UNIVERSITÀ DEGLI STUDI DI TRENTO

Department of Information Engineering and Computer Science

Master's Degree in
Computer Science

Final dissertation

Graph Embedding in 2D

Supervisor

Prof. Andrea Passerini

Co-supervisor

Prof. Fabrizio Costa

Student

Antonio Longa

Academic year 2018/2019

Ringraziamenti

*Vorrei ringraziare il prof. Passerini, relatore di questa tesi di laurea, per la disponibilità e precisione dimostratemi durante tutto il periodo di sviluppo e stesura.
Inoltre, vorrei ringraziare il prof. Costa, correlatore della tesi, per il grande e costante aiuto nello sviluppo e nella stesura della tesi.
Senza di loro questo lavoro non avrebbe preso vita!*

Un grande ringraziamento alla mia famiglia, che con il loro instancabile sostegno, sia economico che morale, mi hanno permesso di arrivare fin qui davanti a voi oggi, contribuendo alla mia formazione personale.

Vorrei infine ringraziare la mia fidanzata e tutti gli amici di Livigno. Per il loro notevole supporto in questi anni.

Per ultimi ma non meno importanti, tutti gli amici conosciuti a Trento, Exeter e Helsinki, per il loro costante appoggio nei momenti piu difficili di questo lungo percorso.

Contents

Abstract

Graphs are powerful structures used to represent relationships among entries. In the graph domain, the embedding of the structure in a real low dimensional space is still an open problem. The main advantages of a low dimensional embedding of sets of graphs are visualisation and density estimation. The visualisation allows observing sets of graphs, while density estimation makes it possible to identify outliers.

The goal of this study is to develop a model that embeds graphs in low dimension vectors. In particular, two conflicting objectives have been considered: one aims to maximize classification accuracy in the low dimension space, while the second maximizes correlation among distances in different spaces. As these objectives are conflicting, two different models have been proposed.

The realisation of what follows has been done at Exeter University- Devon (UK)- under the supervision of Professor Fabrizio Costa, while the advisor Andrea Passerini provided support from the University of Trento.

1 Introduction

A graph is a powerful structure for expressing relationships among entities such as relationships among people in social networks, networks of cities, the behaviour of users in a system, proteins, molecules and many others. Such expressiveness lies in a more complicated extraction of information. As far as a graph is a complex data structure, it exists the need to convert it in a more straightforward representation like vectors. The conversion of a graph into a vector is called embedding, and different levels of granularity can be applied. In essence, it maps either the whole graph or a single node to a vector. Most algorithms embed graphs in high dimensional spaces. Instead, the work discussed aims to embed graphs in a low dimensional space, in particular, a 2-dimensional space. The outstanding advantages of the proposed solution are the visualization of sets of graphs, and it enables density estimations, which can be used to test for outliers.

This work tests several algorithms based on different techniques and evaluates the produced embedding. First, the graph is embedded in a high dimensional space, using three different approaches as discussed below. Then, the space is reduced to two dimensions. Each algorithm is tested on five data sets: a set of proteins, and four sets of molecules. After all, The evaluation of the embedding is done with two different approaches. The first consists in maximizing the accuracy obtained by 1-Nearest-Neighbour classifier, the second maximizes the correlation among distances in high and low dimensions.

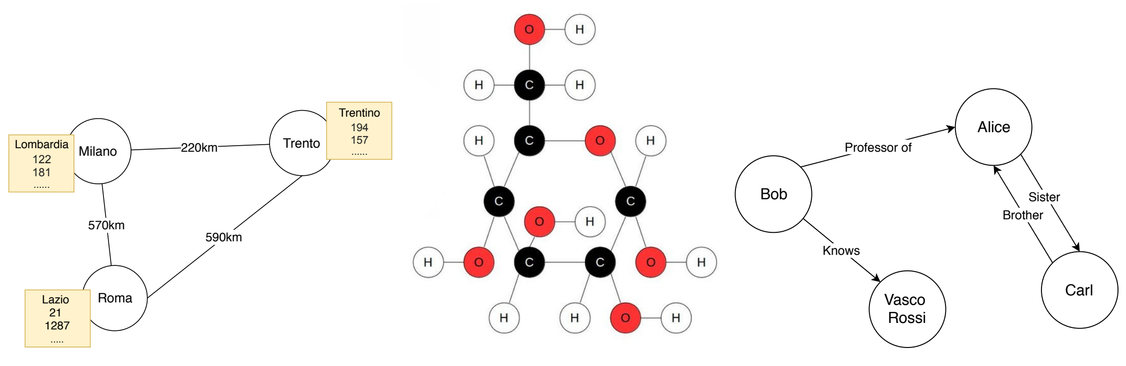
The dissertation is structured as follows. The second chapter explains the theory behind the used algorithms. Some techniques of graphs embedding are briefly explained, giving a more detailed description of embedding based on graphs kernels. In particular, the solution introduced uses Neighborhood Subgraph Pairwise Distance Kernel (NSPDK)[2] as embedder in a high dimensional space. Moreover, an introduction to Neural Networks and Graph Neural Network is given. Spektral[25], a Graph Neural Network python library, is explained in detail, with particular emphasis on a convolutional and a poling layer that will be used in the furthers chapters. The second chapter is concluded with the explanation of three different technique for dimensionality reduction: Principal Component Analysis (PCA), Uniform Manifold Approximation and Projection (UMAP)[14] and Autoencoders.
The third chapter explains the five data sets and describes their main features. Then, sets of graphs are embedded using NSPDK and the second-last layer provided by Spektral. Then the two techniques are evaluated through the accuracy obtained with 1-Nearest-Neighbour classifier in the embedded space. To conclude, the input graph is pre-processed, adding as the node's attributes the node embedding itself. This technique achieves the best 1-Nearest-Neighbour accuracy in 128-dimensional space. Thus it is used as graph embedding in the following chapters.
The fourth chapter is devoted to reducing the embedding dimension from a 128 dimension vector to a two dimensional one. In the beginning, an Autoencoder reduces the dimension up to the target, producing poor results. For this reason, the Autoencoder is used to reduce the dimension down to 32 dimensions.
Motivated by the fact that an Autoencoder does not take advantages from the supervision of the input, a new deep neural network architecture, called Supervised Autoencoder, is developed. The network is constructed by the combination of a deep Autoencoder and a classifier. The Autoencoder tries to rebuild the input, and at the same time, the classifier tries to classify the input. The Supervised Autoencoder attempts to reduce the dimension of the input while considering the belonging class according to a callback procedure. In particular, a special callback is developed, called Sinusoidal Callback. The callback allows specifying the weight of the classifier and the weight of the decoder, during the training phase. Finally, the Supervised Autoencoder is used to reduce the dimension of the input from 128 dimensions to 32 dimensions, and then it is reduced up to 2 using UMAP.

In the fifth chapter, the aim is to reduce the embedding dimension maximizing the Spearman Correlation among distances in the high and low dimensional spaces.
At the beginning of the chapter, the embedding produced by the Spektral based model is evaluated, reporting a weak correlation. For this reason, instead of using the second-last layer of the Graph Neural Network to produce the embedding, a kernel-based embedder is used. The input dimension is then reduced down to two dimensions using several techniques, including the Supervised Autoencoder.
Finally, the sixth chapter draws some conclusions, discuss the models and introduce some of the works that could be carried out in the future.

2 Background

In this chapter is explained the theory of the used methods. In the first section, graphs are introduced, and the theory behind graph embedding is explained. Successively, general embedding techniques are introduced, with a special focus on kernel-based techniques. In particular, a graph kernel, called NSPDK, is explained in details. Such a kernel is used in the thesis to embed graphs. The second section defines neural networks, giving a particular focus on the graph neural networks. Spektral, a DNN Python library, is introduced, and the used convolutional and pooling layer are analyzed. In the last section, three different techniques for dimensionality reduction are explained; such techniques rely on three different approaches to solve the dimension reduction problem.

A Graphs is a data structure that exists in nature, it can represent connection among cities, relationship between people, proteins, molecules, citations network in research areas, knowledge graphs, etc. Formally, a Graph $G = V, E$. where V is a set of *nodes* or *vertices*, and E is a set of *edges*. Sometimes a weight function is given $f(e_{i,j})$, such function compute the weight of the edge between node v_1 and node v_2, defining W, that is a matrix where the element in position $W_{i,j}$ is the weight of the edge $e_{i,j}$. Nodes and Edges may have labels and attributes, adding information into the structure. A graph is said *directed* if edges are arrows, it means that all the edges are directed from one vertex to another. On the other hand, an *undirected* graph does not have direction, in other words, if there is an edge from v_1 to v_2, it means that v_1 reaches v_2, and v_2 reaches v_1.
For example, Figure 2.1a shows a weighted indirect graph with an attribute on nodes, where nodes represent three cities: Rome, Milan and Trento, while edges represent the connections between the cities. In this example, the weight (represented on the edges) is the pairwise distance of the cities, while nodes show labels and also some attributes, that are: Region, altitude and area. Figure 2.1b shows a molecule, in this case, the graph is undirected where only the nodes have attributes. Another example is shown in figure 2.1c, in particular, it shows a social graph representing the relationship among people. It is a directed graph with labels on both edges and nodes.



(a) Cities (b) Molecule (c) Social Network

Figure 2.1: Example of graphs.

A graph can be represented using an *adjacency matrix* or an *adjacency list*, an adjacency matrix is a binary matrix M of the size $|V| \times |V|$, where if the element in position $M_{i,j}$ is equal to one, it means that there is an edge between the node V_i and V_j. the other way to represent a graph is through an adjacency list, that is an array of $|N|$ lists, where the element in position i is a list containing all the vertices connected to V_i. In general, the used representation is task drive, it means that some algorithms are faster on the adjacency matrix while others are faster on the adjacency list. Consider

that if a graph is undirected the adjacency matrix will be symmetric, so the adjacency list would save space. In some cases, the graphs to be represented are huge and sparse, in such a case, a sparse adjacency matrix is used.
Graphs are a powerful structure that can be very complex to analyze, there is a large number of problems related to graphs in different granularity. Some problems are related to the whole graphs such as graph classification, graph clustering, graph optimization, etc. Others problems concern nodes or edges, for instance, node classification, node recommendation, node clustering or edges prediction. On the other hand, there are problems related to subgraphs, such as triple classification in a knowledge graph. However, this thesis aims to embed the whole graph in a low dimension space.

2.1 Graph Embedding

To analyze graphs, and extract information from them, several techniques have been studied, involving distributed graph analytic framework, Deep Neural Network, and Embedding. This thesis aims to embed a graph in low dimensionality, allowing others techniques to work in the embedded domain to solve problems related to graphs.
Cai et al. [1] define some measure needed to formalize the graph embedding problem. The basic idea is to map a graph, or part of it, in a real d-dimensional space, preserving distances among graphs or distances among nodes in both the domain. To compute distances in real spaces there are many function, such as *Euclidean distance* or *Cosine distance*. On the other hand, compute distances among graphs is a more complex task, because of the complexity of the data structure. Furthermore, it is important to distinguish between distances among graphs and distances among nodes. For example, the *first-order proximity* or *second-order proximity* are metrics that compute the distances between two nodes.

Definition 1. *The first-order proximity between node v_i and node v_j is the weight of the edge $e_{i,j} = W_{i,j}$.*

With the first-order proximity, two nodes are more similar if they are connected by an edge with larger weight. The second-order proximity compares the neighbours of the nodes.

Definition 2. *Let $s_i = [W_{i,1}, ..., W_{i,j}, ..., W_{i,|N|}]$ be a vector, representing the weights of the edges connected to the node v_i, where the element in position j is the weight of the edge between the node v_i and the node v_j. The second-order proximity between node v_i and node v_j is the similarity of the neighbour of v_i (s_i) and the neighbour of v_j (s_j).*

Instead, to compute distances between graphs there are different technique, for instance, Edit distance or Feature extraction [12]. The idea behind the edit distance is to transform a graph into another graph, counting the number of atomic operations (additions, deletions, substitutions of nodes or edges, and reversions of edges). This operation associates a cost to each atomic operation, and minimize the total cost to convert a graph into the other. Another technique is called Feature extraction, where the key idea is to extract information from the graph like the number of nodes, degree distribution, density, etc. and then to compute the similarity between graphs, the extracted features are compared with a similarity measure.

Cai [1] defines the embedding problem as follow:

Definition 3. ***Graph embedding:*** *Given a graph $G = (V, E)$, and a predefined dimensionality of the embedding d ($d \ll |V|$), the problem of graph embedding is to convert G into a d-dimensional space, in which the graph property is preserved as much as possible. The graph property can be quantified using proximity measures such as the first- and higher-order proximity. Each graph is represented as either a d-dimensional vector (for a whole graph) or a set of d-dimensional vectors with each vector representing the embedding of part of the graph (e.g., node, edge, substructure).*

The embedding problem has different settings, depending on the input graph and the embedding output.

GRAPH EMBEDDING INPUT

The graph embedding input can be divided in the following three categories:

- Homogeneous
- Heterogeneous
- With auxiliary information

Homogeneous graphs. The Homogeneous graphs are the most basic graph embedding input. In a homogeneous graph, all the nodes belong to the same category, and all the edges belong to the same category. They can also be categorized as directed (or undirected) graphs, and as weighted (or unweighted) graphs. Usually, the weights and the directions may help represent the graph in the embedded space more accurately. For instance, suppose embed the node of a directed graph, if two nodes are connected in both direction, they should be represented nearby, concerning nodes that are connected in only one direction. On the other hand, if the graph is weighted, two nodes connected by an edge with a high weight should be represented closer to nodes that have a connection with a lower weight.

Heterogeneous graphs. Graphs, where nodes or edges are of different types, are called Heterogeneous graphs, examples of this type of graphs are the following.
Multimedia Networks. Multimedia graphs contain a different type of nodes such as text, audio, images. Such graphs represent the interaction among multimedia data. Geng et al. [6] embed a network representing users and images in the same space, facilitating the task of recommendation images to users.
Knowledge graphs. In Knowledge graphs, nodes represent entities and edges represents relationships among entities. Zhao et al. [28] embed the Wikipedia graph using three types of nodes: entity (e), category (c) and word (w), while edges are of three types: e-e, e-c and w-w.

Graph with auxiliary information. The graph embedding input can be a graph with auxiliary information on node/edges/whole-graph. Several types of additional information can be added to a graph, such as::
Labels. Generally, a label is a word that identifies the node, for example in figure 2.1a the label specifies the name of the city that the node is representing. Nodes with the same label should be embedded closer to each other.
Attributes. An attribute associated with a Node or and edge usually is discrete or continues value, describing more information about the node. However, an attribute may also be a word. *Node features.* Usually, node features are text associated with nodes as vectors of words, or documents. However, it may also be images or multimedia.

GRAPH EMBEDDING OUTPUT

The graph embedding output is usually task-driven, it means that the output depends on the problem that has to be solved. There are several types of graph embedding output, and they are explained below.

Node embedding. In this case, the embedding of the graphs is a list of vectors, each element of the list represent a node in the embedded space. The idea that "close" node has to be close also in the embedded space. Where the closeness in the embedded space could be measured using Euclidean Distance, while the closeness in the graph domain could be measured using n-order proximity. How such "closeness" is defined, define different algorithms for graph embedding. Usually, node embedding is used for node classification, node clustering, node recommendation.

