Università degli Studi della Tuscia di Viterbo

Dipartimento per la Innovazione nei Sistemi Biologici, Agroalimentari e Forestali

PhD Programme in

Sciences, Technologies and Biotechnologies for Sustainability - XXXVII cycle

HPC technologies for the development of neural networks and analysis of big data in chemistry, biology, agrifood, forestry and ecology

s.s.d CHEM-03/A

PhD student
Dott. Armando Camerlingo

PhD Coordinator
Prof. Andrea Vannini

PhD Supervisor
Prof. Nico Sanna

Tesi redatta con il sostegno finanziario del programma FSE-REACT-EU,

PON "Ricerca e Innovazione" 2014-2020 (D.M. 1061/2021) Azione IV.4

"Dottorati e contratti di ricerca su tematiche dell'innovazione"

Anno Accademico 2023/24 (A.Y. 2023/24)

Contents

Introduction **1**

1 HPC: an overview and the DIBAF cluster management **10**

2 HPC in Computational Chemistry: analysis of the interaction between Borazine and Noble Gases **29**

3 HPC and Deep Learning: Hyperparameters' optimization with HPE Machine Learning Development Edition **47**

4 DL Applications on 3 Different Types of Datasets **60**

Conclusions **97**

Acknowledgments **101**

Bibliography **103**

Introduction

Thanks to the advances in technology in the last decades, new opportunities have risen in many scientific fields because of the easy retrieval of a great quantity of experimental data. Obviously, this scenario gave birth to new problems and challenges related to the memorization, management and analysis of the large amount of data obtained, making Big Data and Machine Learning very important topics in many scientific fields. Big Data can be defined as data that exceed the capacity and capability of conventional technologic methods and systems [1], a situation that today happens more than ever due to the development of faster network communication methods and more performing measurement tools. Often Big Data is characterized by the so-called 3V:

- Volume: the quantity of data;
- Variety: the type of data that can be structured, unstructured or an in-between;
- Velocity: often data can be already fully accessible, but it can be also

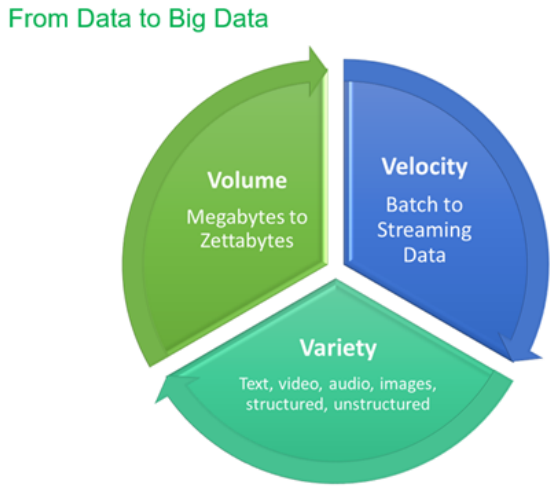

Figure 1 : The 3V of Big Data

produced in real-time with a fixed time interval in which it is sent to the receiver;

New data gathering and management methods were developed to access them and extract information not retrievable with traditional statistical methods. The handling of Big Data in the research process can be described briefly in 4 main steps. These steps can be viewed also as different tiers of insight into the data gathered in which you trade-off the velocity versus the quality in the results obtained from the analysis through the use of more performing technology. The first step is the gathering of the data of interest from digital sources usually, but sometimes also analogic through sensors (devices coupled to machinery, plants, etc.). This information is aggregated (and converted

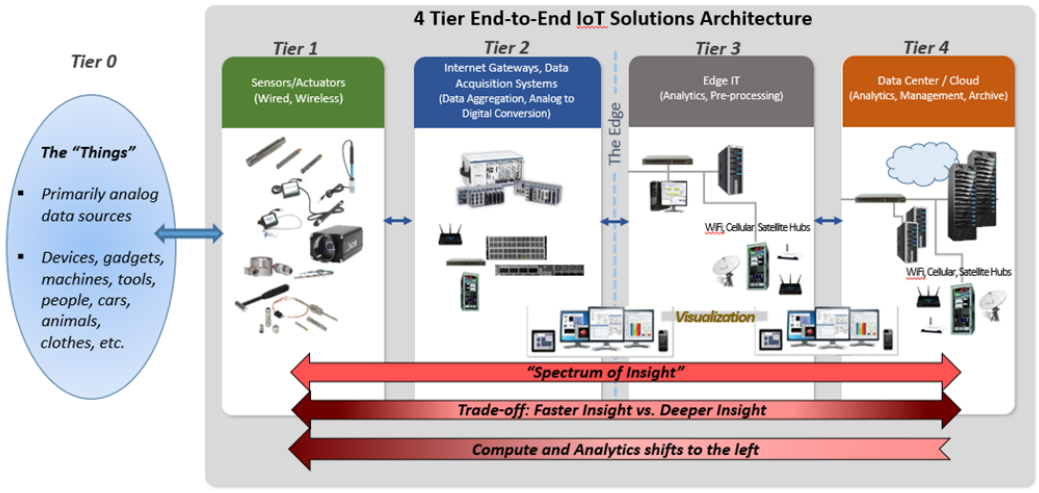

Figure 2 : IoT solutions Architecture

from analog to digital if necessary) through data acquisition systems to be then transmitted by a data-gateway. The next step is the pre-processing of the data, i.e. a series of analyses and operations to guarantee the "quality" of the digital representation of the dataset. This procedure may include check on missing sample values, filters, removal of duplicates or outliers and other manipulations to "clean" the dataset. The third and crucial step is the passage of information from the location in which the data is acquired (and often pre-processed) to more powerful systems to analyse the data, very often elsewhere. The dimension of the datasets can impact the effort and the time of the transmission to the destination system. A solution to this problem is the Edge IT, i.e. a computation model in which the data processing hap-

pens as near as possible to where data is acquired. Also, this increment in the quantity of data is promoting the development of new technology in distributed networks with improved latency and bandwidth. Examples of these innovations are the LoRaWAN and the InfiniBand, two communication protocols at the opposite sides of the bandwidth-latency spectrum, but both at the top focus of tech development for best performance for their purposes. Long Range Wide Area Network (LoRaWan) is a networking protocol that works on very low bandwidth network making it able to transmit data in wider areas and with low power, at the cost of having an high latency and small throughput. These characteristics make LoRaWAN very suitable for IoT sensors, reducing the power need and subsequently also the cost to connect all the units often dislocated on a wide geographic area to the network. InfiniBand (IB) is a communications standard with very high transmission bandwidth and low latency used mainly in HPC clusters. It is used to link the various nodes of aggregated computing systems from small cluster to supercomputers and to the storage device to guarantee high I/O speed. These characteristics makes IB very useful in an HPC environment in which often computational processes are bottle-necked by I/O operations and/or overhead communication time between the nodes of the cluster. The last step is the storage of the data to be utilized to extract information with different possible techniques, the main two being ETL (Extract/Transform/Load) and ELT (Extract/Load/Transform) approaches. After the extraction, ETL

methods apply various transformations to prepare the data in the desired format and then load it in a data warehouse ready to be used by the developers and the applications. Instead, ELT methods swap the two steps by ingesting first the data inside the data warehouse and then apply any needed transformations. Obviously ETL prioritizes the manipulation of the data, while ELT is more fast in the transferral by delaying all the format operations. ETL is more efficient when used in Edge IT solutions because the transformations are performed before the transfer, reducing the data to be transmitted to the storage. Instead, ELT techniques have an heavier load on the network because all the data is transferred to the datacenter, but its computational power give the possibility to perform more complex operations on the data to prepare them for the analyses. Thus, the choice between the two class of techniques is heavily dependent on the data acquisition and data manipulation processes and their weights on the workflow. This is another example of how the nature of data and scope of applications have an impact on the data processing algorithms and it must be object of careful thinking by the data managers.

During this phase, a Data Lake is created, a multi-standard repository containing both the raw and the transformed data to be used for reporting, analytics, visualization and/or as input to Machine Learning algorithms [2]. The Data Lake must be capable to manage different types of data:

- Structured, such as data stored in relational databases;

- Semi-structured, such as XML, ASCII or data from websites;
- Unstructured, such as sensors measurements

and must present a consistent vision of the dataset to the middleware, a higher software stack level, in order to simplify its use as input to algorithms.

Machine Learning (ML) is one of the most efficient and promising approach to the anlysis of Big Data. ML can be described as the study of algorithms that have the capability to improve their results automatically through "experience" by processing datasets of samples (training set) [3].The ML techniques can be divided in two categories: supervised and unsupervised. The first kind of algorithms tries to find a function mapping the input data to one or more target variables and the elements of the training set are "labeled", meaning the output values for them are given, while the second type of models tries to find unknown common properties in a set of samples without known correlation. The training of a supervised ML model is performed with the objective to minimize a loss function on the model output and the real known value through an optimizer that works on the weights associated to the units. Both loss function and optimizer can be chosen at the time of training, a choice that is as impactful as the model structure to achieve good results in the prediction. A correct tuning can improve drastically the performance of the algorithm. Often the numerical precision is sacrificed in order to reduce the computational costs, dropping from double

to single/half floating-point representation of real numbers, usually needed to manage the big size of the dataset. Nonetheless in some scientific fields, results are required in double precision to be useful, such as quantum and classical simulations of chemical systems from a single molecule to biological aggregates. So one of the solutions is to reduce the parameters used by the model, identifying the ones that impact the model the most and use the highest numerical precision without blowing up the computational time. Computational Chemistry is one of those scientific fields in which often there is a need of high numerical precision to get meaningful results making the computations intensive and very time-consuming. ML can be used to replicate these standard methods at a fraction their cost, retaining the numerical precision [4]. An example of application of ML in computational chemistry is the sampling of the Potential Energy Surface (PES) in atomic simulations using ML potentials instead of standard Quantum Mechanics (QM) algorithms. These NN-based methods are trained on datasets from results of previous QM calculations in order to be used on large systems with many atoms [5]. Other fields in which Big Data and ML play a major role are surely ecology and agriculture [6] [7]. The introduction of new sensors and new technologies for the monitoring of on-site variables made the obtainment of data in great quantity to be analysed easier. Examples of this approach are the genomic analysis of breeding species to isolate the best genes to adapt to climatic changes [9] or the study of the correlation between the quality of

the grapes cultivated and problems in the wine fermentation [10]. So, there is a great need to develop and master new tools to help building ML models capable to deal with this new and expanding surge of data and to make the best use of High Performance Computing machines. The main results achieved during the PhD were the following:

- Firstly, the DIBAF (Dipartimento per l'Innovazione nei sistemi Biologici, Agricoli e Forestali) cluster was reorganized and renewed both on the hardware and software side to host the tools necessary for the scientific activities and also to be available for the other research groups. A new Rancher and Kubernetes environment was provided to run the services already running on the cluster and ready for new applications. This software stack for managing microservices is recently becoming a standard in academic and enterprise clusters and during the PhD new ways to exploit HPC technologies were employed to install and use Rancher in an optimal way. In particular, I focused on the use of InfiniBand, an high-performance network subsystem, and GPUs.

- Using the DIBAF cluster, I performed a computational chemistry study on the interaction of Borazine and Noble Gases, exploring deeper these complexes previously analysed by other studies. The findings were reported in a published paper [8].

- Deep Learning algorithms developed and run on the DIBAF cluster

were used to analyse different datasets of interest of research groups of the department. In particular, I devised a CNN model to retrieve information from 1-dimensional spectra, that, even with a single convolutional layer, was able to achieve comparable or better performance of standard chemometrics algorithms. The results have been reported in a submitted paper, now under review.

In the first chapter, the activities of re-organization and renovation of the DIBAF (Dipartimento per l'Innovazione nei sistemi Biologici, Agricoli e Forestali) cluster are described, an important step to obtain the tools necessary to the PhD project. In the second chapter, the studies done in Computational Chemistry are reported and the results obtained in the analysis of the interaction between Borazine and Noble gases, preparatory for one of the application of the research performed during the PhD. The third chapter is dedicated to a description of Deep Learning techniques and of MLDE (Machine Learning Development Environment), a new tool from Hewlett Packard Enterprise to develop and execute neural networks on HPC, with a focus on hyperparameters' selection. In the last chapter, the ML applications on datasets of different origin developed during the PhD are presented.

Chapter 1

HPC: an overview and the DIBAF cluster management

High Performance Computing (HPC) is a term that groups different disciplines, related to technologies, procedures and tools designed to improve the computational capability to solve problems in the most efficient way possible [11]. An HPC system has all the basic components of a classic desktop computer with the difference being the scale and the technology implemented. The biggest HPC clusters in the world have thousands of nodes (a node being a single computational unit) resulting in millions of cores and performance in the order of Exaflop/s (floating operations per second) [12], but even an HPC system in the order of dozens of nodes represents a major improvement for computations compared to a standard desktop computer.

The DIBAF cluster is composed by $\sim$30 nodes (identified by the prefix dn), connected by 1/10 Gbps ethernet and FDR Infiniband (IB) through 3 IB switches and 6 eth switches, 4 with 1 Gbps ports and 2 with 10 Gbps ports.

During my PhD, I participated in the renovation and re-organization of the datacenter at DIBAF (Dipartimento per l'Innovazione nei sistemi Biologici, Agricoli e Forestali), that was divided in two phases. First, an "hardware" phase in which the nodes, the switches and the MSA (Modular Storage Array) storage were moved to new racks cooled by an in-row system, a solution in which the cluster components are isolated from the environment and cooled directly from inside the racks instead of using external sources. Beside the new racks, a new Uninterruptible Power Supply (UPS) was added to prevent power outage and a new IB switch to have available ports for future additions to the cluster. The nodes were re-cabled, both for the ethernet and IB ports, both intra-rack and inter-rack and a file reporting all connections was compiled in order to have an easy way to overview the cluster network. Also a new KVM (keyboard, video, mouse) console was installed to connect up to 16 machines to access them directly through a LCD display and keyboard. The new hardware and the physical re-organization of the existing components was performed mainly by the technicians of the cooling system seller and Prisma SpA, supported and guided by our research group, formed by me, prof. Nico Sanna and dr. Costantino Zazza. I also participated during the re-cabling of the nodes, switches and storage and the interconnection optimization.

The nodes can be divided in different groups:

- Hvnarten: virtualization nodes with Windows Server installations. Used

Figure 1.1: Racks with In-Row Cooling system

to run service Virtual Machines (VM), such as the login front-end, the firewall and the Active Directory;

- dn16, dn17: HPE DL380 gen10 nodes that serve as NFS servers (dn16 main, dn17 backup) that mount the MSA storage ($\sim$ 250 TB) and share it to the other nodes in the cluster. Besides the 1 Gbps connection to the other machines, there is a 10Gbps connection that runs from the 2 servers to the building D in the Riello Campus, in which the offices of many users are situated, in order to provide a fast way to upload and download files to and from the cluster;
- K80 nodes: dn12-dn15 are HPE DL380 gen9 with Nvidia K80s GPUs;
- T4 nodes: dn23-dn27 are HPE DL380 gen10 nodes equipped with 4 Nvidia T4 GPUs;
- CPU nodes: all the remaining HPE DL380 gen10 dn nodes, CPU only;
- older nodes: dn05-dn08, used for educational purpose in University and PhD courses;
- Kubernetes Cluster: dn29, dn30, two DL380 gen10 nodes with 3 Nvidia T4 GPU and dn31, dn32 two DL380 gen10+ nodes;

Besides this "hardware" re-configuration, changes were made also on the software stack present on the nodes to use and manage them in order to provide a multi-purpose environment to the university staff. During this phase,

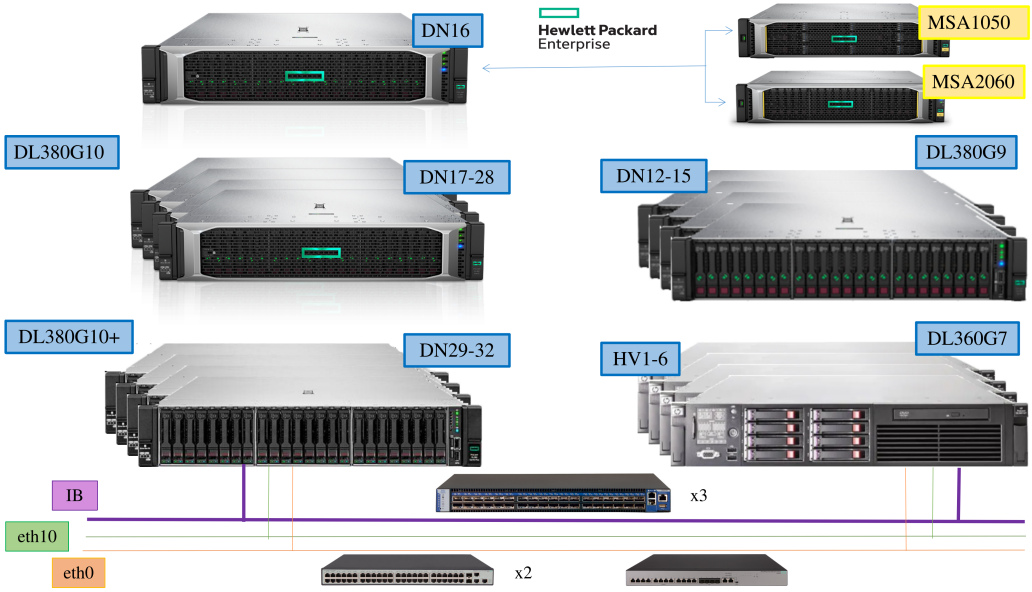

Figure 1.2: DIBAF HPC cluster hardware layout

the experience I gained during the industrial internship required by the "PON - Ricerca e Innovazione 2014-2020" PhD fellowship [15] was very useful. In fact, I did an internship in Prisma S.p.A. which co-funded the PhD, as an HPC support specialist in a team of 4 people for one of their clients. Specifically, my role consisted in the management and study of procedures on an HPC system composed by more than 128 nodes with the objective to widen my skills on those HPC technologies and help me achieve computational results targeting the objectives of my PhD project. During this year, I had the occasion to learn how to manage a cluster made out of more than a hundred computing nodes and the related software needed to use HPC applications, such as workload schedulers, different communication protocols like Infini-

band, shared filesystems and other tools essential to fully use the compute nodes in an efficient way. Among the other activities, we were responsible to advise the client in how to update and/or modify the cluster hardware. The way to make the best decisions is by doing extensive benchmarks. A well-made benchmark is fundamental to assess the possible capabilities of an HPC system or the usefulness of upgrades and structural changes. During my period at Prisma, the customer I was working at required the evaluation of the most appropriate storage and filesystem to use as work area for their HPC simulations, mainly Finite Elements Methods (FEM) algorithms. The three candidates to be used as scratch areas for the jobs were:

- Ext4[1] local directories in the SSD storage of the compute nodes: a standard filesystem used in UNIX machines;
- Tmpfs local directories in the shared memory of the compute nodes: temporary fileystem that store data in volatile memory (RAM) instead of a standard persistent drive;
- Lustre[2] distributed storage accessible to all compute nodes, connected through Infiniband: a parallel filesystem often used in HPC clusters because of the high I/O throughput [14];

The benchmark used to measure the I/O performances of the different setups on a single node with concurrent execution was IOzone V3.465 [13] which can

[1] fourth extended filesystem based on journaling
[2] union of the words Linux and Cluster

test the performance of different file operations among which we chose read, write, re-read and re-write (these last two use memory cache or I/O buffer) as metrics. The tool also permits to use different file and chunk size to verify the system response at different I/O intensities. Overall, for this kind of benchmark we used the following values:

- Chunk size: 4096 B, 8192 B, 16384 B;
- File size: 1024 MB, 2048 MB, 4096 MB;
- Number of threads:1, 2, 4, 6, 8, 16, 32;

From the different measurements, the local filesystems have overall better I/O, with tmpfs being the one with better I/O bandwidth. The effect of caching is outstanding on file operations, at the cost of having a larger RAM memory space on the computing nodes to load the tmpfs filesystem. Besides this results, the benchmark showed also that Lustre can benefit of improved caching/buffering mechanism in particular at an higher number of threads. During the same activity, the effect of the update of the OS of the compute nodes from Centos 7.9 to RockyLinux 8.6 on I/O operations was tested. The same tests as above were performed on two nodes with same technical specifications and different OS. The results were then analysed and visualised using Microsoft Excel. In figures 1.3, 1.4 and table 1.1, the results for file size equal to 4096 MB and 1 thread are reported.

