Università degli Studi di Pavia

Facoltà di Ingegneria

Dipartimento di Ingegneria Industriale e dell'Informazione

Corso di Laurea Magistrale in Bioingegneria

A Deep Learning Model for Molecular Fingerprinting

Un Modello di Deep Learning per il Fingerprinting Molecolare

Relatore:

Prof. Riccardo Bellazzi

Correlatore:

Prof. Blaž Zupan

Tesi di Laurea di

Mattia Cordioli

mat. **437735**

Anno Accademico 2017 - 2018

Abstract

Chemoinformatics is a recently developed discipline that combines chemistry and computer science and has largely benefited the field of drug discovery and design. Indeed, the use of *in silico* techniques allows researchers to avoid expensive experimental tests, selecting or discarding new compounds basing on similarity with target molecules or on the results of predictive models.

The key in those applications, especially in Quantitative Structure-Activity Relationships (QSAR) modelling and, in general, in the use of any statistical or machine learning method, is molecular representation. Molecules can widely vary in size and complexity, and even if standard chemical representation methods can carry all the information about their structure and composition, they are not adequate to input any model or algorithm: a numerical fix-sized representation is necessary.

Fingerprints are fix-sized vectors developed for extracting and representing molecules relevant features. A simple example of fingerprint is a bitstring where each bit represents the presence or absence of predetermined substructures, but a variety of more complex methods have been proposed, including the use of deep learning techniques.

The present work aims to develop a model for embedding, or fingerprinting, molecules into real-valued vectors, starting from SMILES, that are a molecular linear notation. Secondary aim is to implement the model in Orange, a data mining suite, in order to provide a tool to easily carry out chemistry data analysis.

A deep learning model, particularly a convolutional neural network, was chosen, following literature trends for this purpose. The model was trained on data retrieved from PubChem, a public chemistry database. Data regards drug compounds stored in SMILES notation and labelled with terms about their pharmacological actions.

The obtained embedder was thus tested and compared to the state of the art fingerprint in QSAR, Extended-Connectivity Fingerprint (ECFP), on different datasets for classification tasks using standard machine learning methods. It resulted to outperform ECFP in the task of predicting pharmacological actions used for its training, and to reach better or comparable performances on datasets regarding FDA approved compounds and drugs for targeting brain.

A new Chemoinformatics add-on for Orange was developed. It provides a widget implementing the model and allowing to embed molecular data, and a widget for drawing and visualizing 2D molecular structures. The purpose of this tool is to facilitate data analysis on chemistry data.

The work was carried out at Bioinformatics Lab of University of Ljubljana, under the supervision of Professor Blaž Zupan.

Sommario

La chemioinformatica è una disciplina di recente sviluppo che combina la chimica e l'informatica, che ha trovato particolare applicazione nella scoperta e nella progettazione di nuovi farmaci. L'uso di tecniche *in silico*, infatti, permette di evitare di condurre costosi test sperimentali e di selezionare o scartare nuove molecole sulla base della similarità con molecole target o dei risultati di modelli predittivi.

Il punto cardine in questo tipo di applicazioni, specialmente per quanto riguarda lo sviluppo di modelli quantitativi che leghino struttura ed attività molecolare (QSAR) ed in generale per l'utilizzo di metodi statistici e di machine learning, è il modo in cui le molecole vengono rappresentate. Esse infatti hanno una grande variabilità in termini di dimensioni e complessità e, sebbene le rappresentazioni molecolari standard contengano tutte le informazioni circa la loro struttura e composizione, non sono adatte ad essere utilizzate come input per qualsivoglia modello o algoritmo: è necessaria una rappresentazione numerica e di dimensione fissa.

Le caratteristiche rilevanti di una molecola possono essere estratte e rappresentate mediante vettori di dimensione fissa, detti *fingerprint*, il cui esempio più semplice è costituito da vettori binari nei quali ogni bit corrisponde alla presenza o all'assenza di frammenti predefiniti. Metodi differenti e più complessi sono stati proposti in letteratura, incluso l'utilizzo di tecniche di deep learning.

Il presente lavoro mira a sviluppare un modello per l'embedding, o finger-

printing, di molecole in vettori a valori reali, partendo dalla loro notazione testuale, detta SMILES. Lo scopo finale è poi di implementare tale modello in Orange, una suite per il data mining sviluppata in Python.

É stato scelto di utilizzare un modello di deep learning, in particolare una rete neurale convoluzionale, sulla base dei trend in letteratura. Tale modello è stato allenato su dati estratti da PubChem, un database pubblico che raccoglie una vasta collezione di dati molecolari, riguardanti composti farmaceutici salvati in notazione SMILES ed etichettati con termini relativi alla loro azione farmacologica.

Il modello così ottenuto è stato poi testato e comparato con ECFP, un tipo di fingerprint che rappresenta lo stato dell'arte nella modellizzazione struttura-attività. La comparazione è stata effettuata su diversi dataset e per diversi problemi di classificazione, utilizzando algoritmi standard di machine learning. I risultati hanno mostrato come il modello sviluppato abbia performance migliori nella predizione dell'azione farmacologica, così come su due dei tre dataset su cui è stato valutato.

Infine, è stato sviluppato un nuovo "Chemoinformatics add-on" per Orange, che mette a disposizione uno widget per la conversione di molecole utilizzando il modello qui proposto, ed uno widget per la visualizzazione della struttura molecolare 2D, allo scopo di semplicare l'analisi di dati di origine chimica.

Il lavoro è stato sviluppato presso il Laboratorio di Bioinformatica dell'Università di Ljubljana, sotto la supervisione del Professor Blaž Zupan.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis deals with one of the most crucial issues in Chemoinformatics: representing molecules and their chemical structures and features. This is the basis for solving typical tasks Chemoinformatics is related with, such as the construction of mathematical models in QSAR (Quantitative Structure-Activity Relationships), the calculation of molecular similarity for substructure searching and screening and, in general, the managing of large molecules databases in an efficient manner.

In this chapter, related methods and basic principles will be presented. Firstly, standard techniques in molecular representations are described. Then, deep learning concepts will be introduced, along with its state of the art applications to molecular fingerprinting.

Finally, the purpose of this work is presented: to develop a deep learning model to embed molecules starting just from their textual representation.

1.1 Molecular Representations in Chemoinformatics

Advances in computational techniques and capabilities have altered the way pharmaceutical research is carried out. This particularly has benefited the field of drug discovery and design. Chemoinformatics *in silico* methods like virtual screening and molecular similarity searching enable the researchers to drastically reduce the number of molecules and compounds to be experimentally tested, avoiding great costs and saving time. There are several techniques that can be used to conduct a screening, but the basic concept is to search large molecules databases (commonly several hundred thousands of entries) and identify those compounds that are similar to a reference one, already known to be active or to have some properties of interest [2].

Another common approach in drug design is the construction of models which allow to relate the observed activity or properties to the molecular structure. These models are referred to as QSAR (Quantitative Structure-Activity Relationships) or QSPR (Quantitative Structure-Property Relationships) [5]. A variety of models can be used for this purpose: if initially conceptualized and applied to small series of compounds using simple regression methods, QSAR modelling has now grown and evolved to the analysis of very large datasets using a wide variety of statistical and machine learning techniques [6].

What is fundamental in applying these methods is a proper computational representation of the molecule. This is quite obvious in QSAR, since -as in machine learning in general- only choosing the right set of descriptors (i.e. *features*) of the molecule an accurate and significant model can be derived. It is also very important in virtual screening and similarity searching: databases containing thousands to millions of compounds need to be screened fast and efficiently.

Hence, a variety of methods for molecular representations has been developed: in the following the more relevant for this work are described.

1.1.1 Molecular Graphs

Molecular graphs are based on mathematical graph theory. A graph is an abstract structure that contains nodes connected by edges. This can be easily applied to molecules, considering atoms as nodes and bonds as edges. Besides, some properties can be associated with them, like the atomic number or type.

It is thus necessary a mean to store and communicate graphs. Connection tables are a common method: its simplest type consists of two lists, one containing atomic numbers or symbols and the other containing the bonds, specified as couples of bonded atoms. More detailed forms of connection tables can include further details, such the hybridisation state of the atoms or bond order. Figure 1.1 shows an example of connection table for aspirin in the MDL format. This notation can store a variety of information, although it is long and not easily interpretable.

An alternative way to represent molecular graphs is through linear notations. A linear notation consists of a sequence of alphanumeric characters, it is more compact and easier to read. Different of these representation methods

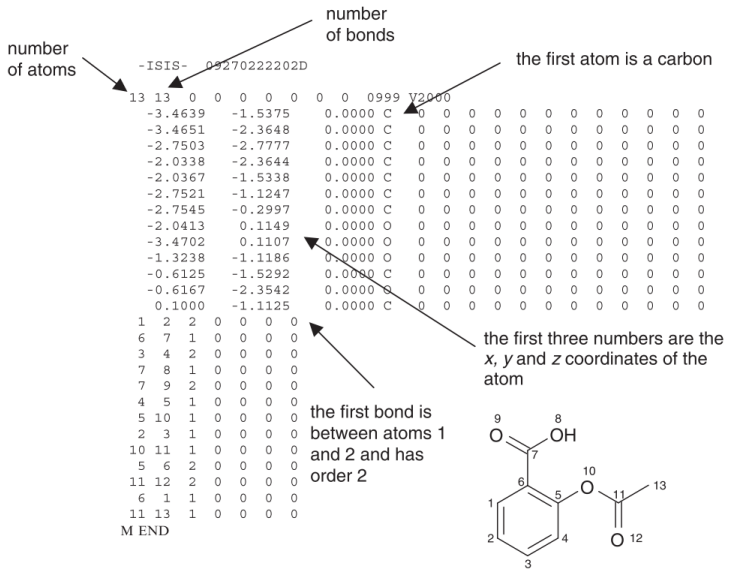


Figure 1.1: The connection table for aspirin in the MDL format [1].

have been proposed, the most common and accepted is the Simplified Molecular Input Line Entry Specification (SMILES).

1.1.2 SMILES

SMILES was introduced in the late 1980's by Weininger [7]. Since then it has become widely used and practically a standard due to its ease of use and comprehension with respect to the previous proposed notations [1]. It is based on graph theory and allows a rigorous structure representation with the use of a very few and natural rules.

Atoms are represented by their atomic symbols, using upper case for normal atoms and lower for aromatic atoms. Hydrogen can be explicitly represented too, but normally they are not. For atoms in the "organic subset", (B, C, N, O, P, S, F, Cl, Br, I), just their symbol letter can be included, otherwise they have to be described in brackets. The formal charge and the number of attached hydrogen atoms have to be specified inside the brackets. The simplest SMILES example is that for methane (CH_4): C.

Double and triple bonds are described respectively with "=" and "#", while single ("-") and aromatic (":") bonds are specified only in some exception case. Thus, for linear structure SMILES can be obtained just omitting hydrogens from the conventional diagrammatic notation. For example acetylen, H−C≡C−H, becomes C#C.

Branches are represented by including them in parenthesis, while cyclic structures are described breaking one specific bond in the ring. The bond is then numbered by a digit after each of the two atoms forming it.

Additional information can be included in the notation, such as chirality and geometrical isomerism. Chiral atoms are specified with "@" and "@@", while *cis-trans* isomerical forms are indicated using slashes. Some examples of SMILES notations for different quite complex molecules are shown in Figure 1.2.

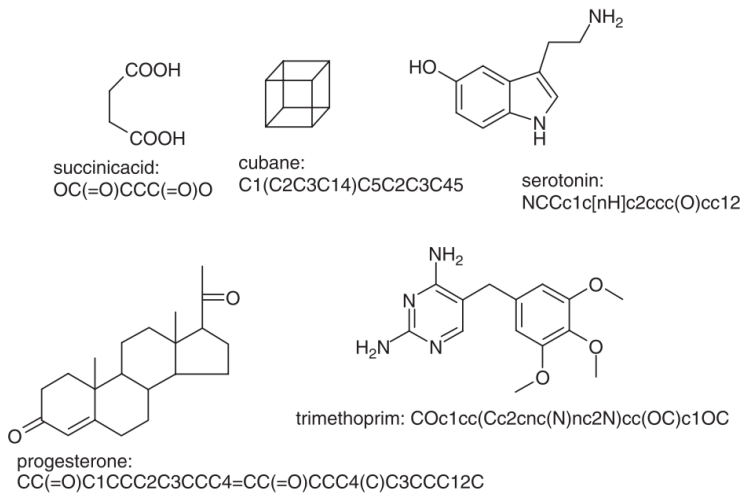


Figure 1.2: Examples of SMILES strings for some complex molecules [1].

The basic encoding rules presented make it clear why this notation has become a standard for its simplicity and comprehensibility. Last but most important thing to note is that, as described far now, SMILES are not unique. The string can be written starting at different location, or breaking the rings at different atoms, thus generating a bunch of equivalent SMILES notations. This is handled by defining a canonical representation [8], based on a unique ordering of the atoms in the molecule structure. A well-established method to do so is the Morgan algorithm [9]: it iterative computes a connectivity value for each atom and uses it to order them.

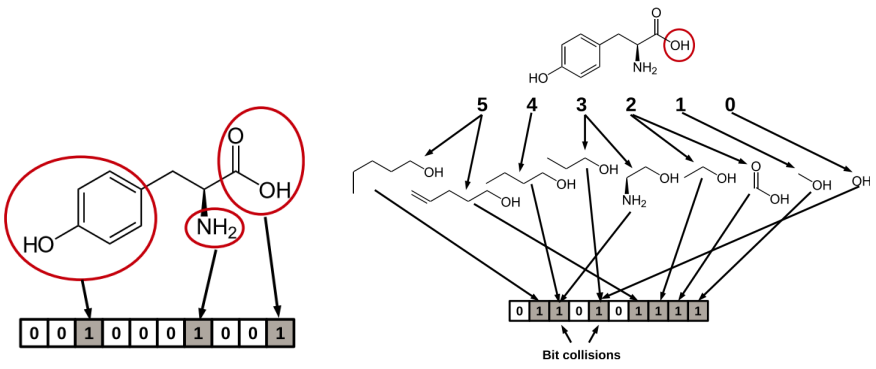
1.1.3 Fingerprints

SMILES has been shown to be the *de-facto* standard for representing molecular structures. It is a very compact and useful notation for storing and communicating all the possible information about structure, but it is not enough for most of Chemoinformatics applications. Thinking of the tasks briefly described in the introduction of this section, this is immediately evident. How can a molecular similarity measure be determined comparing two strings? Or how can a QSAR -or any other machine learning- model be constructed using

as input a string? A more computational representation is necessary. The simplest and more natural encoding for a computer is a binary vector, or bitstring, where each element can be set to two values, "0" and "1". In Chemoinformatics these vectors are referred to as molecular fingerprints, and several types of them have been proposed in literature. The main approaches can be classified as substructure keys-based fingerprints, topological or path-based fingerprints, and circular fingerprints.