Edge embedding. . The embedding of the graph is a list of $|E|$ elements, each element represent the embedded edge. In general, this kind of embedding is used to link prediction or triple classification (a specific application for knowledge graphs).

Graph embedding. In this case, the embedding of the graph is a d-dimensional vector, where d is the size of the embedded space. This type of embedding has several applications, for instance, Nikolentzos et al. [18], embedded a graph into a matrix of dimension $|N| \times d$ where each row represents the embedding of a node, successively they consider the embedded graph as a bag-of-words, in particular, each embedded vertex is a word, and the whole set of words is the embedding of the graph. Successively, they use Earth Mover's Distance (EMD) [20] to compute distances and finally they classify graphs using SVM. Another important application of graph embedding is visualization. Density estimation can be done only on low dimension spaces, so this is another reason to embed the whole graphs. Density estimation allows detecting out-layer, for instance, Zambron et al. [26] embed streams of graphs for out-layer detection.

In this thesis, the graph embedding inputs are homogeneous undirected unweighted graphs, while the graph embedding output is a graph embedding in a 2-dimensional space, for a visualization purpose.

GRAPH EMBEDDING TECHNIQUE
Embedding a graph in a low dimension space preserving as much graph property information as possible. Generally, the way in with such property is defined lie in different algorithms. Some techniques are described below.

Matrix factorization. The idea is to factorize matrices related to the input graph, such matrices could be: Adjacency matrix, Laplacian matrix, Degree matrix, Weight matrix, etc. and then use it as node embedding or graph embedding.

Deep Learning. Deep learning has shown good performance in several fields. For this reason, it has been used to graph embedding, adopting architecture used in other fields or developing new models. Some architecture considers sets of random walk in the graph, instead of the whole graph. For instance, DeepWalk [27] samples a set of paths from the graph, using a truncated random walk, subsequently it uses a natural language model (SikpGram [16]) to embed the graph. Others models use the whole graph, for instance, Autoencoders could be used to produce a node embedding, in particular, if the input is the adjacency matrix then the Autoencoder will embed closer nodes that have a similar neighbourhood. Others architecture adopt convolution to produce graph embedding, using Convolutional Neural Networks.

Optimization Based. Tang et al. [24] develop an objective function to be minimized, such objective function preserves the local and global structure, using first and second-order proximity.

Graph Kernel. Kriege et al. [13] define kernel methods as a set of technique used to compute similarity (kernel) between data points. Formally, given a data set of objects (e.g. graphs) X and let $k : X \times X \to \mathbf{R}$ be a function, then k is a kernel if there is an Hilbert space $\mathcal{H}_k$ and a feature map $\phi : X \to \mathcal{H}_k$ such that $k(x, y) = \langle \phi(x), \phi(y) \rangle$ for each $x, y \in X$, where $\langle \cdot, \cdot \rangle$ is the inner product. The feature map exists if and only if k is a positive-semidefinite function.

Definition 4. *Let k be a function $k : X \times X \to \mathbf{R}$, where X is a real vector. k is a positive semidefinite function if and only if $k(x, x) \geq 0$ for all $x \in X$.*

A graph kernel can be used to embed a graph in a real d-dimensional space. For instance, Shervashidze et al. [21] propose a graphlet kernel, where a graphlet is an induced non-isomorphic subgraph of size k. A graph is embedded in a d-dimensional vector, where the element in position i is the normalized frequency of the graphlet in the whole-graph. Suppose to have a set of graphlets $\{G_1, G_2, \ldots, G_d\}$, then a graph is embedded in a real vector with d dimensions, and the element in position i represent the normalized count of the occurrences of the subgraph in the graph. Another example of embedding based on kernels is subtree patterns, where the graph is decomposed in subtree patterns. On the other hand, kernels based on walk decompose the graph in random walks or shortest path and then they embed the graph in a vector, where the element in position i represent the normalized frequency of the i-th shortest path (or random walk) in the graph. In this thesis, much effort has been involved in graph kernels, in particular in the kernel developed by Costa et al. [2], for this reason, the next subsection explains in details the kernel.

2.1.1 The Neighborhood Subgraph Pairwise Distance Kernel (NSPDK)

In the previous section, kernels have been defined, while in this subsection a specific kernel is explained, after a brief introduction to Convolution kernels. Subsequently, a graph embedder based on the previous kernel is explained.
Explicit Decomposition with Neighborhoods Consider a composite structure $x \in X$, that is a structure that can be decomposed in overlapping substructures, formally $x = \{x_1, x_2, \dots, x_d\}$. The set X_i defined by all the x_i in X is countable.

Definition 5. *Let R be a relation defined as follow $R : X_1 \times X_2 \times \cdots \times X_D \times X \to \{0, 1\}$ such that R is equal to 1 if and only if $\{x_1, x_2, \dots, x_d\}$ are parts of x.*

it follows that

Definition 6. $R^{-1} = \{x_1, x_2, \dots, x_d : R(x_1, x_2, \dots, x_d, x)\}$

Haussler [9] demonstrates that if $\exists$ a kernel $k_d : X_d X_d \to \mathbf{R}$ $\forall d = 1, \dots, D$, and if all the $x, y \in X$ can be decomposed on $x_1, x_2, \dots, x_D$ and $y_1, y_2, \dots, y_D$ respectively, then

$$k(x, y) = \sum_{\substack{x_1,\dots x_D \in R^{-1}(x) \\ y_1,\dots y_D \in R^{-1}(y)}} \prod_{d=1}^{D} k_d(x_d, y_d)$$

is a valid kernel.
The theorem proved by Haussler is the following:

Theorem 1. *If $K_1, K_2, \dots, K_D$ are kernels on X_1X_1,X_2X_2,...,X_DX_D and R is a finite relation on $(X_1X_2 \dots X_DX)$ then $K_1 * K_2 * \cdots * K_D * X$ is a kernel on $X \times X$*

In order to prove such theorem, the following three lemmas are needed:

Lemma 1. *Let K be a kernel on a set $U \times U$, and for all finite, non empty $A, B \subseteq U$, define*

$$K'(A, B) = \sum_{\substack{x \in A \\ y \in B}} K(x, y)$$

then K' is a kernel on the product of the set of all finite, non empty subset of U with itself.

Lemma 2. *Kernels are closed under the products, that is, given two kernels K_1 and K_2, respectively defined on X_1 and X_2. $K = K_1 * K_2$ is still a kernel on X_1X_2.*

Lemma 3. *The zero-extension of a kernel is a valid kernel, for instance, if $S \subseteq X$ and K is a kernel on $S \times S$ then K may be extended to be a kernel on $X \times X$ by defining $K(x, y) = 0$ if x or y is not in S.*

Proof. **Theorem 1.** Let U denote $X_1X_2\dots X_D$, considering that $K_1, K_2, \dots, K_D$ are kernels by definition. Lemma 2 $\tilde{K}$ defined as follow:

$$\tilde{K}(x, y) = \prod_{d=1}^{D} k_d(x_d, y_d)$$

is a valid kernel on $U \times U$.
Since R is finite, by lemma 1, $\tilde{K}'(R^{-1}(x), R^{-1}(y))$ is a kernel on the product of the set of all nonempty $R^{-1}(x)$ such that $x \in X$ with itself.
Finally, since $K_1 * K_2 * \dots K_D(x, y)$ is the zero extension of $K(x, y)$, from lemma 3, follow that it is a kernel on $X \times X$. □

NSPDK
Costa et al. [2] introduce the following notation:

Definition 7. *Given a vertex V and a radius r, $N_r(V)$ is the set of vertices at distance less or equal to r from the node V*

Definition 8. *Given a graph G, a radius r and a vertex V, $N_r^v(G)$ is the induced subgraph of G, rooted in V and with radius r.*

successively, they define the relation $R_{r,d}(A^v, B^u, G)$ between two rooted graphs A^v,B^u and a graph G. Such relation is true if and only if both A^v and B^u in $N_r^v(G)$ with $v \in V(G)$. In other word, given a graph G,a radius r and a distance d, the relation $R_{r,d}$ select all pair of neighborhood graphs of radius r whose roots are at distance d in G.
Successively, they define the decomposition kernel on the relation $R_{r,d}$ over $G \times G$, as follow:

$$\kappa_{r,d}(G, G') = \sum_{\substack{A_v, B_u \in R_{r,d}^{-1}(G) \\ A'_{v'}, B'_{u'} \in R_{r,d}^{-1}(G')}} \delta(A_v, A'_{v'})\delta(B_u, B'_{u'}) \tag{2.1}$$

where $\delta(x, y)$ is the exact matching kernel, and it is equal to 1 if x is isomorphic to y ($x \simeq y$).
Finally, the Neighbourhood Subgraph Pairwise Distance Kernel is defined as:

$$K(G, G') = \sum_r \sum_d \kappa_{r,d}(G, G') \tag{2.2}$$

For efficiency reasons, they fix an upper bound to d and r. Moreover, to avoid that large infrequent subgraphs dominate the kernel value, a normalized version of the equation 2.1 is used:

$$\hat{\kappa}_{r,d}(G, G') = \frac{\kappa_{r,d}(G, G')}{\sqrt{\kappa_{r,d}(G, G')\kappa_{r,d}(G, G')}} \tag{2.3}$$

In the equation 2.2 a exact matching kernel ($\delta(x, y)$) is used. However, the isomorphism problem (ISO) has not been solved with an algorithm in deterministic polynomial time. For this reason, in NSPDK the isomorphism problem is solved using an approximate solution, composed of two steps.
The first step consists in computing a fast graph invariant encoding for G_h and G'_h, successively, they use a hash function to confront the invariant encoding of both graphs.
To compute the graph invariant a label function $L^g : G_h \to \Sigma^*$ is used, where Σ^* is a set of string and G_h is the set of rooted graphs. Such label function attach to vertex v the concatenation of the pair distance-label $\langle D(u, v), L(u)\rangle$, sorted in a lexical order. On the other hand, the hash function $H : \Sigma^* \to \mathbf{N}$ maps the computed graph encoding string in a 32-bit integer.

Idea behind NSPDK The developer of NSPDK tried to combine two popular chemoinformatic fingerprint methods: circular substructure and the atom pair representation. A circular substructure representation assigns to each atom an initial code based on the atom's properties, successively the code of an atom and all its neighbours are hashed to produce a second-order encoding. Such a process is then iterated for a given number of times k, that is the upper bound of the radius up to which features are generated. On the other hand, the atom pair representation encodes combinations of atoms with the length of the shortest path between them.

Graph Embedding with NSPDK
Neighborhood Subgraph Pairwise Distance Kernel has been implemented and realised in C++. However, a more specific Python Library has been released by the developer of NSPDK. In particular, Explicit Decomposition with Neighborhoods (EDeN) uses NSPDK to achieve different tasks, such as graph alignment and graph embedding. In this thesis, the main focus is on the embedding part of EDeN, in fact, in the following two functions to embed graphs and node of a graph are explained. In particular, the function *vectorize* produce a graph embedding, while the function *vertex_vectorize* embed nodes of a graph.

Vectorize has several inputs, here some of them are explained. The first input (*graphs*) is a list of Networkx graphs, each graph must have an attribute called "label" on both edges and nodes. Others two important parameters are the radius (r) and the distance (d), those parameters represent the upper bound radius of each subgraph and the distance between the two subgraphs. However, If the parameter *complexity* is set, then $d = r = complexity$. *Complexity* has a default value equal to 3. Another important parameter is called *nbits*, it represents the number of bits used to embed the graph. In particular, the size of the dimensions of the embedded graphs would be 2^{nbits}. The function vectorize returns a sparse matrix of the size $|graphs| \times 2^{nbits}$, where $|graphs|$ is the length of the input list of graphs, and 2^{nbits} is the size of the space, thus, the i-th row represent the embedding of the i-th graph in the input list.

Vertex_vectorize produce a node embedding from a list of graphs, it has the same input of the function Vectorize, the only difference is that vertex_vectorize returns a list of sparse matrices, with length equal to the number of input graphs. Each matrix has size $|Nodesofthegraph| \times 2^{nbits}$, thus the i-th row of the j-th matrix, is the embedding of the node i of the graph j.

2.2 Neural Network

Artificial Intelligence (AI) is an area of computer science, aiming to mimic human intelligence. Machine Learning (ML) is a subset of AI, that includes statistical techniques to improve the solution to a given task using experience. Finally, Deep Neural Networks (DNN) are techniques of ML, aiming to mimic a biological brain. Generally, a DNN is used for a supervised learning task, that is learning a map between input and output from examples pairs input-output. Convolutional Neural Network and Recursive Neural Networks are examples of DNN used for supervised tasks. However, a DNN can be used also for unsupervised learning, for instance, Deep Belief Network or Autoencoders does not need labelled input.

Perceptron
Figure 2.2 show a simple example of a perceptron, that is the smallest computational unit of a DNN. A perceptron takes N inputs and multiplies them by their weights, successively their summation goes into an activation function, that produced the output.
Formally, given an input real vector $\mathbf{x}$ of cardinality N, and a vector of weights $\mathbf{w}$, the perceptron produce the output computing the activation function $\sigma(\mathbf{w}^T\mathbf{x})$.
Figure 2.3 shows some shape that an activation function could have, some of them are linear, while others are non-linear. The simplest activation function is a threshold, shown in figure 2.3a, it propagates the output only if the sum of the weighted input is greater than zero. Such activation function

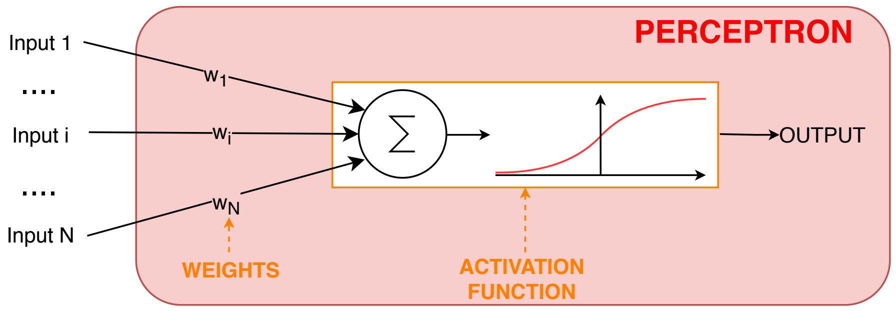


Figure 2.2: Example of a perceptron.

is not quite used, because its derivative is not defined when $x = 0$. It is important to have a derivable activation function because they are used to train the weights associated with the input, through the backward propagation algorithm (explained later). The second figure (2.3b) shows a linear function, where the summed weighted input is propagated as it is. Two interesting observations are the following, firstly, such activation function may not be useful in the classification task, because an output greater than 1 (or smaller than 0) is not appropriate. Secondly, in some tasks, a negative output value could be meaningless. For instance, the prediction of distances between cities cannot be a negative value. To solve the later problem, Nair and Hinton in 2010 proposed a Rectified Linear function (ReLU)[17], and ever since, it has been the most used activation function for deep learning applications. ReLU is shown in figure 2.3c, that is, if the summed weighted input is lesser than zero then the output is zero, while if it is greater than zero then the output is equal to the summed weighted input. On the other hand, one of the most used activation function in classification tasks is called Sigmoid (shown in figure 2.3d), it normalizes the input between 0 and 1. In particular, large positive inputs tend to be closer to 1, while large negative inputs tend to 0. However, the gradient of the function vanishes with large numbers, and this drastically affects the effectiveness of the backward propagation algorithm in a network of perceptrons.

$$\sigma(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ 1 & \text{if } x \geq 0 \end{cases} \qquad \sigma(x) = x \qquad \sigma(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x \geq 0 \end{cases} \qquad \sigma(x) = \frac{1}{1 + e^{-x}}$$

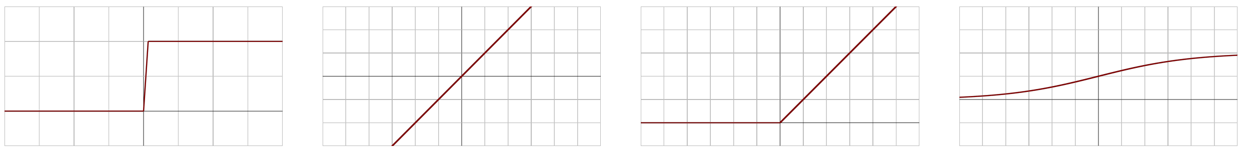

(a) Threshold (b) Linear (c) ReLU (d) Sigmoid

Figure 2.3: Example of activation functions.