Passing from Centos 7.9 to RockyLinux 8.6 improved greatly:

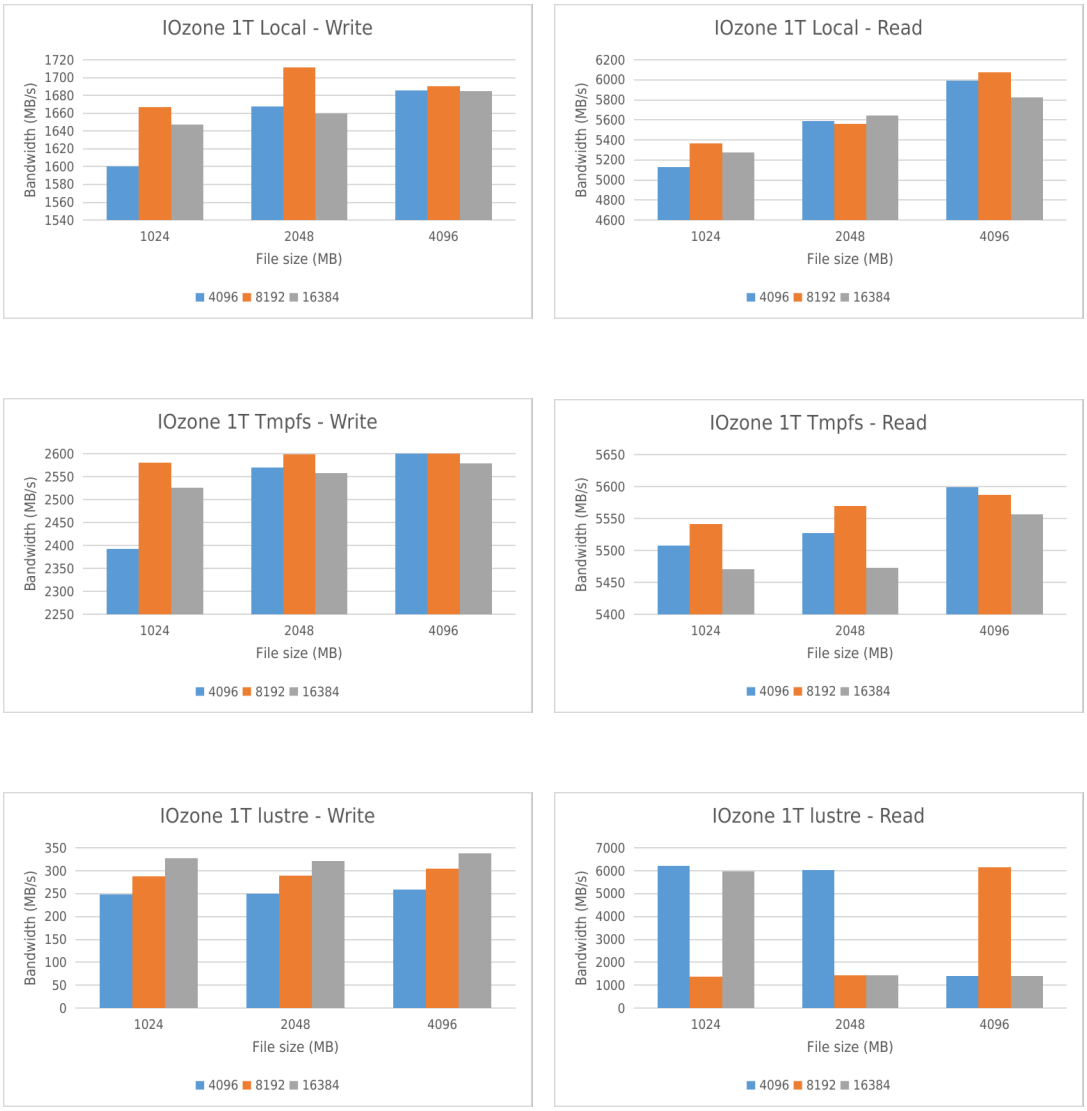

Figure 1.3: IOzone benchmark results for CentOS 7.9. I/O bandwidth vs filesize at different chunk size: blue 4KB, orange 8KB and gray 16KB

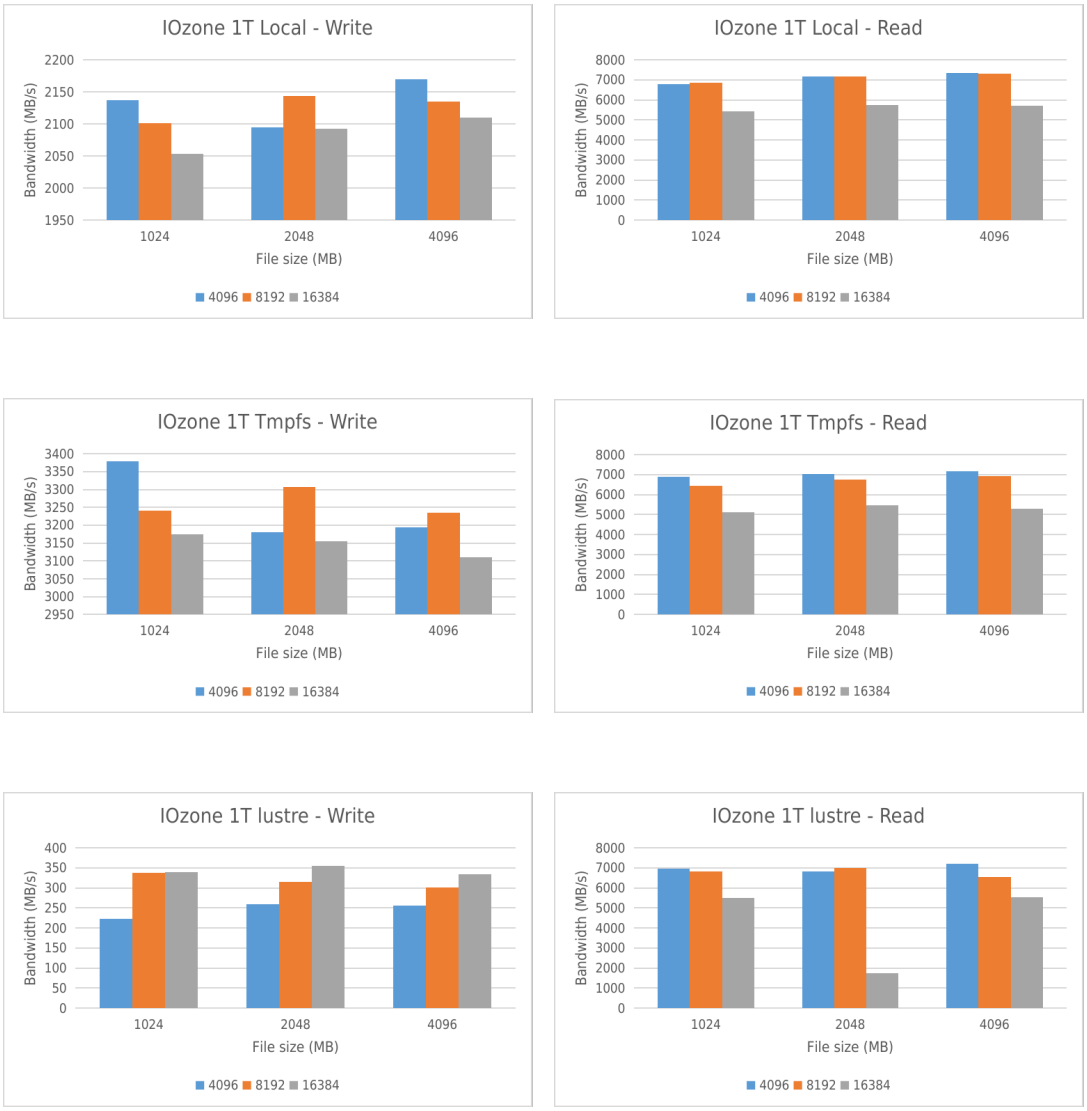

Figure 1.4: IOzone benchmark results for RockyLinux 8.6. I/O bandwidth vs filesize at different chunk size: blue 4KB, orange 8KB and gray 16KB

	RH6.X		**Centos7.9**		**Rocky8.6**	
	Write	*Read*	*Write*	*Read*	*Write*	*Read*
Local	N/A	N/A	1686	5990	2170	7335
Tmpfs	N/A	N/A	2616	5599	3194	7153
Lustre	229	5112	259	1411	256	7188
Local			**100%**	**100%**	**100%**	**100%**
Tmpfs			**155%**	93%	**147%**	98%
Lustre			15%	24%	12%	98%
Local			**100%**	**100%**	**129%**	**122%**
Tmpfs			**100%**	**100%**	**122%**	**128%**
Lustre			**100%**	**100%**	99%	**509%**

Table 1.1: Comparison of I/O performance between different filesystem and OS

- Pure I/O;
- Cached I/O;
- Buffering I/O;

The best results were obtained with Rocky8.6, the newest OS, and Lustre, due to its advantages in parallel I/O and distributed nature. These finding helped the client to choose Lustre to use as working area and also to decide to upgrade the OS to Rocky8.6 on compute nodes.

The software stack present on the DIBAF cluster is heterogenous in order to assist the different research groups that need HPC for their research. The use of the cluster can be divided in two main areas (in some case overlapping): simulations and services. Users can launch their jobs using SLURM workload manager [16] from the login node. There are different queues for different

needs that also reflect the different specifics of the nodes in the cluster:

- K80: 4 older nodes with Nvidia K80 GPUs used for jobs needing double precision GPU capability, such as computational material sciences;
- 4T4: 5 nodes with 4 Nvidia T4 GPUs, used for jobs for which single precision GPU capability is sufficient, such as classical molecular dynamics;
- A100: a single node with Nvidia A100 GPUs, used for specific jobs and machine learning;
- CPU: 7 nodes with no GPUs, used for jobs that does not benefit from GPU usage;

Besides standard jobs, users can also launch Docker [17] containers under the queues through Apptainer [18], that I installed on all nodes during cluster renovations. Apptainer is a container platform created to run application on HPC systems:

- In a simple and portable way: users can create their Sif files on a front-end computer and then they are ready to run on every machine with apptainer installed;
- In a secure way: same user inside and outside the container, no additional privilege on the machine the container is running on.

This software makes possible to take advantages of containerization on HPC for all users. First, all the environment with programs, libraries, data and script can be chosen and prepared by the user and it will be always the same, making it reproducible everywhere, a requisite for scientific research. Also, it can solve the problem of compatibility of older software and/or commercial software needing very particular requirements. Containerization is also the technology chosen for the services that run on the cluster. Before the cluster renovation, there were containers of various kind run by the research groups at DIBAF scattered on many nodes, making sometimes difficult to control and manage them correctly. Also, there were services not installed through docker. Activities were started on the cluster with those objectives:

- Re-organizing the existing services on a selected group of machines;
- Providing a easy way to deploy containerized applications;
- Providing High Availability;
- Simplifying the management of all the containers deployed on the machines.

Kubernetes and Rancher were chosen to achieve these goals. Kubernetes [19] is an open source software for container management, simplifying their deployment and maintenance on groups of multiple nodes, called Kubernetes clusters. In particular, it is able to:

- Load balancing: when a cluster of nodes is available, kubernetes is able to allocate the different replicas of the pods (a group of one or more containers) on each node, based on the available resources. The use of different nodes also guarantees high availabilty for the services running;

- Rollback and rollouts: upgrades and changes to the containers can be performed seamlessly without disturbances for the users by leveraging the multiple replicas on the cluster nodes;

- Self-healing: Kubernetes takes care by himself of failing containers, by restarting or removing them, depending on the strategy chosen when the service was deployed;

- Service discovery and traffic balancing: easy to expose services to the outside, capable to distribute traffic to all the replicas available in case of high request;

So Kubernetes satisfies almost all the objectives set for the DIBAF cluster, but a downside is that it can be a little difficult for new users to navigate and use correctly through command line, also because a service can be composed by a lot of different parts that have to communicate correctly in order to run. Also the Kubernetes cluster setup can have some difficulties when trying to incorporate elements of HPC systems, such as Infiniband and GPUs. Rancher helps with those tasks. Rancher [20] is a tool for the creation and management of Kubernetes clusters, also providing an easier way to run

containerized workloads for users new to this kind of deployment method. Rancher is able to install various Kubernetes distributions, such as RKE1, RKE2, k3s, on different kind of servers, both on cloud (EKS, GKS, AKS) or on-premise, through a graphical web GUI. Also, there is the possibility to import already existing Kubernetes clusters in order to orchestrate them all from a single point of management. Besides that, the Rancher GUI makes easier to new users to deploy their cointanerized services, guiding them in setting up all the different components, while still letting more experienced users have the possibilty to modify directly the configuration files through YAML (Yet Another Markup Language) files to have full control of their pods (the smallest Kubernetes unit). In the DIBAF cluster, Rancher was installed on a RKE2 cluster on 3 virtual machines. This first cluster is used only for Rancher management pods and it is not recommended to launch other deployments on it, so there is no issue in using virtual machines instead of bare-metal. From the GUI, me and prof. Sanna deployed another RKE2 cluster on two HPE DL380 gen10+ machines. Later, 2 other HPE DL380 gen10 nodes with 3 Nvidia T4 GPUs were added to the cluster through a simple operation on Rancher. Usually, a Kubernetes cluster deployed by Rancher uses ethernet networks to communicate internally between its component and externally with other machines and services. In the installation we performed at DIBAF cluster, the IB connections between the nodes was exploited in order to take advantage of the greater bandwidth in respect

to the standard ethernet network. When deploying the Kubernetes cluster nodes through Rancher, two different IP subnets can be defined as internal and external networks to be used for cluster and pod communications. In the DIBAF cluster case the IB network was set up as internal to guarantee a higher bandwidth for cluster communication and the docker images' management, while for the external network it was used an ethernet subnet to access the pods from outside the Kubernetes cluster and consequently from outside the DIBAF cluster. I installed Nvidia GPU operator to permit the use of GPUs available to the pods which can benefit from them. The operator can be installed with an Helm Chart directly from the App section in the Rancher GUI. After the installation, the number of GPUs available to any pods can be defined at any time from the Rancher GUI. In most cases there is the need for persistent storage for pods. Rancher supports different storage classes , such as cloud or host directories, in which volumes can be created. Among those alternatives, Longhorn [21] was chosen as a distributed, highly resilient, Kubernetes storage service for the DIBAF cluster. Longhorn volumes are created with multiple replicas on the nodes of the cluster eliminating the single point of failure in the case of issues with the storage. Through its GUI, it is also easy to set a backup to a secondary storage, in our case NFS, for disaster recovery and data retention. In the case of DIBAF cluster, the two nodes have a local storage providing 22 TB of space to Longhorn. Rancher also helps in the organization and access control

of the Kubernetes cluster. Active directories or other authentication/RBAC tools can be linked to assign roles to the users and control their permissions on the cluster objects. Also, projects can be created to organize the different applications and create access rules to them in order to isolate different user groups and their services, so that they only have the possibility to see and manage their own deployments. To illustrate the capabilities of Rancher, I will illustrate the deployment of two applications: the first for the storage and management of Big Data produced by IoT systems and the second for GPU machine learning, used for the PhD project.

In many cases when dealing with on-field measurements, there is the need to gather the data produced by many sensors in real-time and store it in a database capable of managing a costant flow of entries, but also lightweight to not put stress on the machines that host it. It is also convenient to have a way to visualize the target variables stored through various dashboard easily available to the end users. The solution implemented in the Rancher is composed by three main components: Mosquitto, InfluxDB and Grafana. Mosquitto is a message broker implementing the Message Queueing Telemetry Transport (MQTT) protocol, a communication standard requiring minimal resources in computation power, electrical power and network making it ideal for small sensors. It utilizes a publish and subscribe model, in which the units on the field send ("publish") their measurements together with a timestamp (a "topic") on the network, while a broker on a connected machine that receive

the messages from the different sensors and organizes them, also handling eventual issues with missing messages. The broker is also responsible to send the data to client machine that "subscribe" to receive the topic they need. The messages have a set expiration time during which are stored in the broker, so there is the need of a database to save them for later use. InfluxDB is a time series database making it very suitable for IoT in which the data is usually catalogued in buckets formed by usually a small number of columns that are divided in tags containing metadata to explore the samples and fields containing the measurements, associated to the relative timestamp. InfluxDB is more fast of a classic relational databases because it utilizes timestamps as index which is simpler than the ones usually utilized in other schemes. Now that the data is stored in InfluxDB, users can retrieve it and use it for further analysis and/or visualization. Grafana is a tool that helps end users to navigate and visualize the data by different dashboards easily deployable through many, already available templates. Different data sources can be integrate on Grafana, InfluxDB among them. Users can also interact in real time with the dashboards by selecting the desired timeseries and the time period to be visualized. These graphs can also be exported or download for any further use. An alerting system is also implemented to monitor if any measured variable trespass a determined threshold. These applications were deployed via yaml in the Kubernetes cluster on DIBAF through Rancher for different purposes, such as connectiong IoT devices to DIBAF datacenter,

mainly applied to monitoring chemical measurements in Bolsena lake and Structural Health Monitoring of several buildings. They were also set up to use the shared Longhorn storage to have replicas of the databases in order to not only have high availability of the services, but also a failover copy in case of damages to the filesystems' volumes. Then Rancher made it easier to make it available to end users by setting up node ports used to access the different WebUI of the applications. This type of environment was set up to gather IoT data from sensors deployed in the Bolsena Lake, that measured physical and chemical parameters of the water. Another deployment of these applications had the purpose to receive the measurements of biaxial inclinometers to evaluate the thermal deformation of buildings, monitoring the structural health of buildings. I deployed a second example application in order to have an environment in which to create and do preliminary testing for ML. For this purpose, I launched a Nvidia Rapids Jupyter notebook, using the docker image available on their official repository. This image is already equipped with all the Nvidia drivers needed and, together with the gpu operator installed on the cluster, it makes possibile to use the Nvidia T4 GPUs available on the cluster nodes. The number of GPUs visible by the container can be set through Rancher webUI, making it possible to launch multiple container without the risk of conflicting use. The notebook includes also RAPIDS library for ML such as cuDF and cuML, together with Dask for parallelization and Panel for visualization. PyTorch and Tensorflow can also

be installed to be used as framework to develop Deep Learning models and train them using the GPU if desired. With a notebook similar to this, I did some practice and preliminary studies to learn the basics of computational chemistry, propedeutic for the activities done in the first period of the PhD.

