Università degli Studi di Napoli
Federico II

Dottorato di Ricerca in Matematica ed Applicazioni

XXXVI Ciclo

Ph.D. Thesis

Artificial Intelligence methodologies for Smart Mobility

by

Edoardo Prezioso

Tutor: **P. Festa**
Supervisor: **F. Piccialli**
Co-Supervisor: **S. Cuomo**

Nowadays, the rapid pace of urbanization presents unprecedented opportunities and challenges, catalyzing the adoption of innovative technologies for data-driven decision-making in urban contexts. Central to this transformation is the Smart City paradigm, underpinned by Deep Learning frameworks, a vital subset of Artificial Intelligence. These frameworks, evolving from rudimentary biological models to advanced neural networks, significantly enhance urban life by bolstering safety, efficiency, and sustainability. This work introduces the SMart Analytics in Roads and Transportations (SM.A.R.T) framework, a novel approach employing state-of-the-art forecasting methodology named ENCODE and cutting-edge computer vision techniques for real-time detection and tracking in urban traffic and crowd management. It highlights the pivotal role of neural networks in advancing urban computational research and shaping the future of Smart City initiatives, emphasizing the critical need for ongoing innovation in AI applications.

Contents

Chapter 1

Preface

As the 21st century unfolds, we are witnessing a rapid acceleration in technological advancement, reshaping societies and economies. Artificial Intelligence (AI), once a field of academic interest, now stands at the forefront of this revolution, leading us into the era of Smart Cities and redefining urban living. AI has evolved dramatically since its inception in the mid-20th century, marked by significant breakthroughs and a transition towards more powerful applications [1].

Smart Cities, characterized by the use of digital technology and AI for improved municipal management, represent the practical embodiment of these advancements. Deep Learning (DL), a critical subset of Machine Learning (ML), plays a pivotal role in this transformation [2]. It enables complex pattern recognition and decision-making, essential for processing vast amounts of urban data.

However, the integration of AI and Smart Cities is not without challenges [3]. Issues such as privacy, security, equity, and the digital divide are central concerns. Moreover, the inherent complexity of urban systems often makes the implementation of AI solutions a challenging endeavor.

1.1 Artificial Intelligence

1.1.1 Definition of AI

Artificial Intelligence, commonly abbreviated as AI, refers to the simulation of human intelligence in machines programmed to mimic human thought processes and behavioral patterns. This encompasses a broad range of functionalities, from learning, reasoning, and problem-solving to perception, language understanding, and decision-making. At its core, AI aims to create systems capable of performing tasks that typically require human intelligence. These tasks include interpreting complex data, engaging in various forms of communication, and adapting to new situations with flexibility and creativity. AI systems range from simple, rule-based algorithms to sophisticated neural networks capable of DL, which enables them to learn and adapt autonomously. As AI continues to evolve, it blurs the lines between computational processes and human-like cognition, leading to innovations that were once the realm of science fiction.

Furthermore, AI's diverse applications manifest in forms like virtual assistants, predictive analytics, and autonomous systems. This versatility stems from its ability to process large volumes of data quickly and efficiently, making it indispensable in an era characterized by exponential data growth. AI's integration into various industries has not only streamlined operations but also opened new avenues for innovation and problem-solving. As we advance, AI's role in augmenting human capabilities and transforming societal structures becomes increasingly significant, shaping a future where intelligent machines and human expertise collaborate for enhanced outcomes.

In this context, AI's potential in urban development, specifically in the realm of Smart Cities, is profound. By leveraging AI, cities can become more efficient, responsive, and sustainable. AI-enabled systems can analyze urban data in real-time, optimize traffic flow, enhance public safety, and facilitate effective resource management. This transformative impact of AI in urban spaces is a testament to its evolving role from a theoretical concept to a practical tool driving modern innovation. As AI continues to advance, it holds the promise of not just reshaping our urban environments but also significantly improving the quality of life within them.

1.1.2 Historical Evolution

Tracing the historical evolution of Artificial Intelligence provides a fascinating glimpse into how this field has grown and transformed over the years. The journey of AI began in the mid-20th century, marked by the seminal work of pioneers like Alan Turing, whose famous Turing Test [4] laid the groundwork for the concept of machine intelligence. The 1950s and 1960s saw the first wave of AI, characterized by optimism and significant strides in developing AI programs capable of solving algebra problems and playing chess. A fundamental milestone at that time was the development of the perceptron in 1957 by Frank Rosenblatt [5], which marked the inception of neural networks.

However, this initial enthusiasm faced setbacks during the 1970s, a period known as the ‘AI winter’, due to inflated expectations and subsequent disillusionment. Despite these challenges, the 1980s heralded a resurgence in AI research thanks to the introduction of the backpropagation algorithm, proposed by Rumelhart *et al* [6], which enabled efficient training of multi-layer neural networks. The 1990s continued the positive trend in AI research, driven by improved algorithms, increased computational power, and the advent of the internet. This era saw the development of more sophisticated AI, capable of handling complex tasks and large datasets.

The 21st century marks a pivotal phase in AI’s evolution, with the emergence of DL and big data analytics. A landmark achievement in this era was the development of AlexNet in 2012 [7], a deep convolutional neural network that significantly outperformed existing models in image recognition tasks. This success spurred a wave of research and development in DL, leading to advancements in various domains such as natural language processing, autonomous vehicles, and healthcare diagnostics. These advancements have catapulted AI from academic research labs into the real world, leading to its current status as a transformative force across various sectors. This historical evolution of AI underscores a journey of continuous learning, adaptation, and innovation, setting the stage for its integral role in the development of Smart Cities [3].

1.1.3 AI's Impact on Various Fields

The impact of AI, particularly its ML and DL branches, has been profound and wide-ranging across various fields. In healthcare, AI algorithms have revolutionized diagnostics and treatment planning, enabling more accurate disease detection and personalized medicine [8]. In finance, AI-driven systems have transformed how markets are analyzed, making trading more efficient and predictive [9].

In the realm of transportation, AI's contribution to the development of autonomous vehicles is reshaping urban mobility [10]. Furthermore, in the field of environmental science, AI aids in climate modeling and resource management, providing critical insights into sustainable practices [11].

In the field of cybersecurity, AI's ability to detect and respond to threats has become invaluable, offering advanced solutions for data protection and network security [12]. In agriculture, AI-driven technologies are optimizing crop management and yield predictions, contributing to more sustainable and efficient farming practices [13]. The field of education is also witnessing a transformation with AI-enabled personalized learning and assessment tools, facilitating more effective and engaging learning experiences [14].

In the creative industries, AI has begun to play a role in art, music, and design, pushing the boundaries of creativity and opening up new avenues for artistic expression [15]. Moreover, AI's integration into everyday life through smart home technologies and personal assistants is enhancing convenience and accessibility, reshaping daily routines [16].

These diverse applications of AI not only demonstrate its versatility but also highlight its profound impact on society, economy, and culture.

1.2 An Overview of Smart City Management

1.2.1 Definition of smart cities and key components

A Smart City is an urban area that harnesses technology, particularly information and communication technology (ICT), to improve operational efficiency, share information with the public, and enhance both the quality of government services and citizen welfare [17]. The concept of a Smart City goes beyond the mere implementation of technology; it involves the strategic integration of digital and telecommunication technologies into city infrastructure, public services, and administration.

The primary objective of Smart Cities is to optimize city functions and drive economic growth while improving quality of life for its citizens through smart technology. This involves the use of sensors, IoT devices, data analytics, and other advanced technologies to manage and optimize traditional urban and public services, such as transportation, water supply, waste management, and energy systems. At its core, a Smart City is characterized by its ability to adapt, respond to, and evolve with the changing needs and demands of its inhabitants, making urban living more efficient, sustainable, and enjoyable.

Furthermore, Smart Cities are defined by their use of data-driven decision-making processes [18]. Through the integration of advanced analytics and AI, these cities can intelligently manage resources and services, leading to enhanced urban planning and development. Smart Cities also prioritize citizen engagement and participation, leveraging technology to foster a more connected and inclusive community. This holistic approach to urban management, encompassing technological, economic, and social dimensions, positions Smart Cities at the forefront of urban innovation, transforming the way cities operate and improving the overall quality of life for their residents.

The key components of Smart Cities are critical in orchestrating their functionality and achieving their objectives. Firstly, Internet of Things (IoT) devices form the backbone, collecting data from various sources like traffic lights, cameras, and sensors. This data provides real-time insights into city operations, enabling responsive and adaptive management [19].

Data centers are another crucial component, acting as the central hub where the collected data is stored, processed, and analyzed. These centers employ advanced data analytics and AI algorithms to interpret vast data sets, deriving actionable insights for city management.

Communication networks in Smart Cities, including Wi-Fi, 5G, and fiber optics, ensure uninterrupted and high-speed connectivity. These networks are vital for the seamless transfer of data between IoT devices, data centers, and service providers, facilitating efficient communication and coordination across various city functions. Together, these components form an integrated ecosystem that drives the Smart City framework, enhancing urban life through technology-driven solutions.

1.2.2 Future trends and challenges

As Smart Cities continue to evolve, emerging trends include the increased integration of AI and IoT, leading to even more efficient and responsive urban environments. The use of big data analytics for predictive governance and the adoption of more sustainable, renewable energy sources are also on the rise. However, these advancements come with challenges such as ensuring data privacy, security, and addressing the digital divide to ensure equitable access to technology [20, 21]. Balancing technological innovation with ethical considerations and maintaining robust cybersecurity measures will be crucial for the sustainable growth of Smart Cities in the future.

Furthermore, as Smart Cities become increasingly interconnected, the challenge of interoperability between different technologies and platforms becomes significant. Cities will need to develop standards and protocols that allow various systems to communicate seamlessly. Another emerging trend is the focus on citizen-centric designs, ensuring that Smart City technologies serve the needs and improve the lives of the residents. As we navigate these challenges and trends, the future of Smart Cities looks promising, with the potential to revolutionize urban living, making it more efficient, sustainable, and inclusive.

1.3 Research Objectives and Contributions

As previously evidenced, DL has solidified its status as a quintessential driver for the development of intelligent systems tailored to the smart city paradigm. This dissertation places a significant emphasis on identifying and articulating novel AI methodologies that are specifically suited for smart mobility applications, a domain that is increasingly becoming a focal point of industrial research within the context of smart cities.

The initial phase of this research was dedicated to an in-depth examination of the fundamental components that constitute the essence of Deep Learning. This examination encompassed a thorough analysis of neural networks and the overarching paradigms of learning that underpin DL, providing a foundational understanding necessary for navigating the complexities of smart city challenges.

Progressing from foundational principles to the exploration of contemporary advancements, this study delves into the forefront of Deep Learning innovations. Recent contributions to the literature have shed light on novel approaches to surmount the challenges inherent in smart city applications. Specifically, in the domain of forecasting pertaining to time series data, the emergence of hybrid models has demonstrated significant potential in addressing issues such as high dimensionality and heteroskedasticity, which are prevalent in time series datasets.

Moreover, the exigencies of real-time analytics, especially in the context of processing high-resolution video feeds for smart mobility applications, necessitated a focused exploration of image data complexities. This exploration highlighted how advancements in object detection technologies, particularly through the development and refinement of YOLO models, have provided groundbreaking solutions to the challenges of image-based analytics.

The DL has cemented itself as a primary way to develop an intelligent system for the smart city. In this work, a major focus has been given on finding new AI methodologies which are adapt to such context, in particular for the smart mobility as per the industrial kind of research.

To proceed in this direction, the following steps were followed in this work: firstly, we delved into the study of the main blocks characterizing the DL in general, which are the neural networks and the learning paradigm. Secondly, we explored the state of the art in DL methodologies, given that only recently some contributions opened the way to the solution of challenges.

Indeed, as it has been seen, for tasks related to forecasting, one way to overcome challenges related to time series data are based on the hybrid models, hence there has been the necessity to create frameworks which overcome common problems with such data, like the high dimensionality and the heteroskedacity. On the other hand, for making real-time analytics based on data provided from high-resolution video feed we need to take into account the complexities of the images and how the most recent works in object detection overcome such limits, such as with YOLO models.

Pursuant to the outlined trajectory, this dissertation makes two principal contributions to the domain of smart city analytics: The first is the development and introduction of the ENCODE framework, a pioneering approach in the realm of forecasting. This framework distinguishes itself by delivering superior performance relative to existing state-of-the-art methodologies, while simultaneously demonstrating its utility within the smart city paradigm. The second contribution is the formulation of the SM.A.R.T. analytics tool, a comprehensive system that seamlessly integrates the forecasting capabilities of ENCODE with an advanced object detection framework. This integration enables the precise location, classification, and tracking of objects within video feeds, facilitating the real-time identification of anomalous patterns and behaviors crucial for smart city applications.

This dissertation begins with an exploration of Neural Networks, delving into the intricacies of neural architectures, the underpinnings of the deep learning paradigm, and an examination of cutting-edge models that have shaped the landscape of Deep Learning. Chapter 3 transitions to a detailed analysis of forecasting and object detection methodologies, emphasizing their historical evolution, traditional approaches, and the latest advancements in the field. Chapter 4 scrutinizes the ENCODE framework, initiating with a discourse on "combining" methods and extending to its application in enhancing smart city analytics. Chapter 5 unveils the SM.A.R.T. framework, elucidating the operational mechanics of the YOLOv8 model, its integral components, and its deployment in managing crowds within smart city environments. The thesis culminates in Chapter 6 with a thorough synthesis of the research findings, offering a reflective evaluation of the contributions and their implications for the future of smart mobility and urban analytics.

Part I

Fundamentals of Deep Learning

Chapter 2

Neural Networks

Neural Networks are quintessential to the advancement of Artificial Intelligence and pivotal in the orchestration of Smart City frameworks. This chapter is meticulously structured to elucidate the complex architecture and operational mechanisms of neural networks, tracing their evolution from theoretical models to their instrumental roles in contemporary urban systems. As the narrative unfolds, it will systematically dissect the nuances of neural network design, offering a foundational comprehension essential for appreciating their application in the succeeding contexts of deep learning and Smart City technologies.

Originating from the biological neural networks in animal brains, artificial neural networks are a series of algorithms crafted to discern underlying patterns in data, mirroring the operation of the human brain. These networks consist of layers of nodes or 'neurons', each performing simple operations. When collectively orchestrated, they execute highly complex tasks.

In Section 2.1, the thesis transitions into a more technical and mathematical framework to define the key components of neural networks: neurons, layers, links, and graphs. This section meticulously demystifies each element through mathematical models, elucidating how individual neurons function, how they interconnect to form complex layers, and how these layers interact to create the overarching structure of a neural network. Graph theory is employed to illustrate the intricate web of connections within these networks, providing a comprehensive understanding of the underlying structure and topology. This mathematical perspective is vital for grasping the operational principles of neural networks, laying a solid groundwork for appreciating their computational prowess.

In Section 2.2, the focus shifts to the learning paradigms that underpin neural network functionality. This section delves into the fundamental concepts of how neural networks learn from data, highlighting the role of loss functions and the backpropagation algorithm in training neural networks. It elucidates the process through which these networks adjust and refine their parameters, enabling them to make accurate predictions or classifications. This exploration into the learning mechanisms not only deepens the understanding of neural network operations but also sets the stage for discussing their practical applications in subsequent chapters, particularly in the context of Smart City technologies.

2.1 Neural Architecture

The following mathematical framework which is used to define the architecture, has the aim to provide "containers" for each component of the neural architecture, i.e. the neurons, the layers and the links are defined as mathematical sets made of various components of different nature (for instance, the set of neurons contains weights, bias, activation functions, and more), offering large versatility in the defining of the well known deep learning architectures, which not only applies to our case study, which is smart cities, but also on other contexts.

2.1.1 Neuron

At the core of neural architectures lies the neuron, a fundamental unit encapsulating the essential principles and mechanisms inherent in artificial neural systems. Analogous to its biological counterpart, an artificial neuron is designed to process multiple inputs and produce a single output. However, diverging from the biological model, which utilizes electrochemical signals, the artificial neuron functions through the manipulation of numerical data and mathematical operations.

The operation of an artificial neuron is characterized by a process termed the ‘action’, which encompasses the weighting, aggregation, and activation of inputs. Initially, the neuron receives input values, each of which is assigned a weight, reflecting its relative importance in the computation process. These weighted inputs are then aggregated, forming a combined value that reflects the collective influence of the inputs.

This aggregated value is subsequently passed through an activation function, a pivotal component of the neuron’s action. The activation function introduces non-linearity, enabling the neural network to perform complex, non-linear tasks beyond the capabilities of simple linear processing. This comprehensive action — from weighting and aggregation to activation — is central to the neuron’s functionality, underpinning its ability to contribute effectively to the broader neural network’s computational power.

Mathematical framework

In the following we provide a formal set-theoretic definition of a feedforward neuron and a definition of action of a feedforward neuron:

Definition 2.1.1. *A* neuron *is delineated as the set* $\mathcal{N} = \{\mathcal{M}, \boldsymbol{w}, b, \boldsymbol{\theta}, \phi, \sigma\}$*, where:*

- $\mathcal{M} \subseteq \mathbb{R}^N$ *represents the set of inputs, with* $N \geq 1$*;*
- $\boldsymbol{w} \in \mathbb{R}^N$ *is a vector of parameters corresponding to the inputs, designated as* weights*;*
- $b \in \mathbb{R}$ *is a scalar parameter, termed the* bias*;*
- $\boldsymbol{\theta} \in \mathbb{R}^P$*, with* $P \geq 1$*, encapsulates additional parameters of the neuron;*

- $\phi : \mathcal{M} \to \mathbb{R}$ *is a parametric function governed by parameters* $\boldsymbol{w}$*,* b*, and* $\boldsymbol{\theta}$*, referred to as the* aggregation function*;*
- $\sigma : \mathbb{R} \to \mathbb{R}$ *is a typically nonlinear real function, known as the* activation function*.*

Given a neuron $\mathcal{N} = \{\mathcal{M}, \boldsymbol{w}, b, \boldsymbol{\theta}, \phi, \sigma\}$, the *action* of a neuron is defined as the function $f_{\mathcal{N}} : \mathcal{M} \to \mathbb{R}$ such that:

$$\boldsymbol{x} \mapsto f_{\mathcal{N}}(\boldsymbol{x}; \boldsymbol{w}, b, \boldsymbol{\theta}) = \sigma(\phi(\boldsymbol{x}; \boldsymbol{w}, b, \boldsymbol{\theta})) \tag{2.1}$$

Here, $v_{\mathcal{N}} = f_{\mathcal{N}}(\boldsymbol{x}; \boldsymbol{w}, b, \boldsymbol{\theta})$ denotes the state variable of the feedforward neuron from the input $\boldsymbol{x}$).

Each component within the neuron's architecture plays a crucial role in its overall functionality. The weights and biases are integral for the neuron's learning capability. Initially, these might be set randomly or based on specific strategies, but as the neural network learns, they undergo significant refinement. Weights adjust the influence of each input on the output, encapsulating the network's acquired knowledge, while biases provide flexibility, adjusting the overall output where input scaling is insufficient.

The aggregation function, typically an affine transformation, is fundamental in combining the weighted inputs and biases. This function forms the initial phase of the neuron's computation, setting the stage for the activation process.

Additionally, the neuron may include other parameters, $\boldsymbol{\theta}$, which can be crucial in certain architectures. These parameters offer an additional degree of freedom, allowing the neuron to adapt its behavior more precisely to the specific requirements of a given task or dataset.

The activation function is another vital component. It introduces non-linearity, enabling the neuron to perform complex, non-linear tasks. This function determines how the aggregated input is transformed into an output, allowing the neuron to make decisions, classify data, or recognize patterns. Without this non-linearity, neural networks would be limited to solving only simple, linearly separable problems.

Together, these elements – weights, biases, the aggregation function, additional parameters, and the activation function – harmonize to define the neuron's operation, allowing it to learn from data, make predictions, and contribute effectively to the broader neural network's computational power.

The classical example of a neuron is the *perceptron*, inspired by Rosenblatt's model [5]. This type of neuron operates by applying a linear transformation to the input data, summing the weighted results and the bias, and then making a binary decision based on the aggregate value. Within this framework, it can be defined by setting the additional parameters $\boldsymbol{\theta} = \{\}$, the aggregation function ϕ as the following:

$$\phi(\boldsymbol{x}; \boldsymbol{w}, b) = \boldsymbol{w} \cdot \boldsymbol{x} + b = \sum_{j=1}^{N} w_j x_j + b, \tag{2.2}$$

and the activation function σ to be the following:

$$\sigma(z) = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{otherwise.} \end{cases} \tag{2.3}$$

The perceptron's simplicity, while instrumental in early neural network theory, exhibits limitations, particularly regarding its learning process and decision boundary. These constraints are due to its non-differentiable activation function. In modern neural network applications, neurons with differentiable (almost everywhere) activation functions are preferred. These neurons, encompassing a wider range of activation functions, offer greater flexibility and efficacy in learning complex patterns, hence representing the contemporary standard in neural network design.

2.1.2 Layer

In the realm of neural networks, hierarchical architectures composed of intricately organized computational elements, termed neurons, are fundamental. These neurons are systematically arrayed into stratified levels, each known as a 'layer.' Within these layers, each neuron performs specific transformations on its inputs, effectively refining and processing the data as it traverses the network. The arrangement of neurons into layers is a deliberate and strategic choice, adhering to specific configurations or topologies tailored for distinct data types and computational goals. The architectural layout of these layers is pivotal, as it profoundly impacts the network's functional capabilities and its adeptness at addressing various computational tasks.

The traditional notion of a layer, as delineated in [1], conceptualizes it as a *stack* of neurons where each neuron receives identical inputs. Thus, the collective output of a layer is conceived as an amalgamation or vector of the outputs from individual neurons. However, contemporary research has introduced several variations to this basic layer structure:

1. Layers accommodating multiple inputs, with the Attention layer, proposed by Vaswani *et al* [22], being a prominent example;

2. Layers that preprocess inputs, such as employing random input zeroing mechanisms like Dropout [23];

3. Layers wherein the outputs undergo post-processing transformations, exemplified by the softmax function or the Attention layer once more.

Accordingly, this section will present a refined definition of a layer that encompasses these developments. Addressing point 1 necessitates the introduction of K distinct input sets $\mathcal{M}_1, \mathcal{M}_2, \dots, \mathcal{M}_K$, with $K > 0$. This leads to the requirement of K corresponding stacks of neurons, each processing a unique input set. For points 2 and 3, the introduction of two additional parametric functions is required, one for input filtration and another for output transformation, independent of the neural operations within each neuron.

Stack of neurons

Prior to defining a layer in this extended context, we first define a stack of $n > 0$ neurons as $\boldsymbol{N} = \{\mathcal{N}^{(1)}, \dots \mathcal{N}^{(n)}\}$, where $\mathcal{N}^{(j)} = \{\mathcal{M}, \boldsymbol{w}^{(j)}, b^{(j)}, \boldsymbol{\theta}, \phi, \sigma\}$ for all $j = 1, \dots, n$ This implies that each neuron shares the input set, additional parameters, aggregation function, and activation function. Consequently, the action of a stack is defined by the function $f_{\boldsymbol{N}} : \mathcal{M} \to \mathbb{R}^n$ as follows:

$$f_{\boldsymbol{N}}(\boldsymbol{x}) = \sigma(\phi(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{b}, \boldsymbol{\theta})) \tag{2.4}$$

with $\boldsymbol{W} = [\boldsymbol{w}^{(1)}, \boldsymbol{w}^{(2)}, \dots, \boldsymbol{w}^{(n)}]^\top$ the weights matrix, $\boldsymbol{b} = [b^{(1)}, \dots, b^{(n)}]^\top$ the bias vector.

Having established the concept of a stack of neurons, we now advance to the formal definition of a layer within neural networks. This definition encapsulates the multifaceted nature of layers, recognizing their complexity beyond mere neuron aggregation:

Definition 2.1.2. *Let $\boldsymbol{N} = \{\boldsymbol{N}_1, \boldsymbol{N}_2, \dots, \boldsymbol{N}_K\}$ be K stacks of neurons, with $K > 0$. A* layer *is then defined as the set $\boldsymbol{\mathcal{L}} = \{\boldsymbol{N}, \boldsymbol{\Omega}_I, \boldsymbol{\Omega}_O, \boldsymbol{\Phi}_I, \boldsymbol{\Phi}_O, \boldsymbol{\sigma}\}$, comprising:*

- $\boldsymbol{\Omega}_I$ *and* $\boldsymbol{\Omega}_O$ *two sets of additional parameters which are characteristic to the layer;*
- $\boldsymbol{\Phi}_I : \mathcal{M}_1 \times \cdots \times \mathcal{M}_K \to \mathcal{M}_1 \times \cdots \times \mathcal{M}_K$ *a parametric function named the* preprocessing function *with parameters* $\boldsymbol{\Omega}_I$ *and operates on the input data;*
- $\boldsymbol{\Phi}_O : \mathbb{R}^{n_1} \times \cdots \times \mathbb{R}^{n_k} \to \mathbb{R}^M$ *a parametric function named the* postprocessing function *which has parameters* $\boldsymbol{\Omega}_O$ *and operates on the output of the neurons on each stack;*
- $\boldsymbol{\sigma} : \mathbb{R}^M \to \mathbb{R}^M$ *the layer's activation function.*

The layer's action is delineated by the function $f_{\mathcal{L}}$, mathematically expressed as:

$$f_{\mathcal{L}}(\mathbf{X}; \boldsymbol{\Omega}_I, \boldsymbol{\Omega}_O) = \boldsymbol{\sigma}(\boldsymbol{\Phi}_O(f_{\mathcal{N}*}(\boldsymbol{\Phi}_I(\mathbf{X})))). \tag{2.5}$$

Here, X represents the amalgamation of all input vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_K$ and $f_{\mathcal{N}*} = [f_{\boldsymbol{N}_1}, \ldots, f_{\boldsymbol{N}_K}]$ is the collective vector function encapsulating the actions of all neuron stacks. With X as the input and given fixed weights and biases, we define the *state variable* $\boldsymbol{y} = f_{\mathcal{L}}(\mathsf{X}; \boldsymbol{\Omega}_I, \boldsymbol{\Omega}_O)$. Subsequent sections will explore various common instances of layers, demonstrating the versatility and applicability of this expanded layer definition. Later sections will explore the recent advancements in the literature.

2.1.3 Layer Compatibility and Connections

The creation of an effective neural network necessitates not just the definition of individual layers but also a clear understanding of how these layers interconnect within the network. This interconnection is governed by the principle of layer compatibility. For two layers to be functionally connected, their dimensional properties must align. Specifically, the input set of the succeeding layer ($\mathcal{L}_2$) should encompass the output set of the preceding layer ($\mathcal{L}_1$). This compatibility is essential for the seamless flow of information and the integrated functioning of the layers within the network architecture. The formal definition of the connection between compatible layers is as follows:

Definition 2.1.3. *The connection between two compatible layers, $\mathcal{L}_1$ and $\mathcal{L}_2$, is defined by a set $\mathcal{C}$. This set encompasses the directed link and the weights matrix, which together facilitate the transmission of information from from $\mathcal{L}_1$ to $\mathcal{L}_2$. The directed link ensures unidirectional data flow, while the weights matrix quantifies and modulates the strength of this connection.*

Considering the capacity of a layer to receive inputs from multiple sources, it is common for a single layer to establish connections with several other layers. These multiple connections contribute to the layer's ability to integrate diverse information streams. Consequently, this framework allows for the conceptualization of a neural network as a directed graph, where each node represents a layer, and the edges represent the connections as defined by $\mathcal{C}$. This graph-based representation aids in visualizing the structure and information flow within the network, thereby enhancing our understanding of its operational dynamics.

