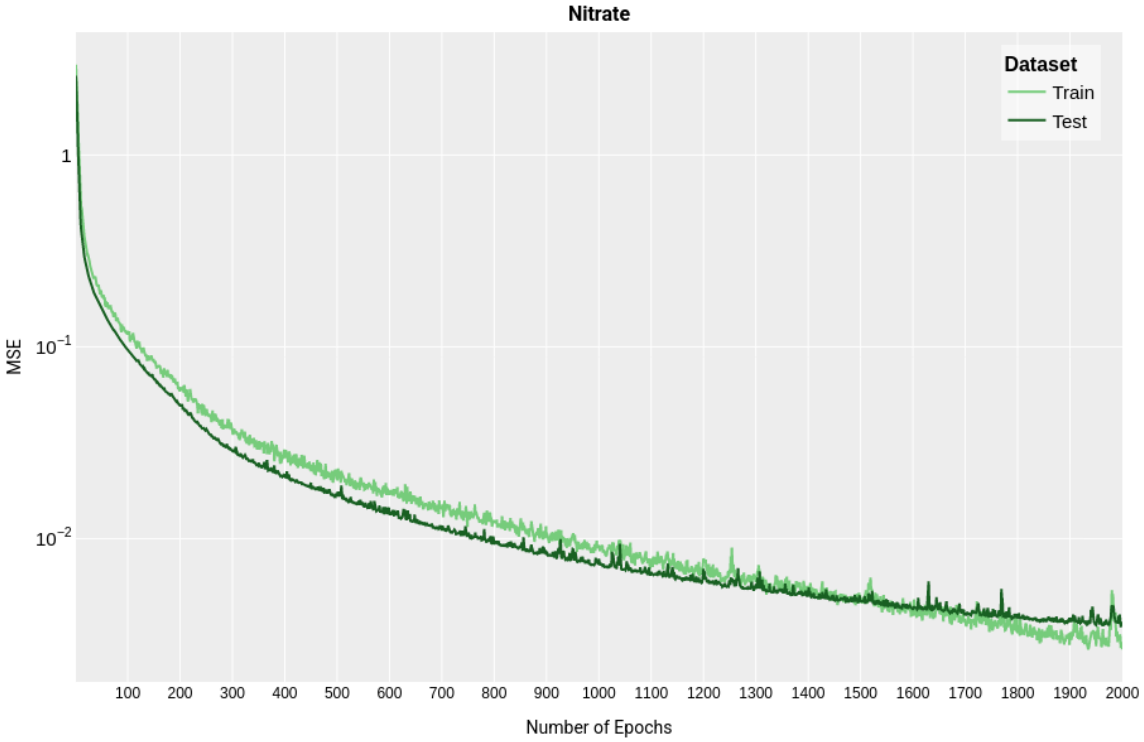


Figure 5.15: Nitrate - Loss of the Train and Test Datasets.
(See in more detail)

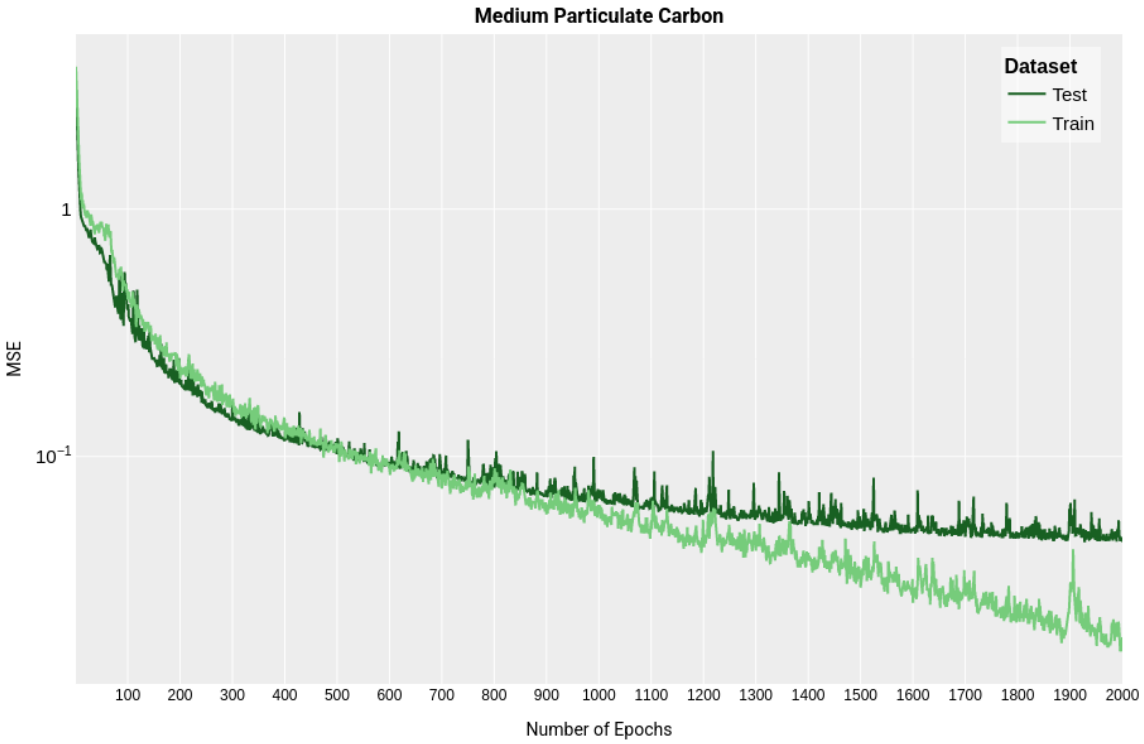


Figure 5.16: Medium Particulate Carbon - Loss of the Train and Test Datasets.
(See in more detail)

5.2 Results Discussion and Comparison

Before delving into the results discussion, it is essential to emphasize that, although our analysis covers the entire depth, our primary focus is on surface-level values. This emphasis is due to the surface variables being more influenced by external forces, such as wind and surface temperature, which contribute to circulation at the surface.

Figures 5.1, 5.2 and 5.5, a good simulation for CHLA, NPP, and R6C at the surface level becomes evident. As we extend our analysis to greater depths, there may appear to be a reduction in our model's ability to replicate these variables. This is also depicted in Figures 5.6, 5.7, and 5.10, where the ARE is computed. It is essential to recognize that the diminished precision observed can be partially attributed to the fact that, at greater depths, the values tend to approach zero. In fact, the values for deeper sea levels exhibit a smaller order of magnitude as compared to the values for levels closer to the surface. Considering this, we conclude that the diminished accuracy at greater depths is not substantial. Nevertheless, it is vital to reiterate that our primary focus lies in the evaluation of these variables specifically at the surface level.

Examining 5.3 and 5.4 we can see that, for variables N1P and N3N, the values remain nearly identical both at the surface level and full depth, highlighting the robust capability of our model to simulate these marine fields across varying degrees of profundity. This observation becomes more prominent when computing the ARE, as displayed in 5.8 and 5.9. This is likely due to the increased homogeneity of variable values at greater sea depths, less affected by external factors like wind and rain.

Significantly, there is no observed escalation in errors along the borders. This is particularly noteworthy, as the convolutional structure within our model predicts values by considering neighboring pixels, an aspect that could be affected at the interface of land and water.

Furthermore, the results from Figure 5.11 reaffirm that CHLA, NPP, and R6C exhibit lower performance in our model. Specifically, focusing on the surface level, these variables show a higher MAE compared to that observed at greater depths. The nearly uniform MAE for N1P and N3N suggests that the tendency of values toward zero at greater depths influences our model's performance, even at the surface level. Additionally, noteworthy is the observation that for CHLA, NPP, and R6C, the MAE remains low at greater depths, while the ARE is higher. This further confirms that the values of these variables at greater depths tend toward zero, causing the ARE to approach infinity.

Finally, Figures 5.12 to 5.16 alongside Table 5.1, show us that the MSE for all variables is extremely low, between 0.0026 and 0.047, with the variable with the lowest loss, N1P, having an MSE of 0.0026. This is no surprise as N1P is the variable for which our model performed best, according to the previously discussed results. Although, as previously stated, the simulation of CHLA, NPP, and R6C becomes less realistic below a certain depth, the MSE remains quite low.

Additionally, the model does not converge in 2000 epochs for some variables, such as N1P and N3N. While we could have trained our model for more epochs and possibly reached convergence, we concluded that the MSE value is already satisfactory (0.0026 and 0.0035, respectively), and running the model for some more epochs would lead to a minimal loss reduction at the expense of more hours of computational time. For some variables, such as CHLA, NPP, and R6C, we observe that beyond a certain epoch, the loss of the train and test datasets begins to drift apart, which indicates that overfitting may be occurring. For that reason, it could be more efficient to add an early-stop routine to automatically stop the training of the model when these two losses show divergent behavior. This could mitigate the effects of overfitting and reduce computational time.

Regarding computational time, our model, utilizing a virtual machine with a single GPU NVIDIA Quadro GV100 and a Double Data Rate Fourth Generation (DDR4) Random-Access

Memory (RAM) of 512GB (8×64GB), took an average of 25 hours to run 2000 epochs, for each variable.

Chapter 6

Conclusion and Future Works

In this thesis, we undertook a comprehensive exploration of the intricate biogeochemical dynamics of the Mediterranean Sea using advanced Deep Learning techniques, particularly Convolutional Vision Transformer with Hierarchical Attention, using as input feature the robust MedBFM dataset managed by the National Institute of Oceanography and Applied Geophysics (OGS) in Italy. Our investigation revealed the model's commendable capacity to simulate key biogeochemical variables, particularly Phosphate and Nitrate, across various depths, affirming its proficiency in capturing marine fields' nuances.

Despite encountering challenges in simulating variables that tend to zero as we go deeper into the sea, the focus on surface-level values aligned with our primary objective. Nonetheless, the model's enhanced performance at greater depths for variables that do not tend to zero showcased its adaptability in navigating the complexities of the marine environment.

The consistently low Mean Squared Error values for all variables, ranging from 0.0026 to 0.047, underscored the model's precision in simulating various marine fields. Phosphate, with a Mean Squared Error of 0.0026, emerged as the variable with the lowest loss, highlighting the model's high accuracy in simulating this particular marine field.

Regarding the trade-off between computational efficiency and model convergence, our observations suggest that, for certain variables like Chlorophyll-a, Net Primary Production, and Medium Particulate Carbon, a conservative approach involves training the model for fewer epochs to mitigate overfitting.

Considering the results and insights gained, avenues for future research become apparent. One promising research direction involves extending the scope of simulation to encompass the entire Mediterranean Sea. This could be achieved through a series of simulations, whereby our model (Table 3.1) is applied to smaller regions that, when combined, form a comprehensive representation of the entire sea. This approach would contribute to a more holistic understanding of the interconnected dynamics within the Mediterranean Sea, capturing the nuances of its diverse ecosystems.

Furthermore, future research could explore utilizing our other suggested model (Table 3.2), if computational resources allow, for simulating the entire Mediterranean Sea in a single simulation. This could present a more integrated approach that might unveil novel insights.

As computational capabilities progress, the future goals are to enhance our knowledge of marine ecosystems and support ongoing efforts for sustainable oceans. In either case, the task of

simulating the entire Mediterranean Sea not only aligns with the evolving landscape of oceanographic research, but also holds significant implications for understanding broader ecological patterns, climate influences, and human impact across the entire basin. As computational capabilities continue to advance, the proposed future works aim to push the boundaries of our understanding of marine ecosystems and contribute to the ongoing efforts in sustainable ocean management.

In summary, the new model created in this thesis has made big progress in simulating marine variables in the Mediterranean Sea. This sets the stage for more research and improvement, keeping our understanding of these ecosystems at the forefront of science.

Bibliography

Bahdanau, D., Cho, K., & Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. *3rd International Conference on Learning Representations]*. http://arxiv.org/abs/1409.0473

Bethoux, J., Gentili, B., Morin, P., Nicolas, E., Pierre, C., & Ruiz-Pino, D. (1999). The Mediterranean Sea: a miniature ocean for climatic and environmental studies and a key for the climatic functioning of the North Atlantic [International Conference on Progress in Oceanography of the Mediterranean Sea, ROME, ITALY, NOV 17-19, 1997]. *Progress in Oceanography*, *44*(1-3), 131–146. https://doi.org/10.1016/S0079-6611(99)00023-3

Bittig, H. C., Steinhoff, T., Claustre, H., Fiedler, B., Williams, N. L., Sauzède, R., Körtzinger, A., & Gattuso, J.-P. (2018). An alternative to static climatologies: Robust estimation of open ocean co2 variables and nutrient concentrations from t, s, and o2 data using bayesian neural networks. *Frontiers in Marine Science*, *5*, 328. https://doi.org/10.3389/fmars.2018.00328

Bolzon, G., Cossarini, G., Lazzari, P., Salon, S., Teruzzi, A., Feudale, L., Di Biagio, V., & & Solidoro, C. (2019). Mediterranean sea biogeochemical analysis and forecast (cmems med-biogeochemistry 2018-2021). *Copernicus Monitoring Environment Marine Service (CMEMS)*. https://doi.org/10.25423/CMCC/MEDSEA_ANALYSIS_FORECAST_BIO_006_014_MEDBFM3

Chahar, A., Chowdhury, A., Thulasidoss, B. K., Reddy, P. V., Patel, H., & Patil, N. (2022). Water quality analysis using deep learning. *1*, 423–426. https://doi.org/10.1109/ICACCS54159.2022.9785189

Chen, L., Zhang, H., Xiao, J., Nie, L., Shao, J., Liu, W., & Chua, T.-S. (2017). Sca-cnn: Spatial and channel-wise attention in convolutional networks for image captioning. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 6298–6306. https://doi.org/10.1109/CVPR.2017.667

Claustre, H., Johnson, K. S., & Takeshita, Y. (2020). Observing the global ocean with biogeochemical-argo. *Annual review of marine science*, *12*, 23–48. https://doi.org/10.1146/annurev-marine-010419-010956

Cossarini, G., Lazzari, P., & Solidoro, C. (2015). Spatiotemporal variability of alkalinity in the mediterranean sea. *Biogeosciences*, *12*, 1647–1658. https://doi.org/10.5194/bg-12-1647-2015

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., & Houlsby, N. (2021). An image is worth 16x16 words: Transformers for image recognition at scale. *International Conference on Learning Representations*. https://arxiv.org/pdf/2010.11929.pdf

Euzen, A., Gaill, F., Lacroix, D., & Cury, P. (2017). The ocean revealed.

Fan, H., Xiong, B., Mangalam, K., Li, Y., Yan, Z., Malik, J., & Feichtenhofer, C. (2021). Multiscale vision transformers. *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, 6804–6815. https://doi.org/10.1109/ICCV48922.2021.00675

Han, M., Su, Z., & Na, X. (2023). Predict water quality using an improved deep learning method based on spatiotemporal feature correlated: A case study of the tanghe reservoir in china. *37*, 2563–2575. https://doi.org/10.1007/s00477-023-02405-4

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 770–778. https://doi.org/10.1109/CVPR.2016.90

Hinton, G. E., Vinyals, O., & Dean, J. (2015). Distilling the knowledge in a neural network. *CoRR*, *abs/1503.02531*. http://arxiv.org/abs/1503.02531

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., & Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. https://arxiv.org/pdf/1704.04861.pdf

Hu, J., Shen, L., Albanie, S., Sun, G., & Wu, E. (2020). Squeeze-and-excitation networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *42*(8), 2011–2023. https://doi.org/10.1109/TPAMI.2019.2913372

Jaderberg, M., Simonyan, K., Zisserman, A., & Kavukcuoglu, K. (2015). Spatial transformer networks. *Advances in Neural Information Processing Systems*, *28*. https://arxiv.org/pdf/1506.02025.pdf

Khan, Y., & See, C. S. (2016). Predicting and analyzing water quality using machine learning: A comprehensive model, 1–6. https://doi.org/10.1109/LISAT.2016.7494106

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Neural Information Processing Systems*, *25*. https://doi.org/10.1145/3065386

Lazzari, P., Solidoro, C., Ibello, V., Salon, S., Teruzzi, A., Béranger, K., Colella, S., & Crise, A. (2012). Seasonal and inter-annual variability of plankton chlorophyll and primary production in the mediterranean sea: A modelling approach. *Biogeosciences*, *9*, 217–233. https://bg.copernicus.org/articles/9/217/2012/

Lazzari, P., Solidoro, C., Ibello, V., Salon, S., Teruzzi, A., Béranger, K., Colella, S., & Crise, A. (2016). Spatial variability of phosphate and nitrate in the mediterranean sea: A modeling approach. *Deep Sea Research Part I: Oceanographic Research Papers*, *108*, 39–52. https://www.sciencedirect.com/science/article/pii/S0967063715301473

Lazzari, P., Teruzzi, A., Salon, S., Campagna, S., Calonaci, C., Colella, S., Tonani, M., & Crise, A. (2010). Pre-operational short-term forecasts for mediterranean sea biogeochemistry. *Ocean Science*, *6*, 25–39. https://os.copernicus.org/articles/6/25/2010/

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324. https://ieeexplore.ieee.org/document/726791

Li, X., Wang, W., Hu, X., & Yang, J. (2019). Selective kernel networks. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 510–519. https://doi.org/10.1109/CVPR.2019.00060

Liu, Y., Wu, Y.-H., Sun, G., Zhang, L., Chhatkuli, A., & Gool, L. V. (2022). Vision transformers with hierarchical attention. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. https://arxiv.org/pdf/2106.03180.pdf

Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., & Guo, B. (2021). Swin transformer: Hierarchical vision transformer using shifted windows. *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, 9992–10002. https://doi.org/10.1109/ICCV48922.2021.00986

Luong, T., Pham, H., & Manning, C. D. (2015). Effective approaches to attention-based neural machine translation, 1412–1421. https://doi.org/10.18653/v1/D15-1166

Park, J., Woo, S., Lee, J.-Y., & Kweon, I. S. (2018). Bam: Bottleneck attention module. *The British Machine Vision Conference*. https://arxiv.org/pdf/1807.06514.pdf

Pietropolli, G., Cossarini, G., & Manzoni, L. (2022). Gans for integration of deterministic model and observations in marine ecosystem. *Progress in Artificial Intelligence. EPIA 2022*, 452–463. https://doi.org/10.1007/978-3-031-16474-3_37

Pietropolli, G., Manzoni, L., & Cossarini, G. (2022). Multivariate relationship in big data collection of ocean observing system. *Applied Sciences*, *1*, 1.

Prasad, D. V. V., Venkataramana, L. Y., Kumar, P. S., Prasannamedha, G., Harshana, S., Srividya, S. J., Harrinei, K., & Indraganti, S. (2022). Analysis and prediction of water quality using deep learning and auto deep learning techniques. *Science of The Total Environment*, *821*, 153311. https://doi.org/10.1016/j.scitotenv.2022.153311

Rodríguez-López, L., Bustos Usta, D., Bravo Alvarez, L., Duran-Llacer, I., Lami, A., Martínez-Retureta, R., & Urrutia, R. (2023). Machine learning algorithms for the estimation of water quality parameters in lake llanquihue in southern chile. *Water*, *15*(11), 1994. https://doi.org/10.3390/w15111994

Roemmich, D. (2009). Argo: The challenge of continuing 10 years of progress. *Oceanography (Washington D.C.)*, *22*, 46–55.

Salon, S., Feudale, L., Teruzzi, A., Solidoro, C., Bolzon, G., Brosich, A., Cossarini, G., Di Biagio, V., Lazzari, P., & Coidessa, G. (2021). High-resolution reanalysis of the mediterranean sea biogeochemistry (1999–2019). *Frontiers in Marine Science*, *8*. https://doi.org/10.3389/fmars.2021.741486

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L.-C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 4510–4520. https://doi.org/10.1109/CVPR.2018.00474

Schneider, A., Tanhua, T., Arne, K., & Wallace, D. (2010). High anthropogenic carbon content in the eastern mediterranean. *Journal of Geophysical Research*, *115*. https://doi.org/10.1029/2010JC006171

Simonyan, K., & Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. *3rd International Conference on Learning Representations (ICLR 2015)*, 1–14. https://arxiv.org/pdf/1409.1556.pdf

Solanki, A., Agrawal, H., & Khare, K. (2015). Predictive analysis of water quality parameters using deep learning. *International Journal of Computer Applications*, *125*, 29–34. https://doi.org/10.5120/ijca2015905874

Sonnewald, M., Lguensat, R., Jones, D., Düben, P., Brajard, J., & Balaji, V. (2021). Bridging observations, theory and numerical simulation of the ocean using machine learning. *Environmental Research Letters*, *16*. https://doi.org/10.1088/1748-9326/ac0eb0

Srinivas, A., Lin, T.-Y., Parmar, N., Shlens, J., Abbeel, P., & Vaswani, A. (2021). Bottleneck transformers for visual recognition. *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 16514–16524. https://doi.org/10.1109/CVPR46437.2021.01625

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., & Rabinovich, A. (2015). Going deeper with convolutions. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 1–9. https://doi.org/10.1109/CVPR.2015.7298594

Teruzzi, A., Bolzon, G., Cossarini, G., Lazzari, P., Salon, S., Crise, A., & & Solidoro, C. (2019). Mediterranean sea biogeochemical reanalysis (cmems med-biogeochemistry). *Copernicus Monitoring Environment Marine Service (CMEMS)*. https://doi.org/10.25423/MEDSEA_REANALYSIS_BIO_006_008

Teruzzi, A., Cerboa, P. D., Cossarini, G., Pascoloa, E., & Salon, S. (2019). Parallel implementation of a data assimilation scheme for operational oceanography: The case of the medbfm model system. *Computers & Geosciences*, *124*, 103–114. https://doi.org/10.1016/j.cageo.2019.01.003

Teruzzi, A., Bolzon, G., Salon, S., Lazzari, P., Solidoro, C., & Cossarini, G. (2018). Assimilation of coastal and open sea biogeochemical data to improve phytoplankton simulation in the mediterranean sea. *Ocean Modelling*, *132*, 46–60. https://doi.org/10.1016/j.ocemod.2018.09.007

Teruzzi, A., Cerbo, P., Cossarini, G., Pascolo, E., & Salon, S. (2019). Parallel implementation of a data assimilation scheme for operational oceanography: The case of the medbfm model system. *Computers and Geosciences*, *124*. https://doi.org/10.1016/j.cageo.2019.01.003

Teruzzi, A., Dobricic, S., Solidoro, C., & Cossarini, G. (2014). A 3-d variational assimilation scheme in coupled transport-biogeochemical models: Forecast of mediterranean biogeochemical properties. *Journal of Geophysical Research: Oceans*, *119*(1), 200–217. https://doi.org/10.1002/2013JC009277

Touvron, H., Cord, M., Douze, M., Massa, F., Sablayrolles, A., & Jégou, H. (2021). Training data-efficient image transformers & distillation through attention. *International Conference on Machine Learning*, 10347–10357. https://arxiv.org/pdf/2012.12877.pdf

Touvron, H., Cord, M., Sablayrolles, A., Synnaeve, G., & Jégou, H. (2021). Going deeper with image transformers. *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, 32–42. https://doi.org/10.1109/ICCV48922.2021.00010

