POLITECNICO DI TORINO

Master's Degree
in Mathematical Engineering

Master's Degree Thesis

Unraveling Urban Mobility: Gramian Angular Fields for Trajectory Classification

Politecnico di Torino

Consiglio Nazionale delle Ricerche

Supervisors
Prof. Stefano Berrone
Dr. Chiara Ravazzi
Dr. Francesco Malandrino
Dr. Fabrizio Dabbene

Candidate
Elisa Salvadori

Academic Year 2023-2024

Abstract

In recent years, the diffusion of GPS-equipped devices has resulted in the generation of vast amounts of spatio-temporal data. This data represents a fundamental resource to conduct analysis on transportation networks. It is therefore of great interest to identify models capable of distinguishing and classifying trajectories to facilitate decision-making processes. For example, in traffic management, trajectory classification may assist in differentiating between different types of transit modes, aiding in congestion prediction and emissions monitoring. However, many existing algorithms necessitate a complex feature engineering process and domain knowledge. In this context, this thesis proposes a neural network-based approach, which eliminates the need for complicated hand-crafted features, using Gramian angular fields and leveraging possibly pre-trained convolutional neural networks. Therefore, we combine these tools to tackle the challenge of multiclass trajectory classification. We demonstrate the effectiveness of our method on an imbalanced dataset simulated with SUMO by classifying different means of transportation – private car, taxi, bus, pedestrian, motorcycle, bicycle – achieving good results in terms of accuracy and F1 score. Our approach is indeed a viable way to harness the power of convolutional neural networks for the task of trajectory classification.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Trajectories represent a collection of spatio-temporal data ordered in time. In recent years, the widespread diffusion of devices equipped with GPS has led to huge quantities of data being generated and collected. Classifying trajectories has become a crucial task in the field of transportation, allowing for example to estimate traffic density or monitor emissions. Many of the algorithms in the literature are based on complex feature engineering processes, such as the one presented in [Landi et al., 2023a], requiring extensive domain knowledge. Therefore, there is the need to find an approach to classify trajectories that can be easily applied to different cases.

Over the past decade, deep learning architectures like convolutional neural networks have gained significant popularity in the field of computer vision. They have been successfully applied to address problems such as object detection and face recognition, resulting in extensive studies in the field. Therefore, it is simple to find in the literature many neural networks successfully used for different tasks, demonstrating their versatility and power.

The ambition of this thesis is to combine the need to classify trajectories without the use of hand-crafted features with the will of leveraging the power of existing deep learning models. In order to do this, we make use of tools already consolidated in the literature and proven effective in other fields as well. Among these tools we find Gramian angular fields and Markov transition field [Wang and Oates, 2015c], which encode both static and dynamical information of a time series, and the Hilbert's curve [Hilbert, 1891], previously used to analyze animals trajectories [Wang and Oates, 2015b], because of the locality preserving property. These tools can be combined with arbitrary neural networks to tackle classification problems efficiently.

In our research, we assess and validate our approach using a realistic trace of the Principality of Monaco, simulated with SUMO, a tool widely used in the field of telecommunications and urban networks, together with state-of-the-art pre-trained neural networks such as MobileNetV2 and ResNet18. In particular, we examine how to reduce the dimensionality of trajectories in order to employ time series algorithms, and how to handle data with varying lengths by testing two compression algorithms. Furthermore, we analyze how different initialization of neural networks can affect performance. We demonstrate how

this approach is easily implementable and fast, reaching an accuracy of 75% in a transportation mode recognition problem on an imbalanced dataset with six classes: private car, bus, bicycle, pedestrian, taxi and motorcycle.

Our approach offers a practical method for leveraging the capabilities of convolutional neural networks in trajectory classification. Furthermore, our methodology works unmodified for arbitrary convolutional neural networks. Accordingly, we are able to reap the benefits of *future* research in the field, including even better-performing DNNs which have not yet came to light. Last, we mention that our code and implementation are publicly available, making it amenable for comparison with new methodologies as well as for utilization with future neural networks.

The structure of the thesis is as follows: in Chapter 2 we introduce the fundamental concepts relating to trajectories, with a review of existing algorithms used for the analysis and classification of time series, both univariate and multivariate. Furthermore, an overview of the datasets commonly used for trajectory classification, both real and simulated, is provided. Chapter 3 focuses on the theory behind neural networks, explaining how convolutional neural networks work. In Chapter 4 we examine the techniques, known as Gramian angular fields and Markov transition field, for imaging time series, illustrating their adaptability to different domains and underlying mathematical foundations. In Chapter 5 we describe our implemented methodology, by illustrating the process of classifying the generated images, starting from trajectories. Chapter 6 finally concludes with the validation of our method, presenting the used dataset and comparing the results.

Part I

Background

Chapter 2

Time Series Classification Review

This chapter explores the fundamental concepts of time series and trajectories within the context of classification. In Section 2.1, we provide a formal definition of time series, trajectories, and the classification of time series. Understanding these definitions is crucial as they lay the groundwork for our exploration into the classification of trajectory data.

Section 2.2 summarizes the key components and the general process involved in the classification of time series. By a brief review of the main approaches introduced in the literature, we aim to provide a comprehensive understanding of the intricacies involved in effectively classifying time series data.

A critical aspect of this exploration involves recognizing the challenges associated with applying non-tailored algorithms for the classification of time series, and in particular trajectory data. Trajectories, being sequences of spatio-temporal information, exhibit unique characteristics that may be inadequately addressed by generic classification algorithms. We underline the main difficulties in applying such non-specialized approaches, emphasizing the need for tailored methodologies to unlock the full potential of trajectory data analysis.

The chapter concludes with a discussion on prominent open datasets in the field and the rationale behind selecting a specific dataset for our purposes. By scrutinizing available datasets, we aim to align our choice with the objectives of our study, ensuring that the selected dataset best serves our exploration into the classification of trajectory data.

As we proceed, this chapter lays the groundwork for a detailed examination of time series classification, paving the way for subsequent discussions on methodologies, challenges, and practical applications within the realm of trajectory data analysis.

2.1 Preliminaries

In this section, we introduce the fundamental concepts of time series and trajectories, laying the groundwork for the subsequent discussion on how the thesis will specifically

focus on the analysis and classification of trajectory data. We define time series and trajectories, drawing attention to their distinctive characteristics and applications.

Definition 2.1 (Time series)**.** A *time series* $\mathbf{X} = (\mathbf{x_1}, \ldots, \mathbf{x_m}) \in \mathbb{R}^{n,m}$ is a collection of m observations measured sequentially in time.

A time series, as defined in Definition 2.1, is a time-ordered sequence. When $n = 1$ we say that we have a *univariate time series* and each data point x_i is a real number, otherwise when $n > 1$ we say that the time series is *multivariate* and each observation $\mathbf{x_i}$ is a vector in $\mathbb{R}^n$.

Time series analysis has wide-ranging applications in various fields such as finance, economics, climate science, and healthcare. It enables the exploration of temporal patterns, trends, and dependencies within the data.

The following definition introduces the concept of trajectory, which is a sequence of spatio-temporal data sorted by increasing time. Each data point in a trajectory is identified by its latitude and longitude coordinates alongside temporal information. Trajectory data is particularly relevant in the context of movement patterns, such as the trajectory of a moving object over space and time.

Definition 2.2 (Trajectory)**.** A *trajectory* $\mathbf{X} = (\mathbf{x_1}, \ldots, \mathbf{x_m})$ is a sequence of spatio-temporal data $\mathbf{x_i} = (\text{lat}_i, \text{lon}_i, t_i)^\top$, indexed in time order, where $\text{lat}_i, \text{lon}_i, t_i$ represent the latitude, longitude, and timestamp of the data point.

A trajectory is therefore nothing more than a multivariate time series of spatial coordinates $\in \mathbb{R}^2$.

In the context of time series analysis, classification pertains to the assignment of predefined labels or categories to sequences of observations in a time-ordered dataset. Unlike regression, which predicts continuous numerical values, classification involves predicting the discrete class or category to which a given time series belongs.

More formally, we refer to the following definitions.

Definition 2.3 (Time series classification dataset)**.** A *time series dataset* $D = \{(\mathbf{X_i}, y_i), i = 1, \ldots, N\}$ is a set of N time series with the corresponding vector of assigned labels $\mathbf{y}$. Each instance of the dataset is a pair of a time series $\mathbf{x_i}$ and a class label $y_i \in C$, where C is the set of the possible discrete class labels.

Definition 2.4 (Classification problem)**.** Given a classification dataset $D = \{(\mathbf{X_i}, y_i), i = 1, \ldots, N\}$, the task of learning a classifier g, which takes $\mathbf{X_i}$ as input and predicts a qualitative response variable $g(\mathbf{X_i})$ (i.e. predicts the class), is called a *classification problem*.

Trajectory classification is of paramount importance in transportation applications. Identifying patterns in transportation trajectories and categorizing them into discrete classes facilitates decision-making processes. For example, in traffic management, trajectory classification may assist in distinguishing between different types of vehicle movements, aiding in congestion prediction and route optimization. In public transportation, it could contribute to the identification of transit modes or the characterization of travel behavior.

Figure 2.1 shows a simplified version of a binary classification problem i.e. the classification involves only two classes, where the objective is to predict the mode of transportation of the unknown trajectory (in red), given the other labeled trajectories.

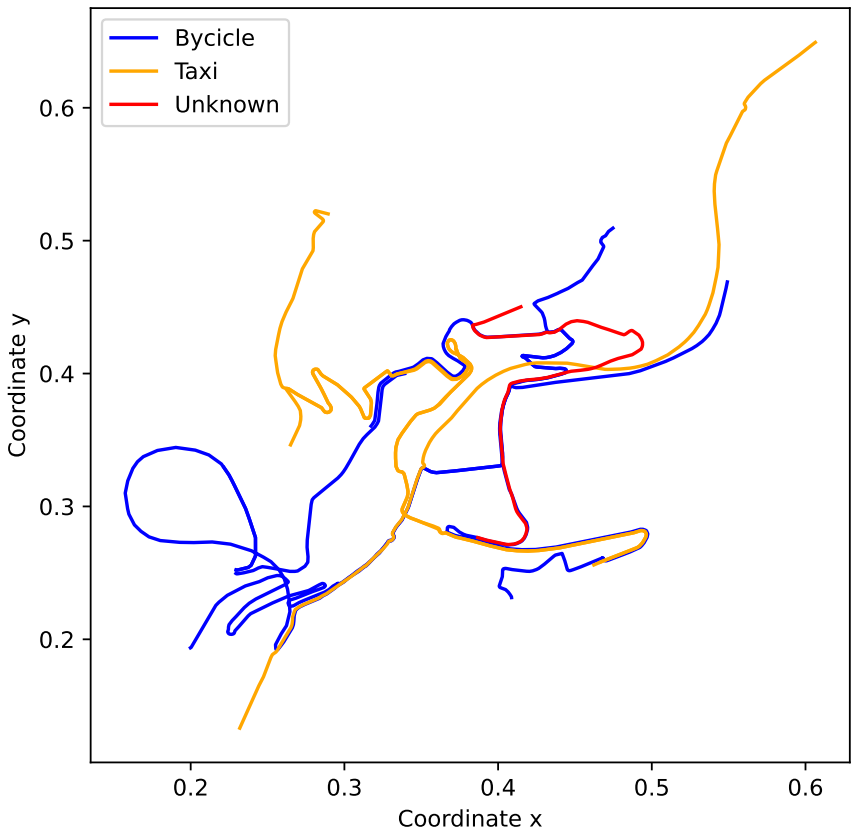

Figure 2.1: Example of a classification of trajectories.

2.2 Classification of Time Series

Conventional classification algorithms encounter challenges when applied to time series data due to the intricate temporal structure characterized by ordered features. To effectively address Time Series Classification (TSC) problems, modifications are imperative.

Similar to [Middlehurst et al., 2023, Ruiz et al., 2020], we present a taxonomy that categorizes classification algorithms into subgroups based on their employed feature extraction techniques. This taxonomy serves as a valuable framework for understanding and navigating the diverse landscape of time series classification methodologies.

Whole series classifiers Whole series classifiers rely their performance on a similarity measure assessing the distance between two given time series. Dynamic Time Warping (DTW) [Sakoe and Chiba, 1978] stands out in this category, frequently employed with the One-Nearest Neighbors (1-NN)[1] classifier as a benchmark in TSC problems. These distances, often referred to as elastic, possess a unique capability to account for misalignments between time series. They can compensate for shifts in the time axis by not rigidly comparing series point by point. This flexibility makes them particularly well-suited for comparing temporally ordered data. Over the years a lot of variations have been proposed, including those aimed at enhancing performance [Jeong et al., 2011, Cuturi and Blondel, 2017] or the approach developed to address the challenges associated with handling multivariate time series [Shokoohi-Yekta et al., 2017].

Subsequence based classifiers Subsequence based classifiers, rather than using the entire time series, focus on portions of it. This category encompasses four distinct approaches: interval based classifiers, shapelet based classifiers, dictionary based classifiers and finally convolution based classifiers. *Interval based classifiers* divide the time series into intervals, from which discriminatory features are extracted. For instance, summary statistics are computed like mean, standard deviation and slope [Deng et al., 2013]. A popular approach belonging to this group, also extended to multivariate time series classification, was introduced in [Middlehurst et al., 2020]. The core concept of *shapelet based classifiers* [Ye and Keogh, 2011] lies on the presence, or absence, of distinctive phase independent patterns i.e. distinctive subsequences could be found at any point in the time series. On the other hand, *dictionary based classifiers* utilize the frequency of subsequences to distinguish the classes, these classifiers rely on a process which approximates and transforms the time series into words and counts their frequencies. Examples within this category icnlude Bag of SFA Symbols (BOSS) [Schäfer, 2015]. Furthermore, *convolution based classifiers* derive discriminatory features via convolutions of the time series with chosen kernels, each kernel is convolved with the time series using a sliding dot product. The most popular algorithm in this category is the Random Convolutional Kernel Transform (ROCKET) [Dempster et al., 2020].

Deep learning based classifiers Over the past decade, deep learning has exploded in popularity, achieving notable successes in fields such as computer vision, natural language processing and speech recognition (see also Chapter 3). The adoption of Convolutional Neural Networks (CNNs) has increased since the breakthrough success of AlexNet [Krizhevsky et al., 2012], which won the ImageNet competition in 2012. Increasing data availability and the rise of Graphical Processing Units (GPUs) have made it possible to train Deep Neural Networks (DNNs) and learn hidden discriminative features in time series. Many studies have extended deep learning techniques to time series analysis. Adopting the terminology of [Fawaz et al., 2018], it is possible to distinguish two main approaches: end-to-end ones and those based on feature extraction such as image based

[1] Classification, using the k-nearest neighbours algorithm, relies on the majority class of the k nearest neighbours.

classifiers. *End-to-end* approaches involve the direct use of raw time series without any feature engineering process. For example, in [Wang et al., 2017], the authors have successfully adapted a Multi-Layer Perpetron (MLP), a Fully Convolutional Network (FCN) and a Residual Network (ResNet) to propose a deep learning baseline for TSC problems. In [Ismail Fawaz et al., 2020], InceptionTime was proposed, an ensemble of five deep learning classifiers built specifically to handle time series data. Moreover TapNet [Zhang et al., 2020] was designed for multivariate TSC problems. For the purposes of this thesis, we mainly focus on image based classifiers, since they offer more flexibility, allowing the use of existing DNNs, regardless the domain and the specific task. In particular, we will explore these techniques as a way to better exploit the capabilities of neural networks in time series analysis. *Image based classifies* rely on imaging time series, Gramian angular fields and Markov transition field, originally proposed in [Wang and Oates, 2015c], are used to retain both static and dynamical information. The static aspect is represented using a Gram matrix, while the dynamical one is captured by discretizing the time series into quantile bins and counting the relative frequency of the transitions from a quantile bin to another. This category is more detailed in Chapter 4.