2.1.4 Neural Network

For the purpose of this thesis, we conceptualize a neural network as a particular kind of Directed Acyclic Graph (DAG). While it is important to acknowledge that not all neural architectures can be accurately represented by a DAG, this model is sufficient and appropriate for the scope of our discussion. The DAG framework allows for a clear and structured understanding of neural networks, especially when considering feedforward architectures.

Graphs

A Directed Acyclic Graph (DAG), denoted as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, is a graph consisting of nodes $\mathcal{V}$ and directed edges $\mathcal{E}$. The defining characteristics of a DAG include the absence of cycles (ensuring that no path forms a loop) and the uniqueness of edges between any given pair of nodes. In this context, a node n_2 is 'reachable' from another node n_1 if there is a directed path from n_1 to n_2.

Feedforward Neural Network Graph (FNNG)

In the context of this thesis, an FNNG is represented as a DAG $\mathcal{G}_{\text{FNN}} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ comprises a sequence of layers $\mathcal{L}_0, \ldots, \mathcal{L}_L$ and $\mathcal{E}$ includes the edges $e_1, \ldots, e_E$ that interconnect these layers. This representation underscores several essential properties of an FNNG:

- **Start and End Nodes:** The network includes distinct entry (input layer) and exit (output layer) points.
- **Connectivity:** Every start node is linked to at least one end node, ensuring a directed progression of data and operations from the input to the output.

- **Edge Uniqueness:** The network maintains a single, unique connection between two layers, avoiding parallel edges and ensuring a clear path of data flow.
- **Acyclic Structure:** Reflecting the DAG architecture, the FNNG does not contain cycles, thereby preventing feedback loops within this model of neural network.

This graph-based conceptualization not only aids in the visualization of a neural network's structure but also illustrates the sequential and interconnected nature of its data processing. By adopting this model within the scope of our study, we can effectively explore the dynamics and properties of feedforward neural network architectures.

2.1.5 Mathematical Model of the Neural Network

A neural network model, within the context of an FNNG, involves interconnected layers represented as action vector fields. The model's total action is parameterized by weights, biases, and neuron-specific parameters.

The FNN can be mathematically encapsulated within the previously descibed framework as follows:

Definition 2.1.4. *Let $\mathcal{G}_{FNN}$ represent a* FNNG. *Let us denote the set of all start layers as $\hat{\mathcal{L}}_{in} = \{\mathcal{L}_{in}^{(1)}, \dots, \mathcal{L}_{in}^{(K_{in})}$, with $K_{in} \geq 1$, and the set of all end layers as $\hat{\mathcal{L}}_{out} = \mathcal{L}_{out}^{(1)}, \dots, \mathcal{L}_{out}^{(K_{out})}$, with $K_{out} \geq 1$. Also, let $\hat{\mathcal{M}}$ and $\hat{\mathcal{T}}$ the Cartesian product of all the inputs sets in each layer and the output sets in each layer, respectively.*

A Feedforward Neural Network model $\boldsymbol{f}_{FNN}$*, or* FNN *model, is defined as the cumulative action of the connected layers in $\mathcal{V}$ through the edges in $\mathcal{E}$ commencing from the start layers $\hat{\mathcal{L}}_{in}$ and concluding at the end layers $\hat{\mathcal{L}}_{out}$. Such function is parameteric with parameters $\boldsymbol{W}$, $\boldsymbol{B}$, and $\boldsymbol{\Theta}$, defined as the set of all the weights, of all the biases and of all the additional parameters of each layer, respectively. The evaluation of the function $\boldsymbol{f}_{FNN}$ with an input value $\boldsymbol{v} = (\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(K)})$ is called the* feedforward propagation *of the neural network for the input $\boldsymbol{v}$.*

The Multilayer Perceptron (MLP), a classical neural network model, is a specific instantiation of this framework. In an MLP, there is a single input set and a single output set, without branches or concatenations. The feedforward propagation in an MLP is essentially the sequential composition of the actions of each layer:

$$\boldsymbol{f}_{\text{FNN}}(\boldsymbol{x}) = \left(\boldsymbol{f}_L \circ \boldsymbol{f}_{L-1} \circ \cdots \circ \boldsymbol{f}_1 \circ \boldsymbol{f}_0\right)(\boldsymbol{x}) \tag{2.6}$$

In summary, this section has delineated a comprehensive mathematical framework encapsulating the concepts of neurons, layers, and their interconnections as distinct elements within a graph structure. This formalization provides a robust foundation for understanding and analyzing the mechanics and dynamics of feedforward neural networks, in particular those applied to the context of smart cities.

2.2 Deep learning paradigm

The essence of neural networks transcends their architectural complexity or the vast number of parameters they encompass. Their true power and uniqueness lie in their extraordinary learning and adaptation capabilities. This learning process, a hallmark of neural networks, is a key achievement in artificial intelligence, underpinned by a blend of mathematical elegance and complexity.

To fully grasp the potential and capabilities of neural networks, it's crucial to understand their foundational mechanism - the learning process. While the architecture defines a neural network's capacity, learning actualizes its potential. This process enables the modeling of complex non-linear relationships, the discovery of latent patterns, and the extraction of knowledge from data.

2.2.1 Optimization problem

At its core, the process of learning in neural networks is done by solving an optimization problem: given a dataset composed of inputs and outputs, we need to find particular weights $\boldsymbol{W}$ and biases $\boldsymbol{B}$ such that the predicted outputs of the network is minimally close to the true outputs. The closedness between two outputs can be expressed through the "loss" function, also rarely known as "cost" function. This kind of problem is derived from Machine Learning (ML), as indeed the main objective is to build a model which "follows the data", hence it is said to be data-driven. In the following subsection we describe in detail such function.

2.2.2 Loss function

In neural network learning, particularly crucial is the concept of a loss function. This function is fundamental in assessing the accuracy of a model's ability to emulate functions based on data. Loss functions act as key performance indicators by quantifying the gap between the model's predictions and the actual data. They not only define the task's objective but also measure the model's deviation from achieving this objective. Crucially, optimization algorithms depend on the contours of the loss function to locate regions where prediction errors are minimized. The choice of an appropriate loss function is therefore vital, as it directly impacts the learning strategy of the model. A well-chosen loss function can streamline training efficiency and bolster the model's ability to generalize effectively. Conversely, an unsuitable loss function might impede the optimization process, leading to inferior model performance.

The most general representation of the loss function involves taking in input the full training dataset, without explicitly indicating how the single samples are combined, but the most common kind of loss function computes the distance between the predicted label and the corresponding ground truth label and the reported value is averaged on all the samples. Mathematically it can be shown as the following function:

$$L(f(\mathbf{X};\mathbf{\Theta}),\mathbf{Y}) = \frac{1}{N}\sum_{i=1}^{N} \ell(f(\mathbf{X}^{(i)};\theta),\mathbf{Y}^{(i)}) \tag{2.7}$$

with f being the action of a model, and *ell* symbolizes the loss for an individual data point.

In the context of regression problems, where the goal is to predict continuous outputs, the loss function typically computes a real distance between the predicted and actual values. This distance effectively captures the magnitude of deviation in numerical terms. Common examples of such loss functions are the Mean Squared Error (MSE) and the Mean Absolute Error (MAE).

Conversely, in classification tasks, the output of a model is a probability distribution over CC classes. The loss function in this scenario is designed to measure the divergence between the predicted probability distribution and the actual class labels. The Cross-Entropy Loss is a prime example used in classification problems. It quantifies the difference in terms of probability, providing a measure of how well the model's probability predictions align with the actual class labels.

This distinction in loss function design underscores the varying nature of regression and classification tasks. While regression loss functions focus on the magnitude of numerical errors, classification loss functions emphasize the accuracy of probabilistic predictions, aligning the learning process with the specific demands of each task type.

In the following we define the mostly known loss functions and the recently developed losses.

List of loss functions

For Regression Tasks we have the following:

- **Mean Squared Error (MSE):** This remains a fundamental loss function for regression, measuring the average of the squares of the errors between predicted and actual values. It emphasizes larger errors due to squaring and is particularly sensitive to outliers.

- **Mean Absolute Error (MAE):** This function computes the average of absolute differences between target and predicted values. It is less sensitive to outliers compared to MSE.
- **Huber Loss:** Combining the best of MSE and MAE, Huber Loss is less sensitive to outliers in data than MSE. It is quadratic for small errors and linear for large errors, determined by a threshold δ.
- **Quantile Loss:** This loss function is used in quantile regression, providing a way to estimate the conditional quantiles of a response variable distribution in the linear model. It's particularly useful in scenarios where distributions are not symmetric, and upper or lower bounds predictions are needed.
- **Log-Cosh Loss:** A smooth version of the MSE, log-cosh loss is the logarithm of the hyperbolic cosine of the prediction error. This function provides the advantage of being almost as robust to outliers as MAE while maintaining the differentiability of MSE.

For Classification Tasks:

- **Binary Cross-Entropy (BCE):** Used for the binary classification tasks, it calculates the cross-entropy loss between the predicted class and the actual label.
- **Categorical Cross-Entropy Loss (a.k.a. Log Loss or Cross Entropy Loss):** The most common loss function for multi-class classification problems. It measures the performance of a classification model whose output is a probability between 0 and 1.
- **Hinge Loss:** Often used with Support Vector Machines (SVMs), hinge loss is primarily employed for "maximum-margin" classification, particularly important in "hard" classification scenarios.

- **Focal Loss:** An advanced variant of cross-entropy, introduced by Lin *et al* [24], focal loss helps to handle class imbalance by assigning more weight to harder, misclassified examples. It's especially useful in object detection where the imbalance between the background and object classes is significant.
- **Kullback-Leibler (KL) Divergence:** Also known as relative entropy, KL Divergence is used in scenarios involving discrete probability distributions, measuring how one probability distribution diverges from a second, reference distribution. Not to confuse with the KL divergence loss used under the context of Variational AutoEncoders [25].

There are further loss functions defined for object detection, but these will be defined in Section 3.2.6.

2.2.3 Gradient descent-based methodologies

After choosing the loss function, we need to find a strategy to minimze such loss. Different kind of optimization exists, but the mostly used family of methodologies for DL is the gradient descent-based method. In the following, we provide the mostly general definition:

Definition 2.2.1. *Given a dataset of n samples, with $n \geq 2$, and given a loss function $L(f(\mathbf{X};\boldsymbol{\Theta}),\mathbf{Y})$, with $f(\cdot;\boldsymbol{\Theta})$ denoting the model of interest, the parameters $\boldsymbol{\Theta}$ can be iteratively updated through a gradient descent scheme. This process is defined as follows:*

$$\boldsymbol{\Theta}^{(t+1)} = \boldsymbol{\Theta}^{(t)} - g(L(f(\mathbf{X};\boldsymbol{\Theta}),\mathbf{Y}),\boldsymbol{\lambda}),\ \forall t > 0, \tag{2.8}$$

where t is the iteration step, g is a parametric function with parameters $\boldsymbol{\lambda}$ which characterizes the iterative method.

In particular, the mostly simple representation is the gradient descent, which is defined with the following update rule:

$$g(L(f(\mathbf{X};\mathbf{\Theta}),\mathbf{Y}),\mu) = \eta\nabla_{\mathbf{\Theta}}L \tag{2.9}$$

with $\nabla_{\mathbf{\Theta}}$ the gradient of the loss function with respect to each parameter and $\eta > 0$ the learning rate. The gradient operator is defined as the following:

$$\nabla L(\mathbf{\Theta}) = \begin{bmatrix} \frac{\partial L}{\partial \mathbf{\Theta}_1} \\ \frac{\partial L}{\partial \mathbf{\Theta}_2} \\ \vdots \\ \frac{\partial L}{\partial \mathbf{\Theta}_n} \end{bmatrix} \tag{2.10}$$

Here we assume that $\mathbf{\Theta}$ can be represented as a vector of n scalar parameters $[\mathbf{\Theta}_1, \mathbf{\Theta}_2, \ldots, \mathbf{\Theta}_n]$. Such gradient vector is oriented to the maximum ascent of the loss landscape. This is why the gradient descent method uses the negation operation to direct the update towards the steepest descent in the loss landscape. The learning rate ∇ plays a critical role in controlling the magnitude of these updates, balancing the speed of convergence against the risk of overshooting minimal points. In a successive section we will describe the mostly known optimization procedures for DL, which is also appliable in the context of Smart City.

2.2.4 Backpropagation algorithm

Backpropagation stands as the cornerstone algorithm enabling the efficient training of neural networks. Fundamentally, it is a mechanism for propagating errors backward from the output towards the input layer, allowing for the adjustment of weights in a manner that minimizes the overall error of the network. This algorithm underpins the training of virtually all types of neural networks, making it indispensable for deep learning. The essence of backpropagation lies in its iterative, layer-by-layer computation of gradients of the loss function with respect to each weight in the network. Here, we will describe the algorithm for the case of a MLP, which is defined by Equation 2.6

for simplicity.

In a neural network composed of H layers $\mathcal{L}_1, \ldots, \mathcal{L}_H$, let us define Z^h the aggregation output at layer $\mathcal{L}_h$. Let $\boldsymbol{W}^h$ and $\boldsymbol{b}^h$ the weights matrix and the bias vector for layer $\mathcal{L}_h$. The propagated error for the h-th layer, denoted by δ^h, is calculated based on the error from the subsequent layer as follows:

$$\delta^h = (\boldsymbol{W}^{h+1})^\top \delta^{h+1} \odot \sigma'_h(Z^h) \tag{2.11}$$

where σ'_h denotes the derivative of the activation function applied at the layer, and $\odot$ signifies the Hadamard (element-wise) product. The partial derivatives of the loss function L with respect to the weights and biases are then given by:

$$\frac{\partial L}{\partial \boldsymbol{W}^h} = \delta^h (\boldsymbol{x}^{h-1})^\top \quad \text{and} \quad \frac{\partial L}{\partial b^h} = \delta^h \tag{2.12}$$

with $\boldsymbol{x}^h$ the state of the layer $\mathcal{L}$ from $\boldsymbol{x}^{h-1}$. This recursive mechanism, hallmarking the backpropagation process, iteratively computes the gradients of the loss function for all parameters by moving backward from the output towards the input layer, thereby facilitating the efficient training of deep neural networks.

This can be done to all the parameters of the layers because there exists the so-called "chain rule" from calculus: let L, f and g three real functions such that it is possible to define the composition $f \circ g \circ L$. Let x, z and y such that $z = f(x)$, $y = g(z)$ and $l = L(y)$. The derivative of L with respect to x can be equivalently expressed as the following:

$$\frac{dL}{dx} = \frac{dL}{dy} \cdot \frac{dg}{dz} \cdot \frac{df}{dx} \tag{2.13}$$

In the layered structure of neural networks, this rule becomes invaluable. It elucidates how changes in weights within one layer influence the overall loss, through their cumulative effect as propagated across subsequent layers.

2.2.5 Beyond the gradiend descent optimization algorithm

Optimization is crucial in the training process of neural networks, directing the iterative adjustment of parameters to enhance task performance [1]. Although gradient descent forms the basis of this process, the complex, high-dimensional loss surfaces encountered in deep learning introduce specific challenges [26]. One example is the abundant presence of local minima in the loss function can influence the converge of the method. This complexity has led to the development of sophisticated optimization methods, which include strategies like adaptive learning rates, momentum, or second-order optimization techniques [27].
One of the most important gradient descent-based methodologies, and also the mostly used due to its flexibility, is the Adam (Adaptive Moment Estimation), introduced by Kingma *et al* [28]. Its update rule is defined as the following:

$$\boldsymbol{\Theta}^{(t+1)} = \boldsymbol{\Theta}^{(t)} - \frac{\eta}{\sqrt{\hat{v}^{(t)} + \epsilon}} \hat{m}^{(t)}, \tag{2.14}$$

where:

- η is the learning rate.
- $\hat{m}(t)$ and $\hat{v}(t)$ are bias-corrected estimates of the first (mean) and second (uncentered variance) moments of the gradients, respectively.
- ϵ is a small constant added for numerical stability (to avoid division by zero).
- The moment estimates are computed as:

$$m^{(t)} = \beta_1 m^{(t-1)} + (1-\beta_1)\nabla_{\boldsymbol{\Theta}} L(f(\mathbf{X}; \boldsymbol{\Theta}^{(t)}), \mathbf{Y}),$$
$$v^{(t)} = \beta_2 v^{(t-1)} + (1-\beta_2)(\nabla_{\boldsymbol{\Theta}} L(f(\mathbf{X}; \boldsymbol{\Theta}^{(t)}), \mathbf{Y}))^2,$$

  where β_1 and β_2 are hyperparameters that control the exponential decay rates of these moving averages.

The Adam algorithm is particularly effective due to its adaptive learning rate, which allows for larger updates for infrequent parameters and smaller updates for frequent ones. It has emerged as a highly popular optimization algorithm within the deep learning community, praised for its consistent performance across a diverse range of tasks and models. Its effectiveness, coupled with minimal requirements for hyperparameter adjustments, highlights its significant role in the training of neural networks, making it well-suited for handling sparse gradients and non-stationary objectives, common in many deep learning applications.

Chapter 3

Deep Learning Applications

Two significant areas of deep learning application has been relevant in the context of smart cities: Time Series Forecasting and Object Detection. This chapter is structured to provide an in-depth analysis of each domain, encompassing their historical development, conventional methodologies, and the advent of deep learning models, with an emphasis on current state-of-the-art approaches.

In the section dedicated to Time Series Forecasting, the discourse commences with an examination of the historical development of forecasting methods. The evolution is traced from traditional statistical models, such as AutoRegressive Integrated Moving Average (ARIMA) and Exponential Smoothing, to more advanced machine learning techniques. These conventional methods form the bedrock of time series analysis, offering fundamental insights into pattern recognition in temporal data. The narrative then progresses to contemporary deep learning approaches, particularly focusing on the capabilities of Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks. These modern methodologies are elucidated in the context of their superior ability to model and predict complex temporal dependencies, representing a paradigm shift in forecasting accuracy and reliability.

The second section, Object Detection, delves into the domain of computer vision, charting its progression from early feature-based detection methods to the innovative deep learning-based approaches that dominate the field today. The historical context outlines the limitations of traditional object detection methods, which were often constrained by the high variability and dimensionality of image data. This sets the stage for the introduction of deep learning models, particularly Convolutional Neural Networks (CNNs), which have revolutionized object detection. The section meticulously analyzes the mechanics of these models, culminating in a discussion on cutting-edge techniques such as the You Only Look Once (YOLO) framework. This analysis not only highlights the technical advancements in object detection but also underscores the profound impact of deep learning in enhancing accuracy and computational efficiency. Overall, Chapter 3 offers a comprehensive and scholarly examination of Time Series Forecasting and Object Detection within the broader context of deep learning applications. Through a critical review of historical developments, standard models, and the transformative influence of deep learning, the chapter provides a nuanced understanding of these two pivotal areas, underscoring the significant strides made in the field and pointing towards future directions and potential advancements.

3.1 Time series forecasting

Time series forecasting represents a critical aspect of data science, where the primary objective is to deduce and anticipate future trends from temporal data. At the heart of such analyses lies the concept of a time series, which is mathematically construed as a sequence $\{\boldsymbol{y}_t\}$, where $\boldsymbol{y}_t$ symbolizing an observation noted at a specific time t, and these observations are indexed in increasing chronological order [29]. These observations are methodically ordered in a chronological sequence [29]. Time series are categorized into two principal types: univariate and multivariate. A univariate time series encapsulates sequential observations of a singular variable over time, exemplified by metrics such as a city's daily temperature or a product's monthly sales. It is characterized by a single datum y_t at each time point t. Conversely, multivariate time series expand upon this by recording a set of observations at each time instance, denoted as $\boldsymbol{y}_t = (y_{1,t}, y_{2,t}, \ldots, y_{n,t})$ for each t, where n represents the number of variables observed [30].

The capability to predict time-series data, irrespective of whether it is univariate or multivariate, offers substantial benefits. Precise forecasts can yield critical insights, leading to enhanced strategies and improved decision-making in various sectors. The field of time-series forecasting, with its rich historical background, has seen the development of diverse methodologies. Initially, the focus was on linear statistical models adept at elucidating basic temporal patterns in the data [31]. The emergence of Machine Learning (ML) and Deep Learning (DL) brought forth models capable of discerning more complex, non-linear relationships, a trait particularly beneficial in multivariate time series analysis [1]. However, with the increasing complexity and volume of data, challenges such as high dimensionality, complex interdependencies between series, noise, and real-world unpredictabilities have become more pronounced. Traditional models, while valuable, often struggle to address these complexities comprehensively. This has led to a growing interest in ensemble and hybrid methods in recent years [32]. These methods combine multiple forecasting techniques to achieve more robust, accurate, and comprehensive predictions. The integration of neural networks, especially, has added depth and adaptability to these approaches, showing promising outcomes in addressing the nuanced challenges of multivariate series prediction.

3.1.1 Historical Evolution of Forecasting Methods

The journey of forecasting has been integral to analytical endeavors, where the main goal is to predict future events based on historical data. The transition from empirical to systematic forecasting methodologies signifies a substantial leap in the field. These evolving methods introduced probabilistic approaches, allowing for the consideration of uncertainties intrinsic to forecasting.

Classical methods

The mid-20th century heralded significant progress in time-series forecasting with models grounded in robust mathematical principles. A pivotal development was the Exponential Smoothing method by Holt [33]. This method, premised on the idea that recent observations serve as potent predictors, assigns varying weights to historical data, thereby enabling adaptive forecasting. Holt's model is mathematically represented as:

$$\hat{y}_{t+1} = \alpha y_t + (1 - \alpha)\hat{y}_t, \tag{3.1}$$

where α, is the smoothing parameter, y_t is the observed value at time t, and $\hat{y}_t$ is the forecasted value.

Following Holt's groundwork, the Holt-Winters Exponential Smoothing method emerged, adept at handling seasonal variations in datasets [34]. Another milestone was the introduction of the ARIMA model, which seamlessly combined autoregressive, differencing, and moving average components to model a wide array of time-series data characteristics [35]. The ARIMA(p, d, q) model is formulated as:

$$\left(1 - \sum_{i=1}^{p} \phi_i L^i\right) (1 - L)^d y_t = \left(1 + \sum_{i=1}^{q} \theta_i L^i\right) \varepsilon_t, \tag{3.2}$$

where d, p and q denote the autoregressive (AR), differencing, and moving average (MA) orders, respectively, and L is the lag operator. ϕ_i and θ_i are parameters associated with the AR and MA segments, respectively, and ε_t is white noise.

To encompass seasonal effects, the SARIMA model was conceived, extending ARIMA with seasonal components [35]. This model is represented as SARIMA$(p, d, q)(P, D, Q)_s$, where P, D, Q are the seasonal orders, and s indicates the number of periods within a season. The SARIMAX model further incorporated exogenous variables (X) into the SARIMA framework, broadening its applicability by accounting for impacting external influences on the time-series [36].

The evolution of forecasting methods also led to the adoption of State Space Models and Kalman Filtering, particularly advantageous in scenarios with non-linearities, missing data, and structural shifts [37]. Transfer Function Models facilitated the analysis of the influence of exogenous variables on primary time-series. The STL method, delineating time-series into trend, seasonality, and residuals, enabled more nuanced forecasting techniques [38].

This progression in classical forecasting methods laid a solid foundation for contemporary approaches. The advancements in these models epitomize the analytical evolution from simplistic linear methods to more sophisticated techniques capable of addressing the intricate dynamics of time-series data.

The later decades of the 20th century marked a significant shift in forecasting methodologies, fueled by advancements in statistical techniques and burgeoning computational power. During this period, Fourier and spectral analyses emerged as pivotal tools, focusing on the frequency components of time-series data and offering novel perspectives in understanding temporal patterns [39].

As the 21st century dawned, traditional statistical approaches like ARIMA and Exponential Smoothing were augmented by innovative models designed to tackle a diverse array of forecasting challenges. Notably, the TBATS model, incorporating Box-Cox transformation, ARIMA errors, trend, and seasonal components, brought a nuanced approach to time-series analysis, effectively addressing challenges in diverse fields ranging from climatology to economics [40]. Concurrently, the multilinear orthogonal autoregressive (MOAR) model and the Block Hankel Tensor ARIMA (BHT-ARIMA) demonstrated advancements in capturing complex temporal dependencies within data [41]. Additionally, the introduction of the Prophet model by Facebook's research team offered a flexible, component-based approach to forecasting, particularly adept at handling seasonal fluctuations and holiday effects [42].

Machine learning models

In parallel with these developments, machine learning (ML) began to reshape the landscape of forecasting. Leveraging its ability to identify non-linear patterns, ML techniques transcended the limitations of traditional model-based assumptions. Support Vector Machines (SVM), initially devised for classification, were adeptly adapted to forecasting through Support Vector Regression (SVR). Particularly in financial forecasting, SVR emerged as a powerful tool due to its proficiency in managing high-dimensional data and complex non-linear relationships [43].

Ensemble methods, notably Random Forests, introduced another layer of sophistication to forecasting. By aggregating predictions from multiple decision trees, Random Forests not only enhanced prediction accuracy but also effectively countered overfitting. This attribute rendered them invaluable in sectors like energy forecasting, where a myriad of factors impact predictions [44]. Ensemble methods exemplify the strength of collective insights, harnessing the diversity of individual models to achieve more robust and accurate predictions.

The evolution of machine learning models in time-series forecasting marked a pivotal transition from classical statistical methods to more dynamic, data-driven approaches. This shift underscored the growing importance of adaptability and complexity in analytical models, as they endeavored to capture the nuanced and intricate patterns inherent in time-series data.

Deep learning models

The integration of deep learning in the realm of time series forecasting has marked a significant shift, primarily owing to the versatility and robustness of neural networks. These networks, particularly adept at approximating a broad spectrum of continuous functions, have shown remarkable adaptability in addressing complex forecasting challenges [45].

Among various architectures, Recurrent Neural Networks (RNNs), especially Long Short-Term Memory (LSTM) units and Gated Recurrent Units (GRU), have emerged as powerful tools for modeling temporal sequences. These architectures are uniquely equipped to capture intricate temporal dependencies inherent in time-series data. Notable advancements include Amazon's LSTM-based DeepAR model, which has made significant strides in probabilistic forecasting [46], and the N-Beats model, recognized for its superior performance in the M4 forecasting competition [47].