Neural Network

A neural network is a directed acyclic graph where nodes are perceptrons, and it is divided into layers, especially, input layer, hidden layers and output layer. Figure 2.4 shows a 3-layer fully connected neural network, in particular, the green circles are the input features and they stay on the input layer, while perceptrons are the red circles, and they stay in two different hidden layers, similarly, the yellow circles represent the output. Finally, each arch is weighted, and such wights are trained via backward propagation, explained in the following.

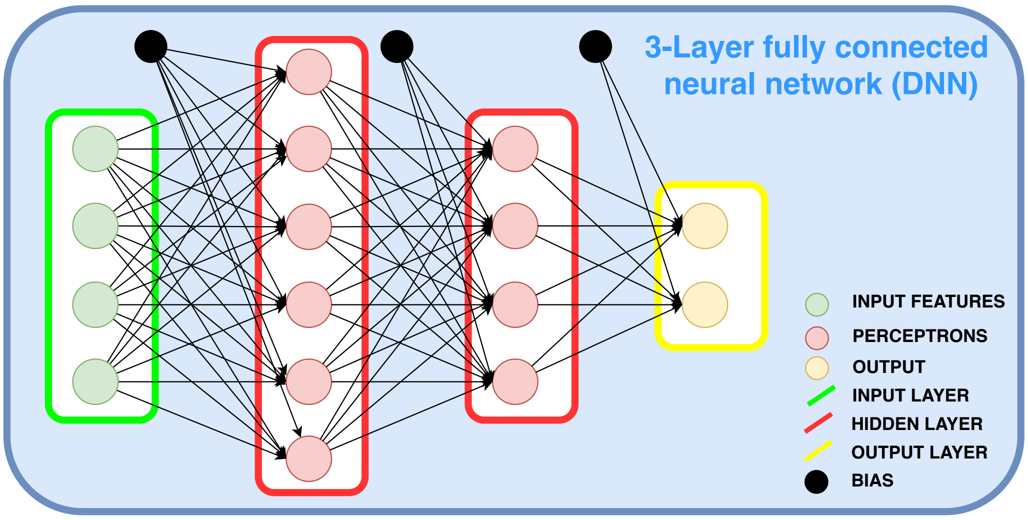


Figure 2.4: Example of a 3-layer fully connected deep neural network.

The output of the neural network is then evaluated using a loss function. The book ”Deep Learning” [7] defines the loss function as follow:

Definition 9. *The function we want to minimize or maximize is called the objective function or criterion. When we are minimizing it, we may also call it the cost function, loss function, or error function.*

In other words, a loss function is used to evaluate the goodness of the predicted output and the real one. There are many loss functions, and usually, the decision of a function depends on the task. Below are reported two loss functions, where the first one (MSE) is usually used for regression tasks, while the other (Cross-Entropy) is usually used in classification tasks.

Mean squared error (MSE)

$$L = \frac{1}{n} \sum_{i=1}^{N} (y_i - \tilde{y}_i)^2$$

Cross Entropy

$$L = - \sum_{i=1}^{N} log(\tilde{y}_i) y_i$$

In a DNN each arch has a weight, in particular, from each node in a layer l_i there are $|l_{i+1}|$ weights, that is, one weight for each node on the next layer, so it easy to see that the number of weights in a network increase quickly. For Example, the network in figure 2.4 has an input layer with four input features, then it has two hidden layers with six and four nodes respectively, the output layer has two nodes, and finally, there are three biases. At the first impression seems that it is a simple network, indeed it is. However, this simple network has $(4+1)*6+(6+1)*4+(4+1)*2 = 68$ weights. For this reason, a notation that will be used in the thesis is introduced.
Each node (a) is identified with two indices, one specifies the layer, while the other specifies the node position in the layer starting from the top to the bottom. For example, the node $a_i^{(l)}$ is the i-th node in the layer l, biases nodes are represented by a b, with the same indices of nodes. A weight is represented by $w_i^{(l)}$, again, l represent the layer and j indicate that the weight is associated to node i. Previously, we said that the activation function (σ) of a perceptron is computed on the weighted sum of the input. Thus, the value of the node $a_i^{(l)}$ is equal to:

$$a_i^{(l)} = \sigma(a_0^{(l-1)} w_0^{(l-1)} + a_1^{(l-1)} w_1^{(l-1)} + \cdots + a_n^{(l-1)} w_n^{(l-1)} + b_0) \tag{2.4}$$

All nodes in a layer can be represented by a vector $\mathbf{a^{(l)}}$ defined as follow:

$$\mathbf{a^{(l)}} = \begin{bmatrix} a_0^{(l)} \\ a_1^{(l)} \\ \vdots \\ a_n^{(l)} \end{bmatrix}$$

Similarly, the biases and the weights can be represented by a vector and a matrix, as shown below

$$\mathbf{b^{(l)}} = \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_n \end{bmatrix} \qquad \mathbf{W} = \begin{bmatrix} w_0^{(0)}, w_1^{(0)}, \dots, w_n^{(0)} \\ w_0^{(1)}, w_1^{(1)}, \dots, w_n^{(1)} \\ \vdots \\ w_0^{(0)}, w_1^{(0)}, \dots, w_n^{(l)} \end{bmatrix}$$

Now, equation 2.2 can be summarized by

$$\mathbf{a^{(l)}} = \sigma(\mathbf{W^{(l)}a^{(l-1)}} + \mathbf{b}) \tag{2.5}$$

As explained previously, the output of each perceptron is then propagated to the nodes in the next layer, thus the output of the neural network (shown in figure 2.4) is computed composing equation 2.2 as follow:

$$\begin{aligned}
&\mathbf{a^{(1)}} = \sigma(\mathbf{W^{(1)}a^{(0)}} + \mathbf{b^{(1)}}) \\
&\mathbf{a^{(2)}} = \sigma(\mathbf{W^{(2)}a^{(1)}} + \mathbf{b^{(2)}}) = \sigma(\mathbf{W^{(2)}}(\mathbf{W^{(1)}a^{(0)}} + \mathbf{b^{(1)}}) + \mathbf{b^{(2)}}) \\
&\dots \\
&\mathbf{a^{(l)}} = \sigma(\mathbf{W^{(l)}a^{((l-1)}} + \mathbf{b^{(l)}}) = \sigma(^{(l)}(\mathbf{W^{(l-1)}}(\mathbf{W^{(l-2)}}(((\dots(\mathbf{W^{(1)}a^{(0)}} + \mathbf{b^{(1)}}) + \dots))) + \mathbf{b^{(l-1)}}) + \mathbf{b^{(l)}})
\end{aligned}$$

As mentioned before, the output of a DNN goes into a loss function (L), and then it is optimized. From an optimization point of view, L is the objective function, and weights and biases are the parameters of the optimization problem. Such objective function is then optimized using an algorithm. One of the easiest algorithms is called gradient descent.
Gradient descent is an iterative optimization algorithm that starts from an initial point, compute the gradient of the function, identifying the direction of maximum increase, successively it moves the point along the negation of the previously computed direction, for a certain distance μ called learning rate. Note, that a fixed learning rate could create problem to the algorithm. For example, suppose to be closer to a local minimum and after one step of gradient descent the point is moved in another position and from the new position gradient descent go back to the previous one, in this situation gradient descent will keep jumping between two-point closer to the local minimum, never reaching it. In general, a good idea is to decrease the learning rate during the iteration of gradient descent. On the other hand, other algorithms learn the learning rate during the execution, like ADAM optimizer. Indeed, an optimization problem is usually split into two optimization subproblem 1) find the optimal direction, and 2) optimize the step size (learning rate).

Backward propagation
Backwards propagation, also known as backpropagation, is an algorithm to adjust the weights of a network.
Given a loss function L,i.e. MSE, an important question is: *How sensitive is the loss function to small changes in the weights?*
that is the partial derivative of the loss function concerning the weights. Formally:

$$L = (\mathbf{a^{(l)}} - \mathbf{y})^2 \tag{2.6}$$

where $\mathbf{a^{(l)}}$ is a vector that contains the output of the last layer of the neural network, and $\mathbf{y}$ is the target. In order to simplify the notation, equation 2.2 is defined as follow

$$\mathbf{a^{(l)}} = \sigma(z^{(l)})$$

then the partial derivative of the loss function is defined as follow:

$$\frac{\partial L}{\partial \mathbf{W^{(l)}}} = \frac{\partial z^{(l)}}{\partial \mathbf{W^{(l)}}} \frac{\partial \mathbf{a^{(l)}}}{\partial z^{(l)}} \frac{\partial L}{\partial \mathbf{a^{(l)}}} \tag{2.7}$$

and it is called chain rule. Looking deeper:

$$\begin{aligned}
\frac{\partial z^{(l)}}{\partial \mathbf{W^{(l)}}} &= \frac{\partial \mathbf{W^{(l)}} \mathbf{a^{(l-1)}} + \mathbf{b^{(l)}}}{\partial \mathbf{W^{(l)}}} = \mathbf{a^{(l-1)}} \\
\frac{\partial \mathbf{a^{(l)}}}{\partial z^{(l)}} &= \sigma'(z^{(l)}) \\
\frac{\partial L}{\partial \mathbf{a^{(l)}}} &= \frac{(\mathbf{a^{(l)}} - \mathbf{y})^2}{\partial \mathbf{a^{(l)}}} = 2(\mathbf{a^{(l)}} - \mathbf{y})
\end{aligned}$$

thus, the derivative of the loss function with respect to the weights is

$$\frac{\partial L}{\partial \mathbf{W^{(l)}}} = 2(\mathbf{a^{(l)}} - \mathbf{y})\mathbf{a^{(l-1)}}\sigma'(z^{(l)}) \tag{2.8}$$

In the same way, the sensitivity of the loss function with small changes in the bias vector, that is the partial derivative of L with respect to $\mathbf{b^{(l)}}$ is computed as follow:

$$\frac{\partial L}{\partial \mathbf{b^{(l)}}} = \frac{\partial z^{(l)}}{\partial \mathbf{b^{(l)}}} \frac{\partial \mathbf{a^{(l)}}}{\partial z^{(l)}} \frac{\partial L}{\partial \mathbf{a^{(l)}}}$$

$$\begin{aligned}
\frac{\partial z^{(l)}}{\partial \mathbf{b^{(l)}}} &= \frac{\partial \mathbf{W^{(l)}} \mathbf{a^{(l-1)}} + \mathbf{b^{(l)}}}{\partial \mathbf{b^{(l)}}} = 1 \\
\frac{\partial \mathbf{a^{(l)}}}{\partial z^{(l)}} &= \sigma'(z^{(l)}) \\
\frac{\partial L}{\partial \mathbf{a^{(l)}}} &= \frac{(\mathbf{a^{(l)}} - \mathbf{y})^2}{\partial \mathbf{a^{(l)}}} = 2(\mathbf{a^{(l)}} - \mathbf{y})
\end{aligned}$$

$$\frac{\partial L}{\partial \mathbf{b^{(l)}}} = 2(\mathbf{a^{(l)}} - \mathbf{y})\sigma'(z^{(l)}) \tag{2.9}$$

This process is repeated in a backward way through the network, and this is why it is called backpropagation. Note that backpropagation is guaranteed to converge in a local minimum, for this reason, usually a DNN is trained and tested several times, and finally, the results are averaged among the networks.

2.2.1 Graph Neural Networks

As previously explained DNNs have become widespread in the last decades, due to huge amounts of available data and to the increasing of the computational power. DNNs are used in many different tasks, such as object detection, video processing, speech recognition and natural language understanding. However, the complexity of the graph has imposed significant challenges on DNNs.
Recently, there is an increasing interest in extending deep learning technique on graphs. Motivated from the results obtained with Convolutional Neural Network, Recursive Neural Network and Autoencoders in other fields.
The first researcher that applied neural network on graphs was Sperduti et al. [22] in 1997. They used a neural network to classify directed acyclic graphs, embedding the graph in a real vector of fixed dimension, and successively use the neural network to classify it. However, the notion of Graph

Neural Network appears for the first time in 2005 from a paper published by Gori et al. [8].
Convolutional Neural Networks are widely used in digital image processing because they allow capturing the local correlation, especially correlation among neighbours pixels. Basically, a convolution neural network learns filters to apply on the input data, to achieve a task. Convolution has been introduced in graph neural network, namely Convolutional graph neural networks or ConvGNN, and they are subdivided in two categories: spatial-based and spectral-based.
Spectral-based approaches define the graph convolution inducing filters from the point of view of graph signal processing, that is, try to remove noise from the graph signal. Generally, a graph is represented by the normalized Laplacian Matrix, defined as $L = I - D^{-1/2}AD^{-1/2}$, where I is the identity matrix, A is the adjacency matrix and D is the diagonal matrix, where the element $D_{j,j}$ is the degree of the node j. Note that the normalized Laplacian Matrix is a real symmetric positive semidefinite matrix, thus it can be decomposed in $L = U\Lambda U^T$, where $U \in \mathbf{R}^{n\times n}$ is the matrix of eigenvectors ordered by eigenvalue, and the matrix Λ is the spectrum, that is the diagonal matrix of eigenvalues. Many ConvGNN relays on the spectrum due to the solid mathematical foundations.
On the other hand, spatial-based approaches define convolution based on the node's neighbourhood. In particular, given a node v, the convolution is applied to v's neighbours. Micheli et al. [15] have been the first researcher that apply convolution spatial-based, summing directly the node's neighbourhood.

2.2.2 Spektral

The library used in this thesis is called Spektral. It has been developed by Daniele Grattarola, and it is published online on GitHub[1]. Spektral is a Python library used for graph deep learning, and it is based on Keras API. The library uses a matrix-based representation, in particular, each graph is defined by tree matrices: **A**, **X** and **E**.

