POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

SEAquence: a Transformer-based IUU Fishing Detection from Maritime AIS Data

Tesi di Laurea Magistrale in
Mobility Engineering - Ingegneria della Mobilità

Author: **Damiano Masuino**

Student ID: 103354
Advisor: Prof. Mark James Carman
Academic Year: 2022-2023

Abstract

More than 70% of the Earth's surface is covered by water, approximately 360,700,000 square kilometers, which is equivalent to 36.19 trillion soccer fields (36 followed by 12 zeroes).

Explorers, navigators, individuals in search of (new) opportunities... the oceans have always welcomed everyone and have consistently been a shared habitat with numerous other animal and plant species. For instance, there are over 250,000 different species of fish, and researchers discover new ones every day (see [24, 28]).

A shared home, an environment that is both comfortable and hostile, calm yet stormy, rigid and accommodating.

The sea is inherently a challenging territory to control, the vast expanses of water often concealed by sheer distance, creating a hospitable ground for various criminal activities. From piracy actions among Greeks and Romans to modern networks of trafficking and trading that circumvent sanctions and embargoes, the sea has consistently provided silent consent to numerous illegal opportunities.

Regulations and standards have continually sought to maintain control over these activities, but often they have proven ineffective or insufficient. A clear example is the practice of Illegal, Unreported, and Unregulated (IUU) Fishing.

In recent years, the increasing availability of data has enabled new monitoring solutions. The objective of this thesis is to analyze how the application of cutting-edge visual analysis techniques can contribute to addressing the absence of a valuable ally for maritime enforcement agencies.

The sea is as rich as it is delicate and must be constantly protected, utilizing new technologies and procedures.

Keywords: IUU, Fishing, Maritime Activities, Transformer, Vision Transformer

Abstract in lingua italiana

Più del 70% della superficie terrestre è coperta dall'acqua. 360700000 km^2, cioè l'equivalente di ben 36,19 trilioni di campi da calcio (36 seguito da 12 zeri).

Esploratori, navigatori, uomini in cerca di (nuove) opportunità... gli oceani hanno sempre accolto tutti e sono da sempre l'habitat condiviso con molte altre specie animali e vegetali. Esistono per esempio più di 250.000 diverse specie di pesci e ogni giorno i ricercatori ne scoprono di nuove (vedi [24, 28]).

Una casa condivisa, un habitat che allo stesso tempo è comodo ed ostile, tranquillo ma burrascoso, rigido ed accondiscendente.

Il mare è intrinsecamente un territorio difficile da controllare: le vaste distese d'acqua spesso nascoste dalla semplice distanza sono un terreno ospitale per molteplici attività criminali. Dalle azioni di pirateria tra Greci e Romani alle moderne reti di traffici e scambi che eludono sanzioni ed embarghi, il mare ha sempre offerto il suo tacito assenso a numerose opportunità illegali.

Le normative e gli standard hanno costantemente cercato di mantenere il controllo su queste attività, ma spesso si sono rivelate inefficaci o insufficienti. Un esempio chiaro è la pratica della Pesca Illegale, Non Dichiarata e Non Regolamentata (IUU).

Negli ultimi anni, la crescente disponibilità di dati ha reso possibili nuove soluzioni di controllo. L'obiettivo di questa tesi è di analizzare come l'applicazione di tecniche all'avanguardia di analisi visuale possa contribuire a risolvere l'assenza di un prezioso alleato per gli enti di controllo marittimi.

Il mare è tanto ricco quanto delicato e deve essere costantemente protetto avvalendosi anche dell'uso di nuove tecnologie e procedure.

Parole chiave: IUU, Pesca, Attività Marine, Transformer, Vision Transformer

Contents

Introduction

Historically, fishing has played a significant role in human civilization. Early societies relied on fishing as a primary food source, contributing to their survival and development.

Fishing has a significant economic impact worldwide. Coastal communities often rely on commercial fishing as a vital industry, providing employment opportunities and contributing to local economies.

Figure 1: Fishing boats.

Given this, this study aims to create a model that identifies fisheries-related activities by isolating them from all others (such as freight or passenger traffic, general offshore activities, etc.). This tool would be a formidable ally in the hands of supervisors, who could rely on greater automation to identify potential illicit actions.

The core data source is the AIS. AIS stands for Automatic Identification System, and it is a device on board a ship that emits a signal designed to uniquely identify the boat's position (its operation will be discussed in more detail in the next chapter).

The sea is as rich as it is delicate, and it must be constantly protected by leaning on the use of state-of-the-art technologies.

Structure of the Thesis

For the purpose of an adequate and exhaustive treatment of the problem that emerged, the thesis is organized as follows.

Chapter 1 defines and explains the background knowledge and concepts that are related to this thesis.

Chapter 2 introduces past works which tackles similar subjects.

In **Chapter 3**, all the research questions are presented and explained.

Chapter 4 presents the datasets (from the AIS network to the data used in this study).

In **Chapter 5** the actual models are described and their experimental use is implemented in the following chapter (**Chapter 6**).

Finally, **Chapter 7** concludes this report by summarizing the work, pointing out critical findings, and advising the possible future work.

1 | Background Theory

In this chapter all notions required to better understand the thesis are summarized.

1.1. Maritime Regulation & IMO

Maritime regulation refers to the set of laws, rules, and international agreements that govern activities and operations related to maritime transportation and commerce. It encompasses a wide range of regulations designed to ensure the safety, security, and environmental sustainability of maritime activities, as well as promote fair and efficient maritime trade.

Within this framework, the *International Maritime Organization* (IMO) is a specialized agency of the United Nations responsible for promoting safe and secure shipping practices on a global scale. The IMO's work covers a wide range of areas related to maritime affairs. It develops and adopts international conventions, protocols, and codes that set out mandatory rules and standards for the design, construction, equipment, operation, and maintenance of ships. These regulations cover aspects such as ship safety, pollution prevention, maritime security, crew training, and the facilitation of international maritime traffic.

1.2. Automatic Identification System

The *AIS* is a tracking and communication system implemented in the maritime domain to improve the safety and efficiency of vessel operations (see [17]).

Specific IMO regulations govern the type of ships that must mandatorily have AIS equipment on board (*International Maritime Organization (IMO), A 29/Res.1106*). Ships with a gross tonnage (GT) of more than 300 tons are generally required to have AIS on board, as are high-speed vessels such as fast ferries or high-speed passenger ships. The device is mandatory in some specific areas, such as congested waterways, ports, or environmentally sensitive areas.

The picture below shows how the AIS network works. The system was created as a means of communication to improve situational awareness. Leaning on a Very High Frequency (VHF) technology, the system emits from the device on board the ship, a signal containing a series of coded information. The time interval between one emission and the next, and the type and resolution of the information it contains, are dependent on a number of variables such as mainly the type of ship and the speed at which it is sailing.

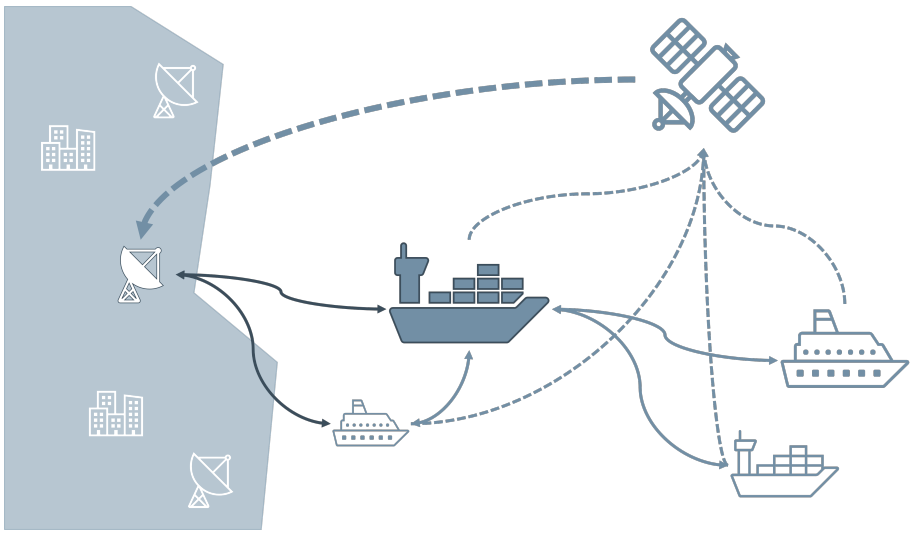

Figure 1.1: AIS Functional Scheme, picture from [27]

The system allows three different types of communication. The first is *ship-to-ship* (S2Ship); in this case, boats have the ability to transmit their own signal and receive signals from other boats, gathering information about traffic and their location (solid light blue arrows in *Figure 1.1*). The second involves satellite communication *(ship-to-satellite*, S2Sat). In this case, a special network of satellites collects data from all vessels and saves them in datasets (dashed light blue arrows in *Figure 1.1*). These data are mainly used for analysis and research purposes, and given the cost and technical difficulties associated with them, they usually are under the responsibility of private entities. The last type of transmission is *ship-to-shore*, S2Shore. These communications are used for maritime traffic management functions by coastal and port authorities. Special land-based antennas pick up the signals thus giving real-time views of ship traffic (solid dark blue arrows in *Figure 1.1*); this information can be then saved and used for future studies. Regardless of location and situation, every message transmitted by AIS contains (at least) the position information and unique identification of the vessel that transmitted it.

An in-depth analysis of both AIS system and information contained in AIS signals will be presented in Chapter 4.

1.3. Illegal, Unreported and Unregulated Fishing

Illegal, Unreported, and Unregulated (IUU) Fishing is a broad term that captures a wide variety of fishing activities (see *Figure 1.2*). As a general description, "*illegal*" fishing activities are those that are carried out in contravention of any national or international regulations (e.g., catching prohibited species, using banned equipment, etc.). "*Unreported*" fishing activities refer to those that are not properly reported or under-reported to the relevant authority. "*Unregulated*" fishing activities encompass those conducted by vessels without a flag state or operating in the gray areas of international regulations.

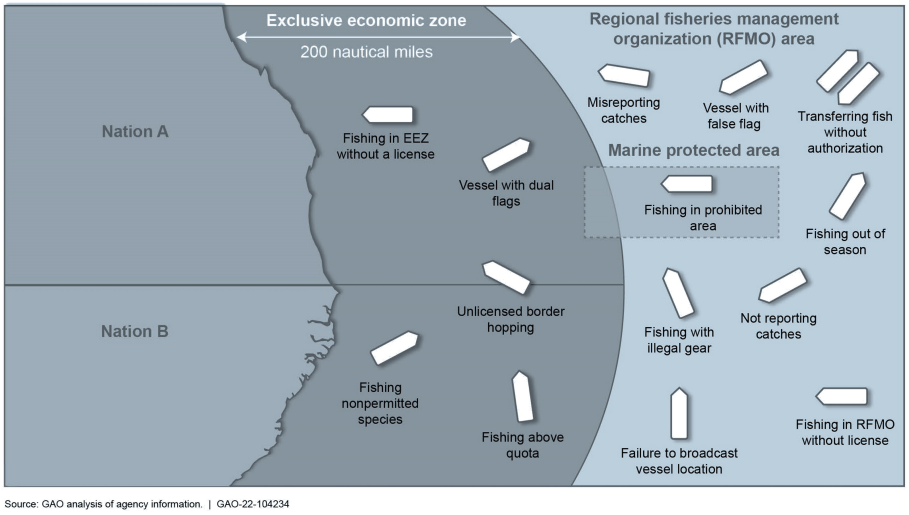

Figure 1.2: Examples of IUUF-related activities.

In a reality where interest in environmental impact is increasing, being able to quantify and control this phenomenon is of vital importance. The study focused on analyzing all three critical types of activities, aiming to create a model capable of identifying the type of activity performed by a vessel (fishing or non-fishing) solely based on the AIS data transmitted by the vessel itself.

1.4. Machine Learning

In this section, a short review of the general notion of Machine Learning is made in order to remind basic concepts.

Machine learning is a branch of artificial intelligence (AI) that involves the development of algorithms and models that allow computer systems to learn from data and make predictions or decisions without being explicitly programmed. It enables computers to automatically analyze and interpret complex patterns and relationships in data, leading to intelligent insights and actions.

At its core, machine learning revolves around the concept of training models on historical data to recognize patterns and make accurate predictions or decisions when faced with new, unseen data. This training process involves exposing the model to a large dataset, where it learns from the features of the data and corresponding outcomes. The model then generalizes this learning to make predictions on new, unseen data. To do that, several approaches are used, such as:

- *Supervised Learning*: in supervised learning, the model is trained on labeled data, where each data instance has corresponding input features and known output labels. The model learns to map the input features to the output labels, enabling it to make predictions or classifications on new, unseen data.
- *Unsupervised Learning*: unsupervised learning involves training models on unlabeled data, where the model learns patterns, structures, or relationships within the data without explicit output labels. It aims to discover inherent patterns or groupings in the data, making it useful for tasks such as clustering, dimensional reduction, and anomaly detection.
- *Reinforcement Learning*: reinforcement learning involves an agent learning to make sequential decisions in an environment to maximize a reward signal. The agent explores the environment, takes action, and receives feedback in the form of rewards or penalties. Through trial and error, the agent learns to optimize its decision-making strategy to achieve the highest cumulative reward.

In this thesis, a supervised (or as will be presented, a *self-supervised*) classification model will be designed and tested.

Machine learning has found applications in various fields, including image and speech recognition, natural language processing (NLP), recommendation systems, fraud detection, financial forecasting, autonomous vehicles, and healthcare, among others. In 2017, a specific family of ML models called Transformers was introduced to manage NLP data. This kind of model is the core of this study and in the following sub-chapter, a more in-depth presentation regarding them will be proposed.

1.4.1. Transformer Architecture

Transformers are a type of deep learning model that has revolutionized the field of natural language processing (NLP) and made significant advancements in other domains as well. The main structure behind them (see *Figure 1.3*) was presented in the 2017 paper *Attention Is All You Need* (see [25]) and it shortly became the state-of-the-art approach for various NLP tasks.

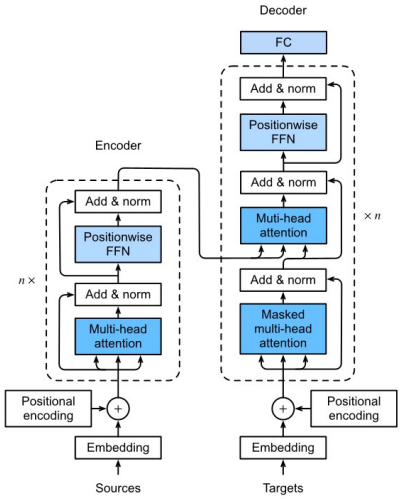

Figure 1.3: Transformer architecture.

The transformer architecture consists of two main components: an encoder and a decoder. The encoder takes an input sequence and processes it, creating a rich representation of the input data. The decoder then uses this representation to generate the desired output sequence. Both encoders and decoders use the concept of 'attention'. Attention basically means to focus on the important pieces of information and to blend out the unimportant pieces.

Specifically, the key innovation of transformers is their ability to capture long-range dependencies and contextual relationships in sequential data, such as sentences or sequences of words or pictures. Unlike traditional recurrent neural networks (RNNs) that process sequences step-by-step, transformers utilize self-attention, which allows the model to weigh the importance of different words or tokens within a sequence and attend to relevant information when making predictions or generating output. It is possible to say that transformers have introduced the concept of "self-supervised learning".

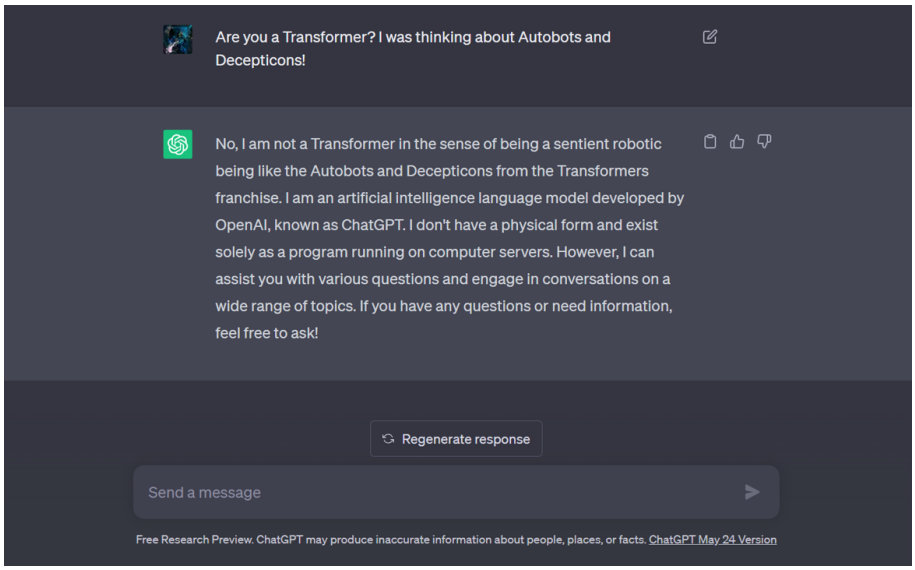

Figure 1.4: OpenAI ChatGPT is nowadays one of the most famous examples of Transformers' potential.

The Transformer is the first transduction model relying solely on self-attention to compute representations of its input and output without using sequence-aligned Recurrent neural networks (RNNs) or convolutions. A description of how the attention mechanism works can be done with the following example. Given an input sequence x of length n (Equation (1.1)) and a target sequence y of length m (Equation (1.2)) in an NMT problem:

$$x = [x_1, x_2, ..., x_n], \tag{1.1}$$

$$y = [y_1, y_2, ..., y_m] \tag{1.2}$$

A NMT architecture such as the one shown in *Figure 1.3*, it is possible to denote as (1.3)

$$h_i i = 1, 2, 3, ..., n \tag{1.3}$$

the *encoder hidden states*, which contains information from all the words forming the input sentence with a strong focus on the $i-th$ word among the n words in the input.

The decoder employs an attention mechanism to generate the target words, prioritizing the most pertinent information extracted from the source sentence. Con-

sequently, the decoder processes every hidden encoder state (annotation) and, in conjunction with the previous hidden decoder state (s_{t-1}), utilizes an alignment model, $a()$, to calculate an attention score:

$$e_{t,i} = a(s_{t-1}, h_i) = v_a^T \tanh(W_a[s_t; h_i]) \tag{1.4}$$

$e_{t,i}$ represents an attention score, serving as a metric to evaluate how well h_i and s_{t-1} align with each other. To achieve this alignment, it has been utilized an additional operation known as "additive attention."