In substructure keys-based fingerprints, the bits of the vector state the presence or absence of a certain substructure or fragment in the molecule, as shown in Figure 1.3(a). It is thus necessary to define a dictionary of relevant substructures to include in the fingerprint, and the number of bits is determined by the number of these selected features. For example, the most widely used fingerprint of this type is MACCS [10] with 166 substructural keys (a 960-bits version also exists).

Topological or hashed fingerprints do not require a fragment dictionary. The first step to encode the molecule is generating all the possible linear paths up to a defined length (usually up to seven) starting from each atom. In Figure 1.3(b) this process is shown for the circled atom. Then, each of these patterns is hashed: it serves as a seed to a pseudo-random number generator



(a) Substructure keys-based (b) Topological

Figure 1.3: Examples of hypotetical 10-bits fingerprints [2].

that outputs a set of bits (typically four or five). This is thus added to the fingerprint with a logical OR. Figure 1.3(b) also shows a characteristic of this type of fingerprint, which is bit collisions. Being each bits set generated by a pseudo-random generator, it is likely that some bits can be set by more than one path. However, it is much less likely that all of the bits set by one path will be set by a different one too. The described method is the one developed by Daylight Chemical Information Systems Inc. for its Daylight fingerprint [11], that consists of up to 2048 bits and considers all possible linear patterns up to length of seven atoms.

Finally, circular fingerprints are also topological fingerprints, but they are different because instead of looking for paths starting from an atom, the patterns considered are the environments of each atoms up to a determined radius. Extended-Connectivity Fingerprints (ECFPs) [12] are the *de facto* standard circular fingerprints [2] and they are specifically designed for structure-activity modelling. They are commonly used considering a diameter of four neighbour atoms, and so referred to as ECFP4.

1.2 Deep Learning and Embedding

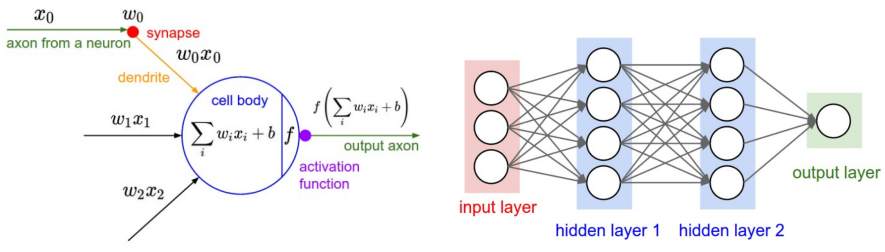
Machine learning methods have been developed from the necessity to tackle those tasks that are easy for people to solve but hard to describe formally, because they involve a great amount of subjective and intuitive knowledge. Thus, it is difficult to input a computer with all the information and a formalized sequence of instructions (an algorithm) needed to solve such tasks. Instead, what machine learning aims for is to develop models with the capability of extract patterns from raw data and use them for classification, prediction or decision making.

What is crucial for the performances of these algorithms is the representation of the input data: most of the efforts in building a model are dedicated to select the most significant features for the problem and to transform the data

properly. However, for many tasks it is difficult to know beforehand which features should be considered. This is for example the case of genomics, where the expressions of thousands of genes are considered, or of image detection, where the features needed to recognize an object have to be described in terms of pixels.

Deep learning solves this, introducing higher-level abstract representations expressed in terms of simpler representations. This can achieve great power and flexibility. Deep learning models are vaguely inspired by biological information processing and communication patterns in the nervous system, so they are commonly referred to as "neural networks". The simplest example of such a network is the multilayer perceptron (MLP). MLP is basically a mathematical function, mapping input data to output values. This function is formed composing many simpler functions each on the top of the other, and each application of a different function provides a new representation of the input data. In this sense this learning process is referred to as "deep", for the number of layers through which the data are transformed [13, 14].

In a more detailed way, the basic computational unit of a neural network is the neuron. Mathematically, it represents an activation function that produces an output based on the weighted sum of the inputs (plus a bias), as exemplified in Figure 1.4(a).



(a) (b)

Figure 1.4: Model of neuron activation and basic neural network architecture [3].

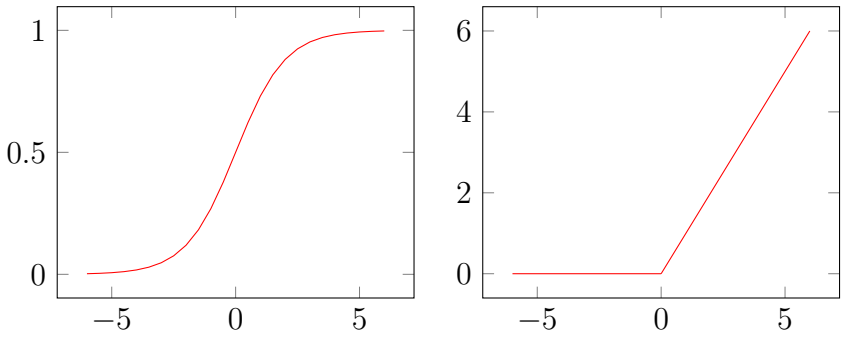


Figure 1.5: Sigmoid and ReLU Activation Functions.

Typical examples of activation functions are sigmoid (Figure 1.5(a)), that takes a real-valued input and squashes it into the range between 0 (non-activated) and 1 (activated), and ReLU (Figure 1.5(b)), that computes the function $f(x) = max(0, x)$ and that has become very popular because it has been found to greatly accelerate the training phase of the network [15].

These basic units are then organized in layers, so that the outputs of the neurons in a layer become the inputs for the following layer (Figure 1.4(b)). The last layer (output layer) represents the results for the task the network is designed to solve. For example, for classification this can be one single neuron scoring the class probability, or multiple neurons scoring different classes probabilities for a multi-classification problem, or even a real-valued output for regression problems.

The neurons of each layer are fully connected to the neurons of the previous and following layers. A weight is assigned to each of these connections, and the learning process essentially consists in updating the weights of the network in order to optimize a cost function that measures the error between the output of the network and the desired output (typically the labels assigned to input examples).

Backpropagation algorithm is used for training neural networks. Back-

propagation uses gradient descent optimization algorithm to update neurons weights, by calculating the gradient of the cost function. The term backpropagation thus refers to the fact that the error (the cost) is calculated at the output and distributed back through the network layers.

Besides solving classical machine learning tasks as classification, regression or prediction making, deep learning is particularly useful for embedding. Embedding is the transformation into number vectors -typically real-valued- of features that otherwise can be difficultly handled by algorithms. For example, it is widely used for images, that are 2D matrices of pixels that need to be transformed into vectors, or in natural language processing, where words are embedded into numbers.

1.2.1 Convolutional Neural Networks

Convolutional Neural Networks (CNN) [16] were originally developed for image recognition but are now commonly used in a wide range of applications and are considered the state of the art in deep learning, also because it was the first method to achieve human-competitive performance on certain tasks [17].

The main distinguishing feature of this kind of networks is the presence of convolutional layers. A convolutional layer takes the input data, convolutes them with a kernel -or filter-, and outputs a feature map.

Input data need to be structured, for example images, that have a 2D structure (or even 3D if RGB channels are considered). The kernel corresponds to a set of weighted connections that, connected to a region of the input, convolves its values and activates neurons in the following hidden layer (i.e. feature/activation map). Figure 1.6 shows how convolution is performed: each value in the feature map is obtained moving the filter across the input matrix by a predefined step size (stride). The size of the kernel and the stride are hyper-parameters, while weights are kept constant while moving the filter. Since these weights are what is gradually updated, or learned, by the CNN

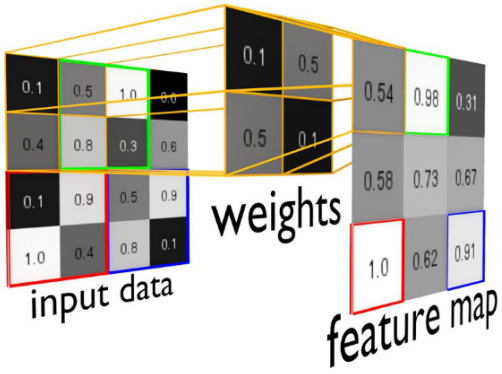


Figure 1.6: Convolution of the input and resulting feature map [4].

during the training, this results in a reduction of the overall number of free parameters and computational costs. The feature map represents higher-level learned features. Each convolutional layer can be composed of multiple filters, resulting in multiple feature maps each of which captures a different abstraction of the data.

Another peculiarity of CNN is the insertion of pooling layers between convolutional layers. Pooling operates a subsampling of the feature map and is used to progressively reduce the spatial size of the representation and the amount of parameters in the network, and for controlling overfitting. MAX pooling is most commonly implemented: a 2*2 filter is applied to a feature map with a stride of 2, and the maximum of the four values in the region is taken.

On the top of a series of convolutional and pooling layers there is a fully connected layer. Neurons in a fully connected layer have connections to all activations in the previous layer, as seen in regular neural networks. This can be an output layer representing the results of the task for which the network is trained, or, if inserted before the last layer, its neurons activations can be considered as an embedded representation of the input.

A fully connected layer occupies most of the parameters, since it is typically connected to a large number of feature maps present in the previous

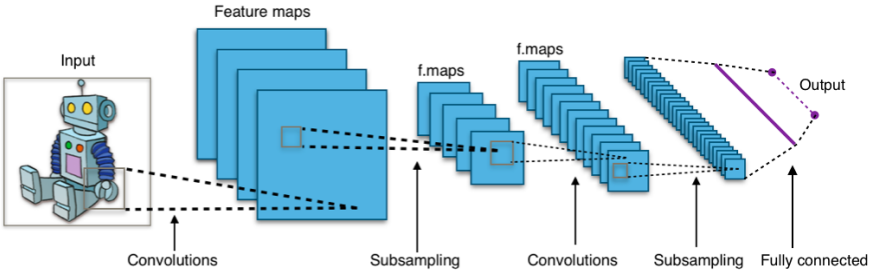


Figure 1.7: CNN Architecture Example.

layer, thus it is prone to overfitting. One method to reduce it is dropout: at each training stage, some nodes are "dropped out" with a certain probability. Incoming and outgoing connections to a dropped-out node are also removed, so that the related weights are not updated. This method decreases overfitting since it avoids training all nodes on all training data. For this reason, it also significantly improves training speed. An example of CNN architecture for image recognition is shown in Figure 1.7 [3, 4, 13, 14].

1.3 Deep Learning in Chemoinformatics

Deep learning may be a particularly well-suited approach for data mining in chemistry and in life sciences in general, because it well deals with large amounts of complex data. Although this, it has never been intensively used in Chemoinformatics and drug discovery and it is an emerging approach in this field [4].

One of the tasks where deep learning techniques may represent a great advantage is molecular fingerprinting. As introduced at the beginning of this chapter, lot of efforts in Chemoinformatics are made to derive significant number vector, length-fixed representations of molecules on which predictive models can be built. Deep learning goes beyond standard fingerprints because it can automatically learn and extract the most relevant features to represent the molecule.

In 2015, *Duvenaud et al.* [18] developed a convolutional neural network that operates on molecular graphs. This architecture is a generalization of standard circular fingerprints (ECPFs). These neural network derived fingerprints offers different advantages over fixed fingerprints:

- by using data adapting the to the specific task, these fingerprints can provide better predictive performance than fixed one. Authors showed that they beat or matched standard fingerprints on different datasets;
- only relevant features are automatically included, while fixed fingerprints consider all the possible substructures;
- in standard fingerprints, each possible fragment is encoded distinctly, with no notion of similarity between fragments. In contrast, each feature of a neural graph fingerprint can be activated by similar but distinct molecular fragments, making the feature representation more meaningful.

In their 2016 work, *Kearnes et al.* [19] achieved results comparable to previous ones by using a simpler network design that was not based on ECFPs. They also used a simpler input featurization containing only atom type, bond type, and graph distances.

Coley et al. [20] developed QSPR models for the prediction of various chemical properties (aqueous solubility, octanol solubility, melting point, and toxicity), using a CNN to derive the features to input a regression model. This work differs form the previous ones in the fact that they used a more information-rich input representation, including, besides molecular graph, atom-level features calculated over the entire molecule, rather than using purely local atomic features. They demonstrates a substantial improvement in model predictive performance for the four property targets listed before. They also showed that with their input featurization large datasets are not needed. In the case of octanol solubility, appropriate molecular representations and their relationship to solubility were found with fewer than 200 training examples.

A complete different approach was followed by *Jastrzębski et al.* [21]. Instead of use graph representation as input, they applied deep learning and natural language processing techniques directly to the textual representation of the molecule, i.e. SMILES. They used of CONTEXT [22], a CNN state of the art method in sentiment analysis. This analogy is motivated by the fact that a molecule can be considered as a sentence composed by different connected clauses (substructures). Conducted experiments shows that this method could outperform state of the art methods based on structural fingerprints in ligand-based virtual screening task, and suggests that the analogy with sentiment analysis is correct and it is possible to learn directly from the raw string representation.

1.4 Aims of the Thesis

This work main goal is to develop an embedder that converts molecular SMILES representations into real-valued vectors. Following the state of the art in this field, the model choice for this purpose is a convolutional neural network.