The field has also seen a surge in hybrid models that synergistically combine traditional statistical approaches with neural networks, enhancing prediction accuracy and robustness. Examples include the integration of ARIMA models with neural network architectures [48, 49] and state space models with deep learning approaches [50]. Additionally, techniques like matrix factorization, used for dimensionality reduction in complex datasets, are being explored to improve forecasting accuracy [51–53]. Innovations in sequential convolutional-recurrent models [54] and hybrid Exponential Smoothing with RNN models [55] have shown promising results in high-dimensional series forecasting.

Recurrent neurons

The foundation RNN-based models is the recurrent neuron, which is an extension of the neuron defined in Chapter 2, which is also known as feedforward neuron. Recurrent neurons introduce a crucial element: memory. Unlike feedforward neurons, recurrent neurons have loops in their architecture, allowing information to persist. This looping mechanism enables recurrent neural networks (RNNs) to maintain a form of internal state or memory, making them inherently adept at processing sequential or temporal data.

Let $\boldsymbol{y}_t$ the time series at instance t and each istance has N features. The recurrent neuron is defined by the following update rule:

$$\boldsymbol{h}_t = \sigma_h \left(\boldsymbol{W}_{hh} \boldsymbol{h}_{t-1}^\top + \boldsymbol{W}_{yh} \boldsymbol{y}_t^\top + \boldsymbol{b}_h \right) \tag{3.3}$$

$$o_t = \sigma_o \left(\boldsymbol{w}_{ho} \boldsymbol{h}_t^\top + b_o \right) \tag{3.4}$$

Where:

- $\boldsymbol{h}_t \in \mathbb{R}^Q$, $Q > 0$ is the hidden state at time t,
- $o_t \in \mathbb{R}$ is the scalar output at current time step,
- $\boldsymbol{W}_{yh} \in \mathbb{R}^{Q \times N}$ is a vector of parameters associated with the input, termed *input weights*;
- $\boldsymbol{W}_{hh} \in \mathbb{R}^{Q \times Q}$ is a matrix of parameters associated with the hidden state, named *hidden state weights*,
- $\boldsymbol{w}_{ho} \in \mathbb{R}^Q$ is a vector of parameters which is used to extract the output from the hidden state;
- $\boldsymbol{b}_h \in \mathbb{R}^Q$ is a vector of parameters named the hidden bias;
- $b_o \in \mathbb{R}^Q$ is the output bias.
- σ_h and σ_o are activation functions.

This definition makes it evident that the hidden state at time t depends on the input at time t and the hidden state at time $t-1$, therefore the hidden state can be considered as the "memory", since the neuron can "remember" the patterns characteristic of a time series. Further extensions can be done to this neuron to improve the control information flow, for instance by "gates", as done by LSTM and GRU layers.

3.1.2 Ensemble Methods

The trajectory of forecasting methodologies through time reveals an increasing awareness of the distinct strengths and weaknesses inherent in various approaches. Traditional statistical models, though proficient in identifying linear trends and patterns, often encounter challenges when confronted with complex, non-linear datasets marked by high dimensionality and interdependencies. In contrast, Machine Learning (ML) and Deep Learning (DL) techniques, while adept at handling such complexities, are sometimes vulnerable to overfitting and noise sensitivity.

Ensemble methods, gaining prominence in the ML domain, operate on the foundational principle that a collective output from multiple models tends to yield more accurate and reliable results than any single model. This notion stems from the understanding that no singular approach is universally superior across all cases. As a result, ensemble techniques aim to harness the cumulative strengths of various models while counterbalancing their individual weaknesses. This philosophy has profoundly influenced predictive modeling, with extensive research demonstrating the efficacy of ensemble methods in enhancing generalization, reducing overfitting, and improving prediction accuracy [56].

Fundamental Pillars of Ensemble Approaches

The efficacy of ensemble methods is anchored in three core principles:

1. **Diversity in Decision Making**: This principle underscores the benefits of integrating decisions from a range of models. By aggregating diverse viewpoints, ensemble methods can explore and interpret complex datasets more thoroughly than single models, often leading to more nuanced and robust conclusions. This diversity is especially advantageous when considering errors; models with uncorrelated errors, when combined, tend to produce a net reduction in overall error [57].

2. **Overcoming Model Limitations**: Recognizing the inherent constraints of individual models, ensemble methods seek to transcend these boundaries. The "No Free Lunch theorem" posits that no single model is best-suited for every sce-

nario. Ensemble approaches, by amalgamating various models, strive to leverage their collective strengths, thereby mitigating the limitations of each constituent model [58].

3. **Quantification of Uncertainty**: Ensemble methods, particularly in Bayesian contexts, excel in offering a more nuanced view of possible outcomes. Rather than providing mere point estimates, these methods present a distribution over potential results, allowing for a deeper understanding of the reliability and variability of predictions. This aspect is crucial in scenarios where understanding the uncertainty of a forecast is as vital as the forecast itself [59].

In summary, ensemble methods in forecasting represent a strategic synthesis of multiple models, aiming to achieve a more balanced and comprehensive approach to predicting future trends. These methods stand as a testament to the evolving landscape of predictive analytics, where the combination of varied analytical perspectives often yields the most insightful and dependable results.

Applications of Ensemble Methods in Time-Series Forecasting

The evolution of time-series forecasting has been significantly marked by the integration of ensemble methodologies. These approaches, often employing linear combinations of model outputs, optimize weights to enhance overall predictive accuracy. An illustration of this is the MSEF technique [60], which merges multiple forecasting models post-decomposition of time-series data using tools like STL. Similarly, the FYTS approach [61] optimizes weights based on model performance metrics to form an aggregated prediction. In sector-specific applications, such as in tourism [62], ensemble methods involve decomposing the time-series into fundamental components, individually modeling each through ensembles, and then integrating these predictions.

Dynamic weight adjustment is a key aspect in some methodologies, exemplified by an ensemble approach for wind energy forecasts [63]. This method iteratively adjusts model weights in response to recent performance, favoring models that have shown effectiveness in the near past, thereby maintaining continuous efficiency.

Beyond linear combination strategies, recent literature on ensemble methods has explored methodologies that delineate the distinct contributions of individual models. EASTPAS, for example, integrates multiple forecasts using statistical ensemble techniques such as bagging, boosting, and stacking [64]. The role of genetic algorithms in ensemble forecasting, as seen in [65], involves optimizing classifier weights through evolutionary computation. This approach has been applied in various forecasting domains, including electricity demand and financial predictions, demonstrating the versatility of genetic algorithms in handling a range of complex forecasting scenarios [66, 67].

Emerging computational intelligence methods are also contributing significantly to this domain. For instance, an approach in wind power forecasting involves training segment-specific neural networks on decomposed time-series data [68]. Another ensemble technique amalgamates predictions from various machine learning models to formulate a unified forecast [69]. The advancement of deep learning, particularly in refining RNN architectures, has enhanced ensemble frameworks. A notable example is an ensemble that combines forecasts from different RNN configurations using a stacking algorithm, thus leveraging the unique strengths of each RNN variant [70]. Hybrid ensemble techniques, such as those combining different model categories, have proven effective in tasks like wind speed prediction and ultra-short-term power demand forecasting [71, 72].

In addition, meta-learning strategies are gaining traction, exemplified by the use of neural networks to identify correlations between data features and optimal forecasting models [73]. This approach enhances both the integration of predictions and insights into model suitability in relation to data characteristics.

Despite the sophistication of these advanced methodologies, the efficacy of fundamental ensemble techniques remains substantial. For instance, simple algebraic combinations, such as uniform averaging or median selections, have often outperformed more complex methods [74]. This finding was echoed in the M4 forecasting competition, highlighting the enduring relevance of basic ensemble approaches [75]. As the field progresses towards intricate ensemble strategies, it is crucial to balance innovation with the proven effectiveness of traditional methods, ensuring reliable and robust forecasting outcomes.

3.2 Object detection

Object detection, a crucial and vibrant field within computer vision, has experienced a transformative shift with the advent of deep learning. This domain, fundamentally concerned with identifying and locating objects within digital images, has evolved from rudimentary techniques to advanced deep learning models capable of astonishing accuracy and speed.

Historically, object detection hinged on manual feature extraction and simplistic algorithms, which were often constrained by limited computational power and the complexity of real-world imagery. This is why, differently from the forecasting problem, the methodologies to tackle the object detection problem were developed much later in the late 2000's. Indeed, the emergence of deep learning, especially convolutional neural networks (CNNs), has radically altered this landscape. Today, object detection systems can not only identify a wide array of objects within an image but also determine their precise location, scale, and even categorize them into various classes.

The impact of these advancements extends well beyond academic interest, permeating numerous aspects of modern life. From enhancing security through surveillance systems to revolutionizing the retail industry with automated checkouts, object detection plays a pivotal role. In medical imaging, it assists in identifying abnormalities, thereby aiding early diagnosis and treatment plans. In the automotive industry, it is a cornerstone technology for developing autonomous vehicles, enabling them to navigate and interact safely with their environment.

The following subsections delve deeper into specific aspects of object detection, starting from the foundational concept of image pattern recognition in Section 3.2.1. We will explore the nuances of various domains where object detection is applied, discuss the intricacies of the object detection task itself, and examine the deep learning models that have driven the field's rapid progress. The section will culminate with an in-depth look at the "You Only Look Once" (YOLO) framework, a groundbreaking approach in object detection, and its main evolutions of that have further enhanced the capabilities of object detection systems.

As we navigate through these subsections, we aim to not only understand the technical underpinnings of these developments but also to appreciate their practical implications and the challenges that lie ahead in this dynamic and ever-evolving field.

3.2.1 Image Pattern Recognition

In the domain of object detection, a nuanced understanding of images as a distinct form of data is essential. Images, fundamentally different from textual or numerical data, are a composite of complex visual elements. They embody intricate spatial relationships, diverse textures, color depth, and luminosity, each contributing to the rich visual narrative they present. Recognizing patterns in such a multi-dimensional and often non-linear space is both a challenge and an opportunity for deep learning models.

Images offer a multidimensional view of the world, capturing scenes through a matrix of pixel intensities, where each pixel represents a blend of color and intensity. This representation is far from straightforward due to the variables at play - from lighting variances to perspective shifts. The challenge in image pattern recognition, therefore, lies in abstracting consistent, defining features from these variabilities.

The spatial arrangement within images adds another layer of complexity. Objects do not exist in isolation but are contextualized by their surroundings. This relational aspect is crucial for accurately deciphering scenes. For instance, the understanding of relative sizes and positions can reveal depth, distance, and the interplay between different elements in an image.

Textures in images are equally telling. They provide cues about the surface characteristics of objects, aiding in differentiating one object from another. Similarly, color depth and luminosity add dimensions of analysis, offering insights into the environmental conditions of the scene.

Given the already established detailed discussion on Convolutional Neural Networks (CNNs) in Chapter 2, it is pertinent to recall their significance in this context. CNNs, with their hierarchical feature extraction capabilities, are adept at navigating through these layers of complexity. Starting from basic edge detection to understanding intricate object forms, CNNs systematically unveil the nuances embedded within images.

This section sets the groundwork for exploring the applications, challenges, and advancements in object detection using deep learning. While acknowledging the comprehensive treatment of CNNs and their operational intricacies in Chapter 2, we now focus on how these models specifically adapt to and excel in the realm of image pattern recognition.

We now briefly go in more detail about the challenges the images pose under the computer vision umbrella.

Image intricacies

In the domain of computer vision, the role of images transcends traditional data analysis, presenting a unique set of challenges and opportunities. Images are not merely collections of data points but complex assemblies of visual information, encompassing color, texture, and spatial relationships. This complexity arises from the inherent high dimensionality of images, where each pixel can be seen as a separate dimension. Consequently, the analysis of images in computer vision goes beyond linear or one-dimensional approaches, requiring an understanding of spatial dynamics and the interplay of various visual elements.

The efficacy of image pattern recognition in computer vision is closely tied to the quality of the images being analyzed. Factors such as resolution, noise, and the conditions of image capture play a pivotal role in determining the quality of visual data. High-resolution images provide a wealth of details but come with increased computational demands, while low-resolution images might lack critical information necessary for accurate analysis.

Noise in images, stemming from various sources, can significantly hamper the identification of key patterns and features. Effective computer vision systems must, therefore, be equipped with strategies to either reduce or compensate for such noise. Additionally, the environmental conditions under which an image is captured—such as lighting, angle, and background—can greatly influence the appearance of objects in the image, adding another layer of complexity to the pattern recognition task.

3.2.2 Convolutional Neuron

The convolutional neuron represents a cornerstone in the field of computer vision. At its core, a convolutional neuron is designed to mimic the way the human visual cortex interprets visual stimuli, focusing on the hierarchical and spatial relationships inherent in visual data. This specialization allows convolutional neural networks (CNNs) to efficiently process and learn from image data by recognizing patterns such as edges, textures, and more complex structures, making them exceedingly effective for tasks ranging from image classification to object detection.

Unlike traditional neurons in a fully connected layer that learn global patterns across the entire input space, convolutional neurons learn local patterns through the application of a convolution operation. This operation involves sliding a filter or kernel over the input image and computing the dot product between the kernel and local regions of the input, producing a feature map. This feature map highlights the presence of specific features or patterns at various locations in the image. Through stacking multiple convolutional layers, each with its unique set of learnable filters, a CNN can learn increasingly complex and abstract representations of the input data.

Mathematically, the operation of a convolutional neuron can be described as the convolution between an input feature map $\mathsf{X} \in \mathbb{R}^{M \times NxC}$ and a kernel or filter $\mathsf{K} \in \mathbb{R}^{K \times K \times C}$, augmented with the addition of the bias term b. For a given position (i, j) in the input space, this operation is defined as follows:

$$\boldsymbol{Y}_{ij} = \sigma\left(\sum_{m=1}^{K}\sum_{n=1}^{K}\sum_{c=1}^{C}\mathsf{X}_{i+\tilde{K}-m,j+\tilde{K}-n,c}\mathsf{K}_{m,n,c} + b\right) \tag{3.5}$$

Where o_{ij} is the convolution output, $\tilde{K} = \lfloor K/2 \rfloor$ is half the size of the kernel rounded to integer by floor, σ the activation function.

A key attribute of this convolutional process is its ability to maintain translational invariance, meaning that the network can recognize a learned feature anywhere in the input field. This property enables CNNs to identify patterns and features regardless of their exact position in the input image, making them exceptionally suited for tasks that require pattern detection across varying spatial contexts. This inherent translational invariance underpins the convolutional neuron's remarkable efficacy in a wide array of pattern recognition applications, even in the face of spatial transformations and variations within the input data [76]. This innovation paved the way to the investigation of object detection through deep learning methodologies, resulting in many works which helped in various applications in the context of Smart City.

3.2.3 Task definition

The intricacy of object detection lies in its requirement to accurately classify each object while also determining its location within a myriad of potential backgrounds and contexts. This task becomes even more challenging considering the variability in object sizes, shapes, and appearances, as well as the presence of overlapping objects and varying environmental conditions such as lighting and occlusion [77].

Furthermore, object detection models must be robust enough to handle diverse scenarios. For instance, in a street scene, a model may need to detect and differentiate between pedestrians, vehicles, street signs, and other elements, all within a single frame. This requires not only high precision in classification but also in localization to ensure accurate and reliable detection. This task's complexity is amplified by the need for real-time processing, particularly in applications such as autonomous driving or real-time surveillance. In these scenarios, the speed of detection is crucial. The models must process and analyze the visual data quickly enough to allow for immediate responses or decision-making [78].

Object detection, therefore, represents a significant advancement in the capabilities of computer vision systems. It combines the challenges of recognizing and classifying multiple objects, determining their exact locations, and doing so with speed and accuracy. This multifaceted nature of object detection makes it a key component in various applications, ranging from security and surveillance to autonomous navigation and interactive media.

Advancements in deep learning, particularly with the development of sophisticated neural network architectures, have greatly enhanced the effectiveness of object detection. These advancements have enabled the creation of models that can handle the high-dimensional data of images, extract relevant features, and perform complex classification and localization tasks with increasing accuracy and efficiency. As a result, object detection continues to be an area of active research and development, driving forward the capabilities of computer vision and its applications in the real world.

Mathematical framework

In this section, we extend the mathematical framework developed in Chapter 2 to rigorously define the task of object detection.

Definition 3.2.1. *Let* $\mathbf{X} \in \mathbb{R}^{H \times W \times C}$ *an image. An object on the image is a tensor* $\mathbf{O} \in \{0,1\}^{H \times W}$ *such that each value at coordinate* (i, j) *is 1 if the object is visible at* (i, j)*, 0 otherwise.*

Definition 3.2.2. *A bounding box of an object O is defined as the rectangle $\boldsymbol{b} = \{x_c, y_c, w, h\}$, with:*

- *x_c and y_c the positional coordinates of the center of the rectangle;*
- *w and h the width and the height of the rectangle*

such that $O \subseteq R$, with $R = \{(x, y) \in [0,1] \times [0,1] \; s.t. |x - x_c| <= w/2 \wedge |y - y_c| <= h/2\}$.

We note here that we do not explicitly define the bounding box as the smallest rectangle containing the object, as we are interested also in capturing objects which are partially occluded, and therefore the rectangle has to cover the missing part of the object.

Definition 3.2.3. *Given the set of the classes $\mathcal{C} = \{c_1, \ldots, c_C\}$, with $C \geq 1$, a detection object corresponding to object O is the set $\boldsymbol{o} = (\boldsymbol{b}, c)$, with $\boldsymbol{b}$ the bounding box of the object, and $c \in \mathcal{C}$ the corresponding class.*

Suppose we have a set of C classes, $\mathcal{C} = \{c_1, \ldots, c_C\}$, with $C \geq 1$. For any image $\mathbf{X}$ there are two possibilities:

- there are no objects with class $c \in \mathcal{C}$
- there are $K = K_{\mathbf{X}}$ objects each with class $c \in \mathcal{C}$, $K > 0$

So, the problem of object detection can be defined as following: given the set of the classes $\mathcal{C}$, for any image tX, the task is to identify all $K = K_{\mathbf{X}}$ detection objects $\boldsymbol{o}_1, \ldots, \boldsymbol{o}_K$ such that each detection object $\boldsymbol{o}_j$ corresponds to object O_j in the image $\mathbf{X}$.

Object detection metrics

Object detection metrics are integral in evaluating the performance of models in correctly identifying and localizing objects within images. Intersection over Union (IoU) is a primary metric used in object detection. It quantifies the overlap between the predicted detection object and the ground truth, calculated as following:

$$\text{IoU}(A, B) = \frac{|A \cup B|}{|A \cap B|} \tag{3.6}$$

where A and B are the two rectangles derived from the detection object, $|C|$ is the area of C. A higher IoU value indicates a higher accuracy of the model in localizing objects. With this metric, it is possible to define the True Positive (TP) as follows:

Let τ a value between 0 and 1 named the overlap confidence. A predicted detection object $\boldsymbol{o}_P$ is defined a TP if it correctly identifies a single label $\boldsymbol{o}_O$ both in terms of class and in terms of overlap if $IoU(R_{\boldsymbol{o}_P}, R_{\boldsymbol{o}_P}) > \tau$. A False positive (FP) is a predicted detection object such that there is no ground truth objects which satisfies the previously mentioned criteria. In case there are multiple predicted detection objects possibly matching a ground truth, the predicted detection object with the highest IoU to the ground truth is selected as TP, while the remaining predicted detection objects are checked with the closest ground truth objects or are considered FP. Finally, a False Negative (FN) is defined as a ground truth object which does not have a matching predicted detection object in terms of class and overlap. These metrics help in understanding the types of errors a model is prone to, which is crucial for practical applications where specific types of errors may have more severe consequences than others.

Precision and Recall are closely linked to FP and FN. They are defined as follows:

$$\text{Precision}(c) = \frac{TP}{TP + FP} \tag{3.7}$$

$$\text{Recall}(c) = \frac{TP}{TP + FN} \tag{3.8}$$

with c a class. Precision measures the proportion of true positives among all detected objects, providing insight into the accuracy of the positive predictions. Recall evaluates the model's capability to identify all actual objects in the dataset. Balancing precision and recall is often necessary, especially in applications where either avoiding false positives or detecting all objects is critical.

The F1 Score is the harmonic mean of precision and recall. It is particularly useful when there's a need to find a balance between precision and recall, and when the dataset is imbalanced with respect to class distribution.

Average Precision (AP) and Mean Average Precision (mAP) further extend these concepts. AP calculates the average precision value for recall over the interval from 0 to 1 and is useful for ranking objects based on detection confidence. mAP, an extension of AP for multiple classes, computes the mean of AP across all classes, providing an overall performance metric for multi-class object detection models.

These metrics collectively offer a comprehensive assessment of an object detection model's performance, addressing different aspects such as accuracy, error types, and the ability to balance various detection needs. Understanding and appropriately utilizing these metrics is crucial for developing and refining effective object detection systems.

3.2.4 Traditional methods

Before the era of deep learning, object detection relied on various traditional methods that were based on handcrafted features and machine learning algorithms. These methods, while innovative for their time, faced limitations in handling complex and dynamic real-world scenarios.

Haar Cascades The Haar Cascade classifier, introduced by Viola and Jones [79], is a seminal method in the field of computer vision, particularly for face detection. Utilizing Haar-like features, which are simple contrast features between rectangular areas, this method applies a cascade approach for efficient detection. It is described mathematically as:

$$\text{Haar-Feature}(\mathbf{x}) = \sum_{\mathbf{x} \in \text{Region}_1} I(\mathbf{x}) - \sum_{\mathbf{x} \in \text{Region}_2} I(\mathbf{x}) \tag{3.9}$$

where $I(\mathbf{x})$ represents the pixel intensity at position $\mathbf{x}$. Though fast and suitable for real-time applications, its effectiveness is limited to detecting primarily frontal faces under specific expressions and lighting conditions, struggling with complex backgrounds or occlusions.

Scale-Invariant Feature Transform (SIFT) SIFT [80] is a method for detecting and describing local features in images. It is renowned for its invariance to image scale, rotation, and partial invariance to illumination changes. Mathematically, a SIFT feature is defined as:

$$\text{SIFT}(\mathbf{x}) = \{\text{Scale}(\mathbf{x}), \text{Orientation}(\mathbf{x}), \text{Descriptor}(\mathbf{x})\} \tag{3.10}$$

where $\text{Scale}(\mathbf{x})$ and $\text{Orientation}(\mathbf{x})$ determine the keypoint characteristics, and $\text{Descriptor}(\mathbf{x})$ is the descriptor vector. SIFT excels in feature matching across varied conditions but is computationally intensive, making it less suitable for real-time applications and less effective in uniformly textured or extremely cluttered scenes.

Histogram of Oriented Gradients (HOG) The HOG method, developed by Dalal *et al* [81], involves calculating and histogramming gradient directions in localized portions of an image. The process of computing a HOG descriptor involves several steps, starting with gradient computation and followed by orientation binning, as represented by:

$$\text{HOG}(\mathbf{I}) = \sum_{\mathbf{x} \in \text{Block}} \text{Histogram}(\nabla \mathbf{I}(\mathbf{x})) \tag{3.11}$$

where $\mathbf{I}$ is the image and $\nabla \mathbf{I}(\mathbf{x})$ is the gradient of the image at point $\mathbf{x}$. This method, particularly effective for pedestrian detection, captures edge and gradient structures well but struggles with non-rigid objects, varying poses, and low-contrast, cluttered environments.

Deformable Part Models (DPM) DPMs, introduced by Felzenszwalb *et al* [82], are a class of models in object detection that cater to objects with deformable parts. They represent an object by multiple parts and spatial relations between them. A DPM can be mathematically represented as:

$$\text{DPM}(\mathbf{O}) = \sum_{i=1}^{n} \text{Part}_i(\mathbf{p}_i) + \sum_{i,j} \text{Spatial-Relation}(\mathbf{p}_i, \mathbf{p}_j) \tag{3.12}$$

where $\mathbf{O}$ is the object, $\text{Part}_i(\mathbf{p}_i)$ represents the i-th part at position $\mathbf{p}_i$, and $\text{Spatial-Relation}(\mathbf{p}_i, \mathbf{p}_j)$ captures the spatial relationship between parts. While DPMs handle occlusions and varied poses effectively, they are computationally demanding and require substantial hand-tuning and training data.

While these methods laid the groundwork for modern object detection, they faced challenges in terms of adaptability, robustness to variations in scale, viewpoint, lighting conditions, and occlusions. This paved the way for the emergence of deep learning-based approaches, significantly improving object detection by learning features directly from data.

3.2.5 Deep learning models

The introduction of Convolutional Neural Networks (CNNs) heralded a paradigm shift in image pattern recognition, particularly in object detection. Their ability to automatically and hierarchically extract features from images, ranging from basic edges to complex object structures, has significantly advanced the field.

Deep learning models, especially those leveraging CNN architectures, have shown remarkable efficacy in object detection tasks. This effectiveness stems from their ability to learn intricate feature representations directly from large amounts of image data. These models have not only surpassed traditional methods in accuracy but also in their ability to generalize across diverse and complex visual environments.

Object detection in the context of deep learning can be formulated as an optimization problem. The goal is to find a deep learning model that minimizes a defined loss function, capturing the discrepancy between predicted and actual object detections. Therefore, such loss function should take into account both the accuracy of object localization (bounding box regression) and the correctness of object classification.

Two classes of deep learning models emerged as the state-of-the-arts at time they were released: the two-stage detectors and single-shot detectors [83]. The former ones, also known as Region Proposal-Based, work by splitting the problem in locating the boxes for the objects, then classifying each identified object, while the latter ones, also known as Regression/Classification-Based, solve the problem with a single model. The next subsections delve in detail about these two kinds of models.

Two-stage detectors

Two-stage detectors represent a pivotal approach in deep learning for object detection, characterized by their distinct, sequential processing stages. This methodology primarily focuses on achieving high accuracy in detecting and classifying objects within images.

One of the primary components in the initial phase of two-stage detectors is the *Selective Search* algorithm. Introduced by Uijlings *et al* [84], Selective Search is a region proposal technique that combines the strength of both exhaustive search and segmentation. It aims to identify a diverse set of potential object regions (proposals) from an image, which are then processed in the subsequent stages of the model. Selective Search works by hierarchically grouping similar pixels and regions based on texture, color, and size compatibility, eventually merging these into larger regions. This process results in a comprehensive set of region proposals that capture objects at various scales and aspect ratios.

One of the early and influential models employing this technique is the Region-based Convolutional Neural Network (R-CNN), proposed by Girshick *et al* [85]. R-CNN utilizes selective search to scan the input image and proposes potential object regions.

Each region is then resized and fed into a Convolutional Neural Network (CNN) to extract a feature vector. These vectors are subsequently classified using a set of Support Vector Machines (SVMs), one for each object class. The final step involves applying a linear regression model to tighten the bounding boxes of the detected objects.

Before Fast R-CNN and Faster R-CNN, He *et al* introduced the Spatial Pyramid Pooling network (SPP-NET)[86], which pools features in multiple regions, producing fixed-length outputs irrespective of the input size. This method solved a problem with the Fully Connected component of the CNN in R-CNN constraining the input size, which would create distortion-based artefacts. Such innovation allows the network to operate on images of various sizes, enhancing its applicability.