This process involves applying a weight matrix to the concatenated vectors s_{t-1} and h_i, wherein v^T acts as the weight vector. This approach facilitates an effective and intuitive way of combining the relevant information from both s_{t-1} and h_i during the attention mechanism.

To convert the annotation values into a range from 0 to 1, a *Softmax* function (see (1.5)) on each attention score is used, which gives the corresponding weight value. This ensures that the weights are appropriately normalized and can be used to emphasize the most relevant information during the attention mechanism.

$$\alpha_{t,i} = Softmax(e_{t,i}) = \frac{\exp(a(s_{t-1}, h_i))}{\sum_{i'=1}^{n} \exp(a(s_{t-1}, h_i))} \tag{1.5}$$

where $\alpha_{t,i}$, measures the alignment between the target word y_t and the source word x_i.

After obtaining these scores (from equations (1.4) and (1.5)), it is possible to proceed to compute the context vector, c_t. The context vector combines the relevant information from the source sentence by utilizing the alignment scores, $\alpha_{t,i}$. This helps in emphasizing the most important parts of the source sentence for generating the target word.

$$c_t = \sum_{i=1}^{n} \alpha_{t,i} h_i \tag{1.6}$$

Now, moving to the decoder stage, the context vector c_t is fed into the decoder along with the previous hidden decoder state s_{t-1} and the previous output y_{t-1}. These inputs are used to calculate the final output, s_t, using a function f that takes into account the context, the previous decoder state, and the previous output.

This entire process is repeated iteratively until reaches the end of the sequence. During each iteration, the decoder generates a new output, and the context vector ensures that the decoder can access relevant information from the source sentence at every step, aiding in the generation of accurate and contextually appropriate target words.

Self-attention is a particular kind of attention mechanism.

Unlike conventional additional attention, which focuses on aligning input and output positions in a sequence, self-attention captures dependencies between different elements of the same input. This unique approach allows the model to consider the context and relationships between all elements, thereby enabling a more comprehensive understanding of the input data.

1.4.2. Vision Transformer Architecture

The main actor of this thesis is a Transformer model of a specific kind: the Vision Transformer (ViT). Traditionally, convolutional neural networks (CNNs) have been the dominant architecture for image analysis.

However, ViTs offer an alternative approach that eliminates the need for handcrafted hierarchical features and enables end-to-end learning.

The core idea behind ViTs was presented in *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale* (see [5]), and it is to treat an image as a sequence of patches, where each patch represents a small region of the image.

These patches are flattened and then fed into the transformer architecture, allowing the model to capture both local and global dependencies within the image. In ViTs, a typical architecture (as shown in *Figure 1.5*) consists of an initial patch embedding layer that converts the image into a sequence of flattened patches.

Patches are then passed through multiple transformer encoder layers, each incorporating self-attention mechanisms to capture relationships between different patches.

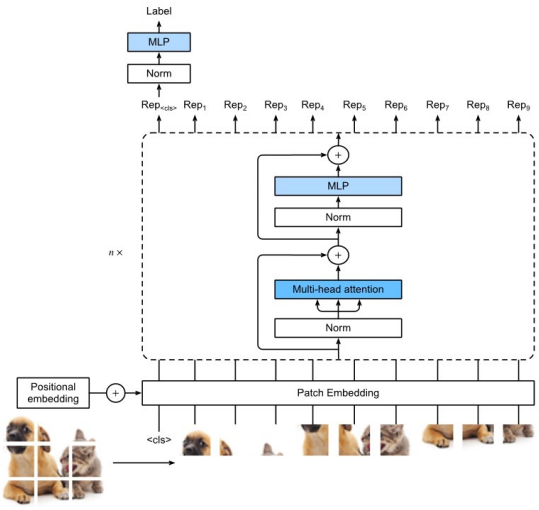

Figure 1.5: Vision Transformer (ViT) architecture.

Finally, a classification head is attached to the model to make predictions based on the learned features.

The main steps of a ViT model can be described as:

1. Split the input image into fixed-size patches
2. Flatten the patches
3. Create lower-dimensional linear embeddings from these flattened image patches
4. Include positional embeddings
5. Feed the sequence as an input to a Transformer encoder
6. Pre-train the ViT model with image labels
7. Fine-tune on the downstream dataset for image classification.

In the ViT architecture, there is no decoder part; only the encoder blocks are employed. At the top of the stack of encoders, an additional linear layer known as the MLP head is added to perform the final classification task. Vision Transformer takes a unique approach by treating image patches as if they were tokens in natural language processing (NLP) tasks.

However, one of the drawbacks of ViT is its dependence on large-scale pre-training on abundant datasets, followed by fine-tuning to achieve state-of-the-art results.

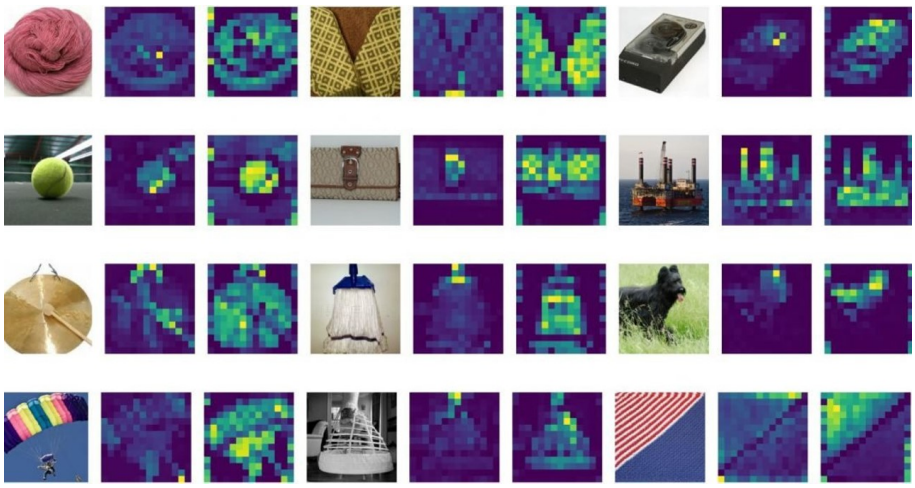

Figure 1.6: Attention masks over analyzed pictures.

In contrast, when dealing with smaller datasets directly, ViT's performance tends to be slightly inferior to traditional convolutional neural networks (CNNs).

This discrepancy arises due to the inherent differences between Transformers and CNNs. Transformers lack certain inductive biases, such as translation invariance and locality, which are essential for generalizing effectively when trained with limited data.

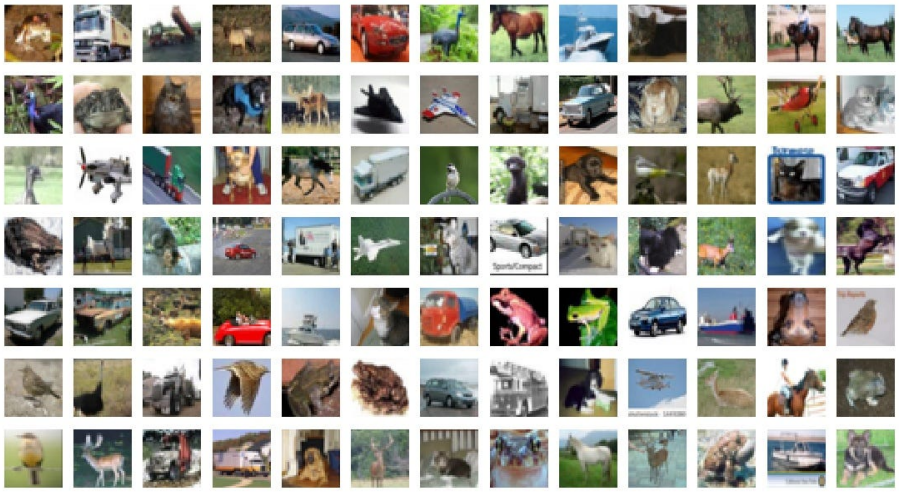

Figure 1.7: Example images in the CIFAR100 training dataset.

On the other hand, ViT excels when trained on vast datasets, outperforming many state-of-the-art architectures in various image recognition benchmarks. This phenomenon can be attributed to the remarkable power of large-scale training, where

the absence of certain inductive biases becomes less of a hindrance, enabling the model to learn intricate patterns and relationships in the data.

The trade-off lies in ViT's requirement for considerable computational resources and data availability to unleash its full potential. Nonetheless, as technology advances and access to extensive datasets become more accessible, ViT's capabilities are likely to continue pushing the boundaries of image recognition and deep learning as a whole.

By pre-training the ViT model on relevant images and then fine-tuning it on the dataset under investigation, the study will showcase the model's ability to adapt and generalize to different tasks. This highlights the potential of Vision Transformers to excel in diverse scenarios, making them a promising choice for a wide range of image-related applications. The findings from this research could provide valuable insights into harnessing the power of ViT to its full extent and unlocking its capabilities in various practical settings.

2 | Related Works

Regarding the topic addressed in this thesis, there is a significant amount of academic material available. This chapter aims to present the relevant literature that helps establish a starting point for the subsequent development of the model. For better comprehension, the material will be divided into two sub-chapters: the first will cover the literature related to the study of maritime activities, while the second will focus specifically on the use of transformers models.

2.1. State of The Art - *Maritime Activities*

There is a rich body of research on analyzing behaviors at sea, such as activity classification or prediction of naval trajectories. In [23] it is possible to find an example of classification performed by extrapolating geometric features from trajectories. In other words, each naval trajectory, represented as a sequence of points (see *Figure 2.1*), has been studied to extract possible descriptive features (e.g., stop times, the average duration of stops, small, medium, and large angle turn counts, etc.). These features have then been used to classify naval activities, considering only cargo ships or fishing vessels (see *Figure 2.2*). The core idea behind the paper is described in [1], [4] and [7].

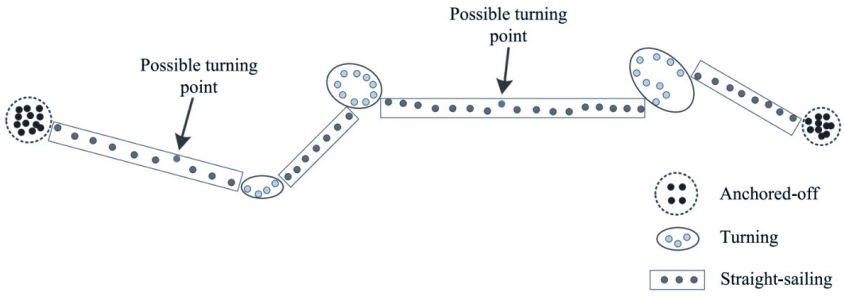

Figure 2.1: Basic movement patterns of ship trajectory, *image from [23].*

For classification, algorithms such as Decision Trees and Support Vector Machines

have been used. A similar study was conducted in the paper [21].

Stochastic approaches were followed in the papers [10], [22], [6] and [11].

Another interesting approach can be found in the paper [12], where the trajectory itself is not considered as a geometric entity, but rather the focus is placed on the spatial arrangement of the vessels (see *Figure 2.2*).

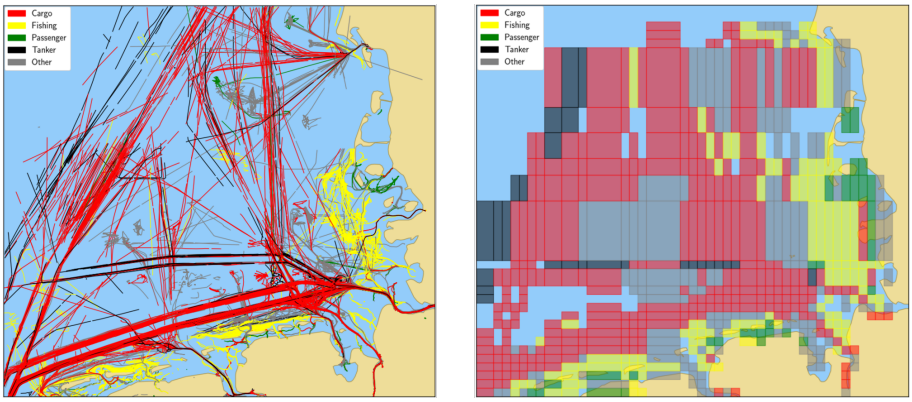

Figure 2.2: Trajectories of different vessel types (left image) and the result of trajectory clustering (right image), *image from [12].*

Geographical areas have been associated with the presence of a particular type of vessel. The classification is done by spatializing the vessel and incorporating additional information. For example, if a large-sized vessel (additional information required) is located in the area associated with high tanker traffic (the black region on the map), then the model classifies it as a tanker. Although the results are encouraging, this model requires detailed data such as the tonnage of the vessel.

The last study mentioned is [30]. In this analysis, (developed from [26], [20] and [3]) the trajectory is still regarded as a geometric entity. Various quantities are associated with each trajectory, such as latitude range, length, etc. The researchers conducted a study using a massive amount of data, extracting typical values of the aforementioned features for each studied vessel type.

The classification task involves linking each new trajectory under study to the most similar class. The performance of this model is very good, although it relies on a large amount of data that is difficult to obtain, and handle, and therefore less practical from an operational perspective.

Figure 2.3: Arctic Prowdler, a vessel whose behavior was classified as "anomalous" by the model in [30].

2.2. State of The Art - *Transformer Applications*

In relation to transformer architecture, some literature has already been presented in the previous chapter. There, a brief discussion was made about what a transformer is and what principles differentiate it from any previous deep learning model. As mentioned, transformers were initially developed as a model for natural language processing, and for this reason, they exhibit high performance when it comes to handling sequential inputs and outputs.

There have been multiple cases where attempts have been made to extend the use of transformers to different domains.

One such case is described in [32]. The aim of this study was to use a slightly more sophisticated version of transformers, called Universal Transformers (UT), to identify anomalies among the trajectories of different taxis. The Universal Transformer (UT) is an extension of the original transformer architecture that allows for dynamic computation during each decoding step. Unlike the standard transformer, where the number of decoding steps is fixed, the UT introduces recurrence, enabling it to adaptively adjust the number of computation steps based on the input. This dynamic nature makes the UT more flexible and capable of handling various sequence lengths. After a preprocessing phase ([31] and [29]), each spatial trajectory was discretized and analyzed as a sequence of points. The sequence was used as input to the UT (see *Figure 2.4*).

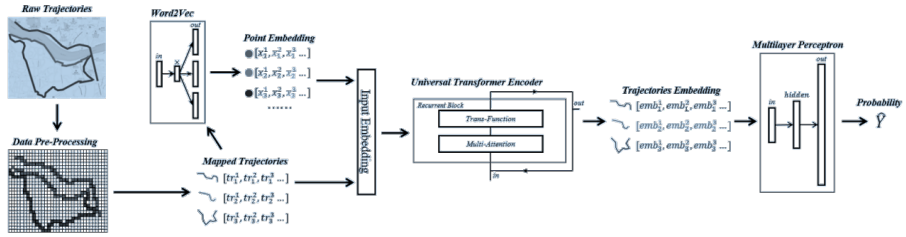

Figure 2.4: The Architecture of UT used in the paper *image from [32].*

This process is very similar to the one proposed by [19]. However, instead of using transformers, they utilize LSTM and RNN neural network models. Some notions

Another example of transformers used in the trajectory domain is [9]. In this case, the trajectory, seen as a set of Cartesian coordinates, is embedded into a high-dimensional space through a linear projection using a weighted matrix. Then, a positional encoding vector is considered to associate a time component with the input embedding vector before using it as initial information for the "vanilla" transformer. [2] and [13] propose similar approaches, both based on analyzing the trajectory data by reflecting it over a high-dimension space.

There are also examples of using transformers in the field of image analysis. [16] describes the use of a Vision Transformer (ViT) to efficiently analyze photos of components during the production line, with the goal of identifying possible anomalies or defects (see *Figure 2.5*). The input picture is divided into pieces and encoded using a Vision Transformer, and the results are then fed into a decoder to reconstruct the original image, which helps the network learn representative features.

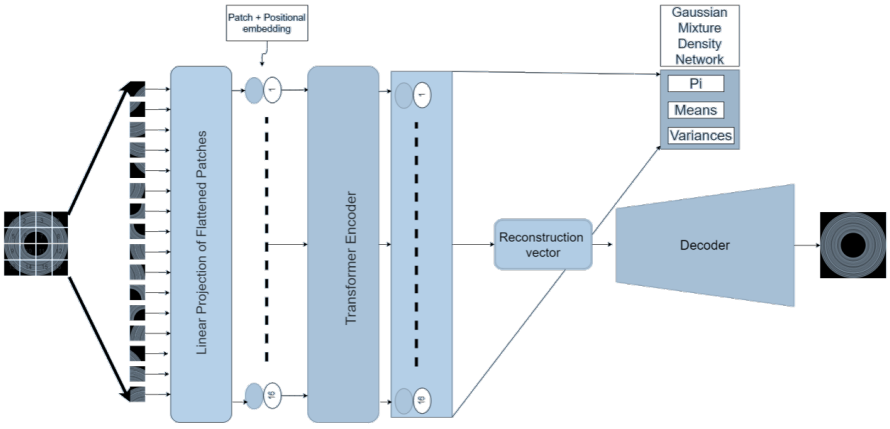

Figure 2.5: The Architecture of ViT used in the paper *image from [16].*

Simultaneously, a Gaussian mixture density network models the distribution of the transformer-encoded features to estimate the distribution of normal data in the latent space. This approach allows for effective anomaly detection based on deviations from the estimated normal data distribution.

The last example is not directly related to the use of transformers but is connected to the theme of this thesis. [8] proposes an innovative method (based on [14], [15], and [18]) for trajectory classification that is not based on the sequence of coordinates but on the creation of representative images, which are then used as input to a CNN model. The trajectories are spatially discretized, transforming them into pixels of a 2-dimensional image (see *Figure 2.6*).