Hybrid classifiers Into this category fall those classifiers which don't belong to only one category and employ hybrid approaches. An algorithm which combines multiple feature extraction techniques is the Collective of Transformation Ensembles (COTE) [Bagnall et al., 2015].

2.3 Classification of Trajectories

Due to the diffusion of GPS (Global Positioning System)-equipped devices, like mobile phones or vehicles, a huge volume of spatio-temporal data are being generated and collected every day.

This data represents a fundamental resource for researchers and industries to conduct analysis on transportation network. It is therefore of great interest to identify models capable of distinguishing and classifying trajectories. Some examples of trajectory classification include transportation mode recognition, such as car, bus or train, or the identification of the user which generated the trajectory. Determining the mode of transportation can be crucial for a transportation agency to monitor emissions or for traffic density estimation.

Trajectory classification methods often utilize either global features extracted from the entire trajectory, or local features derived from subtrajectories. However, it has been proven that these approaches have some weaknesses: global features such as speed, acceleration, etc. present a high sensitivity to traffic congestion [Sun and Ban, 2013], while local features could be really specific to a task, not generalizing very well [Leite da Silva et al., 2019].

In recent years, new tailored algorithms have been proposed to handle trajectory classification problems, from those that exploit computer vision models to obtain better results, such as the ones presented in [Dabiri and Heaslip, 2018, Kontopoulos et al., 2023], to Geolet [Landi et al., 2023b], a new shapelet-based interpretable machine learning model.

In [Wang and Oates, 2015b], the authors adapted GAFs + MTF, an approach originally proposed for time series classification, to spatio-temporal data, by using Hilbert's curve. Unfortunately, they tested the method on very small datasets, which produced poor results and were prone to overfitting, however they suggested the extension of their approach to a variety of time series data, including spatio-temporal one.

In this thesis, we adapt GAFS, to the domain of vehicular trajectories, in order to find a method which is not case specific and which could be generalized to different trajectory classification problems, without the use of complicated hand-crafted features.

2.4 Open Datasets

The availability of trajectory data is limited, while many studies rely on private datasets, others use small public one, making comparisons between algorithms difficult. Additionally, public real-world datasets, due to possible sensor errors, require a careful cleaning step or a filling strategy to estimate the lack of information. Among the real-world open datasets available, we find:

- GeoLife[2]: A dataset frequently used for transportation mode recognition, comprised by daily trajectories of 182 users.
- Taxi[3]: A dataset composed by the trajectories of 442 taxis running in the city of Porto (Portugal).

To overcome these limitations, and since neural networks require a large volume of data to train effectively, realistic simulated datasets can be adopted. In particular, SUMO (Simulation of Urban Mobility) can be a useful tool to generate synthetic datasets that reproduce specific scenarios, with a possible sampling rate of 0.01 s.

[2] https://www.microsoft.com/en-us/download/details.aspx?id=52367

[3] https://www.kaggle.com/datasets/crailtap/taxi-trajectory

Chapter 3

Neural Networks

In this chapter, we investigate the core concepts behind neural networks, their architectures, and the mechanisms by which they learn and make predictions.

Deep learning has become increasingly popular in the field of machine learning, thanks to the increase of computational power and the larger availability of data. Artificial neural networks are deep learning models inspired by biological brain. The first artificial neuron was invented with the intent of simulating the behaviour of biological neurons [McCulloch and Pitts, 1943]. Deep feedforward neural networks, often called multi-layer perceptrons are built by stacking layers of multiple neurons. MLPs are organized into an input layer, one or multiple hidden layers, and then the output layer. Each neuron in the input layer corresponds to an input feature (the variables which we are able to observe), while those in subsequent layers perform computations on these inputs to generate output (their values are not given in the data). Figure 3.1 shows an example of multi-layer perceptron with two hidden layers, each comprised of 3 neurons.

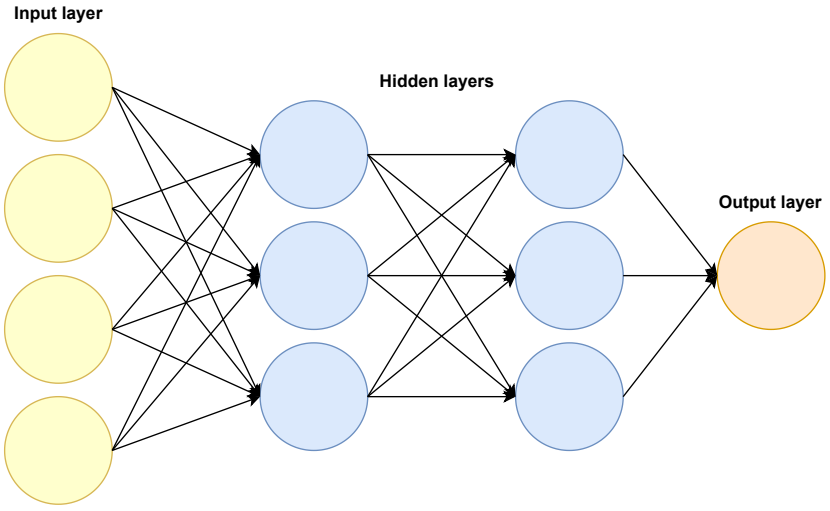

Figure 3.1: A simple deep feedforward neural network with two hidden layers.

These models are called feedforward because during the forward propagation phase, input data are fed into the network, and computation are performed layer by layer to generate predictions i.e. information flows forward through the network. The values of the hidden layers are computed with an activation function, allowing the network to learn highly complex mappings between inputs and outputs. Each neuron receives inputs, applies its activation function and then passes the produced output to the next layer. Artificial neurons of the hidden layers, also called hidden units, rely on two operations:

- Preactivation: it consists in computing the weighted sum of the input signals.
- Activation: after the preactivation, the output is computed through the activation function.

In other words, given the input vector $\mathbf{x}$, the matrix of the weights $\mathbf{W}$, and the bias vector $\mathbf{b}$, an affine transformation $\mathbf{z} = \mathbf{W}^\top \mathbf{x} + \mathbf{b}$ is computed and then the activation function $g(\mathbf{z})$ is applied element-wise. The activation function introduces non-linearity into the network's computations. Popular activation functions include the sigmoid, hyperbolic tangent (tanh), and Rectified Linear Unit (ReLU). We have examined in details the forward propagation phase. In order to asses the performance of a neural network a loss function is used to measure the accuracy of the predicted response variable with respect to the actual one. The common loss-function for classification tasks is the cross entropy loss. Finally, we have to introduce the back-propagation algorithm to understand how neural networks work. The back-propagation is the cornerstone algorithm for training neural networks, it allows the information to flow backwards through the network from the computed loss function, in order to compute the gradient. This process relies on the chain rule from calculus to compute the gradient of the loss function with respect to the network parameters. Then, optimization algorithms, such as stochastic gradient descent or Adam, use the gradient to update the network's weights, minimizing the loss function.

The process of forward propagation, loss calculation, and back-propagation is repeated iteratively for multiple epochs until the model converges to a satisfactory level of performance.

3.1 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a type of deep learning algorithms designed to recognize patterns within data with a grid-like topology, such as images. They are widely adopted in computer vision tasks like image classification, object detection, and image segmentation. Typically, the input to a CNN is a multi-dimensional array representing an image (height $\times$ width $\times$ channels), where each element corresponds to the intensity value of a pixel. Convolutional layers extract features from the input image, such as edges. Subsequently, pooling layers downsample the extracted feature maps, resulting in a lower computational complexity. After convolution and pooling operations, the output is flattened into a one-dimensional array, ready to be fed into fully connected layers. Then, the flattened feature vector is processed by the fully connected layers in order to make predictions.

Therefore, three types of layers compose the CNN architecture.

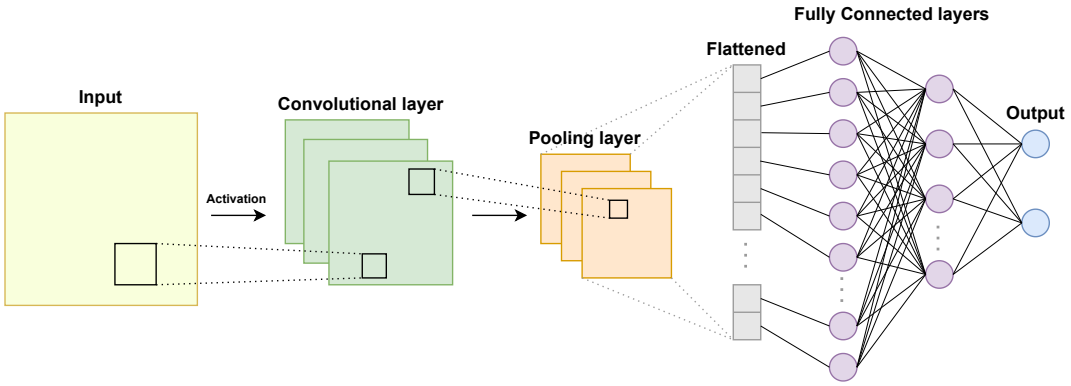

Figure 3.2: Architecture of a convolutional neural network.

- Convolutional layers: a single convolutional layer consist in a set of learnable filters, also known as kernels, which slide over the input data (the image) to produce the output. Each kernel is usually smaller than the input image, resulting in simply a weighted sum over a smaller region. If the kernel does not go outside the image we have a "valid" convolution[1], otherwise the border of the input can be padded in some ways. The output of the convolution operation is a feature map, which represents the presence of specific features within the image, each kernel extracts different features from the input data. Before passing the output to the pooling layer an activation function is applied, commonly the Rectified Linear Unit (ReLU).

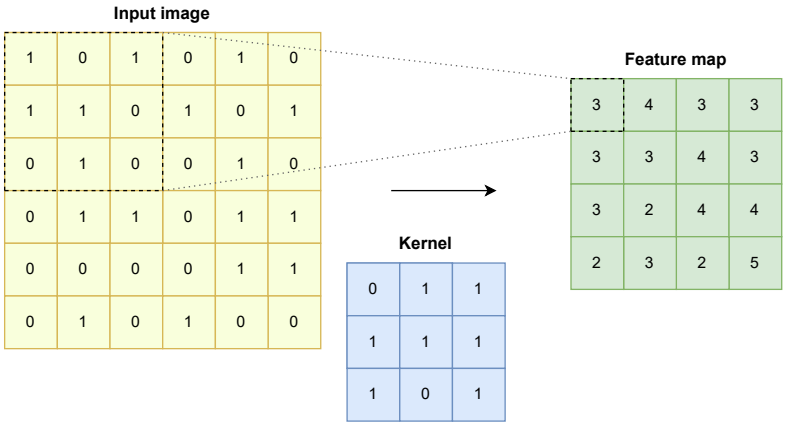

Figure 3.3: Valid convolution. The output is computed by pointwise multiplying the 3×3 kernel weights with the 3×3 regions of the input image and summing.

[1]However, many machine learning libraries implement the cross-correlation instead of the convolution operation [Goodfellow et al., 2016, Prince, 2023].

- Pooling layers: these layers perform pooling operations, in order to reduce the spatial dimension of the feature maps, while retaining the most important information. Max pooling and average pooling are common pooling operations, these operations consist in downsampling the feature maps by selecting the maximum or average value within a specific region.

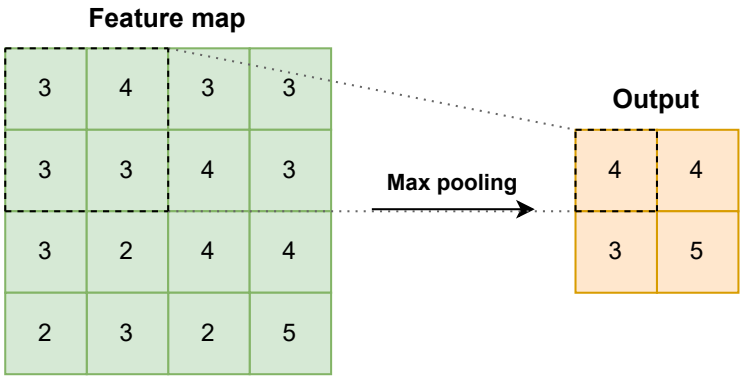

Figure 3.4: Max pooling.

- Fully connected layers: they are typically placed at the end of the CNN architecture. As the name suggests, every neuron in one layer is connected to every neuron in the next layer. They take the flattened output from the preceding layers and perform classification or regression tasks.

As said before, convolutional neural networks are widely used and proven effective for addressing various problems in the field of computer vision. Two popular architectures are MobileNet [Howard et al., 2017] and ResNet [He et al., 2016], the first one is a small model originally proposed for mobile vision applications, which uses depthwise separable convolutions. On the other hand, ResNet was created to make the training of deep neural network easier. For the aims fo this thesis we want to exploit the power of these architectures by extending it to the context of vehicular trajectories.

Chapter 4

Imaging Time Series

In this chapter, our focus is on the applications and theory of Gramian angular fields and Markov transition field.

In Section 4.1, we explore the primary applications of these techniques in the literature, to investigate their adaptable nature in various fields.

Subsequently, in Section 4.2 and Section 4.3, we explore the theory underlying these approaches, offering a detailed overview of how these images are generated from time series. Additionally, we highlight the properties of these matrices using practical and easily understandable examples, aiming for a deeper and more intuitive comprehension of the discussed methodologies.

4.1 Related Work

In [Wang and Oates, 2015a], a novel methodology was proposed to encode time series as images by converting them into a matrix representation, enabling the use of computer vision models for imputation and classification. The authors demonstrated the effectiveness of their technique for both univariate and multivariate time series classification [Wang and Oates, 2015b]. They paired Gramian Angular Fields (GAFs) with Markov Transition Field (MTF) to create three-channel images (RGB) and fed convolutional neural networks.

Since then, GAFs, MTF or the combination of the two, have been widely used to carry out different tasks, across various domains, showcasing their versatility and efficacy. In the medical field, they have been employed for detecting myocardial infarction risk from electrocardiogram (ECG) signals [Zhang et al., 2019] and motor imagery recognition from electroencephalogram (EEG) signals [Bragin and V.G., 2019]. Moreover, their application extends to forecasting day-ahead solar irradiation [Hong et al., 2020], market financial forecasting [Barra et al., 2020] and fault diagnosis [Li et al., 2020, Han et al., 2021]. Beyond these applications, they have proven effective in classifying anomalous diffusion trajectories [Garibo i Orts et al., 2023], in the analysis of autonomous vehicle (AV) driving behavior [You et al., 2023] and imputation of traffic data [Huang et al., 2023]. Nevertheless, as far as we know, this represents the first application of GAFs and MTF in the classification of vehicular trajectories to recognise different means of transportation.

4.2 Gramian Angular Fields

Gramian angular fields derive from the Gram matrix, so now we have to introduce the concept of Gram matrix, which paves the way for the new approach based on a "quasi" Gramian matrix (the term "quasi" is used because the inner products of the GAFs do not satisfy all the properties of inner products).

4.2.1 Gram Matrix

Definition 4.1 (Gram matrix)**.** Given a matrix $\mathbf{X} \in \mathbb{C}^{n,m}$, then the *Gram matrix* $\mathbf{G} \in \mathbb{C}^{m,m}$ is defined as

$$\mathbf{G} = \mathbf{X}^{\dagger}\mathbf{X}, \tag{4.1}$$

where each entry $g_{ij} = \bar{\mathbf{x}}_\mathbf{i}^\top \mathbf{x_j}$ is the standard inner product[1] on $\mathbb{C}^n$ between column j and column i of the original matrix $\mathbf{X}$. Since we only consider the case where $\mathbf{X} \in \mathbb{R}^{n,m}$, then the Gram matrix reduces to $\mathbf{G} = \mathbf{X}^\top\mathbf{X}$ [2] [Lay et al., 2016].