Fast R-CNN [87] improves upon the R-CNN by introducing a Region of Interest (RoI) pooling layer. This model uses a single CNN to extract feature maps from the entire image and then applies RoI pooling to each region proposal to obtain fixed-size feature vectors for classification and bounding box regression. Faster R-CNN [88] enhances this process by integrating the Region Proposal Network (RPN), a fully convolutional network that simultaneously predicts object bounds and objectness scores. This integration significantly speeds up the region proposal step, making the process more efficient.

Region-based Fully Convolutional Network (R-FCN), proposed by Dai *et al* [89] optimizes the process by employing position-sensitive score maps. Unlike previous models, R-FCN maintains a fully convolutional architecture, with the last few layers being position-sensitive RoI pooling layers. This design significantly reduces the computational complexity while maintaining high detection accuracy.

Feature Pyramid Networks (FPN) [90] tackles the challenge of scale variability by constructing a top-down architecture with lateral connections. This network builds a pyramid of feature maps at different levels, enabling the detection of objects at various scales through a single, unified network.

A completely different approach is done by He *et al* with the proposed Mask R-CNN architecture [91]. Indeed, instead of solving directly the object detection task it solves the more general Instance Segmentation task, which involves detecting and segmenting each instance of a specific class. In order to do so, it adds a branch for predicting object masks, enhancing the network's capabilities for instance segmentation tasks. This model separates the task of bounding box recognition and object mask prediction, allowing for more precise object localization.

These advancements in two-stage detectors have significantly contributed to the field of object detection, offering solutions that balance accuracy and computational efficiency. Each model brings unique technical features to address specific challenges in object detection, making them suitable for a wide range of applications where precision and reliability are paramount.

Single-stage Detectors

Single-stage detectors have emerged as a cornerstone in the realm of deep learning for object detection, distinctively characterized by their efficiency and expediency. Foregoing the separate region proposal stage characteristic of two-stage detectors, these models adeptly combine object classification and bounding box regression into one seamless operation. This integration is particularly advantageous in scenarios necessitating real-time responses, such as video surveillance or autonomous driving.

The architecture of single-stage detectors is ingeniously crafted to scan the entire image in one comprehensive sweep. Utilizing a dense grid-based approach, these models predict object categories and spatial coordinates across the image. Each grid cell is independently responsible for detecting objects within its confines, simultaneously classifying the cell and predicting bounding boxes along with confidence scores for the detected objects.

Preliminary works focused on object detection using deep neural networks (DNNs), either as a regression or classification problem, but the challenges were several. Szegedy *et al.* explored using a DNN for regression to create a binary mask for each image and then deduce the bounding boxes (BBs) for object detection [92]. This method struggled with overlapping objects, and the bounding boxes, created by simply upsizing the mask, were often imprecise. Pinheiro et al. [93] designed a CNN with two separate parts: one to create general segmentation masks and another to evaluate the likelihood of an object being centered in a given image patch. This model was efficient as it could produce class scores and segmentations together, with most CNN operations being shared. Yoo et al. proposed AttentionNet [94], a CNN that used an iterative classification method for object detection. The model worked by progressively zeroing in on an object, starting from the image corners and using a series of directions to narrow down to the precise bounding box. While innovative, this model was inefficient, particularly when dealing with multiple object categories and requiring a two-step process. Najibi *et al* created G-CNN [95], a grid-based detector which aimed to adjust a fixed grid of bounding boxes iteratively to closely fit around detected objects. Starting with a multi-scale grid, it trained a regressor to move and resize these grid elements towards the objects. However, G-CNN faced challenges in accurately detecting small or closely overlapping objects.

As it can be seen, many proposed models face various challenges which made their adoption in niche contexts. Major progress was registered when two models were proposed: You Only Look Once (YOLO) [96] and the Single Shot MultiBox Detector (SSD) [97].

YOLO adopts a unique approach by dividing the image into a grid, with each grid cell responsible for predicting objects located within it. This model is known for its exceptional speed, making it a popular choice for real-time detection tasks. More details on the architecture, loss are detailed in the next section.

On the other hand, SSD operates on multiple feature maps at different resolutions, allowing it to handle objects of various sizes more effectively. It discretizes the output space of bounding boxes into a set of default boxes over different aspect ratios and scales per feature map location. During prediction, SSD adjusts these default boxes and predicts the presence of object class instances.

Other single-stage models different from both YOLO and SSD, such as RetinaNet [24], have also emerged, introducing novel concepts like Focal Loss to address the class imbalance problem, which is prevalent in single-stage detectors due to the large number of background samples.

3.2.6 YOLO and derivatives

YOLO, which stands for "You Only Look Once", is a groundbreaking approach in the domain of object detection, renowned for its unique characteristics and architectural innovations. Introduced by Joseph Redmon *et al* [96], YOLO revolutionized the way object detection was approached, shifting from the traditional two-stage process (region proposal followed by classification) to a single-stage, unified framework. This shift marked a significant advancement in terms of speed without substantially compromising accuracy.

The core principle of YOLO lies in its name - the model only looks once at the image, processing the entire picture in a single pass. This is a stark contrast to earlier methods that would scan an image multiple times to detect objects. YOLO divides the input image into a grid of size $S \times S$, with $S > 0$, and each grid cell predicts B bounding boxes plus their confidence, with $B > 0$ and the class probabilities of the entire cell, hence the output is a tensor $\mathbf{Y} \in \mathbb{R}^{S \times S \times (5*B+C}$.

One of the most distinctive features of YOLO is its speed. By simplifying the object detection pipeline and using a single neural network to predict bounding boxes and class probabilities simultaneously, YOLO can achieve real-time performance, making it well-suited for applications requiring fast and efficient object detection, such as video surveillance and autonomous vehicles.

The architecture of YOLO, as shown in Fig. 3.1 is based on 24 CNNs and 2 final Fully Connected layers. Before the first CNN, the three channels in the image are reduced to 1 through a 1×1 convolution layer, followed by a 3×3 convolution layer. The

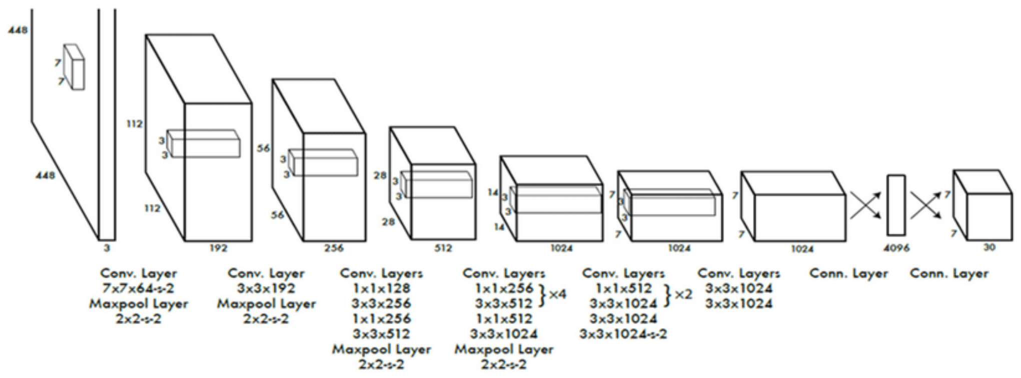


Figure 3.1: Detailed YOLO architecture, CC Redmon *et al.* [96]

CNN layers are responsible for extracting and learning spatial hierarchies of features from the input image. The final activation function is a linear function, while the remaining activation functions in each layer is a function named *leaky rectified linear* and is defined as follows:

$$\sigma(z) = \begin{cases} z & \text{if } z > 0, \\ 0.1z & \text{otherwise} \end{cases}$$

Considering that the output is fixed, and also due to the presence of boxes with very low confidence, a procedure named Non Maximum Suppression (NMS) is used to remove such superflous bounding boxes.

The loss function in YOLO is a critical component that ensures the model is accurately predicting both the class and location of objects. Such loss is defined as the sum of the following loss components:

- *Localization loss*:

$$\begin{aligned} L_{\text{loc}} = \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\ + \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \end{aligned} \tag{3.13}$$

- *Confidence Loss*:

$$\begin{aligned} L_{\text{conf}} = &\sum_{i=0}^{S^2}\sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i\right)^2 \\ &+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2}\sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i\right)^2 \end{aligned} \tag{3.14}$$

- *Classification Loss*:

$$L_{\text{classif}} = \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c\in\text{classes}} \left(p_i(c) - \hat{p}_i(c)\right)^2 \tag{3.15}$$

Here, $\mathbb{1}_{i}^{\text{obj}}$ is 1 if the object appears in the i-th cell, 0 otherwise, while $\mathbb{1}_{ij}^{\text{obj}}$ is 1 if the j-th prediction object in cell i has the highest IoU with the ground truth compared to the other prediction objects in the same cell and $\mathbb{1}_{ij}^{\text{noobj}}$ is 1 if cell i has no ground truth but the j-th prediction is in cell i. λ_{coord} and λ_{noobj} are regularization coefficients which are used to reduce the training instability caused by the respective terms [96]. The loss function is designed to penalize the model more for mistakes in the bounding box predictions compared to the class predictions, reflecting the importance of accurate localization in object detection tasks.

YOLO has undergone several iterations and improvements since its initial release, with each version bringing enhancements in terms of accuracy, speed, and the ability to detect smaller objects. The reasoning why the major focus has been given to YOLO is due to the low execution time compared to the other methodologies [98]. Here we detail the most prominent developments:

- **YOLOv2 (YOLO9000)**: Proposed by Joseph Redmon and Ali Farhadi, [99] this model was a major upgrade focusing on improving the speed and accuracy of the original YOLO model. One of its notable features was the introduction of anchor boxes, which allowed the model to predict more bounding boxes and improve recall. Another improvement is the usage of Batch Normalization to improve training stability and the model generalization. YOLOv2 also introduced the concept of multi-scale training, enabling the model to detect objects at various sizes by dynamically resizing the input image during training. Additionally, it utilized a Darknet-19 architecture, a streamlined version of the original Darknet model, enhancing the model's efficiency.

- **YOLOv3**: YOLOv3, proposed by the same authors of YOLOv2, [100] further refined the model with various enhancements. It incorporated three different scales for detection, improving the model's ability to detect small objects. YOLOv3 utilized a more complex Darknet-53 architecture, which, despite its complexity, maintained a balance between speed and accuracy. The model also introduced the use of logistic regression for objectness prediction, replacing the softmax function used in previous versions. This change improved the precision of the model, especially in multi-label classification scenarios.

- **YOLOv4**: YOLOv4, proposed by Bochkovskiy *et al* [101] aimed to optimize the balance between speed and accuracy while remaining accessible to a wider range of hardware platforms. It integrated several new techniques, such as the use of the CSPDarknet53 backbone, spatial pyramid pooling, and path aggregation network for feature extraction. YOLOv4 also incorporated advanced data augmentation techniques, like Mosaic and CutMix, and employed more sophisticated training strategies. The model showed improvements in both speed and accuracy compared to its predecessors.

- **YOLOv5**: YOLOv5, developed by the Ultralytics team [102] while not an official version from the original YOLO authors, marked a significant development in the YOLO series. It is known for its fast training and inference times, making it highly suitable for real-time applications. YOLOv5 offers various model sizes (small, medium, large, and extra-large) to cater to different computational and accuracy requirements. It incorporates PyTorch for easier deployment and introduces several improvements, such as auto-learning bounding box anchors and more efficient layer operations. Even without academic paper as of now, it has been used with good results in various applications [103–105].

Each iteration of YOLO has contributed to the evolution of object detection, making these models more robust, accurate, and efficient. They have broadened the scope of real-world applications, from autonomous driving [106] to real-time monitoring systems, where quick and reliable object detection is crucial. In Chapter 5, we will discuss about the current SoTA of object detection by using YOLO algorithm, which is YOLOv8, proposed by the same authors as YOLOv5 [107].

Part II

Deep Learning and Smart Cities

Chapter 4

ENCODE: time series forecasting prediction

Multivariate time-series forecasting stands as a complex challenge in the realm of data analytics. This task involves identifying intricate correlations within and across series, often compounded by high dimensionality and the presence of noisy data. As detailed in [108], an effective forecasting method must demonstrate both precision, in terms of a bespoke loss function catered to the problem's specifics, and resilience. Recent years have seen significant efforts towards achieving these goals. A review of the literature indicates a preference for approaches that leverage the strengths of various forecasting models, a strategy underscored by the results of the M4 [109] and M5 [110] competitions. These methods typically involve the integration of multiple techniques, each adept at recognizing specific patterns, to derive a more robust overall prediction. Generally, these approaches are categorized into two main types: *ensembles*, which integrate several predictions through a single connecting layer, and *hybrid methods*, where distinct models operate in tandem.

The success of these approaches is rooted in the foundational research of [111] and [112]. Inspired by these findings and the empirical studies conducted in the preceding section, an innovative ensemble framework has been formulated specifically for multivariate time-series forecasting. This system adeptly uncovers correlations and latent patterns in multivariate datasets using advanced deep learning approaches. It has been aptly named Ensemble Neural Combination for Optimal Dimensionality Encoding in Time-series Forecasting, or ENCODE, reflecting its core functionality and purpose.

Mathematical notation Before detailing the ENCODE model, given the multivariate context, we adopt the following notation conventions:

- $T = \{1, ..., N\}$ denotes the set of time indices;
- The number of time-series is denoted with M, hence we have $X^1, ..., X^M$, with $X^j = \{x_t^j\}_{t \in T}, \ \forall j \in \{1, ..., M\}$;
- With $\mathbf{x}_t \in \mathbb{R}^M, \ \forall t \in T$ we denote an M-dimensional vector of time-series observations at time t, i.e., $\mathbf{x}_t = [x_t^1, ..., x_t^M]$;
- H represents the forecast horizon, corresponding to the expected output matrix $X_{t:t+H} = [\mathbf{x}_j]_{j \in \{t,...,t+H-1\}} \in \mathbb{R}^{H \times M}$;
- the model utilizes K previous lags, meaning the input corresponds to $X_{t-K:t} = [\mathbf{x}_j]_{j \in \{t-K,...,t-1\}}$;
- we define the flattened version of $X_{t-K:t}$, done for algorithmic purposes, as the following:

$$\mathbf{i}_t = [x_{t-K}^1, ..., x_{t-1}^1, x_{t-K}^2, ..., x_{t-K}^M, ..., x_{t-1}^M]_{j \in \{1,...,MK\}} \in \mathbb{R}^{KM}; \tag{4.1}$$

- S seasonalities are identified and compiled into a matrix $S_t \in \{0, 1\}^{H \times S}$, where each row encapsulates the variables representing these characteristics for step-ahead predictions.

The proposed method addresses the complexities of time-series forecasting, such as high dimensionality, noise, missing values, and scale variation, through a structured,

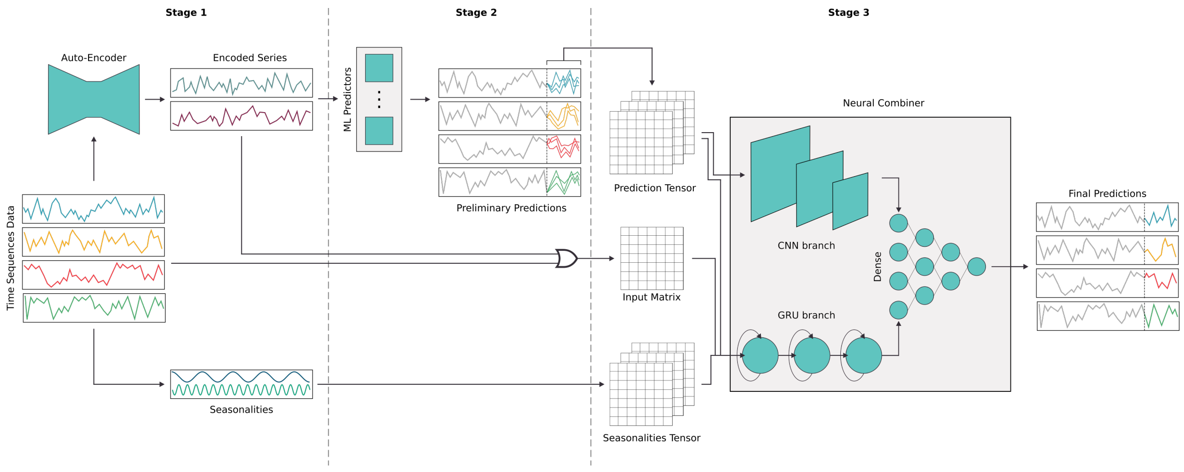


Figure 4.1: The framework consists of three integral components: the Autoencoder (AE), which compresses the lagged time-series to lower its dimensionality and filter out noise; individual prediction models that forecast future values using the encoded variables; and the *Combiner*, which integrates these individual forecasts with seasonal factors and encoded variables to generate the final prediction output.

modular approach. This approach commences with a specialized preprocessing phase, followed by a three-tiered framework aimed at deriving novel insights from data. Initially, an autoencoder (AE) condenses lagged variables into a compressed latent space, setting the stage for further analysis. The second tier involves employing a spectrum of machine learning and statistical models to yield individual forecasts. The culmination of this process is an advanced hybrid neural network, inspired by the methodology outlined earlier in this chapter, functioning as a *Combiner* of these predictions. Notably, this design integrates GRU layers instead of LSTM to streamline the network by reducing parameters and enhancing the convergence process. Inputs for the *Combiner* encompass individual forecasts, seasonal patterns, and an input matrix that either retains the original or employs encoded time-series data, contingent on the data's dimensional complexity. The overarching aim is to leverage both temporal and spatial interconnections among input variables, thereby augmenting the accuracy of predictions.

In terms of innovation, the distinctiveness of the proposed framework, while classified under ensemble-based methods, is not solely in its individual components, but rather in their synergetic interaction to augment performance. Specifically, two features are particularly noteworthy: the AE phase's adept handling of outliers and the neural network's tailored architecture designed to capture temporal dynamics and the interrelations among individual predictive elements.

To contextualize ENCODE within the existing landscape of competing methodologies, it is vital to highlight the proven efficacy of ensemble and hybrid methods in time-series prediction. Like some hybrid methods, ENCODE emphasizes dimensionality reduction and noise filtration. However, ENCODE AE component ensures a non-linear projection of data into a lower-dimensional space. Moving to ensemble techniques, the typical modus operandi involves generating a range of predictions, followed by an integration strategy for the final forecast. While ENCODE mirrors these steps, its *Neural Combiner* not only aggregates initial forecasts but also actively contributes to the prediction through its extraction of knowledge from the time-series data. Our proposal's alignment with these categories of methods is further elucidated in Table 4.1, which outlines the key features that set our approach apart from similar techniques.

4.1 Methodology

As previously highlighted, the ensemble consists of three primary components, detailed in Fig. 4.1 and Algorithm 1. Summarizing, our approach unfolds across three stages: initially, the AE processes the lagged time-series to reduce data dimensionality and filter out noise; subsequent to this, individual predictors utilize the encoded variables to forecast upcoming time-series values; finally, the *Combiner* assimilates individual forecasts, along with the seasonality and encoded variables, to produce the final prediction.

Taking into account the mathematical notation defined at the beginning of this chapter,

Ref.	Methods	Steps	Application Field	Multivariate	Multi Steps-Ahead
[113]	SL, ML and DL with NN combiner	2	General Applications	No	No
[63]	NN with median and globally weighted averages	4	Wind Energy	Yes	Yes
[71]	Ensemble of LSTM with SVR combiner	2	Wind Speed	No	Yes
[65]	ML and Heterogeneous Ensembling	2	General Applications	No	No
[70]	Ensemble of LSTM with RFR and Ridge as combiners	2	General Applications	No	Yes
[114]	SL with NN selector adequate forecasting	2	Industry Sales	No	No
[115]	ML and Wavelet NN	3	Sunspots	No	No
[116]	LSTM based on Ensemble Empirical Mode Decomposition	2	Oil Production	No	No
[117]	Regime identification model and multiple regression trees	2	Traffic Flow	Yes	No
[66]	ML methods and weighted Iterative Ensembling	4	Electricity	Yes	No
[118]	Ensemble of cloud and fuzzy models	2	Weather Time Series	No	No
[72]	Hybrid Ensemble Strategy with LSTM	2	Power Demand	No	Yes
[67]	Neuro-Fuzzy Ensemble Model	2	Finance	No	No
[119]	Four kinds of NN and dynamic weight combination	2	General Applications	No	No
[62]	Decomposition-Ensemble approach with NN	4	Hong Kong tourism demand	No	Yes
[120]	Set of NN with bagging ensembling approach	2	Chinese Stock Market Index	No	No
[121]	Sparse Representation and NN	3	Crude Oil Price	No	Yes
[122]	SL and weighted mean	3	Regional Photovoltaic Power	No	No
[123]	SL and SVR combiner	2	Electric Load Data	No	No
[124]	AutoEncoder and bootstrap aggregation	2	Crude Oil Price	Yes	No
ENCODE	AutoEncoder and Hybrid Neural Combiner	3	General Applications	Yes	Yes

Table 4.1: A comparative overview of methodologies, aligned with ENCODE in literature, is presented, detailing their approach, procedural steps, application domain, uni/multivariate framework, and the scope of their forecasting (single or multiple steps ahead). This enumeration encompasses a range of models, each referenced in scholarly works, and highlights their methodological underpinnings, the sequential stages involved in their processes, specific fields of application, whether they address univariate or multivariate data, and their capability in making short-term or long-term predictions.

the ensemble strategy can be written as the following function:

$$\phi = \phi(\cdot;\theta) : (\mathbf{i}, S) \in \mathbb{R}^{MK} \times \{0,1\}^{H\times S} \to \hat{X} \in \mathbb{R}^{H\times M} \tag{4.2}$$

In other words, the encoder's task is to process an input vector consisting of lagged values and a matrix representing seasonal patterns, leading to the generation of predicted outputs. The functioning of the encoder ϕ is largely influenced by a set of hyperparameters θ, which are optimized through Randomized Grid Search. For simplification in

Algorithm 1 ENCODE

```
1: {Pre-processing Stage}
2: Merge time-series $X^j$ into a dataset $X$.
3: Divide time interval $\{1, ..., N\}$ into $TrainingP = \{1, ..., n_{TrainingP}\}$, $TC\_Val = \{n_{TrainingP} + 1, ..., n_{Validation}\}$,
   and $Test = \{n_{Validation} + 1, ..., N\}$.
4: Split $TC\_Val$ into $TrainingC = \{n_{TrainingP} + 1, ..., n_{TrainingC}\}$ and $Validation = \{n_{TrainingC} +
   1, ..., n_{Validation}\}$
5: Fit the $[0, 1]$ Scaler and transform $X$.
6: Decompose time-series $\{I_t\}_{t \in TrainingP}$ to obtain seasonalities $\{S_t\}_{t \in \{1,...,N\}}$.
7: Obtain the vectors $\{I_t\}_{t \in \{1,...,N\}}$ of lagged inputs.
8: {Encoding Stage}
9: Train AutoEncoder using $\{I_t\}_{t \in TrainingP}$ as training set and $\{I_t\}_{t \in val}$ as EarlyStopping.
10: Encode all variables to obtain $\{\tilde{I}_t\}_{t \in \{1,...,N\}}$ vectors.
11: {Single Predictions Stage}
12: for Predictor $f \in \{1, ..., F\}$ do
13:     Define Hyperparameter Grid for predictor $f$: $\Theta_f$.
14:     Obtain the optimal hyperparameter $\theta_f$ via Randomized Grid Search.
15:     Fit $f$ on $\{\tilde{I}_t\}_{t \in TrainingP}$ using $\theta_f$ hyperparameter and obtain the predicted values $\{P_{f;t}\}_{t \in TC\_Val}$.
16:     Fit $f$ on $\{\tilde{I}_t\}_{t \in TrainingP \cup TC\_Val}$ using $\theta_f$ hyperparameter and obtain the predicted values $\{P_{f;t}\}_{t \in Test}$.
17: end for
18: Merge predictions $\{P_{f;}\}_{t \in TC\_Val \cup Test}$ to obtain $\{P\}_{t \in TrainingC \cup Test}$.
19: {Combiner Stage}
20: Define Hyperparameter Grid for Combiner: $\Theta$.
21: Obtain the optimal hyperparameter $\theta$ via Randomized Grid Search.
22: Train Combiner on $\{I_t\}_{t \in TrainingC}$, $\{P_t\}_{t \in TrainingC}$, and $\{S_t\}_{t \in TrainingC}$ data using $\theta$ hyperparameter and
    $\{I_t\}_{t \in Validation}$, $\{P_t\}_{t \in Validation}$, and $\{S_t\}_{t \in Validation}$ for the early stopping.
23: Exploit $\{I_t\}_{t \in TrainingC}$, $\{P_t\}_{t \in TrainingC}$, and $\{S_t\}_{t \in TrainingC}$ data to obtain final predictions for the Test data
    $\{\hat{X}_t\}_{t \in Test}$.
```

the algorithm, the input vector $\mathbf{i} \in \mathbb{R}^{MK}$ is managed as a single-row matrix $I \in \mathbb{R}^{1 \times MK}$.

Regarding the generation of data subsets for training and analysis, a train-validation-test split strategy is employed. Each segment has a specific role: the *Training-P* set is allocated for the training of individual predictors and, in cases of limited data, for the *Combiner* as well; the *Training-C* set is used for fine-tuning the hyperparameters of each predictor and for the training of the *Combiner*; the most suitable hyperparameters for the *Combiner* are determined using the *Validation* set; and the *Test* set is crucial for evaluating our ensemble model's performance against other established methods. This approach to data partitioning is depicted in Figure 4.2.

To evaluate the effectiveness of our proposed framework, we have conducted compar-

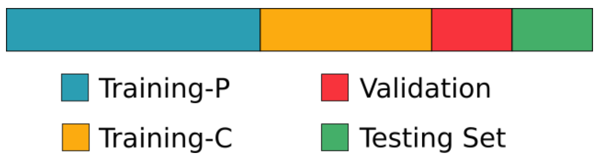


Figure 4.2: The data is divided into four parts: the *Training-P* set for training the encoder and predictors, the *Training-C* set for optimizing these algorithms and training the *Combiner*, with the best hyperparameters selected based on the Validation set, and the *Test* set for benchmark comparisons.

isons with the following methodologies:

- Individual predictions from models incorporated within our *Combiner*, namely Lasso, Ridge, Elastic Net, Random Forest Regression (RFR), XGBoost (XGB), ARIMA, and Support Vector Regression (SVR).

- A simple random walk approach, represented by the ARIMA(0,1,0) model. This model forecasts the future value x^j_{t+1} as the current value x^j_t plus a constant mean μ^j and a random noise $\sigma^j \varepsilon^j_{t+1}$, with σ^j constant, and ε^j_{t+1} signifying white noise. Thus, the forecast for x^j_{t+h} becomes $x^j_t + h\mu^j$ based on the training data.

- Mean and median ensemble strategies, calculated from the prediction vectors of the aforementioned models. These strategies take the average or the median of predictions from the individual models, respectively.