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, 6000–6010. https://arxiv.org/pdf/1706.03762.pdf

Vichi, M., Lovato, T., Butenschön, M., Tedesco, L., Lazzari, P., Cossarini, G., Masina, S., Pinardi, N., Solidoro, C., & Zavatarelli, M. (2020). The biogeochemical flux model (bfm): Equation description and user manual. bfm version 5.2. bfm report series n. 1. http://bfm-community.eu

Wang, F., Jiang, M., Qian, C., Yang, S., Li, C., Zhang, H., Wang, X., & Tang, X. (2017). Residual attention network for image classification. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 6450–6458. https://doi.org/10.1109/CVPR.2017.683

Wang, W., Xie, E., Li, X., Fan, D.-P., Song, K., Liang, D., Lu, T., Luo, P., & Shao, L. (2021). Pyramid vision transformer: A versatile backbone for dense prediction without convolutions. *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, 548–558. https://doi.org/10.1109/ICCV48922.2021.00061

Woo, S., Park, J., Lee, J.-Y., & Kweon, I. S. (2018). Cbam: Convolutional block attention module. *Computer Vision – ECCV 2018: 15th European Conference, Munich, Germany, September 8–14, 2018, Proceedings, Part VII*, 3–19. https://doi.org/10.1007/978-3-030-01234-2_1

Xu, W., Xu, Y., Chang, T., & Tu, Z. (2021). Co-scale conv-attentional image transformers. *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, 9961–9970. https://doi.org/10.1109/ICCV48922.2021.00983

Yuan, L., Chen, Y., Wang, T., Yu, W., Shi, Y., Jiang, Z., Tay, F. E., Feng, J., & Yan, S. (2021). Tokens-to-token vit: Training vision transformers from scratch on imagenet, 538–547. https://doi.org/10.1109/ICCV48922.2021.00060

Zhang, H., Wu, C., Zhang, Z., Zhu, Y., Lin, H., Zhang, Z., Sun, Y., He, T., Mueller, J., Manmatha, R., Li, M., & Smola, A. (2022). Resnest: Split-attention networks. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2735–2745. https://doi.org/10.1109/CVPRW56347.2022.00309

Zhou, D., Kang, B., Jin, X., Yang, L., Lian, X., Jiang, Z., Hou, Q., & Feng, J. (2021). Deepvit: Towards deeper vision transformer. https://arxiv.org/pdf/2103.11886.pdf

APPENDIX I

Convolutional Transformer's Algorithm

I.1 Algorithm for an input size of 30 x 30 x 30

```
import torch
import torch.nn as nn
import torch.nn.functional as F
from timm.models.layers import DropPath, trunc_normal_
from timm.models.vision_transformer import _cfg

dims = [64, 128, 320]
head = 64
kernel_sizes = [5, 3, 5]
expansions = [8, 8, 4]
grid_sizes = [7, 5, 1]
ds_ratios = [8, 4, 2]
depths = [3, 6, 18]
drop_rate = 0
drop_path_rate = 0.1




class Upsample(nn.Module):
    def __init__(self, in_channels, out_channels, kernel_size, padding = 0, stride = 1):
        super().__init__()
        self.convT = nn.ConvTranspose3d(in_channels, out_channels, kernel_size, stride = stride, padding = padding)
        self.bn = nn.GroupNorm(1, out_channels, eps = 1e-6)

    def forward(self, x):
        x = self.convT(x)
        x = self.bn(x)

        return x



```

```
class Downsample(nn.Module):
    def __init__(self, in_channels, out_channels, kernel_size, stride = 1, padding = 0):
        super().__init__()
        self.conv = nn.Conv3d(in_channels, out_channels, kernel_size, stride = stride, padding = padding)
        self.bn = nn.GroupNorm(1, out_channels, eps = 1e-6)

    def forward(self, x):
        x = self.conv(x)
        x = self.bn(x)

        return x




class HMHSA(nn.Module):
    def __init__(self, dim, head, grid_size, ds_ratio, drop = 0.):
        super().__init__()
        self.num_heads = dim // head
        self.grid_size = grid_size
        self.head = head
        self.dim = dim

        assert (self.num_heads * head == dim), "Dim needs to be divisible by Head."

        self.qkv = nn.Conv3d(dim, dim * 3, 1)
        self.proj = nn.Conv3d(dim, dim, 1)
        self.norm = nn.GroupNorm(1, dim, eps = 1e-6)
        self.drop = nn.Dropout3d(drop, inplace = True)

        if self.grid_size > 1:
            self.attention_norm = nn.GroupNorm(1, dim, eps = 1e-6)
            self.avg_pool = nn.AvgPool3d(ds_ratio, stride = ds_ratio)
            self.ds_norm = nn.GroupNorm(1, dim, eps = 1e-6)
            self.q = nn.Conv3d(dim, dim, 1)
            self.kv = nn.Conv3d(dim, dim * 2, 1)

    def forward(self, x):
        N, C, H, W, D = x.shape
        qkv = self.qkv(self.norm(x))

        if self.grid_size > 1:
            grid_h, grid_w, grid_d = H // self.grid_size, W // self.grid_size, D // self.grid_size
            qkv = qkv.reshape(N, 3, self.num_heads, self.head, grid_h, self.grid_size, grid_w, self.grid_size, grid_d, self.grid_size)
            qkv = qkv.permute(1, 0, 2, 4, 6, 8, 5, 7, 9, 3)
            qkv = qkv.reshape(3, -1, self.grid_size * self.grid_size * self.grid_size, self.head)
            query, key, value = qkv[0], qkv[1], qkv[2]

            attention = (query / (self.dim ** (1/2))) @ key.transpose(-2, -1)
            attention = attention.softmax(dim = -1)

            attention_x = (attention @ value).reshape(N, self.num_heads, grid_h, grid_w, grid_d, self.grid_size, self.grid_size, self.grid_size, self.head)
            attention_x = attention_x.permute(0, 1, 8, 2, 5, 3, 6, 4, 7).reshape(N, C, H, W, D)
```

```
            attention_x = self.attention_norm(x + attention_x)

            kv = self.kv(self.ds_norm(self.avg_pool(attention_x)))

            query = self.q(attention_x).reshape(N, self.num_heads, self.head, -1)
            query = query.transpose(-2, -1)
            kv = kv.reshape(N, 2, self.num_heads, self.head, -1)
            kv = kv.permute(1, 0, 2, 4, 3)
            key, value = kv[0], kv[1]

        else:
            qkv = qkv.reshape(N, 3, self.num_heads, self.head, -1)
            qkv = qkv.permute(1, 0, 2, 4, 3)
            query, key, value = qkv[0], qkv[1], qkv[2]

        attention = (query / (self.dim ** (1/2))) @ key.transpose(-2, -1)
        attention = attention.softmax(dim = -1)

        global_attention_x = (attention @ value).transpose(-2, -1).reshape(N, C, H, W, D)

        if self.grid_size > 1:
            global_attention_x = global_attention_x + attention_x

        x = self.drop(self.proj(global_attention_x))

        return x



class MBConv(nn.Module):
    def __init__(self, in_channels, out_channels, expansion, kernel_size, drop = 0., act_layer=nn.SiLU):
        expanded_channels = in_channels * expansion
        out_channels = out_channels
        padding = (kernel_size - 1) // 2
        super().__init__()
        self.conv1 = nn.Sequential(
            nn.GroupNorm(1, in_channels, eps = 1e-6),
            nn.Conv3d(in_channels, expanded_channels, kernel_size = 1, padding = 0, bias = False),
            act_layer(inplace = True)
        )
        self.conv2 = nn.Sequential(
            nn.Conv3d(expanded_channels, expanded_channels, kernel_size = kernel_size, padding = padding, groups = expanded_channels, bias = False),
            act_layer(inplace = True)
        )
        self.conv3 = nn.Sequential(
            nn.Conv3d(expanded_channels, out_channels, kernel_size = 1, padding = 0, bias = False),
            nn.GroupNorm(1, out_channels, eps = 1e-6)
        )
        self.drop = nn.Dropout3d(drop, inplace = True)

    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = self.drop(x)
        x = self.conv3(x)
```

```
        x = self.drop(x)

        return x



class Block(nn.Module):
    def __init__(self, dim, head, grid_size = 1, ds_ratio = 1, expansion = 4, drop = 0., drop_path = 0., kernel_size = 3, act_layer = nn.SiLU):
        super().__init__()
        self.drop_path = DropPath(drop_path) if drop_path > 0. else nn.Identity()
        self.attn = HMHSA(dim, head, grid_size = grid_size, ds_ratio = ds_ratio, drop = drop)
        self.conv = MBConv(in_channels = dim, out_channels = dim, expansion = expansion, kernel_size = kernel_size, drop = drop, act_layer = act_layer)

    def forward(self, x):
        x = x + self.drop_path(self.attn(x))
        x = x + self.drop_path(self.conv(x))
        return x




class HAT_Net(nn.Module):
    def __init__(self, img_size = 224, in_chans = 3, num_classes = 1000, act_layer = nn.SiLU):
        super(HAT_Net, self).__init__()
        self.depths = depths

        # sequential CNNs
        self.CNN = nn.Sequential(
            nn.Conv3d(in_channels = 3, out_channels = 16, kernel_size = 2, stride = 1, padding = 0),
            nn.GroupNorm(1, 16, eps = 1e-6),
            act_layer(inplace = True),
            nn.Conv3d(in_channels = 16, out_channels = 64, kernel_size = 2, stride = 1, padding = 0),
            )

        # block -> H-MSHA + MLP
        self.blocks = []
        dpr = [x.item() for x in torch.linspace(0, drop_path_rate, sum(depths))]
        for stage in range(len(dims)):
            self.blocks.append(nn.ModuleList([Block(
                dim = dims[stage], head = head, kernel_size = kernel_sizes[stage], expansion = expansions[stage],
                grid_size = grid_sizes[stage], ds_ratio = ds_ratios[stage], drop = drop_rate, drop_path = dpr[sum(depths[:stage]) + i])
                for i in range(depths[stage])])) # will calculate each block depth times
        self.blocks = nn.ModuleList(self.blocks)

        # downsamples
        self.ds1 = Downsample(in_channels = dims[0], out_channels = dims[1], kernel_size = 2, stride = 2, padding = 1)
        self.ds2 = Downsample(in_channels = dims[1], out_channels = dims[2], kernel_size = 3, stride = 2, padding = 1)

        # upsamples
```

```
        self.us2 = Upsample(in_channels = dims[2], out_channels = dims[1], kernel_size = 3, stride = 2, padding = 1)
        self.us3 = Upsample(in_channels = dims[1], out_channels = dims[0], kernel_size = 2, stride = 2, padding = 1)

        self.TCNN = nn.Sequential(
            nn.ConvTranspose3d(in_channels = dims[0], out_channels = 16, kernel_size = 2, stride = 1, padding = 0),
            nn.GroupNorm(1, 16, eps = 1e-6),
            act_layer(inplace = True),
            nn.ConvTranspose3d(in_channels = 16, out_channels = 1, kernel_size = 2, stride = 1, padding = 0),
        )

        self.apply(self._init_weights)

    def _init_weights(self, m):
        if isinstance(m, (nn.Linear, nn.Conv3d)):
            trunc_normal_(m.weight, std = .02)
            if m.bias is not None:
                nn.init.constant_(m.bias, 0)
        elif isinstance(m, (nn.LayerNorm, nn.BatchNorm3d, nn.GroupNorm)):
            nn.init.constant_(m.bias, 0)
            nn.init.constant_(m.weight, 1.0)

    def forward(self, x):
        x = self.CNN(x)
        for block in self.blocks[0]:
            x = block(x)
        x = self.ds1(x)
        for block in self.blocks[1]:
            x = block(x)
        x = self.ds2(x)
        for block in self.blocks[2]:
            x = block(x)
        x =self.us2(x)
        x =self.us3(x)
        x =self.TCNN(x)

        return x
```

Listing I.1: File - hmhsa_v2.py

I.2 Algorithm for an input size of 31 x 65 x 75

```
import torch
import torch.nn as nn
import torch.nn.functional as F
from timm.models.layers import DropPath, trunc_normal_
from timm.models.vision_transformer import _cfg

dims = [64, 128, 320, 512]
head = 64
kernel_sizes = [5, 3, 5, 3]
expansions = [8, 8, 4, 4]
grid_sizes = [2, 2, 2, 1]
ds_ratios = [8, 4, 2, 1]
depths = [3, 6, 18, 3]
drop_rate = 0
drop_path_rate = 0.1
```

I

```
16
17
18
19
20 class Upsample(nn.Module):
21     def __init__(self, in_channels, out_channels, kernel_size, padding = 0,
       stride = 1):
22         super().__init__()
23         self.convT = nn.ConvTranspose2d(in_channels, out_channels, kernel_size,
       stride = stride, padding = padding)
24         self.bn = nn.GroupNorm(1, out_channels, eps = 1e-6)
25
26     def forward(self, x):
27         x = self.convT(x)
28         x = self.bn(x)
29
30         return x
31
32
33
34
35 class Downsample(nn.Module):
36     def __init__(self, in_channels, out_channels, kernel_size, stride = 1,
       padding = 0):
37         super().__init__()
38         self.conv = nn.Conv2d(in_channels, out_channels, kernel_size, stride =
       stride, padding = padding)
39         self.bn = nn.GroupNorm(1, out_channels, eps = 1e-6)
40
41     def forward(self, x):
42         x = self.conv(x)
43         x = self.bn(x)
44
45         return x
46
47
48
49
50 class HMHSA(nn.Module):
51     def __init__(self, dim, head, grid_size, ds_ratio, drop = 0.):
52         super().__init__()
53         self.num_heads = dim // head
54         self.grid_size = grid_size
55         self.head = head
56         self.dim = dim
57
58         assert (self.num_heads * head == dim), "Dim needs to be divisible by
       Head."
59
60         self.qkv = nn.Conv2d(dim, dim * 3, 1)
61         self.proj = nn.Conv2d(dim, dim, 1)
62         self.norm = nn.GroupNorm(1, dim, eps = 1e-6)
63         self.drop = nn.Dropout2d(drop, inplace = True)
64
65         if self.grid_size > 1:
66             self.attention_norm = nn.GroupNorm(1, dim, eps = 1e-6)
67             self.avg_pool = nn.AvgPool2d(ds_ratio, stride = ds_ratio)
68             self.ds_norm = nn.GroupNorm(1, dim, eps = 1e-6)
69             self.q = nn.Conv2d(dim, dim, 1)
70             self.kv = nn.Conv2d(dim, dim * 2, 1)
71
```

```
72      def forward(self, x):
73          N, C, H, W = x.shape
74          qkv = self.qkv(self.norm(x))
75
76          if self.grid_size > 1:
77              grid_h, grid_w= H // self.grid_size, W // self.grid_size
78              qkv = qkv.reshape(N, 3, self.num_heads, self.head, grid_h, self.
    grid_size, grid_w, self.grid_size)
79              qkv = qkv.permute(1, 0, 2, 4, 6, 5, 7, 3)
80              qkv = qkv.reshape(3, -1, self.grid_size * self.grid_size, self.head)
81              query, key, value = qkv[0], qkv[1], qkv[2]
82
83              attention = (query / (self.dim ** (1/2))) @ key.transpose(-2, -1)
84              attention = attention.softmax(dim = -1)
85
86              attention_x = (attention @ value).reshape(N, self.num_heads, grid_h,
     grid_w, self.grid_size, self.grid_size, self.head)
87              attention_x = attention_x.permute(0, 1, 6, 2, 4, 3, 5).reshape(N, C,
     H, W)
88              attention_x = self.attention_norm(x + attention_x)
89
90              kv = self.kv(self.ds_norm(self.avg_pool(attention_x)))
91
92              query = self.q(attention_x).reshape(N, self.num_heads, self.head,
    -1)
93              query = query.transpose(-2, -1)
94              kv = kv.reshape(N, 2, self.num_heads, self.head, -1)
95              kv = kv.permute(1, 0, 2, 4, 3)
96              key, value = kv[0], kv[1]
97
98          else:
99              qkv = qkv.reshape(N, 3, self.num_heads, self.head, -1)
100             qkv = qkv.permute(1, 0, 2, 4, 3)
101             query, key, value = qkv[0], qkv[1], qkv[2]
102
103         attention = (query / (self.dim ** (1/2))) @ key.transpose(-2, -1)
104         attention = attention.softmax(dim = -1)
105
106         global_attention_x = (attention @ value).transpose(-2, -1).reshape(N, C,
     H, W) # concatenate
107
108         if self.grid_size > 1:
109             global_attention_x = global_attention_x + attention_x
110
111         x = self.drop(self.proj(global_attention_x)).
112
113         return x
114
115
116
117
118 class MBConv(nn.Module):
119     def __init__(self, in_channels, out_channels, expansion, kernel_size, drop =
     0., act_layer=nn.SiLU):
120         expanded_channels = in_channels * expansion
121         out_channels = out_channels
122         padding = (kernel_size - 1) // 2
123         super().__init__()
124         self.conv1 = nn.Sequential(
125             nn.GroupNorm(1, in_channels, eps = 1e-6),
126             nn.Conv2d(in_channels, expanded_channels, kernel_size = 1, padding =
```

```
    0, bias = False),
            act_layer(inplace = True)
        )
        self.conv2 = nn.Sequential(
            nn.Conv2d(expanded_channels, expanded_channels, kernel_size =
kernel_size, padding = padding, groups = expanded_channels, bias = False),
            act_layer(inplace = True)
        )
        self.conv3 = nn.Sequential(
            nn.Conv2d(expanded_channels, out_channels, kernel_size = 1, padding
= 0, bias = False),
            nn.GroupNorm(1, out_channels, eps = 1e-6)
        )
        self.drop = nn.Dropout2d(drop, inplace = True)

    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = self.drop(x)
        x = self.conv3(x)
        x = self.drop(x)

        return x



class Block(nn.Module):
    def __init__(self, dim, head, grid_size = 1, ds_ratio = 1, expansion = 4,
drop = 0., drop_path = 0., kernel_size = 3, act_layer = nn.SiLU):
        super().__init__()
        self.drop_path = DropPath(drop_path) if drop_path > 0. else nn.Identity
()
        self.attn = HMHSA(dim, head, grid_size = grid_size, ds_ratio = ds_ratio,
 drop = drop)
        self.conv = MBConv(in_channels = dim, out_channels = dim, expansion =
expansion, kernel_size = kernel_size, drop = drop, act_layer = act_layer)

    def forward(self, x):
        x = x + self.drop_path(self.attn(x))
        x = x + self.drop_path(self.conv(x))
        return x




class HAT_Net(nn.Module):
    def __init__(self, img_size = 224, in_chans = 3, num_classes = 1000,
act_layer = nn.SiLU):
        super(HAT_Net, self).__init__()
        self.depths = depths

        # sequential CNNs
        self.CNN = nn.Sequential(
            nn.ConvTranspose2d(in_channels = 3, out_channels = 3, kernel_size =
4, padding = 2, stride = 2),
            nn.GroupNorm(3, 3, eps = 1e-6),
            act_layer(inplace = True),
            nn.ConvTranspose2d(in_channels = 3, out_channels = 3, kernel_size =
4, padding = 2, stride = 2),
            nn.GroupNorm(3, 3, eps = 1e-6),
            act_layer(inplace = True),
```

```
178             nn.Conv2d(in_channels = 3, out_channels = 16, kernel_size = 4,
    stride = 2, padding = 2),
179             nn.GroupNorm(1, 16, eps = 1e-6),
180             act_layer(inplace = True),
181             nn.Conv2d(in_channels = 16, out_channels = 64, kernel_size = 2,
    stride = 2, padding = 0),
182             )
183
184         # block -> H-MSHA + MLP
185         self.blocks = []
186         dpr = [x.item() for x in torch.linspace(0, drop_path_rate, sum(depths))]
187         for stage in range(len(dims)):
188             self.blocks.append(nn.ModuleList([Block(
189                 dim = dims[stage], head = head, kernel_size = kernel_sizes[stage
    ], expansion = expansions[stage],
190                 grid_size = grid_sizes[stage], ds_ratio = ds_ratios[stage], drop
     = drop_rate, drop_path = dpr[sum(depths[:stage]) + i])
191                 for i in range(depths[stage])]))
192         self.blocks = nn.ModuleList(self.blocks)
193
194         # downsamples
195         self.ds1 = Downsample(in_channels = dims[0], out_channels = dims[1],
    kernel_size = 5)
196         self.ds2 = Downsample(in_channels = dims[1], out_channels = dims[2],
    kernel_size = 3)
197         self.ds3 = Downsample(in_channels = dims[2], out_channels = dims[3],
    kernel_size = 2, stride = 2)
198
199         # upsamples
200         self.us1 = Upsample(in_channels = dims[3], out_channels = dims[2],
    kernel_size = 2, stride = 2)
201         self.us2 = Upsample(in_channels = dims[2], out_channels = dims[1],
    kernel_size = 3)
202         self.us3 = Upsample(in_channels = dims[1], out_channels = dims[0],
    kernel_size = 5)
203
204         self.TCNN = nn.Sequential(
205             nn.ConvTranspose2d(in_channels = dims[0], out_channels = 16,
    kernel_size = 2, stride = 2, padding = 0),
206             nn.GroupNorm(1, 16, eps = 1e-6),
207             act_layer(inplace = True),
208             nn.ConvTranspose2d(in_channels = 16, out_channels = 1, kernel_size =
     4, stride = 2, padding = 2),
209             nn.GroupNorm(1, 1, eps = 1e-6),
210             act_layer(inplace = True),
211             nn.Conv2d(in_channels = 1, out_channels = 1, kernel_size = 4, stride
     = 2, padding = 2),
212             nn.GroupNorm(1, 1, eps = 1e-6),
213             act_layer(inplace = True),
214             nn.Conv2d(in_channels = 1, out_channels = 1, kernel_size = 4, stride
     = 2, padding = 2),
215         )
216
217         self.apply(self._init_weights)
218
219     def _init_weights(self, m):
220         if isinstance(m, (nn.Linear, nn.Conv2d)):
221             trunc_normal_(m.weight, std = .02)
222             if m.bias is not None:
223                 nn.init.constant_(m.bias, 0)
224         elif isinstance(m, (nn.LayerNorm, nn.BatchNorm2d, nn.GroupNorm)):
```

I

```
225             nn.init.constant_(m.bias, 0)
226             nn.init.constant_(m.weight, 1.0)
227 
228     def forward(self, x):
229         x = self.CNN(x)
230         for block in self.blocks[0]:
231             x = block(x)
232         x = self.ds1(x)
233         for block in self.blocks[1]:
234             x = block(x)
235         x = self.ds2(x)
236         for block in self.blocks[2]:
237             x = block(x)
238         x = self.ds3(x)
239         for block in self.blocks[3]:
240             x = block(x)
241         x =self.us1(x)
242         x =self.us2(x)
243         x =self.us3(x)
244         x =self.TCNN(x)
245 
246         return x
```