Remark 4.1. The above definition can be written as a function $\Phi : \mathbb{R}^{n,m} \to \mathbb{R}^{m,m}$ whose image's elements are symmetric positive semidefinite matrices. It can be proven that this function is not a linear transformation. We recall the definition of a linear transformation.

Definition 4.2 (Linear transformation). Given two vector spaces V, W over the field $\mathbb{F}$, a function $f : V \to W$ is called a *linear transformation*, if the following properties hold:

1. Additivity: $f(\mathbf{x}+\mathbf{y}) = f(\mathbf{x}) + f(\mathbf{y})$, for all $\mathbf{x}, \mathbf{y} \in V$.
2. Homogeneity: $f(\alpha\mathbf{x}) = \alpha f(\mathbf{x})$, for all $\mathbf{x} \in V, \alpha \in \mathbb{F}$.

But, we have:

1.

$$\begin{aligned}\Phi(\mathbf{A}+\mathbf{B}) &= (\mathbf{A}+\mathbf{B})^\top(\mathbf{A}+\mathbf{B}) \\ &= (\mathbf{A}^\top + \mathbf{B}^\top)(\mathbf{A}+\mathbf{B}) \\ &= \mathbf{A}^\top\mathbf{A} + \mathbf{B}^\top\mathbf{B} + \mathbf{A}^\top\mathbf{B} + \mathbf{B}^\top\mathbf{A} \\ &\neq \mathbf{A}^\top\mathbf{A} + \mathbf{B}^\top\mathbf{B} = \Phi(\mathbf{A}) + \Phi(\mathbf{B}).\end{aligned}$$

2.

$$\begin{aligned}\Phi(\alpha\mathbf{A}) &= (\alpha\mathbf{A})^\top(\alpha\mathbf{A}) \\ &= \alpha^2\mathbf{A}^\top\mathbf{A} \\ &\neq \alpha\mathbf{A}^\top\mathbf{A} = \alpha\Phi(\mathbf{A}), \text{with } \alpha \in \mathbb{R}.\end{aligned}$$

[1] The standard inner product on $\mathbb{C}^n$ is the function $\langle\cdot,\cdot\rangle : \mathbb{C}^n \times \mathbb{C}^n \to \mathbb{C}$ defined by $\langle \mathbf{x}, \mathbf{y}\rangle = \sum_{k=1}^n x_n\bar{y}_n$ [Rynne and Youngson, 2000].

[2] The standard inner product on $\mathbb{R}^n$ (also referred as dot product) is the function $\langle\cdot,\cdot\rangle : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ defined by $\langle \mathbf{x}, \mathbf{y}\rangle = \sum_{k=1}^n x_n y_n$ [Rynne and Youngson, 2000].

As stated before, Gram matrices are symmetric positive semidefinite matrices. The Gram matrix $\mathbf{G}$ is symmetric by definition, because of the symmetry of the standard inner product on $\mathbb{R}^n$, now we prove that it is also positive semidefinite[3]. Given $\mathbf{X} \in \mathbb{R}^{n,m}$ and $\mathbf{v} \in \mathbb{R}^m$, then

$$\mathbf{v}^\top \mathbf{G}\mathbf{v} = \mathbf{v}^\top \mathbf{X}^\top \mathbf{X}\mathbf{v} = (\mathbf{X}\mathbf{v})^\top (\mathbf{X}\mathbf{v}) = \langle \mathbf{X}\mathbf{v}, \mathbf{X}\mathbf{v} \rangle \geq 0.$$

Remark 4.2. From a geometric point of view, given two vectors $\mathbf{x} = (x_1, \ldots, x_d)^\top$ and $\mathbf{y} = (y_1, \ldots, y_d)^\top$ in $\mathbb{R}^2$ or $\mathbb{R}^3$, then the dot product $\langle \mathbf{x}, \mathbf{y} \rangle = \sum_{k=1}^{d} x_k y_k$ satisfies

$$\langle \mathbf{x}, \mathbf{y} \rangle = \|\mathbf{x}\| \, \|\mathbf{y}\| \cos\theta,$$

where θ is the angle (expressed in radians) between them, and $\|x\| = \sqrt{\langle \mathbf{x}, \mathbf{x} \rangle}$, $\|y\| = \sqrt{\langle \mathbf{y}, \mathbf{y} \rangle}$ are the lengths of $\mathbf{x}$ and $\mathbf{y}$ respectively. If $\mathbf{x}$ and $\mathbf{y}$ are unit vectors, then $\langle \mathbf{x}, \mathbf{y} \rangle = \cos\theta$.

Taken $\mathbf{X} \in \mathbb{R}^{2,m}$, if each column of $\mathbf{X}$ has been normalized, then the entry g_{ij} of the Gram matrix reduces to the cosine of the angle between $\mathbf{x_i}$ and $\mathbf{x_j}$, and it measures how similar the two columns are, obtaining

$$\mathbf{G} = \begin{pmatrix} \cos\theta_{1,1} & \cos\theta_{1,2} & \ldots & \cos\theta_{1,m} \\ \vdots & \vdots & \ddots & \vdots \\ \cos\theta_{m,1} & \cos\theta_{m,2} & \ldots & \cos\theta_{m,m} \end{pmatrix},$$

where a value of $\cos\theta = 0$ means that the two column vectors are orthogonal.

Now, focusing on time series analysis, we get deeper in the theory of GAFs.

4.2.2 Gramian Angular Fields Construction

Given a univariate time series $\mathbf{x} = (x_1, \ldots, x_m)$, which takes values on real numbers, it can be rescaled to the range $[0, 1]$ or $[-1, 1]$ in order to have the same order of magnitude for all the observations and to not have a biased scalar product in favor of the greater value.

We have

$$\begin{cases} x_i^* = \dfrac{x_i - x_{\min}}{x_{\max} - x_{\min}} & \text{if the target interval is [0,1]} \\ x_i^* = \dfrac{2x_i - (x_{\min} + x_{\max})}{x_{\max} - x_{\min}} & \text{otherwise,} \end{cases}$$

where $x_{\min}$ and $x_{\max}$ are respectively the minimum and the maximum value of the time series $\mathbf{x}$.

Example 4.1. From now on, we take the time series of the cosine function $x_i = \cos t_i$ for demonstration purposes (Figure 4.1).

[3] A symmetric matrix $\mathbf{X} \in \mathbb{R}^{n,n}$ such that $\mathbf{v}^\top \mathbf{A}\mathbf{v} \geq 0$ for every $\mathbf{v} \in \mathbb{R}^n$, is said to be positive semidefinite [Lay et al., 2016].

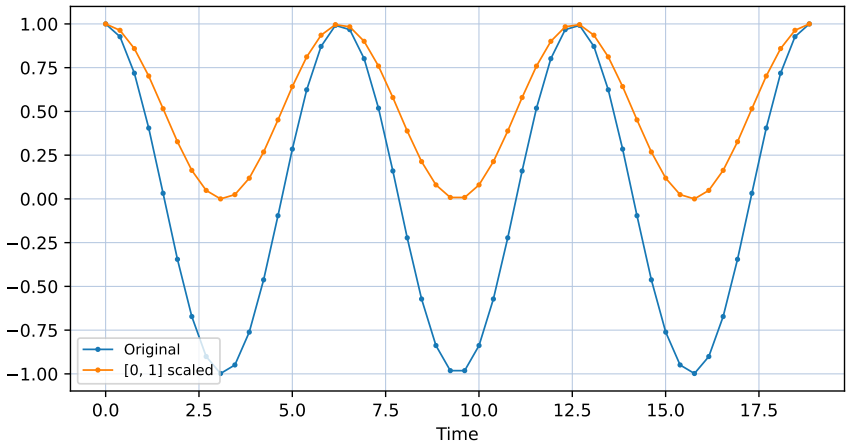

Figure 4.1: Cosine time series $x_i = \cos t_i$ before and after the rescaling to the range [0,1].

The Gram matrix can be now computed, but it is reasonably to observe that since the time series is univariate, the resulting inner product and Gram matrix is simply the product between two real numbers, in details we have

$$\mathbf{G} = \begin{pmatrix} x_1 \cdot x_1 & x_1 \cdot x_2 & \dots & x_1 \cdot x_m \\ \vdots & \vdots & \ddots & \vdots \\ x_m \cdot x_1 & x_m \cdot x_2 & \dots & x_m \cdot x_m \end{pmatrix}.$$

We can easily show that, if the time series is periodic, then the Gram matrix shows the same periodic pattern by rows and by columns.

Example 4.2. Given a time series $\mathbf{x} = (x_1, x_2, \dots, x_{m-1}, x_m) = (x_1, x_2, \dots, x_{m/2}, x_1, x_2, \dots, x_{m/2})$, composed by a time series which repeats itself after $m/2$ observations (period equal to $m/2$), then

$$\mathbf{G} = \begin{pmatrix} x_1 \cdot x_1 & x_1 \cdot x_2 & \dots & x_1 \cdot x_{m/2} & x_1 \cdot x_1 & x_1 \cdot x_2 & \dots & x_1 \cdot x_{m/2} \\ x_2 \cdot x_1 & x_2 \cdot x_2 & \dots & x_2 \cdot x_{m/2} & x_2 \cdot x_1 & x_2 \cdot x_2 & \dots & x_2 \cdot x_{m/2} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{m/2} \cdot x_1 & x_{m/2} \cdot x_2 & \dots & x_{m/2} \cdot x_{m/2} & x_{m/2} \cdot x_1 & x_{m/2} \cdot x_2 & \dots & x_{m/2} \cdot x_{m/2} \\ x_1 \cdot x_1 & x_1 \cdot x_2 & \dots & x_1 \cdot x_{m/2} & x_1 \cdot x_1 & x_1 \cdot x_2 & \dots & x_1 \cdot x_{m/2} \\ x_2 \cdot x_1 & x_2 \cdot x_2 & \dots & x_2 \cdot x_{m/2} & x_2 \cdot x_1 & x_2 \cdot x_2 & \dots & x_2 \cdot x_{m/2} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{m/2} \cdot x_1 & x_{m/2} \cdot x_2 & \dots & x_{m/2} \cdot x_{m/2} & x_{m/2} \cdot x_1 & x_{m/2} \cdot x_2 & \dots & x_{m/2} \cdot x_{m/2} \end{pmatrix},$$

and it is clear the repetition of the smaller $\frac{m}{2} \times \frac{m}{2}$ matrix. This consideration is evident in Figure 4.2.

Example 4.3. For the sake of visualization, we plot the Gram matrix of the cosine time series and of the cosine time series rescaled to [0,1] (see Figure 4.2) .

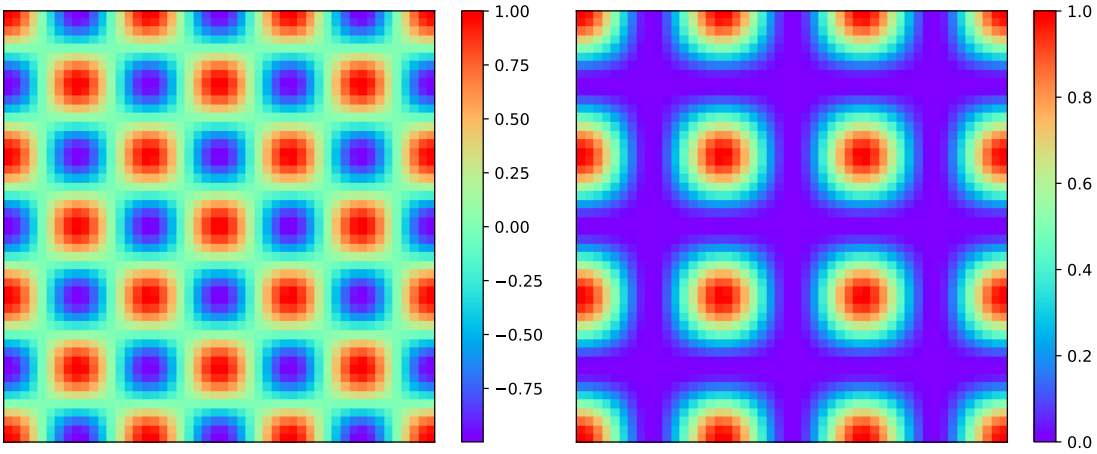

Figure 4.2: Gram matrix computed from the cosine time series and from the cosine time series rescaled to the range [0, 1].

The time series $\mathbf{x}$ is now transformed into another coordinate system in order to exploit the angular perspective.

Definition 4.3. We define a function f as

$$\begin{aligned} f : \mathbb{N} \times [-1, 1] \subset \mathbb{N} \times \mathbb{R} &\longrightarrow \mathbb{R} \times \mathbb{R} \\ (t, x) &\longmapsto (r, \phi), \end{aligned}$$

in this polar coordinate system the value x_i is encoded as the angular cosine and the time stamp t_i as the radius, such as

$$\begin{cases} \phi_i = \arccos(x_i) \\ r_i = \dfrac{t_i}{N}, \end{cases} \tag{4.2}$$

where N is a regularization factor. Without loss of generality we assume that the time stamp of the time series is in the interval $[0, m]$, so the last value m could be used as the constant factor N to scale the radius to the range $[0, 1]$, while the range for the angle is $[0, \pi/2]$ if x_i is in $[0, 1]$ or $[0, \pi]$ if x_i is in $[-1, 1]$.

Remark 4.3. The function $f_1(x) = \cos x$ is strictly decreasing on the interval $[0, \pi]$ and when it is restricted to that interval, the arccosine function is the inverse of it. The function f is therefore a bijection from $\mathbb{N} \times [-1, 1]$ to $[0, 1] \times [0, \pi]$.

Example 4.4. We show the transformation in the new polar coordinate system in Figure 4.3.

With this transformation we can introduce new "inner products" which leverages the angular perspective by considering the trigonometric sum or difference between two points.

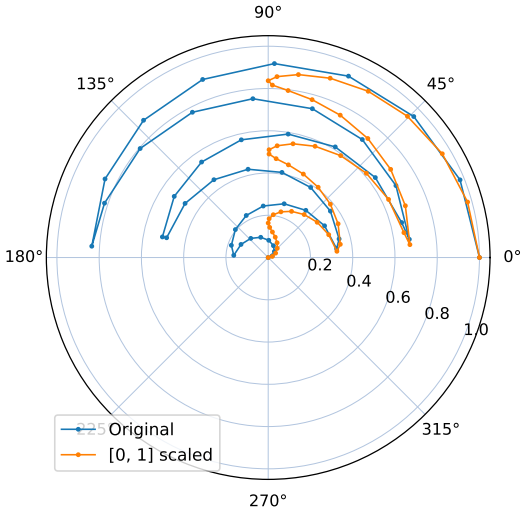

Figure 4.3: Cosine time series before and after the rescaling to the range [0,1], after the transformation in polar coordinates.

Definition 4.4 (Summation inner product)**.** The function $\langle x, y\rangle_s : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ defined by

$$\begin{aligned}\langle x, y\rangle_s &= \cos(\phi_x + \phi_y) = \cos(\arccos x + \arccos y)\\ &= \cos(\arccos x) \cdot \cos(\arccos y) - \sin(\arccos x) \cdot \sin(\arccos y)\\ &= x \cdot y - \sqrt{1 - \cos^2(\arccos x)} \cdot \sqrt{1 - \cos^2(\arccos y)}\\ &= x \cdot y - \sqrt{1 - x^2} \cdot \sqrt{1 - y^2},\end{aligned}$$

is the inner product for the Gramian summation angular field.