Data for training the network are collected from PubChem [1][23], a public repository for information on chemical substances and their biological activities. They consist of drug compounds labelled with pharmacological action terms, so that the learning process of the model is supervised.

The obtained embedding model is then compared to standard fingerprinting method in QSAR, ECFP, using standard machine learning methods (logistic regression and random forest), on both data used for training and different data.

Finally, the aim is to implement the deep model in a molecular embedding tool, along with a molecule visualizer tool in order to create a Chemoinformatics add-on for Orange [2] [24].

[1] https://pubchem.ncbi.nlm.nih.gov

[2] https://orange.biolab.si/

Chapter 2

Methods

In this chapter, data used in this work will be described, along with their preprocessing. Then, the CNN architecture and the supervised classification problem chosen for its training will be presented. Finally, it will be shown how the learned model can be used for molecular embedding.

All the work, from retrieving and analyzing the data to the training and test of the neural network was carried out using Python and its machine learning and deep learning libraries such as Numpy [25], Pandas [26], Sci-kit learn [27] and Keras [28]. Data, codes and IPython notebooks describing the work are available on GitHub[1].

2.1 Data Retrieval

2.1.1 PubChem

PubChem [23] is a public chemistry database maintained by National Institutes of Health (NIH). PubChem is organised in three interlinked databases: Substance, Compound and BioAssay.

[1] `https://github.com/mat-cor/molecules-classification`

PubChem Substance contains information provided by individual contributors about a particular chemical substance, hence different Substance records could provide different information about the same chemical structure.

PubChem Compound, instead, is derived from the chemical structure contents found in the Substance database. A Compound record is a unique normalized structure representation of a chemical found in one or more contributed Substance descriptions, thus aggregating information from different Substance data providers. This database counts ∼92 million entries[2]. Each record contains a variety of information about the compound, such as 2D structure and 3D conformation, all the known possible identifiers, chemical and physical properties and descriptors and lot of further information about vendors, pharmacology, patents, toxicity and so on.

For the purpose of this work, compounds annotated with MeSH Pharmacological Classification have been considered and retrieved. PubChem Compound contains ∼15 000 records[2] with pharmacological actions terms associated. As an example, for Aspirin the following terms are annotated:

- Antipyretics
- Cyclooxygenase Inhibitors
- Platelet Aggregation Inhibitors
- Anti-Inflammatory Agents, Non-Steroidal
- Fibrinolytic Agents

2.1.2 MeSH Pharmacological Actions Terms

Medical Subject Headings (MeSH)[3] is a hierarchically-organized terminology provided by U.S. National Library of Medicine (NLM) and used for indexing, cataloging and searching for biomedical and health-related information

[2] At October 2017

[3] `https://www.nlm.nih.gov/mesh/`

and documents. MeSH is organized in a tree structure. There are 16 main categories: category A for anatomic terms, category B for organisms, C for diseases, D for drugs and chemicals, etc. Each category is further divided into subcategories, in which the descriptors are again arrayed hierarchically from most general to most specific. Every descriptor in this ontology is identified by a "tree number" (each level of the hierarchy is coded by three digits), representing its location in the tree.

The terms of interest for this work are those related to pharmacological actions, identified by the root "D27.505".

2.1.3 PubChemAPI: Programmatic Access to PubChem

PubChem is open-accessible and most of its data are available. It offers different methods for download them, for example via FTP or through programmatic interfaces. The first way was not considered viable, since it would have required to download the entire Compound database and then search for the compounds annotated with pharmacological actions. It was thus chosen to access PubChem in a programmatic manner through *E-Utilities*.

Entrez Programming Utilities (*E-Utilities*) [29] are a set of server-side programs that provide a stable interface into the Entrez system at the National Center for Biotechnology Information (NCBI). The Entrez system comprises 38 different databases, including PubChem and MeSH ontology.

The *E-Utilities* use a fixed URL syntax to search for and retrieve the requested data. A Python library named *PubChemAPI* has been developed to perform these operations and collect the data. The code and a usage example[4] are available on GitHub and are reported in Appendix A.

Accessing PubChem Compound in this way allowed to search for only the compounds with pharmacological terms associated. Data searching and re-

[4] `https://github.com/mat-cor/molecules-classification/blob/master/notebooks/pubchem-api.ipynb`

```
<PharmActionList>
        <string>Cyclooxygenase Inhibitors</string>
        <string>Pharmacologic Actions</string>
        <string>Chemical Actions and Uses</string>
        <string>Enzyme Inhibitors</string>
        <string>Molecular Mechanisms of Pharmacological Action</string>
        <string>Analgesics</string>
        <string>Cardiovascular Agents</string>
        <string>Hematologic Agents</string>
        <string>Peripheral Nervous System Agents</string>
        <string>Platelet Aggregation Inhibitors</string>
        <string>Analgesics, Non-Narcotic</string>
        <string>Anti-Inflammatory Agents</string>
        <string>Anti-Inflammatory Agents, Non-Steroidal</string>
        <string>Antipyretics</string>
        <string>Antirheumatic Agents</string>
        <string>Fibrin Modulating Agents</string>
        <string>Fibrinolytic Agents</string>
        <string>Physiological Effects of Drugs</string>
        <string>Sensory System Agents</string>
        <string>Therapeutic Uses</string>
</PharmActionList>
```

Figure 2.1: The "PharmActionList" node in Aspirin XML summary.

trieval was conducted as follows:

- The ESearch utility is used to retrieve a list of compounds responding to a specific query. Thus, specifying the *pccompound_mesh_pharm* filter allowed to retrieve only the compounds of interest. The actual result of this operation is a list of CIDs (Compound identifiers).

- To retrieve information about the compounds is necessary to use ESummary. This utility takes a CIDs list as input and returns, in an XML format, a summary for each compound in the list. This summary contains all the basic information of interest: compound ID, SMILES and name. It is also present a "PharmActionList" node. As it can be seen in Figure 2.1, however, the terms listed here are much more than that actually associated (for Aspirin, Anti-Inflammatory Agents, Non-Steroidal - Fibrinolytic Agents - Antipyretics - Cyclooxygenase Inhibitors). To retrieve only these terms, it is necessary to link to the MeSH ontology database.

- ELink allows to link together Entrez system databases. Passing as input an ID contained in the origin database, it returns a set of IDs in the destination database that are linked to the input element. Thus, for each compound the CID is passed and the IDs corresponding to the associated MeSH terms are returned. Using ESummary is then possible to finally retrieve the terms and their tree numbers.

There are some limitations in using *E-Utilities*: a limit of 10 000 CIDs can be retrieved using ESearch, so this process has to be iterated to obtain the ∼15 000 pharmacological-annotated compounds. Furthermore, the third step in the presented procedure -linking to MeSH database- is very slow and represents a bottleneck: the time needed to retrieve all the compounds was about 6 hours.

PubChemAPI implements methods for sending HTTP requests in order to use *E-Utilities*, for obtaining and parsing the XML summaries in order to retrieve the desired information, and finally for aggregating them in a Pandas dataframe for further elaborations.

2.2 Data Preprocessing

The dataset obtained as described in the previous section is composed of 15 474 compounds, annotated with 489 different terms. For each of them, CID, SMILES, name and chemical formula are reported, along with associated MeSH pharmacological action terms and their tree numbers. The actual relevant feature are the SMILES strings, that will be the input for the embedding model, and the MeSH terms, that can be considered as labels for the compounds. The others can be considered as meta-features, useful for a chemical interpretation of the data.

Since there are no numerical features, there is no sense in preprocess or tranform the data. However, a filtering of the dataset is still necessary. The following criteria has been applied:

- duplicates rows with same SMILES and terms but with a different compound name have been filtered out;
- terms with absolute frequency < 20 were found not significant to be used as classification target and hence discarded; if a compound was associated only with such terms, it was discarded too;
- only actual pharmacological action terms (i.e. with tree number starting with 'D27.505') have been kept; again, compounds with no such terms have been excluded.

After this filtering process, the dataset is composed of 9 174 records and the number of different MeSH terms is reduced to 191.

2.3 Molecules Embedder

As previously introduced, the aim of this thesis is to develop a tool for embedding molecules starting just from their textual representation. In order to do so, it is first necessary to define a model and a paradigm for its training.

Following literature state of art in Chemoinformatics and in Machine Learning in general, a convolutional neural network has been chosen as model. This method particularly suits the embedding purpose, being capable of learning and extracting significant features at different levels.

Regarding the training of the network, it has been chosen to focus on the problem of predicting pharmacological actions of compounds.

2.3.1 Keras

Keras is a high-level Python library for deep neural networks developing. It runs on top of TensorFlow [30] or Theano [31], two of the most famous open source libraries for high performance numerical computation. Keras allows to

easily design a neural network architecture combining different types of layers and to train it, running both on CPU or GPU.

2.3.2 Pharmacological Actions Classification

Typical Chemoinformatics tasks that involves deep and machine learning methods are classification and regression problems. Regression is essentially about predicting a continuous quantity, such as water solubility or other biological activities, while classification means to assign the input to a discrete class, for example to predict active/inactive compounds. Basing on the number of classes considered, classification problems can be distinguished in binary, multi-class or multi-label. In binary classification, examples are labelled as belonging to a class or not. Multi-class and multi-label problems, instead, involve two or more classes that are, respectively, exclusive or not. Hence, in the case of pharmacological actions terms prediction considered in this work, the classification problem is multi-label, since the compounds can be annotated with more than one term.

2.3.3 CNN Design and Training

Training a neural network means to gradually update its weights in order to minimize a loss or cost function that measures the difference between the output of the network and the actual classification labels of the example. In this work, the focus was on the classification of pharmacological actions, that is a multi-label classification problem, with 191 different labels.

The architecture of the network is reported in Figure 2.2. As it can be seen, the first step consists in converting each SMILES string into an integer sequence. The numbers in this sequence are the keys associated to each symbol in a "vocabulary". With a purpose of generalization and capability to apply the embedder to any possible SMILES, the vocabulary was generated basing on the OpenSMILES project specifications [32] and including all the possible

symbols. These integer sequences are the actual input to the neural network. Hence, input dimension is $1 \times 1\,021$, as it has been chosen to not truncate the longer SMILES in the dataset and to pad with 0 the shorter ones.

The first layer is an "Embedding" layer: it turns positive integers into dense vectors of a fixed size, in this case 64×1. Thus, this layer transforms each integer sequence in a $64 \times 1\,021$ matrix. This step is necessary in order to have a more significant representation to convolute.

The second and the following layers are what makes this neural network *convolutional*. First, the embedded sequences are convolved over a single dimension, with a window size of 3 and a stride of 1, using 32 different filters. The outputs are 32 features maps of dimension $1 \times 1\,019$ (the length is reduced by 2 because no padding was added at the beginning and at the end of the sequences). Then, a MAX pooling layer is used for reducing the number of features. Another convolutional layer and another pooling layer are inserted, finally obtaining 32 features maps of dimension 1×253.

At this point, the feature maps are flattened through a "Flatten" layer: they are transformed into a single dimension in order to obtain a layer of fully connected neurons (in this case, $32 \times 253 = 8\,096$ neurons). A "Dropout" layer with rate 0.25 is interposed to prevent overfitting.

The last two layers are simple fully connected layers, with a dropout with rate 0.5 interposed. The penultimate one has dimension 512 and represents the embedded SMILES. The output layer, with dimension 191, represents the result of the classification, returning the probabilities for each label. For this layer the activation is sigmoid, in order to obtain independent outputs between 0 and 1, representing the probabilities. For all the other layers, the activation chosen was the ReLU function.

For the learning process, it has been chosen binary cross-entropy as loss function and the Adam method [33, 34] as optimizer. Training was run for 100 epochs with a batch size of 64, on GeForce GTX TITAN X GPU.

In this phase, the model was trained on a 70% subsample of the data and

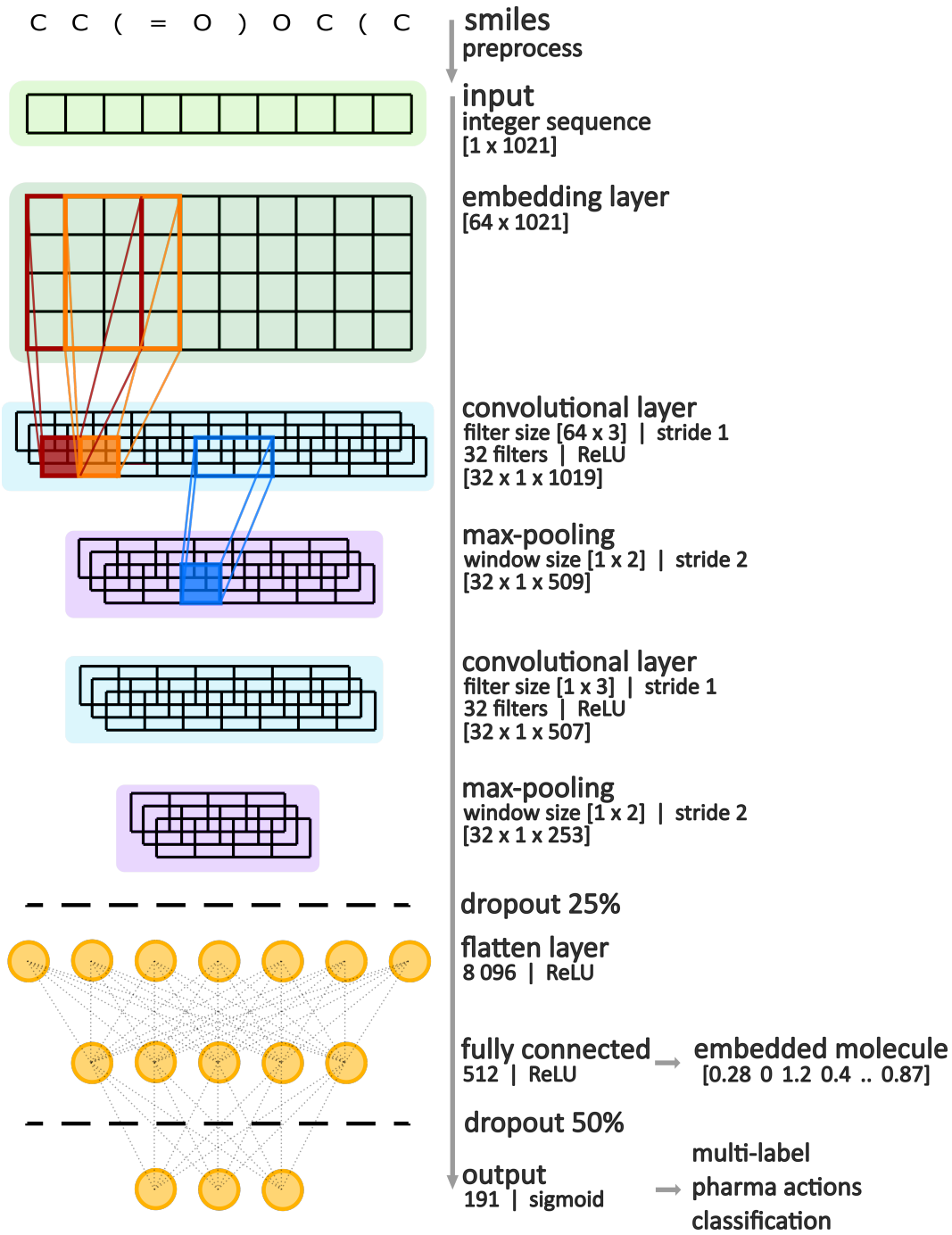


Figure 2.2: CNN Architecture.

evaluated on the remaining 30%, as described in Chapter 3.