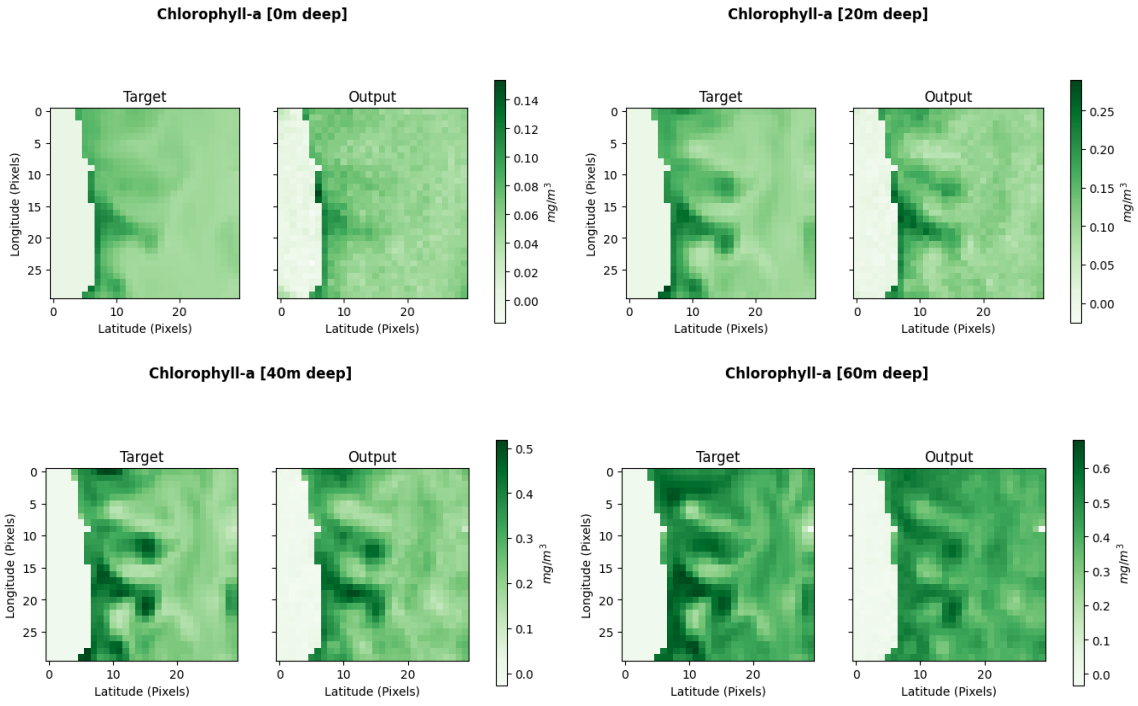
Listing I.2: File - hmhsa_v1.py

APPENDIX II

Additional Results

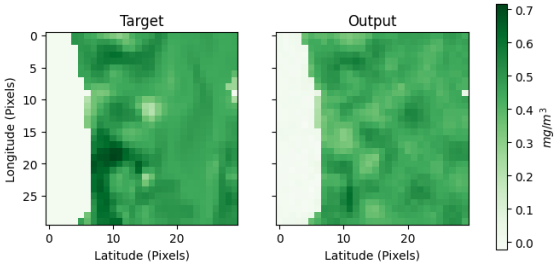
II.1 Model's Output

II.1.1 Chlorophyll-a

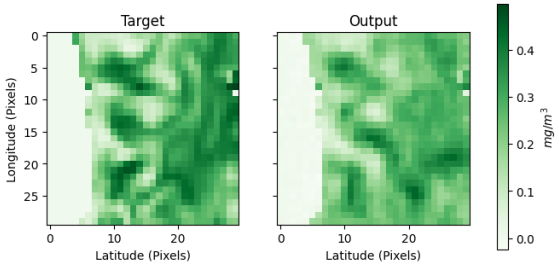


II

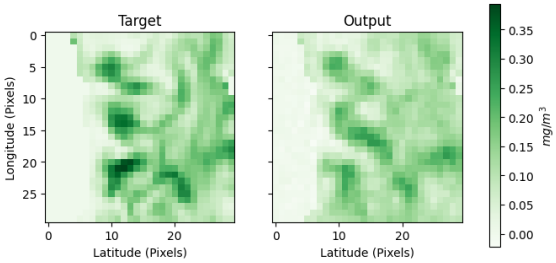
Chlorophyll-a [80m deep]



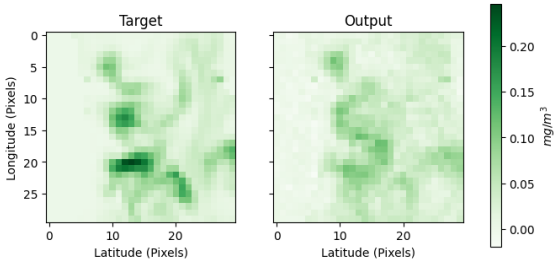
Chlorophyll-a [100m deep]



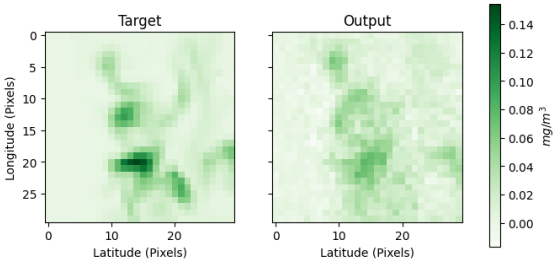
Chlorophyll-a [120m deep]



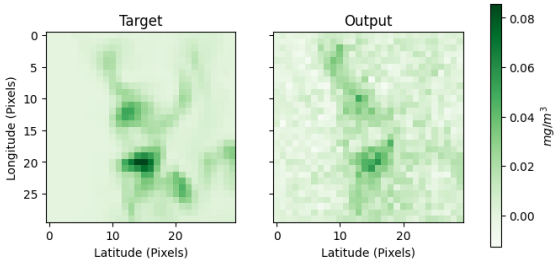
Chlorophyll-a [140m deep]



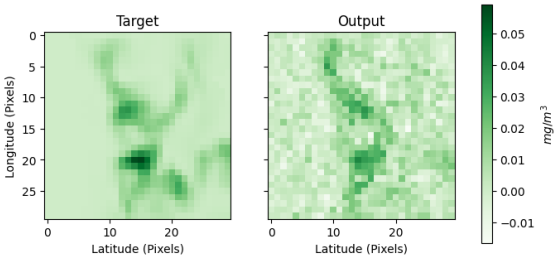
Chlorophyll-a [160m deep]



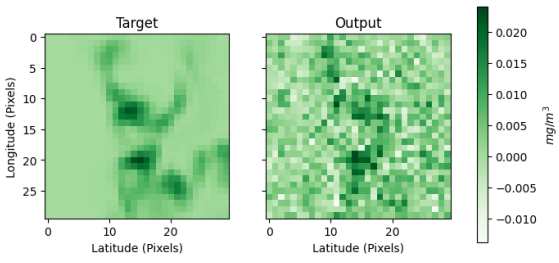
Chlorophyll-a [180m deep]



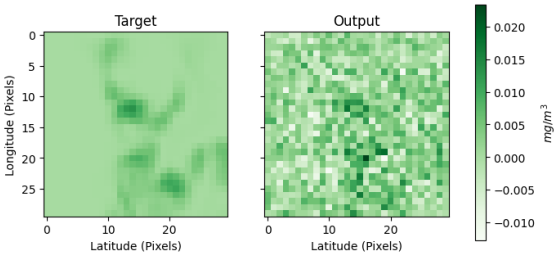
Chlorophyll-a [200m deep]



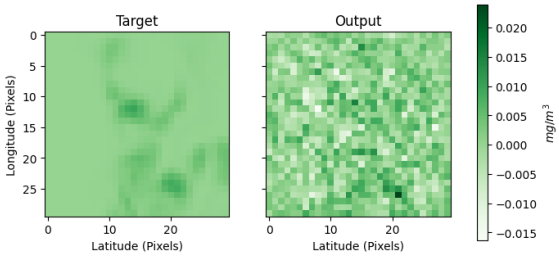
Chlorophyll-a [220m deep]

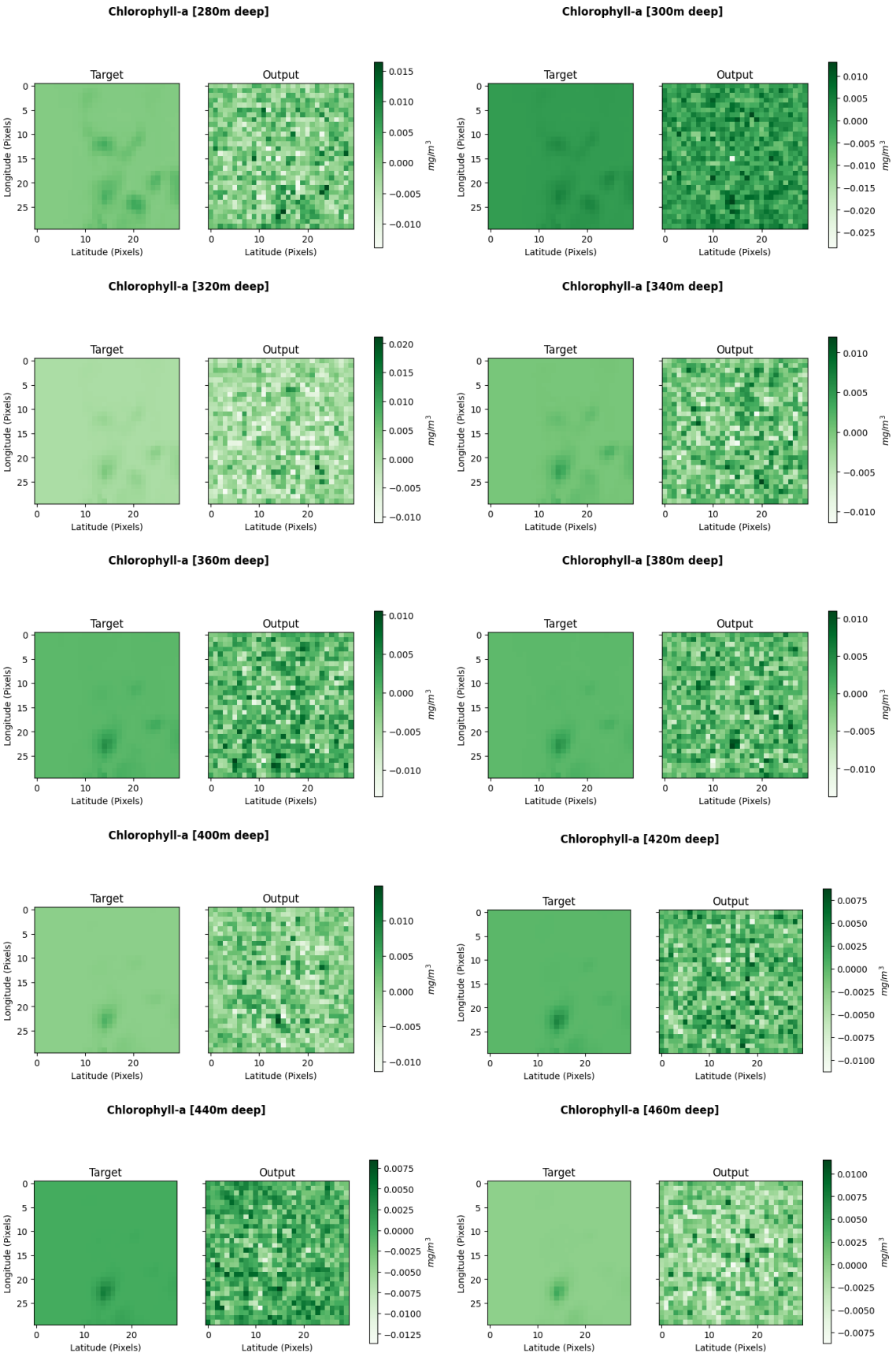


Chlorophyll-a [240m deep]



Chlorophyll-a [260m deep]




Chlorophyll-a [280m deep]
Chlorophyll-a [300m deep]
Chlorophyll-a [320m deep]
Chlorophyll-a [340m deep]
Chlorophyll-a [360m deep]
Chlorophyll-a [380m deep]
Chlorophyll-a [400m deep]
Chlorophyll-a [420m deep]
Chlorophyll-a [440m deep]
Chlorophyll-a [460m deep]
Target
Output
Longitude (Pixels)
Latitude (Pixels)
mg/m^3

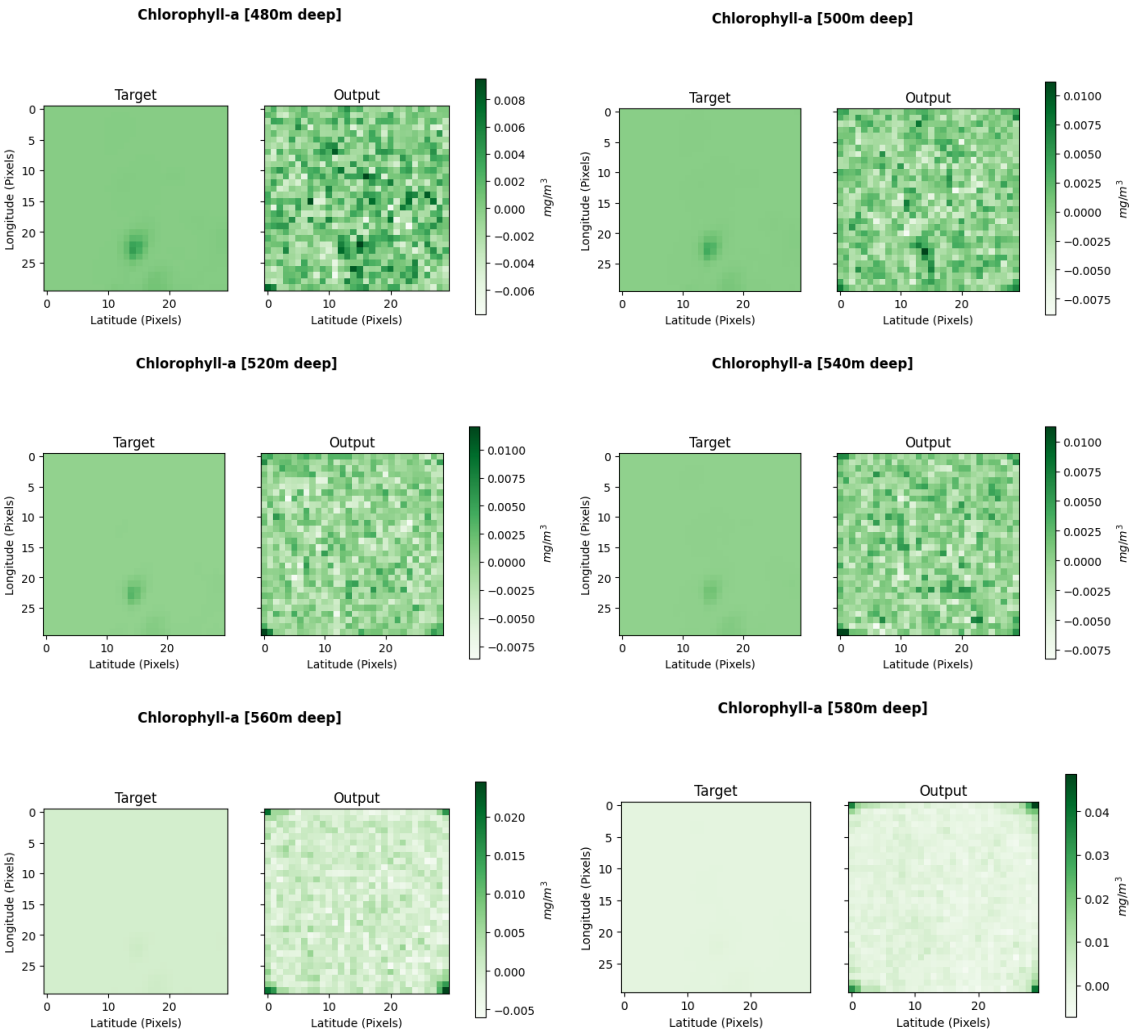
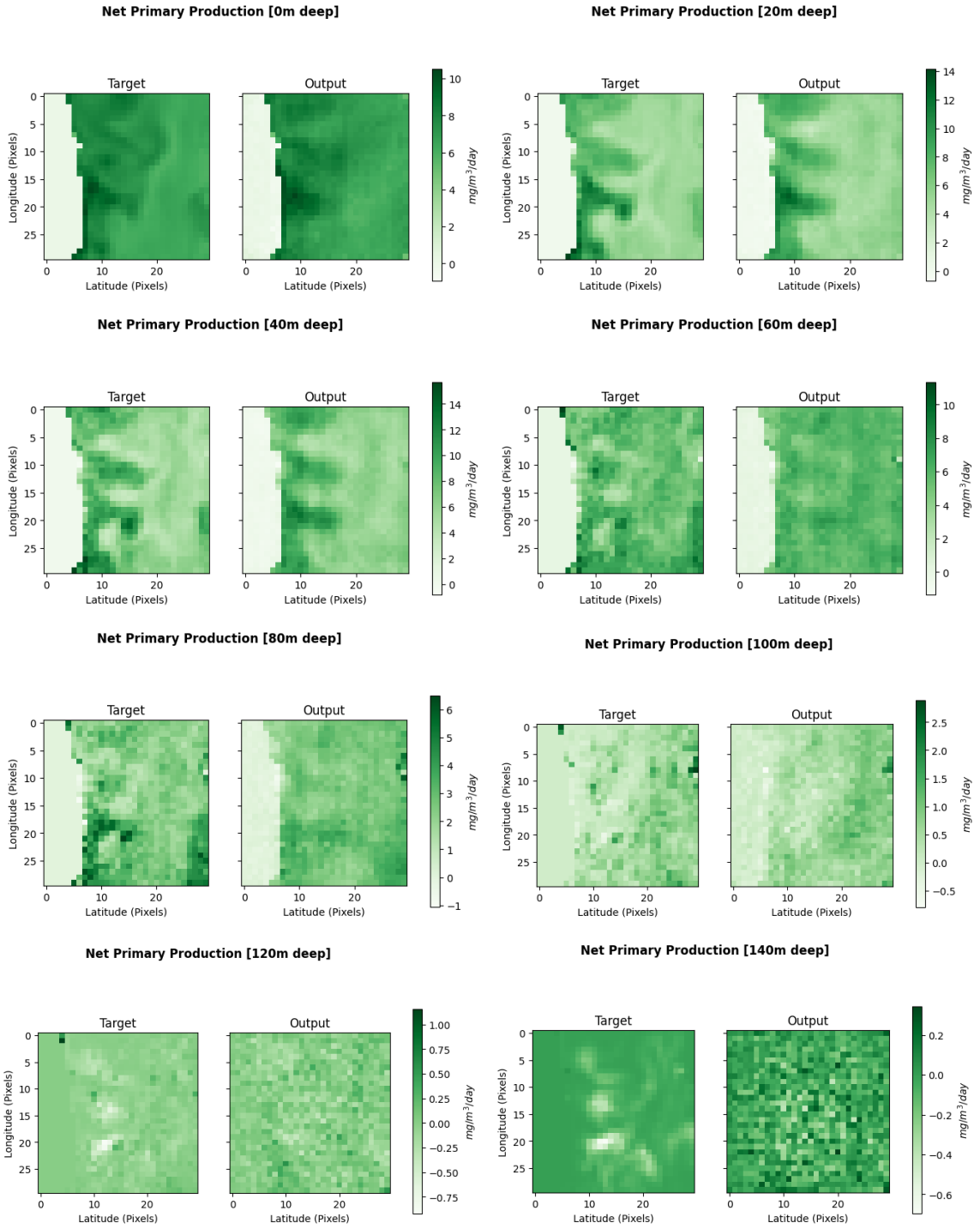


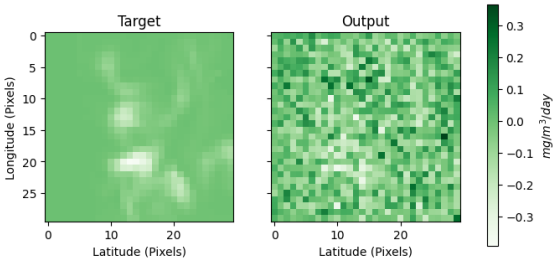
Figure II.1: Chlorophyll-a - Target VS. Output.