Definition 4.5 (Difference inner product)**.** The function $\langle x, y\rangle_d : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ defined by

$$\begin{aligned}\langle x, y\rangle_d &= \sin(\phi_x - \phi_y) = \sin(\arccos x - \arccos y)\\ &= \sin(\arccos x) \cdot \cos(\arccos y) - \cos(\arccos x) \cdot \sin(\arccos y)\\ &= \sqrt{1 - \cos^2(\arccos x)} \cdot y - x \cdot \sqrt{1 - \cos^2(\arccos y)}\\ &= \sqrt{1 - x^2} \cdot y - x \cdot \sqrt{1 - y^2},\end{aligned}$$

is the inner product for the Gramian difference angular field.

We recall the definition of inner product ([Rynne and Youngson, 2000]).

Definition 4.6 (Inner product)**.** Let X be a vector space over $\mathbb{R}$, then a function $\langle \cdot, \cdot \rangle : X \times X \to \mathbb{R}$ is said to be an *inner product on* X if the following conditions hold for all $\mathbf{x}, \mathbf{y}, \mathbf{z} \in X$ and $\alpha, \beta \in \mathbb{R}$:

1. $\langle \mathbf{x}, \mathbf{x} \rangle \geq 0$;
2. $\langle \mathbf{x}, \mathbf{x} \rangle = 0$ if and only if $\mathbf{x} = 0$;
3. $\langle \alpha\mathbf{x} + \beta\mathbf{y}, \mathbf{z} \rangle = \alpha\langle \mathbf{x}, \mathbf{z} \rangle + \beta\langle \mathbf{y}, \mathbf{z} \rangle$;
4. $\langle \mathbf{x}, \mathbf{y} \rangle = \langle \mathbf{y}, \mathbf{x} \rangle$.

It is easy to see that the function $\langle \cdot, \cdot \rangle_s$ is not an inner product on $\mathbb{R}$ (the same consideration also applies to $\langle \cdot, \cdot \rangle_d$).

The resulting matrices are the Gramian Summation Angular Field (GASF) and the Gramian Difference Angular Field (GADF):

$$\mathbf{GASF} = \begin{pmatrix} \cos(\phi_1 + \phi_1) & \cos(\phi_1 + \phi_2) & \dots & \cos(\phi_1 + \phi_m) \\ \vdots & \vdots & \ddots & \vdots \\ \cos(\phi_m + \phi_1) & \cos(\phi_m + \phi_2) & \dots & \cos(\phi_m + \phi_m) \end{pmatrix}; \tag{4.3}$$

$$\mathbf{GADF} = \begin{pmatrix} \sin(\phi_1 - \phi_1) & \sin(\phi_1 - \phi_2) & \dots & \sin(\phi_1 - \phi_m) \\ \vdots & \vdots & \ddots & \vdots \\ \sin(\phi_m - \phi_1) & \sin(\phi_m - \phi_2) & \dots & \sin(\phi_m - \phi_m) \end{pmatrix}. \tag{4.4}$$

Example 4.5. In Figure 4.4 we can see the GASF and the GADF computed from the cosine time series rescaled to the range $[0, 1]$.

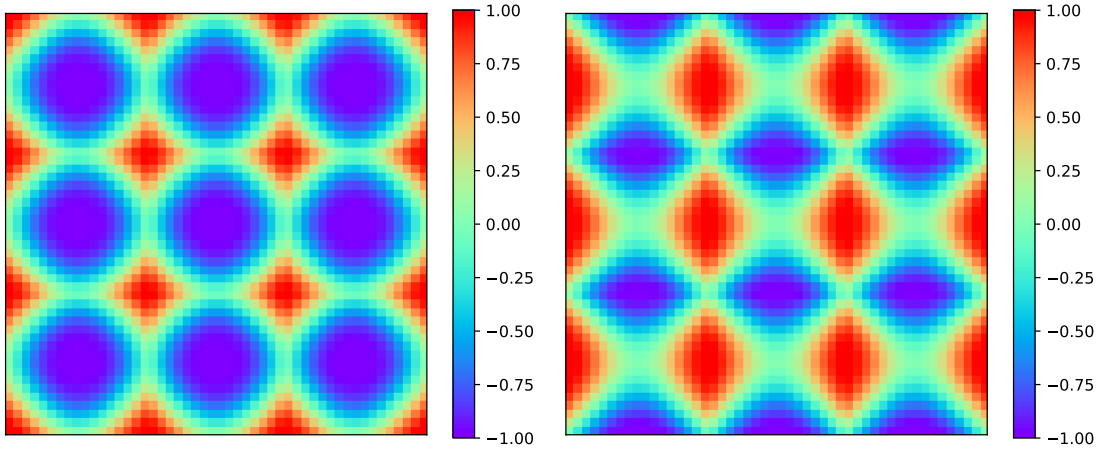

Figure 4.4: Gramian summation angular field and gramian difference angular field computed from the cosine time series rescaled to the range $[0, 1]$.

Remark 4.4. The GAFs provide a useful technique for time series, but we need to analyse the benefits and limitations

- Benefits:

  - The GAFs maintains the temporal information in the matrix, since time augments from the upper-left to the bottom-right and the entry $g_{ij||j-i|=k}$ represents the relation between points with respect to time interval equal to k.
  - The main diagonal of the GASF ($g_{ij||j-i|=0}$) contains the data which allow us to reconstruct the original time series. When the angle ϕ is in $[0, \pi]$ then 2ϕ is in $[0, 2\pi]$ and the inverse of $\cos(2\phi)$ is not unique at the endpoints of the interval $[0, 2\pi]$, reasonably because the function $\cos(2\phi)$ is monotonic only for values of ϕ in the range $[0, \pi/2]$.

    So this ambiguity does not arise if ϕ is in $[0, \pi/2]$, and we can recover the time series by

    $$\cos(\phi) = \sqrt{\frac{\cos(2\phi) + 1}{2}}, \quad \phi \in [0, \pi/2];$$

    in other words, we can recover the original values only for the time series rescaled to the range $[0, 1]$.

    We can not come to the same conclusion with the GADF, because by definition the elements of the main diagonal are all equal to 0 and they do not add any distinctive information to reconstruct the original series.

- Limitations:
  - With the GAFs we transform a univariate time series into a matrix, so the size increases from m to $m \times m$.
  - In addition we can notice that, if $\mathbf{x}$ is a univariate time series, and $\alpha\mathbf{x}$ is the same time series multiplied by α then they produce different Gram matrix, but when they are rescaled to the range [0,1] or $[-1, 1]$ there is no difference between the two time series and so they produce the same Gram matrix.

    In the case of study, when dealing with vehicular trajectories we need to preserve the relative distances, instead of a minimum value $x_{\min}$ and a maximum $x_{\max}$ for each time series we use a global minimum and a global maximum, taking into account all the time series.

4.3 Markov Transition Field

As we said in the previous section, the GAFs are usually combined together with the so called Markov Transition Field (MTF), in a way that GAFs encode static information while MTF encodes dynamical information. The first appearance of a similar framework in the literature is found in [Campanharo et al., 2011].

Given a univariate time series $\mathbf{x} = (x_1, \ldots, x_m)$, a number Q of quantile bins are identified and so each x_i results to be assigned in the corresponding quantile q_i, where $i \in [1, Q]$. Then the frequency w_{ij} relative to the transition from q_i to q_j ($q_i \to q_j$) is computed by

$$w_{ij} = \frac{N_{ij}}{\sum_j N_{ij}},$$

where N_{ij} is the number of transitions from q_i to q_j and the denominator counts all the transition from the quantile q_i, in the manner that two consecutive measurements $x_i \in q_i$ and $x_j \in q_j$ count as a transition $q_i \to q_j$. The resulting matrix $\mathbf{W}$ is a $Q \times Q$ matrix

$$W = \begin{pmatrix} \frac{N_{11}}{\sum_j N_{1j}} & \frac{N_{12}}{\sum_j N_{1j}} & \cdots & \frac{N_{1Q}}{\sum_j N_{1j}} \\ \frac{N_{21}}{\sum_j N_{2j}} & \frac{N_{22}}{\sum_j N_{2j}} & \cdots & \frac{N_{2Q}}{\sum_j N_{2j}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{N_{Q1}}{\sum_j N_{Qj}} & \frac{N_{Q2}}{\sum_j N_{Qj}} & \cdots & \frac{N_{QQ}}{\sum_j N_{Qj}} \end{pmatrix}.$$

The process is illustrated in Figure 4.5.

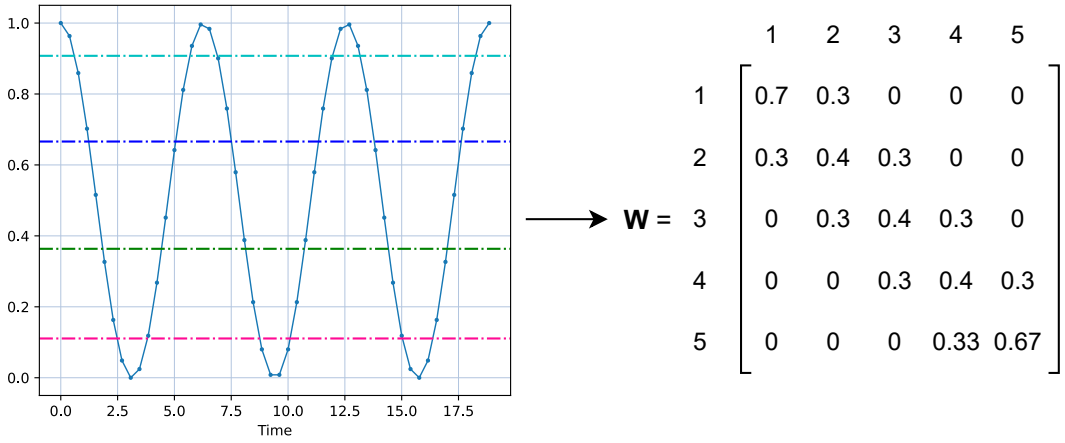

Figure 4.5: Framework for the construction of the matrix W from the cosine time series rescaled to the range $[0, 1]$, using 5 quantile bins.

The MTF matrix $\mathbf{M}$ is instead an $m \times m$ matrix where each entry m_{ij} is equal to $w_{q_{x_i} q_{x_j}}$, or rather it is equal to the transition frequency from the quantile bin of x_i to the one of x_j.

Example 4.6. Figure 4.6 shows the MTF of the cosine time series used in the previous section.

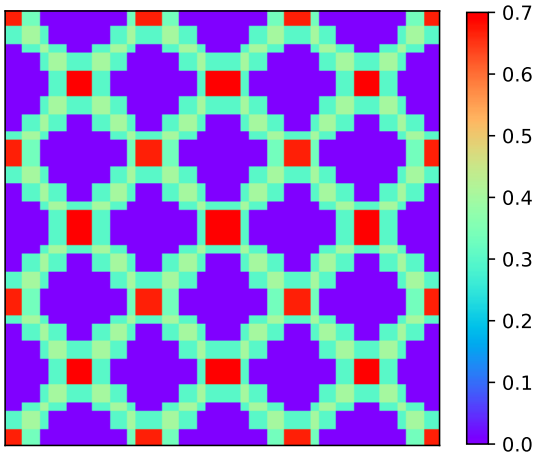

Figure 4.6: Markov transition field computed from the cosine time series $x_i = \cos t_i$ rescaled to the range $[0, 1]$, with number of bins equal to 5.

Part II

Our Analysis

Chapter 5

Proposed Methodology

In this chapter, we dscribe the approach used for classifying vehicular trajectories, combining GAFs with deep learning architectures. We explore different strategies to generate the three-channel images to feed DNNs, evaluating the impact of the use of GAFs, without and in combination with MTF.

Trajectory data, which represent the movement of vehicles over time, usually have different lengths, to address this variability, it is necessary to compress and uniform the lengths of the trajectories. We study two compression techniques, including uniform sampling and a new one that, while involving some loss of information, manages to maintain the shape and the relative time spent in a certain location, in a way that a relative speed of the trajectory is retained.

Subsequently, we evaluate our method using two neural network models widely used in the literature: MobileNetV2 and ResNet18. We analyze the performance of these models in the context of vehicle trajectories classification, comparing the results obtained with the different configurations and strategies explored. The implemented methodology is summarized in Figure 5.1.

5.1 Trajectory Extraction

Step 1. in Figure 5.1 concerns the extraction of trajectories from a dataset of geographic coordinates. It is possible to group the data using a unique identifier (e.g. vehicle ID) to obtain a new dataset composed of trajectories.

Let us now assume that the dataset has been cleaned of anomalous measurements and that there are no gaps to be filled.

5.2 Imaging Trajectories

In this section, which corresponds to step 2. in Figure 5.1, we discuss the tested and implemented methodology for dealing with trajectories and for extracting the data that constitute the input of the neural networks. In particular, starting from a trajectory dataset, we obtain a dataset composed by three-channel images.

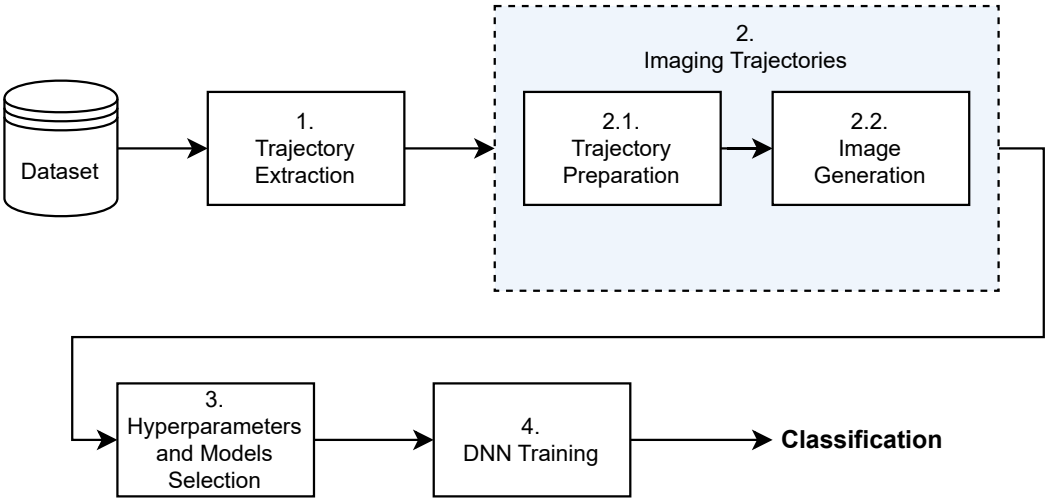

Figure 5.1: Flow chart for the proposed and implemented methodology.

5.2.1 Trajectory Preparation

The datasets containing trajectories often consist of data with varying lengths. Furthermore classification models present in the literature, such as neural networks, require the input data to have a certain size. To address this issue, data compression may be necessary for longer trajectories, whereas shorter ones might need to be filled to reach a certain length. Moreover, to leverage existing algorithms designed forunivariate time series data, we might consider applying dimensionality reduction techniques. This phase corresponds to step 2.1. in Figure 5.1, where the input is a set of trajectories with varying lengths and the output is a set of univariate time series with a predefined length.

Dimensionality Reduction

By the Definition 2.2, a trajectory is a multivariate time series, which contains information concerning the latitude and the longitude. The task of transforming a trajectory into a univariate time series, in order to allow the use of time series algorithms such as GAFs, is challenging but not impossible and it is usually completed using Hilbert's curve.
Towards the end of the 19th century, Georg Cantor proved that the interval $[0, 1]$ and the unit square $[0, 1]^2$ have the same cardinality, suggesting the existence of a bijective map between them.