2.3.4 Embedder

This work goal was to derive a fingerprint representation for molecules. This can be easily done taking the output of the penultimate layer, that results in a real-valued vector representing the input SMILES. The length of this embedding vector is of 512 bits, real and positive valued, since the activation function chosen for this layer was ReLU.

In order to develop a molecule embedder tool and implement it in Orange (as described in Chapter 4), the network has been trained on the entire dataset. The model has then been saved to an HDF5 file containing its architecture and the learned weights. A Python class has been developed to load the model, load new data and convert SMILES into sequences and then pass them to the embedder and obtain the fingerprints.

Chapter 3

Experiments and Results

In this Chapter it will be presented how and on which data the embedding model (in the following referred to as CNNFP for simplicity) has been evaluated.

First, neural network performance for MeSH terms prediction has been evaluated on a portion of the dataset not used for its training, as usually done in machine learning. Then, the embedder trained on the entire dataset has been compared to standard molecular fingerprints: ECFP has been chosen for this purpose, since it is the state of the art approach in QSAR. The comparison was made using standard machine learning methods on the data used for training, on different data for the prediction of MeSH terms not related to pharmacological action and on other QSAR datasets. Finally, an additional comparison was made visualizing molecules fingerprinted with both methods in the t-SNE space.

3.1 Comparison on MeSH Terms Prediction

3.1.1 CNN Performance

As aforementioned in Section 2.3.3, the neural network was firstly trained on a 70% subsample of the dataset, using the remaining portion of data to assess its performance.

When applying the model to new data, the outputs obtained are the predicted probabilities for each class. In order to obtain categorical labels, with "1s" and "0s" indicating which term is associated and which not, probabilities need to be converted choosing a threshold value. Commonly, 0.5 is used, so that probabilities ≥ 0.5 indicate the belonging to the class. In this case, having a multi-label problem with a large number of labels, thresholds have been optimized separately for each label. For each of them, it was selected the threshold value that maximized Matthews Correlation Coefficient. These values were then used to convert the probabilities.

Performance were thus evaluated for each term separately using Area Under ROC Curve (AUC). This choice was made because each label is very unbalanced (some terms appear just 20 times, out of 2 752 examples in test set), thus classification accuracy would not have been a proper measure. In Table 3.1 minimum, maximum and average AUC values are reported. Table 3.2 lists AUC values for five example terms.

Minimum AUC	**Maximum AUC**	**Average AUC**
0.62	0.99	0.87

Table 3.1: CNN Performance for MeSH Terms prediction.

Term	AUC
5-alpha Reductase Inhibitors	0.94
Adjuvants, Anesthesia	0.91
Adjuvants, Immunologic	0.84
Adrenergic Agents	0.89
Adrenergic alpha-1 Receptor Antagonists	0.96

Table 3.2: AUC values for some example terms.

3.1.2 Prediction of Pharmacological Actions Terms

Besides evaluating the performance of the neural network itself, the real interest for the purpose of this thesis was to compare the embedding model with standard fingerprint, ECFP. The first comparison was performed on the entire dataset. Data were converted to ECFP using RDKit [35], a Chemoinformatic tool further described in Chapter 4. The ECFP 512-long version was used, to be comparable to CNNFP that has the same length. As described in Section 1.1.3, ECFP is a circular fingerprints which bits can assume 0/1 values; CNNFP, instead, is real-valued.

The target was the binary classification of each of the pharmacological actions terms (One-Vs-All approach). The model used is Logistic Regression, with the default parameters defined by Scikit-learn. 10-fold Cross Validation was used for AUC evaluation. Table 3.3 reports average AUC values.

CNNFP highly outperformed standard ECFP, demonstrating how accurate

Fingerprint	Average AUC
CNNFP	0.99
ECFP	0.92

Table 3.3: Fingerprints comparison on pharmacological actions prediction.

the embedded representation learned by the network is.

3.1.3 Prediction of Non-Pharmacological Terms

With the filtering procedure described in Section 2.2 some of the compounds were discarded since not associated with terms related to their pharmacological action. This set of 1 091 molecules can thus be used for further comparison between the two fingerprints. The same evaluation described in the previous Section was performed, and average AUC results are reported in Table 3.4. In this case CNNFP could not outperform ECFP, even if the AUC value reached can be considered good enough.

Fingerprint	**Average AUC**
CNNFP	0.83
ECFP	0.92

Table 3.4: Fingerprints comparison on non-pharmacological terms prediction.

3.2 Comparison on Other QSAR Datasets

To assess its capability of generalization, the embedder has been tested on other datasets, retrieved from MoleculeNet.

MoleculeNet [36] is a large benchmark resource for molecular machine learning. It curates and makes available multiple public datasets, along with implementations of previously proposed molecular featurization and learning algorithms. Datasets and methods are available through the open source DeepChem package [37].

From the datasets available in MoleculeNet, those with a classification (and not regression) target were chosen. DeepChem was used to retrieve the data

Task	Classifier	ECFP	CNNFP	CNNFP+ECFP
CT Toxicity	LR	0.72	**0.93**	**0.95**
	RF	0.74	**0.94**	**0.96**
FDA Approval	LR	0.74	**0.92**	**0.95**
	RF	0.74	**0.94**	**0.97**

Table 3.5: Fingerprints comparison on ClinTox dataset.

and the same comparisons were carried out as described in the previous Sections. In addition to that, a fingerprint obtained by concatenation of CNNFP and ECFP was evaluated. Besides Logistic Regression, also Random Forest was used as classifier.

In the following, datasets and results achieved on each of them are described.

3.2.1 ClinTox

The ClinTox dataset compares drugs approved by the FDA and drugs that have failed clinical trials for toxicity reasons. The dataset includes two classification tasks for 1 491 compounds with known chemical structures: (1) clinical trial toxicity (or absence of toxicity) and (2) FDA approval status.

A shown in Table 3.5, for both tasks and for both classifiers CNNFP highly outperformed ECFP. Furthermore, the combination of both fingerprints resulted in even better performances. The reason may rely in the fact that this dataset is composed of drug compounds, even if only 81 of them were found also in the data used to train the embedder.

Classifier	ECFP	CNNFP	CNNFP+ECFP
LR	**0.85**	0.72	0.81
RF	**0.87**	0.79	0.86

Table 3.6: Fingerprints comparison on BACE dataset.

3.2.2 BACE

The BACE dataset provides quantitative and qualitative binding results for a set of inhibitors of human β-secretase 1 (BACE-1). It consists of 1 522 compounds and qualitative binding results, reported as binary labels, were considered as classification target. No one of the compound in BACE dataset is present also in model training data.

Table 3.6 lists the results for this comparison. In this case, the usage of CNNFP led to worst performances for both classifiers. The reason could be related to the fact that the considered molecules are all similar since they represents inhibitors of different classes of the same enzyme.

3.2.3 BBBP

The Blood-Brain barrier penetration (BBBP) dataset comes from a study on the modelling and prediction of the barrier permeability, that is a crucial issue in development of drugs that can target the central nervous system. This dataset includes 2 000 compounds annotated with binary labels representing their permeability properties.

As reported in Table 3.7, using Logistic Regression CNNFP was slightly outperformed by ECFP, while with Random Forest both fingerprints performance are comparable. For both classifier, however, performances were improved by the usage of CNNFP together with ECFP. Also in this case the reason may be related to the fact that the molecules in the dataset are drugs.

Classifier	ECFP	CNNFP	CNNFP+ECFP
LR	**0.83**	0.79	**0.85**
RF	0.88	**0.89**	**0.91**

Table 3.7: Fingerprints comparison on BBBP dataset.

3.3 Data Visualization

Besides their usage in QSAR modelling and in molecular machine learning in general, fingerprints are widely used also to compute molecules similarity. Hence, a good term for comparing fingerprints could be their capability to distinguish different molecules and aggregate similar ones. In order to perform this comparison, fingerprinted data have been embedded in t-SNE space and visualized, as described in the following.

3.3.1 t-SNE Space

T-distributed Stochastic Neighbor Embedding (t-SNE) [38] is a machine learning algorithm developed for the visualization of high-dimensionality data into a two- or three-dimensional space. It is a non-linear dimensionality reduction technique, based on the modelling of each points pair in the high-dimensional space in such a way that similar points are have an high probability of being picked and dissimilar points have an extremely small probability associated. A similar probability distribution is then defined in low-dimensional space and the divergence between the two distributions is optimized.

3.3.2 Comparison

In order to compare the two fingerprints, besides the three datasets previously used, a sample of the training data was used. The five most frequent MeSH terms and the compounds associated with them were selected, and if a

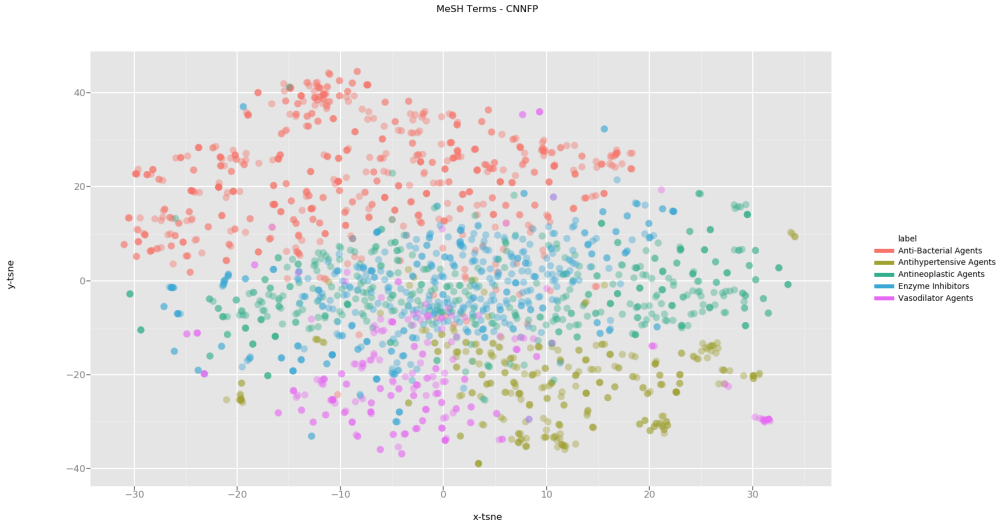
Dataset	CNNFP	ECFP
MeSH Terms	**91%**	52%
ClinTox	**90%**	65%
BACE	**94%**	84%
BBBP	**87%**	67%

Table 3.8: Explained variance for 100 principal components.

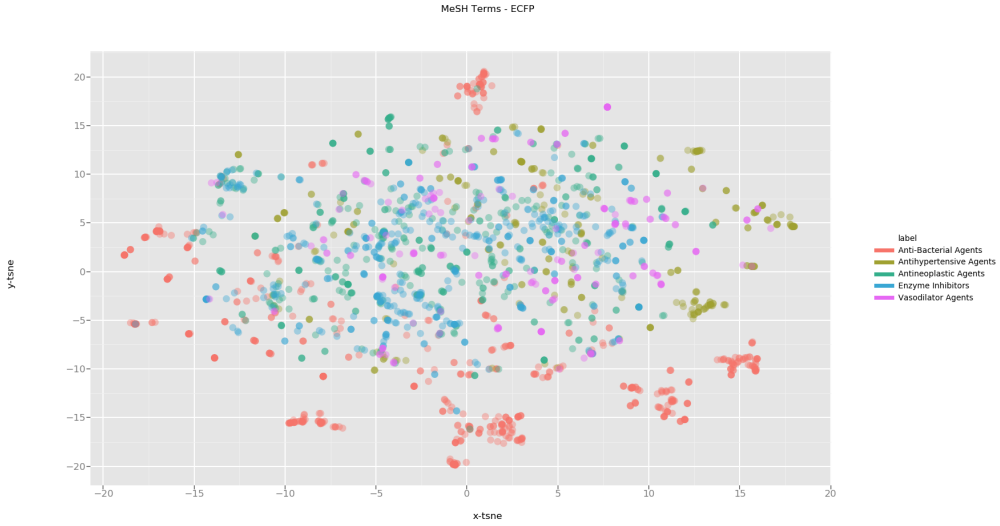
compound was annotated with more than one of these terms, it was discarded.