II.1.2 Net Primary Production

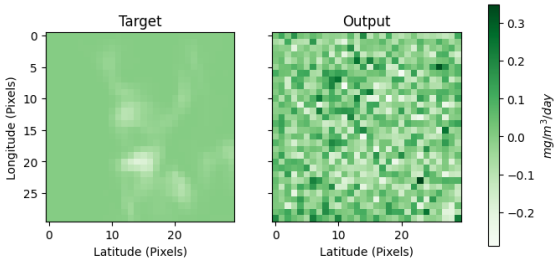


II

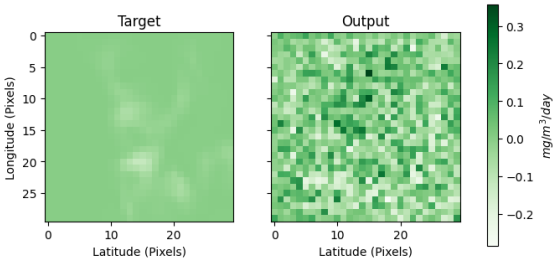
Net Primary Production [160m deep]



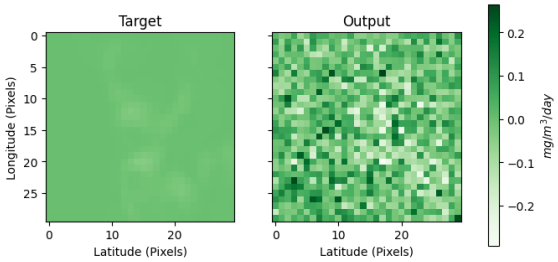
Net Primary Production [180m deep]



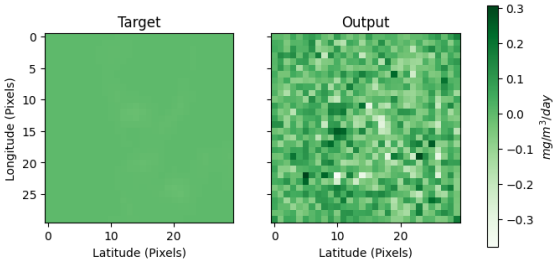
Net Primary Production [200m deep]



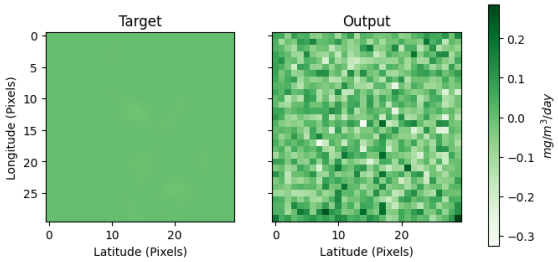
Net Primary Production [220m deep]



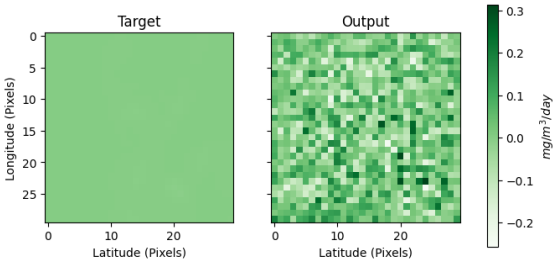
Net Primary Production [240m deep]



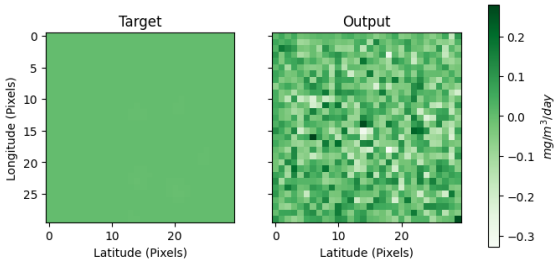
Net Primary Production [260m deep]



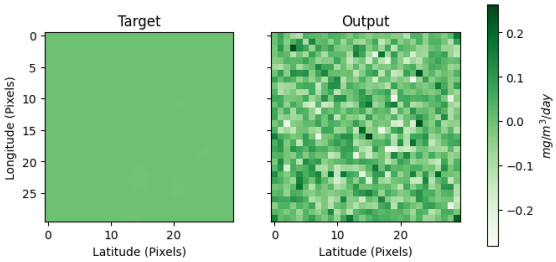
Net Primary Production [280m deep]



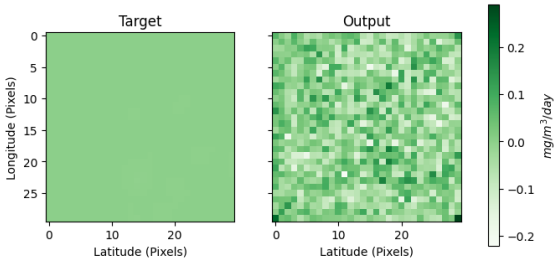
Net Primary Production [300m deep]

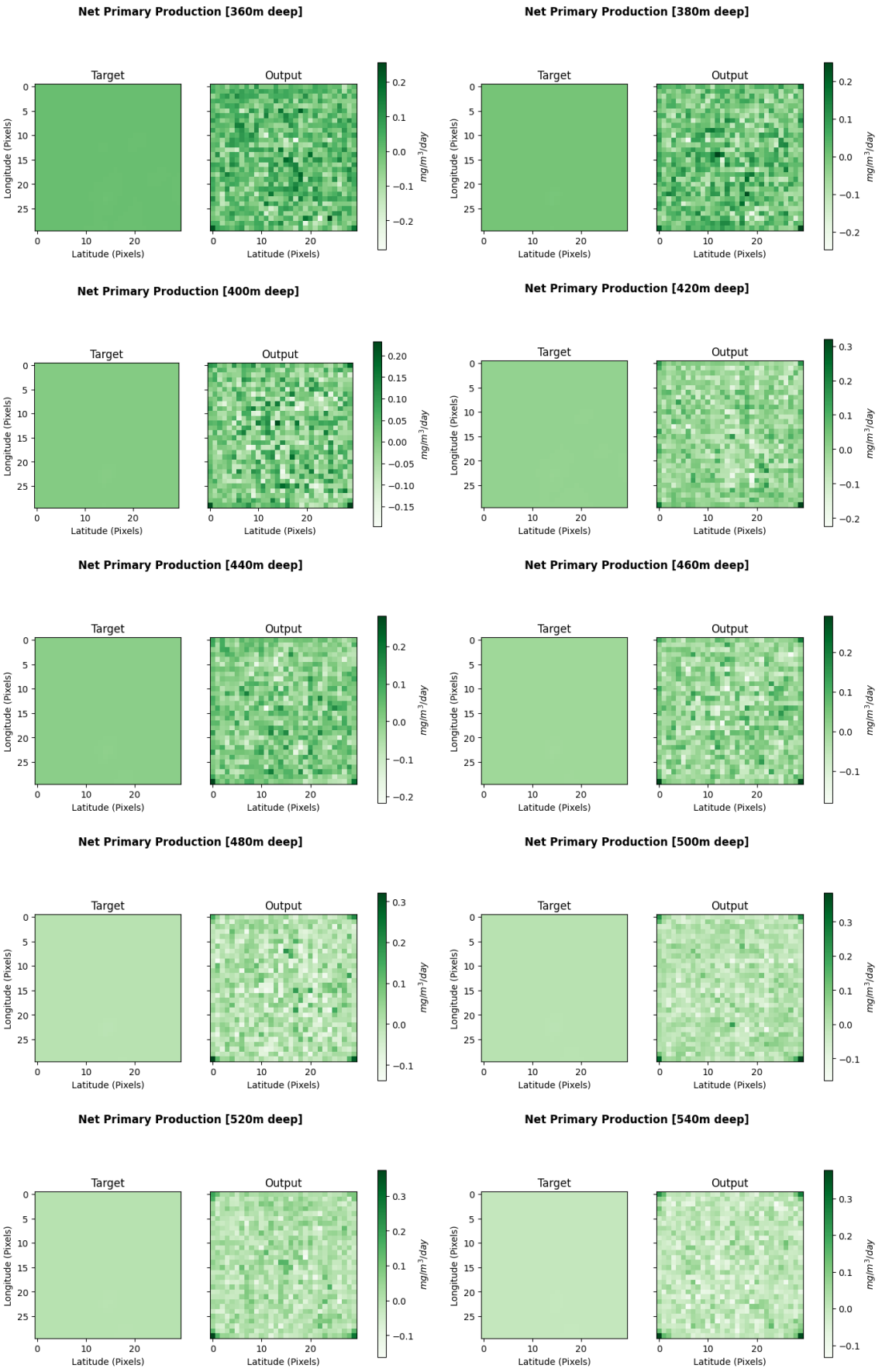


Net Primary Production [320m deep]



Net Primary Production [340m deep]




Net Primary Production [360m deep]
Net Primary Production [380m deep]
Net Primary Production [400m deep]
Net Primary Production [420m deep]
Net Primary Production [440m deep]
Net Primary Production [460m deep]
Net Primary Production [480m deep]
Net Primary Production [500m deep]
Net Primary Production [520m deep]
Net Primary Production [540m deep]
Target
Output
Longitude (Pixels)
Latitude (Pixels)
mg/m^3/day

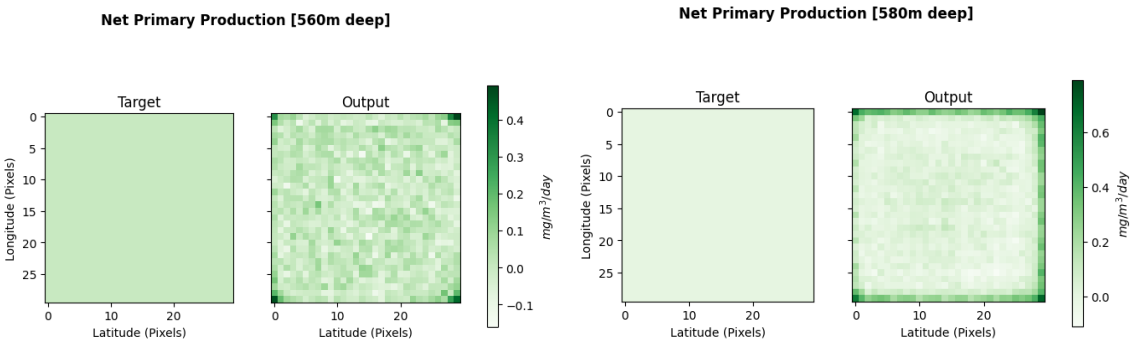
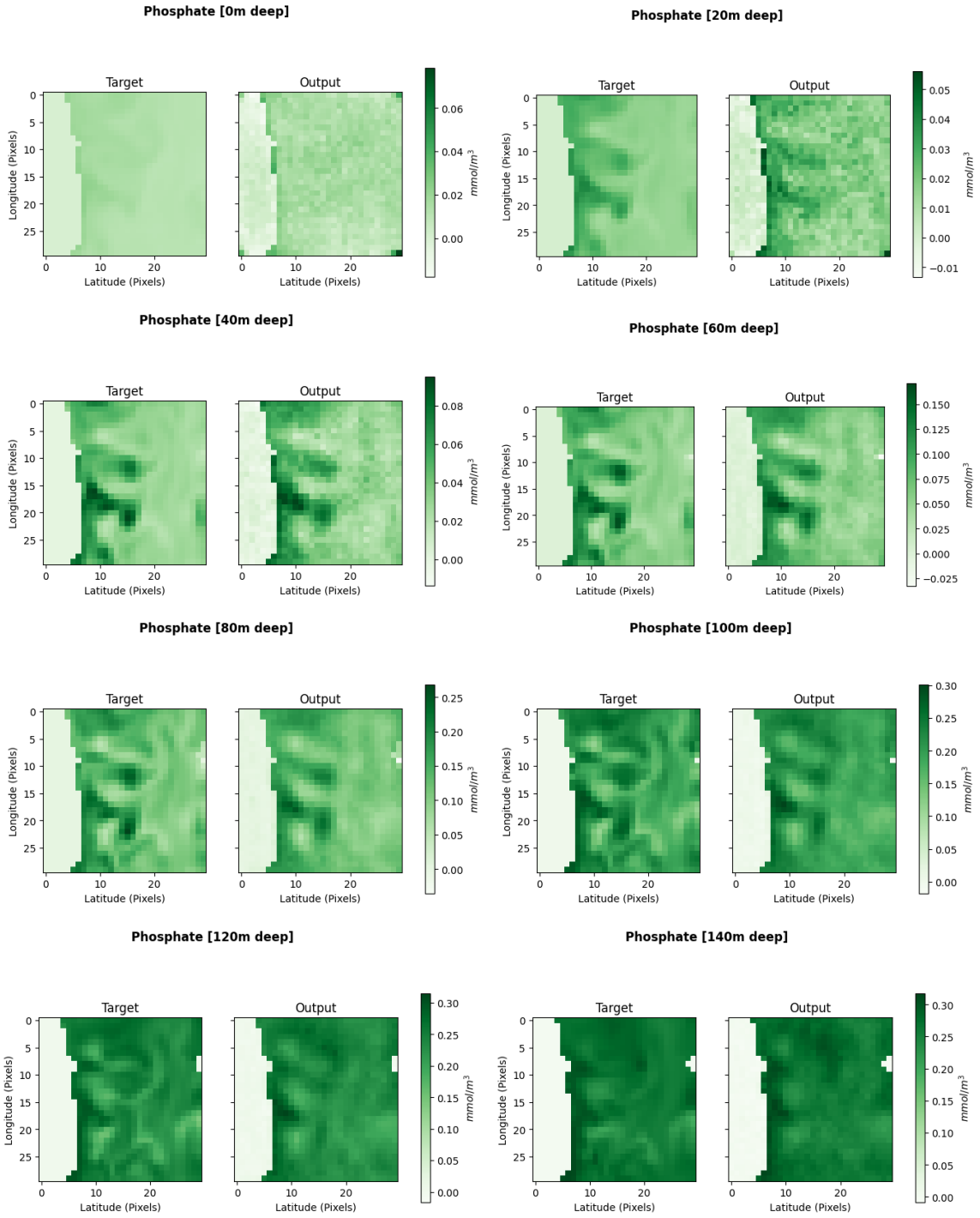
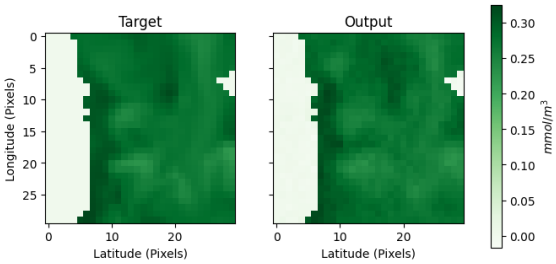


Figure II.2: Net Primary Production - Target VS. Output.

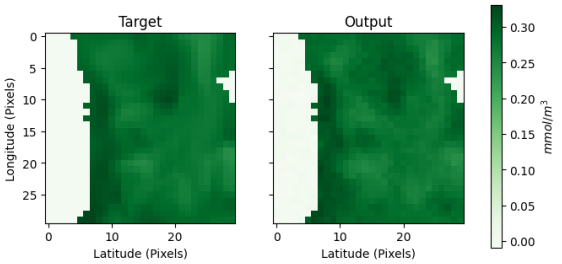
II.1.3 Phosphate



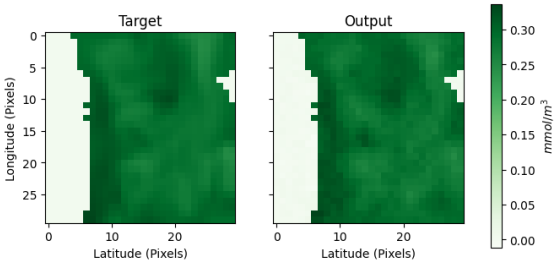
Phosphate [160m deep]



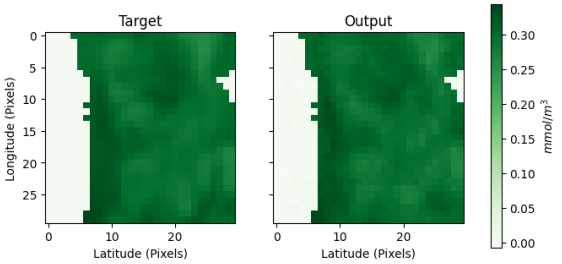
Phosphate [180m deep]



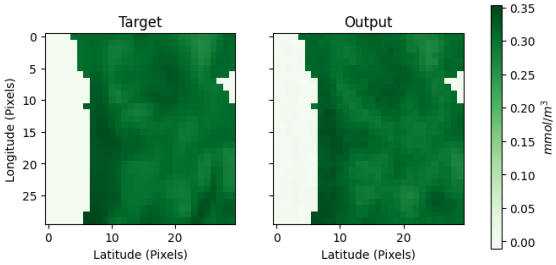
Phosphate [200m deep]



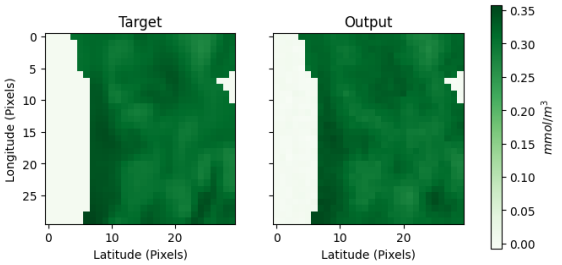
Phosphate [220m deep]



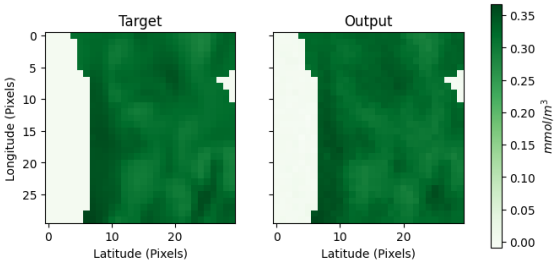
Phosphate [240m deep]



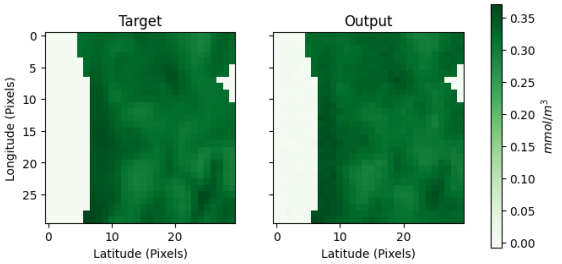
Phosphate [260m deep]



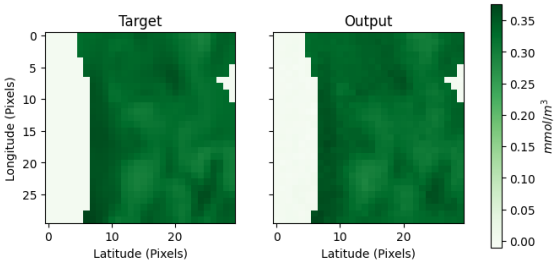
Phosphate [280m deep]



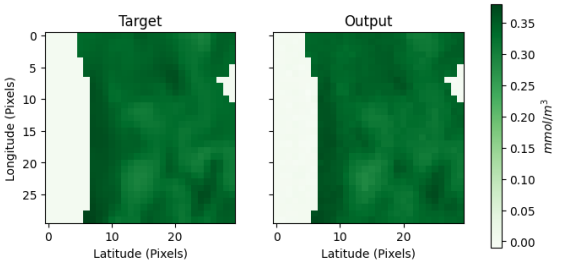
Phosphate [300m deep]

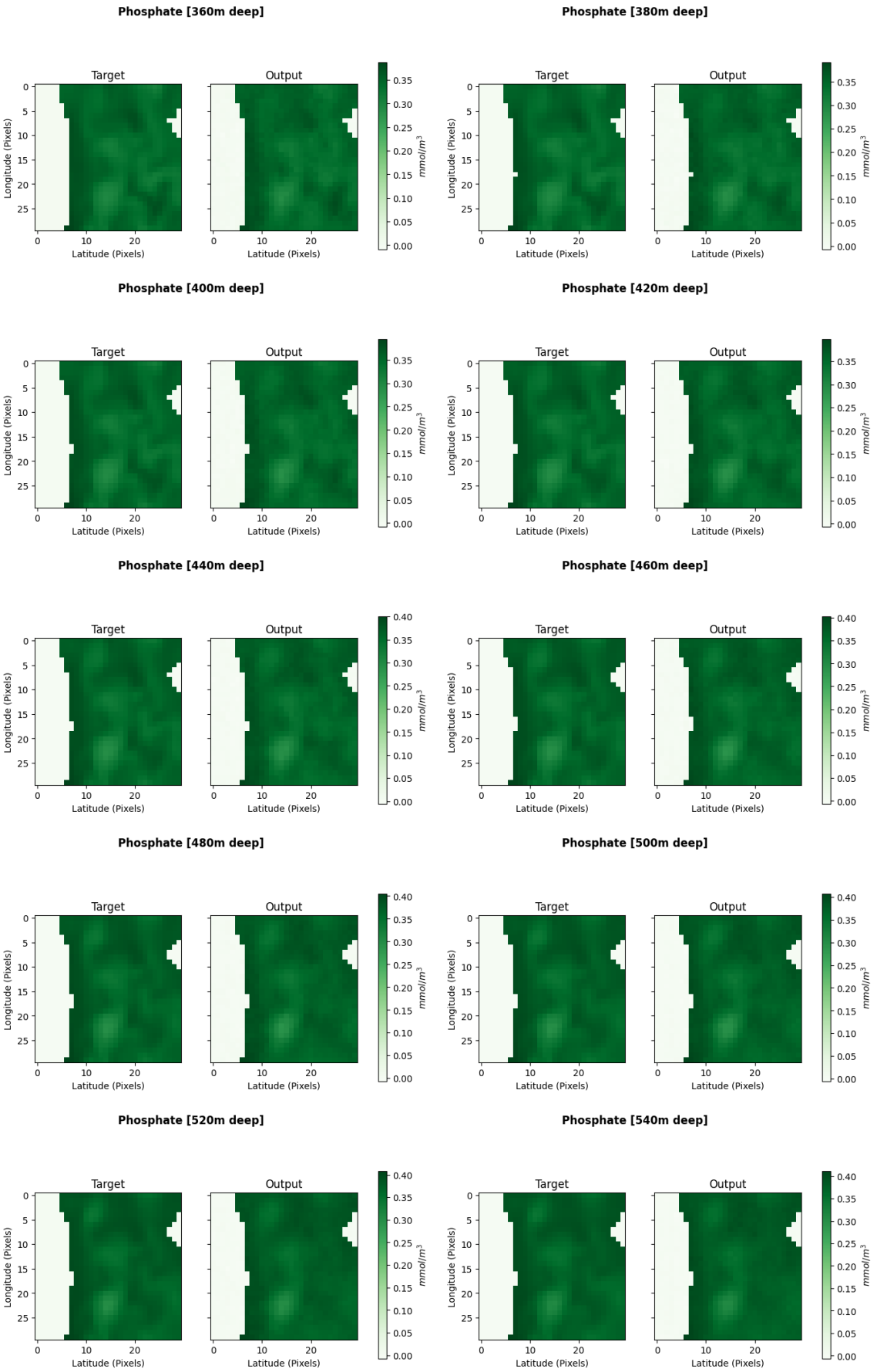


Phosphate [320m deep]



Phosphate [340m deep]




Phosphate [360m deep]
Phosphate [380m deep]
Phosphate [400m deep]
Phosphate [420m deep]
Phosphate [440m deep]
Phosphate [460m deep]
Phosphate [480m deep]
Phosphate [500m deep]
Phosphate [520m deep]
Phosphate [540m deep]
Target
Output
Longitude (Pixels)
Latitude (Pixels)
mmol/m³

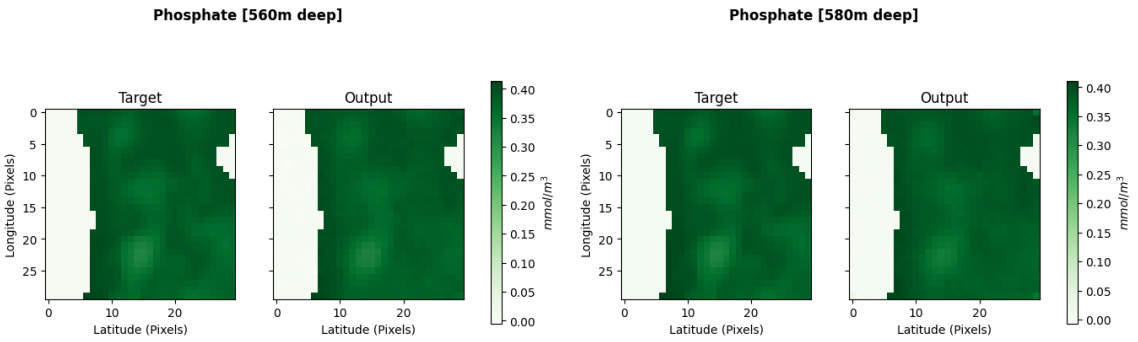
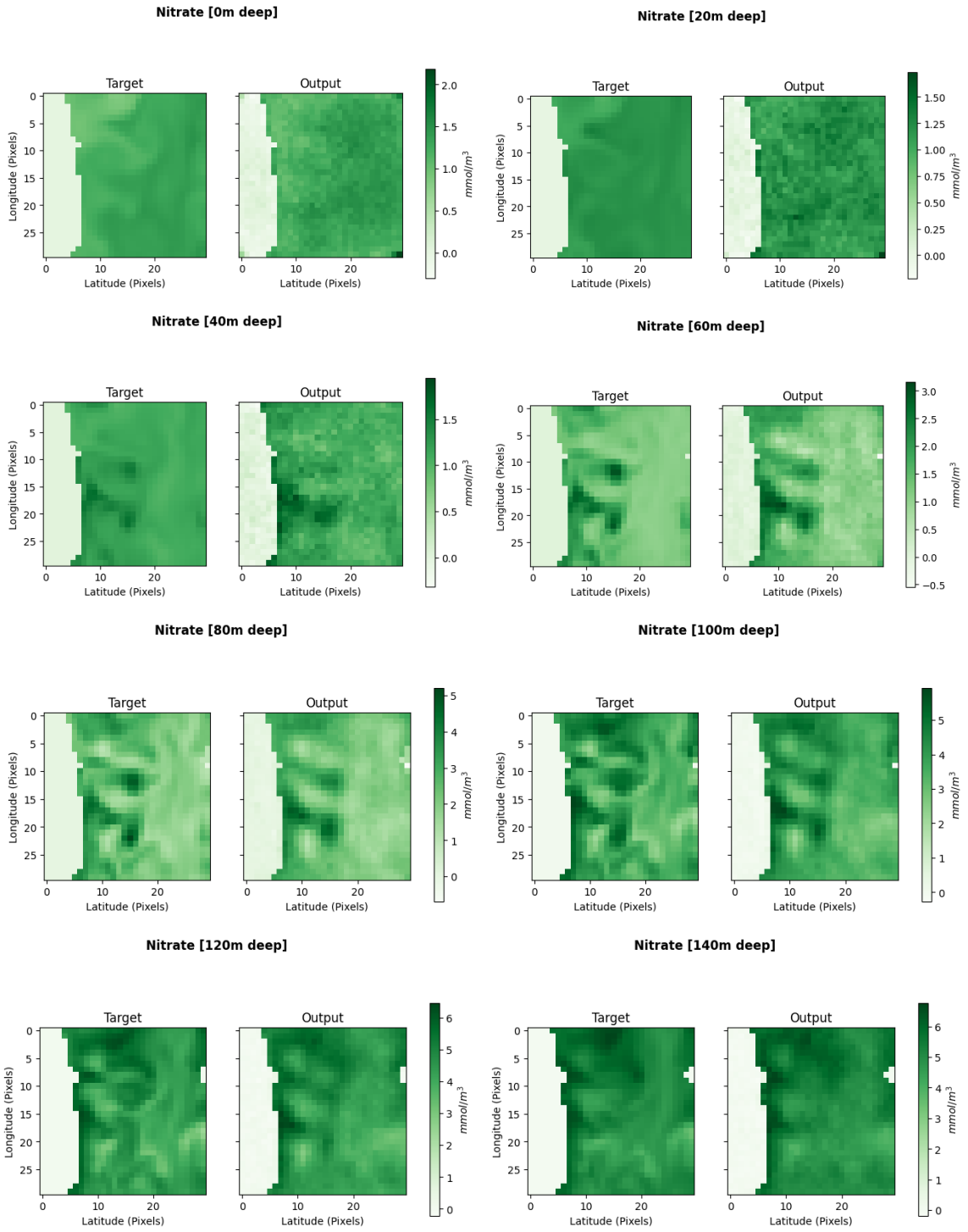


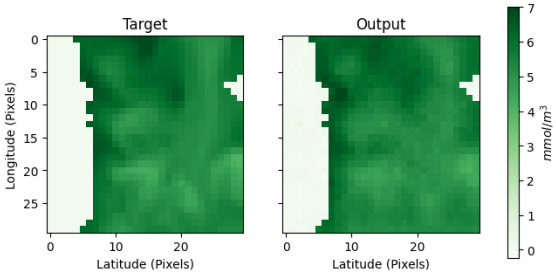
Figure II.3: Phosphate - Target VS. Output.