- Ensembles utilizing Particle Swarm Optimization (PSO, [125]) and Genetic Algorithm (GA, [126]). These techniques are applied to the same models as in point 1 to optimize the weights assigned to each model's predictions.

- The N-Beats model, as detailed in Oreshkin et al. [47], utilizing a Python implementation available on GitHub[1].

- The BHT-ARIMA model, as proposed by Shi et al. [41], with reference to its official GitHub repository[2].

[1]`https://github.com/philipperemy/n-beats`
[2]`https://github.com/huawei-noah/BHT-ARIMA`

- The Prophet model by Taylor and Letham [42].
- The Multivariate Time Graph Neural Network (MTGNN) model, according to Wu et al. [127], adapted to our specific requirements using the official GitHub repository[3].

4.1.1 First Stage: Encoding

The first stage of our ensemble strategy, as delineated in Lines `7-9` of Algorithm 1, utilizes an Autoencoder (AE) to transform the lagged variables of the time-series. This stage is focused on reducing the input vector $\mathbf{i}$ to a more compact form $\tilde{\mathbf{i}}$, while retaining essential data features. Mathematically, this is represented as a transformation:

$$\phi_1 : \mathbf{i} \in \mathbb{R}^{MK} \rightarrow \tilde{\mathbf{i}} \in \mathbb{R}^{V} \quad \text{where } V \ll MK \tag{4.3}$$

This process of encoding plays a crucial role in enhancing the model's robustness by diminishing the impact of potential anomalies or noise within the data. It is particularly beneficial when handling large-scale time-series data, as it reduces the computational demands for training both individual predictors and the final *Combiner*.

In terms of architecture, the AE employs dual GRU layers for both encoding and decoding phases, which has proven effective in various contexts. The size of the latent space is adjusted based on the problem's complexity and the computational resources available, typically ranging between $\frac{1}{4}$ and $\frac{1}{40}$ of the original input dimension. A dynamic learning rate (α) strategy is implemented, where training begins with a higher α and is gradually reduced, particularly when early stopping is triggered. The training ceases when either a minimal α is reached or no significant improvements in the loss function are observed. The central layers of both the encoder and decoder are optimally sized as the mean of the original and reduced dimensions, thus simplifying the hyperparameter selection process.

[3]`https://github.com/nnzhan/MTGNN`

The AE model training involves the *Training-P* and *Training-C* datasets, utilizing early stopping criteria based on the *Validation* set's performance. The encoded variables derived from the *Training-P*, *Training-C*, *Validation*, and *Test* sets are subsequently employed in the subsequent stages of the ensemble process.

4.1.2 Second Stage: Single Predictions

n the second stage of the ensemble process, outlined in Lines `10-17` of Algorithm 1, the focus shifts to generating individual forecasts. This stage utilizes the encoded vector $\tilde{\mathbf{i}}$ as an input for a variety of Machine Learning (ML) and Statistical Learning (SL) techniques. Each method is tasked with predicting future values for every series over the forecast horizon. The strength of this approach lies in its flexibility, allowing the inclusion of diverse predictive algorithms to enrich the array of forecasting strategies. This includes linear regression models like Lasso, Ridge, and ElasticNet, Support Vector Regression (SVR), tree-based models such as Random Forest Regression (RFR) and XGBoost (XGB), and the time-series specific model ARIMA. The latter, in particular, is fitted using the original time-series $\mathbf{i}$ rather than the encoded version $\tilde{\mathbf{i}}$. These methods were selected based on their proven efficacy, both as standalone models and as contributing inputs to the *Combiner*. Given F predictors, the function for the f-th predictor can be expressed as:

$$\phi_{2,f} : \tilde{\mathbf{i}} \in \mathbb{R}^V \rightarrow P_f \in \mathbb{R}^{H \times M} \tag{4.4}$$

where the resulting matrix $P_f = \phi_{2,f}(\tilde{\mathbf{i}})$ incorporates predictions across intervals and series. Collectively, this stage is represented as:

$$\phi_2 : \tilde{\mathbf{i}} \in \mathbb{R}^V \rightarrow P \in \mathbb{R}^{H \times MF} \tag{4.5}$$

with P being the compilation of all individual prediction matrices.

The hyperparameters for each regression model are optimized through Randomized Grid Search, adhering to the following steps:

1. Training each hyperparameter configuration for every algorithm on each series in the *Training-P* dataset.
2. Evaluating the performance on the *Training-C* dataset.
3. Selecting the hyperparameters that minimize the mean square error as the chosen configuration.

The predictors are then trained using the *Training-P* dataset, with their forecasts derived from the *Training-C* dataset. These predictions are crucial for refining the *Combiner*, as they provide insights into out-of-sample prediction accuracy. Ultimately, these models are retrained on the combined *Training-P*, *Training-C*, and *Validation* datasets, maintaining the identified hyperparameters, to yield the final forecasts for the *Test* dataset.

4.1.3 Third Stage: *Combiner*

The final segment of our ensemble framework, delineated in Lines `18-22` of Algorithm 1, introduces the *Combiner*. This crucial component amalgamates individual forecasts into the final prediction.

In this stage, the Convolutional Neural Network (CNN) part examines the collected predictions P, seeking patterns and interconnections that could augment the overall forecast efficacy. Simultaneously, the Gated Recurrent Unit (GRU) segment processes these predictions alongside the encoded variables $\tilde{\mathbf{i}}$ and seasonal factors S. Its role is to unravel temporal dynamics in the data and evaluate each predictor performance across varying scenarios. The insights gathered from both CNN and GRU elements are then merged through dense layers, culminating in the final forecast output. This process can be mathematically described as:

$$\phi_3 : (\tilde{\mathbf{i}}, P, S) \in \mathbb{R}^V \times \mathbb{R}^{H \times MF} \times \{0,1\}^{H \times S} \to \hat{X}_t \in \mathbb{R}^{H \times M} \tag{4.6}$$

Based on empirical findings, a *Combiner* architecture comprising three convolutional layers, two GRU layers, and two dense layers before the final output layer has demonstrated consistent reliability across diverse forecasting scenarios. Hyperparameter Optimization (HPO) is performed on the *Combiner*, focusing on specific hyperparameters like learning rate, weight decay, and dimensions of the layers, using a Randomized Grid Search approach. A learning rate decay strategy, similar to that used in the AE phase, is also implemented.

4.1.4 Final Remarks

In terms of incorporating exogenous variables, ENCODE framework acknowledges their significance in time-series forecasting. These variables are integrated by concatenating them with the target series before introducing them into the AE stage. Consequently, the resulting encoded variables $\tilde{\mathbf{i}}$ reflect patterns from both the target and exogenous series. These integrated insights persist through the individual prediction and *Combiner* stages, enriching the forecasting process.

Addressing seasonality is a key focus, in this work adhering to the methodology suggested in [36]. Seasonal components are identified and measured against the combined magnitude of seasonal and error components, retaining those with a significant ratio (above 0.05) for input into the *Combiner* as tensors.

Finally, non-stationarity in time-series data is a consideration well-handled by the ENCODE method. Particularly in the AE phase, GRUs efficiently filter the series, leveraging their innate ability to manage non-stationary data. This capacity extends to the individual predictors and the *Combiner*, ensuring the methodology effectively processes non-stationary time-series without compromising the accuracy and robustness of the forecasts.

4.2 Experimental Setup

Our experimental setup employs five datasets, with three being public and two private. These datasets are selected based on their relevance and representativeness in various domains. The AutoCorrelation Function (ACF) and Partial AutoCorrelation Function (PACF) are employed to determine the relevant lagged values for analysis. The public datasets used in our experiments include:

- **Electricity**[4]: This dataset comprises daily power consumption data from 370 customers over 2012-2014. After preprocessing, it contains 320 time-series with 1096 observations each. The dataset is divided into various subsets: *Training-P* with 700 observations, *Training-C* with 196 observations, and *Validation* and *Test* sets, each with 100 observations. The dataset includes forecasting for 1, 3, and 7-step horizons, considering weekly and annual seasonalities. A representative time-series from this dataset is shown in Figure 4.3.

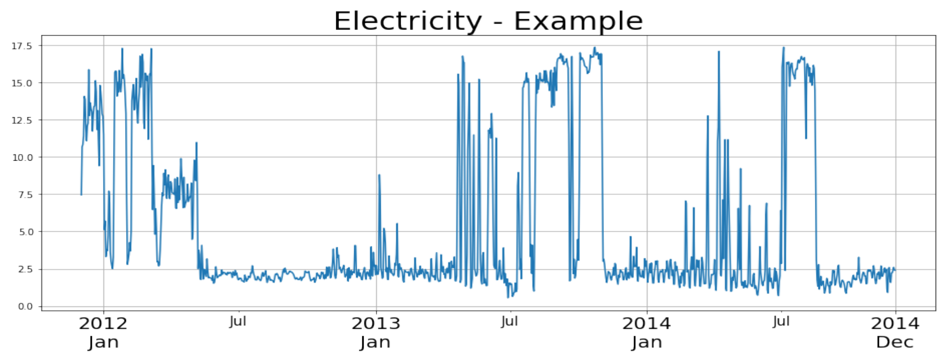


Figure 4.3: Sample time series from the Electricity dataset.

- **SST**[5]: Comprising ocean surface temperature data from 67 Pacific buoys spanning 2000-2019, the SST dataset, after filtering, contains 30 time-series with 2556 observations. Its data division includes 1461 samples for *Training-P*, a middle portion for *Training-C* and *Validation*, and the last 122 samples for *Test*. Pre-

[4]UCI Machine Learning Repository, `https://archive.ics.uci.edu/dataset/321/electricityloaddiagrams20112014`

[5]NOAA Tropical Atmosphere Ocean Project, `https://tao.ndbc.noaa.gov/`

dictions span 1, 3, and 7 steps ahead, with specific prediction lags. A sample from this dataset is depicted in Figure 4.4.

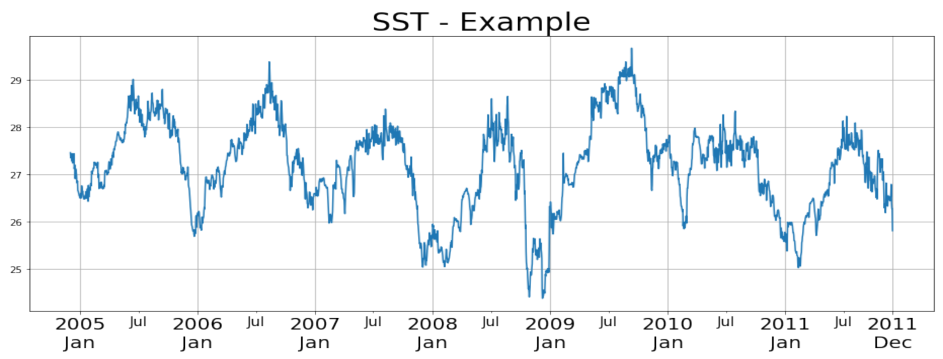


Figure 4.4: Sample time series from the SST dataset.

- **PeMS**[6] ([128]): This dataset includes data from California highways for March-May 2021. With 46 time-series and 1463 observations, it is divided into *Training-P* with 840 samples, *Test* with 168 samples, and the remainder for *Training-C* and *Validation*. Forecasting includes 12, 24, and 36-step horizons, considering daily and weekly seasonalities. Figure 4.5 showcases a sample from this dataset.

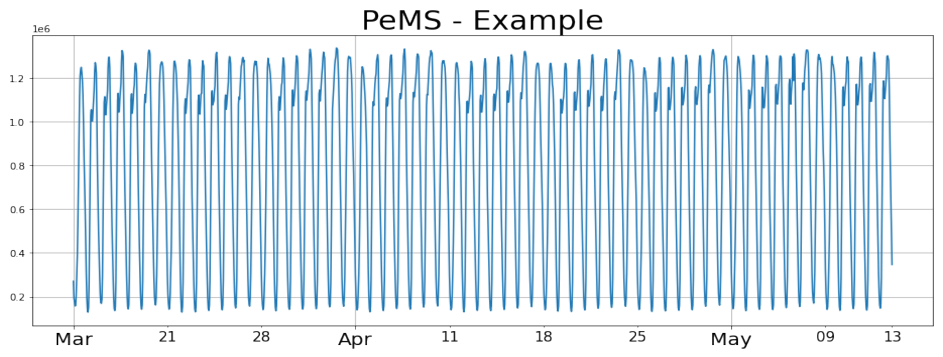


Figure 4.5: Sample time series from the PeMS dataset.

The private datasets include:

- **Healthcare**: This dataset contains daily hospital booking counts for allergy and pneumology in Campania, Italy, along with weather data. With 730 observations

[6] California Department of Transportation, `https://dot.ca.gov/programs/traffic-operations/mpr/pems-source`

across 13 variables, it is divided into *Training-P*, *Test*, and portions for *Training-C* and *Validation.* Forecasts include $1-7$ and $1-14$ steps ahead, with significant weekly seasonality due to weekends.

- **ToIT**: Focused on hourly parking occupancy rates, this dataset comprises 2099 observations. Its division includes *Training-P*, *Training-C*, *Validation*, and *Test* sets, accommodating forecasts for $1-24$ steps ahead. The dataset shows daily and weekly seasonality and incorporates nine exogenous weather series.

Our approach efficacy is evaluated using Root Mean Square Error (RMSE) and Mean Absolute Error (MAE) as loss functions. A block Cross-Validation technique is applied, considering the time-related nature of the datasets. For RMSE, the *Test* set is partitioned into Q segments while maintaining the temporal sequence:

$$Test = Test_1 \uplus \cdots \uplus Test_Q \tag{4.7}$$

With $\uplus$ the disjoint union, and:

$$Test_q = \{t_{q,1}, \ldots, t_{q,n_q}\} \quad \forall q = 1, \ldots, Q \tag{4.8}$$

Then, the error for each segment is assessed as follows:

$$RMSE_q = \sqrt{\frac{1}{n_q}\sum_{j=1}^{n_q} \|X_j - \hat{X}_j\|_2^2} \tag{4.9}$$

Where $X_j, \hat{X}_j \in \mathbb{R}^{H \times M}$ and $\|.\|_2$ is the l^2 norm applied element-wise. The final vector is:

$$\mathbf{RMSE} = [RMSE_1, \ldots, RMSE_Q] \in \mathbb{R}^Q \tag{4.10}$$

The evaluation metrics include the mean (*Mean*), standard deviation (*Std*), and their cumulative value (*Mean+Std*) indicating the confidence interval upper boundary. The MAE follows a similar assessment process. Ten folds are used for Cross-Validation in the Electricity, SST, and Healthcare datasets, and seven folds for PeMS and ToIT due to the hourly data and one-week test set. Data standardization is applied to ensure uniformity across all series during performance assessment, with scaling confined to training and validation datasets to avoid using future data points.

4.3 Results

In our analysis, we introduce a unified metric system to enable a more straightforward and equitable comparison across various prediction methods. This system is applied to each dataset and respective forecast horizon, treating them as individual tests. For every test, the error values associated with the j-th prediction method, denoted as ϵ_j, are standardized to lie within the range of 0 to 1. The standardized error, $\hat{\epsilon}_j$, is calculated as follows:

$$\hat{\epsilon}_j = \frac{\epsilon_j - \min \epsilon}{\max \epsilon - \min \epsilon} \quad j = \text{LASSO, Ridge}, \dots \tag{4.11}$$

The next step involves computing the average standardized error for each prediction technique across all tests. The results of this computation are summarized in Table 4.2, where we highlight the most effective results for each metric in bold.

4.3.1 Discussions

Our experimental findings clearly illustrate the robustness and efficacy of our proposed model, ENCODE. It consistently ranks highly across various evaluation metrics. Delving into specific datasets:

- In the Healthcare dataset, N-Beats emerges as a strong performer, yet ENCODE results are comparable and competitive.
- For the PeMS dataset, ENCODE predominantly outperforms other models, particularly in the metrics of *Mean* and *Mean+Std*. For example, in 12-step forecasts,

MODEL	**RMSE**			**MAE**		
	Mean	**Std**	**Mean+Std**	**Mean**	**Std**	**Mean+Std**
LASSO	0.3188	0.653	0.3712	0.1686	0.6940	0.2661
Ridge	0.2692	0.6304	0.3495	0.1496	0.6996	0.2715
Elastic Net	0.2834	0.5949	0.3576	0.1667	0.6588	0.2770
XGB	0.1937	0.3998	0.2347	0.0827	0.3763	0.1232
Random Forest	0.1738	0.3845	0.2109	**0.0640**	**0.3692**	**0.1002**
SVR	0.2670	0.4239	0.2748	0.4004	0.5069	0.4287
ARIMA	0.5019	0.3084	0.4895	0.3969	0.4544	0.4705
Mean	0.2058	0.4490	0.2574	0.1276	0.4888	0.2020
Median	0.2169	0.5184	0.2773	0.0785	0.5433	0.1712
PSO	0.2513	0.5487	0.2879	0.2118	0.6250	0.2950
Genetic	0.2507	0.4197	0.2889	0.1958	0.5099	0.2709
Random Walk	0.6924	0.4346	0.6857	0.5145	0.5530	0.5814
N-Beats	0.4908	0.6256	0.4982	0.4343	0.4972	0.4719
Prophet	0.3210	0.4380	0.3391	0.3581	0.4129	0.3876
BHT-ARIMA	0.7153	0.4806	0.7069	0.5416	0.5759	0.6188
MTGNN	0.1761	0.427	0.2177	0.0896	0.4095	0.1527
ENCODE	**0.1160**	**0.0959**	**0.0589**	0.2417	0.1515	0.1981

Table 4.2: Comparison between various predictors. The values are derived by normalizing the errors for each dataset and forecast horizon, then averaging these normalized values. This normalization scales the error measure for each dataset and time horizon between 0 and 1, allowing for the comparison of losses across different datasets. Additionally, the same exogenous variables used by ENCODE are also applied to all baseline models that can incorporate exogenous variables.

its *Std* metric is notably superior, and in 24-step predictions, its *Mean+Std* is nearly unmatched.

- With the ToIT dataset, ENCODE demonstrates exceptionally low error rates in both *Mean* and *Mean+Std*.

However, ENCODE performance falters in the Electricity dataset, particularly in the *Mean* and *Mean+Std* metrics, albeit with a consistently low *Std*. This underperformance, which appears as a potential vulnerability, is likely linked to the encoding phase: the complexity of encoding input variables hinders the predictors' ability to utilize the original data effectively. A comparison of 1-step ahead RMSE metrics using encoded and non-encoded inputs (Table 4.3) highlights the significant impact of encoding on predictive accuracy in this context.

The SST dataset results are particularly striking when examining our model performance. Notably, the errors measured by *RMSE* are substantially lower than those

MODEL	**No Enc**			**Enc**		
	Mean	**Std**	**Mean+Std**	**Mean**	**Std**	**Mean+Std**
LASSO	0.075	0.040	0.114	0.139	0.035	0.174
Ridge	0.074	0.039	0.113	0.128	0.039	0.167
Elastic Net	0.074	0.039	0.113	0.128	0.034	0.162
XGB	0.075	0.035	0.110	0.144	0.032	0.175
RFR	0.076	0.035	0.111	0.134	0.034	0.168
SVR	0.086	0.039	0.124	0.149	0.040	0.189

Table 4.3: Comparison of 1-step ahead RMSE for basic regressors using encoded and non-encoded inputs.

of competing models, with MAE scores also being noteworthy. This superior performance can largely be attributed to the model handling of data outliers. These outliers significantly affect the accuracy of auto-regressive models, leading to substantial deviations from actual values during peak periods and their subsequent points. However, our model demonstrates a reduced influence from these outliers, primarily due to the incorporation of the Autoencoder (AE) stage, which effectively mitigates the effects of these peaks.

This dataset results highlight the critical role of the AE stage in increasing resilience to outliers, particularly when it operates independently of the predictive stages. The theoretical underpinning of this observation is linked to the function of the Autoencoder, $\phi_1 : \mathbf{i} \in \mathbb{R}^{MK} \to \tilde{\mathbf{i}} \in \mathbb{R}^{V}$. Considering a scenario with a single encoding layer, the function can be represented as $\phi_1(\mathbf{i}) = tanh(W\mathbf{i})$, where $W \in \mathbb{R}^{V \times MK}$. Given that encoder training depends on back-propagation convergence, it is plausible that the matrix norm remains at or below 1, i.e., $|W| \leq 1$. The nature of the $tanh$ function, especially its derivative $tanh'(z) = \frac{4}{(e^{z}+e^{-z})^2} \leq 1$, imparts a standardizing effect, thereby reducing the influence of outliers.

In conclusion, the experimental results underscore an important insight: although basic statistical models may sometimes surpass ENCODE in performance, ENCODE excels in handling time-series characterized by distinct patterns such as strong seasonalities or irregular anomalies. It is also important to note the significant computational resources required by ENCODE during its training phase, but its testing phase is comparatively efficient, especially when evaluated against the detailed and extended predictive spans of the analyzed time-series.

4.3.2 Statistical Comparison

To rigorously evaluate the performance of the ENCODE framework, a detailed statistical analysis was performed. This analysis aimed to determine whether the average error rates of ENCODE were significantly lower than those of its competitors. For this purpose, the corrected paired Student's t-test, as outlined in [129], was utilized. This test is tailored to compare error vectors across different folds, particularly focusing on the **RMSE** values discussed earlier.

The fundamental hypothesis of this analysis, the null hypothesis, posits that there is no significant reduction in the mean error vector of ENCODE ($\mathbf{RMSE}^{ENCODE}$) when compared to that of the competitors ($\mathbf{RMSE}^{Comp}$). Conversely, the alternative hypothesis suggests that $\mathbf{RMSE}^{ENCODE}$ is significantly lower than $\mathbf{RMSE}^{Comp}$. Notably, the standard deviation used in the test statistic is adjusted to reflect the correlation among the errors.

The p-values derived from this statistical test, considering RMSE as the performance metric across all algorithms, datasets, and forecast horizons, are detailed in Table 4.4. For ease of interpretation and visual clarity, p-values less than 0.05 are underlined, and those below 0.01 are presented in bold.

As indicated in the table, ENCODE shows significant effectiveness on the SST, PeMS, and ToIT datasets, aligning with earlier observations. However, an interesting observation emerges in the SST dataset: despite ENCODE exhibiting notably better mean values and standard deviations compared to other predictors, the statistical test does not consistently reject the null hypothesis. This is particularly evident when ENCODE is compared to certain algorithms. This result is likely influenced by the prevalence of outliers in the SST dataset, which increase the variance of the predictors and thereby affect the outcomes of the statistical test.

Dataset	**Electricity**			**SST**			**PeMS**			**Healthcare**		**ToIT**
Steps	**1**	**3**	**7**	**1**	**3**	**7**	**12**	**24**	**36**	**7**	**14**	**24**
LASSO	0.9999	1.0	0.9948	**0.0013**	**0.0012**	**0.0012**	**0.0**	**0.0009**	**0.0001**	0.8447	0.9891	**0.0**
Ridge	0.9998	0.9998	0.9835	**0.0012**	**0.0011**	**0.0012**	**0.0**	**0.0014**	**0.0001**	0.2432	0.0629	0.0152
Elastic Net	0.9999	1.0	0.9925	**0.0012**	**0.0011**	**0.0012**	**0.0**	**0.0013**	**0.0001**	0.2555	0.065	**0.0002**
XGB	1.0	0.9988	0.9736	**0.0012**	**0.001**	**0.0011**	**0.0028**	0.0747	**0.0001**	0.0269	0.021	0.0164
Random Forest	1.0	0.9987	0.9432	**0.0012**	**0.001**	**0.0011**	**0.0004**	0.4471	**0.0007**	0.6624	0.9169	0.0741
SVR	0.9722	0.9753	0.6771	**0.0008**	**0.0007**	**0.0008**	**0.0018**	0.0135	**0.0002**	0.3607	0.0374	0.0406
ARIMA	0.22	0.9891	0.9662	**0.0016**	**0.0014**	**0.0015**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0004**
Mean	1.0	1.0	0.9968	0.0103	**0.0088**	**0.0098**	**0.0**	**0.0049**	**0.0**	0.6757	0.4361	**0.0031**
Median	1.0	1.0	0.9988	**0.0015**	**0.0014**	**0.0016**	**0.0001**	0.0243	**0.0001**	0.8066	0.8351	0.0147
PSO	0.0544	0.1618	0.0176	**0.0018**	**0.0008**	**0.0**	**0.0**	**0.0**	**0.0**	0.0137	0.2246	**0.0002**
Genetic	0.2972	0.0376	0.0569	**0.0024**	**0.0017**	**0.0**	**0.0**	**0.0014**	**0.0**	**0.0056**	**0.0003**	**0.0007**
Random Walk	0.1098	**0.0**	0.9992	0.0512	0.0432	0.0367	**0.0**	0.0938	**0.0**	**0.0003**	**0.0**	**0.0**
N-Beats	0.1404	0.0939	0.074	0.0267	0.0165	**0.0031**	**0.0011**	0.2495	**0.0004**	0.9924	0.9984	**0.0**
Prophet	0.9228	0.9958	0.9905	0.0237	0.0206	0.0209	**0.0027**	0.0313	**0.0021**	**0.002**	**0.0016**	0.2012
BHT-ARIMA	**0.0001**	**0.0001**	0.0481	0.0509	0.0424	0.0393	**0.0**	**0.0**	NaN	**0.0002**	**0.0001**	**0.0**
MTGNN	1.0	0.9999	0.9994	0.0363	0.0336	0.0308	0.2829	0.4755	**0.0088**	0.4962	**0.0068**	**0.0011**

Table 4.4: P-values from the statistical test using RMSE as the loss function. Values lower than 0.05 and 0.01 are underlined or in bold, respectively.

4.4 Conclusions

This chapter's exploration underpins the development of ENCODE, an innovative ensemble methodology tailored for multivariate time-series forecasting. ENCODE emerges from the idea of skillfully blending diverse predictions using a Neural Network, addressing practical challenges to create an advanced framework. It adeptly combines Machine Learning (ML) and Deep Learning (DL) strategies, effectively discerning complex patterns in multivariate time-series, thereby enhancing the accuracy and reliability of its forecasts. This efficacy is supported by extensive experimental evaluations.

A notable characteristic of ENCODE, which effectively bridges the gap between hybrid and ensemble forecasting techniques, is its ability to regulate data. This feature makes it particularly robust against outliers in time-series data. The initial encoding stage of ENCODE plays a crucial role in reducing the influence of anomalies. Furthermore, in datasets rich with data points, ENCODE demonstrates superior performance in forecast accuracy compared to many existing methods. This indicates that the Hybrid Neural Network, when properly trained, can efficiently harmonize the data-derived patterns with the projections from regression models.

The framework has also evolved to provide both long-term and short-term forecasts, making it suitable for practical applications that demand varying forecasting horizons. This versatility of ENCODE underscores its applicability in diverse real-world settings where different forecasting needs must be met, including Smart City scenarios.