Before applying t-SNE transformation, if the number of features is too high, is recommended to use another dimensionality reduction method to reduce the number of dimensions to a reasonable amount, in order to suppress noise and speed up the computation [38]. Thus, Principal Component Analysis (PCA) was applied to obtain the first 100 principal components. PCA uses an orthogonal transformation to convert original variables into a set of linearly uncorrelated vectors (principal components). This transformation is defined in such a way that each principal components has the largest possible variance, in order to retain the maximum amount of information about the data. Hence, it is interesting to evaluate the cumulative explained variance for the transformed components. As shown in Table 3.8, the cumulative explained variance for the first 100 principal components is much higher using CNNFP than using ECFP for all the compared datasets.

The t-SNE algorithm was then applied to PCA data to obtain a two-dimensional visualization of the original data. The resulting graphs are reported in Figures 3.1 - 3.4, with points coloured by their associated term or class. With both fingerprint methods data could not be separated in well defined clusters, but using CNNFP it is possible to see a better grouping of same labelled points, especially for the five selected MeSH Terms (Figure 3.1) and for ClinTox (Figure 3.2) and BBBP datasets (Figure 3.4).

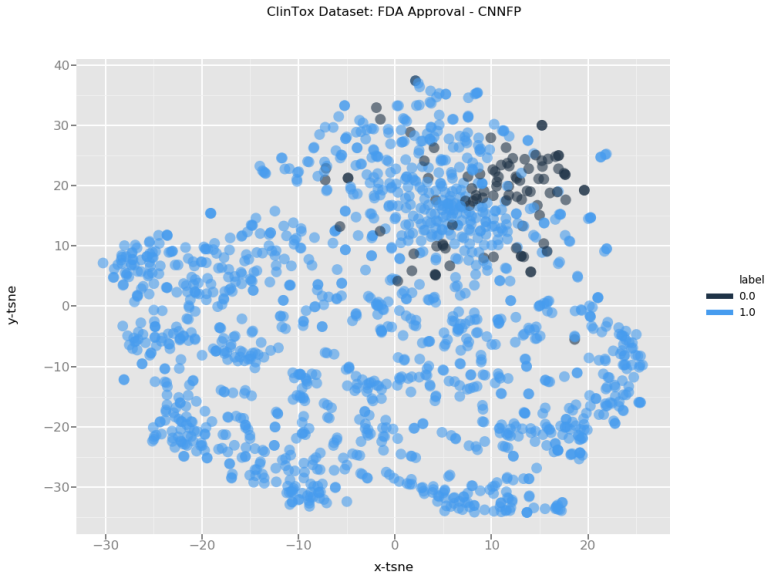


(a)

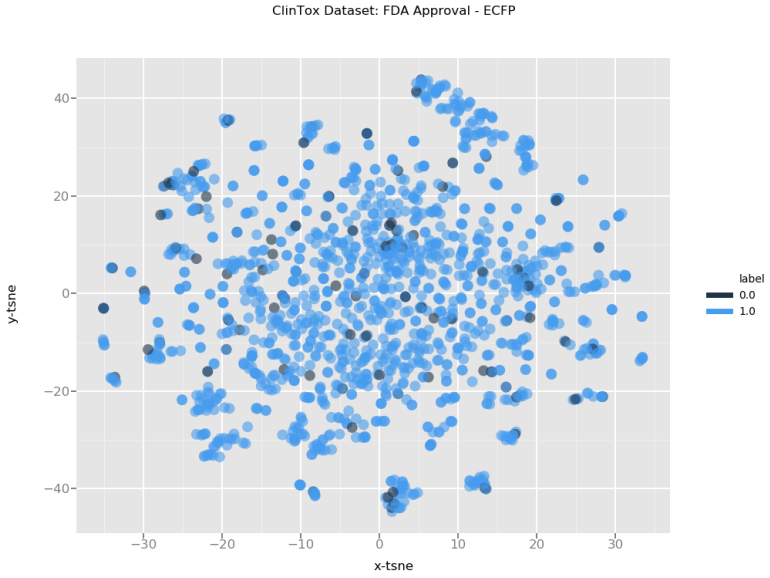


(b)

Figure 3.1: MeSH Terms t-SNE Visualization.

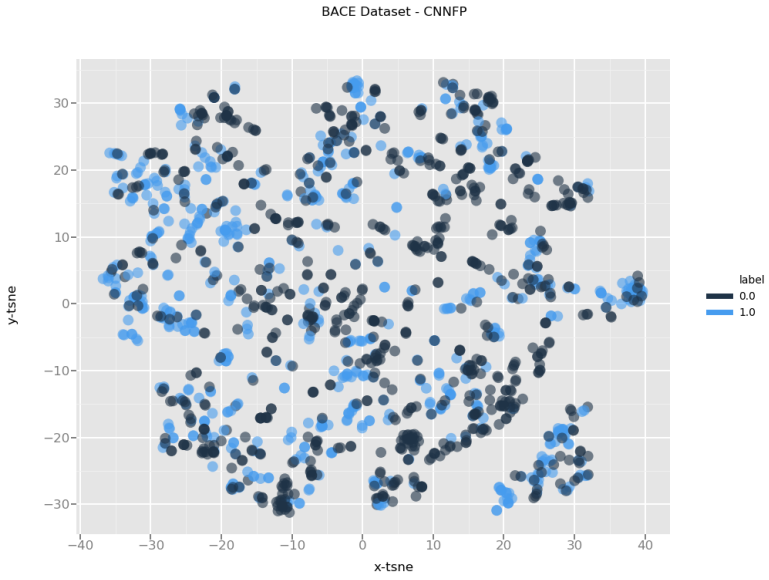


(a)

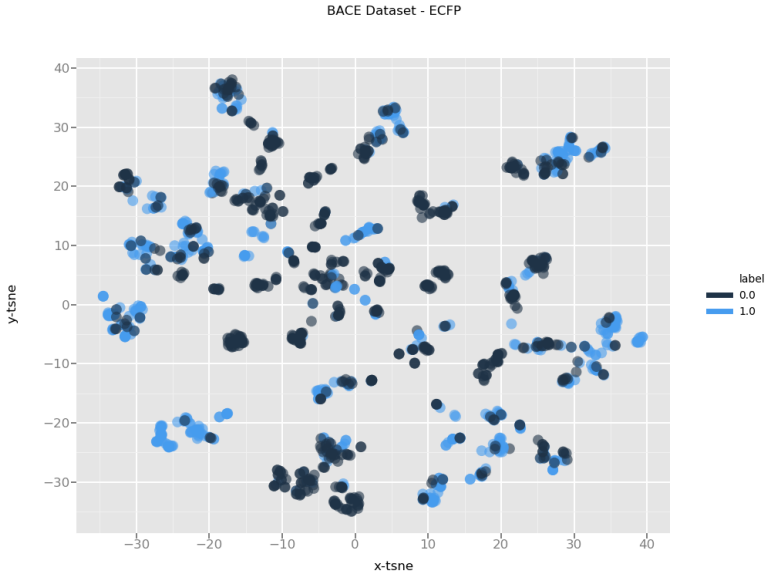


(b)

Figure 3.2: ClinTox t-SNE Visualization.

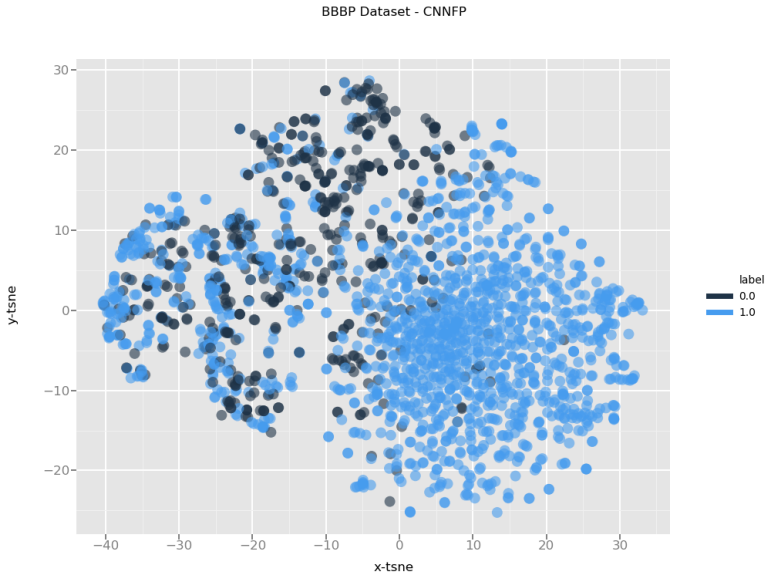


(a)

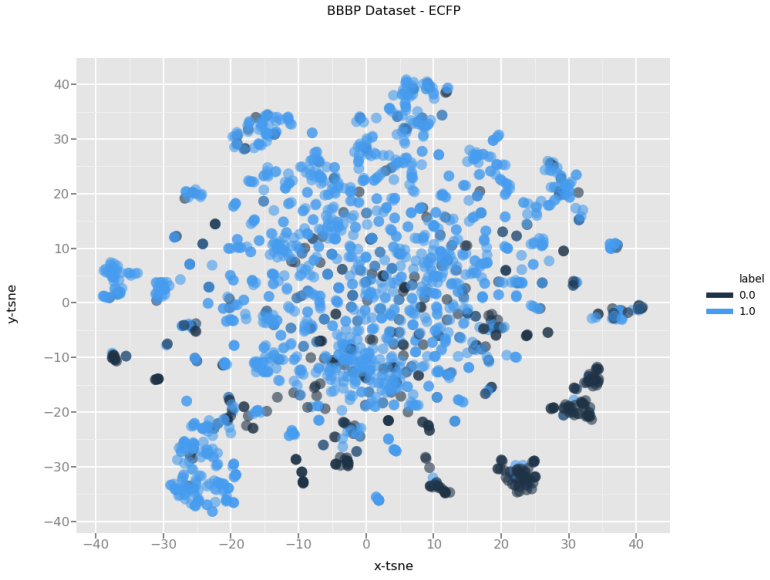


(b)

Figure 3.3: BACE t-SNE Visualization.



(a)



(b)

Figure 3.4: BBBP t-SNE Visualization.

Chapter 4

Software Development

4.1 Orange

Orange [24] is a machine learning and data mining suite developed as an open source project by Bionformatics Lab at University of Ljubljana, Slovenia. Orange main feature is to provide a visual programming tool to easily construct data analysis workflows (Figure 4.1). "Widgets" are the basic components: each of them implements methods for data manipulation and preprocessing, classification and regression, clustering, data projection and evaluation. These components are written in Python and they are both wrappers for C++ classes implementing computationally intensive tasks, as well as scripts that use popular Python libraries for machine learning and scientific computing (e.g. Numpy [25] and Scikit-learn [27]).

Orange core functionalities can be easily extended by add-ons. There are, for example, add-ons for text mining, image analytic and Bioinformatics. As a secondary aim of this work, a new add-on for Chemoinformatics analysis was developed.

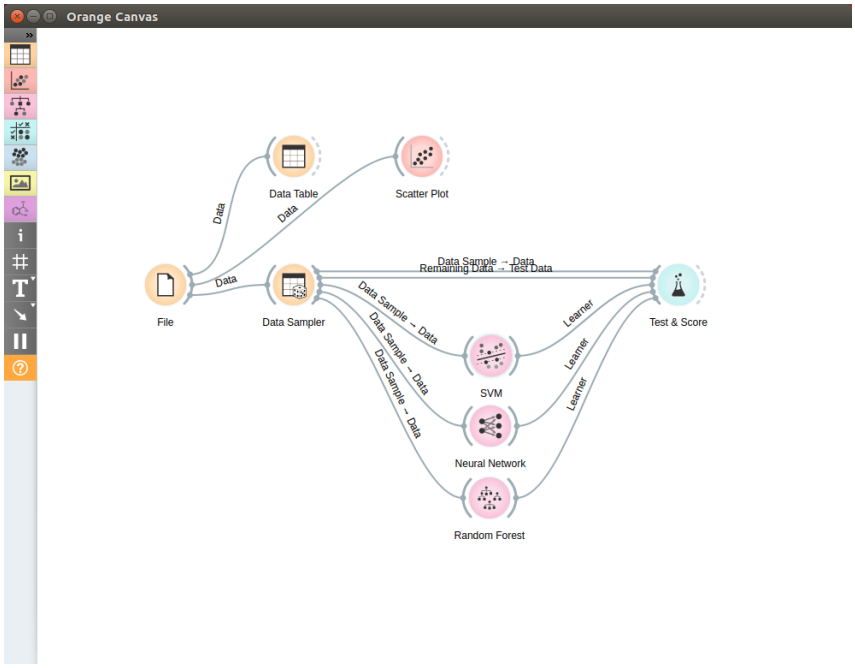


Figure 4.1: Basic Orange workflow for: loading data from any tab- or comma-separated file; inspecting them with Data Table and Scatter Plot widgets; sampling dataset to train and evaluate different learning methods.

4.2 Chemoinformatics Add-on

This add-on, named "Chemoinformatics", was developed in order to provide a tool for easily fingerprinting molecules using the model described in this work, and in general with the aim to facilitate working with chemistry datasets. In this initial version, two widgets are implemented: one for drawing and visualizing molecules structures, the other for molecular embedding. The add-on is available on GitHub[1].