Chapter 2

HPC in Computational Chemistry: analysis of the interaction between Borazine and Noble Gases

Since the invention of the first calculators, it was investigated the opportunity to use these new tools in order to study problems not solvable by an exact solution, thus needing numerical methods. Quantum Mechanics (QM), basis of modern physics and chemistry, offers many examples of models described by very complex systems of equations, that need numerically accurate solutions. This requirement led to the development of a multitude of methods and algorithms capable to obtain solutions with the desired accuracy, taking advantage of the computational resources available during the last 60 years. For example, the task to solve the Schrödinger equation for atomic and molecular systems gave birth to different computational techniques such as the Hartree-Fock (HF) method, the Kohn-Sham method or Configura-

tion Interaction (CI) method [22–24]. All these techniques are based on the equivalence of finding the solution of the Schrödinger equation to a problem of minimization of functionals related to the Hamiltonian of the system under study. These concepts are relevant also in experimental and computational chemistry, where QM has a role when considering phenomenons at atomic and/or molecular scale. So the increase in computational power and the development of new algorithms and methods, such as ML, had an huge impact on chemistry in the last years [25]. One of these applications is in molecular modeling, where neural networks can be used to reproduce structural transitions at a fraction of the cost required by standard simulation techniques. For example, when trying to model bio-molecules at an atomic level, interactions between the electrons and the nucleus have to be taken in account; thus, it must be considered a Schrödinger equation associated to an Hamiltonian composed by the kinetic energies of the different atoms particles and by other terms due to the electrostatic interactions of the particles composing the system. Using Born-Oppenheimer approximation, the Schrödinger equation can be reduced to two separate equations depending from the Potential Energy Surface (PES). The solutions can be found by determining the global point of minima of the PES, thus it is a minimization problem with various parameters based on the gradient and curvature of a function. This scenario is similar to what happens in ML models where the aim is to minimize the loss metric (a function) in regard to the model weights (the

parameters). So, it can be useful to analyse optimization algorithms already developed and well implemented in Computational Chemistry to then apply the knowledge acquired to ML, with the scope of obtaining improvement in the numerical precision and performance. There are different categories of optimal path algorithms, such as guided algorithms, for example Gradient Descent or Newton's Method, or stochastic algorithms, for example the "genetic" approach which, starting from a set of initial random configuration, new ones are iteratively generated until an optimal one is reached that minimizes the target function [26, 27]. There are also methods that calculates approximately the Hessian of the atomic or molecular structure to study the transition states (i.e saddle points) that can appear in the PES. Examples of this kind of algorithms are the Berny geometry optimization algorithm, the Nudged Elastic Band (NEB) method and the Intrinsic Reaction Coordinates (IRC) [23]. The success of these methods in a field in which high precision is fundamental offers interesting ideas to find applications in the optimization problems that are encountered during ML processes and the possibility to obtain results with higher numerical precision in an efficient way.

To enter and explore the world of minimization in Computational Chemistry, a research activity was started for the analysis of complexes of noble-gas atoms with Borazine (Bz) [8]. Sheets of hexagonal boron nitride (hBN) plays a significant role in many fields, for example it is used in aerospace, metallurgy, electronics, engineering materials and biomedical fields [28]. The

adsorption of Noble Gases (Ng) on hBN has already been studied in different papers [29–35], identifying two site-specific adsorption mechanisms, a van der Waals type and a polarization type. Recently it was investigated how different Ng (He, Ne, Ar, Kr) interacts with different hBN structures such as $B_{12}N_{12}H_{12}$, $B_{27}N_{27}H_{18}$, $B_{48}N_{48}H_{24}$ using B3LYP [36, 37] density functional calculations. While the Ng atom is situated on a perpendicular line to the hBN sheet, three binding sites were individuated: the hollow site (H), the position above the B atom (Tb) and the one above the N atom (Tn). For any Ng atom, the H site was the most table one, followed by Tb and Tn in decreasing order and for all structures the greatest interaction energy contribution was the charge transfer, while polarization and dispersion have an equal role but minor to the former [38]. Following these findings, it is interesting to study what happens with Bz, the simplest hBN structure. In particular, the interaction between Bz and different Ng atoms, such as Ne, Ar, Kr, were investigated recently [39] at the Møller-Plesset truncated at second order (MP2) theory level [40], but focusing only on the H site, founding an increase in stability in such order: NeBZ < ArBz < KrBz. So, the interactions of different noble gas atom in the three different binding sites were not explored. So, me and Prof. Sanna, Prof. Grandinetti, Prof. Borocci, Dr. Zazza and dr. Rutigliano started a research activity to explore these cases. Differently from previous studies, it was observed that dispersion had a prevailing role, while polarization and charge transfer gave a minor con-

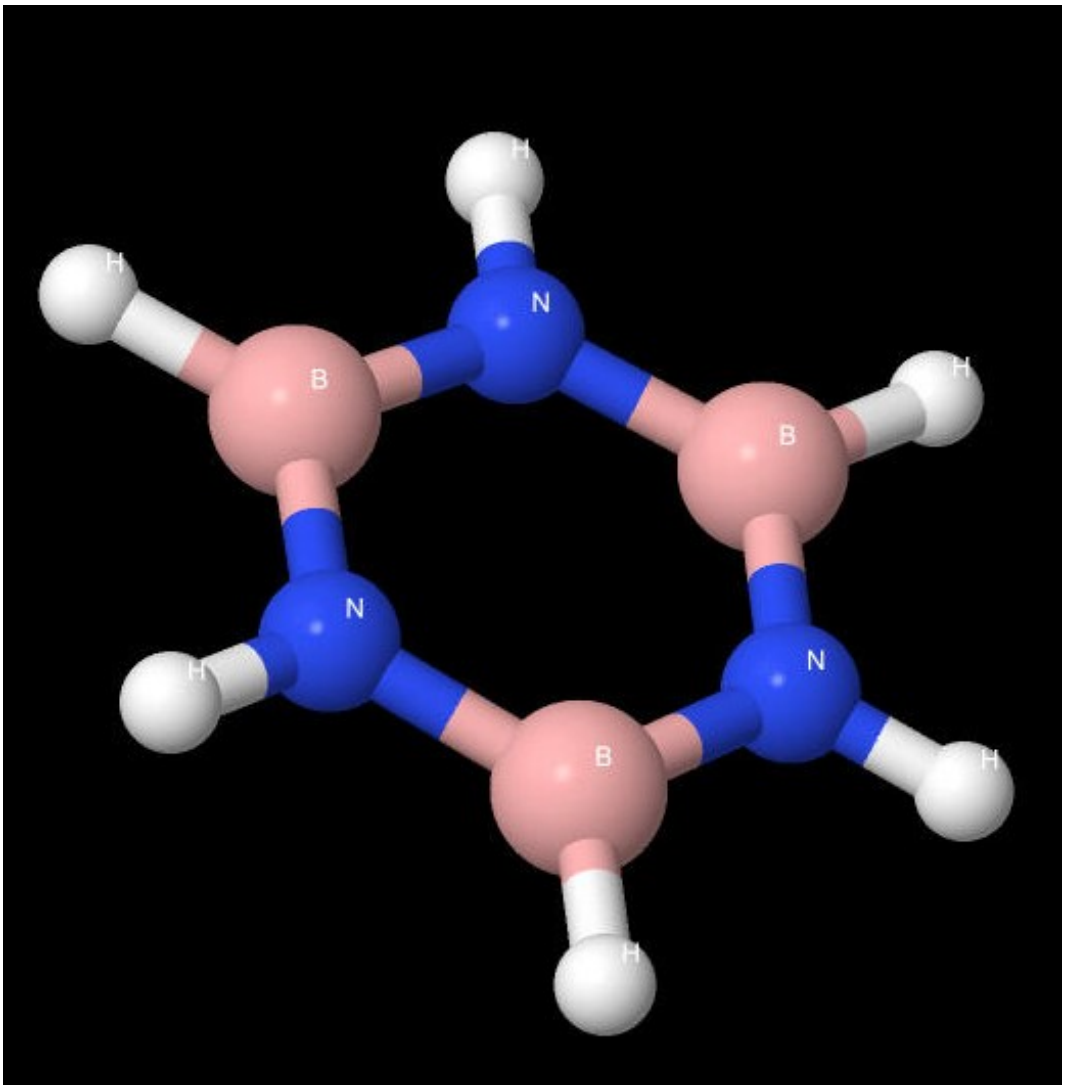

Figure 2.1: Borazine ($B_3N_3H_6$, visualized with Jmol)

tribution to stability. So, the scope of the collaboration is to study further the interaction of Bz with Ng atoms, using Symmetry-Adapted Perturbation Theory [41] and the composite method for bonding analysis described in [42–44]. The activity started with geometry optimizations for the complexes of Bz with different Ng atoms (He, Ne, Ar, Kr, Xe) with three different initial positions for each one. The 3 cases were identified in the following way:

- N-tri: the atom is on the line orthogonal to the plane of the Bz passing through the center of the ring (identifiable with the H site previously nominated);
- N-di: the atom is on the line orthogonal to the plane of the Bz passing through the middle point between 2 N atoms;
- N-mono: the atom is on the line orthogonal to the plane of the Bz passing through the middle point between 2 B atoms;

The calculations are performed with Gaussian09 (G09, Version D1) [45] employing first the MP2 level of theory and then coupled cluster with inclusion of single and double substitutions, CCSD, and an estimate of connected triples, CCSD(T) [46], using as basis sets the Dunning's correlation consistent aug-cc-pVDZ and aug-cc-pVTZ [47] (denoted with aVnZ from here on). For the Xe atom, the aVnZ-PP basis set was used with the Stuttgart/Cologne small-core, scalar-relativistic effective core potential, ECP28.

Aug-cc-pVDZ/MP2	Hartrees (a.u.)	eV	Kcal/mol
BN	-241.95076	-6583.818669	-151826.42657
He	-2.8826672	-78.441407	-1808.901365
Ne	-128.703223	-3502.194754	-80762.509017
Ar	-526.955299	-14339.190896	-330669.513125
Kr	-2752.122267	-74889.097104	-1726983.165017
Xe	-328.416746	-8936.679114	-206084.663553

Table 2.1: Energy values at MP2 level and Aug-cc-pVDZ basis for the Bz and the Ng atoms

Aug-cc-pVDZ/CCSD(T)	Hartrees (a.u.)	eV	Kcal/mol
BN	-242.025906	-6585.863497	-151873.58141
He	-2.889549	-78.628671	-1813.21976
Ne	-128.709295	-3502.359981	-80766.319255
Ar	-526.969684	-14339.582332	-330678.539851
Kr	-2752.133582	-74889.405001	-1726990.265289
Xe	-328.426408	-8936.94203	-206090.726551

Table 2.2: Energy values at CCSD(T) level and Aug-cc-pVDZ basis for the Bz and the Ng atoms

First, the optimal geometry for the Bz molecule was obtained and the energy was calculated, also for each of the Ng atoms alone. The values are reported in tables 2.1, 2.2, 2.3, 2.4.

The distance R between the Ng atom and the plane in which the BZ lies was chosen as the only free parameter to be optimized. As initial point, it was used the the optimized Bz geometry obtained before and the starting volumes for R reported in table 2.5.

The Gaussian jobs to find the optimal geometries of the complexes were launched with the following options:

Aug-cc-pVTZ/MP2	Hartrees (a.u.)	eV	Kcal/mol
BN	-242.166752	-6589.696113	-151961.963626
He	-2.894804	-78.771667	-1816.517323
Ne	-128.805792	-3504.9858	-80826.87205
Ar	-527.024283	-14341.068047	-330712.801248
Kr	-2752.294114	-74893.773301	-1727091.000661
Xe	-328.521141	-8939.519848	-206150.172419

Table 2.3: Energy values at MP2 level and Aug-cc-pVTZ basis for the Bz and the Ng atoms

Aug-cc-pVTZ/CCSD(T)	Hartrees (a.u.)	eV	Kcal/mol
BN	-242.242662	-6591.761731	-152009.59788
He	-2.9005979	-78.929327	-1820.153051
Ne	-128.812648	-3505.172361	-80831.174256
Ar	-527.048758	-14341.734046	-330728.15955
Kr	-2752.31168	-74894.251297	-1727102.02349
Xe	-328.53658	-8939.939965	-206159.86054

Table 2.4: Energy values at CCSD(T) level and Aug-cc-pVTZ basis for the Bz and the Ng atoms

	N-Mono	N-Di	N-Tri
He	1.5 Å	1.5 Å	1.5 Å
Ne	3.3401 Å	3.3143 Å	1.5 Å
Ar	3.3401 Å	3.3143 Å	1.5 Å
Kr	3.5198 Å	3.3143 Å	1.5 Å
Xe	3.5198 Å	3.3143 Å	1.5 Å

Table 2.5: Starting values for the distance R between the Ng atom and the Bz plane

- MP2: the optimization is performed at MP2 level of theory;
- Aug-cc-pVTZ: the Dunning's correlation consistent is used as basis set;
- Opt = (Tight, Zmat): Optimization specifications, Tight specifies that stricter threshold values must be used for the convergence, while Zmat indicates that the input is given using internal coordinates;
- pop = full: all orbitals are printed in the output;
- #P: output files is populated with additional information on the machine used and on the convergence in the SCF;

They were launched using 24 cores and 64 GB of RAM. The results obtained are reported in tables are reported in tables 2.6, 2.7, 2.8.

N-mono	He	Ne	Ar	Kr	Xe
R	3.41 Å	3.34 Å	3.51 Å	3.52 Å	3.64 Å
Distance Ng-B	3.64 Å	3.57 Å	3.73 Å	3.74 Å	3.85 Å
Angolo Ng-B-B	69.8°	69.3°	70.3°	70.3°	70.9°
Angle diedrale Ng-B-B-B	89.6°	89.6°	89.6°	89.6°	90.3°

Table 2.6: Distance and angles of the optimized geometries for N-mono found at MP2 level of theory and Aug-cc-pVTZ basis.

The energy values at the optimal geometries calculated at different level of theory and with different basis set are reported in tables 2.9 - 2.18.

N-di	He	Ne	Ar	Kr	Xe
R	3.38 Å	3.31 Å	3.49 Å	3.50 Å	3.62 Å
Distance Ng-N	3.6 Å	3.53 Å	3.69 Å	3.7 Å	3.82 Å
Angle Ng-N-N	70.2°	69.8°	70.7°	70.8°	71.4°
Angle diedrale Ng-N-N-N	90.4°	90.4°	90.4°	90.4°	89.6°

Table 2.7: Distance and angles of the optimized geometries for N-di found at MP2 level of theory and Aug-cc-pVTZ basis.

N-tri	He	Ne	Ar	Kr	Xe
R	3.34 Å	3.34 Å	3.51 Å	3.53 Å	3.67 Å
Distance Ng-B	3.57 Å	3.57 Å	3.74 Å	3.76 Å	3.88 Å
Distance Ng-N	3.55 Å	3.55 Å	3.72 Å	3.74 Å	3.86 Å
Angle Ng-B-N	66.0Â°	66.0 Â°	67.1Â°	67.2Â°	68.0Â°
Angle diedrale Ng-B-N-B	90.0Â°	90.0Â°	90.0Â°	90.0Â°	90.0Â°

Table 2.8: Distance and angles of the optimized geometries for N-tri found at MP2 level of theory and Aug-cc-pVTZ basis.

Then the stability of these 3 positions is investigated by calculating the related frequencies by using the optimized geometries and the freq options in Gaussian. In all cases, except for HeBz N-di geometry, a true minimum is present on the potential energy surface (PES) as shown in table 2.19. In fact, only for the HeBz N-di case there is a negative frequency value indicating the presence of a saddle point in the PES, while for all the other geometries there are only positive values confirming them as points of minimum for the energy.

In terms of stability, energy values show that N-tri is the most stable betwee the isomers, precisely in this order: N-mono $<$ N-di $<$ N-tri. This situation parallels the results found in previous research on larger Bz clus-

He		Aug-cc-pVTZ		
MP2		Hartrees (a.u.)	eV	Kcal/mol
	E_{N-tri}	-245.061897	-6668.477059	-153778.69493
	E_{N-mono}	-245.061828	-6668.475192	-153778.65188
	E_{N-di}	-245.061842	-6668.475565	-153778.66048
	E_{N-di} - E_{N-mono}	-1.4e-5	-3.7e-4	-8.6e-3
CCSD(T)		Hartrees (a.u.)	eV	Kcal/mol
	E_{N-tri}	-245.1436577	-6670.701882	-153830.00056
	E_{N-mono}	-245.1435782	-6670.699719	-153829.95067
	E_{N-di}	-245.1435933	-6670.70013	-153829.96014
	E_{N-di} - E_{N-mono}	-1.5e-5	-4.1e-4	-9.5e-3

Table 2.9: Energy values for the HeBz structure in the three configurations using Aug-cc-pVTZ basis. The difference between the energy at N-di and N-mono is also reported.

He		Aug-cc-pVDZ		
MP2		Hartrees (a.u.)	eV	Kcal/mol
	E_{N-tri}	-244.833928	-6662.273704	-153635.64219
	E_{N-mono}	-244.833819	-6662.270738	-153635.57379
	E_{N-di}	-244.833834	-6662.271148	-153635.58321
	E_{N-di} - E_{N-mono}	-1.5e-5	-4.1e-4	-9.4e-2
CCSD(T)		Hartrees (a.u.)	eV	Kcal/mol
	E_{N-tri}	-244.916003	-6664.507079	-153687.14504
	E_{N-mono}	-244.915885	-6664.503868	-153687.071
	E_{N-di}	-244.915902	-6664.504331	-153687.08166
	E_{N-di} - E_{N-mono}	-1.7e-5	-4.6e-4	-1.1e-2

Table 2.10: Energy values for the HeBz structure in the three configurations using Aug-cc-pVDZ basis. The difference between the energy at N-di and N-mono is also reported.