II.1.4 Nitrate

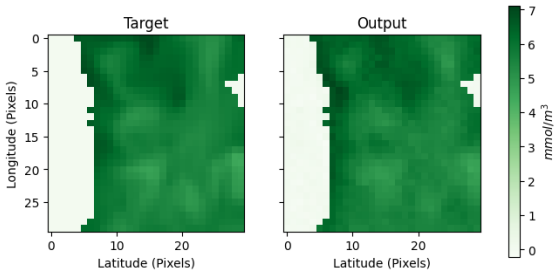


II

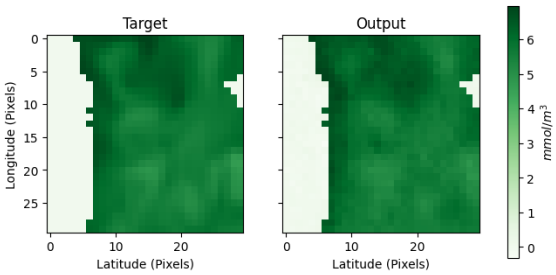
Nitrate [160m deep]



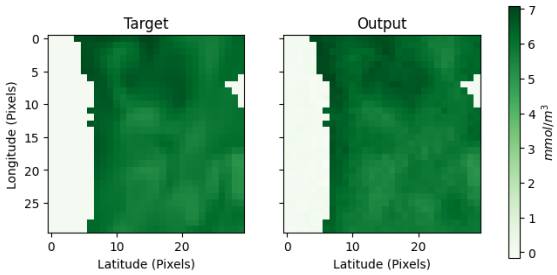
Nitrate [180m deep]



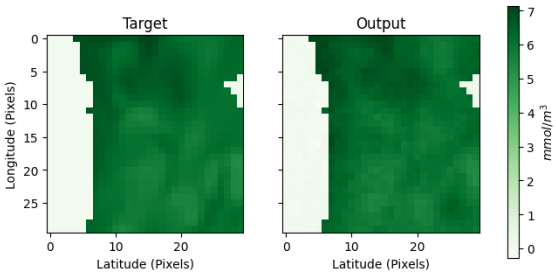
Nitrate [200m deep]



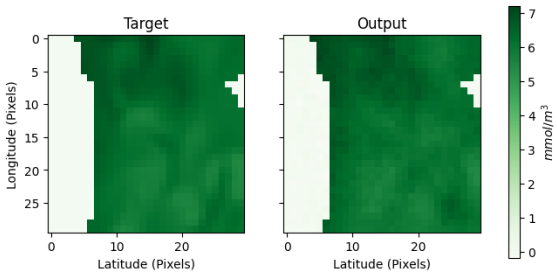
Nitrate [220m deep]



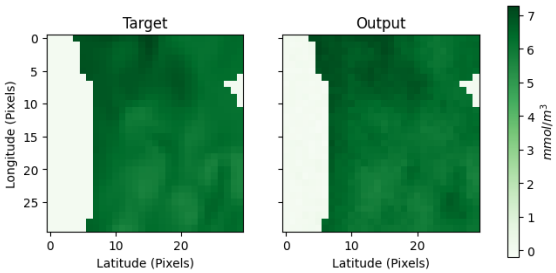
Nitrate [240m deep]



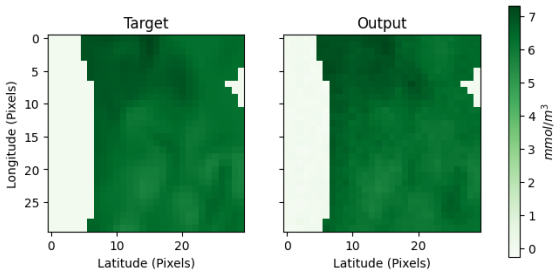
Nitrate [260m deep]



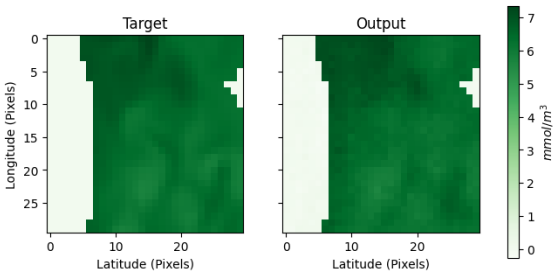
Nitrate [280m deep]



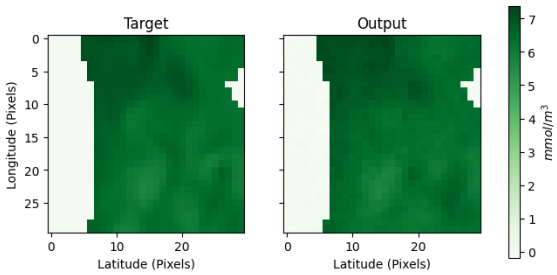
Nitrate [300m deep]



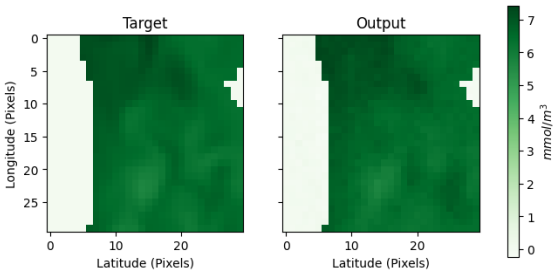
Nitrate [320m deep]



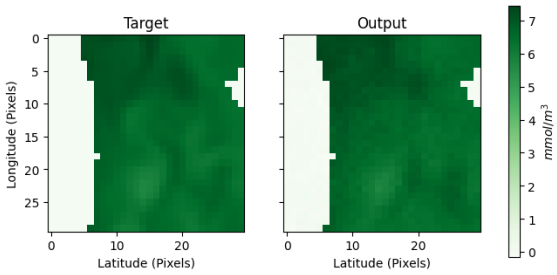
Nitrate [340m deep]



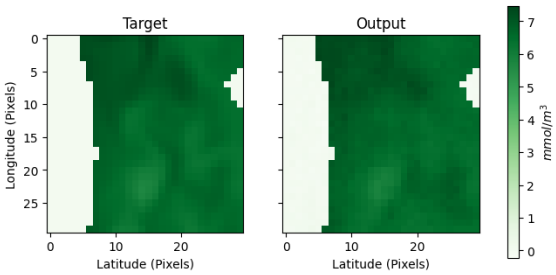
Nitrate [360m deep]



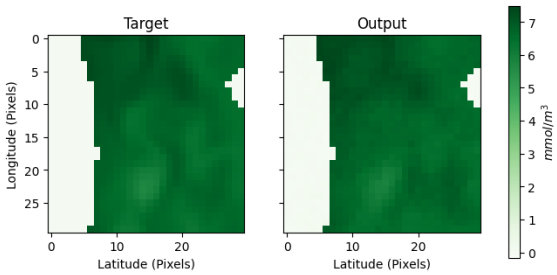
Nitrate [380m deep]



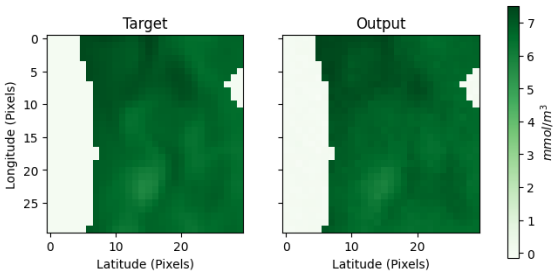
Nitrate [400m deep]



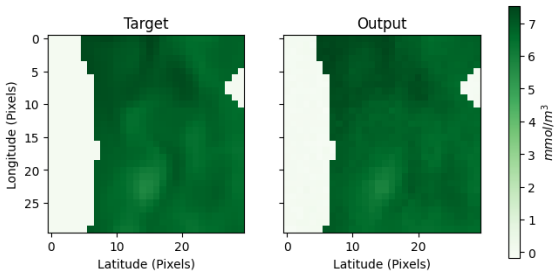
Nitrate [420m deep]



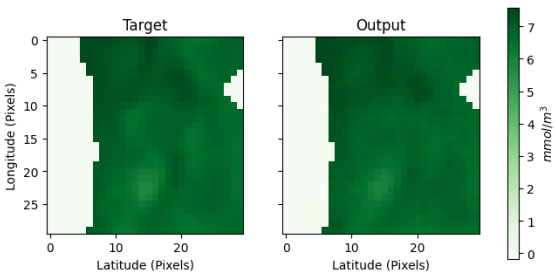
Nitrate [440m deep]



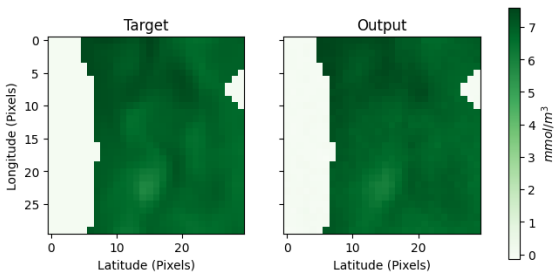
Nitrate [460m deep]



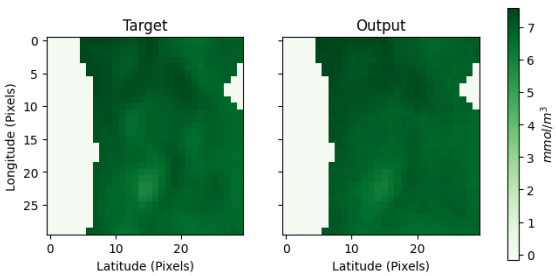
Nitrate [480m deep]



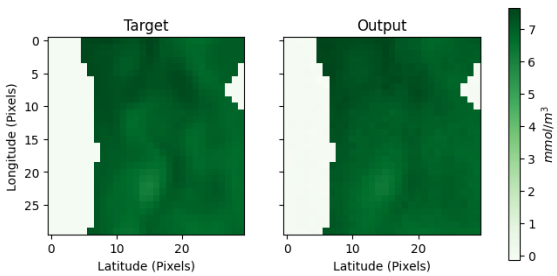
Nitrate [500m deep]



Nitrate [520m deep]



Nitrate [540m deep]



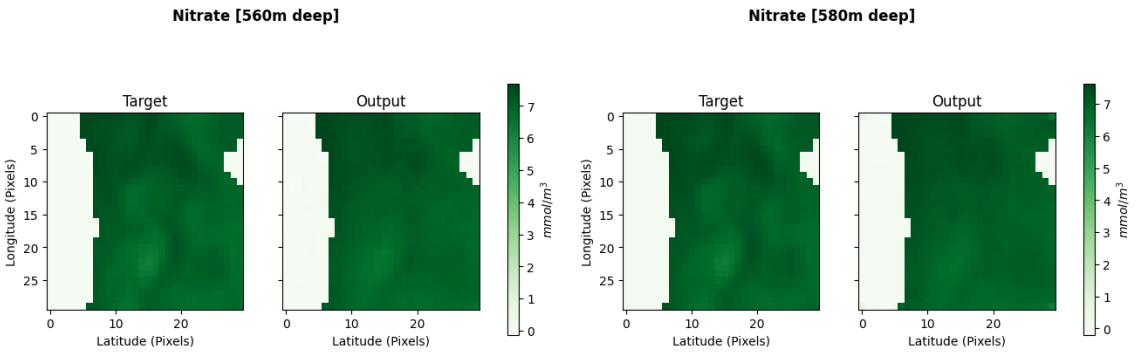
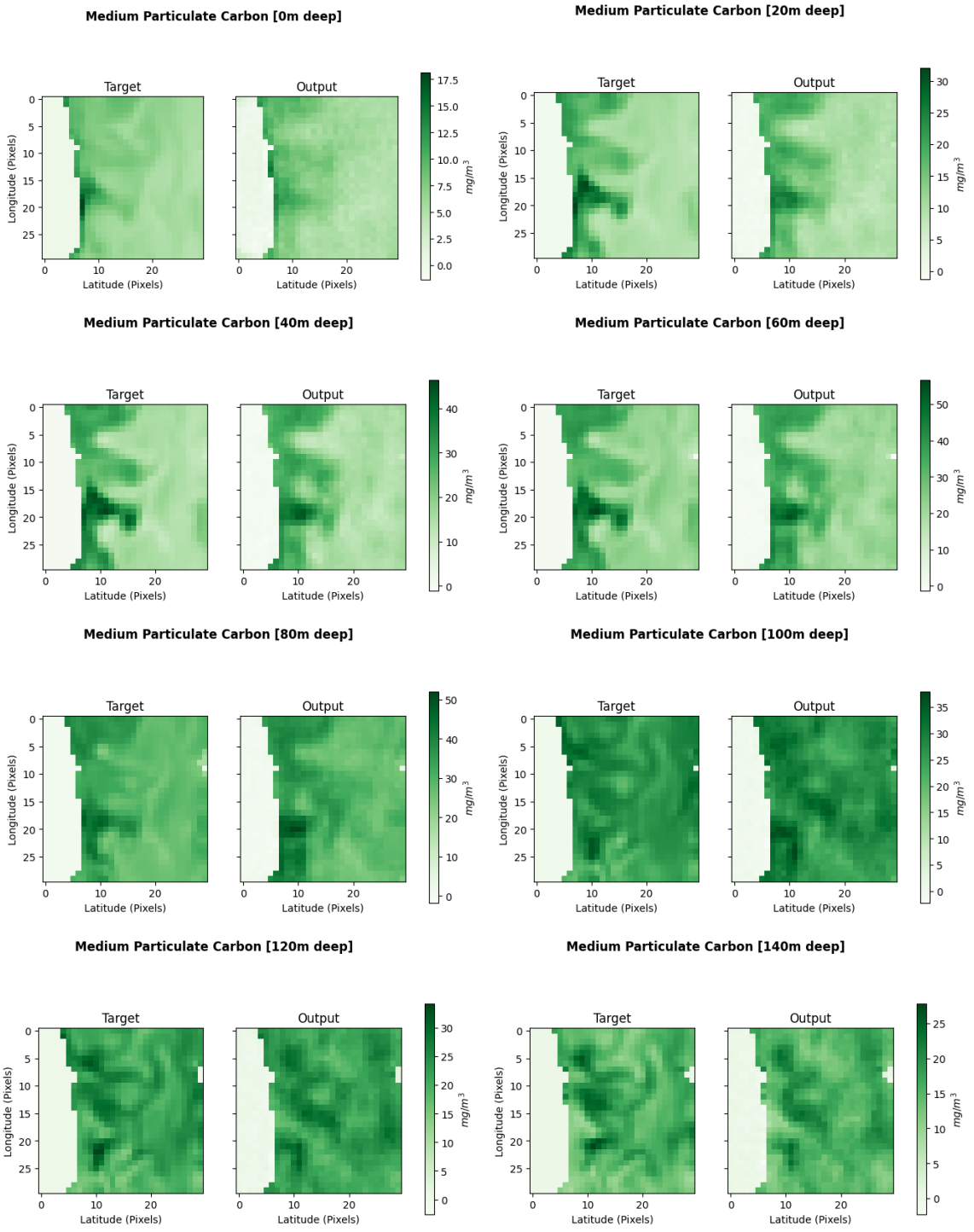


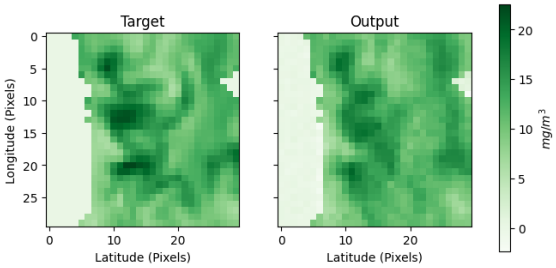
Figure II.4: Nitrate - Target VS. Output.

II.1.5 Medium Particulate Carbon

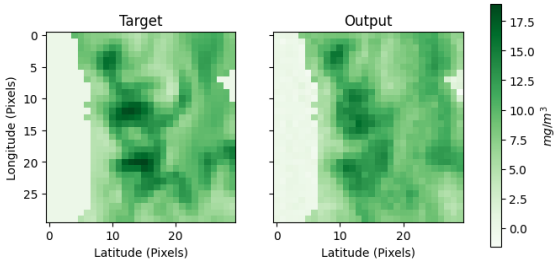


II

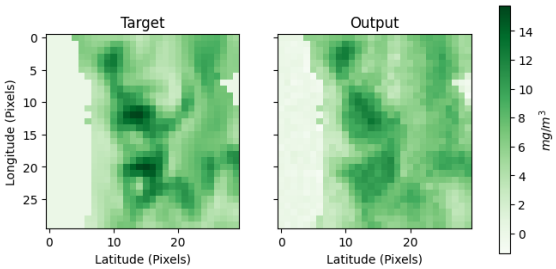
Medium Particulate Carbon [160m deep]



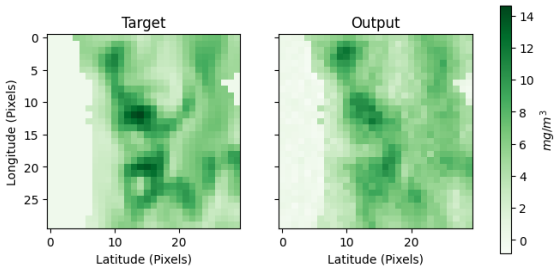
Medium Particulate Carbon [180m deep]



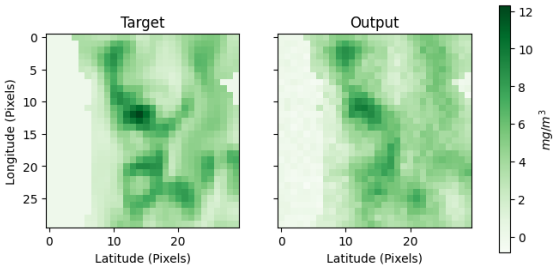
Medium Particulate Carbon [200m deep]



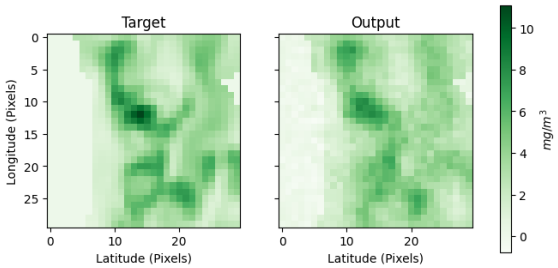
Medium Particulate Carbon [220m deep]



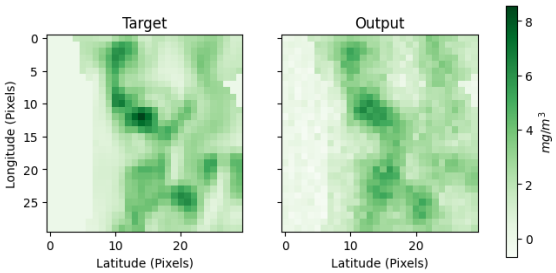
Medium Particulate Carbon [240m deep]



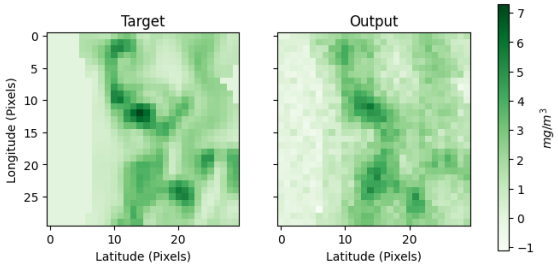
Medium Particulate Carbon [260m deep]



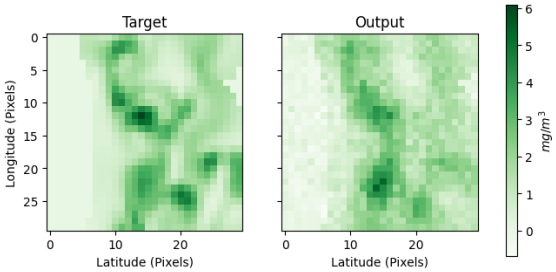
Medium Particulate Carbon [280m deep]