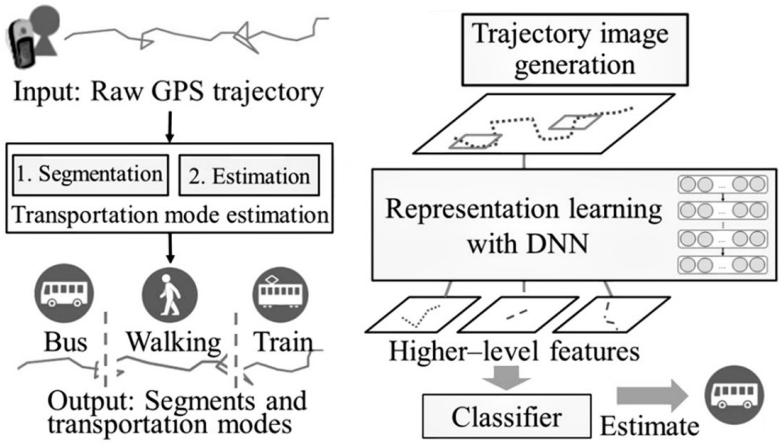

Figure 2.6: The scheme of the process used in the paper *image from [8].*

The study analyzes trajectories found in a city and aims to classify them based on the mode of transportation (such as bicycles, cars, subways, etc.). This paper can be seen as the missing link among all the others. So, why not use Vision Transformers (ViT) to study naval trajectories in a similar way to what is presented in this paper?

This question, along with the points listed above, serves as the starting point for this study.

3 | Research Questions

The purpose behind this thesis is the search for an answer to the following questions:

What is the effectiveness and potential of using ViT models for the detection of IUU activities?

ViT models have shown impressive performance in various image analysis tasks, including object detection and classification.

By leveraging their ability to capture spatial relationships and long-range dependencies, ViTs can potentially be applied to the detection of fishing activities. The use of ViT models for IUU detection can enable the analysis of AIS satellites to identify suspicious or illegal activities at sea. These models can learn to recognize specific patterns, vessel types, or behaviors associated with IUU activities, allowing for more efficient and accurate monitoring and enforcement efforts.

Additionally, the interpretability of ViT models can provide insights into the detected IUU activities, aiding in understanding the underlying factors and informing decision-making processes.

What are the most effective strategies?

The effectiveness of transformers in handling sequential data is widely recognized.

What emerges as the optimal strategy for studying trajectories? Should textual models be employed, treating trajectories as text sequences, or should images of the trajectories be utilized instead? Moreover, what information is advisable to incorporate in the inputs (e.g., the amount of information to be depicted on the images)?

In *Chapter 5* some different strategies will be proposed, starting from the simpler ones and then adding more and more information and complexity.

Is the pre-training dataset always so important?

The strong limiting factor of ViT architectures (and Transformers in general) has been previously described. The need for enormous datasets to train the model often translates to high costs and limited practical usability. Having a highly accurate model confined to only a few supercomputers renders it almost useless for the vast majority of use cases.

However, the question arises: How far is it possible to push the versatility of these models?

In this study, the aim to assess the importance of pretraining will be followed, and there will be an examination of whether a model pre-trained on a completely different dataset from the one used in this study can still achieve satisfactory results in classifying naval activities.

There will be an exploration of the extent to which the pretraining effect impacts the model's performance, observing if a model pre-trained on diverse data can generalize effectively and demonstrate its potential in the task of naval activity classification. By investigating these aspects, this study hopes to gain insights into the transferability and adaptability of Vision Transformer models, shedding light on their capabilities beyond traditional pretraining setups.

How can a model like this be used in maritime activities real-time monitoring?

What are the true potentials of a system like this, and what are the real-world cases where it would be useful?

In particular, this question is focused on how the results of the model explained above can be used in real case studies. So what are the major potentialities (and criticalities) related to the use of this kind of model in everyday life?

In *Chapter 6* after explaining how the models have been tested and applied, some interesting points will be presented from the perspective of the authorities.

In particular, there will be the evaluation of whether there is a possibility of utilizing and harnessing the attention mechanism applied by the model to capture important patterns or critical points in maritime traffic. By studying the attention patterns generated by the Vision Transformer during the classification of naval activities, the study aims to identify if the model can effectively focus on relevant areas and specific

features that are crucial for distinguishing different maritime activities.

Figure 3.1: *Sikorsky MH-60T Jayhawk* helicopter during coastal patrol operations.

Understanding how the attention mechanism works in this context can provide valuable insights into how the model processes and prioritizes information. It may reveal whether the Vision Transformer can automatically detect significant spatial and temporal patterns in the data, potentially aiding in maritime surveillance, anomaly detection, or optimizing navigation strategies.

By analyzing the attention distribution, it is possible to uncover the model's ability to discern relevant patterns and enhance its interpretability, making it an even more valuable tool for understanding and analyzing complex maritime traffic scenarios.

4 | Datasets

This chapter will provide an overview of the available data, analyzing the data obtained from AIS systems and describing their limitations and potential.

4.1. From the AIS to the data

As already mentioned in *Chapter 1*, AIS data is unique because it is exchanged among different entities, for various purposes, and in different ways.

For completeness, it is worth mentioning a bit more information regarding this system. AIS includes a GPS receiver that stores data on the vessel's position and direction. All the collected data is subsequently transmitted in a way that can be interpreted by other AIS transponders. The VHF system is used for this transmission.

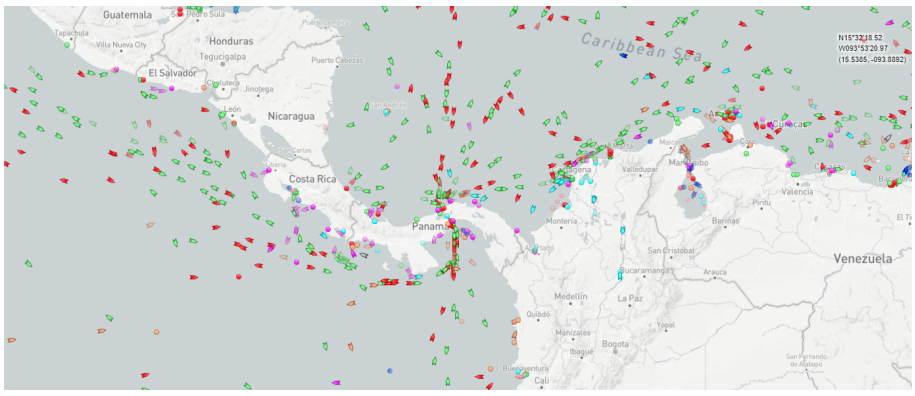

Figure 4.1: Maritime traffic around Panama Canal. Screen-shoot of *MarineTraffic* website, one of the most famous AIS live maps providers.

4.1.1. AIS transponder classes

There are various classes of AIS instruments based on functionalities and range. All commercial vessels over 300 tons and all passenger ships are equipped with **Class A** AIS. These instruments transmit at very high periodicity through a dedicated

antenna and receive data from all types of AIS. These units have a prioritization system for transmitting navigation data, ensuring that signals from different vessels in the same area do not overlap.

The system architecture can handle up to 4500 stations in the same area. Class A AIS must also be equipped with a dedicated display and a computer that analyzes collision risk with every other received signal.

On smaller vessels, many fishing boats, or yachts, it is possible to find **Class B** instruments. They also transmit and receive but are less powerful than Class A. Additionally, they do not have a prioritization system for transmitting navigation data. They may have a dedicated screen or provide information to be displayed on a chart plotter or laptop.

Finally, there are simple receivers, called **Class C**, which do not transmit any information but can receive transmitted data. This allows users to see all vessels equipped with Class A or B AIS in the vicinity. These receivers are very useful for avoiding collisions with commercial traffic, although users still need to remain vigilant.

4.1.2. AIS information

Depending on whether an AIS transceiver is installed on a pleasure boat or a commercial vessel, the transmitted data can be significantly different. The transmission of the following data is generally possible through AIS but is fully utilized in the commercial context.

In recreational boating, the MMSI number, the vessel's name, position, course, and dimensions are usually sufficient. It is possible to summarize all the data transmitted via AIS into 3 main areas, where the bolded information represents the data typically transmitted regardless of the type of vessel.

The data primarily used in this study will be related to ship identification, its type, position, and the time the message was transmitted. In other words, an attempt was made to develop a model that relies solely on the data that is almost always available, without the need for specific information that might not always be complete or present.

Static Data
Name of the Vessel
International Call Sign
MMSI Code
IMO Code
Vessel Type (e.g., fishing, cargo ship, tanker, passenger, etc.)
Dimensions of the Vessel

Table 4.1: Static information contained in AIS messages.

Dynamic Data
Position (Latitude and Longitude)
Speed (SOG, Speed Over Ground)
Route (COG, Course Over Ground)
Time of the Transmission
Navigation Status
Heading
Course Rate of Change

Table 4.2: Dinamic information contained in AIS messages.

Travel-related Data
Current Draft
Max Draft
Destination Port
Departure Port
Estimated Time of Arrival (ETA)
Cargo Type (class of hazardous cargo, if there is any)

Table 4.3: Travel-related information contained in AIS messages.

4.2. AIS communication subjects

As already mentioned several times, the AIS system serves various users. The primary users are undoubtedly different vessels, which use the system to increase their situational awareness. Although AIS does not replace the use of traditional radar, it is undoubtedly a valuable ally and the combined use of both allows for safe navigation even in nighttime or low visibility conditions. Of course, ship-to-ship communications are inherently "closed circuits" and cannot be used (except in specific cases) for research or study purposes.

AIS messages are then captured by specific satellites and collected by data providers for commercial purposes. Having access to such a comprehensive and massive amount of maritime data (practically the live and historical position of almost all vessels worldwide) is undeniably valuable. These data can be easily used for research purposes, as presented here.

However, the cost of the satellite equipment required for their reception makes them available only through private data providers (and at a high price!). Nowadays, their substantial commercial cost makes them practically exploitable only by large entities (public or private) that can afford the expense and thus have access to them.

The third entity that intervenes in AIS communications is the port authorities. As briefly described in Chapter 1, AIS data is collected by port authorities through a network of coastal antennas and used for managing and monitoring traffic along the national coastline.

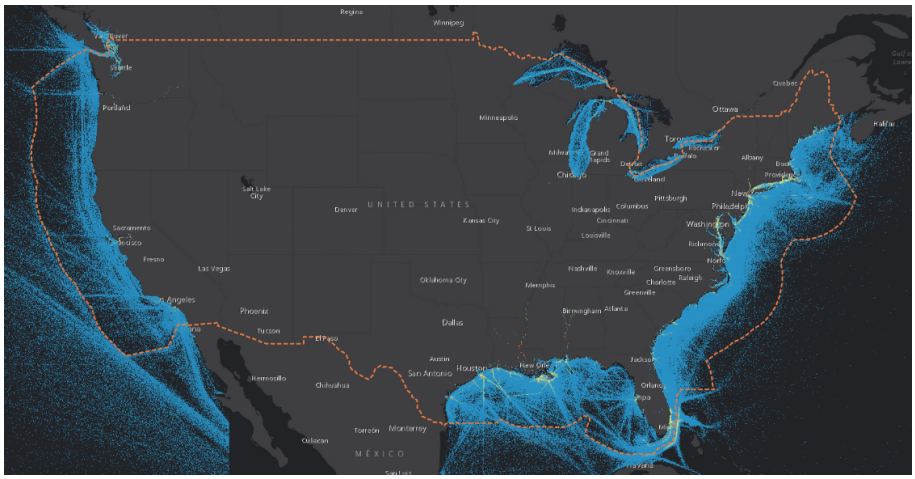

Figure 4.2: *AccessAIS* map, the download portal of US AIS data from *marinecadastre.gov*.

The antennas have a limited range, and data can be collected up to approximately 200 nautical miles from the coast (depending on weather and environmental conditions).

Not all countries make the data collected through this methodology available to the public. Specifically, as of today, only Denmark and the United States have dedicated portals where it is possible to request the download of AIS data.

The United States relies on the portal *marinecadastre.gov*, supported and managed by the government agency NOAA (National Oceanic and Atmospheric Administration). Here, it is possible to download the data collected from the entire network of U.S. coastal antennas since 2009, and the data is available in *.csv* format.

4.3. Data used

4.3.1. *When* and *Where*

After evaluating various options and contacting several environmental agencies, it has been decided to use the American portal as the main source of data.

Regarding the study area, the eastern coast of the United States has been selected. In particular, a rectangular area delimited by the coordinates provided below has been used as the data source for both the training and testing phases of the models.

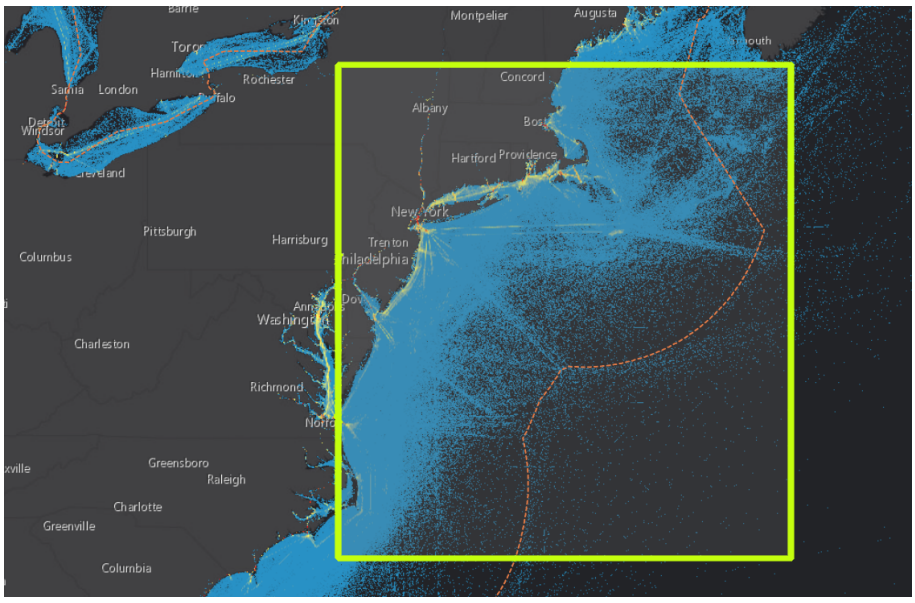

Figure 4.3: The geographic area from which the data was collected.

The specific zone was chosen due to its dense fishing activity. Other areas have lower fishing activity and, therefore, are less critical in terms of fisheries (such as the western coast or the Gulf of Mexico).

The table below summarises the characteristics of the data used in this thesis.

Detail	Value
Longitude Minimum	34.18621
Longitude Maximum	43.40141
Latitude Minimum	-75.94461
Latitude Maximum	-65.05574
TRAIN Data Time Range	from 2019-09-03 to 2019-12-07
TEST Data Time Range	from 2018-12-31 to 2019-01-17

Table 4.4: Dataset description.

4.3.2. How is structured

The data downloaded from the portal is in the usual .csv format and consists of a total of 28 features assigned to each row. Each row corresponds to an AIS signal received, and there is a clear need for data cleaning and organization before the actual study.

The columns comprehensively describe the characteristics mentioned in the tables above (static data, dynamic data, etc.), but not all columns are equally complete. After excluding the columns related to signal creation time, position, vessel speed, and vessel type, all the remaining columns may contain empty fields to varying degrees. This is related to the presence of different classes of AIS devices, which allow sending information with varying levels of completeness and detail.

During the pre-processing phase in the upcoming chapter, when analyzing the number of trajectories, it becomes apparent that the dataset exhibits a slight imbalance in terms of fishing-related activities. Approximately one-fourth of the recorded journeys are associated with fishing activities, while the majority pertain to other types of marine endeavors.

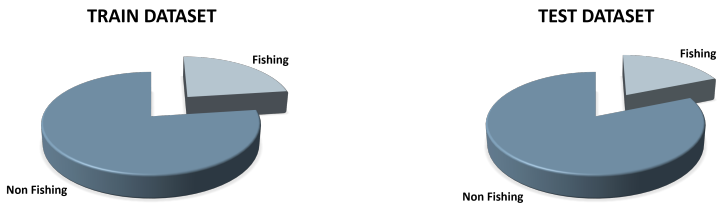

Figure 4.4: Dataset items distribution between fishing and non-fishing activities.

Specifically, the dataset contains a total of 104,566 trajectories in the training set, out of which 23,957 trajectories correspond to fishing vessels. This means that roughly 22.9% of the journeys are linked to fishing-related operations. Similarly, in the testing dataset, which comprises 20,200 trajectories, 3,778 of them are attributed to fishing vessels. Thus, around 18.7% of the trajectories in the testing set are associated with fishing activities.

5 | Models

This chapter will provide a description of all the methodologies used in this thesis. It is divided into 3 main sections: the first one is about the pre-processing phase, where raw AIS data was converted into useful trajectories; then the second will describe the first test done using a classic Transformer architecture and studying maritime trajectories as sequences of characters; the last section will be about the actual application of ViT models to classify naval activities.

5.1. Data Preparation

This initial phase is common to both approaches used for the actual classification. As described in the previous chapter, AIS data appears as long sequences of rows, where each row represents an AIS signal. Therefore, the need to organize all this amount of data is evident, starting from the point cloud related to various coordinates and creating ordered trajectories that are easy to study.

It is indeed possible to represent the initial AIS data as a set of spatially scattered points. Each point is associated with a signal from a vessel, but the trajectories and behaviors of individual ships are not clear at this stage (see *Figure 5.1*).

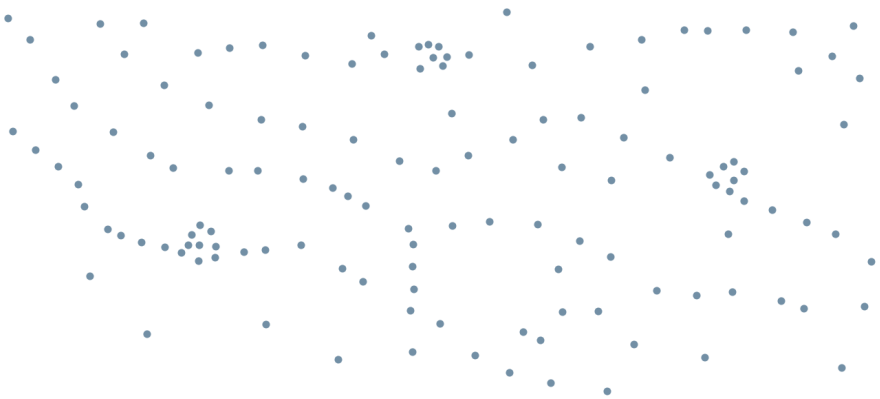

Figure 5.1: Simplified representation of AIS points from the dataset.

After removing any items with NaN values in the features related to position, identification, and type, the first step was to simplify the vast amount of data as much as possible. Specifically, all rows referring to "special" boats, such as military vessels, tugboats, or those involved in search and rescue operations at sea (SAR), were removed. These vehicles were excluded because they exhibit highly different and unpredictable behaviors, often unrelated to existing regulations. Additionally, their number is very limited, and removing the data associated with them does not disturb the rest of the dataset in any way.