Ne		Aug-cc-pVTZ		
MP2		Hartrees (a.u.)	eV	Kcal/mol
	EN-tri	-370.97339	-10094.70493	-232789.36655
	E_{N-mono}	-370.973293	-10094.69673	-232789.30568
	E_{N-di}	-370.973315	-10094.69732	-232789.31949
	E_{N-di} - E_{N-mono}	-2.2e-5	-5.9e-4	-1.4e-2
CCSD(T)		Hartrees (a.u.)	eV	Kcal/mol
	E_{N-tri}	-371.056298	-10096.96098	-232841.39212
	E_{N-mono}	-371.056189	-10096.95801	-232841.32372
	E_{N-di}	-371.056214	-10096.95869	-232841.3394
	E_{N-di} - E_{N-mono}	-2.5e-5	-6.8e-4	-1.6e-2

Table 2.11: Energy values for the NeBz structure in the three configurations using Aug-cc-pVTZ basis. The difference between the energy at N-di and N-mono is also reported.

Ne		Aug-cc-pVDZ		
MP2		Hartrees (a.u.)	eV	Kcal/mol
	E_{N-tri}	-370.65493	-10086.039192	-232589.52984
	E_{N-mono}	-370.654805	-10086.03579	-232589.4514
	E_{N-di}	-370.654829	-10086.036443	-232589.46646
	E_{N-di} - E_{N-mono}	-2.4e-5	-6.5e-4	-1.5e-2
CCSD(T)		Hartrees (a.u.)	eV	Kcal/mol
	E_{N-tri}	-370.736336	-10088.254363	-232640.61289
	E_{N-mono}	-370.736194	-10088.250499	-232640.52378
	E_{N-di}	-370.736222	-10088.251261	-232640.54135
	E_{N-di} - E_{N-mono}	-2.8e-4	-7.6e-4	-1.7e-2

Table 2.12: Energy values for the HeBz structure in the three configurations using Aug-cc-pVDZ basis. The difference between the energy at N-di and N-mono is also reported.

Ar		Aug-cc-pVTZ		
MP2		Hartrees (a.u.)	eV	Kcal/mol
	E_{N-tri}	-769.193314	-20930.82618	-482676.19497
	E_{N-mono}	-769.193077	-20930.81973	-482676.04625
	E_{N-di}	-769.193121	-20930.82092	-482676.07386
	E_{N-di} - E_{N-mono}	-4.4e-5	-1.2e-3	-2.8e-2
CCSD(T)		Hartrees (a.u.)	eV	Kcal/mol
	E_{N-tri}	-769.293465	-20933.551424	-482739.04068
	E_{N-mono}	-769.293269	-20933.546091	-482738.91769
	E_{N-di}	-769.293305	-20933.54707	-482738.94028
	E_{N-di} - E_{N-mono}	-3.6e-5	-9.8e-4	-2.4e-2

Table 2.13: Energy values for the ArBz structure in the three configurations using Aug-cc-pVTZ basis. The difference between the energy at N-di and N-mono is also reported.

Ar		Aug-cc-pVDZ		
MP2		Hartrees (a.u.)	eV	Kcal/mol
	E_{N-tri}	-768.90864	-20923.079798	-482497.5593
	E_{N-mono}	-768.908332	-20923.071417	-482497.36603
	E_{N-di}	-768.908413	-20923.073621	-482497.41685
	E_{N-di} - E_{N-mono}	-8.1e-5	-2.2e-3	-5.1e-2
CCSD(T)		Hartrees (a.u.)	eV	Kcal/mol
	E_{N-tri}	-768.998074	-20925.513422	-482553.67999
	E_{N-mono}	-768.997785	-20925.505558	-482553.49864
	E_{N-di}	-768.997867	-20925.507789	-482553.5501
	E_{N-di} - E_{N-mono}	-8.2e-5	-2.2e-3	-5.2e-2

Table 2.14: Energy values for the ArBz structure in the three configurations using Aug-cc-pVDZ basis. The difference between the energy at N-di and N-mono is also reported.

Kr		Aug-cc-pVTZ		
MP2		Hartrees (a.u.)	eV	Kcal/mol
	E_{N-tri}	-2994.464937	-81483.580192	-1879055.51888
	E_{N-mono}	-2994.464595	-81483.570886	-1879055.30427
	E_{N-di}	-2994.464661	-81483.572682	-1879055.34569
	E_{N-di} - E_{N-mono}	-6.6e-5	-1.8e-3	-4.1e-2
CCSD(T)		Hartrees (a.u.)	eV	Kcal/mol
	E_{N-tri}	-2994.557932	-81486.110716	-1879113.87413
	E_{N-mono}	-2994.557657	-81486.103233	-1879113.70157
	E_{N-di}	-2994.557709	-81486.104648	-1879113.7342
	E_{N-di} - E_{N-mono}	-5.2e-5	-1.4e-3	-3.3e-2

Table 2.15: Energy values for the KrBz structure in the three configurations using Aug-cc-pVTZ basis. The difference between the energy at N-di and N-mono is also reported.

Kr		Aug-cc-pVDZ		
MP2		Hartrees (a.u.)	eV	Kcal/mol
	E_{N-tri}	-2994.077235	-81473.030278	-1878812.23215
	E_{N-mono}	-2994.076827	-81473.019176	-1878811.97612
	E_{N-di}	-2994.076961	-81473.022823	-1878812.06021
	E_{N-di} - E_{N-mono}	-1.3e-4	-3.5e-3	-8.2e-2
CCSD(T)		Hartrees (a.u.)	eV	Kcal/mol
	E_{N-tri}	-2994.163469	-81475.376826	-1878866.34481
	E_{N-mono}	-2994.163095	-81475.366649	-1878866.11012
	E_{N-di}	-2994.163228	-81475.370268	-1878866.19358
	E_{N-di} - E_{N-mono}	-1.3e-4	-3.6e-3	-8.3e-2

Table 2.16: Energy values for the KrBz structure in the three configurations using Aug-cc-pVDZ basis. The difference between the energy at N-di and N-mono is also reported.

Xe		Aug-cc-pVTZ		
MP2		Hartrees (a.u.)	eV	Kcal/mol
	E_{N-tri}	-570.693098	-15529.357596	-358115.40223
	E_{N-mono}	-570.692705	-15529.346902	-358115.15562
	E_{N-di}	-570.692779	-15529.348916	-358115.20207
	E_{N-di} - E_{N-mono}	-7.4e-5	-2.0e-3	-4.6e-2
CCSD(T)		Hartrees (a.u.)	eV	Kcal/mol
	E_{N-tri}	-570.783688	-15531.822677	-358172.24833
	E_{N-mono}	-570.783382	-15531.81435	-358172.05631
	E_{N-di}	-570.783437	-15531.815847	-358172.09082
	E_{N-di} - E_{N-mono}	-5.5e-5	-1.5e-3	-3.5e-2

Table 2.17: Energy values for the XeBz structure in the three configurations using Aug-cc-pVTZ basis. The difference between the energy at N-di and N-mono is also reported.

Xe		Aug-cc-pVDZ		
MP2		Hartrees (a.u.)	eV	Kcal/mol
	E_{N-tri}	-570.372285	-15520.627826	-357914.08899
	E_{N-mono}	-570.371901	-15520.617377	-357913.84803
	E_{N-di}	-570.372018	-15520.62056	-357913.92145
	E_{N-di} - E_{N-mono}	-1.2e-4	-3.2e-3	-7.3e-2
CCSD(T)		Hartrees (a.u.)	eV	Kcal/mol
	E_{N-tri}	-570.456902	-15522.930373	-357967.18697
	E_{N-mono}	-570.456543	-15522.920604	-357966.9617
	E_{N-di}	-570.456669	-15522.924035	-357967.04083
	E_{N-di} - E_{N-mono}	-1.3e-4	-3.4e-3	-7.9e-2

Table 2.18: Energy values for the XeBz structure in the three configurations using Aug-cc-pVDZ basis. The difference between the energy at N-di and N-mono is also reported.

		1	2	3
		A'	A"	A'
He	N-mono	3.2	14.6	52.2
	N-di	-11.0	28.4	51.2
	N-tri	31.7	31.7	52.0
Ne	N-mono	14	14.8	42.9
	N-di	6.5	24.9	39.9
	N-tri	24.7	24.7	38.1
Ar	N-mono	16.5	22.3	56.1
	N-di	7.7	36.0	52.3
	N-tri	36.6	36.6	49.3
Kr	N-mono	19.8	26.1	60.4
	N-di	10.8	42.1	54.5
	N-tri	41.9	41.9	49.6
Xe	N-mono	22.5	28.7	62.7
	N-di	13.8	45.9	56.8
	N-tri	45.0	45.0	50.7

Table 2.19: Frequencies values at the optimized structures

ters with the sites Tb, Tn and H. Also, it is confirmed that stability increase with the periodic number of the Ng atom: HeBz < NeBz < ArBz < KrBz < XeBz, reflecting the analysis performed on the interaction between Ng and hBN. The structures obtained from the optimization performed with Gaussian were used for the SAPT analysis and the composite bonding analysis method. From SAPT analysis, a similar bonding character was found for all the isomers with the energy contributions for E_{elst}[1], E_{ind}[2], E_{disp}[3], E_{exch}[4], as reported in table 2.20.

[1]Electrostatic energy
[2]Induction energy
[3]Dispersive energy
[4]Exchange energy

		E_{elst}	E_{ind}	E_{disp}	E_{exch}	E_{int}^{SAPT}
HeBz	N-mono	-0.043 (12.29)	-0.017 (4.86)	-0.29 (82.86)	0.21	-0.14
	N-di	-0.044 (12.19)	-0.017 (4.71)	-0.30 (83.10)	0.22	-0.14
	N-tri	-0.059 (12.80)	-0.022 (4.77)	-0.38 (82.43)	0.29	-0.17
NeBz	N-mono	-0.22 (23.73)	-0.017 (1.83)	-0.69 (74.43)	0.65	-0.28
	N-di	-0.22 (23.21)	-0.018 (1.90)	-0.71 (74.89)	0.67	-0.28
	N-tri	-0.23 (21.74)	-0.018 (1.70)	-0.81 (76.56)	0.74	-0.32
ArBz	N-mono	-0.65 (24.23)	-0.083 (3.09)	-1.95 (72.68)	1.89	-0.79
	N-di	-0.66 (24.02)	-0.088 (3.20)	-2.00 (72.78)	1.96	-0.79
	N-tri	-0.72 (23.74)	-0.093 (3.07)	-2.22 (73.19)	2.17	-0.86
KrBz	N-mono	-1.18 (28.37)	-0.16 (3.85)	-2.82 (67.79)	3.30	-0.86
	N-di	-1.22 (28.37)	-0.17 (3.95)	-2.91 (67.67)	3.44	-0.86
	N-tri	-1.31 (28.17)	-0.17 (3.66)	-3.17 (68.17)	3.73	-0.92

Table 2.20: SAPT analysis of the NgBz complexes performed with the aVDZ/mbf basis set at the MP2/aVTZ optimized geometries. The values in parentheses are the percentage contributions of E_{elst}, E_{ind}, and E_{disp} to (E_{elst} + E_{ind} + E_{ind}). E_{int}^{SAPT} is the sum of the 4 energy terms. All values in kcal/mol.

Contrary to what reported in [38], E_{disp} is the dominating term, with a contribution arriving at 80% for some of the HeBZ isomers, decreasing for heavier Ng atoms. The opposite happens for E_{elst} which passes from 10% for He to 30% for Kr, while E_{ind} has a minor role in the energy contribution. These findings match with the electronic structure of Borazine. As confirmed by [48], the Bz presents a electronic delocalization within the ring involving the N atoms. So the Ng atoms prefer to bind to those atoms with intense dispersive contacts, due to them being electron-rich. The dispersive nature of the NgBz is also confirmed by the description given by the composite method of bonding analysis [42–44]. In conclusion, the methods used show

that noble gas atoms and borazine form non-covalent bonds involving only the nitrogen atoms with a prevalent dispersive component, that is bigger for N-tri, followed by N-di and N-mono, ordered from the most to the less stable. The results obtained deepened the understanding of the interaction of Noble Gases atoms and Borazine, that it can be used as a starting step to study first the case of the behaviour of Ng atoms near Borazine sheets, then to extend the analysis to a real-world application such as ion thrusters used in space exploration [49].

This activity was propedeutic to understand and study the basis of Computational Chemistry with the purpose of applying DL algorithms to perform part of the operations done by Gaussian. Subsequently, the skill and knowledge obtained when trying to achieve this objective helped with the analysis of other datasets of interest of the department, as it will be described in the next 2 chapters.

Chapter 3

HPC and Deep Learning: Hyperparameters' optimization with HPE Machine Learning Development Edition

Deep Learning (DL) models are a subset of ML based on the use of Neural Networks (NN) to perform different tasks that in most cases can be defined as regression or classification. The most simple example of DL model is the single neuron, also called perceptron, the building unit of NN. It takes as input a vector of features and an associated weights vector representing their "importance" to compute a weighted sum. This result is then passed to an activation function that produces the output. The perceptron structure is schematised in 3.1.

From this simple unit, new versions of NNs were developed by modifying the numbers and the characteristic of the perceptron, such as the function applied to the input features or the activation function. A multi-layer neural

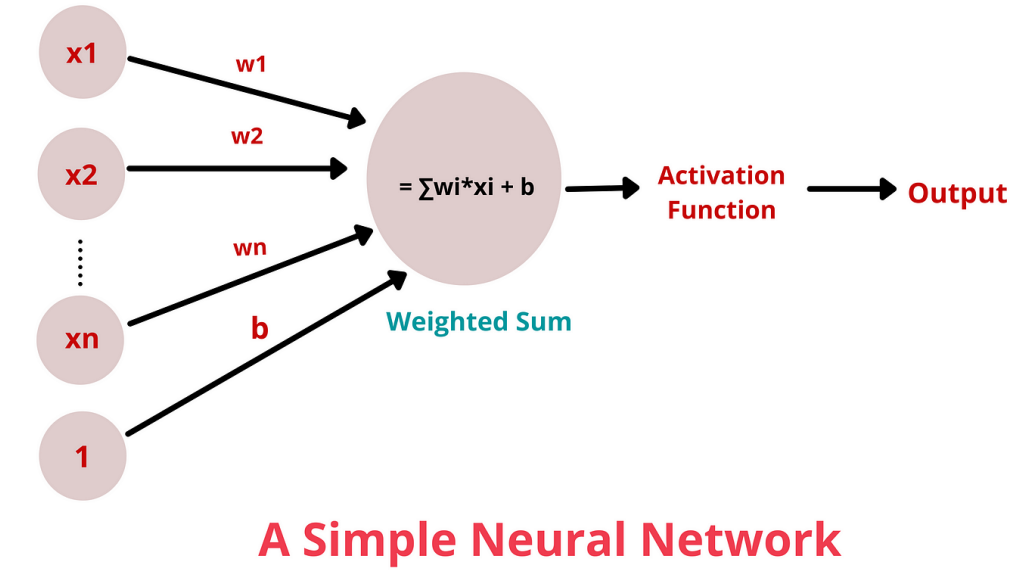

Figure 3.1: Scheme of a Perceptron

network is formed by a great number of neurons linked by connections. There are three kind of units:

- Input units: units that receive the data;
- Output units: units that produce the result of the NN;
- Hidden units: units between input and output;

The input units receive the features of the sample and produce a value to be transmitted at all the hidden units linked to them. In the same fashion, hidden units will pass the processed data between them and finally to the output units that give the desired results, eventually after some activation functions. All these units are characterized by the operation performed on the data received and the associated weights. In supervised ML models, data is provided with labels, i.e. the real values or classes of the quantity to be predicted by the algorithm. In order to be able to produce the desired output, the DL models needs to be trained to individuate an underlying connection between the input and the target variables. During the training phase, the weights are updated through an evaluation of the error between the predicted values and the label, using a loss metric as the target function to be optimized. This process is usually performed through back-propagation:

1. The predicted value is computed by the NN;
2. Comparison between the NN output and the label and estimate of the loss metric;

3. Assignation of a scaling factor dependent of the gradient of the error in order to correct the output for each neuron of the last layer;

4. Assignation of a new value to the weights;

5. Weights are updated also in the precedent layers taking account of the error calculated;

This process must be repeated multiple times to find a good weight configuration. Usually the training is measured in epochs, i.e. all the samples in the training set (often 70% of the data) are processed by the model. There are multiple choices for the optimizer responsible to update the weights, varying in how the gradient is incorporated in the scaling factor or even in the use of the second derivative in the formula. After the training, the model can be tested on the remaining 30% of the samples to evaluate the prediction quality, usually with the same loss metric. This is the most important indicator of the model quality and it is the main driver for guiding changes in the structure and/or the components of the model. In fact, as written above, there are many choices to make when creating the NN and choosing the optimizer and the loss metric, but there are also all the parameters characterizing the different components. Usually two types of parameters are defined when dealing with Machine Learning:

- Model parameters, that are initialized and then updated during the training process, i.e. weights;

- Hyperparameters, that are set from the start and define properties of the layers of the model and the optimizer;

The quality of a model prediction is not only determined by its structure and the optimizer, hyperparameters have also a huge impact on inferencing [58]. The testing of different possible values of all the hyperparameters to find the optimal one can be very time consuming (increasing exponentially for each new parameters added) because it involves the training and the validation of the model for each combination. It also raises the question on what algorithm to use to perform the exploration of the hyperparameters' space [59]. The naive approach is to select different values of hyperparameters in a grid or randomly and train the model with all the possible choices to find the one with the best validation performance. There are also new algorithms more efficient in the search of the optimal hyperparameters' combination implementing different strategies in the start and stop of the training of new models, taking advantage also of multiple nodes, if present. One of them is the Asynchronous Successive Halving Algorithm (ASHA) [60]. The Successive Halving Algorithm (SHA) starts multiple trials together, trains them all for a specific number of batches or epochs (called a rung) and then stops the ones with the worst validation errors, usually 75% of the starting trials, while the rest are trained further. The procedure is repeated for multiple rung until the trial with the best validation performance is chosen. The percentage of the discarded trials can be set through a parameter called divisor