- $\mathbf{A} \in \mathbf{R}^N$ is the adjacency matrix.
- $\mathbf{X} \in \mathbf{R}^{N\times F}$ is a matrix representing node's feature. The i-th row contains F features of the node i.
- $\mathbf{E} \in \mathbf{R}^{N\times N\times S}$ is a 3-dimensional edge feature matrix, and the element in position i, j is a vector of length S representing the features of edge between node i and node j.

Spektral can be used for both nodes-application and graph-application, to do that is has the following three modes. The *Single* mode is used on an individual graph, and it is generally used for nodes-applications, such as node clustering, node classification or node embedding. To work with a set of graphs, Spektral can be used in the mode *Batch* and each graph has its topological structure, such mode is generally used for graph-application, for instance, graphs classification, graphs clustering or graphs embedding. Finally, the *Mixed* mode is used when the input graphs have the same topology but with different nodes/edges attributes, generally, it is known as graph signal processing. Note that the Mixed mode could be seen as a specific case of the Batch mode. However, Spektral handled them in different ways, improving memory efficiency.

Spektral provides several functionalities, such as preloaded data sets, importers, plotters, and DNN layers. Spektral's layers are subdivided into three categories: Convolutional layers, polling layers and base layers. The base layers category contains layers that allow computing operations tensor, such as the inner product or the Minkowski product. On the other end convolutional and pooling layers implement several states of the art technique developed by other researchers. However, in this thesis, only two layers are explained in details, that are Convolutional Graph Attention (GraphAttention) and minimum cut pooling (MinCutPool).

GraphAttention. Attention mechanisms are used for sequence-based input, allowing to work with inputs that have different sizes. When the attention mechanism is used to represent a single sequence, it is called self-attention. In the few last years, convolutional and recurrent neural network have been combined with self-attention, achieving excellent results on sequence representation and

[1]Spektral can be downloaded from https://github.com/danielegrattarola/spektral

machine reading. Motivate from these results, Veličković et al. [25] compute a hidden node representation using self-attention on a node's neighbourhood.
The layer developed by Veličković takes in input a set of node features $h = \{h_1, h_2, \ldots, h_n\}$, with $h_i \in \mathbf{R}^F$, where F is the size of the node's features, and n is the number of the node in the graph. The output of the layers is a vector of new node features $h' = \{h'_1, h'_2, \ldots, h'_n\}$, with $h'_i \in \mathbf{R}^{F'}$. To perform self-attention, each node has a matrix of weights $\mathbf{W} \in \mathbf{R}^{F' \times F}$. The attention coefficient is computed as follow:

$$e_{i,j} = a(\mathbf{W}h_i, \mathbf{W}h_j) \tag{2.10}$$

Where $a(\cdot,\cdot)$ is the attention function and it is defined as $a : \mathbf{R}^{F'} \times \mathbf{R}^{F'} \rightarrow \mathbf{R}$. Given a node i, the equation 2.10 is computed for nodes $j \in N(i)$, where $N(i)$ return the first-order proximity of node i. Note that, the element $e_{i,j}$ represents the importance of the neighbourhood's features on the node i. To make easily the comparison of different nodes equation 2.10 is then normalized as follow:

$$\alpha_{i,j} = softmax(e_{i,k}) = \frac{exp(e_{i,j})}{\sum_{k \in N(i)} exp(e_{i,k})} \tag{2.11}$$

In particular, the attention function used in 2.10 is a feedforward neural network parameterized by a vector $\mathbf{a} \in \mathbf{R}^{2F}$, and it goes into an activation function called LeakyReLU, defined as follow:

$$LeakyReLU = \begin{cases} \beta x & \text{if } x \leq 0 \\ x & \text{if } x \geq 0 \end{cases}$$

with $\beta = 0.2$. Finally, the coefficient compound by the attraction is defined ad follow:

$$\alpha_{i,j} = \frac{exp(LeakyReLU(\mathbf{a}^T[\mathbf{W}h_i || \mathbf{W}h_j]))}{\sum_{k \in N(i)} exp(LeakyReLU(\mathbf{a}^T[\mathbf{W}h_i || \mathbf{W}h_k]))} \tag{2.12}$$

Where the operator $||$ is the concatenation of matrices. Figure 2.5(left) shows how the coefficient is computed, in particular, it can be seen that $\mathbf{W}h_i$ and $\mathbf{W}h_j$ are multiplied by $\mathbf{a}$, then the results go into the LeakyReLU, and finally into the softmax activation function.
The final output features for every node is

$$h'_i = \sigma(\sum_{j \in N(i)} \alpha_{i,j} \mathbf{W}h_j) \tag{2.13}$$

Moreover, to stabilize the process, Veličković repeated the attention process K times, and then concatenate the results, as follow

$$h'_i = ||_{k=1}^{K} \sigma(\sum_{j \in N(i)} \alpha_{i,j} \mathbf{W}h_j) \tag{2.14}$$

Note that the output of 2.14 is a feature matrix in $\mathbf{R}^{K \times F'}$, while with equation 2.13 the output was a single vector of length F.
The stabilization process is shown in figure 2.5(right), in particular, the process is repeated three times $K = 3$, and each colour shows an independent attention computation, finally, the results of each attention are concatenated or averaged to produce the new feature of the node i.

Spektral provides a the Graph Attention convolutional layer, calling the function *spektral.layers.GraphAttention()*. GraphAttention takes in input $\mathbf{X}$ and $\mathbf{A}$, where $\mathbf{X}$ is the matrix representing the node's features and $\mathbf{A}$ is the adjacency matrix. it has several parameters, such as channel, attn_heads, attn_heads_reduction, activation, and others. Channel is the size of the output node's features array. Attn_heads is the number of independent iteration that the attention process does, and its default value is equal to 1. Attn_heads_reduction could be "concat" or "average", and

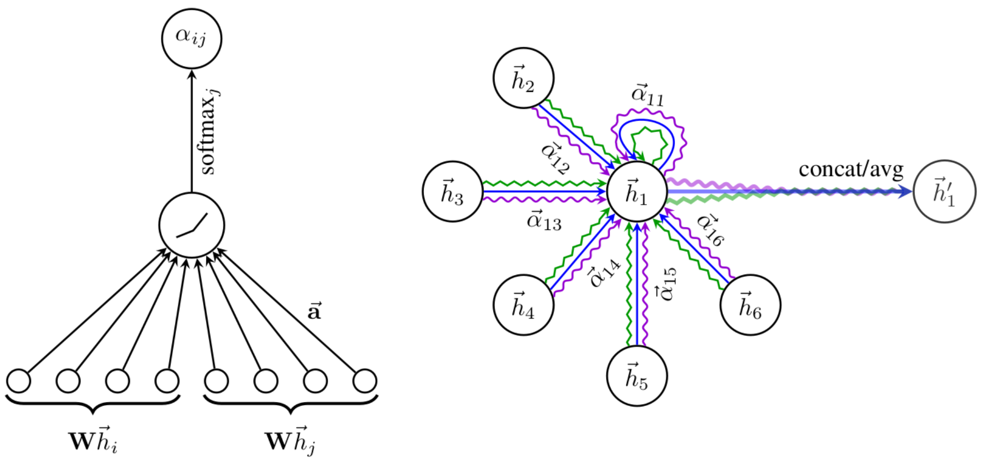


Figure 2.5: Left: the attention mechanism used by the model. Right: an example of a multi attention computation, where $K = 3$.

it defines how different attentions are combined, finally, the parameter activation takes in input an activation function that is used by the layer's nodes.

MinCutPool. The problem of the minimum cut consists of partitioned the set of vertices of a graph minimizing the number of edges cut. Formally, given a graph $G = (V, E)$, K-way normalized minCUT (simply minCUT) is equal to maximize:

$$\frac{1}{K}\sum_{k=1}^{K}\frac{\sum_{i,j\in V_k} E_{i,j}}{\sum_{i\in V_k, j\in V/V_k} E_{i,j}} \tag{2.15}$$

Where K is the number of clusters, the nominator is the sum of the edges within the cluster, and the denominator is the sum of the edges that connect the nodes within the cluster with the nodes outside the cluster. Bianchi et al. [5] develop a polling method based on minCUT. In particular, they map the input features X into a matrix $S \in \mathbf{R}^{N\times K}$, defined as follow:

$$S = softmax(ReLU(\mathbf{XW_1})\mathbf{W_2}) \tag{2.16}$$

Where, W_1 and W_2 are trainable matrices, and then the reduction of the input graph is made by simple matrices multiplications, in particular,

$$\mathbf{A}^{POOL} = \mathbf{S}^T\mathbf{AS}$$
$$\mathbf{X}^{POOL} = \mathbf{S}^T\mathbf{X}$$

The Spektral's function *spektral.layers.MinCutPool()* returns a minCUT pooling layer. The function takes in input $\mathbf{X}$ and $\mathbf{A}$, where $\mathbf{X}$ is the matrix representing node's features and $\mathbf{A}$ is the adjacency matrix, and it returns $\mathbf{X}'$ and $\mathbf{A}'$, that are the reduced features vector and the reduced adjacency matrix. MinCutPool has two main parameters, K and h. K is the number of clusters, thus K is equal to the number of nodes that are kept after the pooling process, while h is the number of hidden units used by the neural network that computes the matrix S of equation 2.16.

In our experiments, the GNN is created with Spektral and it has three GraphAttention (GA) layers, one MinCutPool (MCP) layer and a dense (D) layer. In particular, the network is GA-GA-GA-MCP-D, where GAs layers have 128 components and each Attention are repeated 1 time, while the activation function is ReLU. The MCP layer has k equal to 128. Finally, the Dense layer has c units, where c is equal to the number of classes in the data set of graphs, in our case $c = 2$, and a softmax activation function. Then, Adam optimizer, with learning rate equal to 0.001, is used to compile the network.

2.3 Dimension reduction

Nowadays, an increasing amount of data is produced, moreover, this data lies in a high dimensional real space, where high depends on the application. For example, suppose to consider an image $n \times m$ concatenated as a unique vector, then the vector lies in a $(n \times m \times 3)$-dimensional real space. Thus, a simple 512×512 image is a vector of 786432 dimensions. In this thesis, the aim is to embed a graph in a 2D dimensional space, so dimension reduction is used to reduce the dimensionality of the embedded graph. Moreover, the original representation of the data could be redundant, for example, when the variation is smaller than the sensor's noise, or when many of the variables are correlated with each other.

The formal problem can be defined as follow. Let $\mathbf{X} \in \mathbf{R}^{N \times M}$ be a matrix of samples, where each row $x_i \in \mathbf{R}^M$ represents a sample in a real M-dimensional space, and two metric distance functions δ_m and δ_t. Where $\delta_m : \mathbf{R}^m \times \mathbf{R}^m \to \mathbf{R}$ compute distances between points in $\mathbf{R}^m$, and $\delta_t : \mathbf{R}^t \times \mathbf{R}^t \to \mathbf{R}$ compute distances between points in $\mathbf{R}^t$, with $t \ll m$. A dimensionality reduction is a function $\phi : \mathbf{R}^M \to \mathbf{R}^T$ that maps samples of X into a t-dimensional real space (Y), aiming to minimize the "differences" between distances on different spaces. That is, $\delta_f(x_i, x_k) \sim \delta_f(y_i, y_j)$. Considering that the target space has less degree of freedom with respect to the input space, the function ϕ produces an error. For this reason, ϕ is also defined as the minimization of the squared error $\mathcal{E}_\phi$.

$$\mathcal{E}_\phi = \sum_{1 \le i,j \le n} \mathbf{W}_{i,j}(\delta_m(x_i, x_j) - \delta_t(y_i, y_j))^2 \tag{2.17}$$

Where $\mathbf{W} \in \mathbf{R}^{n \times n}$ is a matrix of weights.

2.3.1 Principal Component Analysis

Principal Component Analysis (PCA) is an important tool in mathematics and computer science, it allows to reduce the input dimension (m) projecting on a linear subspace of lower dimension (t). In particular, it identifies t orthogonal principal components (PC), maximizing the variance of the features, and at the same time minimize the error. Note that the minimization problem in equation 2.17 can be redefined as follow.

$$min \quad ||\mathbf{X} - \mathbf{X}ww^T||_2 \tag{2.18}$$

Where $\mathbf{X}$ is the matrix containing the input samples, w is a unit vector that select the features, and $||\cdot||_2$ is the 2-norm, that is the Euclidean distance. Equation 2.18 can be manipulated as follow:

$$\begin{aligned}
||\mathbf{X} - \mathbf{X}ww^T||_2 &= tr((\mathbf{X} - \mathbf{X}ww^T)(\mathbf{X} - \mathbf{X}ww^T)^T) \\
&= tr((\mathbf{X} - \mathbf{X}ww^T)(\mathbf{X} - ww^T\mathbf{X}^T)^T) \\
&= tr(\mathbf{X}\mathbf{X}^T)) - 2tr(\mathbf{X}ww^T\mathbf{X}^T) + tr(\mathbf{X}ww^Tww^T\mathbf{X}^T) \\
&= constant - tr(\mathbf{X}ww^T\mathbf{X}^T) \\
&= constant - tr(w^T\mathbf{X}^T\mathbf{X}w)
\end{aligned}$$

where $tr(\cdot)$ is the trace of the matrix. Note that minimize a constant minus an expression is equal to maximize the expression. Thus, remembering that $\frac{w^T\mathbf{X}^T\mathbf{X}w}{n-1} = w^T\Sigma w$ is the variance of $\mathbf{X}w$, the minimization problem in equation 2.18 becomes as follow.

$$min \quad (||\mathbf{X} - \mathbf{X}ww^T||_2) = min \quad (constant - constant\ w^T\Sigma w) = max \quad w^T\Sigma w \tag{2.19}$$

This proves that minimize the squared error between distances on the input space ($\mathbf{X}$) and distances on the projected space ($\mathbf{X}w$) is equal to maximise the variance ($w^T\Sigma w$) of the projection.

2.3.2 Uniform Manifold Approximation and Projection for Dimension Reduction

UMAP (Uniform Manifold Approximation and Projection for Dimension Reduction) is an algorithm for dimension reduction developed by McInnes et al. [14]. UMAP can be described in two phases. In the first phase, an undirected weighted graph is constructed from the input. In the second phase, a low dimensional layout of this graph is computed.

Graph construction

In the first phase of UMAP a weighted k-neighbour graph is constructed. Given $\mathbf{X} = \{x_1, x_2, \dots, x_N\}$, $d : \mathbf{X} \times \mathbf{X} \to \mathbf{R}$ and $k \in \mathbf{N}$, where $\mathbf{X}$ is the input data, $d(x_i, x_k)$ is a distance function, and k is the number of nearest neighbor.

For each $x_i \in \mathbf{X}$ compute the set $x_{i,1}, x_{i,2}, \dots, x_{i,k}$, that is the set that contains k nearest neighbour of x_i according to the distance function d.