[1] `https://github.com/mat-cor/orange3-chem`

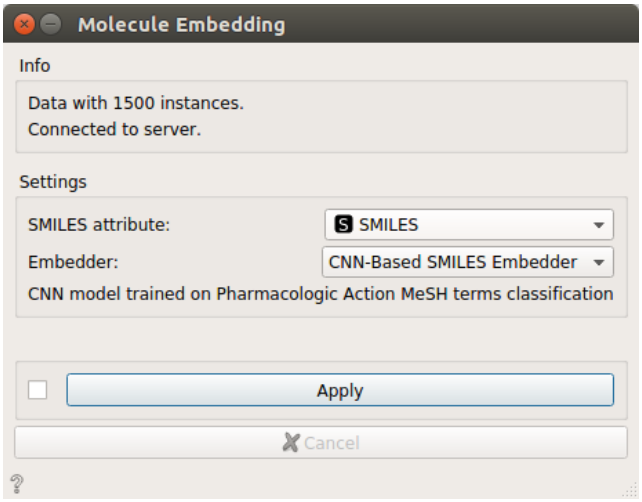
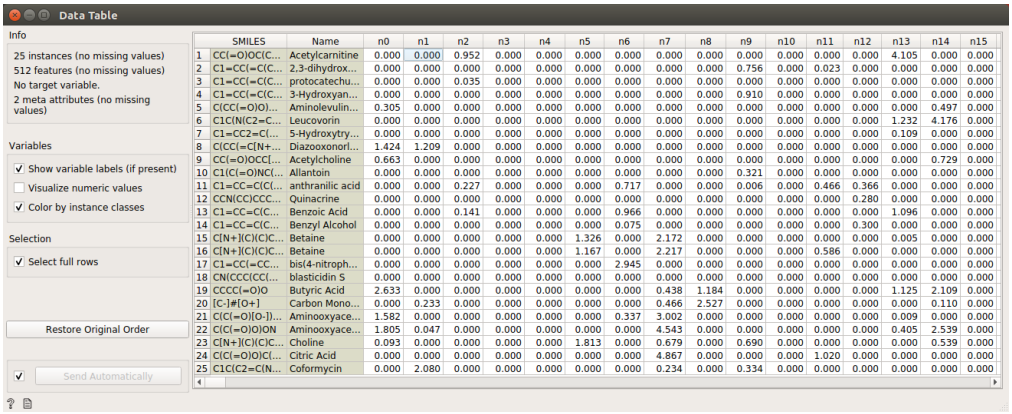


Figure 4.2: Molecule Embedding Widget.

4.2.1 Molecule Embedding Widget

In Chapter 2 it has been described how and on which data the fingerprint model was developed. As explained, Keras library was used to design and train the neural network, that was then saved in the HDF5 format. Hence, to fingerprint new molecules is necessary to load and run the model in Keras, preferably on a GPU for large amounts of data. Furthermore, to input the model SMILES have to be converted into integer sequences according to a pre-defined vocabulary. This vocabulary and the model are saved on Orange server, where it is also implemented a Python class for preprocessing SMILES, load and run the model and return the penultimate layer outputs, that are the desired fingerprints. This back-end runs on a GeForce GTX TITAN X GPU.

When the widget is placed in the workflow and data are passed in input, it parses the dataset columns and presents a selection list to choose the one containing SMILES strings (if the column is named "smiles" is selected by default); a selection list to choose the desired embedder is also present (Figure 4.2). The core class of this widget is *MoleculeEmbedder*: it extends a *HttpClient* class to open and handle the connection with Orange back-end server and to load SMILES and get the results. Once the connection is established, it parses



	SMILES	Name	n0	n1	n2	n3	n4	n5	n6	n7	n8	n9	n10	n11	n12	n13	n14	n15
1	CC(=O)OC(C...	Acetylcarnitine	0.000	0.000	0.952	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	4.105	0.000	0.000
2	C1=CC(=C(C...	2,3-dihydrox...	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.756	0.000	0.023	0.000	0.000	0.000	0.000
3	C1=CC(=C(C...	protocatechu...	0.000	0.000	0.035	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
4	C1=CC(=C(C...	3-Hydroxyan...	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.910	0.000	0.000	0.000	0.000	0.000	0.000
5	C(CC(=O)O)...	Aminolevulin...	0.305	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.497	0.000
6	C1C(N(C2=C...	Leucovorin	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	1.232	4.176	0.000
7	C1=CC2=C(...	5-Hydroxytry...	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.109	0.000	0.000
8	C(CC(=C[N+...	Diazooxonorl...	1.424	1.209	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
9	CC(=O)OC[...	Acetylcholine	0.663	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.729	0.000
10	C1(C(=O)NC(...	Allantoin	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.321	0.000	0.000	0.000	0.000	0.000	0.000
11	C1=CC=C(C(...	anthranilic acid	0.000	0.000	0.227	0.000	0.000	0.000	0.717	0.000	0.000	0.006	0.000	0.466	0.366	0.000	0.000	0.000
12	CCN(CC)CCC...	Quinacrine	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.280	0.000	0.000	0.000
13	C1=CC=C(C...	Benzoic Acid	0.000	0.000	0.141	0.000	0.000	0.000	0.966	0.000	0.000	0.000	0.000	0.000	0.000	1.096	0.000	0.000
14	C1=CC=C(C...	Benzyl Alcohol	0.000	0.000	0.000	0.000	0.000	0.000	0.075	0.000	0.000	0.000	0.000	0.000	0.300	0.000	0.000	0.000
15	C[N+](C)(C)C...	Betaine	0.000	0.000	0.000	0.000	0.000	1.326	0.000	2.172	0.000	0.000	0.000	0.000	0.000	0.005	0.000	0.000
16	C[N+](C)(C)C...	Betaine	0.000	0.000	0.000	0.000	0.000	1.167	0.000	2.217	0.000	0.000	0.000	0.586	0.000	0.000	0.000	0.000
17	C1=CC(=CC...	bis(4-nitroph...	0.000	0.000	0.000	0.000	0.000	0.000	2.945	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
18	CN(CCC(CC(...	blasticidin S	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
19	CCCC(=O)O	Butyric Acid	2.633	0.000	0.000	0.000	0.000	0.000	0.000	0.438	1.184	0.000	0.000	0.000	0.000	1.125	2.109	0.000
20	[C-]#[O+]	Carbon Mono...	0.000	0.233	0.000	0.000	0.000	0.000	0.000	0.466	2.527	0.000	0.000	0.000	0.000	0.000	0.110	0.000
21	C(C(=O)[O-])...	Aminooxyace...	1.582	0.000	0.000	0.000	0.000	0.000	0.337	3.002	0.000	0.000	0.000	0.000	0.000	0.009	0.000	0.000
22	C(C(=O)O)ON	Aminooxyace...	1.805	0.047	0.000	0.000	0.000	0.000	0.000	4.543	0.000	0.000	0.000	0.000	0.000	0.405	2.539	0.000
23	C[N+](C)(C)C...	Choline	0.093	0.000	0.000	0.000	0.000	1.813	0.000	0.679	0.000	0.690	0.000	0.000	0.000	0.000	0.539	0.000
24	C(C(=O)O)C(...	Citric Acid	0.000	0.000	0.000	0.000	0.000	0.000	0.000	4.867	0.000	0.000	0.000	1.020	0.000	0.000	0.000	0.000
25	C1C(C2=C(N...	Coformycin	0.000	2.080	0.000	0.000	0.000	0.000	0.000	0.234	0.000	0.334	0.000	0.000	0.000	0.000	0.000	0.000

Figure 4.3: Data Table showing Molecule Embedding Widget output.

the SMILES, sends them to the server through a *http POST* request and get the response containing the fingerprints. The results are cached locally and a cache key is computed for each SMILES, so that there is no necessity to embed the same data every time they are presented.

The output is a Data Table containing the original attributes with appended the 512 new real-valued features representing the fingerprints (Figure 4.3), allowing to easily carry out any data analysis on them. The code for the *MoleculeEbedder* class is reported in Appendix B.

4.2.2 Molecule Viewer Widget

Molecule Viewer Widget was developed to provide a tool for visual inspecting of chemistry data. The widget draws the 2D structure of the selected molecules, starting from SMILES, and shows them in a viewer. Since SMILES is a linear notation for representing a molecular graph, it retains all the necessary information to depict a molecule structure.

RDKit [35], an open-source toolkit for Chemoinformatics, was used to analyse SMILES strings and derive and draw the molecular structures. Once the Widget is linked, the user has to select the column containing SMILES strings, that are converted by an RDKit method and their depictions displayed. In case

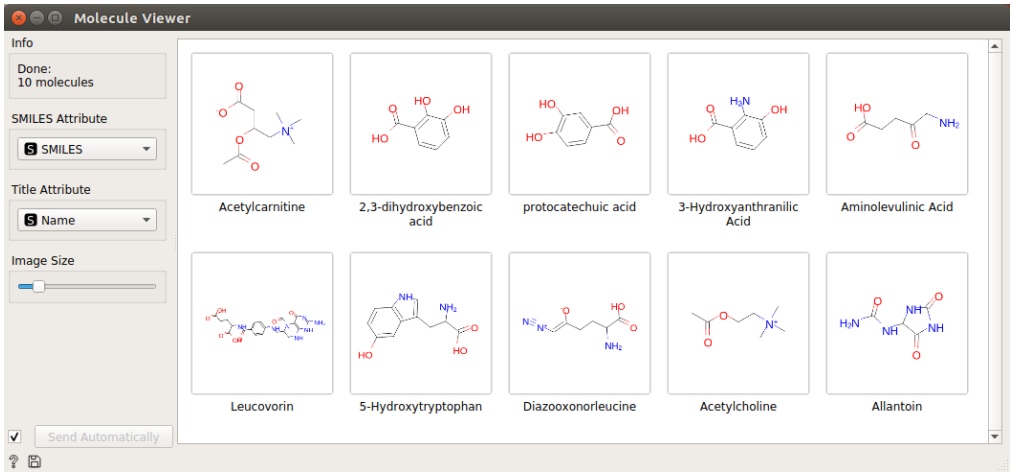


Figure 4.4: Molecule Viewer Widget.

a SMILES is invalid and could not be converted, a blank image is showed. The user can also select which of the other attributes use to label the images. Figure 4.4 shows the Molecule Viewer Widget and its output for 10 molecular structures, labelled by their names.

Chapter 5

Conclusions and Future Developments

5.1 Conclusions

In this thesis a novel molecular fingerprinting model was developed, basing on literature trends in Chemoinformatics and using deep learning methods. Deep neural networks are widely used in a variety of different fields, but their application to drug discovery and design is quite recent. However, these techniques seems promising in being applied to the crucial issue of molecular representation and featurization. The development of this model led to the creation of a Chemoinformatics add-on for Orange, a data mining suite developed in Python. The add-on provides a tool for molecular embedding, implementing the result of this work, and a tool for molecules visualizing.

A first important part of the work consisted in access a public chemistry database (PubChem) to retrieve compounds data and their annotations about pharmacological actions. Then, as in every machine learning application, another important step regarded data filtering and preprocessing.

In order to develop a fingerprinting model, a convolutional neural network was designed and trained in the supervised context of pharmacological actions

prediction. Peculiarity of this architecture is the fact that its input are just SMILES strings, simple textual representations of molecular structures. The result is a model that can embed molecules SMILES into 512-bits real-valued fingerprints.

To assess the quality of this novel fingerprint, it was compared to ECFP, the state of the art method in QSAR. Standard machine learning methods were applied using both fingerprints as input, and CNNFP highly out-performed ECFP in MeSH terms prediction, showing how accurate the featurization learned by the neural network is. This also suggests that the same architecture can be reused and trained on different data to obtain target-specific embedding models. The capability of generalization of the developed fingerprint was then evaluated comparing the two methods on different chemistry datasets. For two out of the three considered datasets, CNNFP could reach better or comparable performance with respect to those reached by ECFP, both when used alone or in combination with ECFP. The reason of that may rely in the fact that both the datasets on which CNNFP performed well are composed by drug compounds, as well as the data used for training. However, only 81 compounds out of 1 491 for the first dataset, and 10 out of 2 000 for the second were found to be present also in the training dataset, hence the good performances have not been biased by this fact. On the third dataset on which the two fingerprints were compared, CNNFP reached slightly worst results than ECFP. The reason in this case could be related to the fact that this dataset contains compounds very similar to each other, since they are drugs to target different classes of the same enzyme.

Finally, the two fingerprints were compared in the visualization of the datasets previously considered. PCA was applied to fingerprinted data in order to reduce dimensionality to a reasonable amount and transform them into two-dimensional t-SNE space. Data points were then coloured by their labels and visualized. CNNFP showed a better capability of representing data, since data with same labels appear better grouped.

5.2 Future Developments

The main focus of this work was on the construction of a molecule embedder, but more attention may be drawn to the problem used for training the model, which is pharmacological actions prediction.

The development of a tool optimized for this purpose, for example by applying another machine learning method on the top of the neural network, may lead to interest application in drug discovery and design. Such a tool may be used for example to test *in silico* and have primary indications about new designed drugs, or it can be used to have indications about alternative uses of existing drug compounds, thus becoming useful for drug repurposing.

Another interesting point to develop regards the features learned by the neural network, exploring the possibility to have a chemical interpretation of them, in terms of significant or recurring molecules substructures.

Finally, there is room for improvements for the Orange Chemoinformatics add-on. Most used standard fingerprinting methods, such as MACCS and ECFP, may be implemented to provide a complete embedding tool. Furthermore, other widgets may be added, for example for substructure searching and virtual screening.

Appendix A

PubChemAPI

A.1 Usage Example

This example shows how the developed *PubChemAPI* Python library can be used to access PubChem and retrieve information about a compound, in this case Aspirin (CID: 2244).