(d). Each rung will keep 1/d trials that will will be trained for a number of epochs d times the amount of the previous rung. The standard algorithm wait for each rung to finish to begin promoting the best trials, resulting in nodes not being utilized for most of the time in case of a multiple instances execution. Asynchronous SHA resolves this issue by promoting the trials as soon as the minimum number needed is reached, while starting new trials on the nodes not utilized, keeping a 100% efficiency. The search of the optimal hyperparameters' combination can be very time-consuming and difficult to set up to use all the nodes in an HPC cluster efficiently. Hewlett Packard Enterprise (HPE) offers a new tool to perform easily hyperparameters's search and other operations with the Machine Learning Development Environment (MLDE) [51]. MLDE is a DL platform that has the aim to help developers to create, launch and manage models on HPC. It is composed by a master node accessible by WebUI, command line or scripts and by one or more agent nodes on which the Jupyter notebooks or ML workloads, called experiments in MLDE, are launched. The installation of these components can be performed in different ways, such as baremetal, docker or Kubernetes with the possibilities to also implement it on top of workload managers (such as SLURM) to hereditate the queues as resource pools with their specifics and user limitations. Depending on the agent nodes' specifics, MLDE can identify one or more slots, which are the units used by the tasks to run. Usually, resource pools are divided in GPU nodes (1 GPU per slot) and CPU only

nodes (a custom number of CPUs can be assigned to each slot). The resource pool and the number of slots can be defined when launching an experiment or a notebook. Besides being an intermediary between the users and HPC systems, MLDE helps also to manage and organize models when there are multiple users working on the same platform. In MLDE a project is a collection of experiments, while a workspace is a collection of projects. While the scope of projects is purely for the sake of organization, workspaces can be used to set up rules on resources and user access. A resource pool can be binded to a workspace so that all its experiments run on that pool, limiting the access to specific machines in a similar fashion to workload managers. Moreover users can be limited to selected workspaces by assigning them roles through local rules or leveraging existing authorization mechanisms such as OAuth 2.0 or OpenID. Another tool that MLDE provides is the model registry. For every trial in an experiment, different checkpoints are saved during the training, capturing the state of the model at a determinate batch. These checkpoints can be archived and organized in the model registry so they can be easily downloaded to other machines to continue training or perform inference on new data. All these features makes MLDE a really convenient environment for ML developers, but its real strength resides in how it is able to leverage the distributed nature of HPC and GPUs. To better understand what it can do, it is useful to describe how an experiment is created and launched on MLDE. The basis of every workflow are Docker images of ba-

sic environments with all the necessary packages and with GPU support if requested. Starting from these containers, an experiment is characterized by two main files: the trial file and the configuration file. The trial file is a Python script in which the model to be trained is defined with all the needed components. MLDE is compatible with most of the main ML python frameworks. This is very convenient because Python was chosen as programming language for this project, due to the many packages already available for data analysis and machine learning, PyTorch [57] being one of them. PyTorch is one of the main ML framework available that provides many of the components needed for build and train a model, such as different layers for NN, optimizers, loss metrics and model saving/loading. PyTorch is fully supported by MLDE. To train a model in MLDE, a user-defined trial class inherited from the determined package module pytorch.PyTorchTrial is used containing the following methods:

- **__init__**: this method's scope is to initialize different elements of the trial. It takes as single input a PyTorchTrialContext instance that provides informations about the trial. The hyperparameters' values are saved in an array through the get_hparam() method in order to be used in the script. The model, the optimizer and the scheduler if needed are defined with the hyperparameters needed and wrapped using the methods from PyTorchTrialContext;

- **build_training_data_loader**: in this method it is defined where to find the training data and how to batch it to be ready to be used as input for the model. Dataloader parameters, such as batch size and number of workers, can be also defined in the experiment configuration file;

- **build_validation_data_loader**: similar to the previous one, but for the validation data;

- **train_batch**: the training loop is defined in this method by choosing the desired loss metric and calling the model to perform the forward step;

- **evaluate_batch**: the method responsible for the validation and the computation of the loss metric or other performance measurement defined by the user;

In the trial script, a dataset class must be defined to use properly the dataloader method provided by the determined package. This class take the raw data and divide it into input and output features applying all the necessary transformations, then it returns a dictionary containing the samples with two keys, input and target. Then the dataset is given as an input parameter to the PyTorchTrial dataloader methods. It also useful to define the model with its forward step in the script outside the trial class in order to be able to pass parameters and/or hyperparameters also to the forward method, a

procedure not possible using the structure described in the PyTorch tutorials of the MLDE documentation. Other classes and methods needed for the model can be also defined in the trial python file. Besides the trial script, the experiment is defined also by its configuration specified in a yaml (Yet Another Markup Language) file. It is composed by different section, each containing different options to specify where and how the experiment must be conducted. After the first lines in which the name of the experiment and the workspace are defined for management purposes, sections are defined for each characteristic of the trial:

- **checkpoint_storage**: The location on in which to save the checkpoints. In order to be persistent, it must be on a shared storage accessible to every node in the MLDE cluster or on cloud instances, such as Amazon s3, Google Cloud Storage or Microsoft Azure;

- **bind_mounts**: directories accessible to every node to be mounted in the containers, useful to pass the training data and scripts needed for the model to run. In this section the path on the nodes and the mount point is defined;

- **data**: the path or url of the data to be used in the training phase and other specifics if needed, such as access credentials;

- **hyperparameters**: the list of the variables and their values to be tested during the experiment. The data type of the hyperparameters

must be declared and, depending on it, which values to give them in the trials. For all the types, the values can be defined exactly giving a list containing them. Alternatively, if the type is numeric (integer or double), a range can be defined giving a maximum and minimum threshold and the count of choices desired. Then, the algorithm will pick in the defined range casually. Here the global batch size is also defined defining how big the training data set will be for each training loop. It is very important when using GPU because different devices have different limits on how much data they can store and this also has an impact on the model performance in training and inferencing;

- **resources**: the number of slots to allocate for each trial and for the entire experiment and the resource pool to use. If MLDE is installed over workload managers, the resource pool corresponds to the job queues and it is able to enable parallelization on their nodes;

- **searcher**: the strategy and/or the algorithm used to search the hyperparameters' space defined by the previous section. The metric option define which quantity to consider and minimize, usually the validation loss defined in the model script. Also the max number of trials, how many of them can run concurrently and the max number of epochs or batch can be defined in this section. Three hyperparameters' search methods are already implemented in MLDE:

  - random: values are chosen casually among the possible ones and each trial is trained fully the desired number of epochs or batches defined in the max_length option in this section. It will keep running until the max number of trials is reached;
  - grid: values are chosen from a grid defined in the hyperparameters' section. So, all the possibile choices must be listed and not defined through a range. It will span all the possible combination, running as much trials are needed;
  - adaptive_asha: the search is performed by using multiple Asynchronous Successive Halving Algorithm (ASHA) instances. The MLDE adaptive implementation starts multiple ASHA instances and it can be decided how "aggressively" to stop the worst trials through the "mode" parameter. So "aggressive" mode will start more trials, but a smaller percentage will be completed, while "conservative" mode will start less trials with less rung, but many of them will complete the full training;

- **entrypoint:** the model to be trained. The name of the .py file containing the trial and the name of the trial itself must be specified. Moreover, additional commands can be added to be executed before or after the training, such as nvidia-smi to monitor the status of the GPU memory;

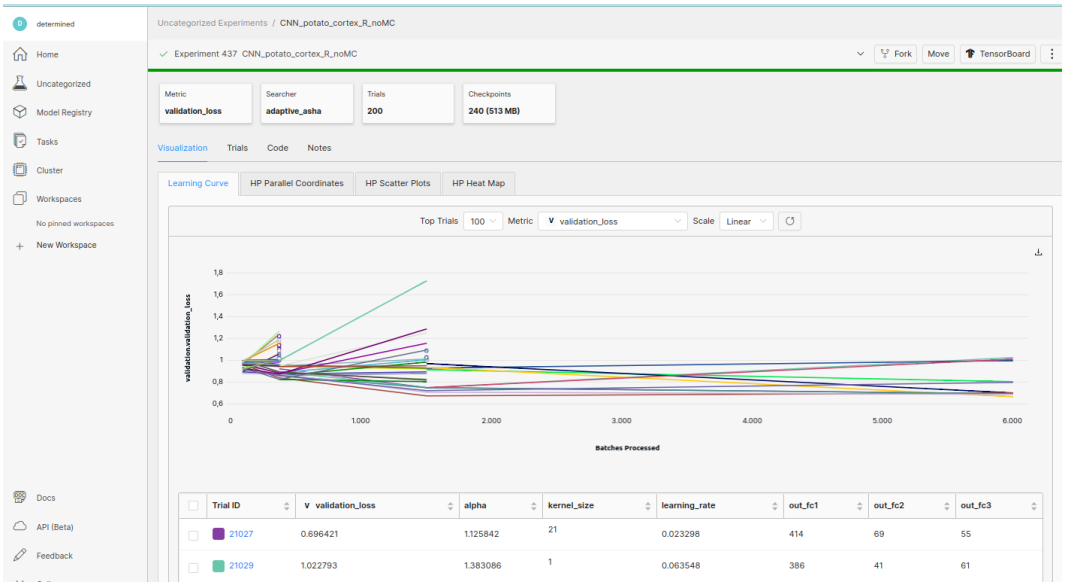

Figure 3.2: An example of experiment visualization in the MLDE WebUI

After creating these two files, the experiment can be launched using the CLI from a machine that can connect to the MLDE master node through the "det" command. The content of the folder in which the files are located are sent to MLDE and the experiment can be monitored through the WebUI, where also the logs of each trial are available. During the execution, graphs of the training and validation loss trend are plotted to see how the model performs with the different hyperparameters combinations.

When the experiment is finished, the different trials can be ordered by the value of the validation error to find the best performing one and save the relative checkpoints to the model registry to be ready to export it to other machines. This tool was used in multiple activities in the PhD project and it proved useful to achieve results described in chapter 4.

Chapter 4

DL Applications on 3 Different Types of Datasets

SCF Calculations

When performing chemical and physical analyses at atomic and molecular level, it is required to solve the Schrödinger equation associated to the system:

$$H(r)\Psi(r) = E\Psi(r)$$

where r is the position vector, $H(r)$ the Hamiltonian operator, $\Psi(r)$ the solution wave, E the energy.

This equation can be exactly solved only for very simple cases, so approximations are needed to obtain solutions. The particles in the nuclei are considerably heavier than electrons, so they move slower. Thus, the Schrödinger equation can approximately be separated in a part describing the wave function of the electrons for a fixed nuclear geometry and in a second part describing the wave function of the nuclei. Starting from this simplified equation,

called Born-Oppenheimer approximation and restricting the set of solutions to single Slater determinants, the Hartree-Fock (HF) equations can be derived through variational principle [23]:

$$\mathbf{F_i}\phi_i^{'} = \epsilon_i\phi_i^{'} \tag{4.1}$$

where $i \in 1, .., N$, N being the number of electrons. $\phi_i^{'}$ are the solutions called canonical molecular orbitals (MO), ϵ_i are the related energies and F_i is called Fock operator, an one-electron energy operator incorporating the kinetic energy, the attraction and repulsion terms due to other particles in the system. The MOs can be expressed in any basis function set desired, but often Gaussian functions are chosen because they compensate a lack in the description of the electrons' structure with being very easy to handle computationally. Let us suppose to have chosen a basis set, χ_α, $\alpha \in 1, ..., M$, conventionally called atomic orbitals (AO). So, MOs can be written as a linear combination of the AOs:

$$\phi_i = \sum_{\alpha}^{M} c_{\alpha i}\chi_\alpha \tag{4.2}$$

So the Fock equation can be rewritten as:

$$\mathbf{F_i}\sum_{\alpha}^{M} c_{\alpha i}\chi_\alpha = \epsilon_i \sum_{\alpha}^{M} c_{\alpha i}\chi_\alpha \tag{4.3}$$

By multiplying basis functions and integrating, Roothaan-Hall equations can be obtained, i.e. the Fock equations in an AO basis that can be written in matrix notation:

$$\mathbf{FC} = \mathbf{SC}\epsilon \tag{4.4}$$

where $S_{\alpha\beta} = \langle\chi_\alpha|\chi_\beta\rangle$ are the overlap elements between basis functions and $F_{\alpha\beta} = \langle\chi_\alpha|\mathbf{F}\chi_\beta\rangle$ contains the Fock matrix elements. Solving the Roothaan-Hall means to find the eigenvalues of the Fock matrix, i.e. the MO coefficients, through diagonalization. The process starts with an initial guess of the coefficients, then the **F** matrix is calculated and diagonalization is performed, producing a new set of coefficents. These operations are done iteratively until the coefficients are close to the ones resulting from the diagonalization within a desired threshold, thus obtaining an SCF solution. The diagonalization of the Fock matrix is the most computationally intensive and time expensive step of the resolution of the Roothaan-Hall equations. Many algorithms have been developed and perfected to optimally perform this task, for example the Jacobi iteration method. During the process, it can be noted that the matrices generated by the SCF technique starts to differ only for certain elements corresponding to the occupied MOs of the structure in exam. So they are the only ones that really impact the computation of the SCF solution, the others being trascurable. Activities were started to try to develop a DL model capable of substituting the first steps of the SCF procedure and producing an initial guess as much as closer to the last final solution at convergence. Moreover, the desired result would be to produce a model suitable to be used not only for the structure used during training, but also with similar cases to the ones given as input. The initial guess obtained by the NN can be provided together with the input to Computational Chemistry Software such as Gaus-

sian or PySCF. Different approaches were tested. The input data composed by intermediate fock matrices and coefficient matrices was produced with PySCF (version 1.7.6a1) [62] for very simple bi-atomic omonuclear molecules using Lithium (Li), Berillium (Be), Boron (B), Carbon (C), Nitrogen (N), Oxygen (O) and Fluorine (F). Pyscf.gto and pyscf.scf.hf packages were used to obtain the training and validation data with the following options:

- Convergence tolerance, i.e. a threshold value for the difference between energies from consecutive iterations, equal to $1e^{-14}$;
- 6-31G as a basis set formed by gaussian functions;
- 5 different initial guesses for the Fock matrix already implemented by PyScf: identity matrix, minao, 1e, i.e. one-electron guess, atom, huckel;

For every approach, the Mean Squared Error (MSE) was used as loss metric and the chosen optimizer was AdamW, a modified version of the classic Adam with a different weight decay implementation, paired with a learning rate scheduler to scale it during the training. The most fundamental requirement for the model is to produce a initial guess in double precision, numerical accuracy needed in such kind of molecular simpulations. Thus, all the weights of the layers composing the NN tried during the research were double precision float variables, making the training and inferencing more computionally expensive. This increase in effort can be balanced by carefully selecting the parameters and hyperparameters of the model in order to

reduce the number of variables to be optimized during the training. The first approach was to train a NN to calculate the energy associated to the system and derive an initial guess from its weights. The model is composed by a singular fully connected layer of $[N, N]$ neurons, where N is the number of basis function, producing a scalar output. The input of the NN are the elements of the Fock matrices of $[N, N]$ dimension at the various iterations of the PySCF iterations, excluding the first one. The target variable is the final energy value calculated by PySCF and is compared to the output of the model using the L1 loss, i.e. the absolute value of the difference. The training set is formed by the Fock matrices generated during the different iterations and the related energy, excluding the Fock matrix at the first iteration that it is used for validating the NN. So the model tries to learn how to predict the energy value taking as input a Fock matrix produced by PySCF after fewer cycles than the number needed to complete the SCF procedure. This approach was quickly dropped because a simple linear combination of the elements of the Fock matrix is not able to approximate the calculation of the energy. In fact, the NN ends giving higher weights to the diagonal elements with the biggest absolute value, making the prediction not accurate and the model not useful for different structures besides the ones used during the training (figure 4.1).

The second tentative was to train a NN with multiple linear layers taking as input the iteration number and giving as output the diagonal elements of

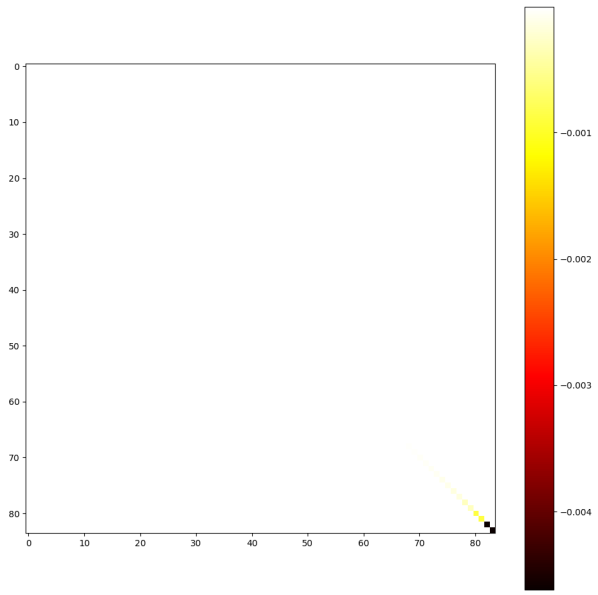

Figure 4.1: Weights' values heatmap for O_2

the Fock matrix, i.e. the eigenvalues, produced at that iteration. The idea behind this approach was that the diagonal is the matrix part with most importance for the calculation of energy in the SCF computation, so the other elements could be dropped during the training in the hope that the NN is capable to approximate the eigenvalues better. The trend of the eigenvalues during the iterations was investigated through plots to have a comparison on how the model was predicting the values. In figure 4.2 the first diagonal element values are visualized.