Successively, for each x_i they compute ρ_i and σ_i, defined as follow:

$$\rho_i = min\{d(x_i, x_{i,j}) | 1 \leq j \leq k\} \text{ with } d(\cdot, \cdot) > 0 \tag{2.20}$$

and set σ_i to be the value such that

$$\sum_{j=1}^{K} exp\left(\frac{-MAX(0, d(x_i, x_{i,j}) - \rho_i)}{\sigma_i}\right) = \log_2(k) \tag{2.21}$$

Then they define the weighted directed graph $\bar{G} = (V, E, w)$, where the vertices V are simply $\mathbf{X}$, while the set of edges are defined as follow:

$$E = \{(x_i, x_{i,j}) | 1 \leq j \leq k, 1 \leq i \leq N\}$$

then the weight function is defined as:

$$w(x_i, x_{i,j}) = exp\left(\frac{-MAX(0, d(x_i, x_{i,j}) - \rho_i)}{\sigma_i}\right) \tag{2.22}$$

Finally, to transform the graph $\bar{G}$ into a undirected graph G, the adjacency matrix A of G is computed as follow:

$$A = \bar{A} + \bar{A}^T - \bar{A} \cdot \bar{A}^T$$

where $\bar{A}$ is the adjacency matrix of $\bar{G}$, and $\cdot$ is the Hadamard product, also known as pairwise product.

Graph Layout

Graph layout algorithms are used to visualize graphs in 2D or 3D. The basic idea is to use forces on nodes and edges, and repeat the application of the forces until an equilibrium is reached. UMAP uses one attractive force applied along the edges and one repulsive force applied among vertices.

In UMAP, given two coordinates $\mathbf{y}_i$ and $\mathbf{y}_j$ of the vertices x_i and x_j, respectively. The attractive force is defined as:

$$\frac{-2ab||\mathbf{y}_i - \mathbf{y}_j||_2^{2(b-1)}}{1 + ||\mathbf{y}_i - \mathbf{y}_j||_2^2} w(x_i, x_j)(\mathbf{y}_i - \mathbf{y}_j)$$

where a and b are hyper-parameter.

Every time that the attractive force is applied to an edge, one of the edge's endpoint is randomly selected and is repulsed using the following repulsive force

$$\frac{b}{(\epsilon + ||\mathbf{y}_i - \mathbf{y}_j||_2^2)(1 + ||\mathbf{y}_i - \mathbf{y}_j||_2^2)} (1 - w(x_i, x_j)(\mathbf{y}_i - \mathbf{y}_j))$$

where ϵ is a small number, avoiding that the denominator became zero.

In brief, given an input data set, UMAP constructs a neighbourhood graph from the data, and then it uses a graph layout algorithm to plot the graphs. Finally, it uses the coordinates of the nodes as the embedding of the input data in a low dimensional space.
The authors of the paper have released a Python library to use UMAP[2]. The main relevant parameter of UMAP are *n_neighbors*, *n_components* and *metric*. *N_neighbors* specifies the number of neighbours used to construct the neighbour graphs. *N_components* is equal to the number of dimension in the embedded space. Finally, the parameter *metric* specifies the metric used to compute distances in both high dimension space and low dimension space. Sadly, UMAP has a bug that makes the embedding nondeterministic, such problem is bypassed in the following through the use of a DNN.

DNN-UMAP. To make the computation of the embedding deterministic, a simple DNN has been used. The DNN is constructed using three dense layers, the input layer has 32 units, the first dense layer has 16 units, the second layer has 8 units and finally the last layer has t units, where t is equal to the dimensions of the output embedding, in our case it is set equal to two. Each layer has a ReLU activation function, except for the last layer that has a sigmoid activation function. The Mean Squared Error loss function is used. The Adam optimization algorithm is used. The network is trained using 100 epochs with a batch size of 32.
In the training phase of the DNN, the training data set is transformed from the input dimension to the target dimension using UMAP, that in our case are 32 and 2 respectively. Then the DNN is trained using the training set as input, and the embedded training set as a target. Then the trained network is used to transform the test data.

2.3.3 Autoencoders

An Autoencoder is a type of Neural Network that works in an unsupervised manner. In the beginning, Autoencoders were developed for data compression, i.e image compression. However, standard compression algorithms, such as JPEG compression, achieved better results concerning an Autoencoder. In 2012 Erhan et al. [4] used Autoencoder to initialize DNN's weights. Later, it has been shown that better initialization of random weights allows achieving the same results. Nowadays, Autoencoders are used for data denoising and dimension reduction for data visualization.
An Autoencoder is composed of an Encoder and a Decoder. The Encoder compresses the input data to a target dimension, while the Decoder reconstructs the input, minimizing reconstruction error. In particular, our Encoder firstly increases its dimension, and successively it reduces the dimension up to the target dimension. The decision to increase the size of the input is due to the latent space. Increasing the input should disentangle the latent variables, that are the values inside the hidden unit.

[2]https://umap-learn.readthedocs.io/en/latest/

3 From Graph to Vector Maximizing Accuracy

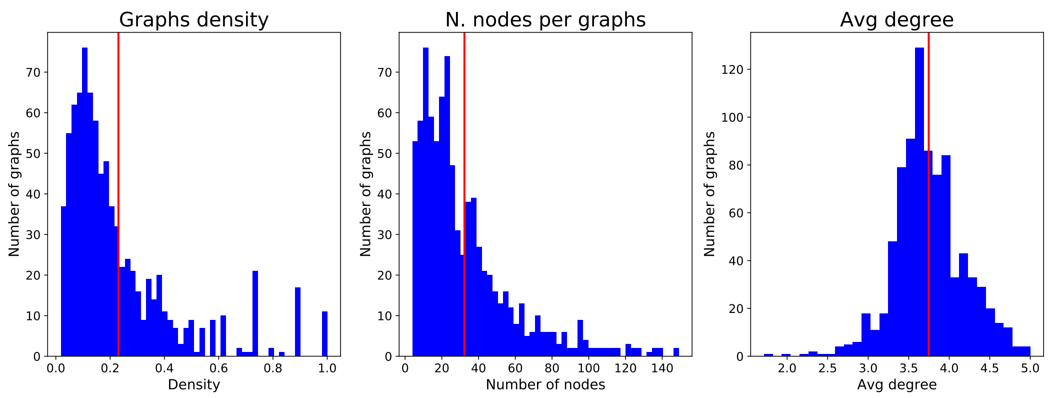
In this chapter, the aim is to embed a graph into the space of real numbers, maximizing the accuracy obtained with 1-Nearest-Neighbour classifier. In particular, three different techniques have been used: Graph Kernel, Graph Neural Network and the combination of the previous two.
Vectorizer is the name of the embedder based on Graph Kernel. The accuracy obtained with 1-Nearest-Neighbour has been studied as the dimensionality of the embedding increases. Successively, a Graph Neural Network (GNN) has been trained on the classification task, and then the output of the second-last layer has been used as the embedding. Finally, the previous two techniques have been combined, adding to each node the embedding of the node itself. The three techniques have been tested on five data sets; one of them represent proteins, while the others represent molecules. For this reason, the first section explains the main features of each data set.

3.1 Data sets

Five data sets have been used to test the techniques, they belong to two main classes Proteins and Molecules. The data set PROTEINS belongs to the first class, while DHFR, Cancer, Leukemia, and AIDS are molecules. Note that all the following data set has been balanced. In the following, the main features of each data set are explained. All the experiments have been run with a laptop Lenovo Thinkpad with an Intel i7 quad-core processor with 2.8GHz and a ram of 16GB, for this reason, all the graphs with more than 150 nodes have been removed, to make feasible run experiments with a laptop.

PROTEINS The data set PROTEINS is labeled with two classes: enzymes and non-enzymes. Such data set has been created by Dobson[3] and it has been downloaded from the Dortmund University website[10]. It is a set of graphs and each of them represents a protein.
Figure 3.1 shows the histogram in blue and the average in red, for each of the selected key features.

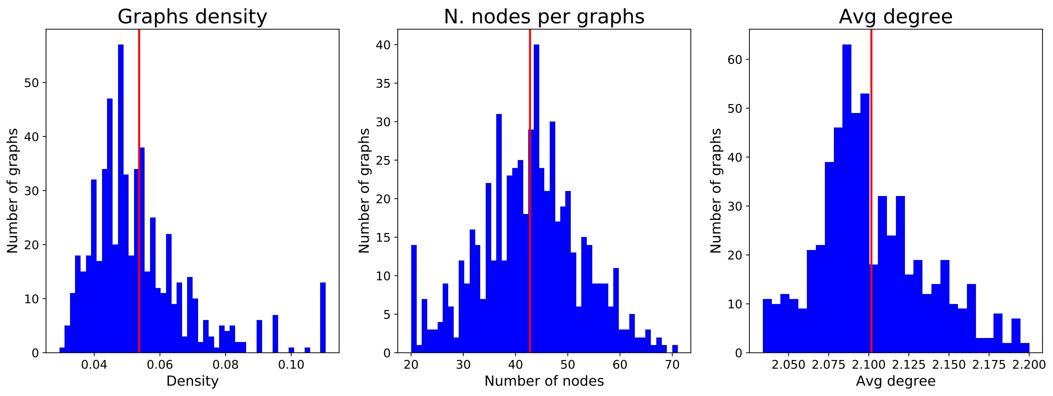
Figure 3.1: Data set: PROTEINS



In particular, PROTEINS has an average density of 0.23 with several graphs with high density, it has 854 graphs with an average of 32.23 nodes per each graph where the smallest graphs has 4 nodes and the biggest one has 149 nodes. A huge deviation can been observed with respect to the mean in comparison with molecules. The minimum degree found in the data set is 1 and the maximum is 12, while the minimum average degree is 1.8 and the maximum average degree is 5.

DHFR DHFR is a data set of inhibitors of dihydrofolate reductase. In particular, molecules have been partitioned by Sutherland[23] according to their capacity to convert dihydrofolate to tetrahydrofolate. Such data set, download from [10], has been balanced. As can be seen from the Figure

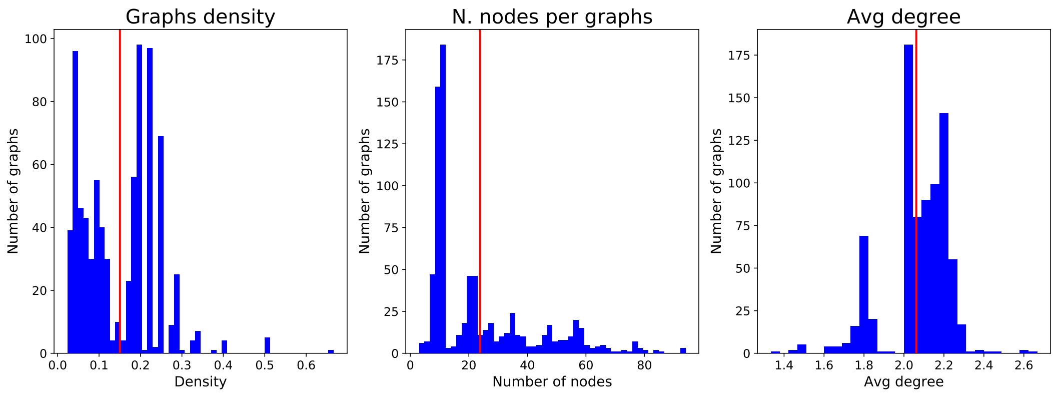
Figure 3.2: Data set: DHFR



3.2, DHFR has 590 graphs, with an average of 42.77 nodes for each graph, the smallest graph in the data set has 20 nodes while the biggest has 71 nodes. The histogram of the density shows that the minimum density is 0.03 and the maximum is 0.19. The smallest degree of the graph in the data set is 1 and the biggest is 4 (they will be the same for all the molecules data sets). The mean of the average density is 2.1.

AIDS The data set AIDS contains 800 molecules compounds. Riesen[19] constructs graphs from the AIDS Antiviral Screen Database of Active Compounds[1]. This data set consists of two classes (active, inactive), which represent molecules with activity against HIV or not. Figure 3.3 shows an

Figure 3.3: Data set: AIDS

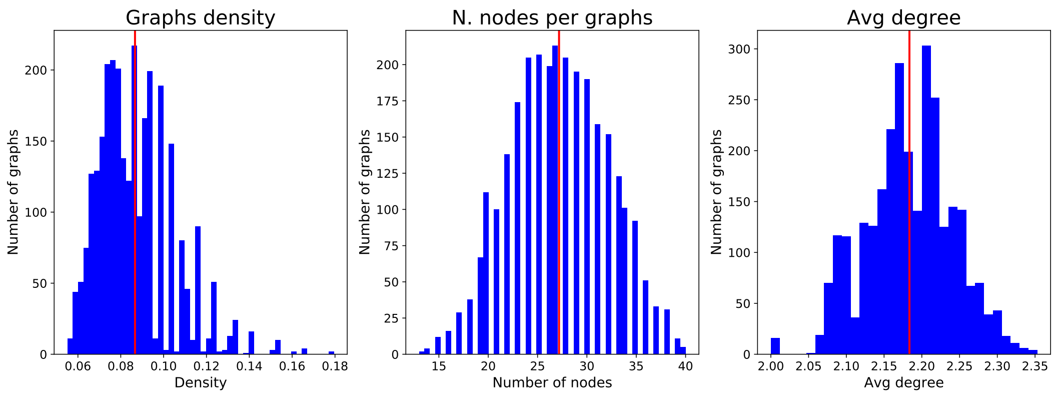


average density 0.15 where the smallest density is 0.03 and the maximum density is 0.67. The data set contains an average of 23.75 nodes for each graph where the smallest graphs have 3 nodes and the biggest has 94 nodes. Finally, it has an average degree of 2.06.

Cancer The data set Cancer has been downloaded from PubChem[2]. It contains molecules compounds active and inactive against the Mdm2, a protein that suppresses the protein p52. If Mdm2 would be inhibited from some molecules then p52 could be used as a new chemotherapeutic strategy. As reported from the figure 3.4 the average density is 0.09 with a maximum of 0.18 and a minimum

[1]The data set can be downloaded from https://wiki.nci.nih.gov/display/NCIDTPdata/AIDS+Antiviral+Screen+Data
[2]https://pubchem.ncbi.nlm.nih.gov/

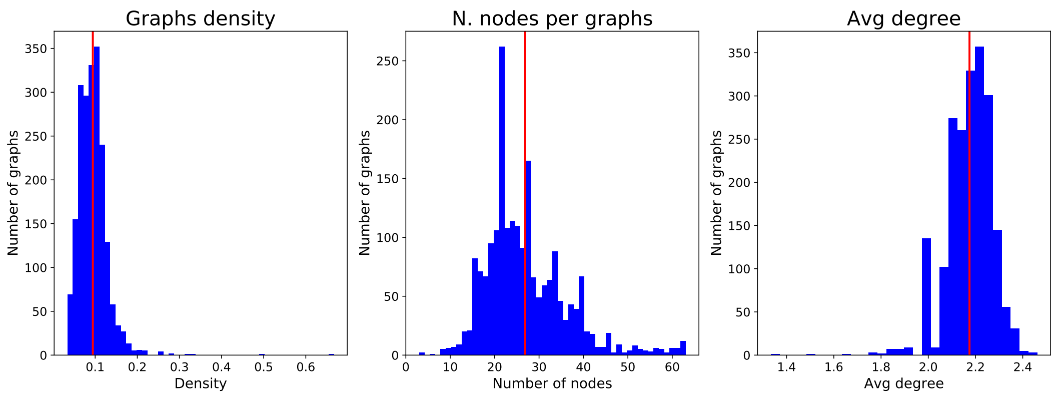
Figure 3.4: Data set: Cancer



of 0.05. The data set contains 2864 graphs representing molecules with an average of 27.18 nodes for each graph, where the smallest graph has 13 nodes and the biggest has 40 nodes. The minimum among the average degrees is 2 and the maximum is 2.35.