All E-utility calls share the same base URL: `https://eutils.ncbi.nlm.nih.gov/entrez/eutils/`. First of all, the ESearch utility is used to retrieve a list of compounds IDs responding to a specific query (it actually returns two variables, *query_key* and *web_env* to access the results). Then, using ESummary, an XML summary is retrieved with informations for each compound in the ID list resulting from the previous search.

```
db = 'pccompound'
query = '2244[uid]pccompound_mesh_pharm[filt]'
query_key, web_env = e_search(db, query)
summary = get_summary(query_key, web_env, db)
```

The following is the summary returned for Aspirin:

```
<DocumentSummary uid="2244">
        <CID>2244</CID>
        <SourceNameList>
```

```
</SourceNameList>
<SourceCategoryList>
        <string>Chemical Vendors</string>
        <string>Research and Development</string>
        <string>Curation Efforts</string>
        <string>Governmental Organizations</string>
        <string>NIH Initiatives</string>
        <string>Subscription Services</string>
        <string>Journal Publishers</string>
        <string>Legacy Depositors</string>
</SourceCategoryList>
<CreateDate>2004/09/16 00:00</CreateDate>
<SynonymList>
        <string>aspirin</string>
        <string>ACETYLSALICYLIC ACID</string>
        <string>2-Acetoxybenzoic acid</string>
        <string>Acido acetilsalicilico [Italian]</string>
        ...
</SynonymList>
<MeSHHeadingList>
        <string>Aspirin</string>
</MeSHHeadingList>
<MeSHTermList>
        <string>Acetysal</string>
        <string>Acylpyrin</string>
        ...
</MeSHTermList>
<PharmActionList>
        <string>Cyclooxygenase Inhibitors</string>
        <string>Pharmacologic Actions</string>
        <string>Chemical Actions and Uses</string>
        <string>Enzyme Inhibitors</string>
        <string>Molecular Mechanisms of Pharmacological Action</string
            >
        <string>Analgesics</string>
        <string>Cardiovascular Agents</string>
        <string>Hematologic Agents</string>
        <string>Peripheral Nervous System Agents</string>
        <string>Platelet Aggregation Inhibitors</string>
        <string>Analgesics, Non-Narcotic</string>
        <string>Anti-Inflammatory Agents</string>
        <string>Anti-Inflammatory Agents, Non-Steroidal</string>
        <string>Antipyretics</string>
        <string>Antirheumatic Agents</string>
        <string>Fibrin Modulating Agents</string>
        <string>Fibrinolytic Agents</string>
        <string>Physiological Effects of Drugs</string>
```

```
            <string>Sensory System Agents</string>
            <string>Therapeutic Uses</string>
        </PharmActionList>
        <CommentList>
        </CommentList>
        <IUPACName>2-acetyloxybenzoic acid</IUPACName>
        <CanonicalSmiles>CC(=O)OC1=CC=CC=C1C(=O)O</CanonicalSmiles>
        <IsomericSmiles>CC(=O)OC1=CC=CC=C1C(=O)O</IsomericSmiles>
        <RotatableBondCount>3</RotatableBondCount>
        <MolecularFormula>C9H8O4</MolecularFormula>
        <MolecularWeight>180.159</MolecularWeight>
        <MolecularWeightSort>180159</MolecularWeightSort>
        <TotalFormalCharge>0</TotalFormalCharge>
        <XLogP>1.2</XLogP>
        <HydrogenBondDonorCount>1</HydrogenBondDonorCount>
        <HydrogenBondAcceptorCount>4</HydrogenBondAcceptorCount>
        ...
</DocumentSummary>
```

As it can be seen, in the summary it's also present a "PharmActionList" node, but the terms listed here are much more than the actually showed in PubChem (Anti-Inflammatory Agents, Non-Steroidal - Fibrinolytic Agents - Antipyretics - Cyclooxygenase Inhibitors). To retrieve only the actually linked MeSH terms, it's necessary to link to the MeSH DB. For each CID, the MeSH IDs of the associated terms are retrieved. With the ID, it's then possible to get the Term name and the associated tree numbers. This can be easily done with the method *get_mesh_info_from_cid()* implemented in *PubChemAPI*.

```
get_mesh_info_from_cid('2244')
```

The result is:

```
['D27.505.696.068', 'D27.505.519.389.310', 'D27
    .505.696.663.850.014.040.500.500', 'D27.505.954.158.030.500', 'D27
    .505.954.329.030.500', 'D27.505.954.502.780', 'D27
    .505.519.421.750', 'D27.505.954.411.320', 'D27.505.954.502.427', '
    D27.505.696.663.850.014.040.500', 'D27.505.954.158.030', 'D27
    .505.954.329.030']
['Antipyretics', 'Cyclooxygenase Inhibitors', 'Platelet Aggregation
    Inhibitors', 'Fibrinolytic Agents', 'Anti-Inflammatory Agents, Non
    -Steroidal']
```

A.2 Code

Full code for the *PubChemAPI* class.

```
"""
Class pubChemAPI for accessing and retrieving molecules information from
    pubChem
"""
import requests
from html.parser import HTMLParser
from xml.dom import minidom

import pandas as pd


def e_search(db, query):
    url2fetch = "https://eutils.ncbi.nlm.nih.gov/entrez/eutils/esearch.fcgi?db
        ="\
                + db + "&usehistory=y&term=" + query
    raw = requests.get(url2fetch).text

    kq = QueryKeyParser()
    kq.feed(raw)
    query_key = kq.q

    we = WebEnvParser()
    we.feed(raw)
    web_env = we.w

    return query_key, web_env


def get_summary(query_key, web_env, db):
    url2fetch = "https://eutils.ncbi.nlm.nih.gov/entrez/eutils/esummary.fcgi?
        db=" + db + "&version=2.0&query_key="\
                + query_key + "&WebEnv=" + web_env
    raw = requests.get(url2fetch).text
    return raw


def parse_summary(raw):
    ndata = raw.replace('\t', '').replace('\n', '')
    xmldoc = minidom.parseString(ndata)
    items = xmldoc.getElementsByTagName('DocumentSummary')
    cids, smiles, names, formulas = [], [], [], []
    for i in items:
        """
```

```
        First of all, verify if the compound has a MeSH Pharma annotation,
            otherwise it's useless to retrieve it
        """
        pharm = i.getElementsByTagName('PharmActionList')
        for p in pharm:
            """ If the node PharmActionList has children it means that the
                compound has pharma annotation"""
            if p.childNodes:
                for cnode in (i.getElementsByTagName('CID')):
                    if cnode.childNodes:
                        cids.append(str(cnode.childNodes[0].data))
                    else:
                        cids.append("?")
                for snode in (i.getElementsByTagName('CanonicalSmiles')):
                    if snode.childNodes:
                        smiles.append(str(snode.childNodes[0].data))
                    else:
                        smiles.append("?")
                for inode in (i.getElementsByTagName('MeSHHeadingList')):
                    if inode.childNodes:
                        names.append(str(inode.childNodes[0].childNodes[0].
                            data))
                    else:
                        names.append("?")
                for fnode in (i.getElementsByTagName('MolecularFormula')):
                    if fnode.childNodes:
                        formulas.append(str(fnode.childNodes[0].data))
                    else:
                        formulas.append("?")

    return cids, smiles, names, formulas


def mesh_link_parser(raw):
    ndata = raw.replace('\t', '').replace('\n', '')
    xmldoc = minidom.parseString(ndata)
    items = xmldoc.getElementsByTagName('Link')
    meshids = []
    for i in items:
        ids = i.getElementsByTagName('Id')
        for uid in ids:
            meshids.append(str(uid.childNodes[0].data))
    return meshids


def get_mesh_summary(meshid):
    meshidstr = ",".join(meshid)
```

```
    url2fetch = "https://eutils.ncbi.nlm.nih.gov/entrez/eutils/esummary.fcgi?
        db=mesh&id=" + meshidstr
    raw = requests.get(url2fetch).text

    return raw


def get_mesh_info(raw):
    ndata = raw.replace('\t', '').replace('\n', '')
    xmldoc = minidom.parseString(ndata)
    items = xmldoc.getElementsByTagName('Item')
    treeNumbers = []
    meshTerms = []
    for i in items:
        if i.attributes['Name'].value == 'TreeNum':
            treeNumbers.append(str(i.childNodes[0].data))
        if i.attributes['Name'].value == 'DS_MeshTerms':
            meshTerms.append(str(i.childNodes[0].childNodes[0].data))

    return treeNumbers, meshTerms


def get_mesh_id(cid):
    url2fetch = "https://eutils.ncbi.nlm.nih.gov/entrez/eutils/elink.fcgi?
        dbfrom=pccompound&version=2.0" \
                "&db=mesh&id=" + cid + "&linkname=pccompound_mesh_pharm"
    raw = requests.get(url2fetch).text

    return mesh_link_parser(raw)


def get_mesh_info_from_cid(cid):
    meshidlist = get_mesh_id(cid)
    raw = get_mesh_summary(meshidlist)
    return get_mesh_info(raw)


def get_data_frame(raw):

    cids, smiles, names, formulas = parse_summary(raw)
    treeids, terms = [], []

    for c in cids:
        tid, ts = get_mesh_info_from_cid(c)
        treeids.append(tid)
        terms.append(ts)
```

```
    d = {'CID': cids, 'SMILES': smiles, 'Name': names, 'Formula': formulas, '
        Terms': terms, 'TreeIds': treeids}
    df = pd.DataFrame(data=d)
    df = df.sort_values('CID')
    return df.reset_index(drop=True)


class QueryKeyParser(HTMLParser):
    def reset(self):
        self.nextq = False
        self.q = ''
        HTMLParser.reset(self)

    def handle_starttag(self, tag, attrs):
        # print(tag, " ", attrs)
        if tag == "querykey":
            self.nextq = True

    def handle_data(self, data):
        if self.nextq:
            self.q = data
            self.nextq = False


class WebEnvParser(HTMLParser):
    def reset(self):
        self.nextw = False
        self.w = ''
        HTMLParser.reset(self)

    def handle_starttag(self, tag, attrs):
        # print(tag, " ", attrs)
        if tag == "webenv":
            self.nextw = True

    def handle_data(self, data):
        if self.nextw:
            self.w = data
            self.nextw = False
```