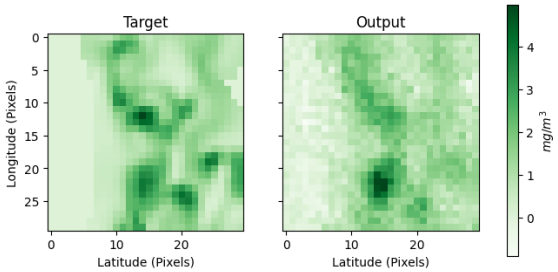
Medium Particulate Carbon [300m deep]



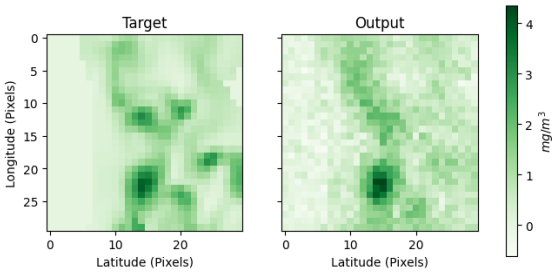
Medium Particulate Carbon [320m deep]



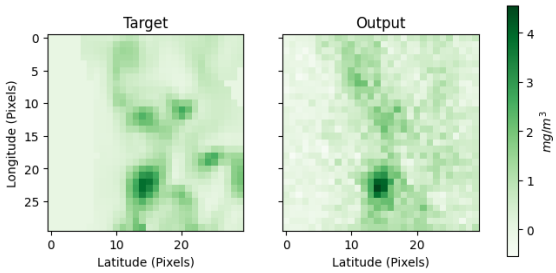
Medium Particulate Carbon [340m deep]



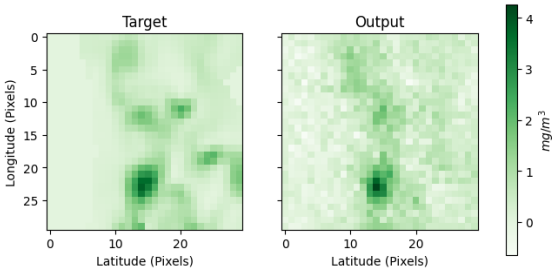
Medium Particulate Carbon [360m deep]



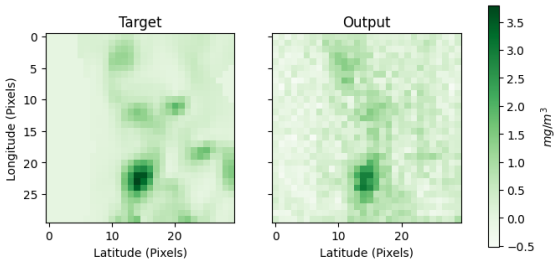
Medium Particulate Carbon [380m deep]



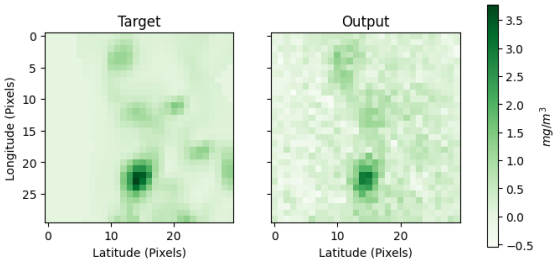
Medium Particulate Carbon [400m deep]



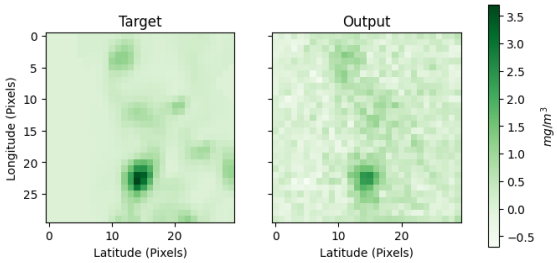
Medium Particulate Carbon [420m deep]



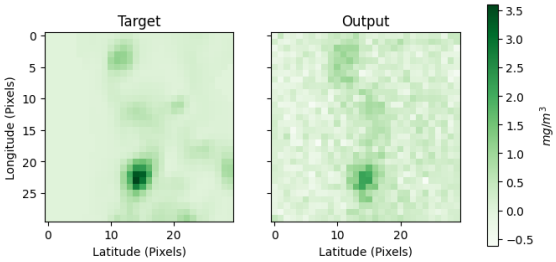
Medium Particulate Carbon [440m deep]



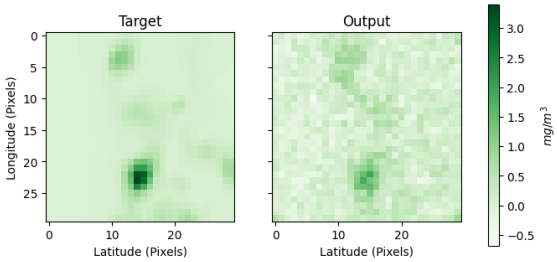
Medium Particulate Carbon [460m deep]



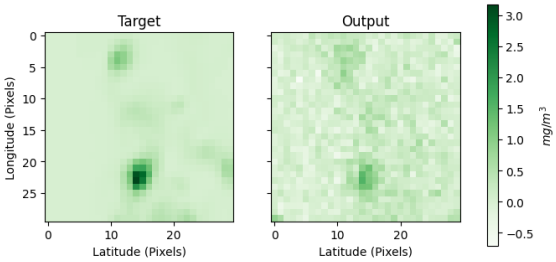
Medium Particulate Carbon [480m deep]



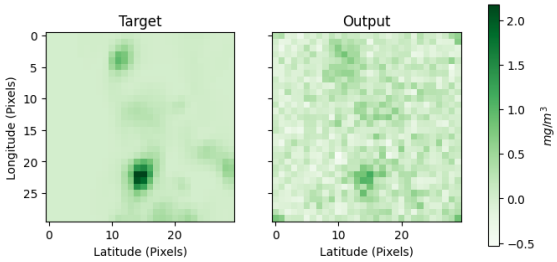
Medium Particulate Carbon [500m deep]



Medium Particulate Carbon [520m deep]



Medium Particulate Carbon [540m deep]



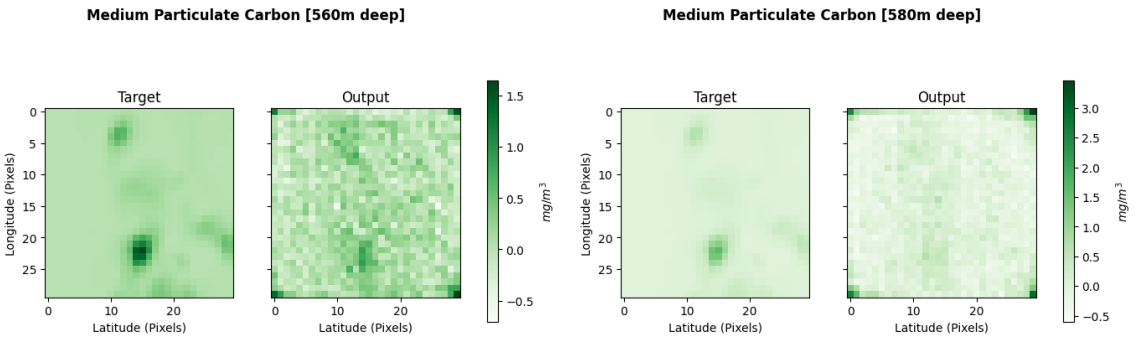
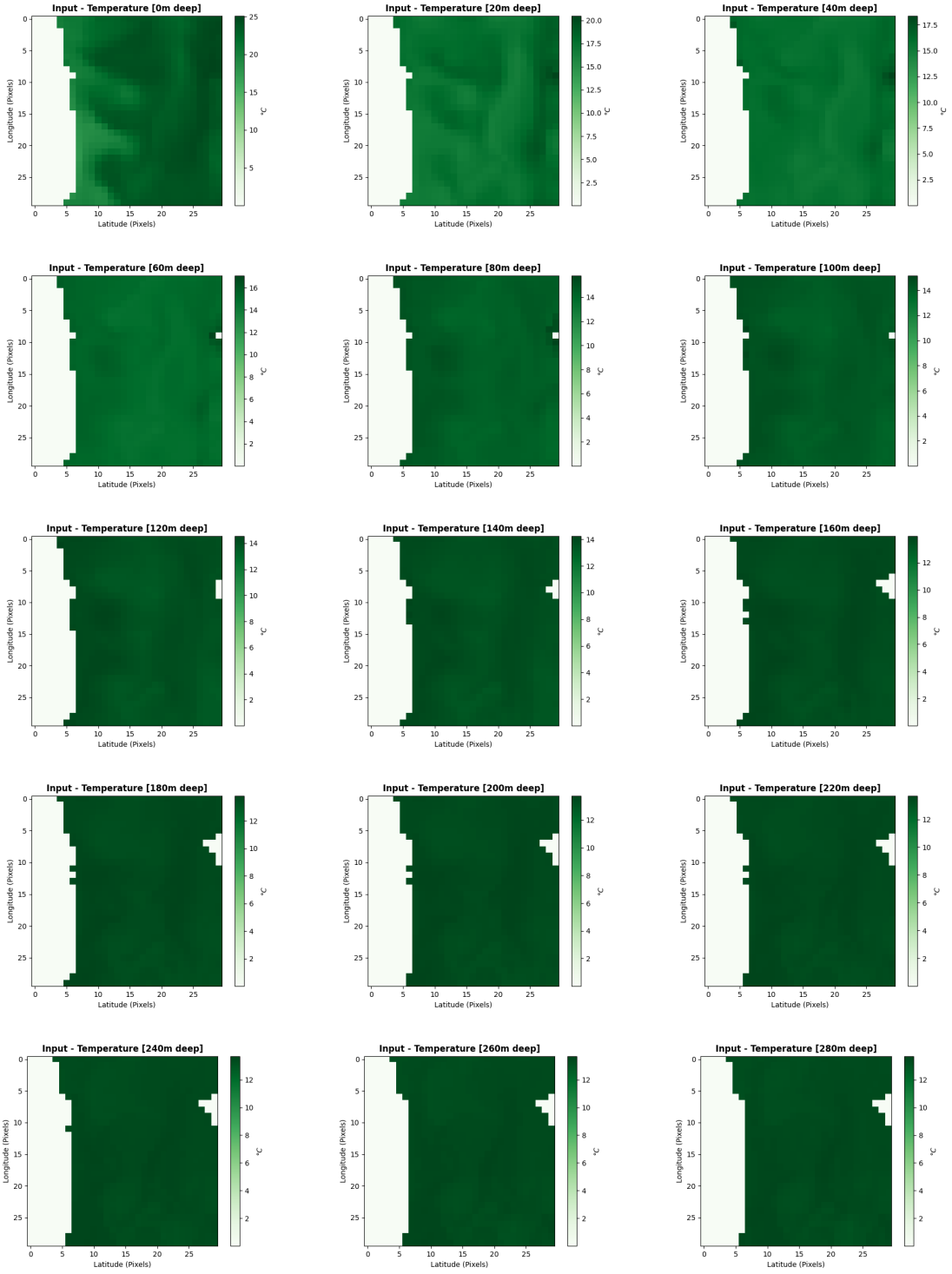


Figure II.5: Medium Particulate Carbon - Target VS. Output.

II.2 Model's Input

II.2.1 Temperature

II



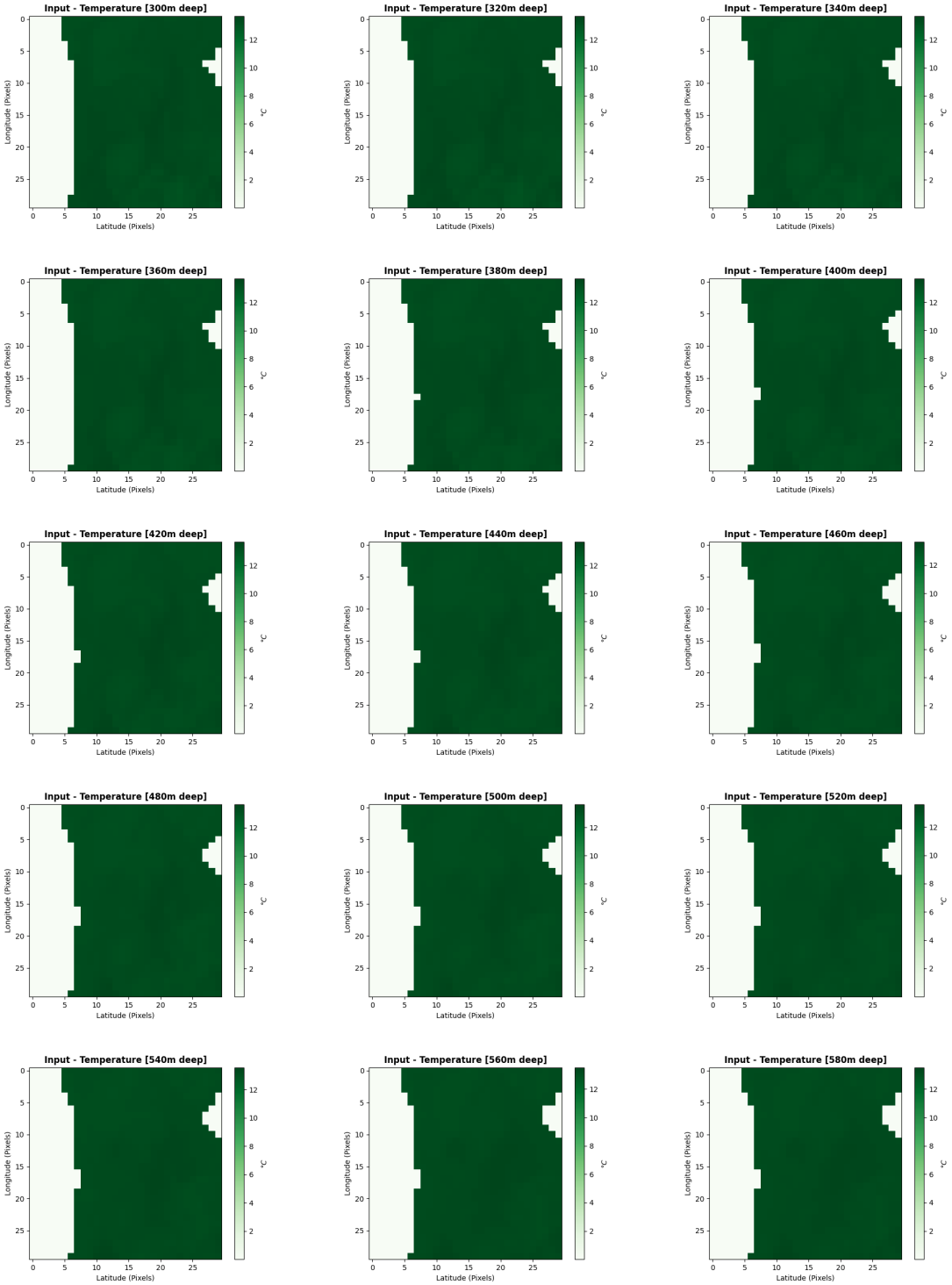
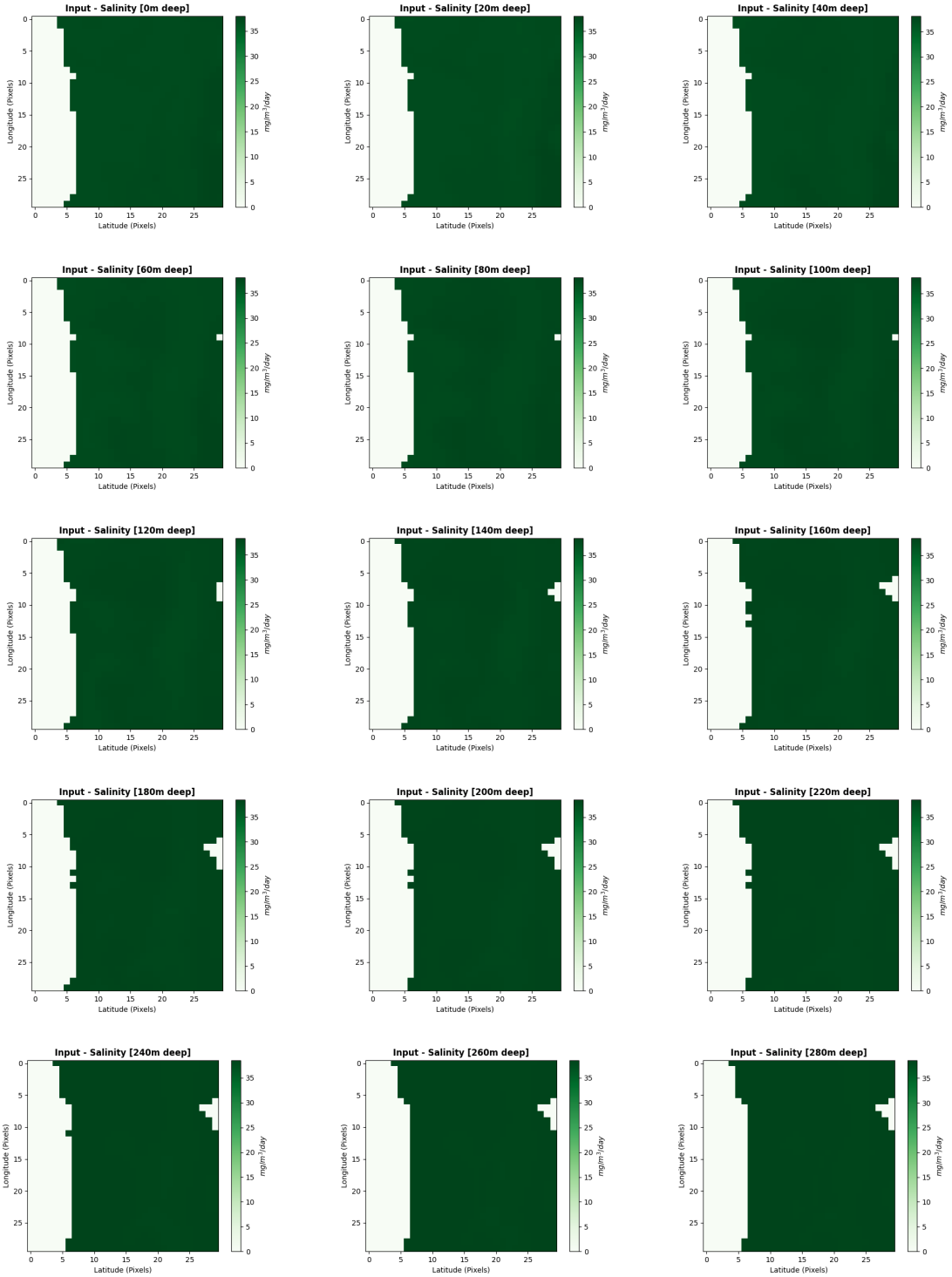


Figure II.6: Temperature - Input.

II.2.2 Salinity



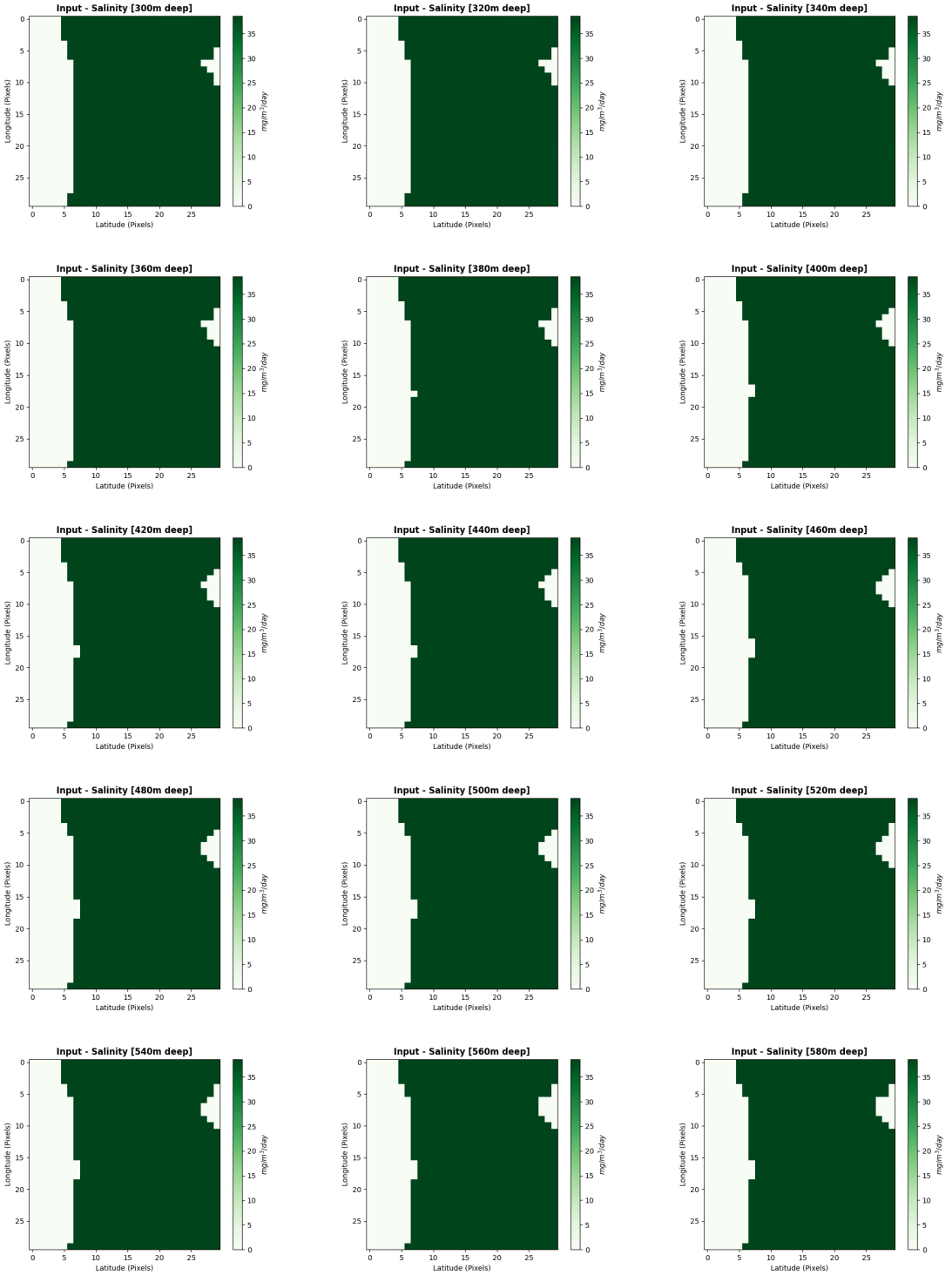
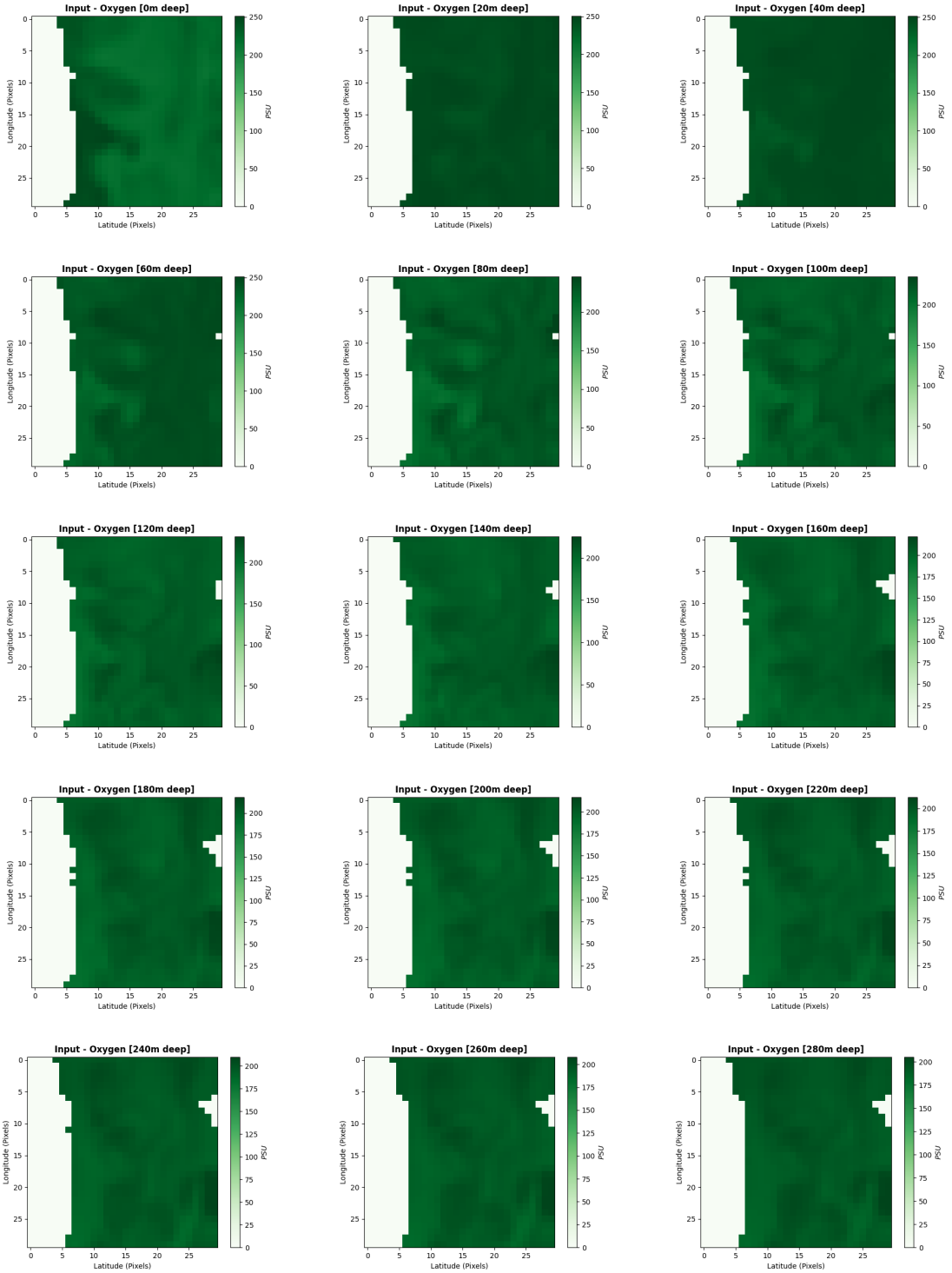


Figure II.7: Salinity - Input.