Even though non-linear activation functions were utilized, the NN finds a

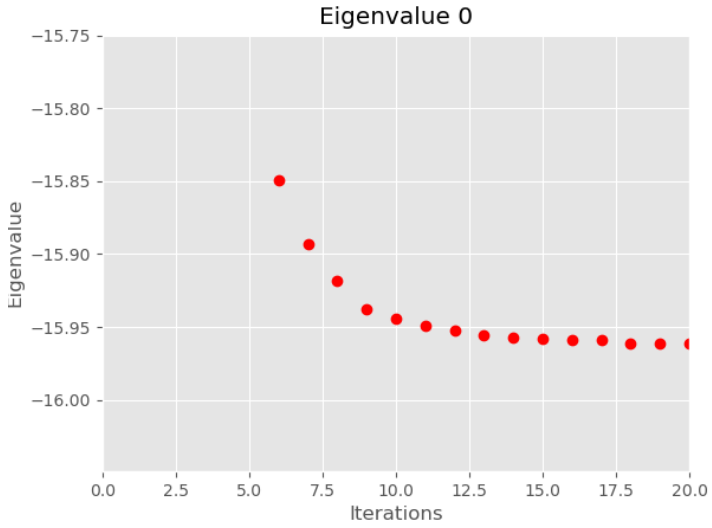

Figure 4.2: Trend of the first eigenvalue in respect to SCF iterations in the case of N_2

linear dependence between the number of iterations and the target variable, instead of plateauing at the right value. This behaviour makes the model not accurate, even if used on the same structure on which the training was performed. The next tentative started from the idea that during diagonalization the same operations on the same elements were performed on the matrix iteratively. So, the NN could be trained to approximate the computation performed at a single step by giving as input the Fock matrices at iteration j with the Fock matrices at iteration $j+1$ as labels. In this way, the final matrix could be obtained by inferencing multiple times from an initial guess. In this case, the model was again composed by a single dense layer with $[N, N]$ neurons to see how the weights associated to each element of the

Fock matrix varied. This was the first approach with better results. The model was trained for 30000 epochs on the dataset formed by all the Fock matrices obtained from PyScf for all 7 diatomic molecules and 5 initial guess. The optimizer's parameters had the following values:

- learning rate = 8;
- weight decay coefficient = 0.2;
- learning rate scheduler scaling factor = 0.6;
- scheduler step = 1000 epochs

The MSE at the end of the training was 0.003. To have an estimate of the quality of the prediction, the density matrix associated to the Fock matrix obtained from the model was given as input to a PySCF job to calculate the energy. The output file was compared to the ones obtained from the initial guesses. The energy value after 1 SCF cycle was comparable to the values obtained with the initial guesses already implemented in PySCF and significantly better than the identity matrix (reported as NN v1 in table 4.1 the values for F_2). However, when applying the model multiple times, the max difference between the predicted Fock matrix and the real one kept increasing, suggesting an "offset" value that it is added each time a new inferencing is performed. To try to avoid this behaviour, different models with the same structure were trained for the different iterations, i.e. model j

Initial Guess	Energy after 1 iteration	Final Energy	total N°of iterations
Identity Matrix	-57.939195	-198.650497	34
1e	-191.878812	-198.650497	11
atom	-198.632769	-198.650497	17
huckel	-198.647643	-198.650497	20
minao	-198.631569	-198.650497	19
NN v1	-196.273227	-198.650497	11
NN v2	-196.705364	-198.650497	14
NN v3	-196.71833	-198.650497	13

Table 4.1: Energy values after 1 SCF cycle with different initial guesses for F_2. The first 5 row are the initial guess implemented in PySCF, followed by the ones generated by the different version of the NN developed. The final energy values and total number of iterations to converge is also reported.

trains on the Fock matrices at iteration j with the Fock matrices at iteration $j+1$ as labels. Then, an initial guess can be produced by using the models in order. The Fock matrix obtained from the inferencing of two subsequent model was used as initial guess and compared as before to the ones already implemented in PySCF (reported as NN v2 in table 4.1 the values for F_2). The last approach was very similar to the previous, but for models $j > 1$ instead to use the Fock matrix at iteration j with the Fock matrix at iteration $j+1$ as label to train the model j, the output from the previous model $j-1$ is used as training data while retaining the same labels. (reported as NN v3 in table 4.1 the values for F_2). These results are not better than the initial guesses already implemented in PySCF, but they are still encouraging in keep investigating this approach in future work to find a suitable NN model capable of producing a good starting Fock Matrix for SCF calculations.

Analysis of Climatic Impact on Cows' Milk Production

The impact of weather conditions and temperature on the production and quality of cow milk is a well-known and established phenomenon [64–66]. However, often this effect is evaluated without taking in account the physical state and the genetics of the animals object of the analysis. The scope of the research started with the group lead by Prof. Giovanni Chillemi is to use Deep Learning models to evaluate 4 target variables of produced milk from a joint dataset of the physiological and genetic characteristics of the cows and weather conditions of the area in which the farm are located. The dataset is composed by more than 1.5 million of functional controls on almost 80000 cows in the span of multiple years, resulting in a size of 75 GB. The input features are the following:

- Animal features:
  - DIM_class: 15-days factor of lactation, each class is formed by 15 days, integer;
  - Parity_class: Number of different times a cow has had offspring, integer;
  - Ida: index of potential milk production bound to genetics;
- Climatic Features

  - CLCT: day-average cloud coverage measured in oktas;
  - RH: day-average relative humidity;
  - TOT_PREC: total precipitations measured in mm;
  - T: day-average temperature measured in Kelvin;
  - WS: wind speed measured in Km/h;

while the target quantities to be predicted are:

- Milk Production, measured in kg;
- Percentage of protein;
- Percentage of fat;
- Somatic Cell Count (SCC), an indicator of the health status of the cow;

The time period chosen to be evaluated is a 90 days period before the date of the functional control in which the animal features and the target variables were measured. A tentative of using DL to analyse the data was performed by the group providing the data using Gradient Boosting Machines (GBM) [67] with result not good enough for the scope of the research, so a different approach was tried.

The dataset of reference was provided through multiple csv files, one for each animal containing all the controls performed on it during its lifespan. Besides the animal features, the climatic measurements of the 90 days prior

of each control are reported to be easily used as input. Among the various types of DL models, LSTMs (Long-Short Term Memory) have really good performances when analyzing time series [68, 69]. So, LSTM seemed a good candidate to analyse the dataset, given the temporal nature of the climatic variables. However, the data is composed also by a set of "static" features, i.e. the animal features measured at fixed points of time (the date of the controls). Thus the arising issue is how to integrate the information contained in the static variables into the LSTM that takes as input the temporal variables.
As a first approach, a model with two "parallel" layers was tested to process the static and the time series data separately [70]. The animal features are given as input to a fully connected layer, while the climatic features are processed through a LSTM layer. Finally, the two outputs are passed together to another fully connected layer that "merges" the previous results and provides the final prediction on the target variables. The final structure of the model is schematized in figure 4.3.

AdamW was used as optimizer and the Mean Absolute Error (MAE) was chosen as loss metric to evaluate the quality of the prediction of the model. MLDE was used to find the optimal combination of the following hyperparameters:

- Hid_size: the number of the hidden units in the LSTM cells, they are the analogous of neurons in a linear layer;

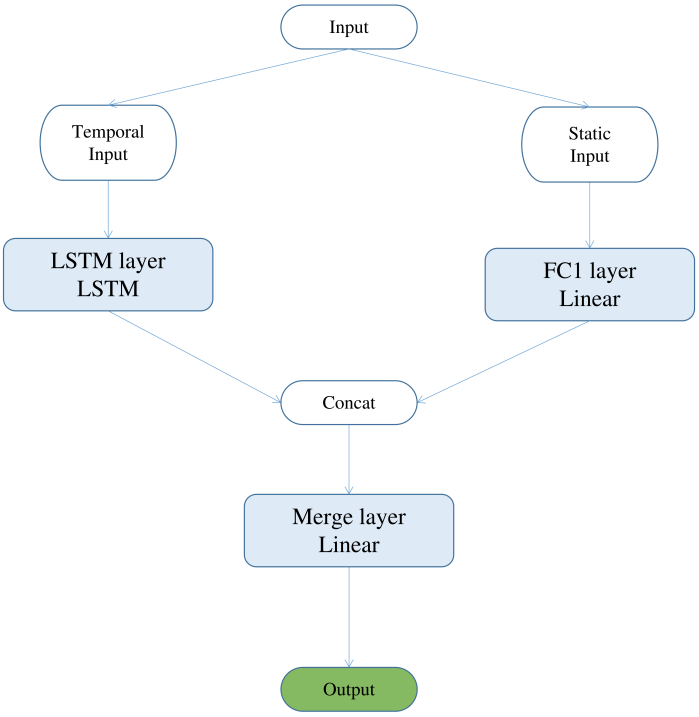

Figure 4.3: "Parallel" LSTM model structure

- Num_layers: the number of consequent LSTM layers, i.e. the number of stacked LSTMs that process the data in succession;
- Out_fc1: the number of neurons of the fully connected layer that takes as input the static variables;
- lr: the learning rate of the optimizer;
- alpha: the coefficient of the Exponential Linear Unit;

50 trials were performed with a maximum number of epochs equal to 4, using ASHA as search algorithm. The best performing hyperparameters' combinations for each of the 4 target variables are reported in table 4.2.

Compared to the errors obtained from the GBM, the LSTM model has higher error values for every target variables. The first tentative to lower the MAE was to train the model with the best performance for more epochs, using the possibility to continue trials in MLDE. It was not successful because the MAE kept oscillating in a range of values near the one found in the first 4

target	hid_size	num_layers	out_fc1	lr	train_loss	val_loss	val_loss (GBM)
Milk	456	3	475	0.004	6.569	4.929	2.8
Fat	452	10	455	0.018	0.533	0.531	0.4
Protein	199	10	94	0.028	0.258	0.244	0.16
SCC	131	4	77	0.051	0.48	0.463	0.36

Table 4.2: Optimal hyperparameter combinations for the 4 target variables using the first version of the model. Training loss and validation loss for the LSTM model are reported. The validation loss for the GBM model previously used is reported

epochs. Taking the milk case as an example, after training for 6 more epochs the resulting validation error was 5.245, higher than the one found before. Due to the fact that further training did not seem to have an impact on the error, a second version of an LSTM model was created. The structure was inspired by the Li, Lyons et al. paper [71]. Instead of feeding the input data to two separate parallel layers, the static features are transformed in order to be used as the initial hidden state for the LSTM layer processing the dynamic features as a way to merge the information learned from the two sets of variables. The animal variables are given as input to a fully connected linear layer followed by a ELU activation function and the resulting output is reshaped in order to be used as initial hidden state by the LSTM layer that processes the climatic variables. A last fully connected layer produces the target variable. The second version of the model is schematised in figure 4.4.

4 models with the previous structure were trained for each of the 4 variables to be predicted. The optimizer and loss metric were the same as the previous version. Also, MLDE was used with the same options for the following hyperparameters:

- HID_SIZE: in this version is not only the hidden size of the LSTM layer,

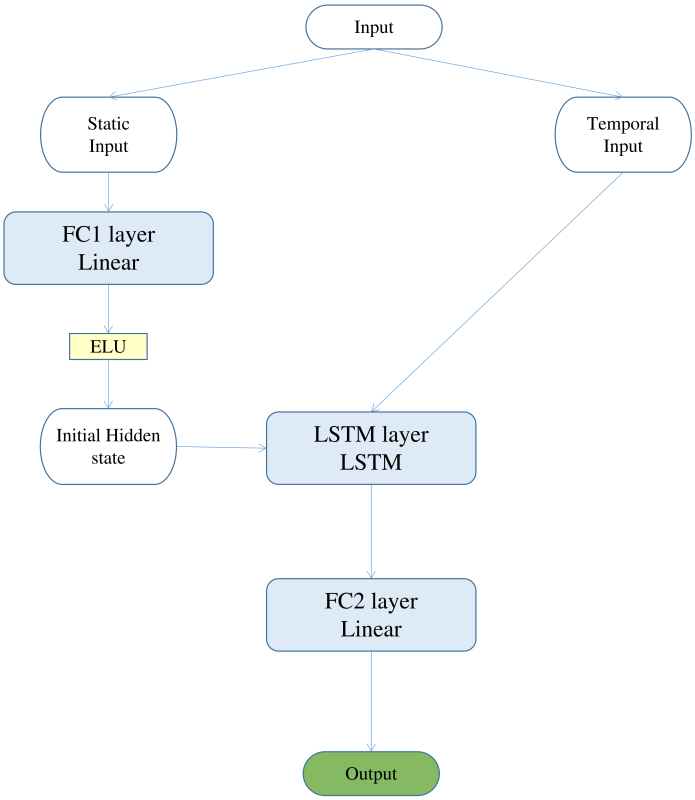

Figure 4.4: model structure

but also the number of neurons in the fully connected layer;

- NUM_LAYERS, as above;
- lr, as above;
- alpha, as above;

The best performing hyperparameters' combinations for each of the 4 target variables are reported in table 4.3.

The results obtained with the second version of the are still worse than the GBM, even when testing with different ranges for the hyperparameters. This suggests that further attempts can be directed in two different directions: the first is to keep using LSTMs and modify the model structure by adding more layers or different activation functions, while the second option is to try different kind of models beside LSTM, such as GBM, as already tried by the Prof. Chillemi group, or transformers [72].

target	hid_size	num_layers	out_fc1	alpha	train_loss	val_loss	val_loss (GBM)
Milk	137	5	0.006	0.439	5.892	6.053	2.8
Fat	207	2	0.007	0.48	0.537	0.519	0.4
Protein	492	8	0.001	0.528	0.31	0.316	0.16
SCC	157	6	0.008	0.306	0.492	0.477	0.36

Table 4.3: Optimal hyperparameter combinations for the 4 target variables using the second version of the model. Training loss and validation loss for the LSTM model are reported. The validation loss for the GBM model previously used is reported

NIR and VIS Spectroscopy

Spectroscopy had always made a heavy use of statistics to analyse the data obtained from instruments, so it was a natural step to take advantage of the progress of Machine Learning to get a deeper understanding of the measurements and to tackle the increase in dataset size [50]. Examples of ML algorithms often used are Principal Component Analysis (PCA), Linear Discriminant Analysis (LDA) and Partial Least Squares (PLS), but Deep Learning models are starting to get more used in spectroscopy. Deep Learning (DL) is a term used to indicate a subset of ML techniques that groups up the different types of Neural Networks characterized by their different structures, learning mechanism and activation functions. These kind of models works with less human intervention than the standard chemometrics and it is possible that they are able to find relations between the data and the target variables not easy to be discovered, trading an improvement in results for a decrease in human interpretability, i.e. a "black box" approach. Among the DL models, Convolutional Neural Networks (CNN) demonstrated to have good performance for heterogenous ML tasks such as object detection, image classification, time series classification and other applications. The success of CNN in all these cases, in particular on image analysis, gave inspiration to use it on 1D spectra, both for classification and regression [52]. The department research group led by Prof. Roberto Moscetti uses visible (vis)

and Near-InfraRed (NIR) spectroscopy to investigate different food phenomena and characteristics [53, 54]. Lately, they are working on two different NIR spectra database using Partial Least Squared Regression (PLS-R) with feature selection and cross-validation to select the best combination of pre-processing treatments. The research group proposed to compare the results obtained from the PLS-R with the various pre-treatments to the output of a DL model including a Convolutional layer. The NN chosen follows the rule-of-thumb structure presented in [52]. The model suggested has the following components:

- A single Convolutional layer on top (CONV);
- 4 Densely Connected layers (FC), with the last one outputting the predicted values;
- Activation functions: Exponential Linear Unit (ELU) after the convolutional layer, Rectified Linear Unit after the densely connected layers (ReLU);
- Weight inizialization using Kaiming technique [55]
- Adam optimizer;
- Mean Squared Error (MSE) as loss metric;

The DL model I developed started with those characteristic and then was modified step by step to increase the accuracy of the prediction. Its final

structure is shown in figure 4.5. The number and typology of layers remained the same, while ELU was used after CONV and every FC instead of applying it only after the former. The activation functions were changed to add complexity to the model in order to better approximate the relation between input and output variables. For the optimizer, AdamW optimizer [56] was chosen instead of the standard Adam because it implements better weight decay for regularization in the case it is needed to avoid overfitting. Root Mean Squared Error (RMSE) was used as loss metric instead of MSE to have a direct comparison to the PLS-R models that use it as principal measurement for the validation performance. A learning rate scheduler was implemented in order to scale down the parameter by a coefficient every time a chosen number of epochs during the training phase. As in the previous section, it is desirable to have results in double precision to be comparable to the quantities measured in laboratory. So, all the layers' weights and hyperparameters were set up as double precision float variables in order to obtain an output with the same precision. The research cases on which the comparison between CNN and PLS-R using MLDE to explore the hyperparameters' space were the following:

- Prediction of dry substance content in Potatoes from NIR spectra;
- Prediction of percentage of adulteration of olive oil mixed with other vegetable oil from visible and NIR spectra;

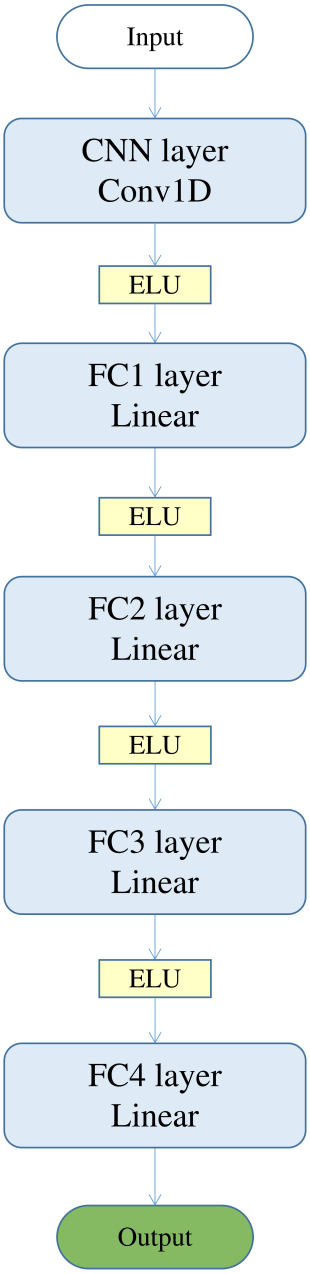

Figure 4.5: CNN model structure

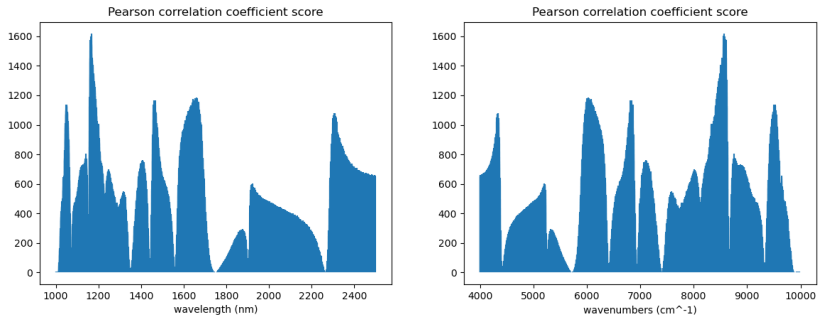

Figure 4.6: Example graph of Pearson Correlation Score on NIR spectra absorbance values

The same model structure was used for both cases.

To achieve a reduction in the 3112 input variables of the NIR spectra, the Pearson Correlation Coefficient [61] was used to score the features, then re-ordered by these values. The scikit-learn [63] feature_selection package was used to perform the calculation, in particular the method SelectKbest with the f_regression as score function. After re-ordering the input variables, the number of the top features can be selected to be the only ones used for training.