A few years later, an Italian mathematician named Giuseppe Peano, demonstrated the existence of a continuous surjective map between the unit interval and the unit square [Peano, 1890], constructing the first *space-filling curve*. The curve that is of particular interest for this thesis is Hilbert's space-filling curve [Hilbert, 1891], which leverages the idea that if the line segment can be mapped onto a square, then, after partitioning the line segment into four subintervals and the square into four subsquares, is it possible to map

each subinterval to the corresponding subsquare, with adjacent subintervals mapped onto adjacents subsquares as well. In particular, starting with the unit square, then Hilbert's space-filling curve is recursively constructed. At each iteration, the side length of the parent square is halved and it is divided into four subsquares, consequently, a space-filling curve is obtained for each subsquare by scaling-down, reflecting or rotating the original curve. At the limiting case, where the number of iterations goes to infinity, the size of the square shrink to zero, defining a unique point and the Hilbert's curve is built. If the recursion is early stopped, and the squares contain multiple points, then a representative point for each square is chosen. The construction is shown in Figure 5.2.

Definition 5.1 (Hilbert's curve)**.** We can define the *Hilbert's curve* with the following mapping algorithm, taken from Bader [2012].

- For each parameter $t \in [0, 1]$, then a sequence of nested intervals

$$[0, 1] \supset [a_1, b_1] \supset \cdots \supset [a_n, b_n] \supset \ldots,$$

  exists and each interval is obtained by splitting its predecessor into four subintervals of equal size and taking the one which contains t.

- Any sequence of intervals can be mapped to a sequence of nested subsquares.

- The constructed nested subsquares will converge to a uniquely defined point which is the image of t.

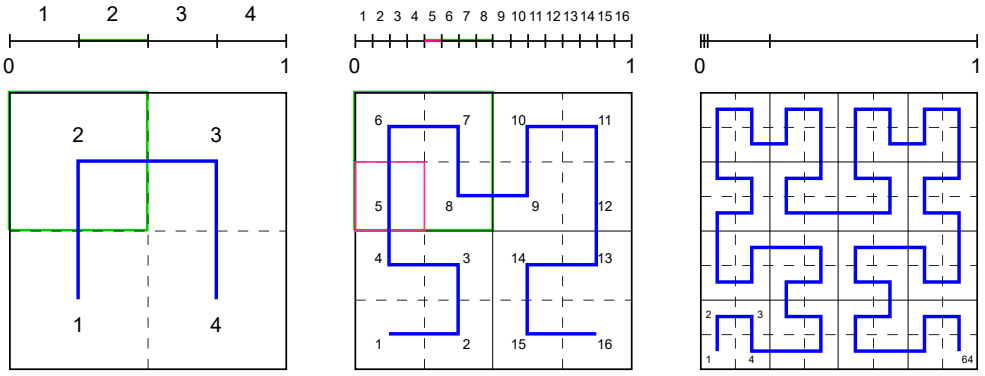

Figure 5.2: First three iterations of Hilbert's space-filling curve, originally depicted in [Hilbert, 1891].

We have defined the process which maps a given parameter t onto a point of the unit square. But now, we are interested in the inverse problem, given a point of the unit square (x, y), we want to find the parameter t. The Hilbert's curve is surjective, so there exist

points of the unit square which are images of multiple parameters t (in Figure 5.2 is it possible to see that the center of the unit square lies in the corner of three non-adjacent squares, leading to three values of t with the same image). So an inverse mapping does not exist in the strict sense, but it is forced to be unique.
The inverse mapping is called the *Hilbert's index*, and given a point (x, y) of the unit square, it computes the parameter t via a recursion algorithm which works similar to the one of the Hilbert's curve. We follow the implementation given in [Bader, 2012] (see Algorithm 1). To generate at least 4^{n_i} different indexes, given n_i the number of iterations,

Algorithm 1: Algorithm to compute the Hilbert's index, given a point of the unit square and a given accuracy [Bader, 2012].

```
Function hilbertIndex (x, y, eps);
Input  : x, y: coordinates of the point (x, y) ∈ [0, 1]^2,
         eps: required accuracy
if eps > 1 then
    return 0;
end
if x < 0.5 then
    if y < 0.5 then
        return (0 + hilbertIndex(2 * y, 2 * x, 4 * eps))/4;
    else
        return (1 + hilbertIndex(2 * x, 2 * y − 1, 4 * eps))/4;
    end
else
    if y ≥ 0.5 then
        return (2 + hilbertIndex(2 * x − 1, 2 * y − 1, 4 * eps))/4;
    else
        return (3 + hilbertIndex(1 − 2 * y, 2 − 2 * x, 4 * eps))/4;
    end
end
```

the accuracy *eps* cannot exceed $eps_{\max_{n_i}} = 4^{-(n_i-1)}$.

Hilbert space-filling curves has proven to be versatile in multiple applications due to its locality preserving property, such as clustering [Moon et al., 2001] or to transform animal trajectories [Wang and Oates, 2015b], thus we can exploit this characteristic to allow the conversion of a trajectory into a time series, while retaining spatio-temporal information.

Example 5.1. We illustrate the process of converting a trajectory into a time series with a simple example. Given the trajectory depicted in Figure 5.3:

$$\mathbf{X} = (\mathbf{x_1}, \ldots, \mathbf{x_m}) = \begin{pmatrix} 0.1 & 0.15 & 0.45 & 0.6 & 0.3 & 0.35 & 0.6 \\ 0.05 & 0.5 & 0.15 & 0.3 & 0.55 & 0.8 & 0.95 \end{pmatrix},$$

using $4^2 = 16$ different zones, the resulting univariate time series is

$$\mathbf{x} = (x_1, \ldots, x_m) = (\ 0.0,\quad 0.0,\quad 0.0625,\quad 0.8125,\quad 0.4375,\quad 0.375,\quad 0.5625\ ).$$

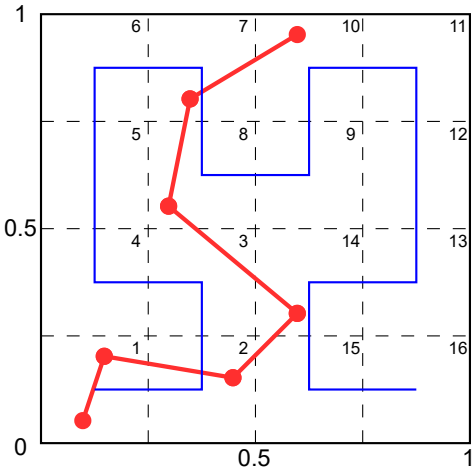

Figure 5.3: Example of how the function to map a trajectories into a time series works.

At the first iteration, the four zones are mapped into $0, 0.25, 0.5, 0.75$ by dividing the unit interval into four subintervals, the last point of the trajectory is in the zone numbered with 3, which corresponds to 0.5; at the second iteration it is in the zone 10, which corresponds to the second square of the parent square, leading to a result of $0.5 + 0.0625$, where 0.0625 is the length of each subintervals.

Compression and Filling Strategies

When considering compressing a trajectory, we could easily think to trim it up to a certain length. However, with the advances in GPS technology, we expect to acquire and collect spatio-temporal data almost in real-time. Consequently, small sampling rates (e.g. a sampling rate of 0.001 s) could result in static patterns since changes might not occur frequently. Hence, it is desirable to account for variations when employing GAFs to generate images with different pixel values.

The most straightforward approach is the *uniform sampling*, which consists in down-sampling the data i.e. only points at fixed time intervals are retained from the original trajectory. Nevertheless, this leads to excessive loss of information, as changes may occur between two sampled points and not be recorded. In particular, after the transformation with the Hilbert's curve, the time series can only takes values in sets with cardinality equal to a power of four, so uniform sampling is not the best choice because we might lose points which are visited.

To address this problem, we introduce a new compression algorithm named *squeeze* that takes into account the permanence of a vehicle in a location while maintaining the original shape of the time series. This is clear in Example 5.2.

Example 5.2. Given a time series $\mathbf{x}$, we compare the behaviour of the uniform sampling

algorithm with the one of the squeeze algorithm.

$$\mathbf{x} = (\ 0.0,\quad 0.0,\quad 0.0625,\quad 0.0625,\quad 0.0625,\quad 0.8125,\quad 0.375,\quad 0.375,\quad 0.375,\quad 0.375\ ),$$

the compressed trajectory using the uniform sampling algorithm, by imposing the final length equal to five, is

$$\mathbf{x}' = (\ 0.0,\quad 0.0625,\quad 0.0625,\quad 0.375,\quad 0.375\ ),$$

while with the squeeze algorithm is

$$\mathbf{x}'' = (\ 0.0,\quad 0.0625,\quad 0.8125,\quad 0.375,\quad 0.375\ ),$$

we can easily see that with the uniform sampling we lose the point with value 0.8125, while with squeeze we retain all the information about visited locations and also the relative time spent in them, in fact the point with value 0.375 is still the one with the greatest frequency of occurence.

We have discussed how to compress longer trajectories, now we examine how to deal with shorter ones. The simple strategy is to fill them with mirrored copies, like a vehicle which travels the same path in reverse. This strategy is intuitively better than appending the original copy of them, because it avoids spatial jumps (see Example 5.3).

Example 5.3. Given the time series $\mathbf{x}$, we show how the filling strategy works.

$$\mathbf{x} = (\ 0.0,\quad 0.0625,\quad 0.375\ ),$$

the filled times series up to a length equal to five is

$$\mathbf{x}' = (\ 0.0,\quad 0.0625,\quad 0.8125,\quad 0.8125,\quad 0.0625\ ).$$

5.2.2 Image Generation

Now, we study the various ways of composing three-channel images. As stated in Chapter 4, from the GASF matrix it is possible to reconstruct the original time series, we therefore focus only on those approaches that allow us to obtain the data used to generate the images, which paves the way to the generation of synthetic data. The input of this phase, which is the step 2.2. in Figure 5.1, is a set of univariate time series and the output is a set of three-channel images.

In order to make the discussion more understandable, we adopt the following notation to give a name to different strategies: the name of a strategy is equal to "trajectory preparation + construction of image". For the step 2.1. in Figure 5.1, the possible values are "SQ/US" to say which of the compression algorithms is involved, between squeeze ("SQ") and uniform sampling ("US"). For the step 2.2. we use the notation "(channel1 + channel2 + channel3)", where the possible values for each channel are "GASF/GADF/MTF/0", where "0" means a matrix with all entries equal to 0.

1. US + (GASF + GADF + 0): this approach consists in transforming the trajectory into a univariate time series with Hilbert's curve, compress it with uniform sampling and finally generate the three-channel image by stacking together the GASF, the GADF and the 0-value matrix.

   Example 5.4. We use the same trajectory of Example 5.1 and the resulting time series. Figure 5.4 shows the three-channel image.

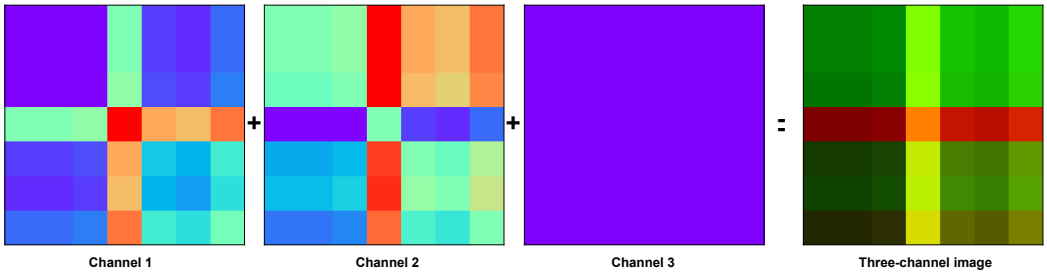

Figure 5.4: Construction of the colored image, which is composed by GASF, GADF and 0-value matrix.

2. SQ + (GASF + GADF + 0): with this approach the colored image is generated similar to the previous one, with the difference that instead of involving the use of the uniform sampling, the squeeze algorithm is employed.

3. US + (GASF + GADF + MTF): similar to the second approach, after the transformation with the Hilbert's curve, the time series is therefore compressed with uniform sampling, and the three-channel image is constituted by GASF, GADF and MTF.

   Example 5.5. We use the same time series of Example 5.4, the final colored image is depicted in Figure 5.5.

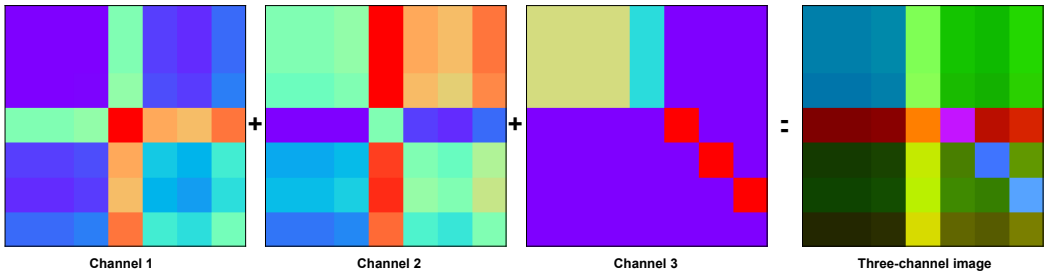

Figure 5.5: Construction of the colored image, which is composed by GASF, GADF and MTF.

4. SQ + (GASF + GADF + MTF): this strategy is similar to the previous one, and it involves the transformation with the Hilbert's curve, the compression with the squeeze algorithm and the stacking of GASF, GADF and MTF matrices.

In order to better account for the implemented methodology we show in Table 5.1 the input and the output of the main steps.

Table 5.1: Input and output for the main steps of the methodology.

Step	**Input**	**Output**
1. Trajectory Extraction	Set of geographic coordinates	Set of trajectories with varying lengths
2.1. Trajectory Preparation	Set of trajectories with varying lengths	Set of univariate time series of equal length
2.2. Image Generation	Set of univariate time series of equal length	Set of colored images

5.3 Classification

The classification is carried out in python language, using PyTorch for DNNs architectures and the pyts library [Faouzi and Janati, 2020] for imaging time series.

5.3.1 Hyperparameters and Models Selection

We train and validate two architectures of the literature: MobileNetV2 [Sandler et al., 2018] and ResNet18 [He et al., 2016].
The main hyperparameters to consider are the ones concerning the imaging of the trajectories. In particular, we try two values for the epsilon parameter of the Hilbert's curve which lead to: $\{4^5, 4^6\}$ different indexes. The final length of the trajectories is fixed to 224, since the chosen neural networks support images of size 224×224.

We chose to test 4 different settings:

- MobileNetV2 pre-trained on ImageNet-1K dataset (default setting for weights): instead of using random weights, we finetune the MobileNetV2 network by initializing it with the pre-trained network on the ImageNet-1K dataset.
- MobileNetV2 not pre-trained: MobileNetV2 with random initialization.
- ResNet18 pre-trained on ImageNet-1K dataset (default setting for weights).
- ResNet18 not pre-trained.

We use the Cross-Entropy Loss as the loss function for each configuration, defined by

$$L_{CE} = -\frac{\sum_{n=1}^{N}\sum_{c=1}^{C} t_{c,n}\log(p_{c,n})}{N},$$

where C is the set of possible class labels, N is the number of instances, $p_{c,n}$ is the probability for instance n to be in the class c and $t_{c,n}$ is equal to 1 if the target class of the instance n is c, while is 0 otherwise.
We use Adam optimizer with default settings [Kingma and Ba, 2014] and we fix the batch size to 20 items.

5.3.2 DNN Training

We split our dataset into:

- Training set: composed by the 60% of the original dataset.
- Validation set: the 20% of the original dataset.
- Test set: the remaining 20%.