Leukemia This data set contains molecules active and inactive against the growth of the HL-60(TB), it is a human Leukemia tumor cell line extracted from a 36-year-old woman who was originally reported to have acute promyelocytic leukemia. The data set contains 2038 molecules with an average

Figure 3.5: Data set: Leukemia



of 26.86 nodes for each graph. The smallest graph in the data set has 3 nodes and the biggest one has 63 nodes. Figure 3.5 show an average density of 0.09 where the maximum density in the data set is 0.67 and the minimum is 0.03. Finally, the minimum average degree is 1.3, the maximum is 2.5 and the mean of the average degrees is 2.17.

Name	**Type**	**# of Graphs**	**# of Nodes**			**Density**			**Degree**		
			Max	Min	Avg	Max	Min	Avg	Max	Min	Avg
PROTEINS	Prot.	854	149	4	32.35	1	0.02	0.23	12	1	3.75
DHFR	Mole.	590	71	20	42.77	0.19	0.03	0.54	4	1	2.10
AIDS	Mole.	800	94	3	23.75	0.67	0.02	0.15	4	1	2.06
Cancer	Mole.	2864	40	13	27.18	0.18	0.05	0.09	4	1	2.18
Leukemia	Mole.	2038	63	3	26.86	0.67	0.03	0.09	4	1	2.17

Table 3.1: Summery data set features

Table 3.1 summarize type, number of graphs and three key feature of the data sets: number of nodes, density, and degree. In particular, it specifies the maximum(Max), the minimum(Min) and the average(Avg) of the selected key feature. It is easy to see that Proteins has the greatest deviation concerning the average for each feature.

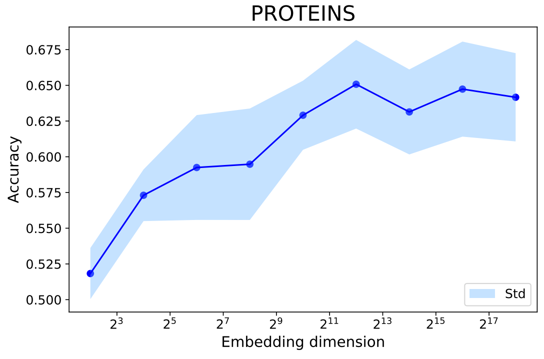
3.2 Graph Kernel

As explained in section 2.1.1 EDeN allow to embed a graph in a real vector space. The square of the parameter *nbits* defines the dimensionality of the embedding. The tested number of bits are shown in table 3.2, it also shows the resulting size of the embedding dimension space. Vectorizer has been used with *complexity* equal to 2. The accuracy has been computed averaging the results obtained

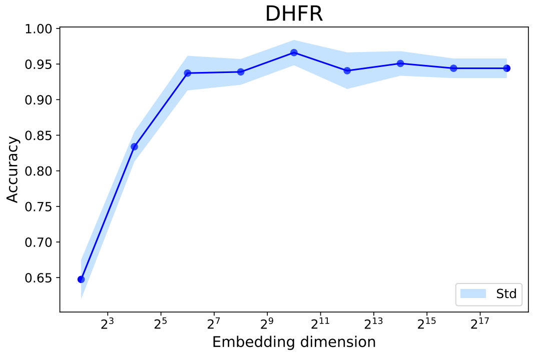
nbits	Embedding dim. space
2	4
4	16
6	64
8	256
10	1024
12	4096
14	16384
16	65536
18	262144

Table 3.2: Conversion from the number of bits to embedding dimensionality.

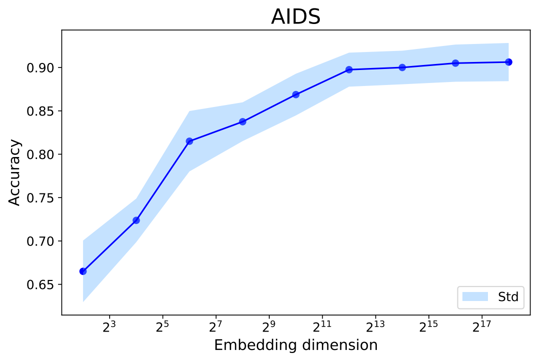
with *5*-fold cross-validation, and 1-Nearest-Neighbours classifier. As shown in figure 3.6 every data set report an increase of the accuracy when the parameter *nbits* grow. In particular, all the data sets tend to have a quick-rising up to 8 bits and then they stabilize. The parameter *nbits* has been set equal to 8 for all the data sets in the following experiments, to have a trade-off between accuracy and embedding size, recalling from table 3.2 that 8 bits will be a vector of 256 elements.



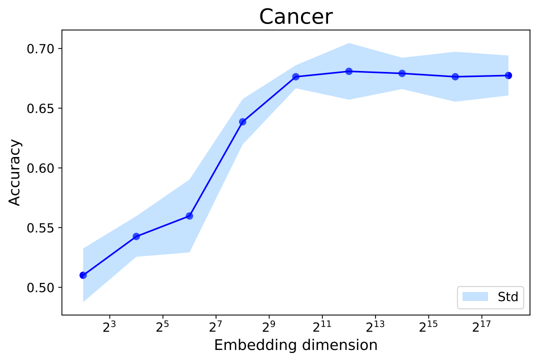
(a) PROTEINS



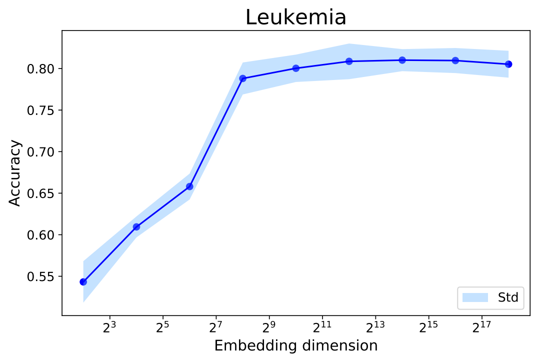
(b) DHFR



(c) AIDS



(d) Cancer



(e) Leukemia

Figure 3.6: This figure reports the accuracy obtained using Vectorize Vertex and Random Forest, as the embedding dimensionality size increases.

3.3 Graph Neural Network

Section 2.2.2 explained how Spektral works. In this section, the output of the second-last layer of the GNN is used to produce the embedding. The input data sets have been divided into train and test, 70% and 30% of the entire data set respectively, and after the training phase, the second-last layer has been removed from the trained model, in such way, for each input graph, the network produces an embedding in a vector of 128 elements. To avoid over-fitting, a Callback called EarlyStopping has been used, it monitors the trend of the validation loss, and when it does not decrease for a fixed period of epochs (called *patience*), it stops the execution. All the following experiments have been run with 200 epochs and the parameter patience was set to 25.
Table 3.3 summarize the accuracy obtained from 1-Nearest-Neighbours classifier on a real vector space with 128 dimensions.

Data set	Accuracy	Std
PROTEINS	0.70	±0.03
DHFR	0.80	±0.02
AIDS	0.98	±0.01
Cancer	0.58	±0.01
Leukemia	0.61	±0.02

Table 3.3: Accuracy obtained with 1-Nearest-Neighbour classifier on an embedding in a 128-dimensional real space, produced by the second-last layer of Spektral.

3.4 Graph Kernel and Graph Neural Network

As explained in the section 2.1.1 EDeN allow to embed the nodes of a graphs in a vector space, thus in this section, the embedding of the node has been added as attribute of the node itself, then the GNN has been used on the manipulated graph to embed it in a real space of 128 dimensions. Finally, the embedding has been evaluated computing the accuracy with 1-Nearest-Neighbour classifier in 5-fold cross-validation.

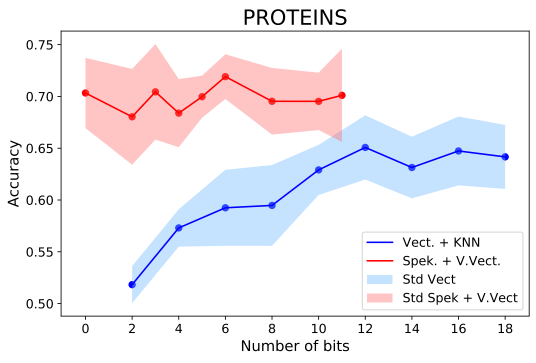
Figure 3.7 summarize the results of this chapter. Results obtained in section 3.2 are shown in blue. While the red line shows the results obtained using Spectral and Vectorize Vertex from EDeN. Moreover, the red point with abscissa equal to 0 shows the results obtained from Spektral without any additional information on the node of the graphs.

Figure 3.7a shows the results obtained on PROTEINS. As expected, Spektral does not take advantage from added information by the kernel, in fact, the accuracy tends to be around 0.7, this is since Vectorizer is a Kernel specifically developed for molecules. Looking at the results obtained from the Vectorizer can be seen an increase of 0.15. However, Spektral performs better then Vectorizer.

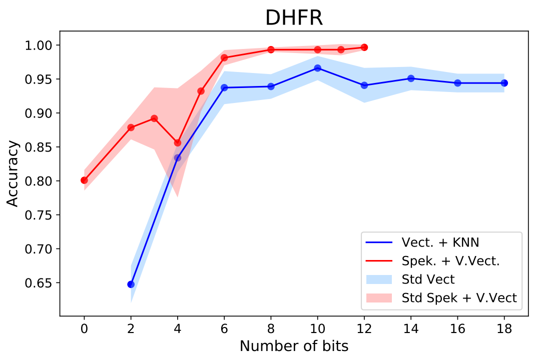
The results obtained on DHFR are shown in figure 3.7b. Adding information to the nodes of the graphs boosts the accuracy from 0.8 up 1, while Vectorizer is still less powerful than Spektral.

Figure 3.7c shows the results obtained on AIDS, again Spektral win against Vectorizer, the best accuracy obtained is 1, with Spektral and 3 bits for Vectorizer Vertex.

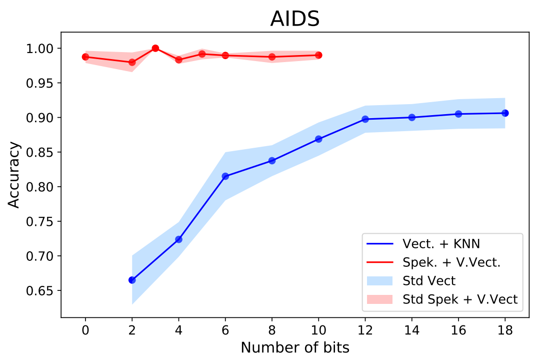
Finally, Figures 3.7d and 3.7e show the results on Cancer and Leukemia. In these cases, up to 8 an 6 bits respectively, Spektral performs better than Vectorizer, and then Vectorizer obtains a better accuracy. However, an interesting note is that Spektral produces an embedding in a real space of 128 elements, while Vectorizer produces an embedding of 2^{nbits}. In particular, Both the data set obtain the best accuracy with 12 bits, that is an embedding in a real space with 4096 dimensions.



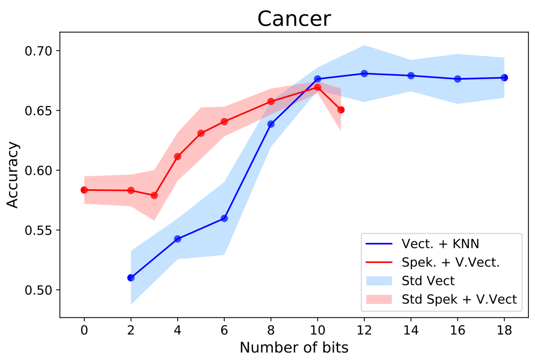
(a) PROTEINS



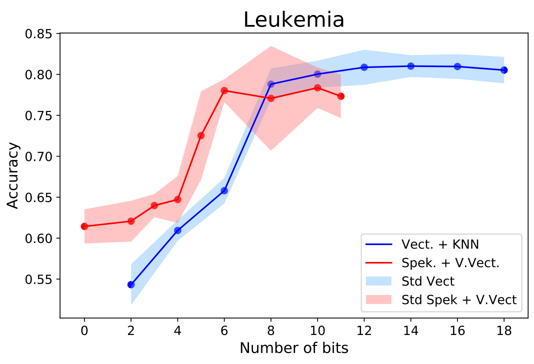
(b) DHFR



(c) AIDS



(d) Cancer



(e) Leukemia

Figure 3.7: Comparison of the accuracy of 1-Nearest-Neighbour on all data sets for the three techniques proposed.

Table 3.4 shows the accuracy obtained with 1-Nearest-Neighbour classifier for each method developed in this chapter. From the table, it is easy to see that three out of five data sets achieve better results with the model based on the combination of Spektral and Vectorizer. Even if, the Vectorizer achieve better results in two data sets, it produces an embedding in a real space with several dimension more with respect to the model based on Spektral.

	Spektral	Vectorizer			Spektral plus Vectorizer
	$\mathbf{R}^{128}$	$\mathbf{R}^{256}$	$\mathbf{R}^{1024}$	$\mathbf{R}^{262144}$	$\mathbf{R}^{128}$
PROTEINS	0.70	0.59	0.63	0.65	**0.70**
DHFR	0.80	0.94	0.94	0.95	**1**
AIDS	0.98	0.85	0.87	0.91	**1**
Cancer	0.58	0.63	0.66	**0.67**	0.64
Leukemia	0.61	0.79	0.80	**0.81**	0.77

Table 3.4: Accuracy of 1-Nearest-Neighbour classifier in different embedding spaces, using three different techniques.

3.5 Limitations

This chapter shows how three different techniques perform on the five data sets. However, this thesis aim is to embed graphs in a low dimension real vector space, and 128 is still too large to visualize, optimize and detect out layers. In the following chapters the dimensionality of the embedding will be reduced using others technique, starting from a vector of 128 elements produced by Spektral and Vertex Vectorizer with 8 bits.
A fourth experiment has been done, producing a node embedding with 14, 16, and 18 bits and then the dimension has been reduced using PCA up to 128, 256 and 512 elements. However, such a technique did not show any improvement, for this reason, results are not shown.