II.2.3 Oxygen


Input - Oxygen [0m deep]
Input - Oxygen [20m deep]
Input - Oxygen [40m deep]
Input - Oxygen [60m deep]
Input - Oxygen [80m deep]
Input - Oxygen [100m deep]
Input - Oxygen [120m deep]
Input - Oxygen [140m deep]
Input - Oxygen [160m deep]
Input - Oxygen [180m deep]
Input - Oxygen [200m deep]
Input - Oxygen [220m deep]
Input - Oxygen [240m deep]
Input - Oxygen [260m deep]
Input - Oxygen [280m deep]
Longitude (Pixels)
Latitude (Pixels)
PSU

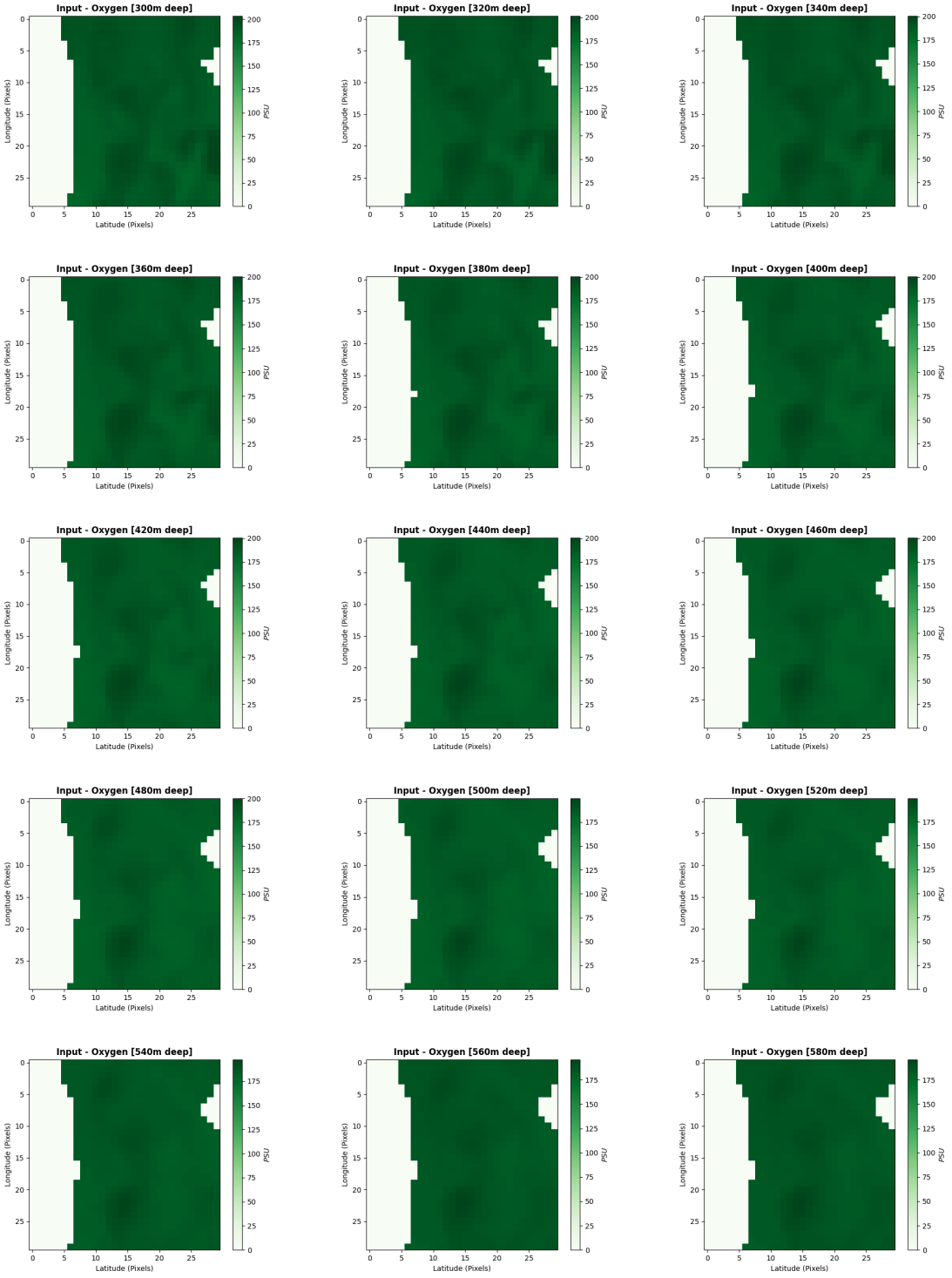
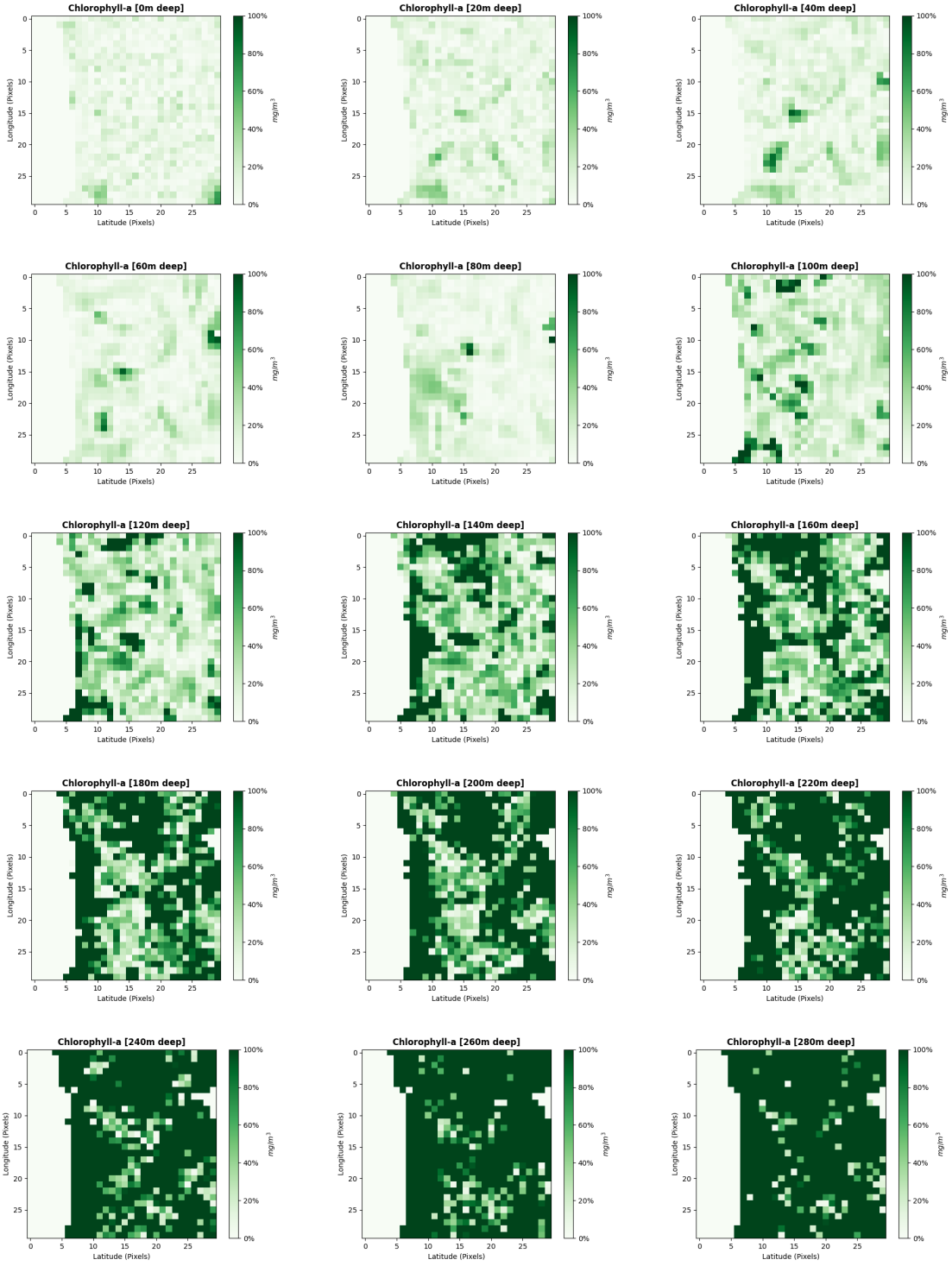


Figure II.8: Salinity - Input.

II.3 Absolute Relative Error

II.3.1 Chlorophyll-a



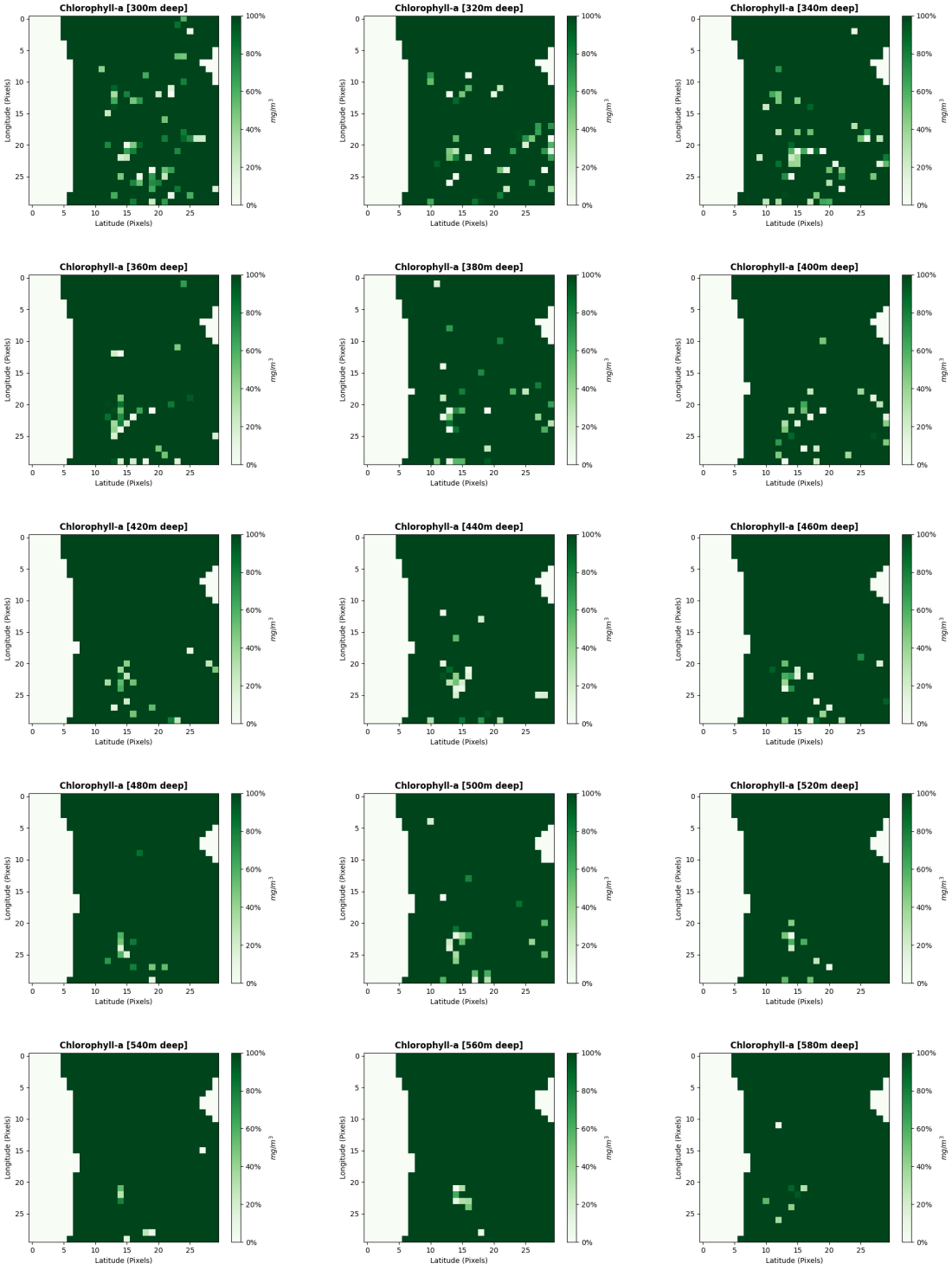
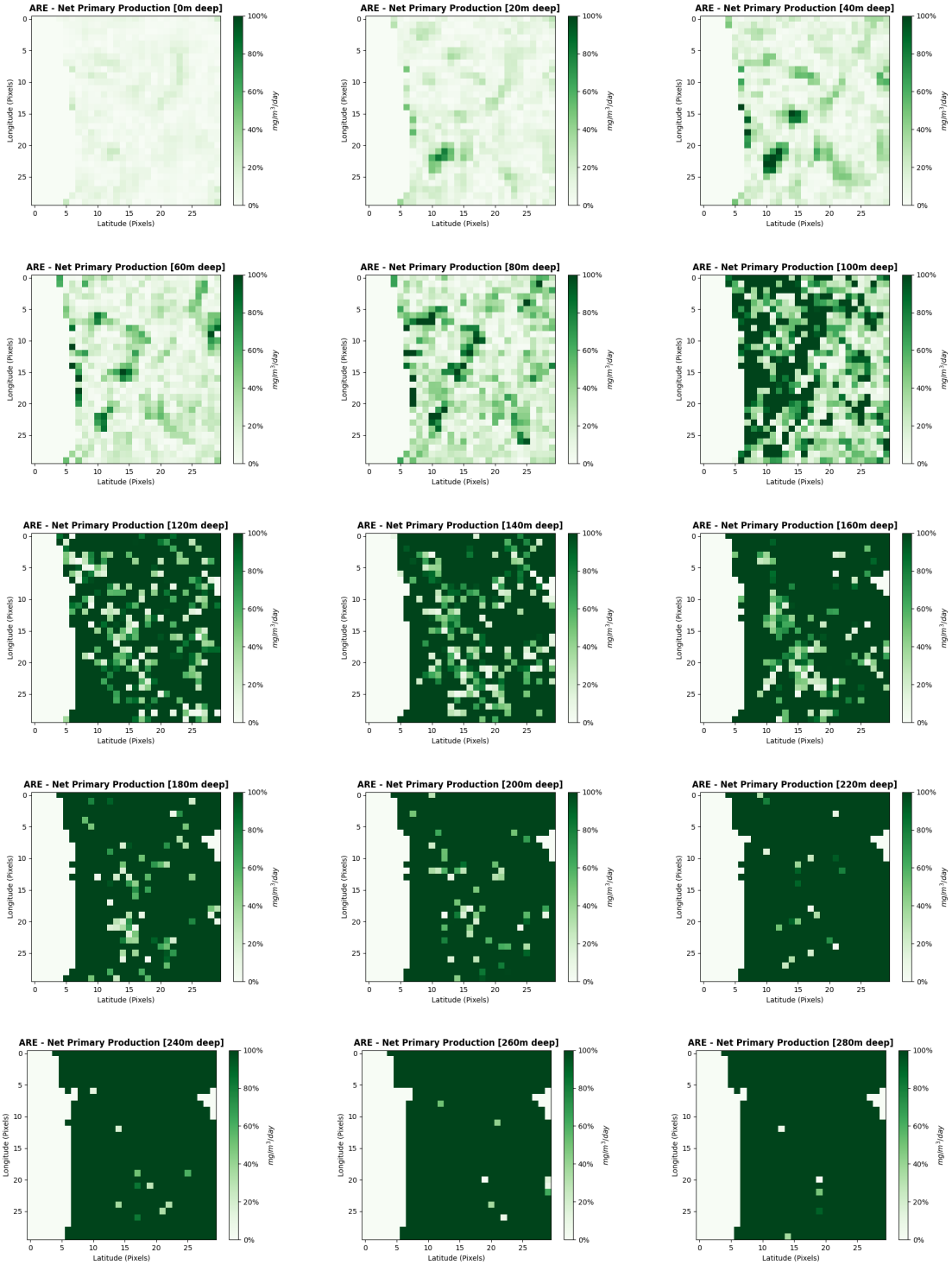


Figure II.9: Chlorophyll-a - ARE.