The model is trained using the training set, we fix the maximum number of epochs to 50, but in order to avoid overfitting the optimum number of epochs is decided with early stopping (when the validation loss does not decrease anymore). The performance of the model is then evaluated using the test set.

Chapter 6

Results

As discussed in Chapter 5 we have implemented different approaches to obtain three-channel images to feed DNNs.
In this chapter we focus on the evaluation of the proposed methodology. Section 6.1 summarizes the main characteristics of the used dataset to test our approach. In Section 6.2 we define the metrics which allow us to evaluate and compare the diverse implemented strategies, then, this chapter concludes with a discussion and comparison of them.

We test our models on a machine with Tesla V100 GPU and IBM POWER9 processor.

6.1 Description of the Used Dataset

In order to evaluate our approach we use a synthetic dataset generated with SUMO, which is an open source traffic simulation package widely used in telecommunication and urban mobility fields. SUMO networks are encoded in Cartesian coordinates, using the UTM projection. The origin is shifted by making the lower left corner of the network corresponds to the point (0,0). The simulated scenario is the one concerning the Principality of Monaco: MoST [Codecá and Härri, 2017]. The simulated dataset contains a large number of classes with imbalanced distribution, so we aggregate some of them and we retain only information regarding coordinate x, coordinate y, time, the ID of the vehicle and the class label.

Each instance of the dataset is characterized by:

- VehicleID: the ID of the vehicle;
- Coordinate x: the value of the coordinate x in meters (m);
- Coordinate y: the value of the coordinate y in meters (m);
- Time: the time in seconds (s);
- Class: the transportation mode.

The data are collected with a sampling rate of 0.25 s, so after grouping by VehicleID, trajectories with lengths less than 4 i.e. trajectories less than 1 s are discarded, since

they could be vehicles simulated at the end of the simulation. We show the trajectories in Figure 6.1, after rescaling them into [0,1].

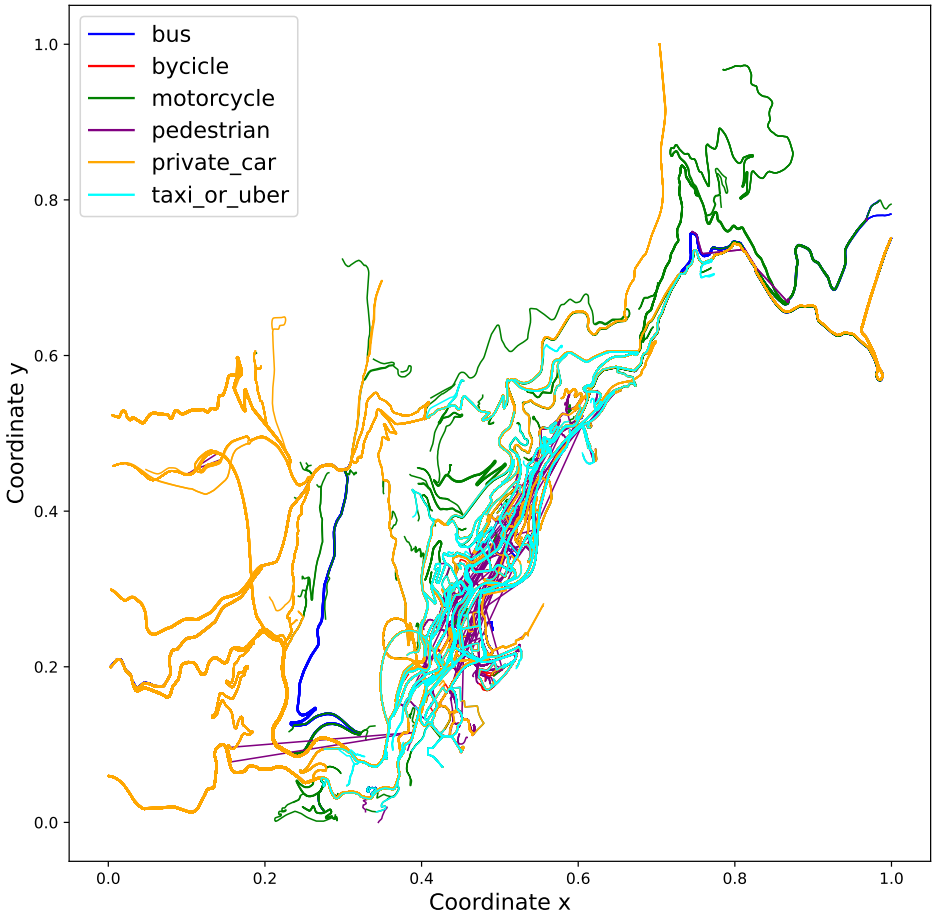

Figure 6.1: Trajectories in the space colored by class label.

We can easily observe in Figure 6.2 that the resulting dataset is not well-balanced, this could potentially lead to further issues when attempting to classify the data, because the model may be biased in favor of larger classes.

As we expected, trajectories have also very different lengths, we show the average trajectory length per class in Table 6.1. In Chapter 5 we have discussed different strategies to compress the trajectories up to a certain length, the considered approaches are now validated in 6.3 using the metrics defined in Section 6.2.

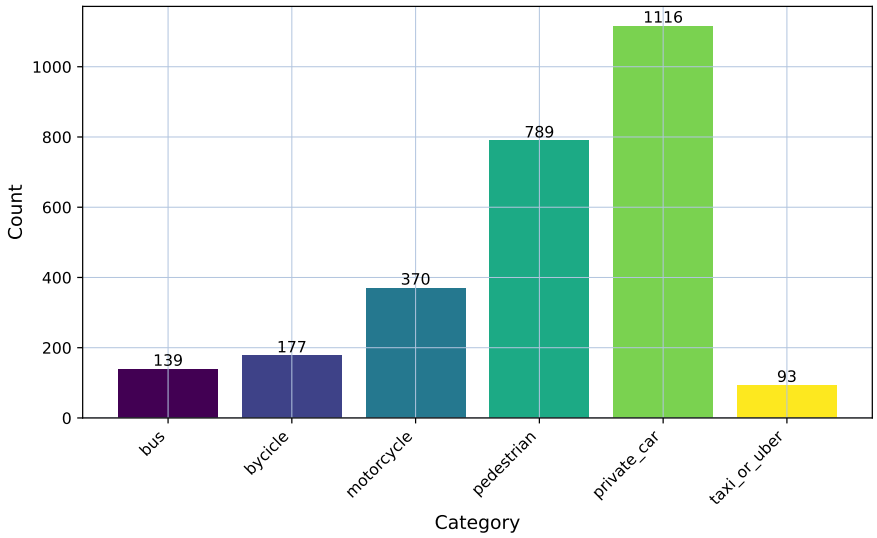

Figure 6.2: Bar plot of instances per class.

Class	Average length
Bus	3718.79
Bicycle	1366.82
Motorcycle	2217.74
Pedestrian	408.54
Private Car	2549.83
Taxi/Uber	1004.61

Table 6.1: Average length per class.

6.2 Evaluation Metrics

To effectively asses the performance of our methodology we employ Accuracy and F1 score:

$$Accuracy = \frac{\#Correct\ predections}{\#Predictions},$$

$$F1_j = 2 \times \frac{Precision_j \times Recall_j}{Precision_j + Recall_j},$$

$$Precision_j = \frac{TP_j}{TP_j + FP_j},$$

$$Recall_j = \frac{TP_j}{TP_j + FN_j},$$

where, TP_j is the number of samples correctly classified as category j, while TN_j quantifies the instances accurately identified as not belonging to category j. FP_j represents the

number of samples incorrectly categorized as j, and FN_j denotes the quantity of samples inaccurately classified as not j, despite belonging to category j. The F1 score is the harmonic mean of precision and recall, thus making it much better as an evaluation metric when dealing with imbalanced datasets.

6.3 Comparison of Results

In Chapter 5 we presented four different approaches to generate an image from a trajectory. Additionally, we discussed using two different epsilon values for the Hilbert's curve and two distinct neural network models, pre-trained or not. So, in this section, our aim is to compare the performance of different settings using the F1 score and the accuracy as evaluation metrics. We also take into account the time taken to train the neural networks and retain the loss values.

We focus whether our approach shows good performance , highlighting how an accuracy of 75% can be achieved with just a few training epochs. Since we tested 32 different setting, starting from the configuration with the best performance, we show how the results change by varying only one parameter at once. We introduce a notation similar to the one of Chapter 5 to differentiate the settings: the name of a configuration is equal to "trajectory preparation + construction of image + employed architecture". Differently from the previous chapter, we add "1024/4096" based on the choice of the epsilon value for the Hilbert's curve, resulting in "1024 US/1024 SQ/ 4096 US/4096 SQ". The architecture can be "MOB Y/MOB N/RES Y/RES N", whether the employed architecture is MobileNetV2, pre-trained or not, or ResNet18.

We now show the graphics for the best result obtained.

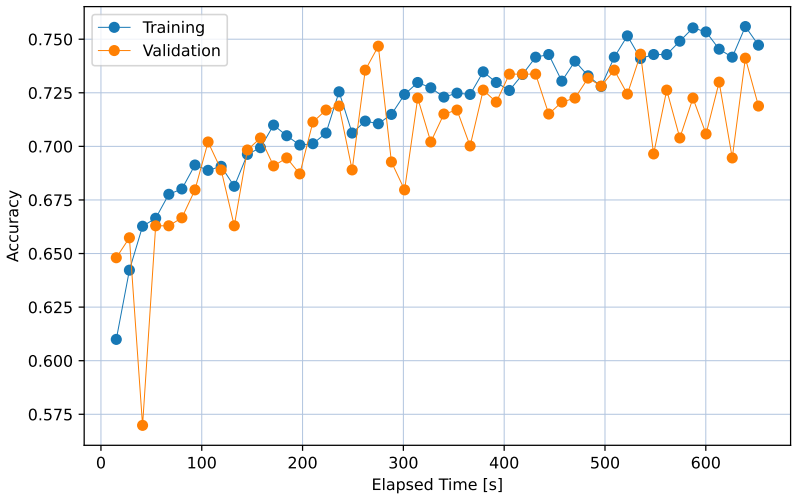

Figure 6.3: Training accuracy and validation accuracy in function of time (s) for the best-performing configuration. Markers correspond to epochs.

Figure 6.3 shows the accuracy for both the validation and training sets for the best configuration i.e. 4096 SQ + (GASF + GADF + MTF) + MOB Y, which is the one combining Hilbert's curve with 4^6 different indexes, the squeeze algorithm as compression strategy, GAFs + MTF to build the colored images, and the pre-trained MobileNetV2 architecture. Each point represents an epoch for a total of 50. We can notice that the model converges in very few epochs, reaching an accuracy around 75 %. In less than 10 minutes it has completed all the training and validation phase. In later steps the two lines begin to separate, showing the typical behavior of neural networks, learning too much from the training set and no longer being able to generalize correctly.

We now investigate the confusion matrix, to analyze how the different labels are classified.

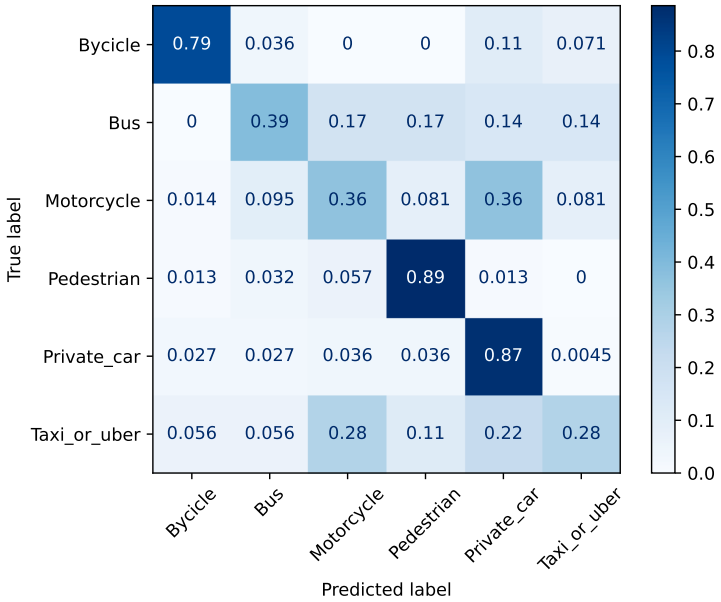

Figure 6.4: Confusion matrix for the best-performing configuration.

In Figure 6.4 it is possible to notice that, despite the dataset is not well-balanced, the model correctly classifies the instances of the most numerous classes, but also for the least numerous one i.e. "taxi or uber", the fraction of items correctly classified is larger than 0. Thus it confirms that out methodology is a viable way to address a multiclass classification problem. Moreover in Figure 6.5 we can observe that the loss function initially decreases quickly, both for the training and for the validation set. After 30 epochs the loss of the validation set no longer decreases, but rather begins to increase.

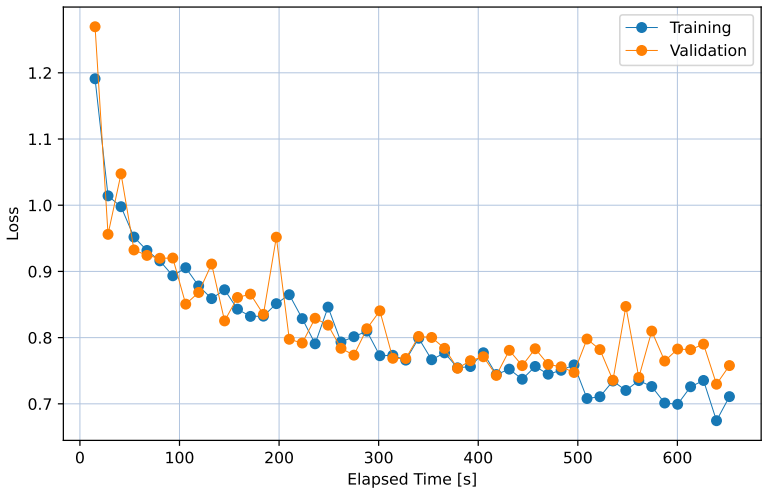

Figure 6.5: Training loss and validation loss in function of time (s) for the best-performing configuration. Markers corresponds to epochs.

As anticipated, now we empirically evaluate how the change in one step of the methodology can influence the results.

In Figure 6.6 we can notice that just the change of the initialization of the weights from the pre-trained network to random initialization can affect a lot the performance. The accuracy is lower compared to the one of the best configuration and the accuracy of the validation set suffers from this different initialization of weights showing oscillating behavior. As displayed in Figure 6.7, the number of correct prediction has decreased for all the classes, except for the two largest classes. In fact, this model cannot classify very well because it identifies everything with the largest classes.

Changing now the employed architecture in pre-trained ResNet18, we can notice in Figure 6.8 and Figure 6.9 that a simpler model with fewer parameters is to be preferred to a more complicated one, because the use of ResNet18 does not improve our performance either in terms of accuracy or in terms of recall.

We can investigate the impact of the MTF by replacing it with a matrix with all the entries equal to 0. The results seem to be very similar in terms of accuracy (Figure 6.10), but we should not rely only on accuracy, since we have an imbalanced dataset. Looking carefully to the confusion matrix depicted in Figure 6.11, we can observe that the model never predicts the taxi class, preferring all the others. Therefore we can state that the channel relating to MTFs is fundamental for capturing dynamical information of the trajectories.