Chapter 5

SMart Analytics in Roads and Transportations (SM.A.R.T)

The preceding chapters have meticulously established a robust theoretical foundation for Neural Networks, delved into the intricate depths of Deep Learning methodologies, and explored the nuanced domain of forecasting with its associated complexities and state-of-the-art approaches. In the realm of object detection, a significant emphasis was placed on the evolution and advancements in Deep Learning models, specifically highlighting the YOLO series as a paradigm of innovation. Additionally, the dissertation introduced the ENCODE model, an advanced and sophisticated framework designed to navigate and address the multifaceted challenges inherent in forecasting.

Building upon these foundational elements, this chapter introduces the final integrative framework: SMart Analytics in Roads and Transportation (SM.A.R.T). This framework is conceived to synergistically combine the predictive analytics capabilities of the ENCODE model with the avant-garde object detection methodologies epitomized by the YOLO series. However, before we embark on an in-depth exploration of SM.A.R.T, it is critical to discuss several key components: an elucidation of YOLOv8 as the state-of-the-art in object detection, and an overview of contemporary models in object tracking.

SM.A.R.T represents an ambitious integration at the nexus of predictive analytics and visual cognition. This synthesis aims to provide a comprehensive and holistic approach to addressing the dynamic and complex challenges in the domain of roads and transportation. By harmonizing predictive analytics with real-time visual data analysis, SM.A.R.T aspires to not only foresee but also to perceptively respond to the evolving scenarios in transportation systems. This chapter aims to articulate the architecture, operational mechanisms, and potential applications of SM.A.R.T, thereby establishing a new analytics tool combining forecasting and object detection technologies in the context of intelligent transportation systems.

5.1 YOLOv8

YOLOv8 [107] stands as a remarkable advancement in the YOLO (You Only Look Once) series, a lineage of models renowned for transforming the landscape of real-time object detection. The authors are the same as YOLOv5, and similarly to the model, it was released as a software without a preliminary academic publication. Still, it provides significant enhancements to its predecessors that elevate its performance, accuracy, and efficiency, with also versatility in working not only in object detection but also classification, segmentation, pose estimation.

5.1.1 Architecture

YOLOv8 architecture is shown in Fig. 5.1. The model's architecture can be divided into two primary components: the backbone and the head. Such macro-block structure is derived from the backbone-bottleneck-head architecture introduced by YOLOv4 [101], with the bottleneck part being incorporated as a sub-block of the convolutional block C2f, and each block plays a crucial role in the overall detection process.

- **Backbone:** The backbone is responsible for feature extraction. It processes the

[1]URL: `https://github.com/ultralytics/ultralytics/issues/189#issue-1527158137` .

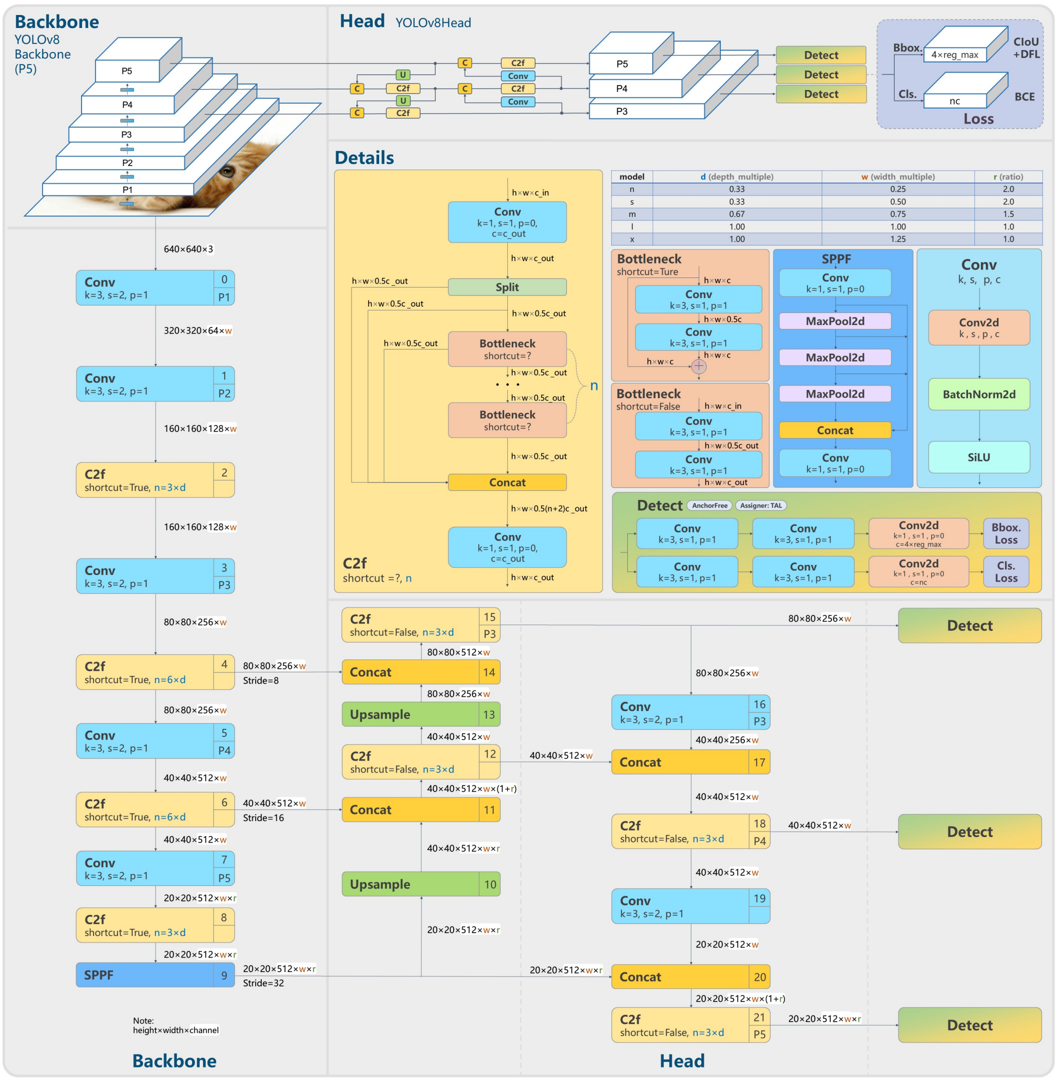


Figure 5.1: The YOLOv8 architecture. CC Range King from GitHub[1].

input image and extracts a rich set of features that are essential for detecting objects. This backbone network typically uses a series of convolutional layers. Each layer is designed to capture different aspects of the image, from basic edges and textures in the initial layers to more complex shapes and patterns in the deeper layers. For YOLOv8, the structuring is based on Feature Pyramid Networks (FPNs) [90] and Spatial Pyramid Pooling (SPP) [86], enhancing feature extraction. Furthermore, the first convolutional layer's kernel size might be adjusted to better capture fine details in the image.

- **Head:** The head is the final part of the architecture, where the actual object detection takes place. This segment of the network typically involves layers that perform two key tasks: classifying the objects present in the image and predicting their bounding boxes. Further customizations allow to solve different tasks, as done by YOLOv8 which deploys a Decoupled version of the head, providing also more precise localization and classification, improving the overall accuracy of the model.

Furthermore, one of the key features of YOLOv8 is its scalability. Just like YOLOv5, It comes in various sizes (small, medium, large, and extra-large), each tailored to balance speed and accuracy based on the computational resources and requirements of the application. Each size depends on two parameters: $w \in \mathbb{R}_+$ and $d \in \mathbb{R}_+$, with w, named the width factor, changing the number of output channels in each convolution block, while with d, named the depth factor, changing the number of consecutive bottleneck submodules in each convolution block. This scalability ensures that YOLOv5 can be deployed in diverse environments, from edge devices with limited computational capacity to powerful servers.

5.1.2 Learning procedure

Important improvements come from the training of YOLOv8, such as the loss function. Indeed, in place of using the mean squared error loss for the classification, it uses the Binary Cross Entropy loss, while for the bounding box the loss is replaced with the Complete Intersection Over Union (CIoU) loss [130, 131] and for the box confidence loss the Distribution Focal Loss (DFL) [132], which improve the performances both in terms of training epochs and in terms of the metrics.

YOLOv8 demonstrates exceptional performance, particularly in environments where real-time detection is crucial. Its ability to process images rapidly without compromising on accuracy makes it an ideal candidate for applications in autonomous driving, surveillance, and real-time monitoring systems [133].

Tamang *et al* used a fine-tuned version of YOLOv8 model for the COVID-19 mask recognition, with the capability of recognizing individuals either wearing a mask correctly, incorrectly, or not wearing at all [134].

Aboah *et al* uses a data processing technique named few-shot data learning to generate a robust helmet detection model, using YOLOv8 as the detection model, giving the best results when compared with the predecessors [135].

5.2 Object tracking

The integration of object detection models within video stream pipelines introduces a unique set of challenges, particularly in the context of identifying and following objects across successive frames. While object detection algorithms excel at locating and classifying objects within a single frame, their capabilities are limited to that specific snapshot in time. This limitation becomes apparent when attempting to understand the continuity and movement of objects across a series of frames in a video. To bridge this gap, the introduction of object tracking mechanisms plays a pivotal role.

Object tracking, in essence, extends the functionality of object detection by adding a temporal dimension to the analysis. It involves not only recognizing objects frame by frame but also maintaining a consistent identification of each object over time. This process involves assigning a unique identifier to each detected object and then tracking these identifiers across successive frames. The primary goal of object tracking is to establish a coherent and continuous narrative of each object's movement and interactions within the video stream.

To achieve effective tracking, algorithms must overcome several challenges, such as varying object speeds, changes in appearance, occlusions, and varying environmental conditions. Advanced tracking methods employ sophisticated techniques to maintain object consistency. These may include predictive modeling, where the algorithm anticipates an object's future position based on its past trajectory, and adaptive learning, where the model adjusts its understanding of an object's appearance as it changes over time [136].

One common approach in object tracking is the use of Kalman filters or particle filters, which predict an object's future state based on its current state and previous states. Another approach is feature-based tracking, where specific features of an object (such as color, texture, or shape) are tracked over time. Deep learning-based trackers have also gained prominence, leveraging the power of neural networks to learn complex patterns of movement and appearance changes, thus enhancing the accuracy and robustness of the tracking process.

A notable advancement in the realm of object tracking is the development of ByteTrack, proposed by Zhang *et al* [137]. This innovative method employs a strategy where each detection box with a high confidence score is assigned a unique tracking ID. Simultaneously, it retains detection boxes with lower scores as a contingency for situations where tracking may be momentarily lost. This approach is particularly effective in scenarios involving occlusions or when objects are only partially visible in the frame. To maintain the continuity of tracking, ByteTrack utilizes a Kalman Filter, which aids in forecasting the subsequent positions of the objects in the video frames. This as-

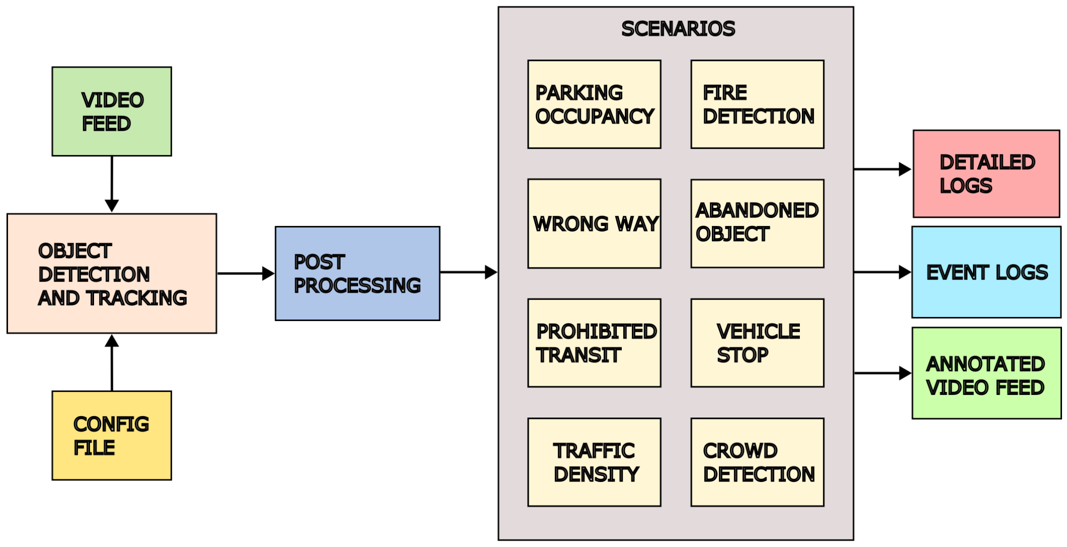


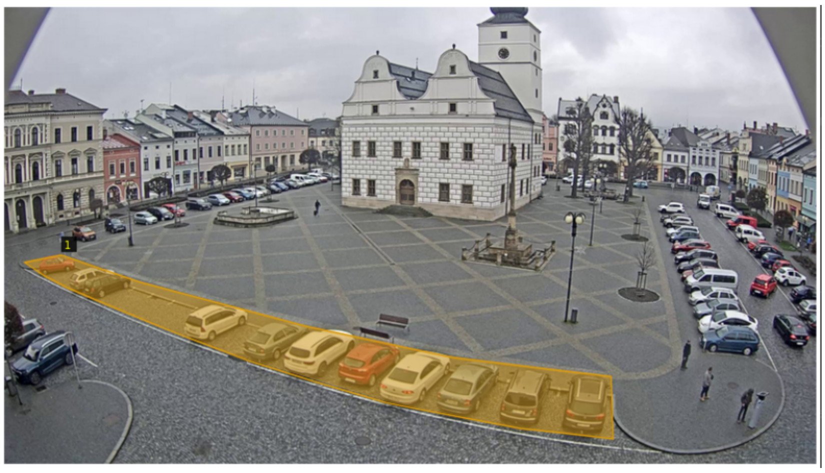
Figure 5.2: The object detection block of SM.A.R.T.

pect of ByteTrack contributes significantly to its robustness and accuracy, making it an adaptable solution for diverse tracking applications. Recent comparative studies, [138, 139].

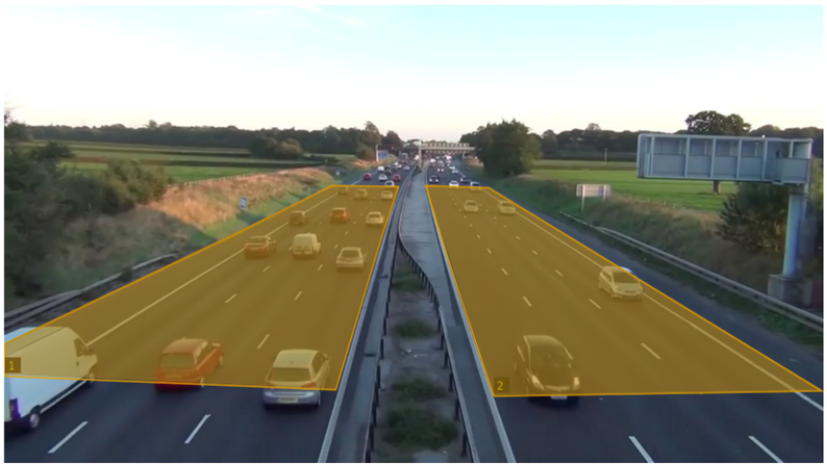
In conclusion, object tracking is a vital component in the realm of video analytics, enabling the extension of object detection capabilities to dynamic, real-world scenarios. By providing a continuous understanding of objects' movements and interactions, object tracking algorithms add significant value to applications such as surveillance, traffic monitoring, and sports analytics, among others. The ongoing advancements in this field continue to push the boundaries of what is possible in understanding and interpreting motion within video data. For the SM.A.R.T. framework, we use BYTE-Track as the tracking choice due to it being the SoTA, as previously indicated.

5.3 The framework

The main objective of this framework is to use the forecasting component for predictive analytics for the smart mobility, for instance parking occupancy prediction, and to use the object detection and tracking component to find relevant events.



(a) Parking occupancy



(b) Traffic density analysis

Figure 5.3: Examples of video frames with highlighted the corresponding Region of Interests (ROIs) on which the analysis is focused on.

As mentioned before, the proposed model is divided in two components: a forecasting part block with the ENCODE model, and an object detection block with YOLOv8 and BYTETrack. Figure 5.2 outlines the proposed framework on the side of the object detection. The object detection block consists of three main stages: an Object Detection and Tracking stage, Post-processing stage, and the Scenarios stage.

- The Object Detection and Tracking Stage gathers each video frame from the stream, processes such video frame through YOLOv8 to extract the objects, then passed through BYTETrack for the identification of each object.

- The Post-processing stage takes the tracked objects and assigns such objects to relevant Regions of Interests (ROIs), where the analytics is intended to focus on. For example, the Parking occupancy module requires as ROIs the parking slots, either as a single slot or multiple slots. Another example is for the Traffic Density module, which requires a ROI on a street. These two cases are shown in Fig. 5.3a and 5.3b, respectively. In the former case the ROI denotes a group of 9 parking slots near the plaza, while for the latter case there are two ROIs, each working on a side of the highway. Furthermore, the module computes an estimation of speed and direction of each object in a rolling time window with size fixed but configurable.

- The Scenarios stage performs various tasks based on the collected objects on specific RoIs. The available tasks are: parking occupancy, wrong-way driving detection, prohibited transit, traffic density, fire detection, abandoned object detection, vehicle stop detection and crowd detection.

The output of the object detection-based analysis is composed by three different data: the Detailed Logs data which gathers all the detected objects, the Event Logs data which collects all the important data, and the Annotated Video Feed streaming data, which is optional, for showing the detected objects on the screen.

5.4 An example of application: Smart City Crowd Management

The phenomenon of urbanization has led to significant population growth in cities, often resulting in high-density areas. This concentration of people raises various challenges, including safety concerns, crowd congestion, and the need for efficient urban planning. Consequently, effective crowd management has become a vital consideration for city administrators, urban planners, and public safety officials.

With the widespread adoption of video surveillance, monitoring crowd dynamics has become more feasible. However, manually extracting and managing such data is resource-intensive and practically unsustainable. Therefore, the development of automated systems to process and analyze this data is not just beneficial but essential.

The object detection component of the SM.A.R.T. framework addresses these challenges by utilizing advanced object detection and tracking models to monitor pedestrian movements. It captures data through video feeds, which are then processed to provide actionable insights for crowd analysis.

5.4.1 Methodology

The data retrieval process begins from the Detailed Log data of the SM.A.R.T. framework. After choosing the RoI where the crowd has been intended to be collected, for each time step we extract from the detected objects with class 'person' the lower center position of the bounding box in pixel coordinates, and, if available, a tracker ID. Objects lacking a tracker ID are excluded to ensure data reliability.

To analyze crowd behavior accurately, it is crucial to convert pixel coordinates into spatial coordinates. This is where the homographic transformation plays a key role, allowing us to map the camera view onto a real-world plane. Once we have spatial coordinates, various analytical methods are applied, ranging from trajectory analysis (both moving and stationary) to more advanced predictive analytics. These analyses can include statistical methods like Markov chains, which help in predicting future movement patterns within the area of interest, contributing significantly to real-time decision making and proactive crowd management strategies.

Consequently, we are able to generate all the trajectories of the persons located on the plaza in a time interval and stored as a dataset, making it essential for the following analysis.

Homographic Transformation

The Homographic Transformation is a mathematical technique used in image processing to establish a mapping between two planar surfaces. This transformation is particularly useful when we need to relate points in one image plane to another, assuming that these points lie in a region with negligible curvature relative to the Earth's surface.

Consider two planar surfaces S and S', each with their respective coordinate systems $O, \boldsymbol{e}_x, \boldsymbol{e}_y$ and $O', \boldsymbol{e}'_x, \boldsymbol{e}'_y$. The origins of these coordinate systems are denoted by O and O', while $\boldsymbol{e}_x$, $\boldsymbol{e}_y$, $\boldsymbol{e}'_x$, and $\boldsymbol{e}'_y$ represent the orthonormal vectors on planes S and S', respectively. Suppose we have selected points $P_1, P_2, \ldots, P_K$ on S and corresponding points $P'_1, P'_2, \ldots, P'_K$ on S' such that there is a one-to-one mapping between P_j and P'_j for $j = 1, \ldots, K$, where $K \geq 4$.

A Homography is a function $f_H : S \to S'$ defined by:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = H \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}. \tag{5.1}$$

where $(x, y) \in S$, $(x', y') \in S'$, and $H = [h_{ij}]$ is a non-singular matrix representing the homographic transformation. The points $(x, y, 1)$ and $(x', y', 1)$ are the corresponding homogeneous coordinates in S and S', respectively. The transformation aims to minimize a metric related to the distance between P'_j and $f_H(P_j)$ for $j = 1, \ldots, K$.

The matrix H possesses eight degrees of freedom, which theoretically allows it to be determined uniquely from four pairs of corresponding points in the two image planes. However, in practical applications such as crowd detection through video cameras, it is common to use more than four point pairs. This approach increases the robustness of the estimation and compensates for measurement noise and errors. The most prevalent method to estimate the coefficients of the homography matrix is the RANSAC algorithm [140], known for its robustness against outliers.

In the context of crowd detection, selecting K points from the camera view and the corresponding K points from a satellite view enables the computation of the homography transformation. This transformation is then used to convert pixel coordinates from the camera view into spatial coordinates, providing a vital link between image data and physical locations.

Such spatial mapping is indispensable for accurate crowd analysis, allowing for the precise localization and tracking of individuals or groups in crowded urban environments. By transforming the perspective of the camera to a top-down view, it becomes feasible to apply algorithms for crowd density estimation, movement patterns analysis, and anomaly detection in real-time. This capability is crucial for smart city applications like traffic management, public safety monitoring, and urban planning, where understanding and responding to crowd dynamics can significantly enhance efficiency and safety.

Markov chain

Markov chains offer a robust mathematical model for predicting the future location of pedestrians based on their current and previous locations. K order Markov chains are particularly insightful as they incorporate both the current state and a series of previous states in the prediction process. Formally, in a sequence of random variables $X_0, X_1, ..., X_n$, where each X_i represents the state at time i, the probability of transitioning to the next state X_{n+1} is conditional on the current state X_n and the previous k states $X_{n-1}, X_{n-2}, ..., X_{n-k}$.

To estimate these transition probabilities, the maximum likelihood estimation method is employed. This involves counting the occurrences of transitions from state j to state i, given the current and k previous states, and then normalizing these counts. The probability is thus given by:

$$P(i|j, X_{n-1}, ..., X_{n-k}) = \frac{count(i, j, X_{n-1}, ..., X_{n-k})}{\sum_i count(i, j, X_{n-1}, ..., X_{n-k})} \tag{5.2}$$

In Smart Cities, these K order Markov chains become instrumental for modeling pedestrian movements. By leveraging historical data, they predict future pedestrian locations, aiding in urban planning and public safety. However, it's important to note the limitations of this approach. The accuracy of these predictions depends on the quality and volume of historical data, and the assumption of dependency on only a limited history might oversimplify complex behaviors. Therefore, while Markov chains are valuable tools, they should be used with an understanding of their inherent constraints and in combination with other predictive methods.

5.4.2 Experimental setup and results

In the context of crowd analysis within our experimental setup, the study focused on several key aspects to understand pedestrian dynamics in the designated Region of Interest (RoI). These aspects included:

- **Crowd Density Analysis**: This involved assessing the spatial distribution of pedestrians across different areas within the RoI. The goal was to identify patterns in how people congregate and move, providing insights into high-density zones and potential areas of congestion.
- **Trajectory Analysis**: An examination of the movement paths taken by individuals within the RoI. By tracking the trajectories, the study aimed to uncover common routes, movement patterns, and any recurrent pedestrian behaviors within the observed area.
- **Trajectory Prediction Analysis**: This aspect focused on predicting future pedestrian movements based on historical trajectory data. The objective was to gauge the predictability of pedestrian paths and understand how past movements

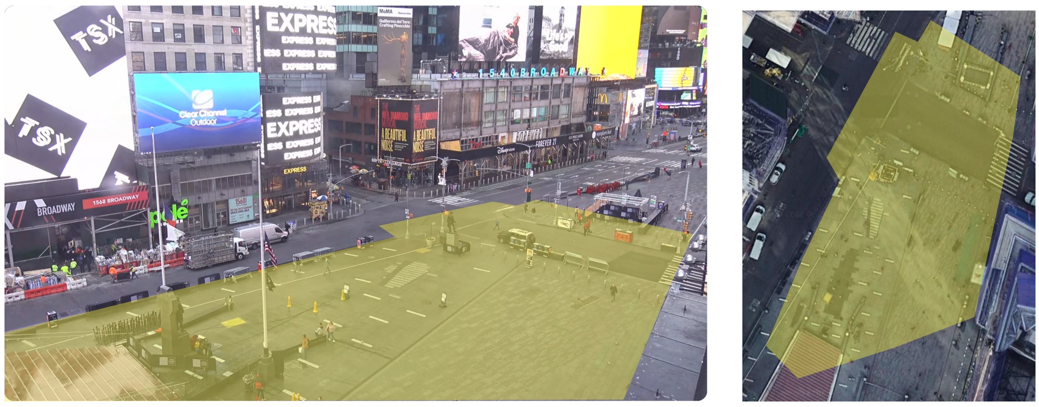

(a) Live video (b) Satellite view

Figure 5.4: Times Square, New York, used for this study. In yellow the selected RoI.

could inform future pedestrian flow within the RoI. Here we apply Markov chain modelling for the task.

Data preparation

For the purpose of this work, a live stream from Times Square [2] was utilized as the video source for analysis. This footage, captured from an elevated vantage point, provided an expansive view overlooking the northern area of Times Square, specifically around the intersection of W 47th Street and Broadway (Duffy Square). For illustrative clarity, an exemplary frame from this live stream is depicted in Figure 5.4a, accompanied by a satellite image of the same location. The satellite image has been rotated 180 degrees to align with the perspective of the video, as shown in Figure 5.4b. The period of data collection spanned from April 16th to April 23rd, 2023. This timeframe was specifically chosen to facilitate an in-depth analysis of the data, allowing for segmentation by both hourly intervals and days of the week, thereby enriching the comprehensiveness of the study.

The selection of Times Square for our study was motivated by its status as a nexus of high pedestrian activity, with the area serving as a convergence point for diverse

[2]EarthCam Live: Times Square in 4K: https://youtu.be/1-iS7LArMPA

activities. This location offers an eclectic mix of elements that are emblematic of both tourism and rapid urban movement, making it a prime site for data gathering and analysis. Key features of this bustling area include:

- The Red Stairs, a prominent and often crowded landmark situated in the lower left quadrant of the live video feed. This spot is a popular gathering place and offers a unique vantage point for observing the area.
- The Duffy Statue, located in close proximity to the Red Stairs, adds historical significance and serves as another point of interest within the square.
- The arrangement of Tables near the adjacent building, discernible in the lower right portion of the video frame. This area is often frequented by locals and tourists alike, providing a space for relaxation amidst the urban hustle.
- The Broadway crossing, visible in the middle section of the plaza, is a vital thoroughfare that exemplifies the dynamic and fast-paced nature of the area.

The varied nature of these spots, ranging from leisure areas to busy crossings, offers a comprehensive perspective on urban dynamics and pedestrian behavior.