It is easy to notice how the human eye can almost instantly identify different trajectories among the cluster of points. However, it is a completely different challenge for an algorithm to achieve the same task. To tackle this, a decision was made to leverage the study of various anchorage areas to create a robust method that autonomously identifies different trajectories. This method starts with a simple analysis of the individual AIS point messages transmitted.

As per maritime regulations, ships are obligated to emit AIS signals at regular intervals, even during periods when they are anchored. This continuous emission of signals results in the formation of clusters of data points, as demonstrated in the image above. Each cluster represents the AIS transmissions from individual vessels while they are stationed at the anchor. The primary objective of this thesis was, therefore, to outline a procedure for creating trajectories from these simple points, which would then be used for classification purposes.

5.1.1. Trips Mining by applying MEC

The process of transforming these discrete data points into coherent trajectories presented several complexities. Vessels within an anchorage area may have varying degrees of movement, leading to irregular patterns in the clustered data. Some ships may remain relatively stationary with slight drift, while others might experience more significant shifts due to currents or other factors.

To study this phenomenon, the Minimal Enclosing Circle (MEC) algorithm was chosen as the primary tool.

The first step involved converting the coordinate system from degrees (longitude and latitude) to Universal Transverse Mercator (UTM). This transformation allowed the evaluation of different AIS signals as points in a two-dimensional reference system.

The second step was to use the unique International Maritime Organization (IMO)

identification code associated with each AIS signal to partition the vast dataset into sub-datasets, each pertaining to a single vessel (see *Figure 5.2*). Subsequently, each of these datasets was temporally sorted in ascending order, from the oldest to the most recent AIS signal. By implementing these steps, the study was able to organize the AIS data in a more manageable and structured manner, facilitating the application of the Minimal Enclosing Circle algorithm.

Figure 5.2: Sub-dataset containing AIS messages of one vessel only.

The Minimum Enclosing Circle (MEC) algorithm is a geometric algorithm used to find the smallest circle that encloses a given set of points in a two-dimensional plane. The algorithm's basic idea revolves around finding the center and radius of the circle that encloses the points while ensuring no point lies outside the circle.

For each point, the following five points were considered, and the Minimum Enclosing Circle (MEC) was calculated. A threshold of 75 meters was used to determine whether that point was related to a navigation phase or if the ship was anchored at that point. In other words, if a ship emits a series of AIS signals close to each other (such that the circle enclosing five consecutive signals has a radius smaller than the 75-meter limit), it is assumed that the ship is anchored and making minimal movements (see *Figure 5.3*).

Within the sub-datasets, the points were then divided between those related to navigation phases and those related to anchoring. For further simplification, consecutive points related to anchoring were condensed into a single point (see *Figure 5.4*) whose position was the average of the positions of the AIS signals where the ship was anchored.

Figure 5.3: Cluster of points related to an anchoring phase in dark blue.

Figure 5.4: Simplification of the dataset by approximating the cluster as a single point.

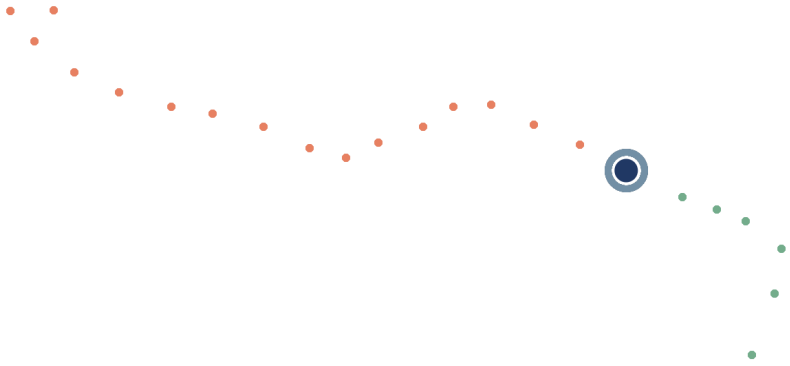

Figure 5.5: Definition of individual trajectories as a set of consecutive points between two anchoring points.

After defining the anchoring points, defining the trajectories becomes straightforward. The trajectories are the lines described by all consecutive points that lie between two anchoring points. The algorithm identifies these trajectories and assigns a unique identifier to each of them.

In *Figure 5.6* it is possible to see two trajectories (the green and the orange ones) separated by a point related to an anchoring phase.

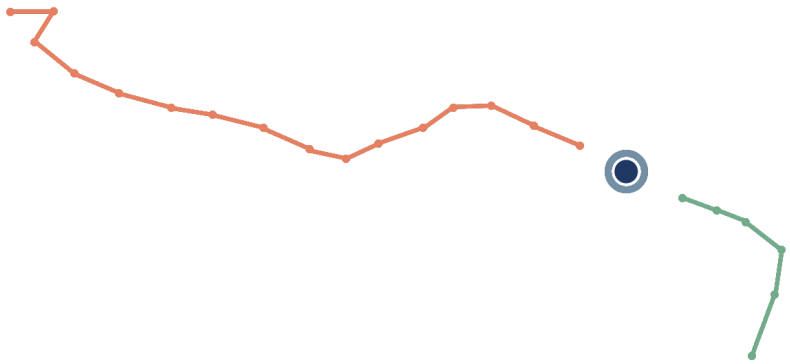

Figure 5.6: Two defined trajectories.

To increase the effectiveness and accuracy of this methodology, a second criterion was simultaneously used for defining the trajectories. They were not only "separated" by the anchoring points but also by a temporal factor. Specifically, whenever there was a time gap exceeding a predefined limit (24 hours) between two AIS signals, the end of the previous voyage and the start of the new one were automatically defined, even without any anchoring phase between the two.

5.1.2. Data Preparation - Algorithm

Pseudo-algorithms of the data preparation phase.

Algorithm 5.1 Data Preparation

```
Load required libraries and raw AIS data
for vessels do
    Order items in sub-dataset
    for items do
        Calculate MEC with 5 consecutive items
        if radius > threshold then
            Item is related to navigation phase
        else
            Item is related to anchoring phase
        end if
    end for
end for
Condensate consecutive anchoring phase-related points
Define trips and identify them via a unique ID
Split trips if there is a time gap greater than the threshold within it
```

5.2. Trajectory as a sentence

As previously mentioned, Transformer architectures are considered state-of-the-art when it comes to natural language processing. As the first methodology to use in identifying fishing activities, it was decided to apply them in their "natural habitat" by transforming trajectories into sentences.

Of course, it is not possible to do this using commonly used words, so spatial discretization has been exploited to describe trajectories not as a sequence of spatial coordinates but rather as a sequence of simple numerical characters.

Each trajectory was considered by discretizing the space it extends over. Each portion of discretized space was associated with a unique number. Therefore, the trajectory was described by a sequence of unique numbers referring to the sections of space it passes through.

5.2.1. From the trajectories to the sequences

To study fishing activities using Transformer architectures, the first step was to process the raw trajectory data. To achieve this, the continuous spatial information of each trajectory was discretized, effectively dividing the entire geographical area of interest into smaller sections (see *Figure 5.8*). These sections were then uniquely labeled with numerical identifiers, which enabled the transformation of spatial data into a structured sequence of discrete values (see *Figure 5.10*).

Imagine the geographical region where fishing activities are being monitored as a grid-like system. Each cell or section of this grid is assigned a specific numeric label, creating a lookup table that connects the spatial positions with their respective unique numerical representations. This process effectively transforms the spatial trajectories of fishing vessels into a language-like sequence of numbers, making them amenable to analysis using the power of Transformer models.

Once the trajectories were encoded into these sequences of unique numeric labels, they could be treated as natural language sentences, where each number corresponds to a specific "word" in the trajectory "sentence." With this representation in hand, Transformer architectures, renowned for their exceptional capabilities in understanding and generating natural language, could be applied to the task of classifying fishing activities. In the following sequence of images, it is possible to observe the progression from a trajectory (it is an example, not from the dataset used) to its spatial discretization and then to the sequence of IDs.

Figure 5.7: Trajectory before spatial discretization.

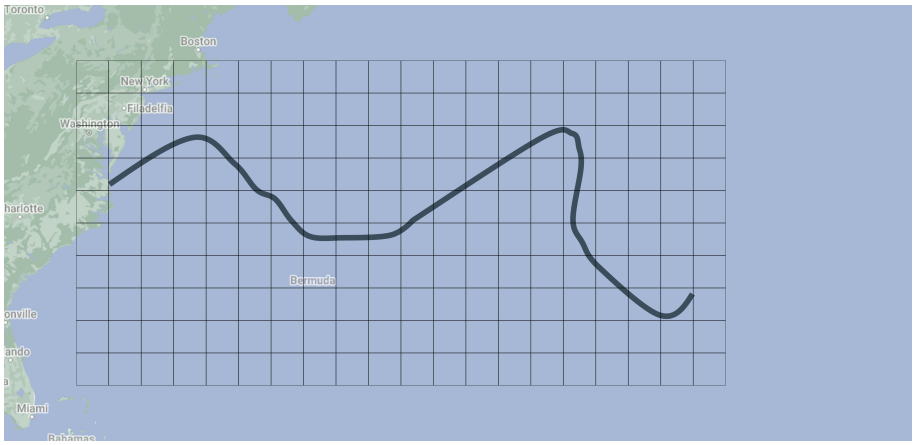

Figure 5.8: Grid-based spatial discretization.

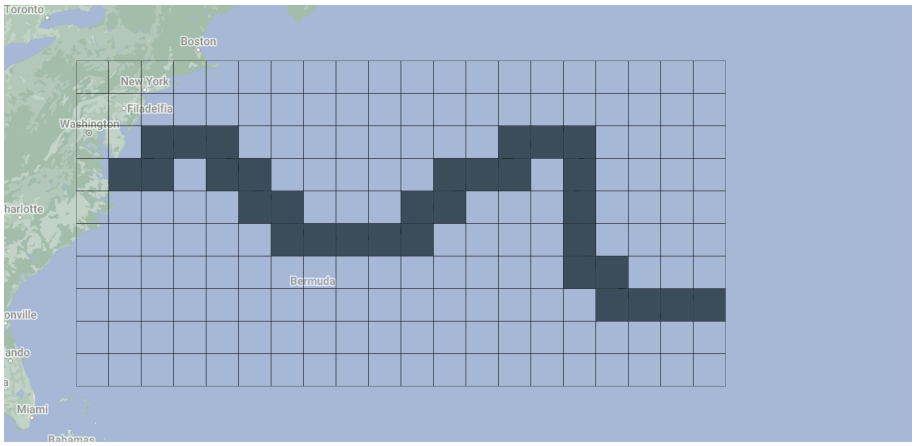

Figure 5.9: Cells touched by the trajectory.

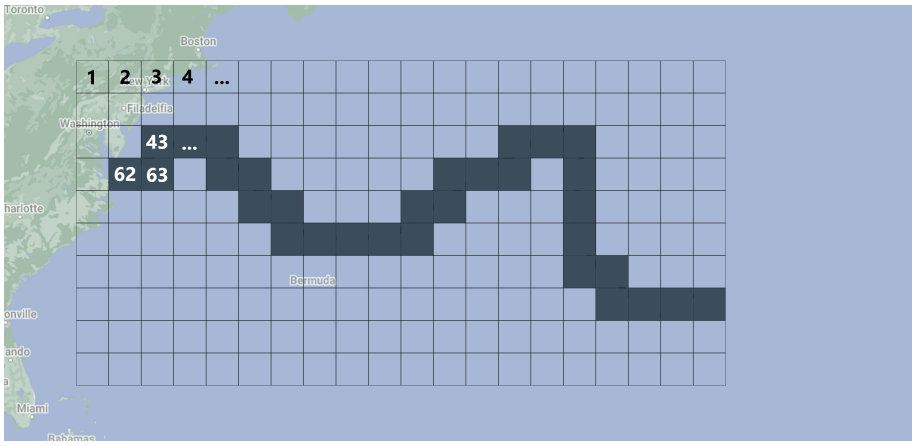

Figure 5.10: The output is simply the sequence of cell IDs.

5.2.2. Sequence classification

In this section, a briefly explanation of the model used for trajectory classification will be presented. The details concerning the execution of scripts and programs will be addressed together with the results in the subsequent, dedicated chapter. To classify different character sequences, specifically ship trajectories, as either related to fishing activities or not, researchers opted to build a transformer model from scratch.

Regarding the model related to trajectory classification as textual sequences, it can be analyzed and described by breaking it down into 7 main blocks.

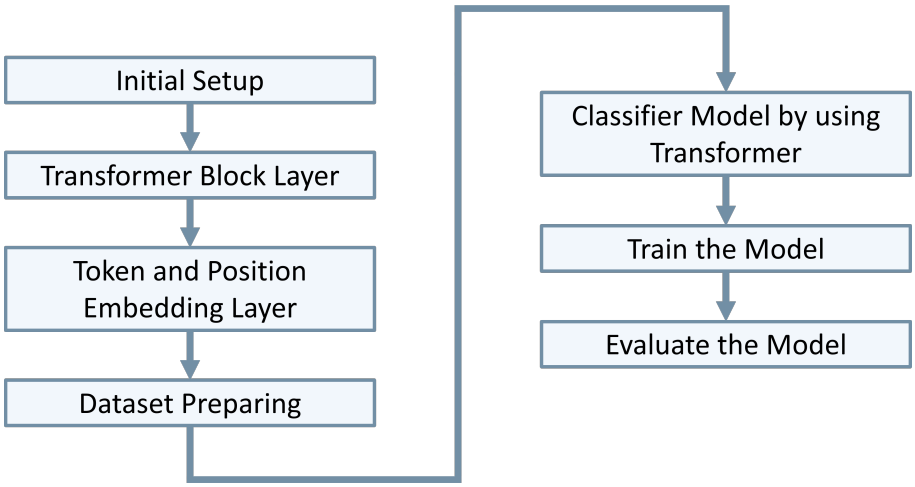

Figure 5.11: Sequence classification model main steps.

In the initial setup section, the code imports all the necessary libraries, including Keras and TensorFlow. Keras module provides a user-friendly and intuitive interface

for building, training, and deploying deep learning models, while TensorFlow, on the other hand, is an open-source deep learning framework that allows users to create complex computational graphs and execute them efficiently on various hardware platforms, including CPUs, GPUs, and TPUs (Tensor Processing Units).

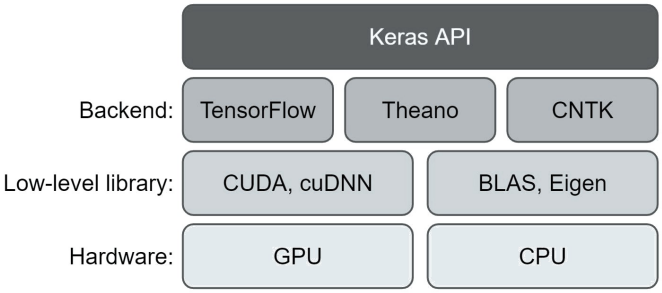

Figure 5.12: Keras and TensorFlow relationship.

An appropriate flag set by the user allows them to choose the type of hardware to be utilized by the code, either by selecting GPU acceleration or running on CPU. This option became necessary due to the different computing environments that users may have access to. The code was initially developed and tested on a desktop PC, but later required virtual environments with GPUs available for training. This step is common in both approaches presented in the thesis.

The second macro step of the code is the definition of the transformer block (named "*TransformerBlock*"). At this stage, a custom Keras layer is defined, with the goal of representing the transformer architecture. It is a basic version of a Transformer with all the characteristics discussed previously in Chapter 1 (focusing on the self-attention mechanism).

The third step in the code is the implementation of the "*TokenAndPositionEmbedding*" class, which serves as a custom Keras layer for embedding the input text data (i.e., the trajectories) as tokens and their corresponding positions. This layer is a crucial component in preparing the text data for processing through the subsequent Transformer blocks. The output of the layer is the sum of token and position embeddings, effectively incorporating both the spatial information (positions) and semantic information (tokens) into the continuous vector representations of the in-

put text data. This prepares the data for further processing through the subsequent Transformer layers, enabling the model to capture the relationships and dependencies between tokens in the text sequences effectively.

In the fourth step, the dataset is loaded. Minor modifications to the data are done at this stage.

The fifth stage is where the model architecture is defined. The input sequences are passed through the custom *TokenAndPositionEmbedding* prepared in the second step. Then, the data is passed through the *TransformerBlock* layer. The final output from the *TransformerBlock* is averaged across all time steps using a particular pooling layer (specifically called "*GlobalAveragePooling1D*"), followed by a couple of Dense layers to classify the text into two classes (positive or negative sentiment). The two classes are referred to as fishing or not-fishing maritime activities.

The fifth and sixth points are where the actual model defined in the previous points is used. The model is compiled with an optimizer, loss function, and evaluation metric. The code proceeds to train the model on the training data and evaluate its performance on the validation data. The training is done for 10 epochs with a batch size of 32, but further details will be discussed in the following chapter.

5.2.3. Trajectory as a sequence - Algorithm

Pseudo-algorithms of the first classification method.

Algorithm 5.2 Sequence Creation

```
Load required libraries and pre-processed data
for trips do
    Define grid size (regarding spatial discretization)
    for logs do
        Calculate the discretized space where the AIS log follows
        if square not equal to previous then
            Add square ID to the sequence
        end if
    end for
end for
```

Algorithm 5.3 Sequence Classification

```
Load required libraries and sequences
Define the Transformer
for sequences do
    Prepare sequence
end for
for epochs do
    Use the sequences to train the Transformer model
end for
Test the Transformer model
```

5.3. Trajectory as a picture

In this section, the second methodology used for naval activity classification will be presented, involving the consideration of both trajectory data and trajectory images. The research direction shifted towards this approach due to the less-than-ideal results obtained in the first methodology. It was hypothesized that leveraging the Transformer's ability to capture shape features in trajectory images could potentially improve the classification performance.

Before delving into the methodology, it is essential to recap the similarities and differences with the first approach. As previously mentioned, the focus is now different: transformers will be no longer used to classify sequences, but pictures. The core nature of the transformers model will remain the same.

In contrast to the first approach, where a single trajectory creation methodology was employed, in this case, the study explored various trajectory image creation techniques. As detailed in the next chapter, four different types of trajectory images were generated and analyzed, each incorporating a different level of information.