We have shown models with 4096 zones of the Hilbert's curve. Intuitively we might think that reducing the number of different zones from 4096 to 1024 could lead to univariate time series that are very similar to each other, because many two-dimensional points are mapped into the same index, but overall the Hilbert's curve is effective in retaining

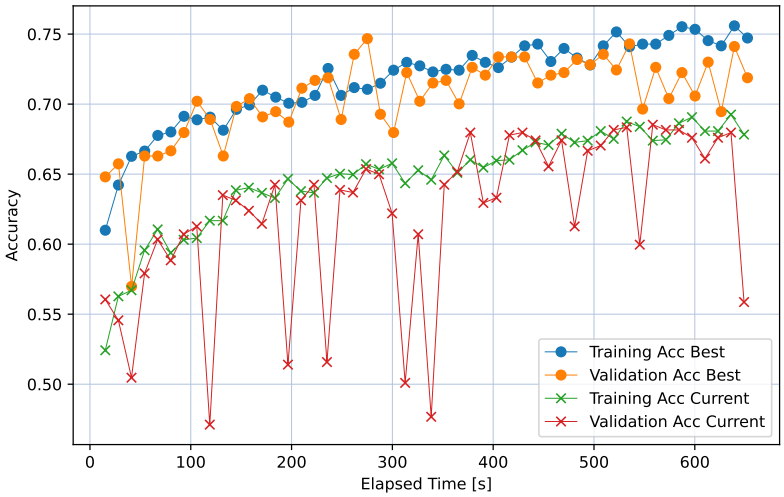

Figure 6.6: Training accuracy and validation accuracy in function of time (s) for the configuration 4096 SQ + (GASF + GADF + MTF) + MOB N, compared to the best one. Markers corresponds to epochs.

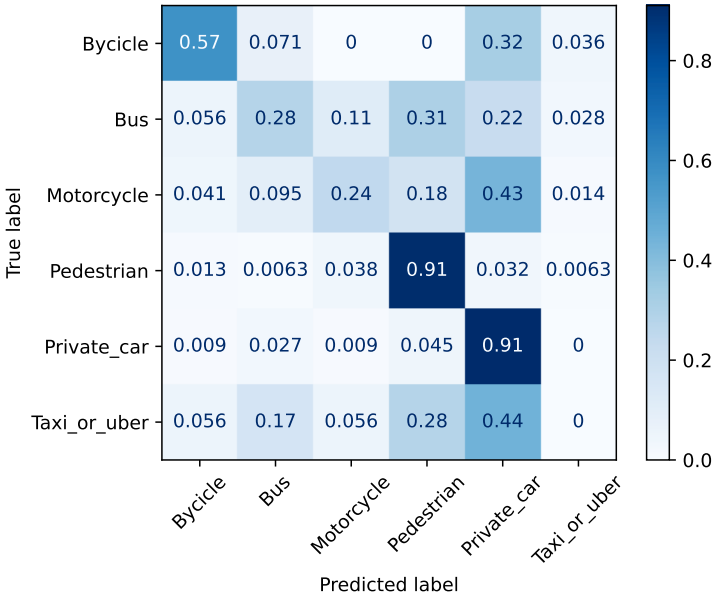

Figure 6.7: Confusion matrix for the configuration 4096 SQ + (GASF + GADF + MTF) + MOB N.

the spatio-temporal information of the trajectories (see Figure 6.12 and Figure 6.13).

Finally, we expected that uniform sampling is not the best way to compress trajectories

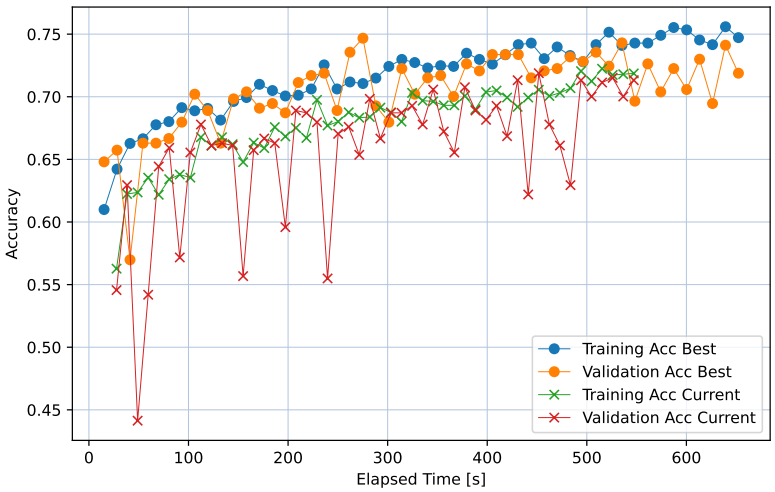

Figure 6.8: Training accuracy and validation accuracy in function of time (s) for the configuration 4096 SQ + (GASF + GADF + MTF) + RES Y, compared to the best one. Markers correspond to epochs.

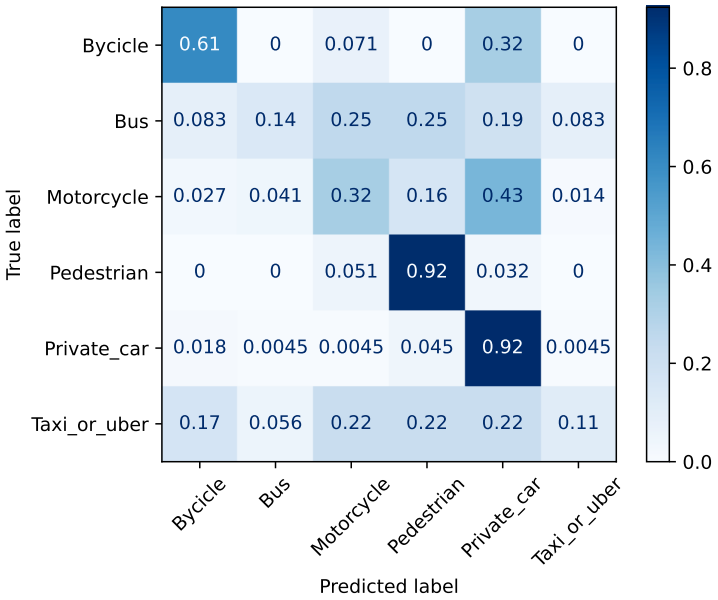

Figure 6.9: Confusion matrix for the configuration 4096 SQ + (GASF + GADF + MTF) + RES Y.

because it leads to excessive loss of information. In fact, as we can notice in Figure 6.14 and Figure 6.15, smaller classes are disadvantaged in favor of the larger ones, leading

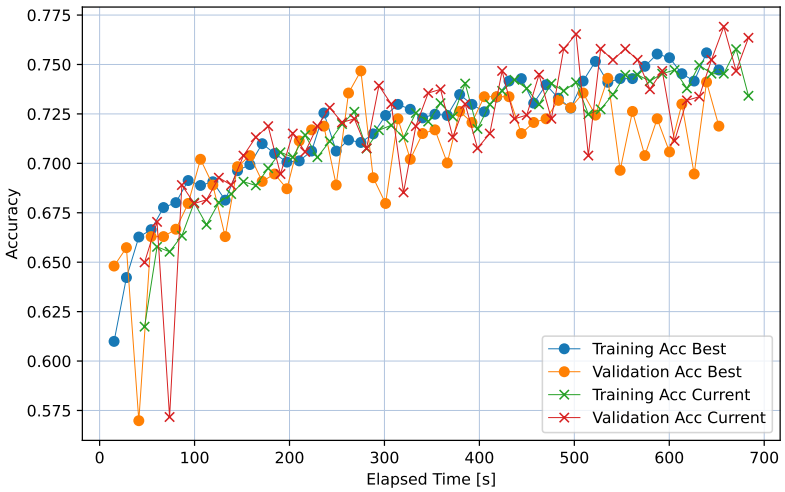

Figure 6.10: Training accuracy and validation accuracy in function of time (s) for the configuration 4096 SQ + (GASF + GADF + 0) + MOB Y, compared to the best one. Markers correspond to epochs.

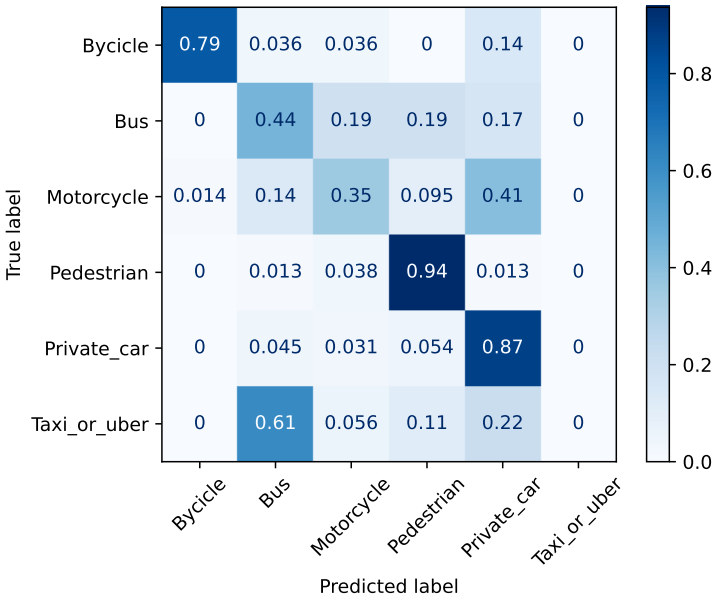

Figure 6.11: Confusion matrix for the configuration 4096 SQ + (GASF + GADF + 0) + MOB Y.

to very similar accuracy (Figure 6.14), but lower recall for classes like “bus” or “taxi or uber”.

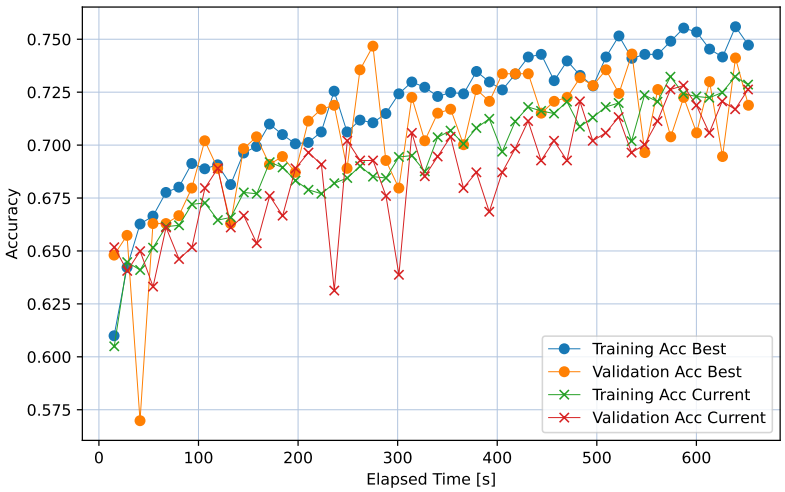

Figure 6.12: Training accuracy and validation accuracy in function of time (s) for the configuration 1024 SQ + (GASF + GADF + MTF) + MOB Y, compared to the best one. Markers correspond to epochs.

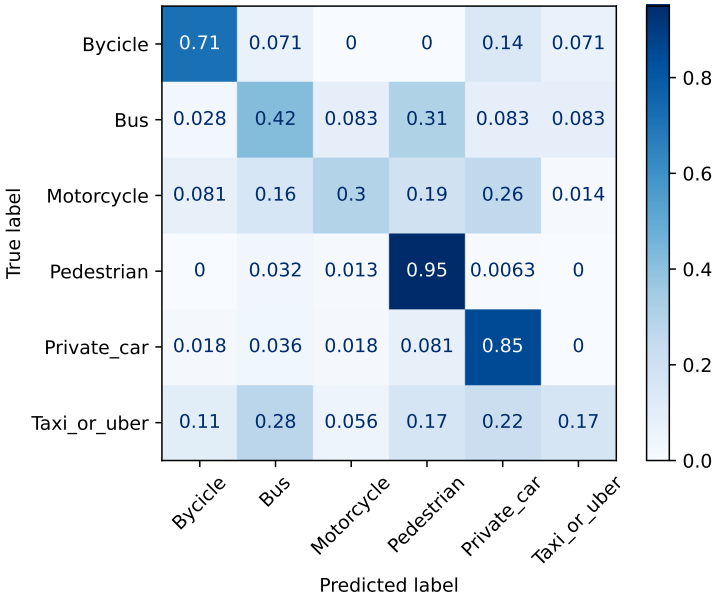

Figure 6.13: Confusion matrix for the configuration 1024 SQ + (GASF + GADF + MTF) + MOB Y.

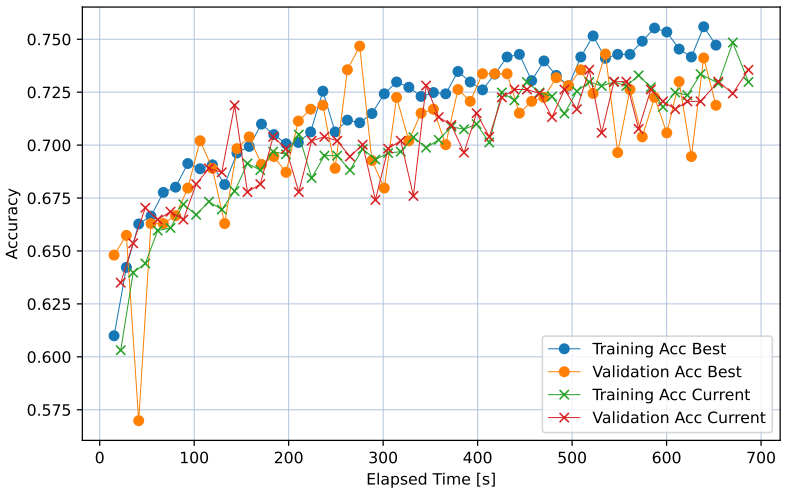

Figure 6.14: Training accuracy and validation accuracy in function of time (s) for the configuration 4096 US + (GASF + GADF + MTF) + MOB Y, compared to the best one. Markers correspond to epochs.

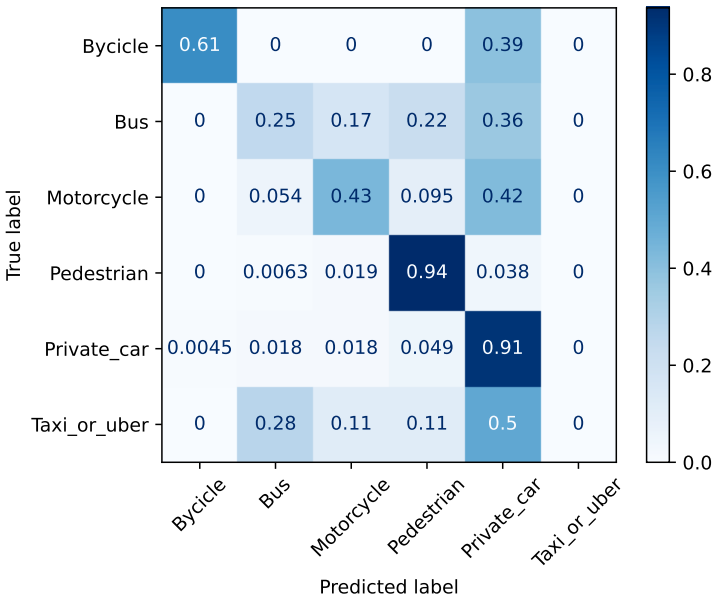

Figure 6.15: Confusion matrix for the configuration 4096 US + (GASF + GADF + MTF) + MOB Y.

6.3.1 Summary

In Table 6.2 we compare the weighted F1 score, the accuracy, and the loss computed on the test set, and the time for the training and validation phase of all the implemented and

tested settings. In particular the best five values for each column are colored in green, while the worst five are colored in orange, to make the comparison easier.