Crowd density

The study of crowd density in Times Square, as depicted in Figures 5.5 and 5.6, offers a detailed visualization of both hourly and daily crowd distributions. These heatmaps reveal that the lower section of the plaza consistently experiences higher foot traffic, with three particular spots emerging as focal points of activity: the Duffy Statue, the Red Stairs, and the area surrounding the tables near the building.
A deeper analysis of crowd patterns based on time reveals intriguing behaviors. During weekends, a pronounced increase in crowd density is noticeable around the Duffy Statue and the tables area, indicating these spots as popular weekend destinations. In contrast, weekdays exhibit a more dispersed crowd pattern across the plaza.

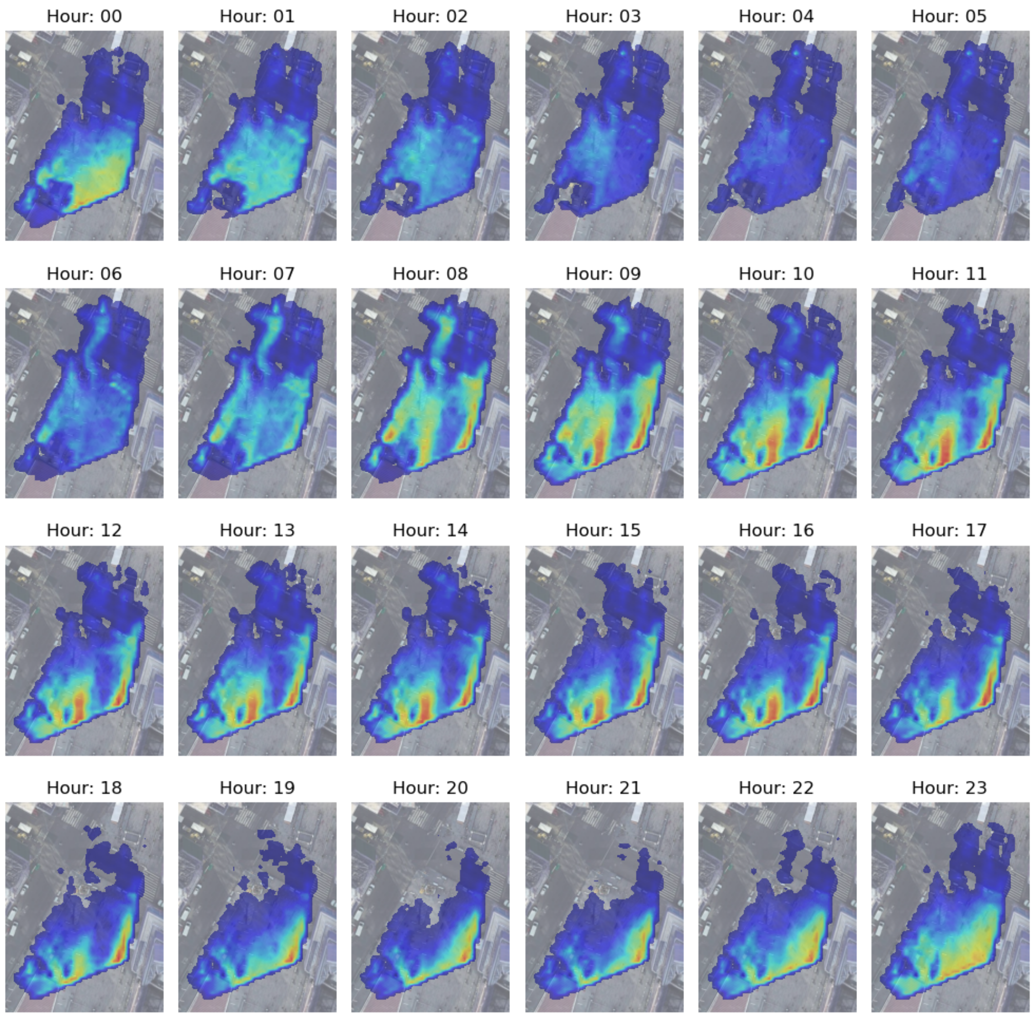


Figure 5.5: Cumulative hourly crowd density estimation from April 16th, 2023, to April 23rd, 2023. The color gradient ranges from blue (lowest density) to red (highest density), with a maximum value of 1.

Hourly trends further reveal that crowd density peaks around midnight, mid-morning, and during the afternoon hours. A notable pattern is observed with respect to the Duffy Statue, which tends to attract more people during daylight, whereas the tables area near the building becomes a more popular gathering spot as night falls. These insights provide a comprehensive understanding of pedestrian flow and congregation patterns in one of the world's most vibrant urban spaces.

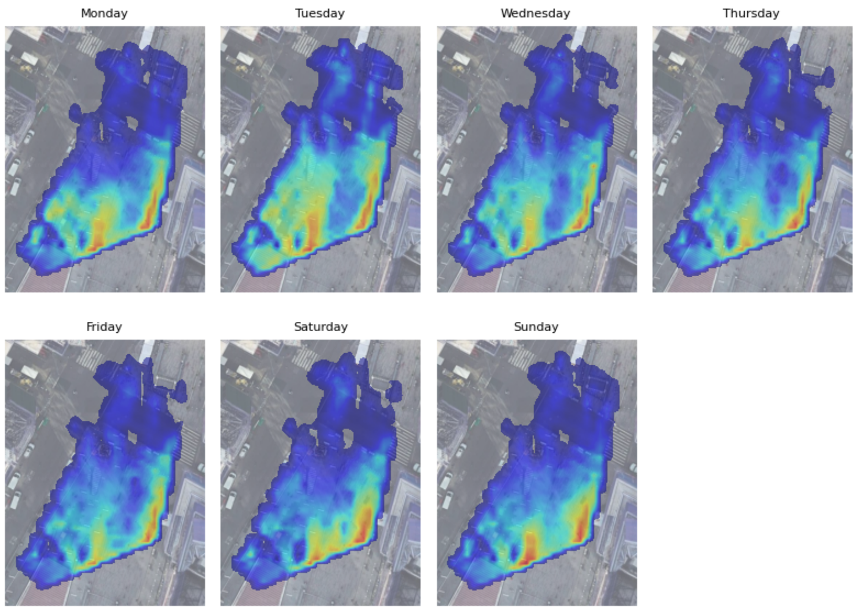


Figure 5.6: Cumulative daily crowd density estimation from April 16th, 2023, to April 23rd, 2023. The color spectrum indicates crowd density from blue (lowest) to red (highest), with a maximum value of 1.

5.4.3 Analysis of the Trajectories

The study of pedestrian trajectories in Times Square provides an insightful perspective into movement patterns. As depicted in Figure 5.7, the analysis of crowd paths uncovers two predominant directions: north-to-south and the reverse, south-to-north. This observation aligns with the typical flow of pedestrian traffic in this bustling urban area.

Given the symmetry in pedestrian behaviors along these principal directions, the analysis focused on trajectories heading from north to south. As shown in Figure 5.8, hourly breakdowns reveal distinct patterns. Morning hours, particularly from 5 A.M. to 9 A.M., are characterized by longer north-to-south trajectories, likely indicative of people commuting to their workplaces. Conversely, evening hours display shorter paths, suggesting a shift in the square's use towards leisure and tourism-related activities.

The exploration of pedestrian velocities in Times Square offers intriguing insights, particularly when correlated with their activities within the plaza. The heatmap of

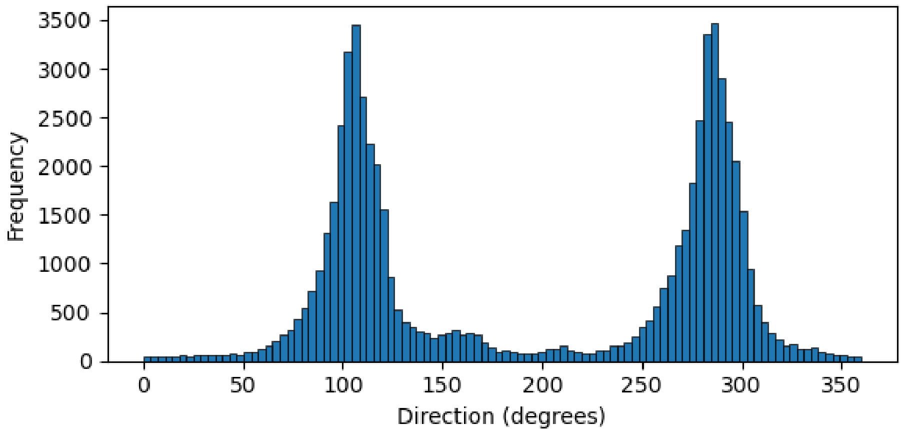


Figure 5.7: Distribution of crowd trajectory directions from April 16th, 2023 to April 23rd, 2023. The histogram shows 90 degrees for south-to-north movement and 270 degrees for north-to-south movement.

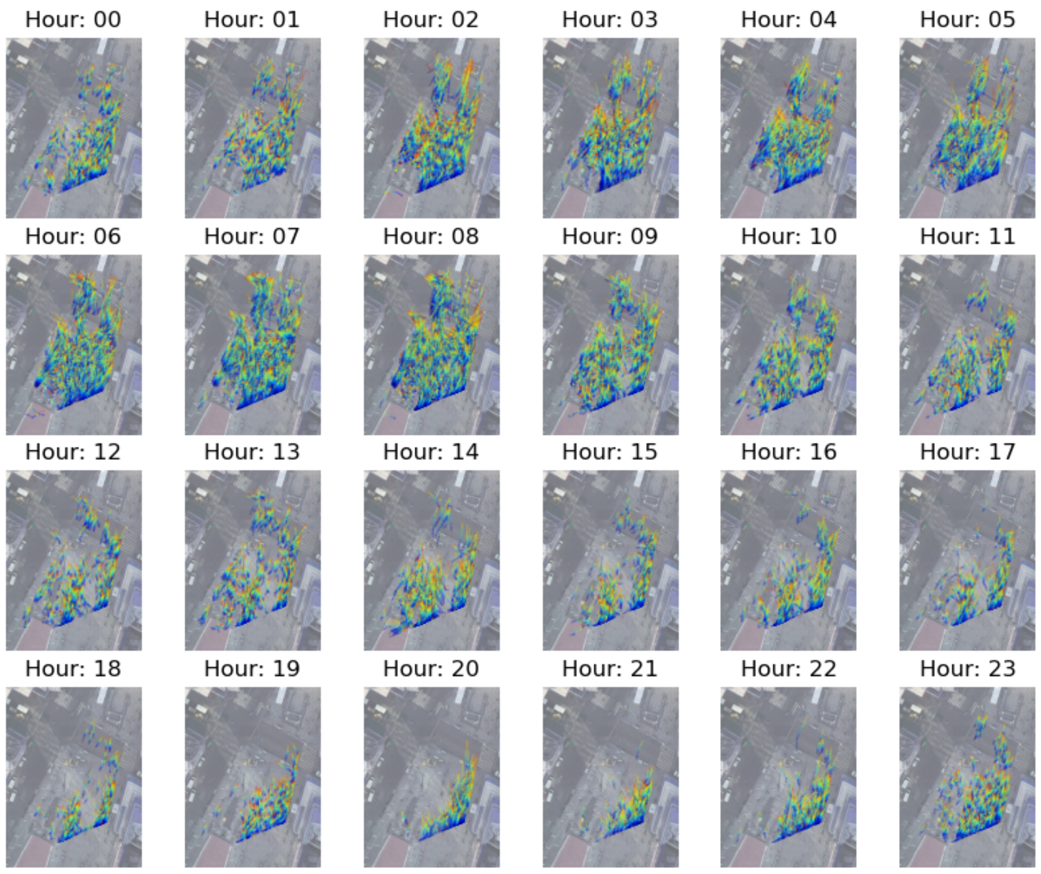


Figure 5.8: Hourly distribution of north-to-south crowd trajectories from April 16th, 2023 to April 23rd, 2023. Each trajectory starts with a blue marker, gradually transitioning to red towards its end.

velocity modules, as seen in Figure 5.9, indicates that higher velocities are more prevalent along longer trajectories, especially near crossings and the outer areas of the plaza. This trend suggests that these areas are frequented by pedestrians who are primarily transiting through the plaza rather than lingering.

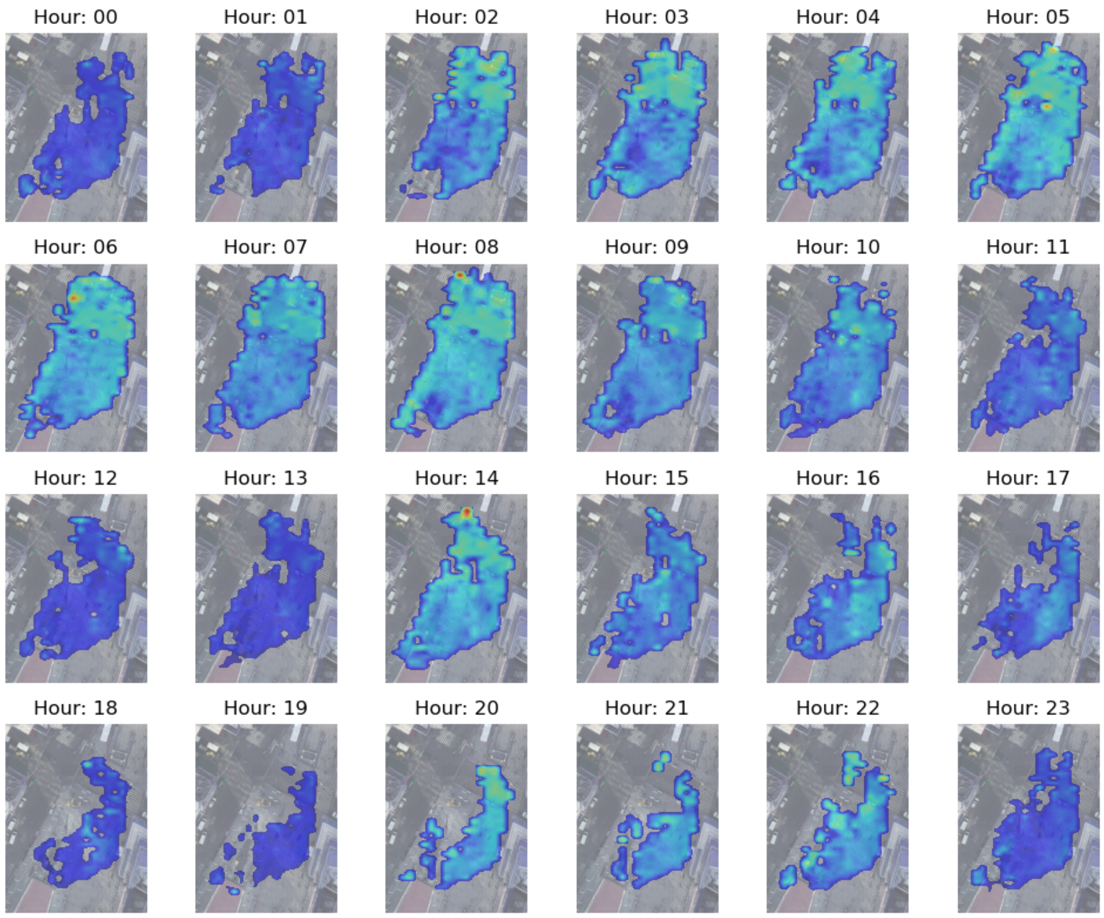


Figure 5.9: Distribution of average trajectory speeds from April 16th, 2023 to April 23rd, 2023, logarithmically transformed for clarity ($log_1 0(speed + 1)$), with speed in km/h. The color gradient from blue to red indicates increasing velocity values.

An interesting pattern emerges from the upper left corner of the velocity heatmaps, revealing high-speed zones in areas lacking designated pedestrian crossings. This observation suggests potentially risky pedestrian behavior, underscoring the utility of velocity data in identifying and addressing safety concerns in public spaces.

Conversely, areas with lower pedestrian velocities correlate with locations where individuals are more likely to spend time, such as near popular attractions or leisure spots within the plaza. This pattern is further substantiated in Figure 5.10, which visualizes the average density of 'staying trajectories' – paths indicative of pedestrians spending extended periods within specific zones of the plaza.

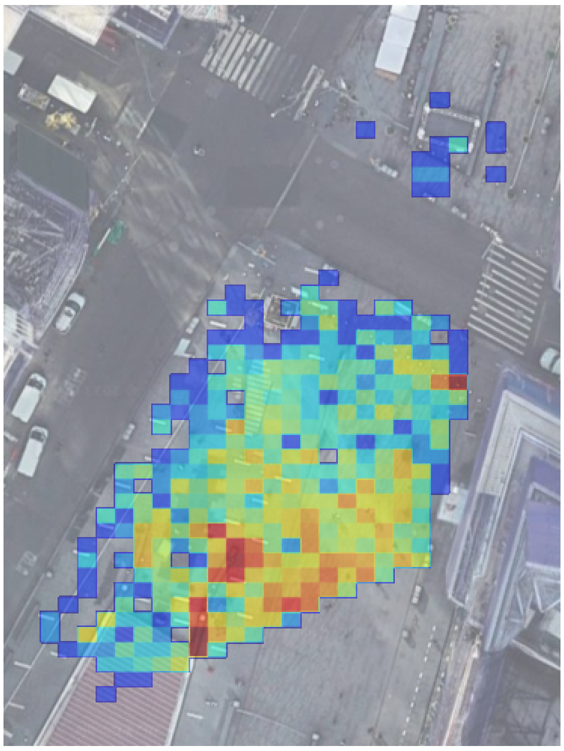


Figure 5.10: Normalized density distribution of staying trajectories from April 16th, 2023 to April 23rd, 2023. The gradient from blue to red denotes lower to higher density areas, with the highest density marked as 1.

This distribution highlights areas of congregation, such as near statues or seating areas, where pedestrians are likely engaged in leisurely activities or visiting attractions. The juxtaposition of these findings – high velocity in transit areas and low velocity in congregational spots – paints a comprehensive picture of pedestrian behavior and activity patterns in Times Square.

Predictive analytics

To model the movement patterns of pedestrians in Times Square, we employed a grid-based approach to transform the area into a matrix of cells, each uniquely identified. This method allowed us to conceptualize pedestrian movements as transitions between these cells, effectively creating a discrete representation of their trajectories. Each step within a trajectory corresponds to a pedestrian's movement from one cell to an adjacent one. A visualization of the grid is provided in Fig. 5.11.

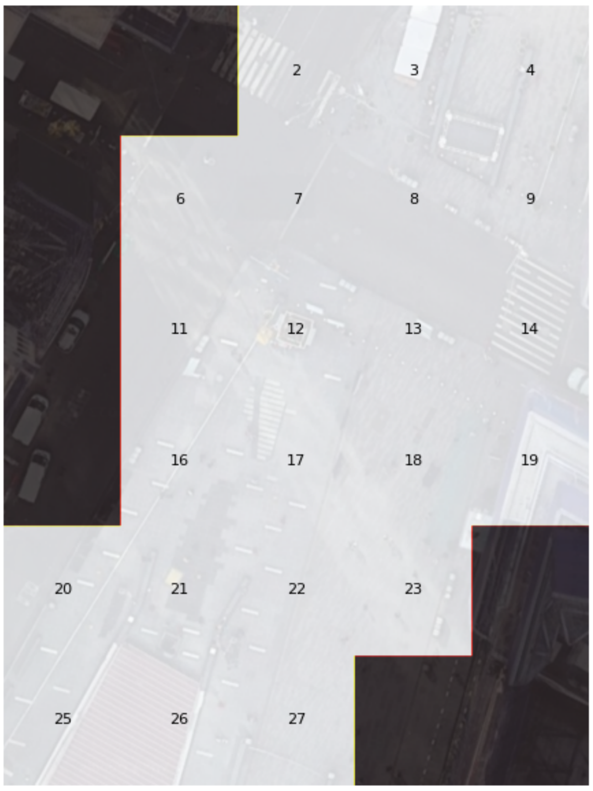


Figure 5.11: Grid partitioning of the RoI based on the people distribution.

We calculated the transition probabilities between adjacent cells by tallying the frequency of pedestrian movements from one cell to another and normalizing these counts. This process led to the construction of a transition matrix, specific to the current time interval, which encapsulates the likelihood of movement from one cell to its neighbors. By leveraging the transition matrix, we projected the future locations of pedestrians for several time intervals ahead. This predictive model is based on multiplying the current pedestrian location probabilities by the transition matrix, enabling us to forecast the most probable paths pedestrians will take in the near future.

Figure 5.12 illustrates this predictive process in action, showcasing the distribution of the last step in a pedestrian's trajectory, alongside both the actual and predicted

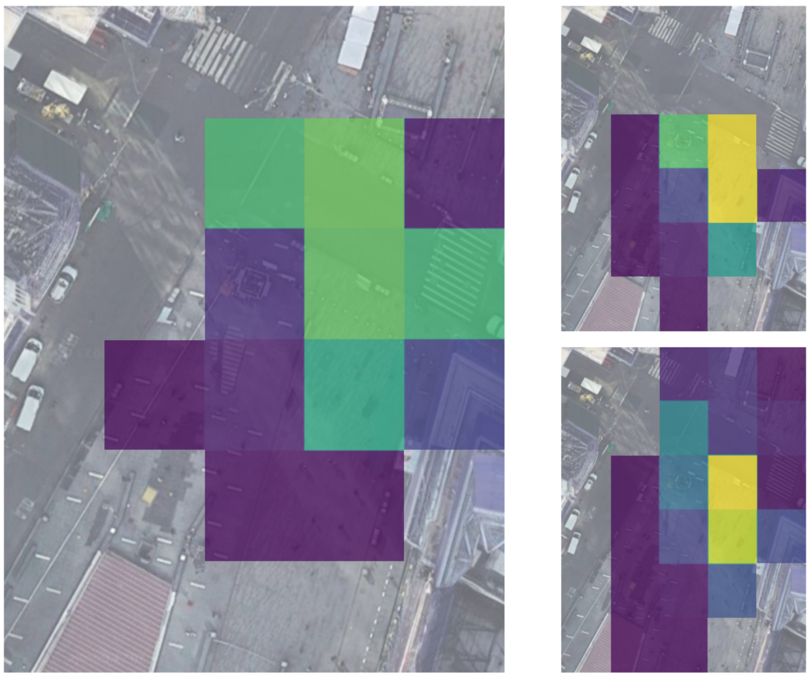


Figure 5.12: Predictive analysis of pedestrian movements from north to south between April 16th and April 23rd, 2023. The figure shows the distribution of the last step on the left, the actual final step in the top right, and the predicted final step distribution in the bottom right, using a viridis color map with a maximum value of 0.3.

final step distributions. This analysis provides a quantitative and visual representation of pedestrian movement patterns, offering valuable insights into the dynamics of pedestrian flow in urban spaces like Times Square.

Utilizing as splitting strategy of the dataset of the trajectories the split into training (80%) and test (20%) segments, we achieved a commendable 70% accuracy in our predictions on the test paths. The illustrative representation in Figure 5.12 effectively demonstrates this predictive accuracy. The left panel of the figure showcases the current density of pedestrians' locations, providing a snapshot of their distribution across the area. For this particular visualization, we focused solely on trajectories moving from south to north, yet the methodology is readily applicable to the entire dataset.

The lower-right panel in the figure reveals our occupancy predictions, which are based on the current positioning snapshot. In comparison, the upper-right panel displays the actual occupancy at a subsequent time. This juxtaposition underscores the reliability of our predictive model in estimating future pedestrian locations in Times Square based on current occupancy data.

The implications of these occupancy predictions are substantial, particularly in the realms of urban planning and crowd management. By accurately forecasting pedestrian movement patterns and densities, city planners and managers can make more informed decisions regarding resource allocation, emergency response, and public safety measures. Furthermore, such insights can be instrumental in enhancing the overall functionality and user experience of public spaces like Times Square.

Chapter 6

Conclusions

As cities continue to expand and evolve, the smart city paradigm has emerged as a critical solution to the challenges of urbanization. This thesis has highlighted the pivotal role of deep learning in transforming urban management, addressing the growing demands for data processing and security in an increasingly digital world. The focus on surveillance cameras has led us through an exploration of artificial neural networks, convolutional neural networks, and the latest advancements in object detection technologies.

The pace of innovation in this field, with significant advancements emerging in mere months, underscores the relentless human endeavor to understand and leverage technology for smarter, more efficient urban environments. This work has not only shed light on the current state of AI-driven systems in urban surveillance but also hinted at the vast potential for future advancements. As we stand at the intersection of technological innovation and urban development, it is evident that the journey towards truly intelligent cities is just beginning, filled with opportunities for further research and implementation.

6.1 Research outcomes

In the preceding chapters of this thesis, we embarked on an extensive exploration of the multifaceted domain of neural networks, which forms the bedrock of our analytical framework for smart city analytics. Our journey commenced with a thorough historical review, tracing the evolution of neural networks and dissecting their fundamental components. This deep dive illuminated the core elements – the computational units, the intricate layering, and the dynamic interconnections that collectively shape a network's architecture. A clear comprehension of these principles is not merely academic but instrumental in harnessing neural networks' full potential in practical applications. Thanks to this exploration, we found characteristics which are not present in other kind of AI models, like ML, which are: flexibility of the neural architecture, the multimodality of the models, and the development of learning strategies which are appropriate for the determined task. The culmination of this research has produced two kinds of results: one for the forecasting, with the introduction of the ENCODE methodology as discussed in Chapter 4, another for the development of a real-time high resolution video feed based analytics system which is then combined with ENCODE under the umbrella named SM.A.R.T., discussed in Chapter 5. The integration of these advanced technologies into smart city infrastructures marks a major advancement toward the vision of interconnected, data-driven urban spaces. This venture not only highlights the practical applicability of neural networks but also emphasizes their pivotal role in transforming urban living and governance. The research methodologies explored throughout this study have also been applied to additional projects, resulting in the publication of the following works:

- Giampaolo, F., Gatta, F., Prezioso, E., Cuomo, S., Zhou, M., Fortino, G., & Piccialli, F. (2023). ENCODE - Ensemble neural combination for optimal dimensionality encoding in time-series forecasting. Information Fusion, 100, 101918.
- Prezioso, E., Giampaolo, F., Izzo, S., Savoia, M., & Piccialli, F. (2023, October).

Integrating Object Detection and Advanced Analytics for Smart City Crowd Management. In 2023 IEEE International Conference on Networking, Sensing and Control (ICNSC) (Vol. 1, pp. 1-6). IEEE.

- Piccialli, F., Cuomo, S., Giampaolo, F. and Prezioso, E., Universita Degli Studi di Napoli di Federico II, 2023. Prediction method and related system. U.S. Patent Application 17/815,737.

- Chiaro, D., Prezioso, E., Ianni, M., & Giampaolo, F. (2023). FL-Enhance: A federated learning framework for balancing non-IID data with augmented and shared compressed samples. Information Fusion, 98, 101836.