II.3.2 Net Primary Production



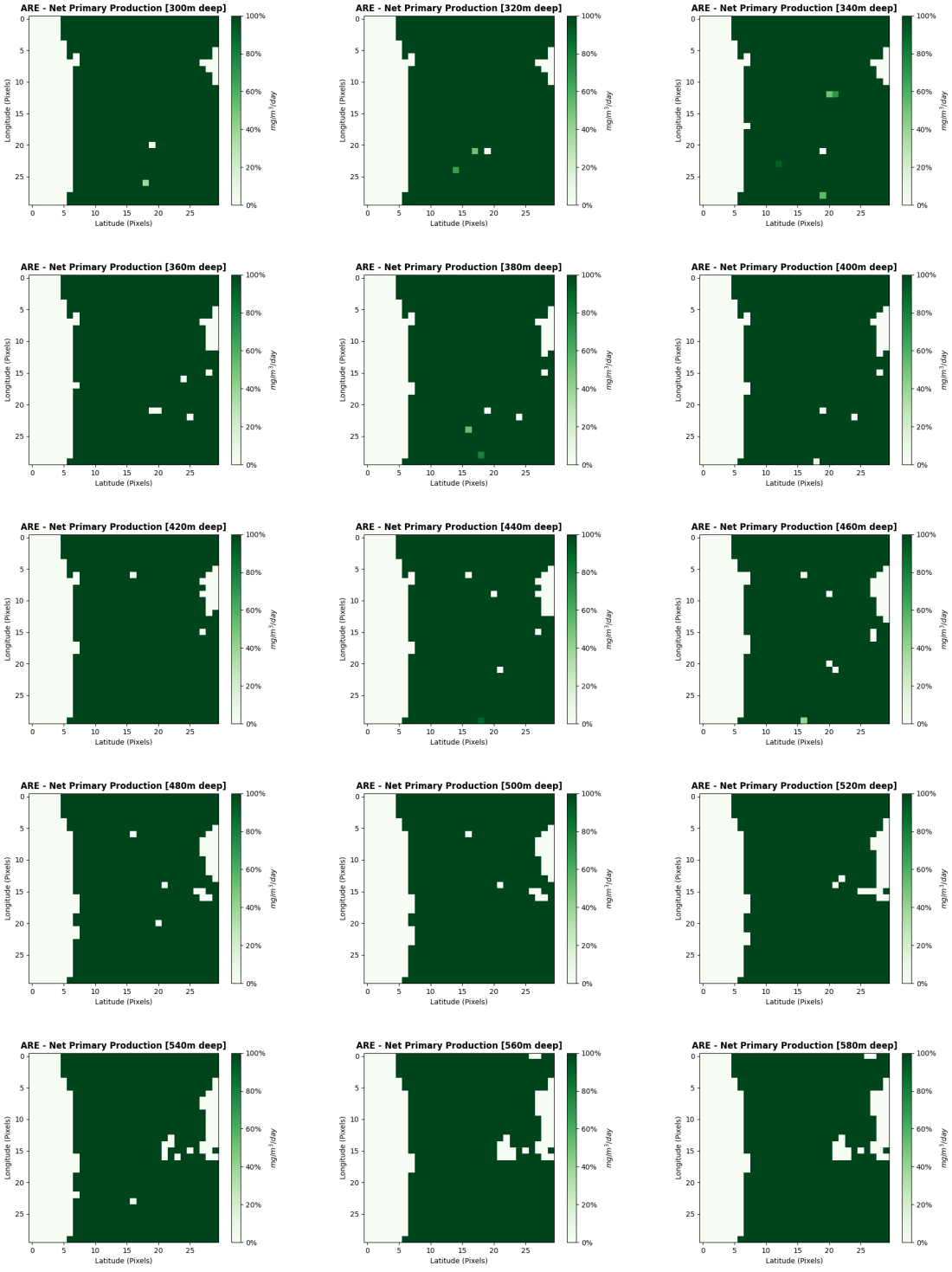
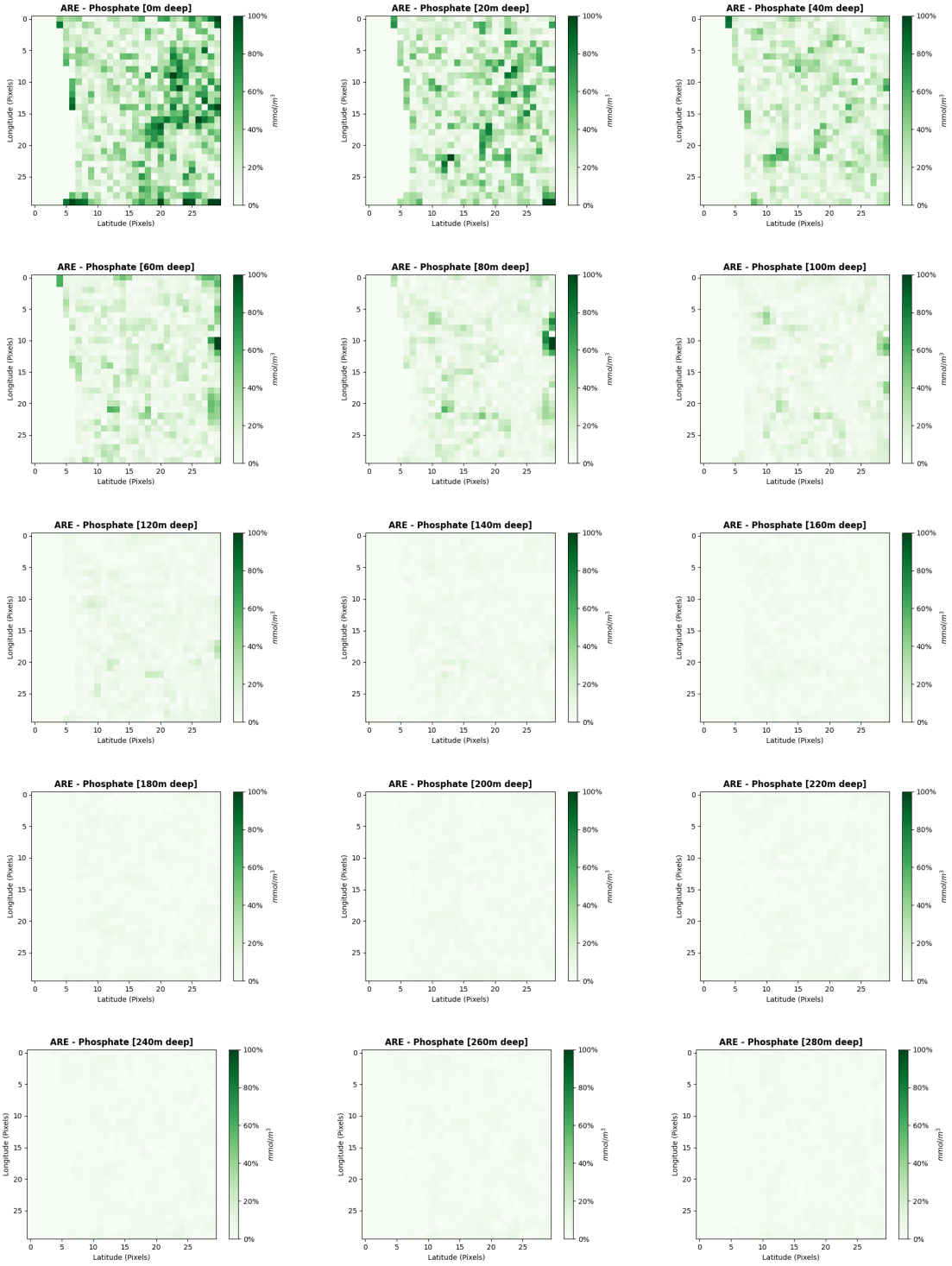


Figure II.10: Net Primary Production - ARE.

II.3.3 Phosphate



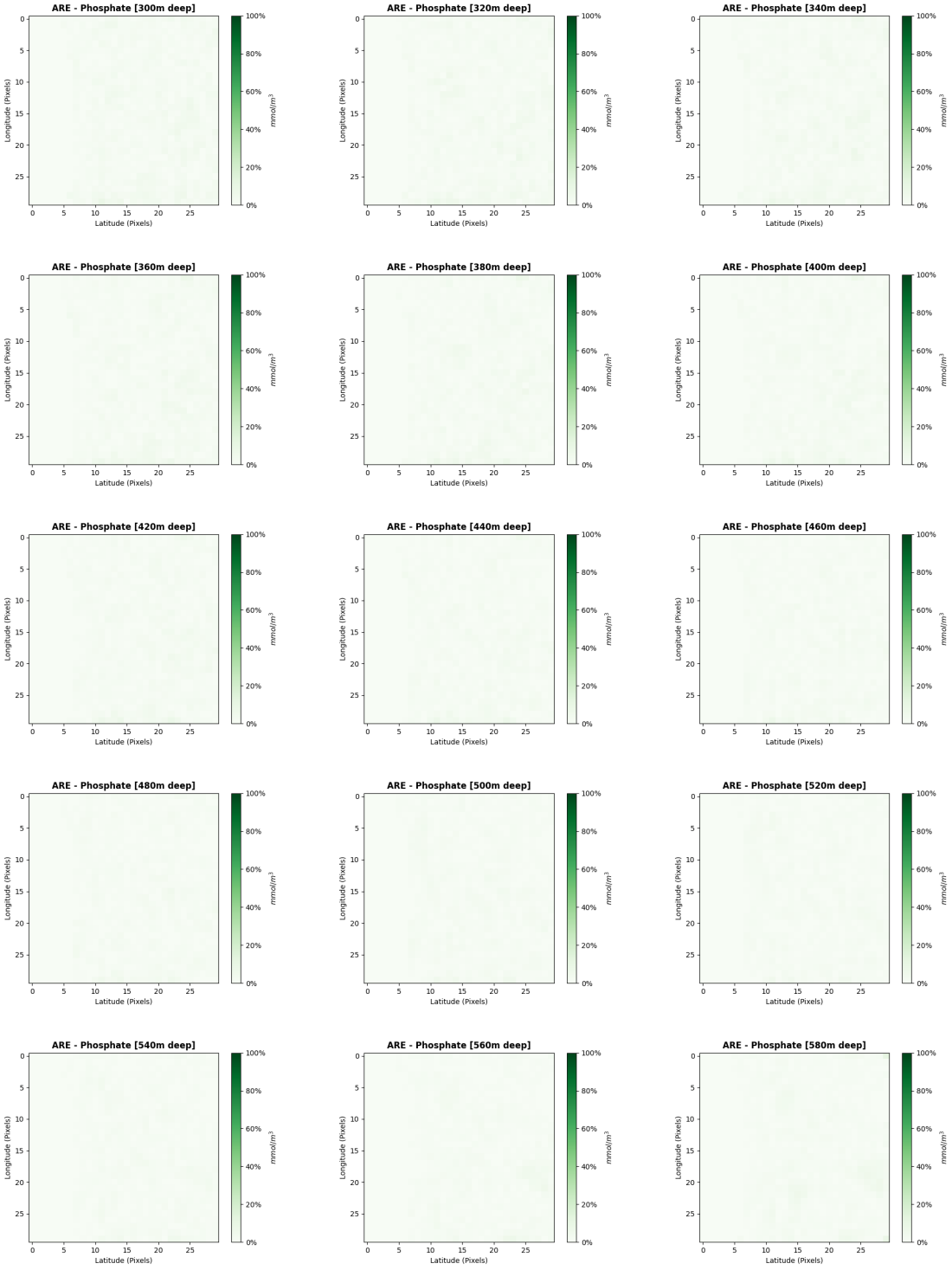
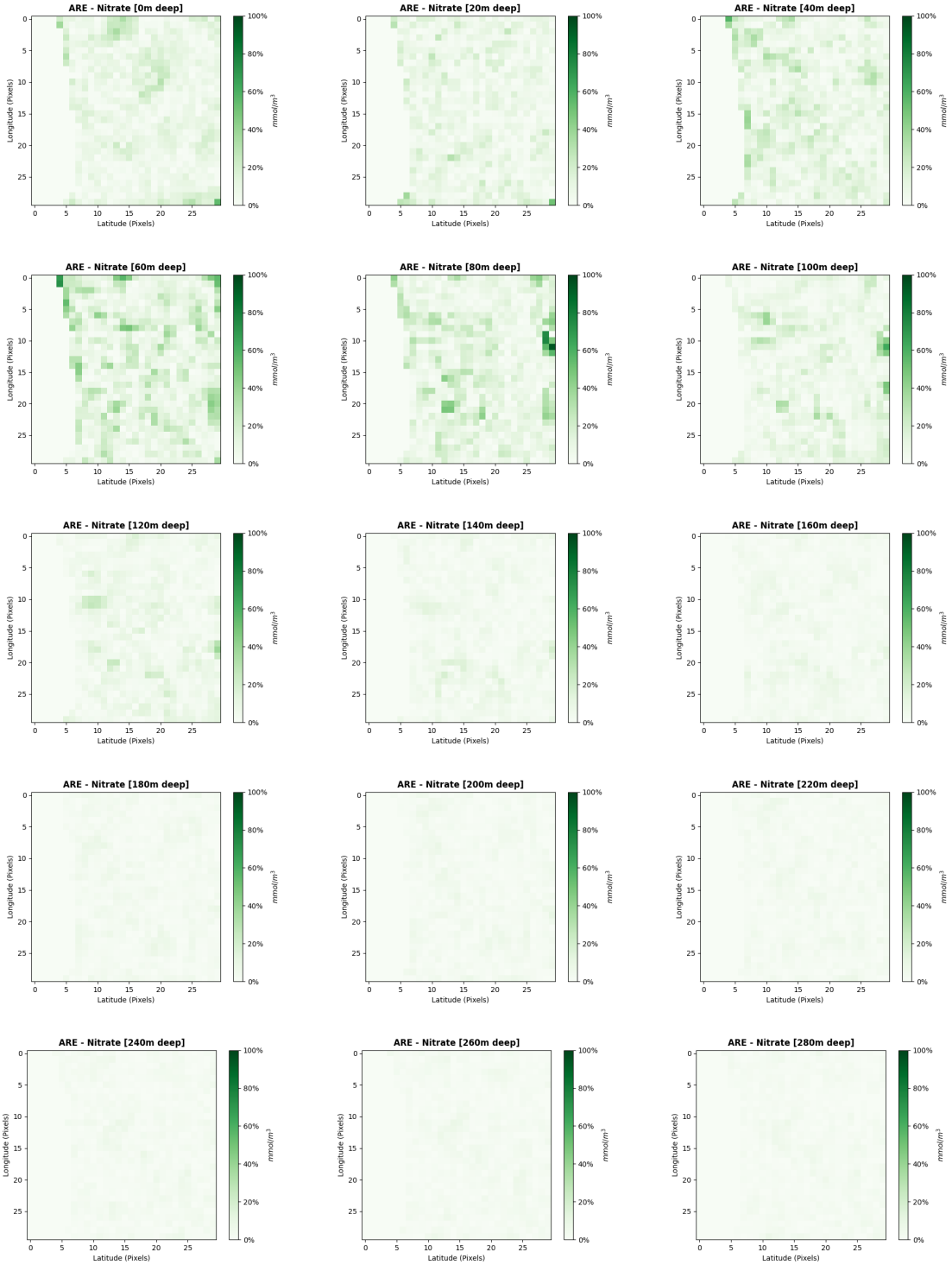


Figure II.11: Phosphate - ARE.

II.3.4 Nitrate



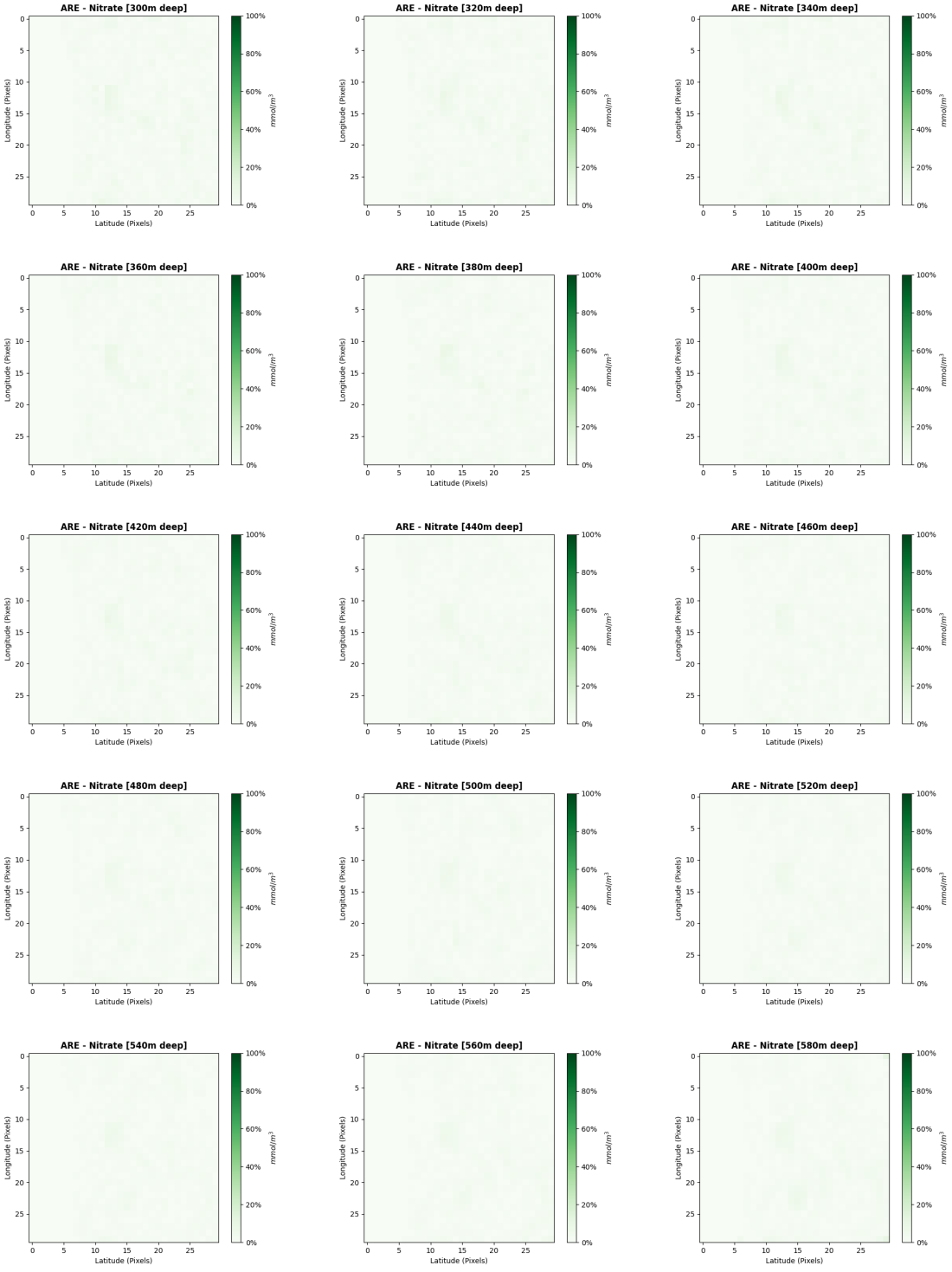
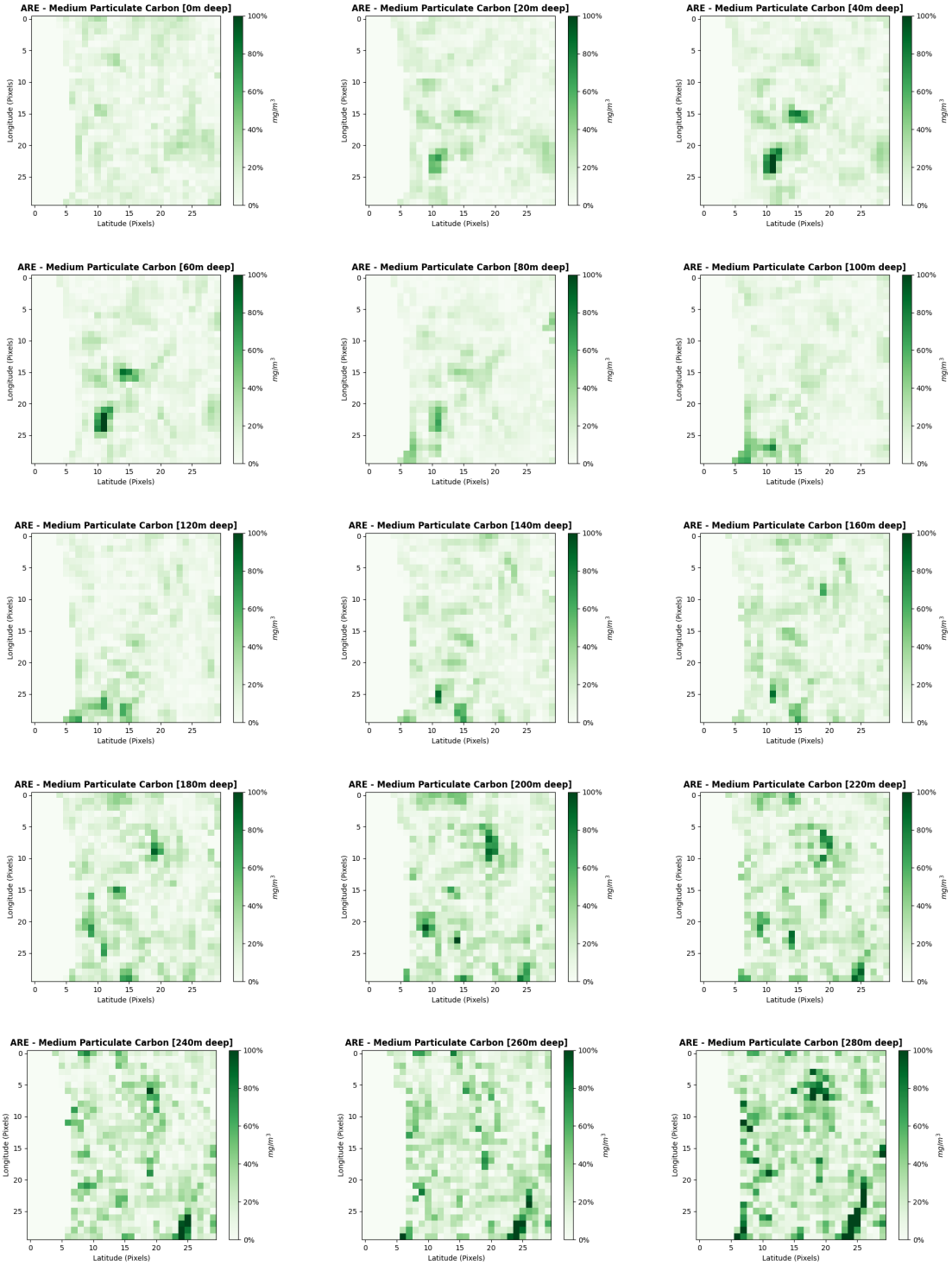


Figure II.12: Nitrate - ARE.

II.3.5 Medium Particulate Carbon



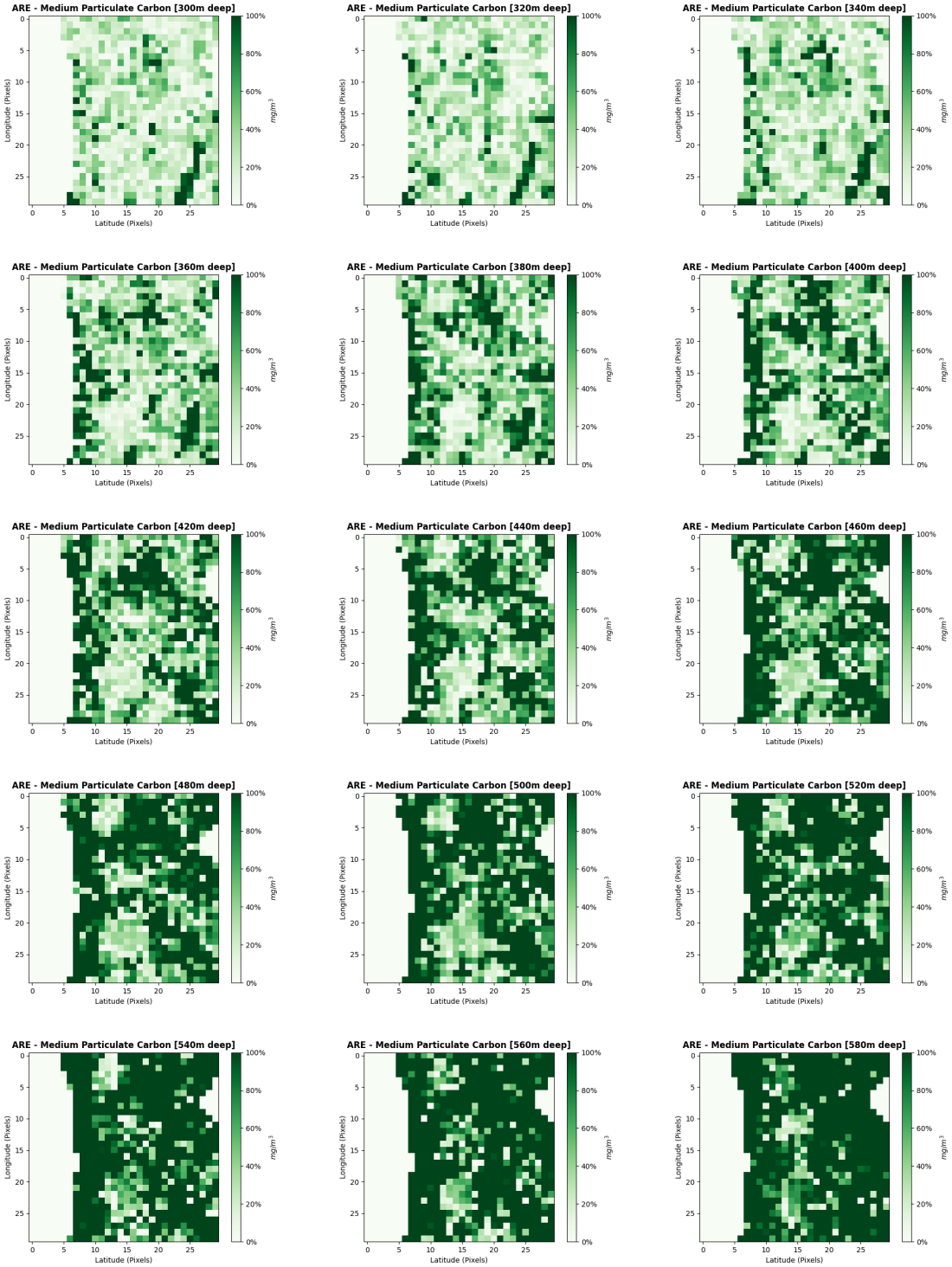


Figure II.13: Medium Particulate Carbon - ARE.