The biggest difference is that there will be used a pre-trained version of ViT. The decision to use a pre-trained model offered a practical and efficient solution to achieve favorable results and it was driven by two primary reasons:

1. Transformers, as seen before, have demonstrated exceptional capabilities in Natural Language Processing tasks. However, training these models from scratch necessitates vast amounts of data, which, in this case, were not readily available. Despite the abundance of AIS (Automatic Identification System) data that captures ship movements, the time and resources required to train a Vision transformer model from scratch were prohibitive (several times the data required to train a vanilla transformer). Consequently, the most viable approach was to utilize an existing pre-trained model and fine-tune it specifically for the image classification task.

2. Making the best of the situation: by utilizing a pre-trained model that had been trained on diverse datasets, potentially dissimilar from the data used in this particular study, researchers had the opportunity to assess the adaptability and versatility of these models. It allowed them to explore how effectively the pre-trained model could be fine-tuned to perform well on a specific task, even in the presence of variations in the data distribution.

5.3.1. From the trajectories to the images

Also in this case, to study fishing activities using Transformer architectures, the first step was to process the raw trajectory data. It was necessary to find a way to generate a vast volume of images within a reasonable time frame and with consistent criteria.

The first approach was to leverage the spatial discretization described earlier, which was used to create textual sequences, for generating images as well. In essence, the groundwork was already in place: instead of extrapolating a sequence of unique numbers, the approach involved saving an image depicting the spatial sections through which the trajectory passed (see *Figure 5.12*, it is an example of picture developed from the same trajectory used to show the previous sequence approach).

To clarify further, the spatial discretization process, which was initially used to create sequences of discrete points representing the trajectory's spatial positions, was adapted for generating trajectory images.

The generated images were square-shaped, and the scale varied from one image to another to ensure that the entire trajectory could be represented as large as possible.

Figure 5.13: Trajectory as an image.

This approach was similar to the one used with textual sequences, where the discretization scale was not constant. As trajectories could vary significantly in length and spatial coverage, using a fixed scale for all images might result in important details being lost in longer trajectories or wasting space in shorter ones. Therefore in this first case, the scale was adjusted for each image to ensure that the trajectory was adequately captured and that important spatial features were preserved.

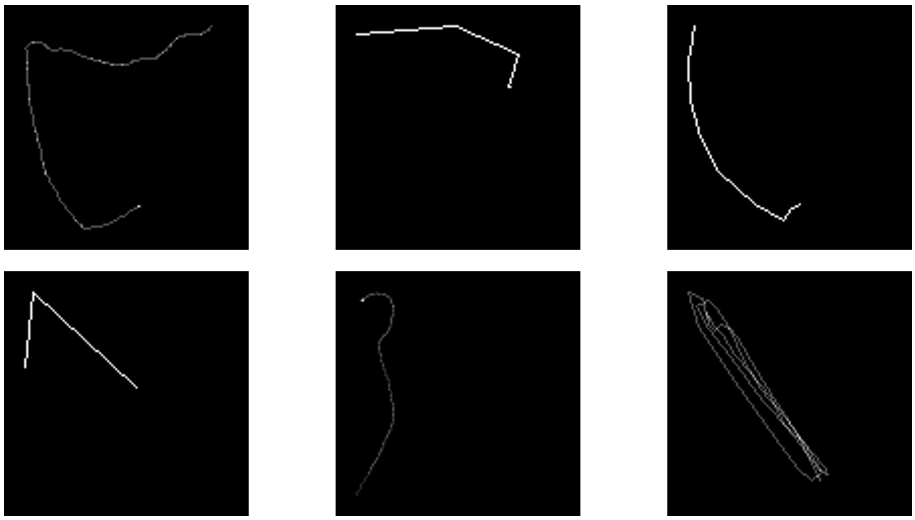

Figure 5.14: First image synthesis approach.

In the images above and below (*Figure 5.13*), it is possible to see six examples of the trajectory images used. Notice the granularity of the images, which is determined by the spatial discretization method employed. All images are in black and white and have dimensions of 100px by 100px (*Figure 5.14*). Each pixel represents one of the spatial sections resulting from the spatial discretization process. Specifically, each white pixel represents a space section where the ship has passed.

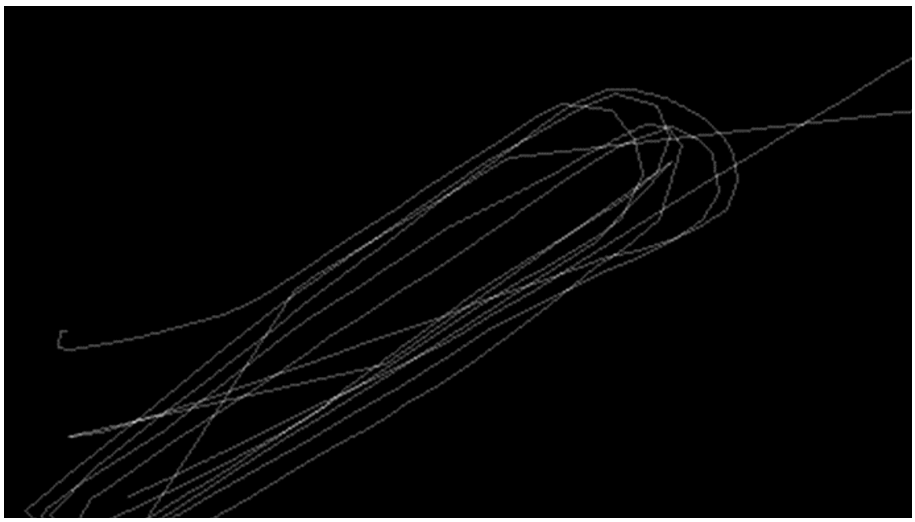

Figure 5.15: Zoom over pixels, related to spatial discretization.

After observing a significant improvement in the results (as discussed in the subsequent chapter), the decision was made to test additional variations of trajectory images.

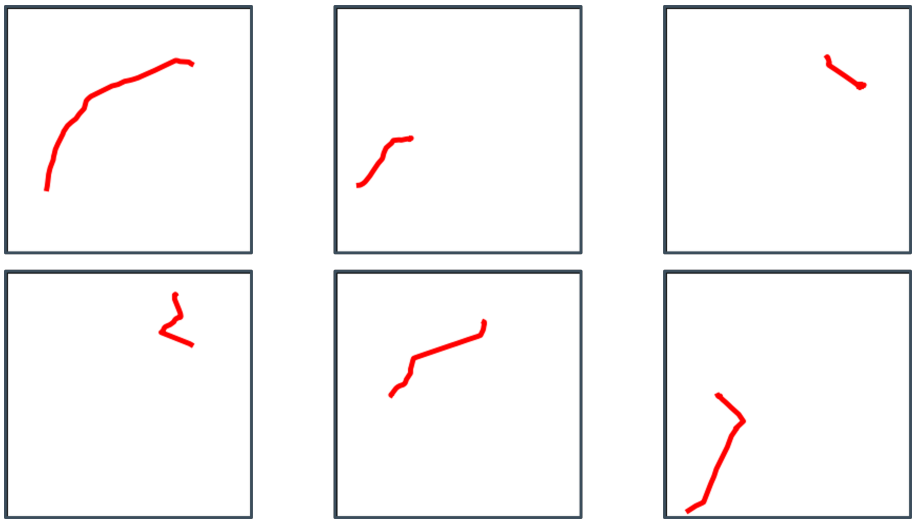

Figure 5.16: Second image synthesis approach.

The first evolution involved a change in the zoom perspective. Instead of focusing on the trajectory's shape, the attention shifted toward the ship's geographic position. In the second graphical approach, images were created with a constant scale and geographic center for all trajectories (*Figure 5.15*). From this point onward, no spatial discretization is used. The trajectories are represented as images of lines formed by simply connecting the points of the AIS message coordinates.

By maintaining a consistent scale and center across all images, the shape of the trajectory was no longer emphasized, particularly for shorter trajectories. However, this approach provided valuable information about the geographic location of the trajectory.

The images may appear visually similar to the first approach, but they are fundamentally different. In the first approach, the emphasis was on highlighting the shape of the trajectories, whereas in these images, the focus shifted toward their geographic distribution.

This distinction will become even more apparent with the next further evolution.

The third set of images created and used is, in fact, the same as described above in the second point. The trajectories are the same, and the methodology is identical, with the only difference being the addition of a map as the background. It is easy to notice that the zoom and center of the images are constant across all images.

Figure 5.17: Third image synthesis approach.

The fourth (and final) approach represents a further evolution in terms of the amount of information included. Multiple indicators have been added for both the starting and ending points of the trajectory. Additionally, the trajectory's color representation is no longer constant; instead, it is depicted using a range of tones based on the normalized velocity of the ship.

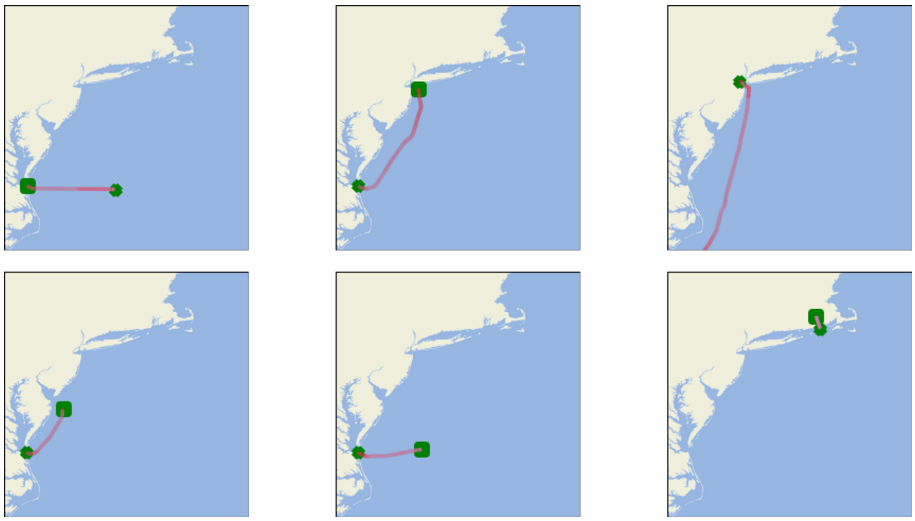

Figure 5.18: Fourth image synthesis approach.

In this approach, trajectory images are still represented as continuous lines connecting consecutive points from the AIS message coordinates. However, several indicators are now incorporated at both the starting and ending points of the trajectory. These indicators could represent various characteristics or events related to

the ship's journey, providing more context and information about the trajectory's beginning and end.

By including multiple indicators and employing a variable color representation, the trajectory images become enriched with more relevant information. This approach allows the model to capture and learn from not only the spatial path of the trajectory but also the variations in velocity and other critical indicators during the ship's movement.

Having presented the four different approaches used to create four distinct datasets, the model employed for their classification will now be described. Remarkably, this model was used without any modifications on all four types of images, enabling meaningful comparisons across them.

5.3.2. Image classification

In this section, a description of the model used for image classification will be performed.

Following a similar process to what was done for the analysis of the model used for sequence classification, the model used here has been divided into steps, which will now be analyzed one by one.

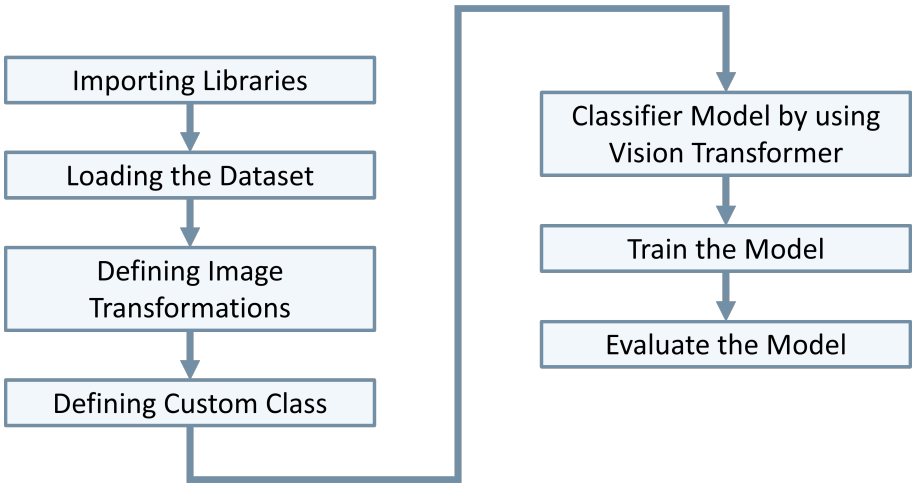

Figure 5.19: Image classification model main steps.

Similarly to the previous approach, the first step involves initializing the actual model, and all the required libraries for the model are loaded.

In the second step, the data used for image classification is loaded. During this stage,

the different datasets created were loaded one by one to evaluate their performance from a classification perspective.

To efficiently handle a large number of images, a special command class from the Hugging Face library called "*ImageFolder*" was utilized. This class streamlines the process of organizing and loading images into the model. By using the *ImageFolder* command, the numerous images were exported in the form of a comprehensive dataset stored in a single file. This approach significantly eased the management and sharing of the dataset, as dealing with large amounts of image data can be challenging and may require careful handling. This greatly facilitated the sharing of files, which was essential given the necessity to use virtual computing environments.

The need to utilize virtual computing environments arises from the size and complexity of the dataset. These environments provide the necessary computational resources, such as GPU support, to efficiently train deep learning models like the Vision Transformer (ViT) on large-scale datasets. By efficiently organizing and loading the data using the *ImageFolder* command and leveraging virtual computing environments, the image classification model can be effectively trained and evaluated on diverse datasets.

In the third stage, the script defines image transformations using "*transforms.Compose*" from *torchvision* library. The transformations include converting images to tensors, resizing them to (224, 224) pixels, and normalizing the pixel values. This step is required to use the images as input for the transformer layers.

This passage can be a source of confusion, so let's take a moment to clarify. When referring to "image transformations" here, the discussion does not pertain to the actual Transformers model. Instead, the focus is on the phase of image transformations (such as resizing, formatting, etc.) that are required before using images with the Vision Transformer (ViT) model.

The size of the tensors is not constant through the different datasets used. The first image synthesis approach has a result B&W pictures (so 2-bit colors), while all the others create images in RGBA format (four color channels: Red, Green, Blue, and Alpha, and so a four-level tensor).

In the fourth step, the script defines a custom dataset class named *ImageDataset*, which is tailored for the ViT model. It inherits from *torch.utils.data.Dataset* and provides methods to access images and their corresponding labels.

In the fifth step, the code defines a custom PyTorch module for the Vision Trans-

former (ViT) model. The custom module is called *ViT*, and it extends the functionality of the *ViTModel* class provided by the Hugging Face's transformers library. The *ViT* class combines the Vision Transformer model with a linear classifier for multi-class classification tasks. It allows for fine-tuning the pre-trained ViT model on a specific image classification dataset. The attention weights are also made available during the forward pass, which can be useful for understanding the model's attention mechanism.

The last steps are quite similar to the ones referred to in the sequences analysis. In them, the code loads the model, loss function (cross-entropy), and optimizer (Adam). Training data is then loaded using the *ImageDataset* class and then fed into the model in batches. The model is trained for a specified number of epochs (10), and the loss and accuracy are printed at the end of each epoch.

The script provides also a prediction function to make predictions using the trained ViT model. It takes an image as input, pre-processes it, and returns the predicted label.

5.3.3. Trajectory as a picture - Algorithm

See the **Code Annex** for the code of this last approach.

6 | Experiments

In this chapter, the focus shifts to the "practical" aspect of this thesis. Delving into the theory or structure of the models (which has already been described earlier) is not the objective. Instead, the chapter addresses the practical issues associated with experimentation. Providing a brief overview of the programming environment used and outlining any assumptions made will serve as an introduction to the actual experimentation with the classification models.

6.1. Implementation

Throughout the entire process of planning, designing, and building the models described in this thesis, the Python programming language has been used as the sole tool.

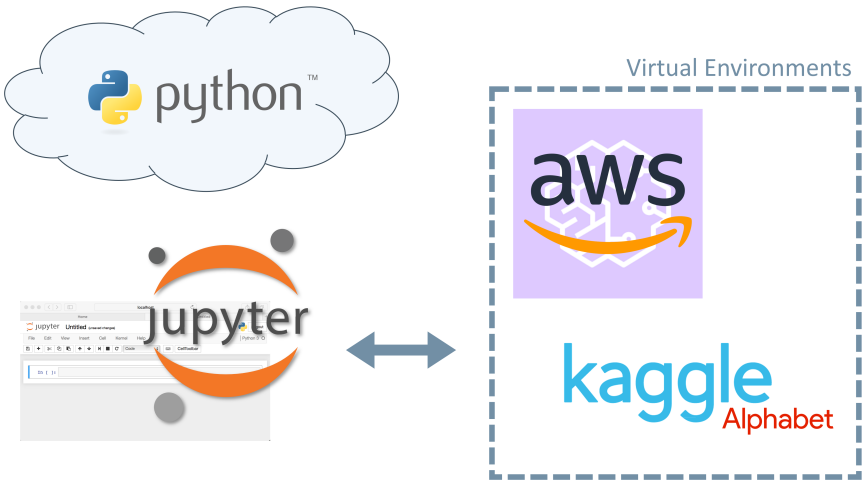

Figure 6.1: Implementation environments.

Python is a versatile and widely-used high-level programming language known for its simplicity, readability, and extensive ecosystem of libraries and frameworks. It has gained immense popularity in the field of data science, machine learning, and

artificial intelligence due to its ease of use and robust community support. Python offers a wide range of libraries and tools specifically tailored for tasks like data manipulation, numerical computing, and machine learning, some of which have been already mentioned in the previous chapter.

The actual coding process was performed using the Jupyter Notebook application on a desktop PC. Jupyter Notebook is an interactive computing environment that allows users to create and share documents combining live code, visualizations, narrative text, and other rich media. Its user-friendliness and the ability to test small code stages live were particularly useful during the more complex steps. Once the code was defined, the execution phase was carried out on different cloud platforms.

The codes related to the pre-processing of raw AIS data were executed on the AWS SageMaker cloud portal, mainly due to its unlimited CPU computation hours (this step did not require GPU acceleration).

On the other hand, the experimentation with different Transformer-based approaches was more challenging. Like many deep learning models, Transformers have a highly efficient training phase when executed on GPUs. For this reason, they were executed on another cloud environment: Kaggle, from Alphabet, which provides users with several hours of weekly GPU computation.