Index	F1	Acc	Loss	Elapsed Time (s)
4096 SQ + (GASF + GADF + MTF) + MOB Y	0.741	0.749	0.801	652.259
4096 SQ + (GASF + GADF + MTF) + MOB N	0.692	0.726	0.855	648.856
4096 SQ + (GASF + GADF + MTF) + RES Y	0.710	0.743	0.802	547.177
4096 SQ + (GASF + GADF + MTF) + RES N	0.654	0.700	0.886	547.087
4096 SQ + (GASF + GADF + 0) + MOB Y	0.737	0.756	0.657	683.384
4096 SQ + (GASF + GADF + 0) + MOB N	0.545	0.642	1.000	683.669
4096 SQ + (GASF + GADF + 0) + RES Y	0.709	0.741	0.731	555.286
4096 SQ + (GASF + GADF + 0) + RES N	0.599	0.667	1.001	558.305
1024 SQ + (GASF + GADF + MTF) + MOB Y	0.727	0.743	0.753	651.867
1024 SQ + (GASF + GADF + MTF) + MOB N	0.664	0.721	0.847	651.473
1024 SQ + (GASF + GADF + MTF) + RES Y	0.668	0.687	0.933	583.002
1024 SQ + (GASF + GADF + MTF) + RES N	0.583	0.654	0.975	577.568
1024 SQ + (GASF + GADF + 0) + MOB Y	0.697	0.726	0.791	652.398
1024 SQ + (GASF + GADF + 0) + MOB N	0.645	0.704	0.880	652.500
1024 SQ + (GASF + GADF + 0) + RES Y	0.706	0.730	0.778	576.443
1024 SQ + (GASF + GADF + 0) + RES N	0.659	0.700	0.901	577.167
4096 US + (GASF + GADF + MTF) + MOB Y	0.733	0.762	0.722	686.629
4096 US + (GASF + GADF + MTF) + MOB N	0.649	0.702	0.915	695.882
4096 US + (GASF + GADF + MTF) + RES Y	0.733	0.754	0.776	537.496
4096 US + (GASF + GADF + MTF) + RES N	0.612	0.667	0.911	537.630
4096 US + (GASF + GADF + 0) + MOB Y	0.737	0.754	0.740	661.851
4096 US + (GASF + GADF + 0) + MOB N	0.566	0.654	0.976	703.964
4096 US + (GASF + GADF + 0) + RES Y	0.725	0.752	0.781	529.151
4096 US + (GASF + GADF + 0) + RES N	0.599	0.659	0.965	537.976
1024 US + (GASF + GADF + MTF) + MOB Y	0.695	0.723	0.806	683.272
1024 US + (GASF + GADF + MTF) + MOB N	0.630	0.680	0.918	682.742
1024 US + (GASF + GADF + MTF) + RES Y	0.730	0.752	0.792	561.957
1024 US + (GASF + GADF + MTF) + RES N	0.611	0.672	0.971	564.601
1024 US + (GASF + GADF + 0) + MOB Y	0.712	0.736	0.785	680.260
1024 US + (GASF + GADF + 0) + MOB N	0.588	0.659	0.982	649.259
1024 US + (GASF + GADF + 0) + RES Y	0.633	0.682	0.847	555.016
1024 US + (GASF + GADF + 0) + RES N	0.576	0.655	0.982	565.933

Table 6.2: Performance metrics of different settings. Green and orange background correspond to the five best- and worst-performing configurations.

Chapter 7

Conclusion

We have proposed a methodology that leverages deep learning models and avoids the use of complex feature extraction processes. In particular, our main tools include Gramian angular fields, Markov transition field, and the Hilbert's curve. We combine these tools to reduce the dimensionality of the trajectories, allowing the use of time series algorithms, and to generate three-channel images to feed possibly pre-trained neural networks such as MobileNetV2 and ResNet18.

We have demonstrated the effectiveness of our methodology in the context of trajectory classification, reaching an accuracy of 75% (and an F1 score equal to 0.74) on a realistic dataset of the Principality of Monaco (simulated with SUMO) for a transportation mode recognition problem.

Our research is linked to many future research possibilities. We can extend our approach to other classification scenarios or validating it with real-world datasets. Furthermore, it may be useful to further investigate the impact of different neural network models and configurations. Additionally, it would also be interesting to explore strategies to deal with imbalanced datasets, such as random over sampling the smaller classes or employ loss functions which take in consideration different weights for the classes, in order to obtain a more robust classifier.

This work therefore represents a contribution to the field of trajectory classification, laying groundwork for further research to improve traffic monitoring, aiding to more efficient urban planning.

Appendix A

Code

Our code, as well as the used dataset, is available through the following GitHub profile: https://github.com/SalThesis.

In this appendix we show the main functions implemented and used in the step 2.1. Trajectory Preparation (see Figure 5.1). In particular we show the compression algorithm and the one for duplicating the time series. Here follows the definition of the function to duplicate our time series.

```
def duplicate(my_list, K):
    list_new = my_list
    if len(list_new) > K:
        print('The list exceeds the limit')
        return list_new
    while len(list_new) < K:
        list_new = list_new + list_new[::-1]
    list_new = list_new[0:K]
    return list_new
```

Now we show the compression algorithm with the definition of the *squeeze* function.

```
import numpy as np
import math


def ceil_list(my_list, final_length):
    if len(my_list) <= final_length:
        final_list = my_list
        return final_list

    old_length = len(my_list)
    final_list = []
    seg_length = 1
    for index, value in enumerate(my_list):
        if index != len(my_list) - 1:
            if value == my_list[index + 1]:
                seg_length += 1
            else:
```

```
                final_list = final_list + [value] * math.ceil((
    seg_length * final_length / old_length))
                seg_length = 1
        else:
            final_list = final_list + [value] * math.ceil((seg_length *
    final_length / old_length))
            seg_length = 1
    return final_list

def index_and_length(my_list):

    list_start_index = []
    list_length = []
    seg_length = 1
    for index, value in enumerate(my_list):
        if index != len(my_list) - 1:
            if value == my_list[index + 1]:
                seg_length += 1
            else:
                list_start_index.append(index + 1 - seg_length)
                list_length.append(seg_length)
                seg_length = 1
        else:
            list_start_index.append(index + 1 - seg_length)
            list_length.append(seg_length)
            seg_length = 1
    return list_start_index, list_length, my_list

def squeeze(my_list, final_length):
    my_list = ceil_list(my_list, final_length)
    list_start_index, list_length, my_list = index_and_length(my_list)

    index = sorted(range(len(list_length)), key = lambda k: list_length[
    k], reverse = True)
    list_start_index = [list_start_index[i] for i in index]
    list_length = [list_length[i] for i in index]

    list_length_updated = list_length.copy()

    j = 0
    while( (np.isnan(np.array(my_list)).sum() < len(my_list) -
    final_length) and (len(my_list) > final_length)):
        if all(item == 1 for item in list_length_updated) == True:
            print('The list cannot be compressed')
            break
        else:
            if list_length_updated[j] > 1:
                removed_index = list_start_index[j]
                my_list[removed_index] = float('nan')
                list_start_index.append(list_start_index[j] + 1)
                list_length_updated.append(list_length_updated[j] - 1)
                list_start_index.pop(0)
                list_length_updated.pop(0)
```

```
67              else:
68                  list_start_index.append(list_start_index[j])
69                  list_length_updated.append(list_length_updated[j])
70                  list_start_index.pop(0)
71                  list_length_updated.pop(0)
72      my_list = list(filter(lambda x: not math.isnan(x), my_list))
73
74      return my_list
```

Bibliography

Michael Bader. *Space-Filling Curves: An Introduction with Applications in Scientific Computing*. Springer Publishing Company, Incorporated, 2012.

Anthony Bagnall, Jason Lines, Jon Hills, and Aaron Bostrom. Time-series classification with cote: The collective of transformation-based ensembles. *IEEE Transactions on Knowledge and Data Engineering*, 27:1–1, 09 2015.

Silvio Barra, Salvatore Carta, Andrea Corriga, Alessandro Podda, and Diego Reforgiato Recupero. Deep learning and time series-to-image encoding for financial forecasting. *IEEE/CAA Journal of Automatica Sinica*, 7, 2020.

A.D. Bragin and Spitsyn V.G. Electroencephalogram analysis based on gramian angular field transformation. *GraphiCon'2019 Proceedings*, Volume 2, 2019.

Andriana Campanharo, Mehmet Sirer, R. Malmgren, Fernando Ramos, and Luís Amaral. Duality between time series and networks. *PloS one*, 6, 2011.

Lara Codecá and Jérôme Härri. Towards multimodal mobility simulation of c-its: The monaco sumo traffic scenario. In *2017 IEEE Vehicular Networking Conference (VNC)*, pages 97–100, 2017.

Marco Cuturi and Mathieu Blondel. Soft-dtw: A differentiable loss function for time-series. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 894–903. JMLR.org, 2017.

Sina Dabiri and Kevin Heaslip. Inferring transportation modes from gps trajectories using a convolutional neural network. *Transportation Research Part C: Emerging Technologies*, 86:360–371, 2018.

Angus Dempster, François Petitjean, and Geoffrey I. Webb. Rocket: exceptionally fast and accurate time series classification using random convolutional kernels. *Data Min. Knowl. Discov.*, 34(5):1454–1495, sep 2020.

Houtao Deng, George Runger, Eugene Tuv, and Martyanov Vladimir. A time series forest for classification and feature extraction. *Information Sciences*, 239:142–153, 2013.

Johann Faouzi and Hicham Janati. pyts: A python package for time series classification. *Journal of Machine Learning Research*, 21(46):1–6, 2020.

Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. Deep learning for time series classification: a review. *Data Mining and Knowledge Discovery*, 33:917 – 963, 2018.

Òscar Garibo i Orts, Nicolas Firbas, Laura Sebastiá, and Alberto Conejero. Gramian angular fields for leveraging pretrained computer vision models with anomalous diffusion trajectories. *Physical Review E*, 107, 2023.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

Bin Han, Hui Zhang, Ming Sun, and Fengtong Wu. A new bearing fault diagnosis method based on capsule network and markov transition field/gramian angular field. *Sensors*, 21, 2021.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. doi: 10.1109/CVPR.2016.90.

David R. Hilbert. Über die stetige abbildung einer linie auf ein flächenstück, 1891.

Ying-Yi Hong, Mr. John Joel Martinez, and Arnel Fajardo. Day-ahead solar irradiation forecasting utilizing gramian angular field and convolutional long short-term memory. *IEEE Access*, PP, 2020.

Andrew Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications, 04 2017.

Tongge Huang, Pranamesh Chakraborty, and Anuj Sharma. Deep convolutional generative adversarial networks for traffic data imputation encoding time series as images. *International Journal of Transportation Science and Technology*, 12, 2023.

Hassan Ismail Fawaz, Benjamin Lucas, Germain Forestier, Charlotte Pelletier, Daniel F. Schmidt, Jonathan Weber, Geoffrey I. Webb, Lhassane Idoumghar, Pierre-Alain Muller, and François Petitjean. Inceptiontime: Finding alexnet for time series classification. *Data mining and knowledge discovery*, 34(6):1936–1962, 2020.

Young-Seon Jeong, Myong K. Jeong, and Olufemi A. Omitaomu. Weighted dynamic time warping for time series classification. *Pattern Recognition*, 44(9), 2011. Computer Analysis of Images and Patterns.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

Ioannis Kontopoulos, Antonios Makris, and Konstantinos Tserpes. Traclets: A trajectory representation and classification library. *SoftwareX*, 21:101306, 2023.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

Cristiano Landi, Riccardo Guidotti, Mirco Nanni, and Anna Monreale. The trajectory interval forest classifier for trajectory classification, 12 2023a.

Cristiano Landi, Francesco Spinnato, Riccardo Guidotti, Anna Monreale, and Mirco Nanni. Geolet: An interpretable model for trajectory classification. In Bruno Crémilleux, Sibylle Hess, and Siegfried Nijssen, editors, *Advances in Intelligent Data Analysis XXI*, pages 236–248, Cham, 2023b. Springer Nature Switzerland.

D.C. Lay, S.R. Lay, and J. McDonald. *Linear Algebra and Its Applications*. Pearson, 2016.

Camila Leite da Silva, Lucas May Petry, and Vania Bogorny. A survey and comparison of trajectory classification methods. In *2019 8th Brazilian Conference on Intelligent Systems (BRACIS)*, pages 788–793, 2019.

Chunlin Li, Jianbin Xiong, Xingtong Zhu, Qinghua Zhang, and Shuize Wang. Fault diagnosis method based on encoding time series and convolutional neural network. *IEEE Access*, PP, 2020.

Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4):115–133, 1943.

Matthew Middlehurst, James Large, and A. Bagnall. The canonical interval forest (cif) classifier for time series classification. *2020 IEEE International Conference on Big Data (Big Data)*, pages 188–195, 2020.

Matthew Middlehurst, Patrick Schäfer, and Anthony Bagnall. Bake off redux: a review and experimental evaluation of recent time series classification algorithms, 2023.

Bongki Moon, H. V. Jagadish, Christos Faloutsos, and J. Saltz. Analysis of the clustering properties of the hilbert space-filling curve. *IEEE Trans. Knowl. Data Eng.*, 13:124–141, 2001.

Giuseppe Peano. Sur une courbe, qui remplit toute une aire plane. *Mathematische Annalen*, 36:157–160, 1890.

Simon J.D. Prince. *Understanding Deep Learning*. MIT Press, 2023.

Alejandro Pasos Ruiz, Michael Flynn, James Large, Matthew Middlehurst, and A. Bagnall. The great multivariate time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery*, 35: 401 – 449, 2020.

B.P. Rynne and M.A. Youngson. *Linear Functional Analysis*. Springer undergraduate mathematics series. Springer, 2000.

H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 26(1): 43–49, 1978.

M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4510–4520, Los Alamitos, CA, USA, jun 2018. IEEE Computer Society.

Patrick Schäfer. The boss is concerned with time series classification in the presence of noise. *Data Mining and Knowledge Discovery*, 29, 11 2015.

Mohammad Shokoohi-Yekta, Bing Hu, Hongxia Jin, Jun Wang, and Eamonn Keogh. Generalizing dtw to the multi-dimensional case requires an adaptive approach. *Data Mining and Knowledge Discovery*, 31, 01 2017.

Zhanbo Sun and Xuegang (Jeff) Ban. Vehicle classification using gps data. *Transportation Research Part C: Emerging Technologies*, 37:102 – 117, 12 2013.

Zhiguang Wang and Tim Oates. Imaging time-series to improve classification and imputation, 2015a.

Zhiguang Wang and Tim Oates. Spatially encoding temporal correlations to classify temporal data using convolutional neural networks, 2015b.

Zhiguang Wang and Tim Oates. Encoding time series as images for visual inspection and classification using tiled convolutional neural networks, 01 2015c.

Zhiguang Wang, Weizhong Yan, and Tim Oates. Time series classification from scratch with deep neural networks: A strong baseline. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 1578–1585, 2017.

Lexiang Ye and Eamonn Keogh. Time series shapelets: A novel technique that allows accurate, interpretable and fast classification. *Data Min. Knowl. Discov.*, 22:149–182, 01 2011.

Junwei You, Ying Chen, Zhuoyu Jiang, Zhangchi Liu, Zilin Huang, Yifeng Ding, and Bin Ran. Exploring driving behavior for autonomous vehicles based on gramian angular field vision transformer, 2023.

Gong Zhang, Yujuan Si, Di Wang, Weiyi Yang, and Yongjian Sun. Automated detection of myocardial infarction using a gramian angular field and principal component analysis network. *IEEE access*, 7, 2019.

Xuchao Zhang, Yifeng Gao, Jessica Lin, and Chang-Tien Lu. Tapnet: Multivariate time series classification with attentional prototypical network. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(04):6845–6852, Apr. 2020.