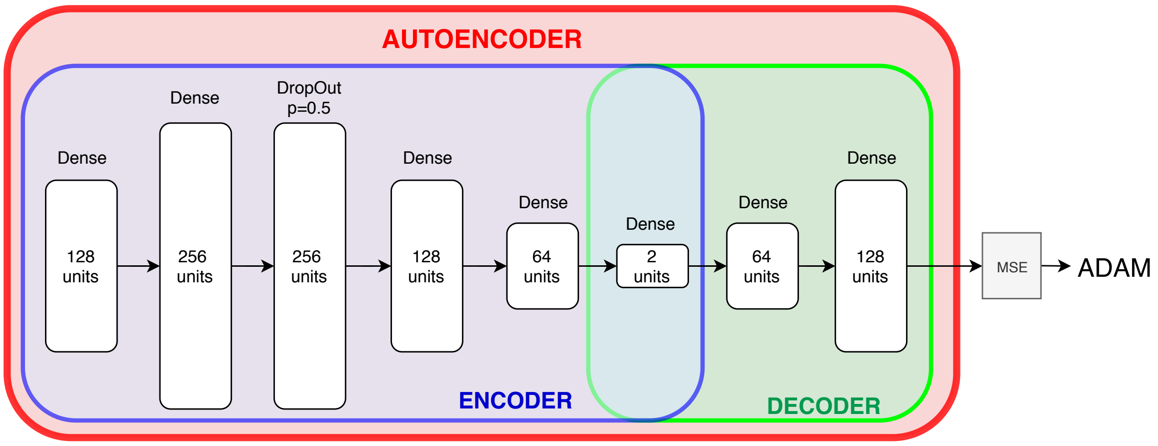
4 From Vectors to Low Dimensionality Vectors Maximizing Accuracy

This chapter aims to reduce the dimension of the embedded graph from 128 dimensions to two dimensions, to do that several techniques have been implemented and tested. In particular, the first approach is to reduce the dimensions using an Autoencoder directly to the target dimensions. However, the first approach reports pore results. For this reason, the size of the Autoencoder embedding has been increased up to 32, and then the new embedding dimensions have been reduced to 2 using DNN-UMAP.
Successively, because an Autoencoder does not take any advantages from the belonging class of the input, a new Autoencoder has been developed, aiming to take under consideration the input belonging class, such Autoencoder has been called Supervised Autoencoder, and it has been used to reduce the dimensions up to 32. However, this new model requires to weight the losses of the Autoencoder and the Classifier, to do this, a specific callback, called Sinusoidal Callback, has been developed. Finally, the 32 dimensions real space has been reduced to a two dimensions space using DNN-UMAP.

4.1 Autoencoder to target

In this section, the dimension of the input vector is reduced from 128 to 2, and to do this Autoencoders have been used. An Autoencoder is an unsupervised Neural Network to rebuild the input after compressing it.
Figure 4.1 shows the architecture of the used network. In particular, the Autoencoder is composed by an Encoder and a Decoder, the Encoder compresses the input size, while the decoder rebuilds the compressed input to the original size. The Encoder, which has 5 dense layers and 1 dropout layer, increases the size of the input before compressing it to the target dimension, while the Decoder has two dense layers. Each layer has a Rectified Linear Unit (ReLU) activation function, except for the last layer of the Encoder that has a sigmoid activation function. ADAM algorithm has been used as the optimizer and Mean Square Error as the loss function.

Figure 4.1: Autoencoder used to reduce the dimensions of the graph embedding up to 2.



To improve the performance of the Autoencoder a preprocessing has been done on the input data. In particular, all the elements in the data set has been normalized along the dimensions using *MinMax*

Scaling. Equation 4.1 shows the formula.

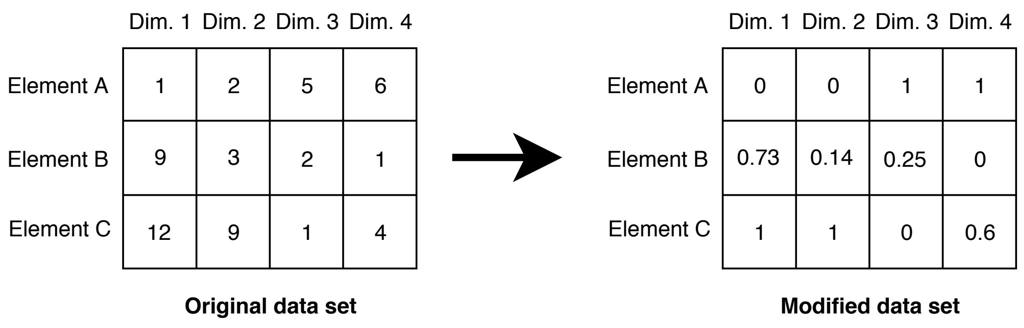
$$X_{scaled} = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{4.1}$$

To better explain the preprocessing, figure 4.2 shows a toy example, using the following data set of three elements in $\mathbf{R}^4$:

$$\begin{aligned} \text{A} =& \quad [1,2,5,6] \\ \text{B} =& \quad [9,3,2,1] \\ \text{C} =& \quad [12,9,1,4] \end{aligned}$$

The preprocessing procedure is applied to the dimensions, thus the column $[1, 9, 12]$ is scaled into $[0, 0.73, 1]$. Figure 4.2 shows the scaling of the entire toy data set. For each data set, the following

Figure 4.2: Toy example of preprocessing.



	Dim. 1	Dim. 2	Dim. 3	Dim. 4
Element A	1	2	5	6
Element B	9	3	2	1
Element C	12	9	1	4

Original data set

	Dim. 1	Dim. 2	Dim. 3	Dim. 4
Element A	0	0	1	1
Element B	0.73	0.14	0.25	0
Element C	1	1	0	0.6

Modified data set

procedure has been repeated:

1. Preprocess the input data set.
2. Train the Autoencoder on the training set.
3. Use the Encoder to compress the entire data set.
4. Evaluate the performance of 1-Nearest-Neighbour classifier on the compressed train and test set.

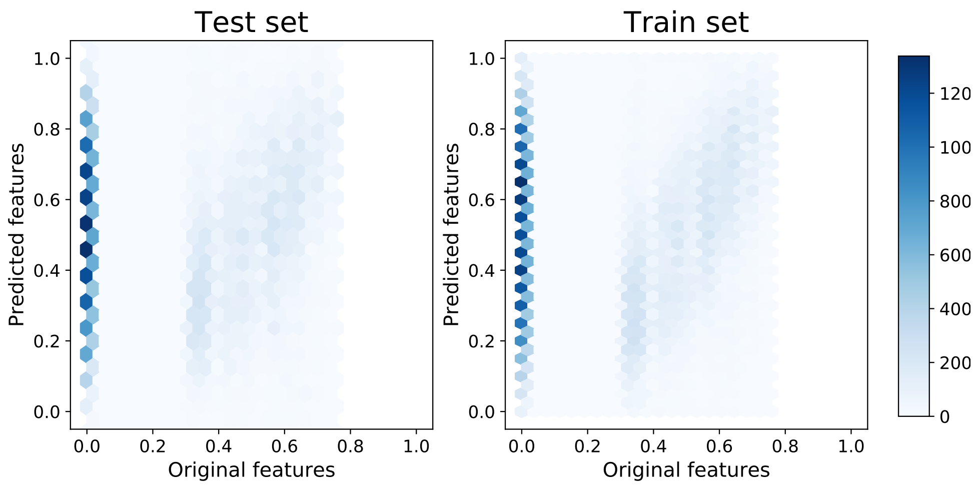
Table 4.1 shows the averaged results of the previous procedure applied to each data set, and the **Accuracy Loss** that is the loss of accuracy obtained by 1-Nearest-Neighbour classifier passing from $\mathbf{R}^{128}$ to $\mathbf{R}^2$. It is notable that, all the data sets have been balanced and each of them is a binary classification problem, so a random classifier would obtain 50% accuracy. For this reason, such a technique cannot be considered successful.

Name	**Train**		**Test**		
	Acc	**Std**	**Acc**	**Std**	**Accuracy Loss w.r.t $\mathbf{R}^{128}$**
PROTEINS	0.49	±0.00	0.53	±0.02	-0.17
DHFR	0.50	±0.00	0.52	±0.02	-0.48
AIDS	0.50	±0.01	0.53	±0.01	-0.47
Cancer	0.52	±0.01	0.51	±0.01	-0.13
Leukemia	0.49	±0.00	0.53	±0.02	-0.25

Table 4.1: The accuracy obtained with 1-Nearest-Neighbour classifier on all the data sets in a real space with 2 dimensions, and loss of accuracy concerning the same classifier on 128 dimensions on the test set.

Such poor results are due to the fact that the Autoencoder is not able to reconstruct the input, neither in the train nor test set. Figure 4.3 shows the comparison of the original features and the one predicted from the Autoencoder, for both the train and the test set, while the darkness shows the density of the features. A perfect Autoencoder should draw a line of 45 degrees passing through the points $[0, 0]$ and $[1, 1]$. Clearly, the Autoencoder is not able to reconstruct the input. For this reason, in the following section, the target dimension is increased.

Figure 4.3: Reconstruction error on DHFR.



4.2 Autoencoder to middle dimension and DNN-UMAP

In this section, the number of dimensions is reduced, using an Autoencoder, the idea is to maintain the accuracy obtained, with 1-Nearest-Neighbour classifier, in the previous chapter on the embedding of the graphs in vectors of 128 elements. Successively, the new target dimension will be reduced with another technique.

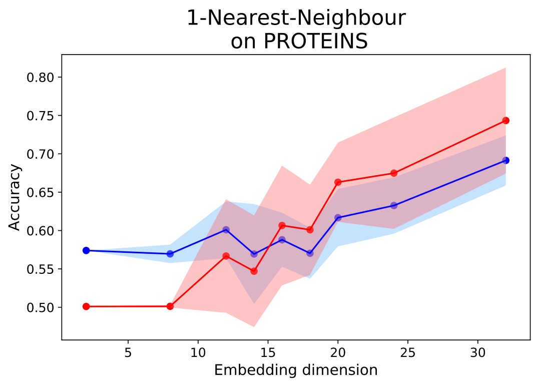
4.2.1 Medium dimension analysis

To find the new target dimension, the obtained performances have been studied when the dimension of the last layer of the encoder, shown in figure 4.1, grow. In particular, the following dimensions are the ones that have been tested: 2,8,12,14,16,18,20,24 and 32, tracing the reconstruction loss and the accuracy of 1-Nearest-Neighbour classifier on the embedding space. Again, each experiment has been repeated 5 times in cross-validation, the Autoencoder has been trained with 300 epochs, and the patience parameter of the EarlyStopping is equal to 20.
Figure 4.4 shows the results obtained on the data set PROTEINS. In particular, 4.4a shows the accuracy obtained from 1-Nearest-Neighbour classifier when the bottleneck of the Autoencoder increases. While figure 4.4b shows the mean square error of the Autoencoder. On the data set PROTEINS, the best results are achieved when the bottleneck of the Autoencoder is 32, obtaining an averaged accuracy of 1-Nearest-Neighbour classifier equal to 0.69 with a standard deviation of 0.03 on the test set, while, the mean of the reconstruction error is equal to 0.15 with a standard deviation of 0.04.

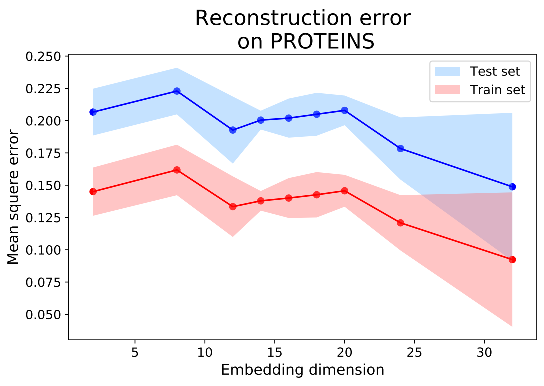
Figure 4.5 shows the results obtained on the data set DHFR. Again, the best accuracy has been achieved within an embedding of 32 units. The average 1-Nearest-Neighbour accuracy is 0.99 with a standard deviation of 0.01 on the test set, while, the average reconstruction error on the test set is equal to 0.01.

Notable results have been achieved on the data set AIDS. The accuracy obtained on a real space of 128 dimensions has been maintained on an encoding of 32 units. In particular, figure 4.6a reports an averaged accuracy of 1 of 1-Nearest-Neighbour classifier, with 0 standard deviation.

On the data set Cancer, we obtained an increase of classification accuracy from 0.51 (in a 2 dimension space) up to 0.57 (in a real space with 24 dimensions), while, the minimum reconstruction error has been obtained with 32 units, and it is equal to 0.02.

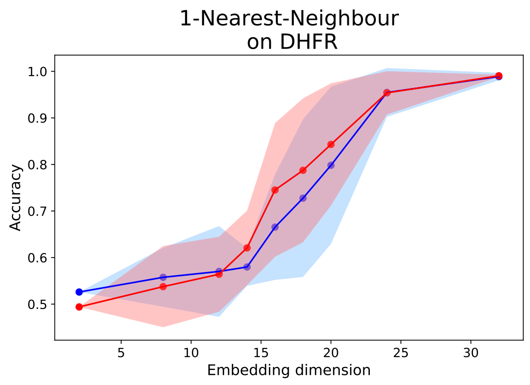


(a) Accuracy of 1-Nearest-Neighbour classifier.

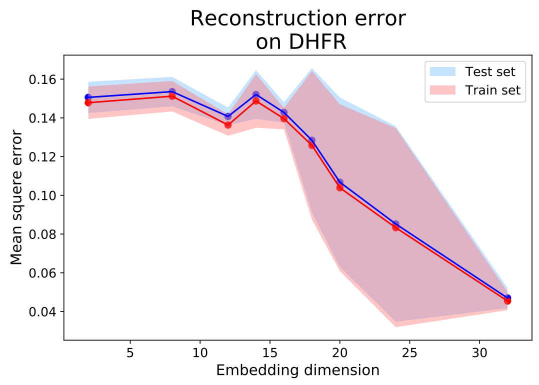


(b) Reconstruction error.

Figure 4.4: Accuracy of 1-Nearest-Neighbour classifier on the embedded space, and Mean Square Error of Autoencoder, reconstructing the input. On the PROTEINS data set.

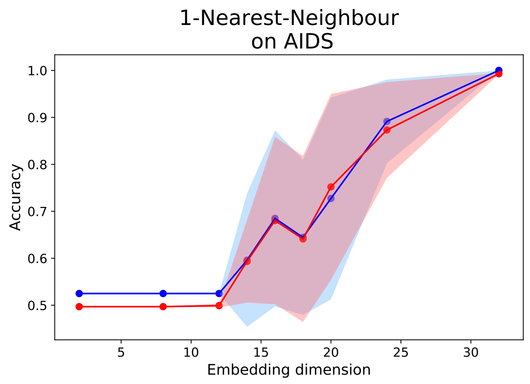


(a) Accuracy of 1-Nearest-Neighbour classifier.

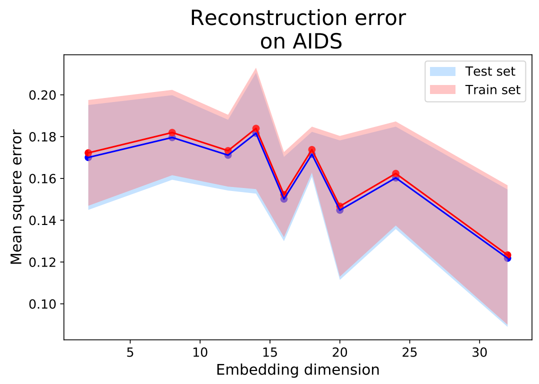


(b) Reconstruction error.

Figure 4.5: Accuracy of 1-Nearest-Neighbour classifier on the embedded space, and Mean Square Error of Autoencoder, reconstructing the input. On the DHFR data set.