The specifications of the computational environments used are described in the table below.

Characteristic	Desktop PC	Google Kaggle	AWS SageMaker
CPU Model	Intel i7 4770	Intel Xeon	Intel Xeon Scalable
GPU Model	NVIDIA GTX760	NVIDIA TeslaP100	NVIDIA TeslaP100
RAM	16GB	8GB to 64GB	16GB to 32GB
V-RAM	2GB	13 GB to 16 GB	4GB to 8GB
CPU hours	-	unlimited	unlimited
GPU hours	-	30 per week	4 to 8 per day
Availability	-	very high	none
Cloud storage	-	5GB	5GB
Ease of use	-	low	high

Table 6.1: Description of the different computational environments (free-plans only are considered regarding cloud services).

6.2. Assumptions

Some assumptions have been made, both for the sake of simplifying processes and due to the nature of the data. They are listed below and briefly described.

- To classify fishing activities, the vessel type indicated by the AIS (Automatic Identification System) transponder was used. This means that if a boat has the identifier "fishing vessel," its activity will be classified as fishing activity, regardless of whether it is actively fishing at that particular moment or not.
- It is assumed that in the dataset used for training, there is a 100% correct indication regarding the vessel type transmitted through the AIS signal and later used for classification. There is no available dataset with absolute truth to reference for this purpose.

Figure 6.2: Drone footage of a fishing boat.

- The same datasets (both for training and testing) were used in all the cases reported in the previous chapter. From the study of sequences to that of images, the same journeys from the same area were the subjects of classification. This is not an assumption but a characteristic that is essential to emphasize the pursuit of balance among different cases.

6.3. Metrics

In the next section, the results of each version of the presented models will be shown. To facilitate comparison, various commonly used metrics for evaluating classification models have been employed. Below is a brief overview of these metrics.

A Confusion Matrix (*Figure 6.3*) is a popular representation of the performance of classification models. The matrix shows the number of correctly and incorrectly classified examples, compared to the actual outcomes (true values) in the test data. It is easy to understand that a goof classifier should describe a matrix with high values on the diagonal spots.

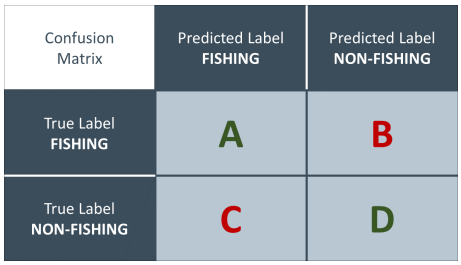

Figure 6.3: Example of Confusion Matrix.

When discussing binary classification, it is possible to refer to true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN). Considering the example table above, the true positives and true negatives (TP and TN) would be represented by cells A and D, respectively, while the false positives and false negatives (FP and FN) would be represented by cells B and C, respectively.

Accuracy is the percentage of correctly classified samples out of the total samples (Equation (6.1)). It provides a general measure of the model's overall performance.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{6.1}$$

Precision, also known as positive predictive value (PPV), represents the proportion of true positive predictions out of all positive predictions (Equation (6.2)). It indicates the model's ability to avoid false positive errors.

$$Precision = \frac{TP}{FP + TN} \tag{6.2}$$

Recall is also known as sensitivity or true positive rate (TPR), and it represents the

proportion of true positive predictions out of all actual positive samples. It indicates the model's ability to capture all positive samples (Equation (6.3)).

$$Recall = \frac{TP}{TP + FN} \tag{6.3}$$

F1-Score is the harmonic mean of precision and recall (Equation (6.4)). It provides a balanced measure between precision and recall, especially when classes are imbalanced.

$$F1Score = \frac{2TP}{2TP + FP + FN} \tag{6.4}$$

6.4. Sequences - Experiments

The first test carried out was the one done considering the trajectories as text sequences.

During the training of the model, each epoch took approximately 12-15 minutes, resulting in a total time of about 2 and a half hours for 10 epochs.

6.4.1. Sequences - Results

In the table below, the confusion matrix obtained by testing the model on the test dataset and the main comparison metrics are represented.

Trajectory as a sequence	Predicted Label **FISHING**	Predicted Label **NON-FISHING**
True Label **FISHING**	**14**	**3764**
True Label **NON-FISHING**	**25**	**16397**

Figure 6.4: Confusion Matrix - Sequences.

Metric	Value
Accuracy	0.8124
PPV	0.0037
TPR	0.3590
F1 Score	0.0073

Table 6.2: Metrics - Sequences.

Even though the accuracy may seem reasonable, one should not be deceived. The fact that the dataset tends to be imbalanced toward one of the two classes (the number of trajectories associated with activities other than fishing is significantly higher) means that if this model tends to predict all instances as belonging to the majority class, it can achieve high accuracy. However, when studying the F1 score, it becomes evident that the model is neither accurate nor correct in its predictions.

6.5. Pictures - Experiments

The model used for image classification differs significantly from the one used for sequences. One of the distinctive features is that it starts from a pre-trained transformer architecture and then fine-tunes it.

The model used as the base architecture is "*google/vit-base-patch16-224-in21k*." The specific variant refers to the base configuration of the model. In this case, the image is divided into non-overlapping patches of size 16x16 pixels. The input image size of 224x224 pixels and the model is pre-trained on a large-scale dataset containing 21,000 classes. This means it has seen a wide range of visual concepts during pre-training, which can make it more adept at recognizing different objects and patterns.

During the training (fine-tuning) of the model, the time required for each epoch was not constant but changed from one kind of picture to another. By considering the first and the second picture versions, 10 to 15 minutes were required, resulting in a total time of about 2 and a half hours for 10 epochs. The third and fourth datasets contain pictures far more complex, and so the required time was about 30 minutes longer for each epoch, defining a total training time of about 7 hours and a half.

6.5.1. Pictures - Results

In the table below, the confusion matrix obtained by testing the model on the test dataset and the main comparison metrics are represented.

As can be seen from the first dataset, the results are better when compared to the previous case.

A significant improvement in quality was achieved by imposing constant focus and centering, while the model does not seem to show significant performance gains or losses with the addition of further information.

Trajectory as a picture (#1)	Predicted Label **FISHING**	Predicted Label **NON-FISHING**
True Label **FISHING**	**1247**	**2531**
True Label **NON-FISHING**	**2641**	**13781**

Figure 6.5: Confusion Matrix - Pictures (dataset 1).

Metric	Value
Accuracy	0.7440
PPV	0.3301
TPR	0.3207
F1 Score	0.3253

Table 6.3: Metrics - Pictures (dataset 1).

Trajectory as a picture (#2)	Predicted Label **FISHING**	Predicted Label **NON-FISHING**
True Label **FISHING**	**3056**	**722**
True Label **NON-FISHING**	**247**	**16175**

Figure 6.6: Confusion Matrix - Pictures (dataset 2).

Metric	Value
Accuracy	0.9520
PPV	0.8089
TPR	0.9252
F1 Score	0.8632

Table 6.4: Metrics - Pictures (dataset 2).

Trajectory as a picture (#3)	Predicted Label **FISHING**	Predicted Label **NON-FISHING**
True Label **FISHING**	**3098**	**680**
True Label **NON-FISHING**	**229**	**16193**

Figure 6.7: Confusion Matrix - Pictures (dataset 3).

Metric	Value
Accuracy	0.9550
PPV	0.8200
TPR	0.9312
F1 Score	0.8721

Table 6.5: Metrics - Pictures (dataset 3).

Trajectory as a picture (#4)	Predicted Label **FISHING**	Predicted Label **NON-FISHING**
True Label **FISHING**	**3279**	**499**
True Label **NON-FISHING**	**434**	**15988**

Figure 6.8: Confusion Matrix - Pictures (dataset 4).

Metric	Value
Accuracy	0.9538
PPV	0.8679
TPR	0.8831
F1 Score	0.8755

Table 6.6: Metrics - Pictures (dataset 4).

The above-mentioned results show a progressively better evolution in the model's performance, albeit reaching a certain asymptote.

Even though in the final chapter, they will be thoroughly discussed, it is interesting to note that through the images, the model can accurately identify activities related to fishing with very satisfactory performance.

6.6. Real Case Scenarios

Ignoring for a moment the actual performance of classification models and focusing solely on visual approaches (image classification), it is possible to draw some interesting conclusions.

For example, which vessels are classified by the models as fishing boats, perhaps incorrectly? Or which fishing boats are engaged in activities classified as "non-fishing"?

The answer to the second question is straightforward: fishing boats often transit between fishing areas, and during these journeys, they may exhibit behaviors similar to non-fishing vessels traveling along the same routes. This similarity in behavior can lead to misclassifications by image classification models.

Regarding the first question, identifying which vessels are mistakenly classified as fishing boats by image classification models requires a more detailed analysis.

To do this, the boats that the latest model presented (ViT classifier) classified as engaged in fishing activity, despite having a different transponder identifier from that of fishing vessels, were studied.

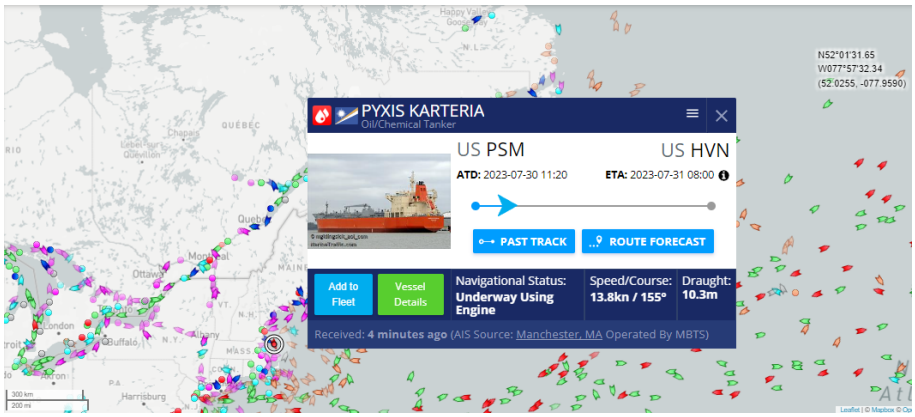

Figure 6.9: View of vessel details on MarineTraffic website.

Through an additional Python script, an analysis was performed using the website *MarineTraffic*. Each IMO identifier code associated with mistakenly classified vessels was compared with the web portal to gather further information about the respective boat.

It is important to note that the portal cannot be considered an absolute source of truth. It is a dataset maintained and updated by community volunteers, so this

study should only be regarded as a curiosity by researchers. However, it could serve as evidence of how this model could be utilized by law enforcement and coast guards (public entities with access to more comprehensive and official datasets and information).

6.6.1. Examples of False Negatives: The Seawolf vessel

A particular boat has fueled the interest of researchers. Although it appears as a generic AIS message "other type," the Seawolf looks like a fishing boat.

Figure 6.10: Seawolf vessel (pictures from marinetraffic.com).

The community on the website confirms this hypothesis, effectively categorizing it as a fishing boat.

Having sparked interest, further investigation into the vessel's history was conducted to understand the possible reasons behind the mismatch between its activities and the AIS indication. It was discovered that the boat, built in 1982, is now used for research and education purposes. Its owner is Stony Brook University, which specializes in marine and meteorological sciences. The ship is, therefore, used for educational activities, often in situations comparable to those of fishing vessels. This new information sheds light on the vessel's current role and activities, explaining the "other type" designation in the AIS message. While the ship was originally a fishing boat, it has been repurposed for research and educational endeavors, leading to the discrepancy in its classification. This finding highlights the importance of considering historical context and any possible changes in a vessel's purpose when interpreting AIS data and identifying ship types. It is evident that this model can be utilized to detect anomalous maritime activities (such as fishing boats attempting to conceal their presence by changing their AIS radio classification). However, it should always be complemented with a minimum research effort to fully understand the possible reasons behind false negatives.

7 | Conclusions

In this final chapter, all the possible conclusions of the work will be summarized, highlighting critical points, and proposing future developments.

In this thesis, researchers explored the possibility of harnessing the potential of transformer architectures in the context of marine activity classification.

Specifically, they aimed to determine the most effective strategy to achieve this goal and identify the practical applications of such an approach. Transformer architectures have demonstrated remarkable capabilities in natural language processing tasks, image recognition, and various other domains. The researchers hypothesized that these architectures could be adapted and fine-tuned for marine activity classification using Automatic Identification System (AIS) data.

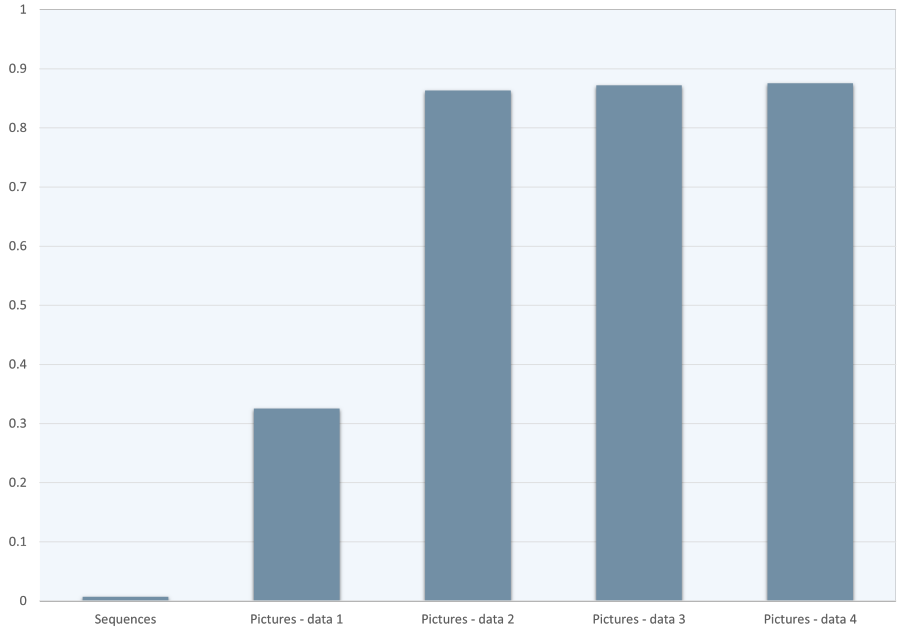

Figure 7.1: F1 scores comparison.

In the graph above, it is posible to see a comparison of F1 scores for different models used. The first one on the left refers to the classifier model utilizing sequences, while

the others are related to trajectory classifications using the produced images (with their four corresponding strategies).

It is useful to remember that the F1 score ranges from 0 to 1, with 1 being the best possible score. A perfect F1 score of 1 indicates that the model has achieved both perfect precision and perfect recall, meaning it correctly identifies all positive instances (true positives) and does not misclassify any negative instances (false positives) or vice versa (false negatives). An F1 score of 0 means that either precision or recall is 0, indicating a model's complete failure in correctly classifying one of the classes.

Evaluating the different scores, it is evident that studying trajectories as images yields significantly better results compared to studying naval routes as sequences.

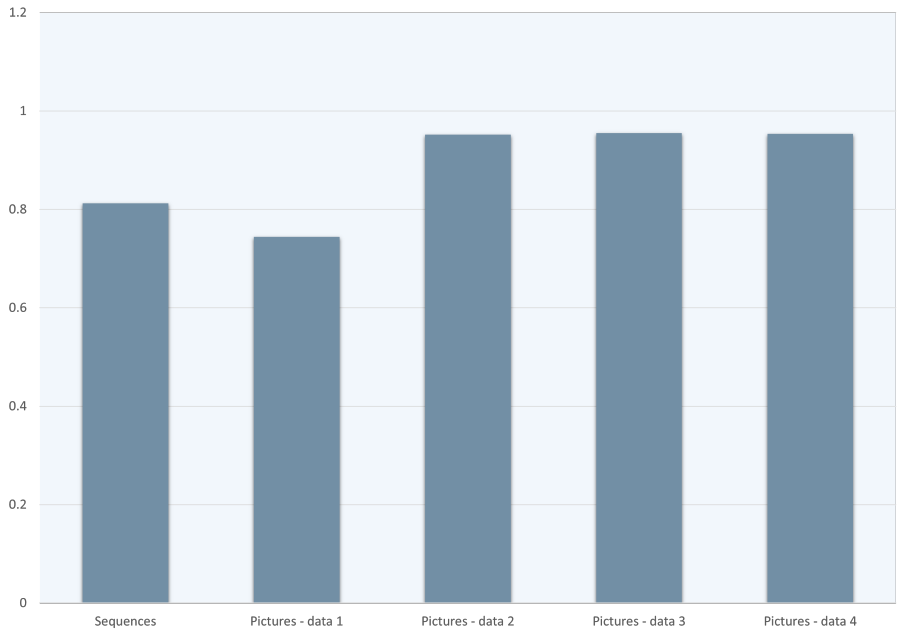

Figure 7.2: Accuracy scores comparison.

A slightly different outcome is obtained by comparing the accuracies of the different models. This highlights the fact that the net value of accuracy tends to be not entirely reliable in the case of even partially imbalanced datasets.

As presented earlier, the datasets used in both the training and testing phases exhibit a relatively significant imbalance (further data processing through oversampling techniques was not pursued precisely because the imbalance was not at levels requiring it).

Such imbalance (a clear majority of items related to non-fishing activities) allows for misleading results: the first model is poor, as it classifies almost all trajectories as "non-fishing," yet it still achieves a respectable accuracy value.

Another clear conclusion is that the classification of fixed-scale and zoomed images (focusing on geographical dispersion and not just the shape of the trajectory) promises better results than other alternatives.

The quantity of information contained in each image does not seem to significantly impact the performance. Ships often undergo prolonged periods of cruising at constant speeds, which may explain why considering speed in the analysis resulted in almost negligible improvements.

The time required for model training was roughly constant across different types of images, but the process of creating the images varied significantly. The first two datasets took only a few hours, while the other two took several days. The rendering of complex images demands more computational power, and since it did not bring any significant benefits, it may not be the optimal option for real-time operational environments or real-time maritime traffic analysis.

The Transformer architectures, particularly the Vision Transformer (ViT), demonstrated versatility. Excellent classification results were achieved even when starting from models pre-trained on entirely different images. This highlights the immense potential of an ideal model trained solely on a massive number of naval scenarios.

In summary, using images to study trajectories and leveraging the power of Transformer architectures, especially ViT, holds great promise for improving the classification of maritime activities. The study of fixed-scale and zoomed images and the consideration of speed may be crucial factors in achieving accurate results. However, the practical implementation should consider the computational demands and real-time operational requirements when using complex image rendering.