Appendix B

Chemoinformatics add-on

Molecule Embedding Widget

Molecule Embedding Widget is composed of three Python classes, one for the visualization of the widget and two for handling server connection. The core *MoleculeEmbedder* class is here reported to show how data are passed to the server and the results retrieved.

```
import logging
import random
import uuid
from itertools import islice
from os.path import join, isfile
from AnyQt.QtCore import QSettings

import cachecontrol.caches
import numpy as np
import requests

from Orange.data import ContinuousVariable, Domain, Table
from Orange.misc.environ import cache_dir

from orangecontrib.chem.http2_client import Http2Client
from orangecontrib.chem.http2_client import MaxNumberOfRequestsError
from orangecontrib.chem.utils import md5_hash
from orangecontrib.chem.utils import save_pickle, load_pickle

log = logging.getLogger(__name__)
```

```
MODELS = {
    'smiles': {
        'name': 'CNN-Based SMILES Embedder',
        'description': 'CNN model trained on Pharmacologic Action MeSH terms
            classification',
        'target_smiles_length': 1021,
        'layers': ['penultimate'],
        'order': 0
    },
}


class EmbeddingCancelledException(Exception):
    """Thrown when the embedding task is cancelled from another thread.
    (i.e. MoleculeEmbedder.cancelled attribute is set to True).
    """


class MoleculeEmbedder(Http2Client):
    """Client side functionality for accessing a remote http2
    molecule embedding backend.

    """
    _cache_file_blueprint = '{:s}_{:s}_embeddings.pickle'
    MAX_REPEATS = 4
    CANNOT_LOAD = "cannot load"

    def __init__(self, model="smiles", layer="penultimate",
                 server_url='api.garaza.io:443'):
        super().__init__(server_url)
        model_settings = self._get_model_settings_confidently(model, layer)
        self._model = model
        self._layer = layer
        self._target_smiles_length = model_settings['target_smiles_length']

        cache_file_path = self._cache_file_blueprint.format(model, layer)
        self._cache_file_path = join(cache_dir(), cache_file_path)
        self._cache_dict = self._init_cache()

        self._session = cachecontrol.CacheControl(
            requests.session(),
            cache=cachecontrol.caches.FileCache(
                join(cache_dir(), __name__ + ".MoleculeEmbedder.httpcache"))
        )
```

```
    # attribute that offers support for cancelling the embedding
    # if ran in another thread
    self.cancelled = False
    self.machine_id = \
        QSettings().value('error-reporting/machine-id', '', type=str)  \
        or str(uuid.getnode())
    self.session_id = None

@staticmethod
def _get_model_settings_confidently(model, layer):
    if model not in MODELS.keys():
        model_error = "'{:s}' is not a valid model, should be one of: {:s}
            "
        available_models = ', '.join(MODELS.keys())
        raise ValueError(model_error.format(model, available_models))

    model_settings = MODELS[model]

    if layer not in model_settings['layers']:
        layer_error = (
            "'{:s}' is not a valid layer for the '{:s}'"
            " model, should be one of: {:s}")
        available_layers = ', '.join(model_settings['layers'])
        raise ValueError(layer_error.format(layer, model, available_layers
            ))

    return model_settings

def _init_cache(self):
    if isfile(self._cache_file_path):
        try:
            return load_pickle(self._cache_file_path)
        except EOFError:
            return {}

    return {}

def __call__(self, *args, **kwargs):
    if len(args) and isinstance(args[0], Table) or \
            ("data" in kwargs and isinstance(kwargs["data"], Table)):
        return self.from_table(*args, **kwargs)
    elif (len(args) and isinstance(args[0], (np.ndarray, list))) or \
            ("smiles" in kwargs and isinstance(kwargs["smiles"], (np.
                ndarray, list))):
        return self.from_smiles(*args, **kwargs)
    else:
        raise TypeError
```

```
def from_table(self, data, col="SMILES", smiles_processed_callback=None):
    # !!
    smiles = data[:, col].metas.flatten()
    embeddings = self.from_smiles(smiles, smiles_processed_callback)
    return MoleculeEmbedder.prepare_output_data(data, embeddings)

def from_smiles(self, smiles, smiles_processed_callback=None):
    """Send the smiles to the remote server in batches. The batch size
    parameter is set by the http2 remote peer (i.e. the server).

    Parameters
    ----------
    smiles: list
        A list of smiles for moelcules to be embedded.

    smiles_processed_callback: callable (default=None)
        A function that is called after each smiles is fully processed
        by either getting a successful response from the server,
        getting the result from cache or skipping the smiles.

    Returns
    -------
    embeddings: array-like
        Array-like of float16 arrays (embeddings) for
        successfully embedded smiles and Nones for skipped smiles.

    Raises
    ------
    ConnectionError:
        If disconnected or connection with the server is lost
        during the embedding process.

    EmbeddingCancelledException:
        If cancelled attribute is set to True (default=False).
    """
    if not self.is_connected_to_server():
        self.reconnect_to_server()

    self.session_id = str(random.randint(1, 1e10))
    all_embeddings = [None] * len(smiles)
    repeats_counter = 0

    # repeat while all smiles has embeddings or
    # while counter counts out (prevents cycling)
    while len([el for el in all_embeddings if el is None]) > 0 and \
        repeats_counter < self.MAX_REPEATS:
```

```
            # take all smiles without embeddings yet
            selected_indices = [i for i, v in enumerate(all_embeddings)
                                if v is None]
            smiles_wo_emb = [(smiles[i], i) for i in selected_indices]

            for batch in self._yield_in_batches(smiles_wo_emb):
                b_smiles, b_indices = zip(*batch)
                try:
                    embeddings = self._send_to_server(
                        b_smiles, smiles_processed_callback, repeats_counter
                    )
                except MaxNumberOfRequestsError:
                    # maximum number of http2 requests through a single
                    # connection is exceeded and a remote peer has closed
                    # the connection so establish a new connection and retry
                    # with the same batch (should happen rarely as the setting
                    # is usually set to >= 1000 requests in http2)
                    self.reconnect_to_server()
                    embeddings = [None] * len(batch)

                # insert embeddings into the list
                for i, emb in zip(b_indices, embeddings):
                    all_embeddings[i] = emb

                self.persist_cache()
            repeats_counter += 1

        ## change smiles that were not loaded from 'cannot loaded' to None
        all_embeddings = \
            [None if not isinstance(el, np.ndarray) and el == self.CANNOT_LOAD
             else el for el in all_embeddings]

        return np.array(all_embeddings)

    def _yield_in_batches(self, list_):
        gen_ = (s for s in list_)
        batch_size = self._max_concurrent_streams

        num_yielded = 0

        while True:
            batch = list(islice(gen_, batch_size))
            num_yielded += len(batch)

            yield batch
```

```
            if num_yielded == len(list_):
                return

    def _send_to_server(self, smiles_list, smiles_processed_callback, retry_n)
        :
        """ Load smiles and compute cache keys and send requests to
        an http2 server for valid ones.
        """
        cache_keys = []
        http_streams = []

        for smiles in smiles_list:
            if self.cancelled:
                raise EmbeddingCancelledException()

            if not smiles:
                # skip the sending because smiles was skipped at loading
                http_streams.append(None)
                cache_keys.append(None)
                continue

            cache_key = md5_hash(smiles.encode('utf-8'))
            cache_keys.append(cache_key)
            if cache_key in self._cache_dict:
                # skip the sending because smiles is present in the
                # local cache
                http_streams.append(None)
                continue

            try:
                headers = {
                    'Content-Type': 'text/plain',
                    'Content-Length': str(len(smiles))
                }
                stream_id = self._send_request(
                    method='POST',
                    url='/chem/' + self._model +
                        '?machine={}&session={}&retry={}'
                        .format(self.machine_id, self.session_id, retry_n),
                    headers=headers,
                    body_bytes=smiles.encode('utf-8')
                )
                http_streams.append(stream_id)
            except ConnectionError:
                self.persist_cache()
                raise
```

```
    # wait for the responses in a blocking manner
    return self._get_responses_from_server(
        http_streams,
        cache_keys,
        smiles_processed_callback
    )

def _get_responses_from_server(self, http_streams, cache_keys,
                               smiles_processed_callback):
    """Wait for responses from an http2 server in a blocking manner."""
    embeddings = []

    for stream_id, cache_key in zip(http_streams, cache_keys):
        if self.cancelled:
            raise EmbeddingCancelledException()

        if not stream_id and not cache_key:
            # when smiles cannot be loaded
            embeddings.append(self.CANNOT_LOAD)

            if smiles_processed_callback:
                smiles_processed_callback(success=False)
            continue

        if not stream_id:
            # skip rest of the waiting because smiles was either
            # skipped at loading or is present in the local cache
            embedding = self._get_cached_result_or_none(cache_key)
            embeddings.append(embedding)

            if smiles_processed_callback:
                smiles_processed_callback(success=embedding is not None)
            continue

        try:
            response = self._get_json_response_or_none(stream_id)
        except ConnectionError:
            self.persist_cache()
            raise

        if not response or 'embedding' not in response:
            # returned response is not a valid json response
            # or the embedding key not present in the json
            embeddings.append(None)
        else:
            # successful response
            embedding = np.array(response['embedding'], dtype=np.float16)
```

```
                embeddings.append(embedding)
                self._cache_dict[cache_key] = embedding

            if smiles_processed_callback:
                smiles_processed_callback(embeddings[-1] is not None)

        return embeddings

    def _get_cached_result_or_none(self, cache_key):
        if cache_key in self._cache_dict:
            return self._cache_dict[cache_key]
        return None

    def __enter__(self):
        return self

    def __exit__(self, exception_type, exception_value, traceback):
        self.disconnect_from_server()

    def clear_cache(self):
        self._cache_dict = {}
        self.persist_cache()

    def persist_cache(self):
        save_pickle(self._cache_dict, self._cache_file_path)

    @staticmethod
    def construct_output_data_table(embedded_smiles, embeddings):
        X = np.hstack((embedded_smiles.X, embeddings))
        Y = embedded_smiles.Y

        attributes = [ContinuousVariable.make('n{:d}'.format(d))
                      for d in range(embeddings.shape[1])]
        attributes = list(embedded_smiles.domain.attributes) + attributes

        domain = Domain(
            attributes=attributes,
            class_vars=embedded_smiles.domain.class_vars,
            metas=embedded_smiles.domain.metas
        )

        return Table(domain, X, Y, embedded_smiles.metas)

    @staticmethod
    def prepare_output_data(input_data, embeddings):
        skipped_smiles_bool = np.array([x is None for x in embeddings])
```

```
        if np.any(skipped_smiles_bool):
            skipped_smiles = input_data[skipped_smiles_bool]
            skipped_smiles = Table(skipped_smiles)
            skipped_smiles.ids = input_data.ids[skipped_smiles_bool]
            num_skipped = len(skipped_smiles)
        else:
            num_skipped = 0
            skipped_smiles = None

        embedded_smiles_bool = np.logical_not(skipped_smiles_bool)

        if np.any(embedded_smiles_bool):
            embedded_smiles = input_data[embedded_smiles_bool]

            embeddings = embeddings[embedded_smiles_bool]
            embeddings = np.stack(embeddings)

            embedded_smiles = MoleculeEmbedder.construct_output_data_table(
                embedded_smiles,
                embeddings
            )
            embedded_smiles.ids = input_data.ids[embedded_smiles_bool]
        else:
            embedded_smiles = None

        return embedded_smiles, skipped_smiles, num_skipped

    @staticmethod
    def filter_string_attributes(data):
        metas = data.domain.metas
        return [m for m in metas if m.is_string]
```

Bibliography

[1] A. R. Leach and V. J. Gillet. *An Introduction to Chemoinformatics*. Springer Science & Business Media, 2007.

[2] A. Cereto-Massagué, M. J. Ojeda, C. Valls, M. Mulero, S. Garcia-Vallvé, and G. Pujadas. Molecular fingerprint similarity search in virtual screening. *Methods*, 71(C):58–63, 2015.

[3] Li Fei-Fei, A. Karpathy, and J. Johnson. Cs231n: Convolutional Neural Networks for Visual Recognition, Course Notes, 2018. `http://cs231n.stanford.edu/`.

[4] E. Gawehn, J. A. Hiss, and G. Schneider. Deep Learning in Drug Discovery. *Molecular Informatics*, 35(1):3–14, 2016.

[5] J. Bajorath. *Chemoinformatics: Concepts, Methods and Tools for Drug Discovery*. Springer Science & Business Media, 2004.

[6] A. Steffen, T. Kogej, C. Tyrchan, O. Engkvist, A. Cereto-Massagué, M. J. Ojeda, C. Valls, M. Mulero, S. Garcia-Vallvé, G. Pujadas, A. Cherkasov, E. N. Muratov, D. Fourches, A. Varnek, I. Igor, M. Cronin, J. Dearden, P. Gramatica, Y. C. Martin, V. Consonni, V. E. Kuz, R. Cramer, J. B. O. Mitchell, and D. G. Truhlar. QSAR Modeling: Where have you been? Where are you going to? *Journal of computational chemistry*, 4(12):4977–5010, 2015.

[7] D. Weininger. SMILES, a Chemical Language and Information System: 1: Introduction to Methodology and Encoding Rules. *Journal of Chemical Information and Computer Sciences*, 28(1):31–36, 1988.

[8] D. Weininger, A. Weininger, and J. L. Weininger. SMILES. 2. Algorithm for Generation of Unique SMILES Notation. *Journal of Chemical Information and Computer Sciences*, 29(2):97–101, 1989.

[9] H. L. Morgan. The Generation of a Unique Machine Description for Chemical Structures—A Technique Developed at Chemical Abstracts Service. *Journal of Chemical Documentation*, 5(2):107–113, 1965.

[10] J. L. Durant, B. A. Leland, D. R. Henry, and J. G. Nourse. Reoptimization of MDL keys for use in drug discovery. *Journal of Chemical Information and Computer Sciences*, 42(6):1273–1280, 2002.

[11] C. A. James, Weininger D., and Delany J. *Daylight Theory Manual*. Daylight Chemical Information Systems, Inc., 2002.

[12] D. Rogers and M. Hahn. Extended-Connectivity Fingerprints. *Journal of Chemical Information and Modeling*, 50(5):742–754, 2010.

[13] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

[14] L. Deng and D. Yu. *Deep Learning: Methods and Applications*. NOW Publishers, May 2014.

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.

[16] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, 1(4):541–551, 1989.

[17] D. Cireşan, U. Meier, and J. Schmidhuber. Multi-column Deep Neural Networks for Image Classification. *arXiv preprint arXiv:1202.2745*, 2012.

[18] D. Duvenaud, D. Maclaurin, J. Aguilera-Iparraguirre, R. Gómez-Bombarelli, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams. Convolutional Networks on Graphs for Learning Molecular Fingerprints. In *Advances in Neural Information Processing Systems*, pages 2224–2232, 2015.

[19] S. Kearnes, K. McCloskey, M. Berndl, V. Pande, and P. Riley. Molecular graph convolutions: moving beyond fingerprints. *Journal of Computer-Aided Molecular Design*, 30(8):595–608, 2016.

[20] C. W. Coley, R. Barzilay, W. H. Green, T. S. Jaakkola, and K. F. Jensen. Convolutional Embedding of Attributed Molecular Graphs for Physical Property Prediction. *Journal of Chemical Information and Modeling*, 57(8):1757–1772, 2017.

[21] S. Jastrzębski, D. Leśniak, and W. M. Czarnecki. Learning to SMILE(S). *arXiv preprint arXiv:1602.06289*, pages 1–7, 2016.

[22] R. Johnson and T. Zhang. Semi-supervised Convolutional Neural Networks for Text Categorization via Region Embedding. In *Advances in Neural Information Processing Systems 28*, pages 919–927, 2015.

[23] S. Kim, P. A. Thiessen, Evan E. Bolton, J. Chen, G. Fu, A. Gindulyte, L. Han, J. He, S. He, B. A. Shoemaker, J. Wang, B. Yu, J. Zhang, and S. H. Bryant. PubChem substance and compound databases. *Nucleic Acids Research*, 44(D1):D1202–D1213, 2016.

[24] J. Demšar, T. Curk, A. Erjavec, Č. Gorup, T. Hočevar, M. Milutinovič, M. Možina, M. Polajnar, M. Toplak, A Starič, M. Štajdohar, L. Umek, L. Žagar, J. Žbontar, M. Žitnik, and B. Zupan. Orange: Data mining toolbox in python. *Journal of Machine Learning Research*, 14:2349–2353, 2013.

[25] T. E. Oliphant. *A guide to NumPy*, volume 1. Trelgol Publishing USA, 2006.

[26] W. McKinney et al. Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference*, pages 51–56, 2010.

[27] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[28] F. Chollet et al. Keras. `https://keras.io`, 2015.

[29] E. Sayers. Entrez Programming Utilities Help. `https://http://www.ncbi.nlm.nih.gov/books/NBK25499`, 2009.

[30] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, and M. Devin. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint*, 2016.

[31] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv preprint*, 2016.

[32] C. A. James. OpenSMILES specification. `http://opensmiles.org/opensmiles.html`, 2007.

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

[34] J. R. Sashank, K. Satyen, and K. Sanjiv. On the convergence of adam and beyond. In *International Conference on Learning Representations*, 2018.

[35] G. Landrum. Rdkit: Open-source cheminformatics. `https://github.com/rdkit/rdkit`.

[36] Z. Wu, B. Ramsundar, E. N. Feinberg, J. Gomes, C. Geniesse, A. S. Pappu, K. Leswing, and V. Pande. Moleculenet: A benchmark for molecular machine learning, 2017. *arXiv preprint arXiv:1703.00564*, 2017.

[37] B. Ramsundar et al. Deepchem. `https://deepchem.io/`, 2016.

[38] L. J. P. Van Der Maaten and G. E. Hinton. Visualizing high-dimensional data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

Acknowledgements

I would first like to thank, with a huge *Hvala lepa*, Professor Blaž Zupan that gave me the opportunity to be part of his Lab and develop this work under his supervision, and all the University of Ljubljana Bioinformatics Lab members. They made me immediately feel welcome and part of the Orange family, and helped me whenever I had any trouble or question. I gained a lot of experience working with them all but, most importantly, I could appreciate a really high-quality researching team. Particularly, I would like to thank Dr. Andrej Čopar and Dr. Marko Toplak for the help and the patience in the development of the Chemoinformatics add-on.

I miei ringraziamenti vanno poi al Professor Riccardo Bellazzi, che mi ha dato la possibilità di preparare questa tesi all'estero, indirizzandomi prontamente verso il Laboratorio del Professor Zupan, e per l'aiuto nella stesura e revisione del lavoro.

Infine, devo ringraziare tutti coloro i quali mi hanno sostenuto, supportato e sopportato durante questo mio percorso, *in primis* i miei genitori che hanno sempre creduto in me, non facendomi mai mancare le possibilità ed il supporto necessari a realizzare i miei obbiettivi. Grazie a tutti i miei amici e compagni con cui ho condiviso questi anni universitari, in particolare ai miei Capi e a Carolina, al mio fianco fin dall'inizio.