By using MLDE, I was able to easily test many different hyperparameters' combinations, in a faster way than performing the search of the optimal values manually and without writing a dedicated script to do it. This is a way to explore more vastly the hyperparameters' space, specially when an environment with multiple GPUs is available. The hyperparameters chosen to be varied through MLDE were:

- Kernel_size : the dimension of the filter used to analyze the data in the CNN layer, i.e. the window of neighbouring features processed for each one of them;
- n_features : the number of input features to use for training, selected the ones with the highest Pearson Correlation Coefficient;
- Alpha: the coefficient of the Exponential Linear Unit;
- out_fc1: the number of neurons of the first linear fully connected layer;
- out_fc2: the number of neurons of the second linear fully connected layer;
- out_fc3: the number of neurons of the third linear fully connected layer;
- lr: the learning rate of the optimizer;

The following hyperparameters relative to the optimizer were instead set as constant:

- b1 = 0.9, b2 = 0.999: coefficients to compute the gradients and their squares;
- eps = 1e-8: additive term to the denominator to avoid numerical instability, important when dealing with double precision weights;
- weight_decay = 0: coefficient for weight decay, a regularization term;

The trials were trained with batches of 256 samples for 500 epochs and their max number was set to 200, each one using a single slot. The first versions of the model without feature selection were run on the HPE Grenoble MLDE cluster, which I was kindly given access during my research period abroad in Basel at HPE HPC/AI EMEA Research Lab (ERL) guided by Utz-Uwe Haus. I was given permission to run the experiments on a resource pool with 16 GPU slots available on 8 Nvidia A100 with 80 GB memory. Then the experiments with the new versions were continued on the DIBAF cluster using an HPE ProLiant DL380 machine equipped with 2 Intel(R) Xeon(R) Gold 6240R CPUs with 24 cores each, 256 GB of RAM and an Nvidia GPU mod. A100 with 40 GB of VRAM. The underlying software stack present on the machine was composed of Ubuntu v20.04.6 LTS, Nvidia CUDA Toolkit v11.2, Nvidia driver v460.73.01, and Docker v26.1.4. As development software, the Nvidia RAPIDS Notebook (docker image: 23.12-cuda11.2-py3.10) was used as a DL framework in combination with PyTorch v2.2.0. MLDE v0.25.1 was used for model training, model validation and the selection of the best hyperparameters' combination.

Prediction of Dry Matter Content in Potatoes

The monitoring of dry matter content in potato is very useful to assess the physiological maturity of the samples, moreover can give hints on how and how long to storage the tubers [74]. The research group of prof. Moscetti

model		# samples	Size
Cortex	Training set (augmented)	2820	71.4 MB
	Validation set	120	3.75 MB
Parenchima	Training set (augmented)	4020	101.8 MB
	Validation set	172	4.21 MB
Parenchima ext.	Training set (augmented)	2620	66.3 MB
	Validation set	111	2.73 MB
Parenchima int.	Training set (augmented)	1410	35.7 MB
	Validation set	60	1.47 MB
Peel	Training set (augmented)	2820	71.4 MB
	Validation set	120	3.75 MB
Pulp	Training set (augmented)	6840	173.2 MB
	Validation set	292	7.1 MB

Table 4.4: Size of the datasets containing the samples for the different part of the potatoes.

used FT-NIR spectroscopy to gather data on different cultivar of potatoes, together with the measurement of dry substance. So the dataset was composed by NIR measurements on pieces of 12 different cultivar of potatoes to evaluate the dry substance content in 6 parts of the tuber: cortex, external parenchyma, internal parenchyma, parenchyma, peel and pulp. The data was provided pre-treated using different statistical procedures, in double precision and augmented to increase the numerosity of the training set. Matlab was used to perform these operations and the resulting data, comprising of the training and validation sets, was exported in .mat format to be used in Python for the DL model. The size of the datasets are reported in table 4.4.

Six different models with the same structure described above were trained using MLDE for the different part of the potatoes. The layers and the hy-

perparameters chosen are the same of the previous section.

As a first tentative, all the 3112 features of the data already pre-treated were used as input of the model without feature selection. The best models found with MLDE are reported in table 4.5.

After these first runs, feature selection with the Pearson correlation coefficient was applied obtaining the results in table 4.6.

The value of errors using the models with feature selection were comparable to the ones obtained by PLS-R, except for the peel which was also higher than the other parts of the potato.

The next tentative was to try to perform mean centering on the training set and validation set separately instead of doing it on the entire dataset. The results are displayed in table 4.7.

With this new adjustement, validation error is under 1 for each model, improving the quality of the models and obtaining results comparable to the ones obtained through PLS-R.

model	kernel_size	n_features	alpha	out_fc1	out_fc2	out_fc3	lr	RMSE-T	RMSE-V
Cortex	3	3112	0.1	232	62	9	0.024	1.363	1.666
Parenchima	5	3112	1.343	279	118	57	0.022	0.3	1.283
Parenchima Ext.	11	3112	1.38	168	156	65	0.048	1.08	1.571
Parenchima Int.	1	3112	1.213	485	46	42	0.031	0.246	1.13
Peel	5	3112	0.737	145	96	105	0.096	2.732	2.067
Pulp	21	3112	0.553	170	64	67	0.081	1.653	1.419

Table 4.5: Optimal hyperparameters' combinations for the six models with no feature selection. Training loss RMSE-T and validation loss RMSE-V for each model are reported.

model	kernel_size	n_features	alpha	out_fc1	out_fc2	out_fc3	lr	RMSE-T	RMSE-V
Cortex	7	3000	0.988	228	193	125	0.033	2.728	1.483
Parenchima	7	2500	0.823	230	52	9	0.025	0.602	1.161
Parenchima Ext.	11	2500	0.833	336	81	89	0.04	1.265	1.384
Parenchima Int.	1	2700	1.342	354	119	29	0.061	0.553	1.069
Peel	21	1500	0.521	212	121	31	0.01	0.991	2.015
Pulp	21	1900	0.577	418	63	42	0.012	1.37	1.338

Table 4.6: Optimal hyperparameters' combinations for the six models with Pearson feature selection. Training loss RMSE-T and validation loss RMSE-V for each model are reported.

model	kernel_size	n_features	alpha	out_fc1	out_fc2	out_fc3	lr	RMSE-T	RMSE-V
Cortex	21	1100	0.63	190	124	73	0.016	0.509	0.66
Parenchima	5	2700	1.078	368	82	41	0.034	0.65	0.828
Parenchima Ext.	3	1500	0.544	324	122	113	0.038	0.734	0.849
Parenchima Int.	3	100	0.862	157	62	75	0.065	0.366	0.721
Peel	17	1700	0.645	347	188	98	0.018	0.8	0.845
Pulp	9	1900	0.371	329	114	101	0.018	0.427	0.589

Table 4.7: Optimal hyperparameters' combinations for the 6 models with Pearson feature selection after performing mean centering on the training and validation sets separately. Training loss RMSE-T and validation loss RMSE-V for each model are reported.

model	kernel_size	n_features	alpha	out_fc1	out_fc2	out_fc3	lr	RMSE-T	RMSE-V
Cortex	5	109	0.80	166	136	96	0.031	0.507	0.663
Parenchima	9	75	0.523	128	94	80	0.019	0.482	0.765
Parenchima Ext.	15	98	1.206	272	132	109	0.02	0.75	0.873
Parenchima Int.	11	84	1.187	179	127	12	0.013	0.088	0.794
Peel	5	93	0.712	196	48	63	0.02	0.655	0.934
Pulp	1	99	0.416	256	99	24	0.018	0.349	0.46

Table 4.8: Optimal hyperparameters' combinations for the 6 models using PLS-R selected features after performing mean centering on the training and validation sets separately. Training loss RMSE-T and validation loss RMSE-V for each model are reported.

As a last tentative to lower further the errors, the reduced features obtained from the PLS-R model were used for the input instead of selecting them with the Pearson Correlation Coefficient, trying to leverage both types of approach. The results of this new method is reported in table 4.8.

The final models obtained can be compared to the RMSE errors obtained by the PLS-R models developed previously. The comparison is reported in table 4.9. From the results, CNN models with feature selection perform better than PLS-R models in every case, with the most improvement for the peel samples. The two types of feature selection seems to not affect heavily the validation error. So a correct hyperparameter selection, performed with MLDE, helped to find DL models capable of providing prediction more accurate than classical statistical chemometrics techniques. The importance of the choice of hyperparameters will be highlighted also on the results on the other dataset obtained from spectroscopy in the following section.

	Model	N. spectra	RMSE-T	RMSE-V
Cortex	PLS-R with reduced features	109	1.314	1.198
	PLS-R with all features	3112	1.551	1.506
	CNN with Pearson selected features	1100	0.509	0.66
	CNN with PLS-R selected features	109	0.507	0.663
Parenchima	PLS-R with reduced features	75	1.346	1.293
	PLS-R with all features	3112	1.438	1.256
	CNN with Pearson selected features	2700	0.65	0.828
	CNN with PLS-R selected features	75	0.482	0.765
Parenchima Ext.	PLS-R with reduced features	98	1.356	1.222
	PLS-R with all features	3112	1.244	1.174
	CNN with Pearson selected features	1500	0.734	0.849
	CNN with PLS-R selected features	98	0.75	0.873
Parenchima Int.	PLS-R with reduced features	84	0.965	1.019
	PLS-R with all features	3112	0,97	1,217
	CNN with Pearson selected features	100	0.366	0.721
	CNN with PLS-R selected features	84	0.088	0.794
Peel	PLS-R with reduced features	93	1.845	1.911
	PLS-R with all features	3112	2.269	1.922
	CNN with Pearson selected features	1700	0.8	0.845
	CNN with PLS-R selected features	93	0.655	0.934
Pulp	PLS-R with reduced features	99	1.297	1.277
	PLS-R with all features	3112	1.518	1.404
	CNN with Pearson selected features	1900	0.427	0.589
	CNN with PLS-R selected features	99	0.349	0.46

Table 4.9: Performance comparison between the best PLS-R and CNN models for each potato parts using the two different criteria to select the input features. The training error RMSE-T and the validation error RMSE-T are reported.

Detection of Adulteration of Olive Oil

Extra-virgin olive oil (EVOO) is a product of high economic value and the presence of adulterants can impact heavily on its quality. The use of spectroscopy can help to detect external products in a olive oil samples [75] [76]. The aim of the activity was to compare the PLS-R and the CNN performance on evaluating the percentage of polluting vegetable oil, such as maize, peanut, soy and sunflower, mixed with olive oil from visible (vis) and NIR spectra absorbance values. The dataset was provided both raw and pre-treated with different statistical procedures, in double precision and augmented to increase the numerosity of the training set. Matlab was used to perform these operations and the resulting data, comprising of the training and validation sets, was exported in .mat format to be used in Python for the deep learning model. The size of the datasets are reported in table 4.10.

Four different models with the structure described at the start of this section were trained for each adulterant separately. The first version of the model worked on the vis and NIR spectra fused together for a total of 3633 input features with no selection. The results are displayed in table 4.11.

Then, the models were trained on the same dataset with feature reduction based on the Pearson Correlation Coefficient. The hyperparameters' combinations with the best performance are reported in table 4.12.

The next step in the tuning of the model was to evaluate its performance when trained on the vis and NIR spectra separately with feature selection.

model			# samples	Size
Maize	VIS	Training set (augmented)	2450	11.6 MB
		Validation set	129	0.61 MB
	NIR	Training set (augmented)	2460	60.1 MB
		Validation set	129	3.16 MB
Peanut	VIS	Training set (augmented)	2460	11.7 MB
		Validation set	129	0.61 MB
	NIR	Training set (augmented)	2450	60.1 MB
		Validation set	129	3.16 MB
Soy	VIS	Training set (augmented)	2450	11,6 MB
		Validation set	129	0.61 MB
	NIR	Training set (augmented)	2440	59.8 MB
		Validation set	128	3.13 MB
Sunflower	VIS	Training set (augmented)	2440	11.6 MB
		Validation set	128	0.61 MB
	NIR	Training set (augmented)	2460	60.1 MB
		Validation set	129	3.16 MB

Table 4.10: Size of the datasets containing the samples for the different vegetable oils.

model	kernel_size	n_features	alpha	out_fc1	out_fc2	out_fc3	lr	RMSE-T	RMSE-V
Maize	5	3633	0.982	342	103	50	0.013	0.619	1.324
Peanut	13	3633	0.727	286	236	36	0.017	1.512	1.304
Soy	21	3633	0.663	297	65	104	0.022	1.407	1.301
Sunflower	17	3633	0.411	184	223	55	0.011	0.844	1.15

Table 4.11: Optimal hyperparameters' combinations for the four models using the full combined vis and NIR spectra. Training loss RMSE-T and validation loss RMSE-V for each model are reported.

model	kernel_size	n_features	alpha	out_fc1	out_fc2	out_fc3	lr	RMSE-T	RMSE-V
Maize	13	900	0.625	260	94	82	0.033	1.783	1.203
Peanut	19	700	0.67	362	201	111	0.019	1.046	0.886
Soy	21	1700	0.409	267	226	63	0.028	1.971	1.317
Sunflower	19	900	0.306	149	43	80	0.011	0.844	1.15

Table 4.12: Optimal hyperparameters' combinations for the four models using the full combined vis and NIR spectra and Pearson feature selection. Training loss RMSE-T and validation loss RMSE-V for each model are reported.

So two models for each type of oil was obtained and the results are displayed in tables 4.13 and 4.14.

The validation errors for the model when training was performed separately on the two spectra obtained performance close to the full spectra, except for the peanut oil. The high RMSE during validation was due to the fact that for low percentages of adulteration the NIR measurements are very similar to the ones obtained from pure EVOO. Thus the CNN is not able to discriminate between the different samples and returns as RMSE a

model	kernel_size	n_features	alpha	out_fc1	out_fc2	out_fc3	lr	RMSE-T	RMSE-V
Maize	9	100	0.647	170	163	44	0.024	1.459	1.543
Peanut	9	150	0.786	167	176	40	0.019	1.082	1.123
Soy	15	400	0.456	318	226	70	0.023	1.317	1.233
Sunflower	19	450	0.83	154	113	111	0.022	1.252	1.181

Table 4.13: Optimal hyperparameters' combinations for the four models using vis spectrum and Pearson feature selection. Training loss RMSE-T and validation loss RMSE-V for each model are reported.

model	kernel_size	n_features	alpha	out_fc1	out_fc2	out_fc3	lr	RMSE-T	RMSE-V
Maize	13	2700	0.898	466	71	13	0.011	2.393	1.897
Peanut	19	1500	0.786	361	158	51	0.011	20.14	22.14
Soy	1	1900	0.492	250	167	118	0.012	1.962	1.11
Sunflower	7	300	0.726	279	104	108	0.025	1.438	1.26

Table 4.14: Optimal hyperparameters' combinations for the four models using NIR spectrum and Pearson feature selection. Training loss RMSE-T and validation loss RMSE-V for each model are reported.

value close to the standard deviation of the training set, i.e. ~ 22. All these MLDE experiments were performed on raw data without any statistic pretreatment. As a next step, the research group performed different operations on the data to improve their PLS-R models and provided it to be also used as input for the CNN.

The results obtained using the modified data are reported in tables 4.15 and 4.16

All the checkpoints obtained with MLDE were exported and used to obtain all the prediction values on the samples in the validation set to evaluate also the coefficient of determination R^2 and the bias to compare the quality of the models with the PLS-R previously performed on the same data [73]. Instead of just using the RMSE as above, the best trials were selected using a cumulative performance index (cpi) taking into account all the metrics chosen to evaluate the models. The cpi developed and calculated by the research group of Prof. Moscetti uses PCA to analyse the performance metrics and producing a value between 0 and 1 from the loadings and the proportions of

total variance explained by each component. The comparison between the best CNN models and PLS-R models selected with cpi are reported in tables 4.17 and 4.18.

It is important to point out that the values of the RMSE reported in tables 4.17 and 4.18 have been re-scaled back with the same parameters used to perform mean centering and/or auto-scaling in the data pretreatment. Looking at the values of the performance metrics, the CNN and PLS-r perform pretty much similar on NIR spectra, with the latter having slightly better RMSE on every adulterant except maize oil. Instead, CNN has better performance on every adulterant when considering the vis spectra, making it a real alternative to standard chemometrics techniques. An article with the results has been submitted and it is at present in the revision phase [77]. Again, it is highlighted how the correct choice of hyperparameters, paired with an effective pre-processing of data, is able to produce models able to obtain the desired results.

model	M.C.	kernel_size	n_features	alpha	out_fc1	out_fc2	out_fc3	lr	RMSE-T	RMSE-V
Maize	X	13	350	0.227	226	199	40	0.016	0.096	0.069
Maize	✓	15	400	0.6	246	119	50	0.011	0.062	0.064
Peanut	X	7	100	0.452	165	92	124	0.013	0.057	0.054
Peanut	✓	21	250	0.515	476	50	11	0.022	0.052	0.071
Soy	X	15	250	0.489	229	242	48	0.017	0.066	0.06
Soy	✓	15	450	0.68	244	195	54	0.011	0.057	0.064
Sunflower	X	11	400	0.278	370	130	10	0.011	0.07	0.063
Sunflower	✓	3	100	0.291	176	143	47	0.01	0.044	0.068

Table 4.15: Optimal hyperparameters' combinations for the four models using vis spectrum and Pearson feature selection, trained on pre-treated data with and without Mean Centering (M.C.). Training loss RMSE-T and validation loss RMSE-V for each model are reported.

model	M.C.	kernel_size	n_features	alpha	out_fc1	out_fc2	out_fc3	lr	RMSE-T	RMSE-V
Maize	X	13	100	0.717	302	90	108	0.017	0.985	1.049
Maize	✓	5	1100	0.613	495	130	110	0.042	1.029	37.06
Maize	X	3	1500	0.387	504	48	19	0.044	0.99	1.094
Maize	✓	1	2300	0.947	135	225	9	0.036	0.992	1.094
Peanut	X	13	2700	0.393	260	119	107	0.085	0.994	1.092
Peanut	✓	13	1500	0.7	230	173	11	0.012	0.025	0.052
Soy	X	1	1500	1.478	432	77	108	0.015	1	1.096
Soy	✓	19	700	1.126	337	228	59	0.011	0.057	0.093
Sunflower	X	19	700	0.912	229	50	45	0.014	0.061	0.07
Sunflower	✓	5	300	1.002	291	63	120	0.017	0.046	0.05

Table 4.16: Optimal hyperparameters' combinations for the four models using NIR spectrum and Pearson feature selection, trained on pre-treated data with and without Mean Centering (M.C.). Training loss RMSE-T and validation loss RMSE-V for each model are reported.