7.1. Further Developments

From a future perspective, several further advancements can be made in this area.

Firstly, a more in-depth study of the type of information to include in the images could lead to new insights. While this study did not find information that significantly improved classification, it does not mean that such information does not exist. Exploring different features or representations in the images might unveil

additional patterns and correlations that could enhance classification performance. Including meteorological conditions would help to take into consideration the fact that weather may influence some ships' trajectories. There is also a bit of further experimenting room by considering different ways to create the pictures themselves (i.e., considering an image of the trajectory where for each picture, the center of the image is the starting point of the maritime trip).

Secondly, there is potential for studying how this model could be used for classifying other types of maritime activities or diving into more detailed classifications, such as different types of fishing practices. Expanding the scope of the model's capabilities to handle various activities can widen its applicability and usefulness in monitoring and managing maritime resources and security.

Thirdly, there is some room for development considering different ways of leveraging this kind of deep learning architecture. Just consider, for example, how it could be interesting to apply the models described here, which are already trained on different geographical areas from where they have been trained, or even use them not for the purpose of pure classification, but for example, to detect anomalies in naval trajectories.

Lastly, developing a system that integrates real-time maritime activity on a single dashboard, similar to the live map available on websites like marinetraffic.com, along with indications of activities classified as suspicious by the model (false negatives), would be highly beneficial. Such an integrated approach would increase both the effectiveness and efficiency in combating Illegal, Unreported, and Unregulated (IUU) activities. In this way a far more dept analysis could be interesting to do, bringing together information from different perspectives, such as aggregate information about a vessel across multiple trajectories of itself. By quickly flagging potential suspicious behaviors, authorities can take prompt action, improving maritime security and resource management.

Overall, continued research and development in these directions can lead to more robust and versatile models for maritime activity classification, enhancing maritime monitoring, management, and security measures. Using AI models like this can be a valuable tool in monitoring and identifying unusual maritime behavior. It can help authorities and researchers identify potential cases of misclassified vessels or vessels engaging in suspicious activities. Nevertheless, it is essential to acknowledge that these models may have limitations and may not always provide a complete or accurate picture.

Bibliography

[1] K. G. Aarsæther. Estimating navigation patterns from ais. *The Journal of Navigation*, 2019.

[2] K. Bae. Transformer networks for trajectory classification. *2022 IEEE International Conference on Big Data and Smart Computing (BigComp)*, 2022.

[3] L. F. Chuah. Extracting shipping route patterns by trajectory clustering model based on automatic identification system data. *Sustainability*, 2018.

[4] G. K. D. de Vries. Machine learning for vessel trajectories using compression, alignments and domain knowledge. *Expert Systems with Applications*, 2018.

[5] A. Dosovitskiy. An image is worth 16x16 words: Transformers for image recognition at scale. *ICLR 2021*, 2020.

[6] E. d'Afflisio. Maritime anomaly detection based on mean-reverting stochastic processes applied to a real-world scenario. *21st International Conference on Information Fusion (FUSION)*, 2018.

[7] M. Elwakdy. A novel trajectories classification approach for different types of ships using a polynomial function and anfis. *International Conference on Image Processing, Computer Vision, and Pattern Recognition*, 2015.

[8] Y. Endo. Classifying spatial trajectories using representation learning. *Int J Data Sci Anal 2016*, 2016.

[9] F. Giuliari. Transformer networks for trajectory forecasting. *25th International Conference on Pattern Recognition (ICPR)*, 2020.

[10] F. Katsilieris. Detection of malicious ais position spoofing by exploiting radar information. *16th International Conference on Information Fusion (FUSION)*, 2013.

[11] R. Keane. Detecting motion anomalies. *8th ACM SIGSPATIAL Workshop on GeoStreaming*, 2017.

[12] P. Kraus. Ship classification based on trajectory data with machine-learning methods. *The 19th International Radar Symposium IRS 2018*, 2018.

[13] Y. Liang. Trajformer: Efficient trajectory classification with transformers. *CIKM '22, October 17–21, 2022, Atlanta, GA, USA*, 2022.

[14] H. Liu. End-to-end trajectory transportation mode classification using bi-lstm recurrent neural network. *12th International Conference on Intelligent Systems and Knowledge Engineering (ISKE)*, 2017.

[15] H. Liu. Spatio-temporal gru for trajectory classification. *2019 IEEE International Conference on Data Mining (ICDM)*, 2018.

[16] P. Mishra. Vt-adl: A vision transformer network for image anomaly detection and localization. *arXiv:2104.10036v1*, 2021.

[17] NATO. Ais (automatic identification system) overview, 2021. URL here.

[18] A. Navaz. Convolutional lstm based transportation mode learning from raw gps trajectories. *IET Intelligent Transport Systems*, 2020.

[19] D. D. Nguyen. Vessel trajectory prediction using sequence-to-sequence models over spatial grid. *DEBS '18'*, 2018.

[20] G. Palotta. Vessel pattern knowledge discovery from ais data: A framework for anomaly detection and route prediction. *Entropy*, 2017.

[21] D. S. Pedroche. Architecture for trajectory-based fishing ship classification with ais data. *Sensors 2020, 20, 3782; doi:10.3390/s20133782*, 2020.

[22] A. Radon. Contextual verification for false alarm reduction in maritime anomaly detection. *IEEE International Conference on Big Data*, 2015.

[23] K. Sheng. Research on ship classification based on trajectory features. *Cambridge University Press*, 2017.

[24] USGS. How much water is there on earth?, 2019. URL here.

[25] A. Vaswani. Attention is all you need. *31st Conference on Neural Information Processing Systems*, 2017.

[26] Z. Wei. Ais trajectory simplification algorithm considering ship behaviours. *Ocean Eng.*, 2020.

[27] Wikipedia. Illegal, unreported and unregulated fishing, 2023. URL here.

[28] WROMS. World register of marine species website, 2023. URL here.

[29] H. Wu. A fast trajectory outlier detection approach via driving behavior modeling. *CIKM'17, November 6-10, 2017*, 2017.

[30] Z. Yan. Ship classification and anomaly detection based on spaceborne ais data considering behavior characteristic. *Sensors 2022, 22, 7713*, 2022.

[31] D. Zhang. ibat: Detecting anomalous taxi trajectories from gps traces. *UbiComp'11*, 2011.

[32] Y. Zhang. Ut-atd: Universal transformer for anomalous trajectory detection by embedding trajectory information. *10.18293/DMSVIVA2021-011*, 2021.

A | Appendix A

In this chapter, the code used to create and classify images will be presented.

A.1. Picture Creation

```
FILE_PS = 'Datasets/AIS_CLEAN_Data/TRAIN_POSTPROC_mk2.parquet'
import math
import folium
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.collections as mcoll
import matplotlib.colors as mcolors
import matplotlib.path as mpath
from tqdm import tqdm
import gc
import geopandas as gpd
import io
import cartopy
import cartopy.crs as ccrs
import glob
import imageio
from PIL import Image
import shapefile
from shapely.geometry import shape,mapping, Point, Polygon,
    MultiPolygon
import pyproj
import imgkit
def utm_to_latlon(easting, northing, zone_number, zone_letter,
    northern_hemisphere=True):
    utm_proj = pyproj.Proj(proj='utm', zone=zone_number, ellps='
        WGS84', north=northern_hemisphere)
    lon, lat = utm_proj(easting, northing, inverse=True)
    return lat, lon
def Average(lst):
    return sum(lst) / len(lst)
def trip_sequencer(trip_X,trip_Y,cell_size,xmin,ymin):
    trip_grid = np.zeros((len(trip_X),2), dtype=int)
    for i in range(len(trip_X)):
        x_value = math.ceil((trip_X[i]-xmin)/cell_size)
        y_value = math.ceil((trip_Y[i]-ymin)/cell_size)
        trip_grid[i,0] = x_value
        trip_grid[i,1] = y_value
    return trip_grid
def trip_completor(trip_sequence):
    iterations = 2
    i = 1
    while i < iterations:
```

```
41          distance_with_previous = abs((trip_sequence[i,0] -
                trip_sequence[i-1,0]))+\
42                  abs((trip_sequence[i,1] - trip_sequence[i-1,1]))
43          if distance_with_previous>1:
44              points = generate_integer_points(trip_sequence[i-1,:],
                    trip_sequence[i,:])
45              trip_sequence = np.concatenate((trip_sequence[:i],
                    points, trip_sequence[i:]), axis=0)
46          i = i+1
47          iterations = len(trip_sequence)
48      return trip_sequence
49  def PRE_trip_sequencer(trip_X,trip_Y,N_pixels,border,print_FLAG):
50      xmin, xmax = min(trip_X)-border, max(trip_X)+border
51      ymin, ymax = min(trip_Y)-border, max(trip_Y)+border
52      xrange = xmax-xmin
53      yrange = ymax-ymin
54
55      cell_size = math.ceil(max(xrange,yrange)/N_pixels) #[m/pixel]
56
57      if xrange>yrange:
58          ymax = ymax+(xrange-yrange)
59      else:
60          xmax = xmax+(yrange-xrange)
61
62      xrange = xmax-xmin
63      yrange = ymax-ymin
64
65      num_cells_x = math.ceil(xrange/cell_size)
66      num_cells_y = math.ceil(yrange/cell_size)
67      num_cells_tot = num_cells_x*num_cells_y
68      if print_FLAG:
69          print(num_cells_tot,' cells created')
70      return xmin, xmax, ymin, ymax, cell_size, num_cells_x,
            num_cells_y
71  def generate_integer_points(start, end):
72      dx = int(round(end[0] - start[0]))
73      dy = int(round(end[1] - start[1]))
74      points = np.zeros((max(abs(dx), abs(dy)) + 1,2), dtype=int)
75      for i in range(max(abs(dx), abs(dy)) + 1):
76          x = int(round(start[0] + i*dx/(max(abs(dx), abs(dy)))))
77          y = int(round(start[1] + i*dy/(max(abs(dx), abs(dy)))))
78          points[i,0] = x
79          points[i,1] = y
80      points = points[1:-1]
81      return points
82  def POST_trip_sequencer(adding,N_pixels,trip_sequence,padding):
83      trajectory_matrix = np.zeros((N_pixels+1,N_pixels+1), dtype=int
            )
84      trajectory_matrix_2bit = np.zeros((N_pixels+1,N_pixels+1),
            dtype=int)
85      for i in range(len(trip_sequence)):
86          old = trajectory_matrix[trip_sequence[i,0],trip_sequence[i
                ,1]]
87          trajectory_matrix[trip_sequence[i,0],trip_sequence[i,1]] =
                old + adding
88      for i in range(len(trajectory_matrix)):
89          for j in range(len(trajectory_matrix)):
90              if trajectory_matrix[i,j]>0:
91                  trajectory_matrix_2bit[i,j] = 1
92      trajectory_matrix = np.pad(trajectory_matrix, padding, mode='
            constant')
93      trajectory_matrix_2bit = np.pad(trajectory_matrix_2bit, padding
            , mode='constant')
94      return trajectory_matrix, trajectory_matrix_2bit
```

```
def split_list(lst, split_size):
    return [lst[i:i+split_size] for i in range(0, len(lst), split_size)]
def colorline(
    x, y, z=None, cmap=plt.get_cmap('copper'), norm=plt.Normalize(0.0, 1.0),
        linewidth=3, alpha=1.0):
    if z is None:
        z = np.linspace(0.0, 1.0, len(x))
    if not hasattr(z, "__iter__"):
        z = np.array([z])
    z = np.asarray(z)
    segments = make_segments(x, y)
    lc = mcoll.LineCollection(segments, array=z, cmap=cmap, norm=norm,
                              linewidth=linewidth, alpha=alpha)
    ax = plt.gca()
    ax.add_collection(lc)
    return lc
def make_segments(x, y):
    points = np.array([x, y]).T.reshape(-1, 1, 2)
    segments = np.concatenate([points[:-1], points[1:]], axis=1)
    return segments
AIS_dataframe = pd.read_parquet(FILE_PS, engine='pyarrow')
TripID_list = list(AIS_dataframe['TripID'].unique())
url = 'https://stamen-tiles-{s}.a.ssl.fastly.net/toner-background/{z}/{x}/{y}{r}.png'
tile_layer = folium.TileLayer(tiles=url,name='Custom',attr=' ')
color_1 = '#FF0000' #red
color_2 = '#c27ba0' #pinkish
color_cmap = mcolors.LinearSegmentedColormap.from_list('custom_colormap', [color_1, color_2])
alredydone = True
if alredydone == False:
    minimum_area = 1e-6
    trips_to_remove = []

    for Trip in tqdm(TripID_list, position=0, leave=True):
        this_trip = AIS_dataframe.loc[AIS_dataframe['TripID'] == Trip]
        min_lat = this_trip['LAT'].min()
        max_lat = this_trip['LAT'].max()
        min_lon = this_trip['LON'].min()
        max_lon = this_trip['LON'].max()

        if (max_lat - min_lat) * (max_lon - min_lon) < minimum_area:
            trips_to_remove.append(Trip)

    AIS_dataframe2 = AIS_dataframe[~AIS_dataframe['TripID'].isin(trips_to_remove)]
    AIS_dataframe2.to_parquet('output.parquet', engine="pyarrow")
    TripID_list = list(AIS_dataframe2['TripID'].unique())
    print(len(list(AIS_dataframe['TripID'].unique())) - len(TripID_list))
Trip_analyzed = 100023
this_trip = AIS_dataframe.loc[AIS_dataframe['TripID'] == Trip_analyzed]
trip_X = this_trip['X'].tolist()
trip_Y = this_trip['Y'].tolist()
trip_lat = this_trip['LAT'].tolist()
trip_lon = this_trip['LON'].tolist()
trip_type = this_trip['VesselType'].tolist()[0]
print('Vessel Type is:',trip_type)
```

```
map = folium.Map(location=[Average(trip_lat), Average(trip_lon)], zoom_start=8,zoom_control=False,\
                 scrollWheelZoom=True,dragging=False,tiles=tile_layer,attr=' ',width=512, height=512)
points = []
folium.PolyLine(list(zip(trip_lat, trip_lon)), color='red').add_to(map)
display(map)
min_lat = this_trip['LAT'].min()
max_lat = this_trip['LAT'].max()
min_lon = this_trip['LON'].min()
max_lon = this_trip['LON'].max()
fig = plt.figure(figsize=(15,15))
ax = plt.axes(projection=ccrs.PlateCarree())
ax.set_extent([-76.75, -67.95, 34.60, 43.40], crs=ccrs.PlateCarree())
#ax.stock_img()
ax.add_feature(cartopy.feature.LAND)
ax.add_feature(cartopy.feature.OCEAN)
#ax.add_feature(cartopy.feature.COASTLINE,linewidth=0.3)
#ax.add_feature(cartopy.feature.BORDERS, linestyle=':',linewidth=0.3)
#ax.add_feature(cartopy.feature.LAKES, alpha=0.5)
#ax.add_feature(cartopy.feature.RIVERS)
#ax.plot(this_trip['LON'], this_trip['LAT'], color='red', linewidth=1.5, transform=ccrs.Geodetic())
max_speed = max(this_trip['SOG'])
colorline(this_trip['LON'], this_trip['LAT'],(this_trip['SOG']/max_speed),cmap=color_cmap, linewidth=5)
ax.scatter(list(this_trip['LON'])[0], list(this_trip['LAT'])[0], color='green', linewidth=20, marker='x',\
           transform=ccrs.Geodetic())
ax.scatter(list(this_trip['LON'])[-1], list(this_trip['LAT'])[-1], color='green', linewidth=20,marker='s',\
           transform=ccrs.Geodetic())
plt.savefig("test.png",pad_inches=0, bbox_inches='tight')
this_trip = AIS_dataframe.loc[AIS_dataframe['TripID'] == Trip_analyzed]
min_lat = this_trip['LAT'].min()
max_lat = this_trip['LAT'].max()
min_lon = this_trip['LON'].min()
max_lon = this_trip['LON'].max()

print((max_lat - min_lat) * (max_lon - min_lon))
test_savings = False

if test_savings:
    path_wkthmltoimage = r'E:\Programs\PC Programs\Wkhtmltopdf\bin\wkhtmltoimage.exe'
    config = imgkit.config(wkhtmltoimage=path_wkthmltoimage)
    options = {'quiet': '','crop-w': '512'}

    map.save('test.html')

    imgkit.from_file('test.html', 'out.jpg',options=options, config= config)

    img_data = map._to_png()
    img = Image.open(io.BytesIO(img_data))
    img.save('trajectory_map.png')
border = 10
N_pixels = 100
xmin, xmax, ymin, ymax, cell_size, num_cells_x, num_cells_y = PRE_trip_sequencer(trip_X,trip_Y,N_pixels,border,True)
```

```
trip_sequence = trip_sequencer(trip_X,trip_Y,cell_size,xmin,ymin)
trip_sequence = trip_completor(trip_sequence)
adding = 1
trajectory_matrix, trajectory_matrix_2bit = POST_trip_sequencer(
    adding,N_pixels,trip_sequence,10)
plt.figure(figsize=(20,20))
ax = plt.axes()
ax.matshow(trajectory_matrix, cmap=plt.cm.inferno)
plt.axis('off')
#plt.savefig("test.png", bbox_inches='tight')
plt.show()
create_ANALOG = False
create_2Bit = False

border = 0
N_pixels = 512
adding = 1
fig_size = 20
padding = 10

trips_ID = []
trips_IMO = []
trips_VesselType = []
trips_Fishing = []
alredydone = True
if alredydone == False:
    for Trip in tqdm(TripID_list, position=0, leave=True):
        this_trip = AIS_dataframe.loc[AIS_dataframe['TripID'] ==
            Trip]
        trips_ID.append(Trip)
        if len(this_trip['IMO'])>0:
            trips_IMO.append(this_trip['IMO'].iat[0])
            trips_VesselType.append(this_trip['VesselType'].iat[0])
            if this_trip['VesselType'].iat[0] == 30:
                trips_Fishing.append(1)
            else:
                trips_Fishing.append(0)

    image_resume = pd.DataFrame(list(zip(trips_ID, trips_IMO,
        trips_VesselType,trips_Fishing)),
                columns =['TripID', 'IMO', 'Type','isFishing'])
    image_resume.to_csv('TRAIN_resume.csv')
alredydone = True
if alredydone == False:
    for Trip in tqdm(TripID_list, position=0, leave=True):
            this_trip = AIS_dataframe.loc[AIS_dataframe['TripID']
                == Trip]
            if len(this_trip['IMO'])>0:
                map = folium.Map(location=[Average(this_trip['LAT'
                    ]), Average(this_trip['LON'])],\
                                 zoom_start=8,zoom_control=False,\
                            scrollWheelZoom=False,dragging=False,
                                tiles=tile_layer,attr=' ',\
                                  width=N_pixels, height=N_pixels)
                folium.PolyLine(list(zip(this_trip['LAT'],
                    this_trip['LON'])), color='red').add_to(map)
                if this_trip['VesselType'].iat[0] == 30:
                    name = 'Outputs/Pictures/Folium/trainHTML/
                        Fishing/'+str(Trip)+'.html'
                    map.save(name)
                else:
                    name = 'Outputs/Pictures/Folium/trainHTML/
                        Non_Fishing/'+str(Trip)+'.html'
                    map.save(name)
```

```
%%capture --no-display

for Trip in tqdm(TripID_list, position=0, leave=True):
    if create_ANALOG or create_2Bit:
        this_trip = AIS_dataframe.loc[AIS_dataframe['TripID'] ==
            Trip]
        trip_X = this_trip['X'].tolist()
        trip_Y = this_trip['Y'].tolist()
        xmin, xmax, ymin, ymax, cell_size, num_cells_x, num_cells_y
            = PRE_trip_sequencer(trip_X,trip_Y,N_pixels,border,
            False)
        trip_sequence = trip_sequencer(trip_X,trip_Y,cell_size,xmin
            ,ymin)
        trip_sequence = trip_completor(trip_sequence)
        trajectory_matrix, trajectory_matrix_2bit =
            POST_trip_sequencer(adding,N_pixels,trip_sequence,
            padding)
    if create_ANALOG:
        if this_trip['VesselType'].iat[0] == 30:
            name = 'Outputs/Pictures/Analog/train/Fishing/'+str(
                Trip)+'.png'
            imageio.imwrite(name, trajectory_matrix)
        else:
            name = 'Outputs/Pictures/Analog/train/Non_Fishing/'+str
                (Trip)+'.png'
            imageio.imwrite(name, trajectory_matrix)
    if create_2Bit:
        if this_trip['VesselType'].iat[0] == 30:
            name = 'Outputs/Pictures/2Bit/train/Fishing/'+str(Trip)
                +'_2bit.png'
            imageio.imwrite(name, trajectory_matrix_2bit)
        else:
            name = 'Outputs/Pictures/2Bit/train/Non_Fishing/'+str(
                Trip)+'_2bit.png'
            imageio.imwrite(name, trajectory_matrix_2bit)
    gc.collect()
TripID_list_splitted = split_list(TripID_list, 25000)
len(TripID_list_splitted)
for Trip in tqdm(TripID_list_splitted[0], position=0, leave=True):
    this_trip = AIS_dataframe.loc[AIS_dataframe['TripID'] == Trip]
    plt.figure(figsize=(3,3))
    ax = plt.axes(projection=ccrs.PlateCarree())
    ax.set_extent([-76.75, -67.95, 34.60, 43.40], crs=ccrs.
        PlateCarree())
    ax.add_feature(cartopy.feature.LAND)
    ax.add_feature(cartopy.feature.OCEAN)
    colorline(this_trip['LON'], this_trip['LAT'],(this_trip['SOG']/
        max(this_trip['SOG'])),cmap=color_cmap, linewidth=2.5)
    ax.scatter(list(this_trip['LON'])[0], list(this_trip['LAT'])
        [0], color='green',\
                linewidth=5, marker='x',transform=ccrs.Geodetic())
    ax.scatter(list(this_trip['LON'])[-1], list(this_trip['LAT'])
        [-1], color='green',\
                linewidth=5,marker='s',transform=ccrs.Geodetic())
    if this_trip['VesselType'].iat[0] == 30:
        name = 'Outputs/Pictures/Cartopy_mk2/test/Fishing/'+str(
            Trip)+'.png'
    else:
        name = 'Outputs/Pictures/Cartopy_mk2/test/Non_Fishing/'+str
            (Trip)+'.png'
    plt.savefig(name,pad_inches=0, bbox_inches='tight')
    plt.close('all')
```