- Prezioso, E., Giampaolo, F., Mazzocca, C., Bujari, A., Mele, V., & Amato, F. (2021). Machine Learning insights for behavioural data analysis supporting the Autonomous Vehicles scenario. IEEE Internet of Things Journal.

- Piccialli, F., Giampaolo, F., Prezioso, E., Camacho, D., & Acampora, G. (2021). Artificial intelligence and healthcare: Forecasting of medical bookings through multi-source time-series fusion. Information Fusion, 74, 1-16.

- Piccialli, F., Giampaolo, F., Prezioso, E., Crisci, D., & Cuomo, S. (2021). Predictive analytics for smart parking: A deep learning approach in forecasting of iot data. ACM Transactions on Internet Technology (TOIT), 21(3), 1-21.

6.2 Final Thoughts

Throughout the discussion, we have underscored several persistent challenges in the field, including but not limited to scalability, interpretability, and the pursuit of more efficient learning algorithms. Despite these challenges, the central narrative of this thesis emphasizes the significant impact and escalating relevance of neural networks within contemporary technological paradigms, particularly in the smart city context. Their fundamental role in computational methodologies is not just a reflection of their current capabilities but also a testament to their immense potential for future innovations. This thesis, through its rigorous examination of neural networks, advanced forecasting methods, and cutting-edge object detection models such as YOLO, illuminates their indispensability and the boundless possibilities they present for innovation. This profound understanding and continuous evolution of neural networks are paramount in shaping the future trajectory of computational research and advancing the development of smart cities.

Bibliography

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*, MIT press (2016).

[2] Y. LeCun, Y. Bengio, and G. Hinton, *Deep learning*, Nature **521**, 436–444 (2015).

[3] D. Luckey, H. Fritz, D. Legatiuk, K. Dragos, and K. Smarsly. *Artificial intelligence techniques for smart city applications.* In *Proceedings of the 18th International Conference on Computing in Civil and Building Engineering: ICCCBE 2020*, pages 3–15. Springer, (2021).

[4] A. M. Turing, *Computing Machinery and Intelligence*, Mind **59**, 433–460 (1950).

[5] F. Rosenblatt, *The perceptron: A probabilistic model for information storage and retrieval in the brain*, Psychological Review **65**, 386–408 (1958).

[6] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning representations by back-propagating errors*, Nature **323**, 533–536 (1986).

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton, *Imagenet classification with deep convolutional neural networks*, Advances in neural information processing systems **25** (2012).

[8] P. Kumar, S. Chauhan, and L. K. Awasthi, *Artificial intelligence in healthcare: review, ethics, trust challenges & future research directions*, Engineering Applications of Artificial Intelligence **120**, 105894 (2023).

[9] R. Kumar, N. Grover, R. Singh, S. Kathuria, A. Kumar, and A. Bansal. *Imperative Role of Artificial Intelligence and Big Data in Finance and Banking Sector.* In *2023 International Conference on Sustainable Computing and Data Communication Systems (ICSCDS)*, pages 523–527. IEEE, (2023).

[10] H. Ning, R. Yin, A. Ullah, and F. Shi, *A survey on hybrid human-artificial intelligence for autonomous driving*, IEEE Transactions on Intelligent Transportation Systems **23**, 6011–6026 (2021).

[11] S. Zhong, K. Zhang, M. Bagheri, J. G. Burken, A. Gu, B. Li, X. Ma, B. L. Marrone, Z. J. Ren, J. Schrier, et al., *Machine learning: new ideas and tools in environmental science and engineering*, Environmental Science & Technology **55**, 12741–12754 (2021).

[12] R. Kaur, D. Gabrijelčič, and T. Klobučar, *Artificial intelligence for cybersecurity: Literature review and future research directions*, Information Fusion , 101804 (2023).

[13] Y. Akkem, S. K. Biswas, and A. Varanasi, *Smart farming using artificial intelligence: A review*, Engineering Applications of Artificial Intelligence **120**, 105899 (2023).

[14] T. K. Chiu, Q. Xia, X. Zhou, C. S. Chai, and M. Cheng, *Systematic literature review on opportunities, challenges, and future research recommendations of artificial intelligence in education*, Computers and Education: Artificial Intelligence **4**, 100118 (2023).

[15] A. Oksanen, A. Cvetkovic, N. Akin, R. Latikka, J. Bergdahl, Y. Chen, and N. Savela, *Artificial intelligence in fine arts: A systematic review of empirical research*, Computers in Human Behavior: Artificial Humans , 100004 (2023).

[16] P. Rodriguez-Garcia, Y. Li, D. Lopez-Lopez, and A. A. Juan, *Strategic decision*

making in smart home ecosystems: A review on the use of artificial intelligence and Internet of things, Internet of Things , 100772 (2023).

[17] A. M. Townsend, *Smart cities: Big data, civic hackers, and the quest for a new utopia*, WW Norton & Company (2013).

[18] Z. Engin, J. van Dijk, T. Lan, P. A. Longley, P. Treleaven, M. Batty, and A. Penn, *Data-driven urban management: Mapping the landscape*, Journal of Urban Management **9**, 140–150 (2020).

[19] J. Kandt and M. Batty, *Smart cities, big data and urban policy: Towards urban analytics for the long run*, Cities **109**, 102992 (2021).

[20] A. N. Muhammad, A. M. Aseere, H. Chiroma, H. Shah, A. Y. Gital, and I. A. T. Hashem, *Deep learning application in smart cities: recent development, taxonomy, challenges and research prospects*, Neural computing and applications **33**, 2973–3009 (2021).

[21] E. Ismagilova, L. Hughes, N. P. Rana, and Y. K. Dwivedi, *Security, privacy and risks within smart cities: Literature review and development of a smart city interaction framework*, Information Systems Frontiers , 1–22 (2020).

[22] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, *Attention is all you need*, Advances in neural information processing systems **30** (2017).

[23] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, *Dropout: a simple way to prevent neural networks from overfitting*, The journal of machine learning research **15**, 1929–1958 (2014).

[24] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. *Focal loss for dense object detection.* In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, (2017).

[25] D. P. Kingma and M. Welling. *Auto-Encoding Variational Bayes.* In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, (2014).

[26] Y. N. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio. *Identifying and attacking the saddle point problem in high-dimensional non-convex optimization.* In *Advances in neural information processing systems*, pages 2933–2941, (2014).

[27] S. Ruder, *An overview of gradient descent optimization algorithms*, arXiv preprint arXiv:1609.04747 (2016).

[28] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, arXiv preprint arXiv:1412.6980 (2014).

[29] P. J. Brockwell and R. A. Davis, *Introduction to time series and forecasting*, Springer (2002).

[30] H. Lütkepohl, *New introduction to multiple time series analysis*, Springer Science & Business Media (2005).

[31] G. Box and G. Jenkins, *Time Series Analysis: Forecasting and Control*, Holden-Day (1976).

[32] G. Zhang, B. E. Patuwo, and M. Y. Hu, *Forecasting with artificial neural networks:: The state of the art*, International journal of forecasting **14**, 35–62 (1998).

[33] C. C. Holt, *Forecasting seasonals and trends by exponentially weighted moving averages*, O.N.R. Research Memorandum **52** (1957).

[34] P. R. Winters, *Forecasting sales by exponentially weighted moving averages*, Management Science **6**, 324–342 (1960).

[35] G. E. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, *Time series analysis: forecasting and control*, John Wiley & Sons (2015).

[36] R. J. Hyndman and G. Athanasopoulos, *Forecasting: principles and practice*, OTexts (2018).

[37] R. E. Kalman, *A new approach to linear filtering and prediction problems*, Journal of Basic Engineering **82**, 35–45 (1960).

[38] R. B. Cleveland, W. S. Cleveland, J. E. McRae, and I. Terpenning, *STL: A Seasonal-Trend Decomposition Procedure Based on Loess*, Journal of Official Statistics **6**, 3–73 (1990).

[39] G. M. Jenkins and M. Priestley, *The spectral analysis of time-series*, Journal of the Royal Statistical Society: Series B (Methodological) **19**, 1–12 (1957).

[40] P. Jing, Y. Su, X. Jin, and C. Zhang, *High-order temporal correlation model learning for time-series prediction*, IEEE transactions on cybernetics **49**, 2385–2397 (2018).

[41] Q. Shi, J. Yin, J. Cai, A. Cichocki, T. Yokota, L. Chen, M. Yuan, and J. Zeng, *Block Hankel Tensor ARIMA for Multiple Short Time Series Forecasting*, Proceedings of the AAAI Conference on Artificial Intelligence **34**, 5758–5766 (2020).

[42] S. J. Taylor and B. Letham, *Forecasting at scale*, The American Statistician **72**, 37–45 (2018).

[43] K.-J. Kim, *Financial time series forecasting using support vector machines*, Neurocomputing **55**, 307–319 (2003).

[44] K. Amasyali and N. M. El-Gohary, *A review of data-driven building energy consumption prediction studies*, Renewable and Sustainable Energy Reviews **81**, 1192–1205 (2018).

[45] K. Hornik, M. Stinchcombe, and H. White, *Multilayer feedforward networks are universal approximators*, Neural networks **2**, 359–366 (1989).

[46] D. Salinas, V. Flunkert, J. Gasthaus, and T. Januschowski, *DeepAR: Probabilistic forecasting with autoregressive recurrent networks*, International Journal of Forecasting **36**, 1181–1191 (2020).

[47] B. N. Oreshkin, D. Carpov, N. Chapados, and Y. Bengio. *N-BEATS: Neural basis expansion analysis for interpretable time series forecasting.* In *ICLR*. OpenReview.net, (2020).

[48] G. P. Zhang, *Time series forecasting using a hybrid ARIMA and neural network model*, Neurocomputing **50**, 159–175 (2003).

[49] X. Wang and M. Meng, *A Hybrid Neural Network and ARIMA Model for Energy Consumption Forcasting.*, J. Comput. **7**, 1184–1190 (2012).

[50] S. S. Rangapuram, M. W. Seeger, J. Gasthaus, L. Stella, Y. Wang, and T. Januschowski, *Deep state space models for time series forecasting*, Advances in neural information processing systems **31**, 7785–7794 (2018).

[51] H.-F. Yu, N. Rao, and I. S. Dhillon. *Temporal Regularized Matrix Factorization for High-dimensional Time Series Prediction.* In *NIPS*, pages 847–855, (2016).

[52] D. Salinas, M. Bohlke-Schneider, L. Callot, R. Medico, and J. Gasthaus, *High-dimensional multivariate forecasting with low-rank gaussian copula processes*, Advances in neural information processing systems **32** (2019).

[53] R. Sen, H.-F. Yu, and I. S. Dhillon, *Think globally, act locally: A deep neural network approach to high-dimensional time series forecasting*, Advances in neural information processing systems **32** (2019).

[54] G. Lai, W.-C. Chang, Y. Yang, and H. Liu. *Modeling long-and short-term temporal patterns with deep neural networks.* In *The 41st international ACM SIGIR*

conference on research & development in information retrieval, pages 95–104, (2018).

[55] S. Smyl, *A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting*, International Journal of Forecasting **36**, 75–85 (2020).

[56] D. Opitz and R. Maclin, *Popular ensemble methods: An empirical study*, Journal of artificial intelligence research **11**, 169–198 (1999).

[57] R. Polikar, *Ensemble based systems in decision making*, IEEE Circuits and Systems Magazine **6**, 21–45 (2006).

[58] D. H. Wolpert and W. G. Macready, *No free lunch theorems for optimization*, IEEE transactions on evolutionary computation **1**, 67–82 (1997).

[59] S. Lee, N. Ybarra, K. Jeyaseelan, S. Faria, N. Kopek, P. Brisebois, J. D. Bradley, C. Robinson, J. Seuntjens, and I. El Naqa, *Bayesian network ensemble as a multivariate strategy to predict radiation pneumonitis risk*, Medical physics **42**, 2421–2430 (2015).

[60] A. Galicia, R. Talavera-Llames, A. Troncoso, I. Koprinska, and F. Martínez-Álvarez, *Multi-step forecasting for big data time series based on ensemble learning*, Knowledge-Based Systems **163**, 830–841 (2019).

[61] D. Shaub, *Fast and accurate yearly time series forecasting with forecast combinations*, International Journal of Forecasting **36**, 116–120 (2020).

[62] G. Xie, Y. Qian, and S. Wang, *A decomposition-ensemble approach for tourism forecasting*, Annals of Tourism Research **81**, 102891 (2020).

[63] S. Al-Dahidi, P. Baraldi, E. Zio, and E. Legnani. *A dynamic weighting ensemble approach for wind energy production prediction.* In *2017 2nd International Conference on System Reliability and Safety (ICSRS)*, pages 296–302. IEEE, (2017).

[64] M. H. D. M. Ribeiro and L. dos Santos Coelho, *Ensemble approach based on bagging, boosting and stacking for short-term prediction in agribusiness time series*, Applied Soft Computing **86**, 105837 (2020).

[65] A. Chitra and S. Uma, *An ensemble model of multiple classifiers for time series prediction*, International Journal of Computer Theory and Engineering **2**, 454 (2010).

[66] W. Shen, V. Babushkin, Z. Aung, and W. L. Woon. *An ensemble model for day-ahead electricity demand time series forecasting.* In *Proceedings of the fourth international conference on Future energy systems*, pages 51–62, (2013).

[67] A. Vlasenko, N. Vlasenko, O. Vynokurova, Y. Bodyanskiy, and D. Peleshko, *A novel ensemble neuro-fuzzy model for financial time series forecasting*, Data **4**, 126 (2019).

[68] P. Musikawan, K. Sunat, and Y. Kongsorot, *Wind power forecasting using a heterogeneous ensemble of decomposition-based NNRW techniques*, ECTI Transactions on Computer and Information Technology **14**, 122–138 (2020).

[69] S. Kaushik, A. Choudhury, N. Dasgupta, S. Natarajan, L. A. Pickett, and V. Dutt. *Ensemble of multi-headed machine learning architectures for time-series forecasting of healthcare expenditures.* In P. Johri, J. K. Verma, and S. Paul, editors, *Applications of Machine Learning*, pages 199–216, Singapore, (2020). Springer Singapore.

[70] S. Krstanovic and H. Paulheim. *Ensembles of recurrent neural networks for robust time series forecasting.* In *International Conference on Innovative Techniques and Applications of Artificial Intelligence*, pages 34–46. Springer, (2017).

[71] J. Chen, G.-Q. Zeng, W. Zhou, W. Du, and K.-D. Lu, *Wind speed forecasting*

using nonlinear-learning ensemble of deep learning time series prediction and extremal optimization, Energy Conversion and Management **165**, 681–695 (2018).

[72] M. Tan, S. Yuan, S. Li, Y. Su, H. Li, and F. He, *Ultra-short-term industrial power demand forecasting using LSTM based hybrid ensemble learning*, IEEE Transactions on Power Systems **35**, 2937–2948 (2019).

[73] M. Kück, S. F. Crone, and M. Freitag. *Meta-learning with neural networks and landmarking for forecasting model selection: an empirical evaluation of different feature sets applied to industry data.* In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 1499–1506. IEEE, (2016).

[74] J. S. Armstrong, *Combining forecasts*, Springer (2001).

[75] S. Makridakis, E. Spiliotis, and V. Assimakopoulos, *The M4 competition: Results, findings, conclusion and way forward*, International Journal of Forecasting **34**, 802–808 (2018).

[76] J. C. Myburgh, C. Mouton, and M. H. Davel. *Tracking translation invariance in CNNs.* In *Southern African Conference for Artificial Intelligence Research*, pages 282–295. Springer, (2020).

[77] Z. Zou, K. Chen, Z. Shi, Y. Guo, and J. Ye, *Object Detection in 20 Years: A Survey*, Proceedings of the IEEE **111**, 257–276 (2023).

[78] A. Gupta, A. Anpalagan, L. Guan, and A. S. Khwaja, *Deep learning for object detection and scene perception in self-driving cars: Survey, challenges, and open issues*, Array **10**, 100057 (2021).

[79] P. Viola and M. Jones. *Rapid object detection using a boosted cascade of simple features.* In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, volume 1, pages I–I, (2001).

[80] D. G. Lowe, *Distinctive image features from scale-invariant keypoints*, International journal of computer vision **60**, 91–110 (2004).

[81] N. Dalal and B. Triggs. *Histograms of oriented gradients for human detection.* In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 886–893 vol. 1, (2005).

[82] P. Felzenszwalb, D. McAllester, and D. Ramanan. *A discriminatively trained, multiscale, deformable part model.* In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, (2008).

[83] Z.-Q. Zhao, P. Zheng, S.-t. Xu, and X. Wu, *Object detection with deep learning: A review*, IEEE transactions on neural networks and learning systems **30**, 3212–3232 (2019).

[84] J. R. Uijlings, K. E. Van De Sande, T. Gevers, and A. W. Smeulders, *Selective search for object recognition*, International journal of computer vision **104**, 154–171 (2013).

[85] R. Girshick, J. Donahue, T. Darrell, and J. Malik. *Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation.* In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, (2014).

[86] K. He, X. Zhang, S. Ren, and J. Sun, *Spatial pyramid pooling in deep convolutional networks for visual recognition*, IEEE transactions on pattern analysis and machine intelligence **37**, 1904–1916 (2015).

[87] R. Girshick. *Fast r-cnn.* In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, (2015).

[88] S. Ren, K. He, R. Girshick, and J. Sun, *Faster r-cnn: Towards real-time object detection with region proposal networks*, Advances in neural information processing systems **28** (2015).

[89] J. Dai, Y. Li, K. He, and J. Sun, *R-fcn: Object detection via region-based fully convolutional networks*, Advances in neural information processing systems **29** (2016).

[90] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie. *Feature pyramid networks for object detection.* In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, (2017).

[91] K. He, G. Gkioxari, P. Dollár, and R. Girshick. *Mask r-cnn.* In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, (2017).

[92] C. Szegedy, A. Toshev, and D. Erhan, *Deep neural networks for object detection*, Advances in neural information processing systems **26** (2013).

[93] P. O. O Pinheiro, R. Collobert, and P. Dollár, *Learning to segment object candidates*, Advances in neural information processing systems **28** (2015).

[94] D. Yoo, S. Park, J.-Y. Lee, A. S. Paek, and I. So Kweon. *Attentionnet: Aggregating weak directions for accurate object detection.* In *Proceedings of the IEEE international conference on computer vision*, pages 2659–2667, (2015).

[95] M. Najibi, M. Rastegari, and L. S. Davis. *G-cnn: an iterative grid based object detector.* In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2369–2377, (2016).

[96] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. *You only look once: Unified, real-time object detection.* In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, (2016).

[97] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. *Ssd: Single shot multibox detector.* In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*, pages 21–37. Springer, (2016).

[98] T. Diwan, G. Anirudh, and J. V. Tembhurne, *Object detection using YOLO: Challenges, architectural successors, datasets and applications*, multimedia Tools and Applications **82**, 9243–9275 (2023).

[99] J. Redmon and A. Farhadi. *YOLO9000: better, faster, stronger.* In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, (2017).

[100] J. Redmon and A. Farhadi. *YOLOv3: An Incremental Improvement*, (2018).

[101] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao. *YOLOv4: Optimal Speed and Accuracy of Object Detection*, (2020).

[102] G. Jocher. *YOLOv5 by Ultralytics*, (2020).

[103] Y. Jing, Y. Ren, Y. Liu, D. Wang, and L. Yu, *Automatic extraction of damaged houses by earthquake based on improved YOLOv5: A case study in Yangbi*, Remote Sensing **14**, 382 (2022).

[104] H. Wang, S. Zhang, S. Zhao, Q. Wang, D. Li, and R. Zhao, *Real-time detection and tracking of fish abnormal behavior based on improved YOLOV5 and SiamRPN++*, Computers and Electronics in Agriculture **192**, 106512 (2022).

[105] S. S. A. Zaidi, M. S. Ansari, A. Aslam, N. Kanwal, M. Asghar, and B. Lee, *A survey of modern deep learning based object detection models*, Digital Signal Processing **126**, 103514 (2022).

[106] A. Sarda, S. Dixit, and A. Bhan. *Object detection for autonomous driving using yolo [you only look once] algorithm.* In *2021 Third international conference on intelligent communication technologies and virtual mobile networks (ICICV)*, pages 1370–1374. IEEE, (2021).

[107] G. Jocher, A. Chaurasia, and J. Qiu. *Ultralytics YOLO*, (2023).

[108] E. Spiliotis, K. Nikolopoulos, and V. Assimakopoulos, *Tales from tails: On the empirical distributions of forecasting errors and their implication to risk*, International Journal of Forecasting **35**, 687–698 (2019).

[109] S. Makridakis, E. Spiliotis, and V. Assimakopoulos, *The M4 Competition: 100,000 time series and 61 forecasting methods*, International Journal of Forecasting **36**, 54–74 (2020).

[110] S. Makridakis, E. Spiliotis, and V. Assimakopoulos, *M5 accuracy competition: Results, findings, and conclusions*, International Journal of Forecasting **38**, 1346–1364 (2022).

[111] J. M. Bates and C. W. J. Granger, *The combination of forecasts*, J. Operat. Res. Soc. **20**, 451–468 (1969).

[112] R. T. Clemen, *Combining forecasts: A review and annotated bibliography*, International journal of forecasting **5**, 559–583 (1989).

[113] R. Adhikari. *A neural network based linear ensemble framework for time series forecasting.* In *Neurocomputing*, volume 157, pages 231–242. Elsevier, (2015).

[114] M. Kück, S. F. Crone, and M. Freitag. *Meta-learning with neural networks and landmarking for forecasting model selection an empirical evaluation of different feature sets applied to industry data.* In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 1499–1506. IEEE, (2016).

[115] G. Li, S. Wang, et al., *Sunspots time-series prediction based on complementary ensemble empirical mode decomposition and wavelet neural network*, Mathematical Problems in Engineering **2017** (2017).

[116] W. Liu, W. D. Liu, and J. Gu, *Forecasting oil production using ensemble empirical model decomposition based Long Short-Term Memory neural network*, Journal of Petroleum Science and Engineering **189**, 107013 (2020).

[117] Z. Lu, J. Xia, M. Wang, Q. Nie, and J. Ou, *Short-term traffic flow forecasting via multi-regime modeling and ensemble learning*, Applied Sciences **10**, 356 (2020).

[118] E. Soares, P. C. J. Costa, B. Costa, and D. Leite, *Ensemble of evolving data clouds and fuzzy models for weather time series prediction*, Applied Soft Computing **64**, 445–453 (2018).

[119] L. Wang, Z. Wang, H. Qu, and S. Liu, *Optimal forecast combination based on neural networks for time series forecasting*, Applied Soft Computing **66**, 1–17 (2018).

[120] B. Yang, Z.-J. Gong, and W. Yang. *Stock market index prediction using deep neural network ensemble.* In *2017 36th Chinese Control Conference (CCC)*, pages 3882–3887. IEEE, (2017).

[121] L. Yu, Y. Zhao, and L. Tang, *Ensemble forecasting for complex time series using sparse representation and neural networks*, Journal of Forecasting **36**, 122–138 (2017).

[122] X. Zhang, Y. Li, S. Lu, H. F. Hamann, B.-M. Hodge, and B. Lehman, *A solar time based analog ensemble method for regional solar power forecasting*, IEEE Transactions on Sustainable Energy **10**, 268–279 (2018).

[123] W. Zhao, F. Wang, and D. Niu, *The application of support vector machine in load forecasting*, Journal of Computational Physics **7**, 1615–1622 (2012).

[124] Y. Zhao, J. Li, and L. Yu, *A deep learning ensemble approach for crude oil price forecasting*, Energy Economics **66**, 9–16 (2017).

[125] A. Kausar, M. Ishtiaq, M. A. Jaffar, and A. M. Mirza. *Optimization of ensemble based decision using PSO.* In *Proceedings of the World Congress on Engineering*, volume 1, pages 1–6. IAENG, (2010).

[126] Z. Wu and Y. Chen. *Genetic algorithm based selective neural network ensemble.* In *IJCAI-01: Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, volume 3, pages 1323–1328. Morgan Kaufmann Publishers Inc., (2001).

[127] Z. Wu, S. Pan, G. Long, J. Jiang, X. Chang, and C. Zhang. *Connecting the dots: Multivariate time series forecasting with graph neural networks.* In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 753–763, (2020).

[128] C. Chen, K. Petty, A. Skabardonis, P. Varaiya, and Z. Jia, *Freeway performance measurement system: mining loop detector data*, Transportation Research Record **1748**, 96–102 (2001).

[129] C. Nadeau and Y. Bengio, *Inference for the generalization error*, Advances in neural information processing systems **12** (1999).

[130] Z. Zheng, P. Wang, W. Liu, J. Li, R. Ye, and D. Ren. *Distance-IoU Loss: Faster and Better Learning for Bounding Box Regression.* In *The AAAI Conference on Artificial Intelligence (AAAI)*, pages 12993–13000, (2020).

[131] Z. Zheng, P. Wang, D. Ren, W. Liu, R. Ye, Q. Hu, and W. Zuo, *Enhancing Geometric Factors in Model Learning and Inference for Object Detection and Instance Segmentation*, IEEE Transactions on cybernetics **52**, 8574–8586 (2021).

[132] X. Li, W. Wang, L. Wu, S. Chen, X. Hu, J. Li, J. Tang, and J. Yang, *Generalized focal loss: Learning qualified and distributed bounding boxes for dense object detection*, Advances in Neural Information Processing Systems **33**, 21002–21012 (2020).

[133] F. M. Talaat and H. ZainEldin, *An improved fire detection approach based on*

YOLO-v8 for smart cities, Neural Computing and Applications **35**, 20939–20954 (2023).

[134] S. Tamang, B. Sen, A. Pradhan, K. Sharma, and V. K. Singh, *Enhancing covid-19 safety: Exploring yolov8 object detection for accurate face mask classification*, International Journal of Intelligent Systems and Applications in Engineering **11**, 892–897 (2023).

[135] A. Aboah, B. Wang, U. Bagci, and Y. Adu-Gyamfi. *Real-time multi-class helmet violation detection using few-shot data sampling technique and yolov8*. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5349–5357, (2023).

[136] W. Luo, J. Xing, A. Milan, X. Zhang, W. Liu, and T.-K. Kim, *Multiple object tracking: A literature review*, Artificial intelligence **293**, 103448 (2021).

[137] Y. Zhang, P. Sun, Y. Jiang, D. Yu, F. Weng, Z. Yuan, P. Luo, W. Liu, and X. Wang. *Bytetrack: Multi-object tracking by associating every detection box*. In *European Conference on Computer Vision*, pages 1–21. Springer, (2022).

[138] H. Agrawal, A. Halder, and P. Chattopadhyay, *A systematic survey on recent deep learning-based approaches to multi-object tracking*, Multimedia Tools and Applications (2023).

[139] D. Gloudemans, G. Zachár, Y. Wang, J. Ji, M. Nice, M. Bunting, W. W. Barbour, J. Sprinkle, B. Piccoli, M. L. D. Monache, et al. *So you think you can track?* In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 4528–4538, (2024).

[140] M. A. Fischler and R. C. Bolles, *Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography*, Communications of the ACM **24**, 381–395 (1981).