Adulterant		Data Augmentation	RMSE-T		BIAS		R2	
			T	V	T	V	T	V
Peanut	PLS	X	1.205	1.250	-5.330e-15	-1.746e-01	0.997	0.997
		✓	1.029	1.133	2.470e-13	-1.903e-01	0.997	0.998
	CNN raw	✓	20.276	22.143	4.505e-02	5.091e-02	0.500	0.494
	CNN pretreated	✓	1.517	1.625	8.987e-01	6.523e-01	0.999	0.997
Sunflower	PLS	X	1.021	1.211	-4.090e-14	2.965e-01	0.998	0.997
		✓	1.049	1.190	-2.260e-13	2.911e-01	0.997	0.997
	CNN raw	✓	1.661	1.666	-6.823e-01	-4.394e-01	0.994	0.994
	CNN pretreated	✓	1.432	1.402	-4.977e-01	-4.266e-01	0.996	0.996
Maize	PLS	X	1.641	1.471	-2.490e-14	1.230e-03	0.993	0.996
		✓	1.750	1.518	7.820e-14	-3.651e-02	0.993	0.995
	CNN raw	✓	2.108	1.933	-1.180E+00	-1.077E+00	0.993	0.995
	CNN pretreated	✓	1.631	1.423	-1.397e-02	-2.672e-02	0.994	0.996
Soy	PLS	X	1.161	0.987	-5.330e-14	-1.874e-02	0.997	0.998
		✓	1.216	1.031	5.560e-13	-1.689e-02	0.996	0.998
	CNN raw	✓	1.978	1.980	1.475E+00	1.453E+00	0.996	0.996
	CNN pretreated	✓	1.918	2.082	-1.198e-02	1.013e-01	0.997	0.996

Table 4.17: Performance metrics for NIR spectra

Adulterant		Data Augmentation	RMSE		BIAS		R2	
			T	V	T	V	T	V
Peanut	PLS	X	2.164	2.421	-2.130e-14	4.602e-02	0.989	0.988
		✓	2.334	2.529	-1.780e-14	1.168e-01	0.987	0.987
	CNN raw	✓	1.225	1.220	2.718e-01	1.246e-01	0.997	0.997
	CNN pretreated	✓	1.163	1.174	2.645e-01	2.026e-01	0.997	0.998
Sunflower	PLS	X	2.924	3.670	-8.880e-15	1.605e-01	0.976	0.970
		✓	2.631	3.088	5.510e-14	-1.334e-02	0.980	0.978
	CNN raw	✓	1.402	1.398	3.885e-01	4.417e-01	0.995	0.996
	CNN pretreated	✓	1.341	1.464	-4.012e-01	-4.143e-01	0.996	0.996
Maize	PLS	X	1.754	1.922	-2.130e-14	-1.324e-01	0.993	0.993
		✓	1.826	1.946	-1.420e-13	-9.440e-02	0.992	0.992
	CNN raw	✓	1.380	1.390	3.415e-01	1.892e-01	0.996	0.996
	CNN pretreated	✓	1.358	1.336	3.535e-01	3.581e-01	0.996	0.997
Soy	PLS	X	2.058	2.051	-3.380e-14	1.081e-01	0.990	0.991
		✓	2.218	2.315	-5.510e-14	1.579e-01	0.988	0.989
	CNN raw	✓	1.584	1.553	4.899e-01	6.119e-01	0.995	0.996
	CNN pretreated	✓	1.443	1.422	-8.855e-02	1.030e-01	0.995	0.996

Table 4.18: Performance metrics for vis spectra

Conclusions

The advancements in Machine Learning and the availability of Big Data opened up a lot of new opportunities in scientific research, but also new challenges arose in how to apply these algorithms and manage this high dimension-high variety data. High Performance Computing technologies are the most viable and efficient way to face these new problems. During the PhD project the HPC cluster at the Department for Innovation in Biological, Agri-Food and Forest Systems was updated, both in hardware and software, to be an important resource not only on the research activities already carried out by the different research groups, but also for new initiatives leveraging Big Data and Deep Learning. On the hardware side, new components, ranging from new cooling solutions to new network technologies, were added to the cluster to improve its stability and overall optimization, accompanied by a re-organization and re-cabling of the parts already present. These activities were accompanied also by an update also on the middleware and software side. In order to carry out these changes, the experience gained during the work internship in Prisma S.p.a., part of the "PON - Ricerca

e Innovazione 2014-2020" PhD fellowship, was useful to gain the necessary knowledge. As a first step, new computing job queues were created for the different department activities in order to use the cluster resources in a more efficient way. New tools were made available to the users, such as Apptainer to run Docker containers on the different queues. In order to re-organize the services already present on the DIBAF cluster and make it easier to manage them and launch new ones, a Kubernetes cluster with GPU access was created through Rancher. Different services are running in this new environment, such as databases for sensors' measurements and Jupyter notebooks used for the scripts and the development of the ML models during the PhD project. One of those activities was the analysis of the interaction between Borazine and Noble Gases, useful not only to learn how to use an HPC cluster, but also to enter the world of computational chemistry, in which optimization algorithms (that have a central role in ML) are very well implemented. The results of these analyses were reported in a published paper on a peer-reviewed journal. To further support the activities to apply Deep Learning models to datasets of interest of the department, HPE Machine Learning Development Environment was installed on the cluster. MLDE is a new tool that has the scope to help developers to launch DL models on HPC systems, taking care also of parallelization and the utilization of GPUs. I had the opportunity to test it during the research period abroad at HPE HPC/AI EMEA Research Lab in Basel as part of the PhD fellowship. One

of the main features of MLDE is model hyperparameters' optimisation. Hyperparameters impact heavily models' performance and finding an optimal combination can be time-consuming. MLDE implements different algorithms to search the hyperparameters' space and find their best values using a validation metric as criterium. This feature was used in the models developed during the PhD activities to find the optimal hyperparameters' combinations. This approach was very effective on two of the datasets analysed like in my PhD project. These two datasets were composed by absorbance values in the NIR spectra measured on potatoes and olive oil (in this case together with values also in the visible spectra). Convolutional Neural Network was chosen among DL models to analyse the one-dimensional spectra because in literature it is showed to have very good performance on images and videos. Moreover, it was chosen to use double precision parameters and hyperparameters to obtain results with the same precision, fundamental to be feasible for the study cases. Using MLDE the optimal hyperparameters were chosen for each model trained on the different subset of samples. The models with the best performances were compared to PLS-R, a standard chemometrics statistical technique, and the CNN showed to have similar performance for the NIR spectra and better performance on Visible spectra with results of such a good quality to be selected for preparing two publications, one submitted, now under review, and one in preparation. These activities shows how DL can be very useful in analysing datasets of different origin that can

be encountered in many scientific fields. In particular, a correct choice of crucial hyperparameters of the model can impact the quality of prediction and, consequently, the effective application of DL in scientific applications. However, it is important to highlight the role of HPC systems without which the storage and the pretreatment of data, the training and the inferencing of models could not be possible, even more when dealing with the need of results with high numerical precision. Machine Learning, especially Deep Learning, can fully express their potential together with HPC systems and offer new possibilities in the analysis of Big Data.

Acknowledgments

I would like to express my special thanks to all people that helped and gave support to successfully carry out the PhD activities. In particular, I want to thank Prof. Nico Sanna, my PhD supervisor, who assisted me during the PhD, always available to share his knowledge and to confront and give precious advices. I want to thank also dr. Costantino Zazza, with which I shared not only research activities, but also many days working on the DIBAF cluster. The PhD was supported financially by the FSE-REACT-EU, PON "Ricerca&Innovazione" 2014-2020 program and by Prisma S.p.A, to which goes a particular acknowledgement. In fact, beside the financial participation, Prisma was able to give me the opportunity first to have a real work experience lead by Dr. Gianni Morelli, that I thank for his guidance and help. Secondly, Prisma facilitated the organization of my research period abroad at HPE AI&ML Research Group, a really teaching and important experience. I extend my acknowledgement for their support and kindness to all the people in Basel and in particular to Dr. Utz-Uwe Haus, head of the research group. Finally, I want to thank my family and friends, near and far,

who helped me during this three years' journey.

Bibliography

[1] Ward J.S., Barker A., Undefined by data: A survey of big data definitions., *arXiv*, *Cornell University Library*, url: https://arxiv.org/abs/1309.5821, 2013.

[2] url: https://aws.amazon.com/it/big-data/datalakes-and-analytics/what-is-a-data-lake/

[3] Mitchell T. M., Machine Learning, *McGraw-Hill*, 1247-1256, 1997

[4] Hagg A., Krischner K. N., Open-Source Machine Learning in Computational Chemistry, *Journal of Chemical Information and Modeling*, vol 63 (15), 4505-4532, 2023

[5] Shi Y., Tang Z., Machine Learning for Chemistry: Basics and Applications, *Engineering*, 27, 70-83, 2023

[6] Hampton S. E., Strasser C.A., Big data and the future of ecology, *Frontiers in Ecology and the Environment*, 11(3), 2013

[7] Kamilaris A., Kartakoullis A., Prenafeta-Boldú F. X., A review on the practice of big data analysis in agriculture, *Computers and Electronics in Agriculture* 143, 23-37, 2017

[8] Borocci S., Camerlingo A. et al., Complexes of the noble-gas atoms with borazine: Theoretical insights into structure, stability, and bonding character, *Chemical Physics Letters*, Volume 834, 2024

[9] Passamonti M.M., Somenzi E. et al, The Quest for Genes Involved in Adaptation to Climate Change in Ruminant Livestock, *Animals* 11(10), 2833, 2021

[10] Urtubia A., Perez-Correa et al., Using data mining techniques to predict industrial wine problem fermentations, *Food Control* 18 (1), 1512-1517, 2007

[11] Sterling T., Brodowicz M., Anderson M., High performance computing: modern systems and practices, *Morgan Kaufmann*, 2017

[12] top500, url:https://top500.org/lists/top500/list/2024/06/, consulted in September 2024

[13] Iozone Filesystem Benchmark, url: https://www.iozone.org/, consulted in Septemeber 2024

[14] Lustre, url: https://www.lustre.org/, consulted in September 2024

[15] PON Ricerca e Innovazione 2014-2020 PhD,url: https://www.ponricerca.gov.it/notizie/2021/dottorati-su-tematiche-dell-innovazione-e-green-nuove-risorse-dal-pon-ricerca-e-innovazione/, consulted in September 2024

[16] SLURM workload manager, url: https://slurm.schedmd.com/documentation.html, consulted in September 2024

[17] Docker, url: https://www.docker.com/, consulted in September 2024

[18] Apptainer, url: https://apptainer.org/, consulted in September 2024

[19] Kubernetes, url: https://kubernetes.io/, consulted in September 2024

[20] Rancher, url: https://www.rancher.com/, consulted in September 2024

[21] Longhorn, https://longhorn.io/, Cambridge University Press, consulted in September 2024

[22] Lowe J. P., Quantum Chemistry, *Academic Press*, second edition, 1993

[23] Jensen F., Introduction to computational chemistry, *John Wiley & Sons*, 1999

[24] Messiah A., Quantum Mechanics, *John Wiley & Sons*, 1967

[25] Artrith N., Butler K. T. et al., Best practices in machine learning for chemistry, *Nature Chemistry*, VOL 13, 505-508, 2021

[26] Nocedal J., Wright S.J., Numerical Optimization, *Springer*, second edition, 2006

[27] Goldberg D. E., Genetic Algorithms in Search, Optimization, and Machine Learning, *Addison-Welsey publishing company, Inc.*, 1989

[28] Auwärter W., Hexagonal boron nitride monolayers on metal supports: Versatile templates for atoms, molecules and nanostructures, *Surface Science Reports* 74, 1-95, 2019

[29] Levy A.C., Rybolt T.R., Pierotti R.A., High-temperature physical adsorption ofargon, krypton, and xenon on hexagonal boron nitride, *Journal of Colloid and Interface Science* 70, 74-82, 1979

[30] Shrestha P., Alkhafaji M.T. et al., Adsorption studies on boron nitride substrates, *Langmuir* 10, 3244-3249, 1944

[31] Morishige K., Inoue K., Imai K., X-ray study of Kr, Xe, and N2 monolayers on boron nitride, *Langmuir* 12, 4889-4891, 1996

[32] Li W., Shrestha P. et al., Monolayer Kr films adsorbed on BN, *Physics Review B* 54, 8833-8843, 1996

[33] Diama A., Migone A.D., Multilayer Kr films adsorbed on BN, *Physics Review B* 60, 16103-16108, 1999

[34] Dil H., Lobo-Checa J. et al., Surface trapping of atoms and molecules with dipole rings, *Science* 319, 1824-1826, 2008

[35] Widmer R., Passerone D., Probing the selectivity of a nanostructured surface by xenon adsorption, *Nanoscale* 2, 502-508, 2010

[36] Becke A.D., Density-functional thermochemistry. III. The role of exact exchange, *Journal of Chemical Physics* 98, 5648-5652, 1993

[37] Lee C., Yang W., Parr R.G., Development of the Colle-Salvetti correlation-energy formula into a functional of the electron density, *Physics Review B*, 785, 1988

[38] Glendening E.D., Natural energy decomposition analysis: Explicit evaluation of electrostatic and polarization effects with application to aqueous clusters of alkali metal cations and neutrals, *Journal of the American Chemical Society* 118, 2473-2482, 1996

[39] Bavafa S., Nowroozi A., Ebrahimi A., Ab initio study of aerogen-bonds between some heterocyclic compounds of benzene with the noble gas elements (Ne, Ar, and Kr), *Structural Chemistry* 31, 435-445, 2020

[40] C. Møller, M.S. Plesset, Note on an approximation treatment for many-electron systems, *Physics Review* 46, 618-622, 1934

[41] Jeziorski B., Moszynski R., Szalewicz K., Perturbation theory approach to intermolecular potential energy surfaces of van der Waals complexes, *Chemical Reviews* 94, 1887-1930, 1994

[42] Borocci S., Giordani M., Grandinetti F., Bonding motifs of noble-gas compounds as described by the local electron energy density, *The Journal of Physical Chemistry A* 119, 6528-6541, 2015

[43] Borocci S., Grandinetti F. et al., Classifying the chemical bonds involving the noble-gas atoms, New *Journal of Chemistry* 44, 14536-14550, 2020

[44] Borocci S., Grandinetti F., Sanna N., Noble gas compounds: A general procedure of bonding analysis, *The Journal of Chemical Physics* 44, 014104, 2022

[45] Frisch M.J., Trucks G.W. et al., Gaussian09, Revision D.01, *Gaussian Inc.*, Wallingford CT, 2013

[46] Raghavachari K., Trucks G.W. et al., A fifth-order perturbation comparison of electron correlation theories, *Chemical Physics Letters* 157, 479-483, 1989

[47] Pritchard B.P., Altarawy D. et al., New basis set exchange: an open, up-to-date resource for the molecular sciences community, *Jour-*

nal of Chemical Information and Modeling 59, 4814-4820, 2019, (url: https://www.basissetexchange.org)

[48] Merino-García M.D.R., Soriano-Agueda L.A. et al., Benzene and borazine, so different, yet so similar: insight from experimental charge density analysis, *Inorganic Chemistry* 61, 6785-6798, 2022

[49] Beattie J.R., Matossian J.N. et al., Xenon ion propulsion subsystem, *Journal of Propulsion and Power*, vol.5, number 4, 1989

[50] Meza Ramirez C.A., Greenop M. et al., Applications of machine learning in spectroscopy., *Applied Spectroscopy Reviews* 56(8-10), 733-763, 2021

[51] MLDE, url: https://hpe-mlde.determined.ai/index.html, consulted in September 2024

[52] Chenhao C., Fearn T., Modern practical convolutional neural networks for multivariate regression: applications to NIR calibration, *Chemometrics and Intelligent Laboratory Systems*, volume 182, 9-20, 2018

[53] Moscetti R., Haff R.P et al., Nondestructive detection of insect infested chestnuts based on NIR spectroscopy, *Postharvest Biology and Technology*, volume 87, 88-94, 2014

[54] Moscetti R., Haff R.P et al., Feasibility of NIR spectroscopy to detect olive fruit infested by Bactrocera oleae, *Postharvest Biology and Technology*, volume 99, 58-62, 2015

[55] He K., Zhang X. et al., Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification, *2015 IEEE International Conference on Computer Vision (ICCV)*, 1026-1034, 2015

[56] Loshcilov I., Hutter F., Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[57] Paszke A. et al., PyTorch: An Imperative Style, High-Performance Deep Learning Library., *Advances in Neural Information Processing Systems 32*, Curran Associates, Inc., pp. 8024-8035, 2019.

[58] Yang L., Shami A., On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing* 415, 295-316, 2020

[59] Bischl B., Binder M. et al., Hyperparameter optimization: Foundations, algorithms, best practices, and open challenges, *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 13.2, e1484, 2023

[60] Li, L., Jamieson, K.G. et al., A System for Massively Parallel Hyperparameter Tuning. *arXiv: Learning*, 2018

[61] Berman, J. J., Data Simplification, *Morgan Kaufmann*, 2016

[62] Sun Q., Berkelbach T.C. et al, PySCF: the Python-based simulations of chemistry framework, *WIREs Computational Molecolar Sciences*, 8, e1340, 2018

[63] Pedregosa F., Varoquaux G. et al., Scikit-learn: Machine Learning in Python, *JMLR*, 2011

[64] Klinedinst, P.L., Wilhite D.A. et al., The potential effects of climate change on summer season dairy cattle milk production and reproduction, *Climatic Change*, 1993

[65] Gaughan, J.B., Cawdell-Smith, A.J., Impact of climate change on livestock production and reproduction, *Climate Change Impacts on Livestock: Adaptation and Mitigation*, Springer, pp. 51-60, 2015

[66] Passamonti M.M., Somenzi E., et al., The Quest for Genes Involved in Adaptation to Climate Change in Ruminant Livestock, *Animals*, 11(10), 2833, 2021

[67] Friedman J. H., Greedy Function Approximation: A Gradient Boosting Machine., *Annals of Statistics*, 1189-1232, 2001.

[68] Van Houdt G., Mosquera C., Nápoles G., A Review on the Long Short-Term Memory Model, *Artificial Intelligence Review*, Volume 53, 5929-5955, 2020

[69] Greff K., Srivastava R. K. et al., LSTM: A Search Space Odyssey, *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 10, 2222-2232, 2017

[70] Leontjeva A., Kuzovkin I., Combining Static and Dynamic Features for Multivariate Sequence Classification, *2016 IEEE 3rd International Conference on Data Science and Advanced Analytics (DSAA)*, pp. 21-30, 2016

[71] Li D., Lyons P. et al., Integrating Static and Time-Series Data in Deep Recurrent Models for Oncology Early Warning Systems, *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, Association for Computing Machinery, 913-936, 2021

[72] Lim B., Zohren S., Time-series forecasting with deep learning: a survey, *Philosophical Transactions of Royal Society, A*, 379, 2021

[73] Nallan Chakravartula S.S., Moscetti R. et al., Use of convolutional neural network (CNN) combined with FT-NIR spectroscopy to predict food adulteration: A case study on coffee, *Food Control*, vol 135, 108816, 2022

[74] Heltoft P., Wold A.B., Maturity indicators for prediction of potato (Solanum tuberosum L.) quality during storage, *Postharvest Biology and Technology*, vol 129, pp 97-106, July, 2017

[75] Meng, X., Yin, C. et al., Rapid detection of adulteration of olive oil with soybean oil combined with chemometrics by Fourier transform infrared, visible-near-infrared and excitation-emission matrix fluorescence

spectroscopy: A comparative study., *Food Chemistry*, 405(PA), 134828, 2023

[76] Ye, Q., Meng, X., Highly efficient authentication of edible oils by FTIR spectroscopy coupled with chemometrics., *Food Chemistry*, 385, 132661, 2022

[77] Bandiera A., Camerlingo A. et al., Can Deep Chemometrics enhance the accuracy of EVOO adulteration detection from spectral data?, *Computers and Electronics in Agriculture*, Submitted, 2025