A.2. Picture Classification

```
from datasets import load_dataset
import numpy as np
import pandas as pd
import torch
import datasets
import cv2
import torch.nn as nn
from transformers import ViTModel, ViTConfig
from torchvision import transforms
from torch.optim import Adam
from torch.utils.data import DataLoader
from tqdm import tqdm
import matplotlib.pyplot as plt
from sklearn import metrics
from sklearn.metrics import roc_auc_score, roc_curve
color_1 = 'red'
color_2 = 'blue'
color_bck = 'white'
#dataset = load_dataset("imagefolder", data_dir='/Users/masui/
    iCloudDrive/LastBoatNotLeast-master/Outputs/Pictures/Analog')
#dataset.save_to_disk('Datasets/AIS_CLEAN_Data/ImageFolder_Data/
    Analog_dataset')
#dataset = load_dataset("imagefolder", data_dir='/Users/masui/
    iCloudDrive/LastBoatNotLeast-master/Outputs/Pictures/Cartopy')
#dataset.save_to_disk('Datasets/AIS_CLEAN_Data/ImageFolder_Data/
    Cartopy_dataset')
#dataset = load_dataset("imagefolder", data_dir='/Users/masui/
    iCloudDrive/LastBoatNotLeast-master/Outputs/Pictures/Cartopy_mk2
    ')
#dataset.save_to_disk('Datasets/AIS_CLEAN_Data/ImageFolder_Data/
    Cartopy_mk2_dataset')
#dataset = load_dataset("imagefolder", data_dir='/Users/masui/
    iCloudDrive/LastBoatNotLeast-master/Outputs/Pictures/Cartopy_mk3
    ')
#dataset.save_to_disk('Datasets/AIS_CLEAN_Data/ImageFolder_Data/
    Cartopy_mk3_dataset')
#print(dataset)
#dataset = datasets.load_from_disk('/kaggle/input/analog-dataset/
    Analog_dataset')
#dataset = datasets.load_from_disk('Datasets/AIS_CLEAN_Data/
    ImageFolder_Data/Analog_dataset')
#dataset = datasets.load_from_disk('/kaggle/input/cartopy-dataset/
    Cartopy_dataset')
#dataset = datasets.load_from_disk('Datasets/AIS_CLEAN_Data/
    ImageFolder_Data/Cartopy_dataset')
#dataset = datasets.load_from_disk('/kaggle/input/cartopymk2-
    dataset/Cartopy_mk2_dataset')
#dataset = datasets.load_from_disk('Datasets/AIS_CLEAN_Data/
    ImageFolder_Data/Cartopy_mk2_dataset')
#dataset = datasets.load_from_disk('/kaggle/input/cartopy-mk3/
    Cartopy_mk3_dataset')
dataset = datasets.load_from_disk('Datasets/AIS_CLEAN_Data/
    ImageFolder_Data/Cartopy_mk3_dataset')
print(dataset)
labels = dataset["train"].features["label"].names
num_labels = len(dataset["train"].features["label"].names)
label2id, id2label = dict(), dict()
for i, label in enumerate(labels):
    label2id[label] = i
    id2label[i] = label
```

```
transform = transforms.Compose([transforms.ToTensor(),transforms.Resize((224, 224)),
        #transforms.Normalize(mean=[0.5, 0.5, 0.5],std=[0.5, 0.5, 0.5])])
        transforms.Normalize(mean=[0.5, 0.5, 0.5, 0.5],std=[0.5, 0.5, 0.5, 0.5])]) #COLOR RGBA
        #transforms.Normalize(mean=[0.5],std=[0.5])]) #B&W

class ImageDataset(torch.utils.data.Dataset):

    def __init__(self, input_data):
        self.input_data = input_data
        self.transform = transforms.Compose([
        transforms.ToTensor(),
        transforms.Resize((224, 224), antialias=True),
        #transforms.Normalize(mean=[0.5, 0.5, 0.5],std=[0.5, 0.5, 0.5])])
        transforms.Normalize(mean=[0.5, 0.5, 0.5, 0.5],std=[0.5, 0.5, 0.5, 0.5])]) #COLOR RGBA
        #transforms.Normalize(mean=[0.5],std=[0.5])]) #B&W

    def __len__(self):
        return len(self.input_data)

    def get_images(self, idx):
        return self.transform(self.input_data[idx]['image'])

    def get_labels(self, idx):
        return self.input_data[idx]['label']

    def __getitem__(self, idx):
        # Get input data in a batch
        train_images = self.get_images(idx)
        train_labels = self.get_labels(idx)
        return train_images, train_labels

class ViT(nn.Module):
    def __init__(self, config=ViTConfig(), num_labels=20, model_checkpoint='google/vit-base-patch16-224-in21k'):
        super(ViT, self).__init__()
        self.vit = ViTModel.from_pretrained(model_checkpoint, num_channels=4, add_pooling_layer=False, ignore_mismatched_sizes=True)
        self.classifier = nn.Linear(config.hidden_size, num_labels)

    def forward(self, x):
        outputs = self.vit(x, output_attentions=True)
        x = outputs['last_hidden_state']
        attention_weights = outputs['attentions']
        output = self.classifier(x[:, 0, :])
        return output, attention_weights

def model_train(dataset, epochs, learning_rate, bs):
    use_cuda = torch.cuda.is_available()
    device = torch.device("cuda" if use_cuda else "cpu")
    # Load model, loss function, and optimizer
    model = ViT().to(device)
    criterion = nn.CrossEntropyLoss().to(device)
    optimizer = Adam(model.parameters(), lr=learning_rate)
    # Load batch image
    train_dataset = ImageDataset(dataset)
    train_dataloader = DataLoader(train_dataset, num_workers=1, batch_size=bs, shuffle=True)
    # Fine tuning loop
```

```
98      for i in range(epochs):
99          total_acc_train = 0
100         total_loss_train = 0.0
101         for train_image, train_label in tqdm(train_dataloader):
102             output, attention_weights = model(train_image.to(device
                    ))
103             loss = criterion(output, train_label.to(device))
104             acc = (output.argmax(dim=1) == train_label.to(device)).
                    sum().item()
105             total_acc_train += acc
106             total_loss_train += loss.item()
107
108             loss.backward()
109             optimizer.step()
110             optimizer.zero_grad()
111         print(f'Epochs: {i + 1} | Loss: {total_loss_train / len(
                train_dataset): .3f} | Accuracy: {total_acc_train / len(
                train_dataset): .3f}')
112     return model
113
114
115 def predict(img):
116
117     use_cuda = torch.cuda.is_available()
118     device = torch.device("cuda" if use_cuda else "cpu")
119     transform = transforms.Compose([
120         transforms.ToTensor(),
121         transforms.Resize((224, 224)),
122         transforms.Normalize(mean=[0.5, 0.5, 0.5, 0.5],std=[0.5,
                0.5, 0.5, 0.5])])
123         #transforms.Normalize(mean=[0.5, 0.5, 0.5],std=[0.5, 0.5,
                0.5])])
124         #transforms.Normalize(mean=[0.5],std=[0.5])])
125     img = transform(img)
126     output, attention_weights = trained_model(img.unsqueeze(0).to(
            device))
127     prediction = output.argmax(dim=1).item()
128
129     return id2label[prediction]
130
131 #hyperparameters
132 EPOCHS = 10
133 LEARNING_RATE = 1e-6
134 BATCH_SIZE = 16
135 #train the model
136 trained_model = model_train(dataset['train'], EPOCHS, LEARNING_RATE
        , BATCH_SIZE)
137 image = dataset['test'][7]['image']
138 image = transform(image)
139
140 device = torch.device('cuda' if torch.cuda.is_available() else 'cpu
        ')
141 image = image.to(device)
142 output, attention_weights = trained_model(image.unsqueeze(0).to(
        device))
143
144 #print(attention_weights)
145 torch.save(trained_model,'/kaggle/working/
        trained_model_10_Epo_Cartopymk3')
146 #trained_model = torch.load("/kaggle/input/trained/Trained/
        TrainedIC_10_Epo_Analog",map_location=torch.device('cuda')) #cpu
         or cuda
147 #trained_model = torch.load("/kaggle/input/trained/Trained/
        TrainedIC_10_Epo_Cartopy",map_location=torch.device('cuda')) #
```

```
    cpu or cuda
#trained_model = torch.load("/kaggle/input/trained/Trained/
    TrainedIC_10_Epo_Cartopy_mk2",map_location=torch.device('cuda'))
     #cpu or cuda
#trained_model = torch.load("/kaggle/input/trained/Trained/
    TrainedIC_10_Epo_Cartopy_mk3",map_location=torch.device('cuda'))
     #cpu or cuda

trained_model = torch.load("Trained/
    TrainedIC_10_Epo_Cartopy_mk3_wtAtt",map_location=torch.device('
    cpu')) #cpu or cuda
print('is',id2label[dataset['test'][7]['label']],', predicted',
    predict(dataset['test'][7]['image']))
dataset['test'][7]['image']
true_labels = []
predicted_labels = []
datasetname = 'test'
for i in tqdm(range(len(dataset[datasetname])), position=0, leave=
    True):
     true_labels.append(dataset[datasetname][i]['label'])
     predicted_labels.append(label2id[predict(dataset[datasetname][i
        ]['image'])])
confusion_matrix = metrics.confusion_matrix(true_labels,
    predicted_labels)
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix =
    confusion_matrix, display_labels = ['Fishing', 'Non Fishing'])
cm_display.plot()
plt.show()
print(metrics.classification_report(true_labels, predicted_labels))
EVALUATION_resume = pd.DataFrame(list(zip(true_labels,
    predicted_labels)),\
                                 columns =['TRUE_labels', '
                                    PREDICTED_labels'])
EVALUATION_resume.to_csv('EVALUATION_resume.csv')
fpr, tpr, _ = roc_curve(true_labels, predicted_labels)
AUC_Score = roc_auc_score(true_labels, predicted_labels)
print(AUC_Score)
plt.figure(figsize=(20, 15))
colors = [color_1,color_1]
linestyles = ['solid','dotted']
plt.plot(fpr, tpr,linestyle=linestyles[0],color=colors[0],linewidth
    =3.5)

plt.title('ROC Curves')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend()
plt.grid(alpha=0.2, color=color_2)
ax = plt.gca()
ax.set_facecolor(color_bck)
plt.show()
```

List of Figures

List of Tables

List of Acronyms

AIS

Ringraziamenti

In primo luogo, voglio ringraziare *Mark James Carman*, il mio relatore. È un professore che ti fa innamorare di ciò che studi, trovando un equilibrio complesso ma costante tra simpatia, serietà, semplicità e completezza. Mi ha sempre guidato, senza mai impormi decisioni o percorsi, e penso sii la metodologia migliore con cui affrontare un percorso di questo genere. Mi reputo fortunato ad aver potuto redigere questa tesi sotto la sua guida, e sono felice di aver lavorato con estrema serenità apprendendo tutto ciò che potevo da lui.

La magistrale è per molti versi la coronazione di un percorso accademico, la fine di un periodo più o meno lungo che trattiene lo studente tra le mura del Politecnico; il *Poli*, come lo chiamiamo noi...

Ebbene il mio percorso al *Poli* non è sempre stato felice o lineare. Ha visto alti e bassi ma col senno di poi mi sento sicuro nell'affermare che questi sbalzi e vicissitudini lo abbiano reso più ricco, non semplicemente più difficoltoso. Io stesso mi sento una persona più ricca, e penso sii la cosa più importante e bella che si possa dire, salutando un'università. Per questo motivo sono sinceramente convinto che questi ringraziamenti debbano diventare delle vere e proprie dediche verso tutti quelli che mi hanno appoggiato, dandomi man forte con ciò che mi serviva per andare avanti e per affrontare ogni salita, gioendo poi assieme della discesa.

Dedico questa tesi a *Laura* e *Stefano*, i miei genitori. Dentro di me so che non necessitano di presentazioni o spiegazioni, ma il dovere letterario ahimè me lo impone. Posso riassumerli dicendo che sono ciò che io voglio diventare da grande. Io desidero essere un genitore come loro, perché so quanto fortunato ci si sente ad essere loro figlio. Mamma e Papà ci sono sempre stati, non mi hanno mai fatto mancare nulla ed anzi, mi hanno sempre supportato nel poter raggiungere ogni traguardo che mi imponevo. Grazie.

Dedico questa tesi a *Paola* e *Gabriele*, i miei zii. Coinquilini talvolta improvvisati, talvolta semplicemente indispensabili. Mi hanno dato ben più che un semplice tetto. Sono stati la compagnia durante anni di pianti, gioie e soprattutto pandemia, in-

segandomi pian piano "*cosa si fa quando si diventa grandi*" e porgendomi sempre non solo una mano, ma l'intero braccio se serviva un aiuto. Il letto al 4° piano e $\frac{3}{4}$ avrà per sempre un posto (a tratti scomodo) nel mio cuore. *Nota*: la zia non correggerà (anche) questo testo, quindi, è probabile che il lettore debba sopperire ad eventuali errori grammaticali...

Dedico questa tesi ai miei prozii, *Marcella* e *Marino*. Io li definisco nonni d'adozione, ma risulta semplicemente riduttivo. È inutile negare quanto abbia appreso da loro in tutti questi anni, e valutando il bagaglio di nozioni che mi hanno dato, se mi sento ora un *vero ingegnere* è solo grazie a loro.

Dedico questa tesi a *Guido*, *Gabriele*, *Andrea*, *Lorenzo*, *Sofia*, *Gianluca*, e a tutti i miei compagni di viaggio che ogni giorno qui in università mi hanno sopportato, si, con la o, prima ancore che supportato. Mi hanno reso questo viaggio divertente, aspetto fondamentale ma di certo non scontato, e ne sarò sempre grato.

Dedico questa tesi a *Jacopo*, *Alessandro* e *Jonatan*, gli amici di una vita. Mi hanno dimostrato come la mia paura che la distanza potesse rovinare il rapporto di amicizia era del tutto infondata; forse perché in questo caso l'amicizia è semplicemente troppo forte per essere rovinata da qualcosa.

Dedico infine questa tesi a *Margherita*, il mio faro sicuro. Lei è stata la certezza che mi serviva nelle mie giornate. Ha partecipato ad ogni mia gioia e ad ogni mio lamento, non stancandosi mai ed anzi dimostrandomi un amore che tutt'ora non penso di meritare ma che voglio solo ricambiare, giorno dopo giorno passato al suo fianco.

D.M.