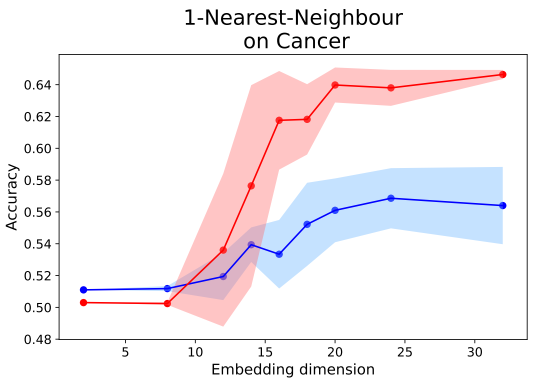
(a) Accuracy of 1-Nearest-Neighbour classifier.



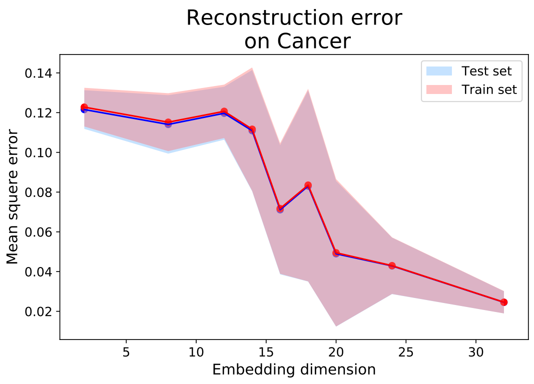
(b) Reconstruction error.

Figure 4.6: Accuracy of 1-Nearest-Neighbour classifier on the embedded space, and Mean Square Error of Autoencoder, reconstructing the input. On the AIDS data set.

Finally, on the data set Leukemia, an averaged accuracy of 0.7 has been obtained in an embedding of 24 units, while, the minimum reconstruction error is equal to 0.04 and it has been obtained when the inner bottleneck of the Autoencoder is a dense layer of 32 units.

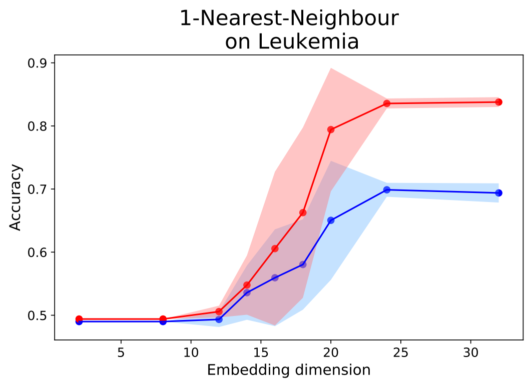


(a) Accuracy of 1-Nearest-Neighbour classifier.

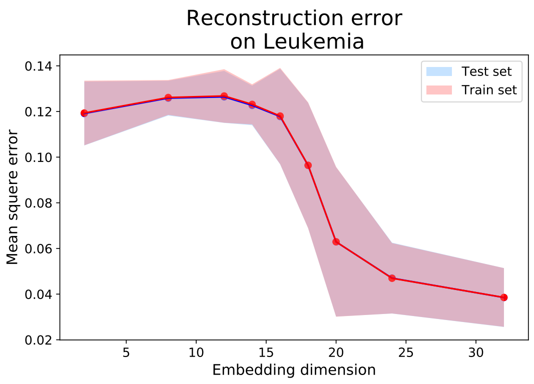


(b) Reconstruction error.

Figure 4.7: Accuracy of 1-Nearest-Neighbour classifier on the embedded space, and Mean Square Error of Autoencoder, reconstructing the input. On the Cancer data set.



(a) Accuracy of 1-Nearest-Neighbour classifier.



(b) Reconstruction error.

Figure 4.8: Accuracy of 1-Nearest-Neighbour classifier on the embedded space, and Mean Square Error of Autoencoder, reconstructing the input. On the Leukemia data set.

Even if the best embedding dimensions for the data set Cancer and Leukemia is 24, encoding in a 32 dimension space is used, for two main reasons. Firstly, the majority of the data sets achieve the best results with an encoder in a 32 dimensions space. Secondly, in all the data sets the minimum reconstruction error is achieved on an embedding of 32 dimensions.
Figure 4.9 compares the ability of the Autoencoder to reconstruct the input. In particular, it shows the predicted features on the ordinate axis and the real features on the abscissas axis. The first two plots show the reconstruction error on test and train set when the size of the encoding is fixed to 32. While the others show the reconstruction error on the test and train set when the encoding is to a 2 dimension real space. Clearly, the Autoencoder archives better results in both train and test set passing through 32 units instead of 2 units.

Figure 4.9: Reconstruction error of the Autoencoder passing through 32 units and 2 untis, in both test and train set.

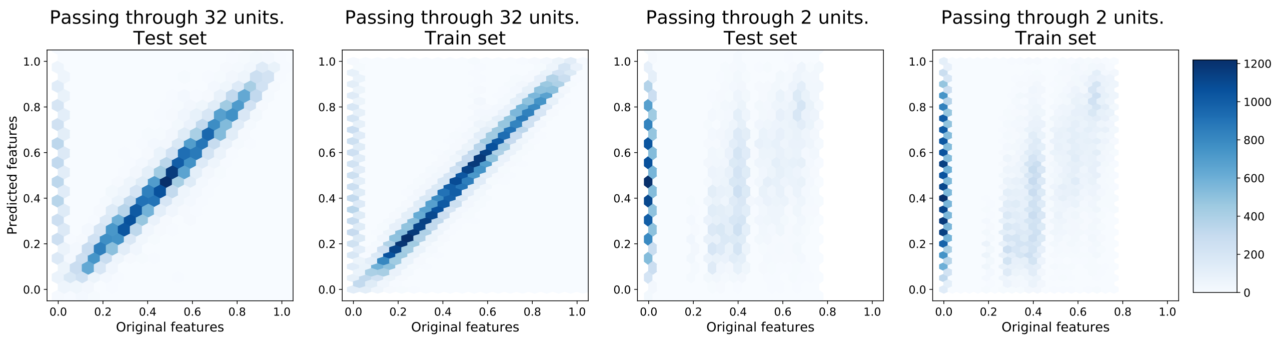


Table 4.2 summarizes the results obtained so far, showing that a simple Autoencoder with an encoding to a vector of 32 dimensions can keep the performance obtained on a vector of 128 elements in three out of five data set. As explained in section 2.3.3 an Autoencoder is an unsupervised deep learning architecture, such technique does not take any advantage from the input label, this could explain the loss of 10% of the accuracy on the data sets Leukemia and Cancer.
However, an embedding in 32 dimensions is still too large to be visualized or optimize using technique such as Bayesian Optimization.

Name	**Train**		**Test**		
					Accuracy Loss w.r.t $\mathbf{R}^{128}$
	Acc	**Std**	**Acc**	**Std**	
PROTEINS	0.74	±0.07	0.69	±0.03	-0.01
DHFR	0.99	±0.00	0.99	±0.01	-0.01
AIDS	0.99	±0.00	1.00	±0.00	0.00
Cancer	0.64	±0.01	0.56	±0.01	-0.08
Leukemia	0.83	±0.01	0.69	±0.01	-0.09

Table 4.2: The accuracy obtained with 1-Nearest-Neighbour classifier on all the data sets in a real space with 32 dimensions, and loss of accuracy concerning the same classifier on 128 dimensions on the test set.

4.2.2 DNN-UMAP

In this subsection, the dimension of the input vector is reduced using DNN-UMAP, explained in section 2.3.2. The used Deep Neural Network has been trained with 100 epochs and the patience parameter of the Early Stopping was 10. Each experiment has been repeated five times to compute the standard deviation.
The following figures show the embedding obtained on the train and test set. Since each experiment has been repeated five times, the shown embedding is the one that reports an accuracy closer to the mean of the accuracy of the five experiments. Each figure shows in red the positive samples and in blue the negative ones. A Gaussian Kernel Density Estimator has been fitted on both, negative and positive samples, to show level curves and density of the color.
Figure 4.10 shows the embedding of the data set PROTEINS. In particular, the reported embedding obtained an accuracy of 0.70 and 0.81, respectively on test and train set. While the averaged accuracy is 0.69 on test and 0.81 in the train set. As can be seen from the figure, the algorithm tries to create three islands of points, two of them containing positive samples and one of them negative. An interesting analysis that could be done, is to investigate the main features of the graphs within the same target but in different islands.

The following figure shows the embedding of the data set DHFR. Again, we report one embedding out of five experiments we did. Again, we report the results that have the accuracy of the embedding closer to the average accuracy, and in this case, figure 4.11 reports the embedding that obtained the

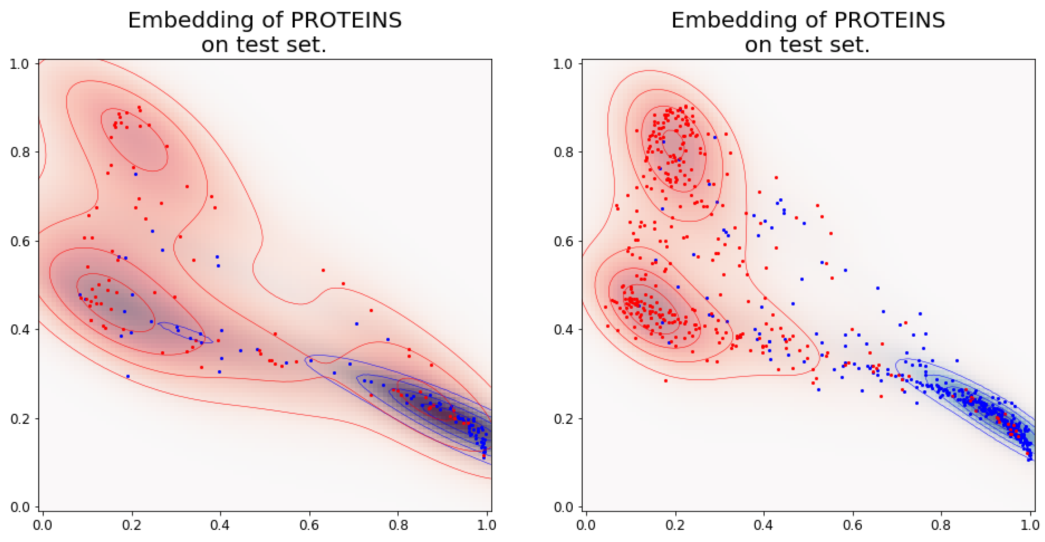


Figure 4.10: PROTEINS graphs embedded in a two-dimensional space using Spektral + Autoencoder + DNN-UMAP

same accuracy of the average among the experiments, in both test and train set.

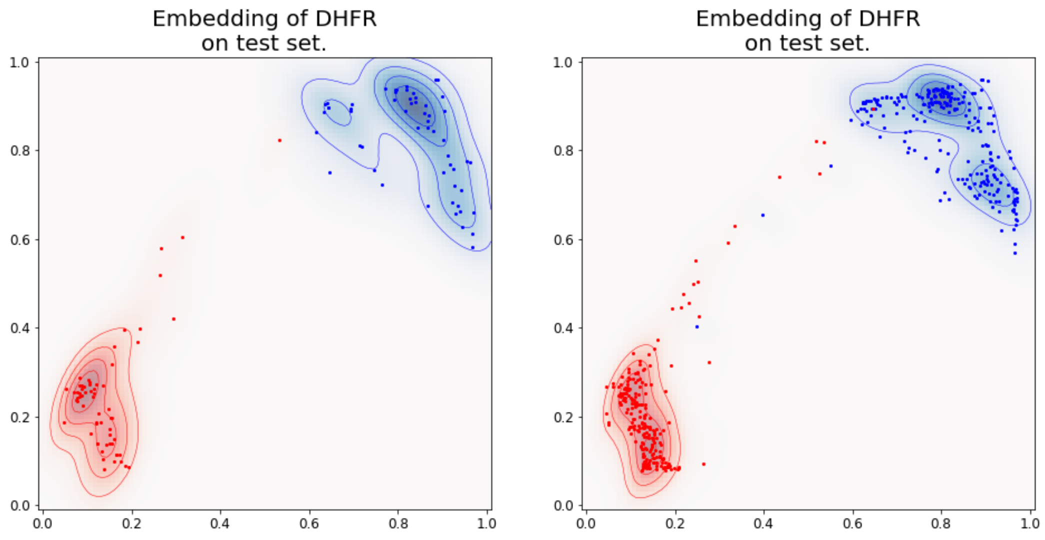


Figure 4.11: DHFR graphs embedded in a two-dimensional space using Spektral + Autoencoder + DNN-UMAP

Figure 4.12 reports the embedding of the data set AIDS. The reported embedding obtained an accuracy of 0.99 on the test set, and an accuracy of 0.99 on the train set, identical to the averaged results along with the experiments.

The embedding shown in figure 4.13 reports an accuracy of 0.59 on the test set and 0.83 on the train set. While the average accuracy on Cancer data set embedded in a 2D space, is 0.59 on the test set and 0.82 on the training set, with a standard deviation of 0.01 and 0.00 in test and train set respectively. Such results are less satisfying concerning the previous one. However, we want to remain that the accuracy obtained in the input dimension was 0.56 on the test set and 0.64 on the train set,

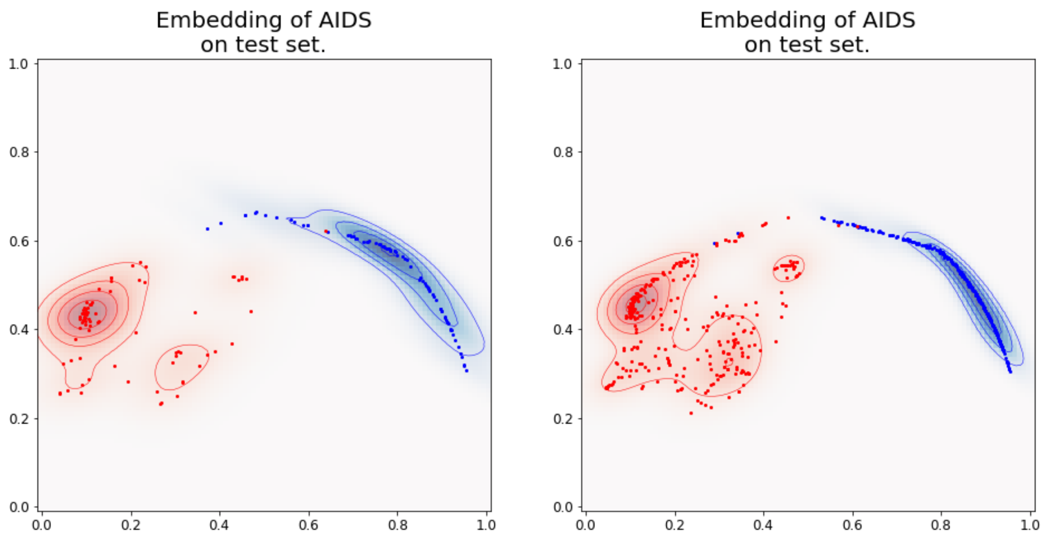


Figure 4.12: AIDS graphs embedded in a two-dimensional space using Spektral + Autoencoder + DNN-UMAP

so even if the results are not excellent we still improved the accuracy passing from $\mathbf{R}^{32}$ to $\mathbf{R}^2$.

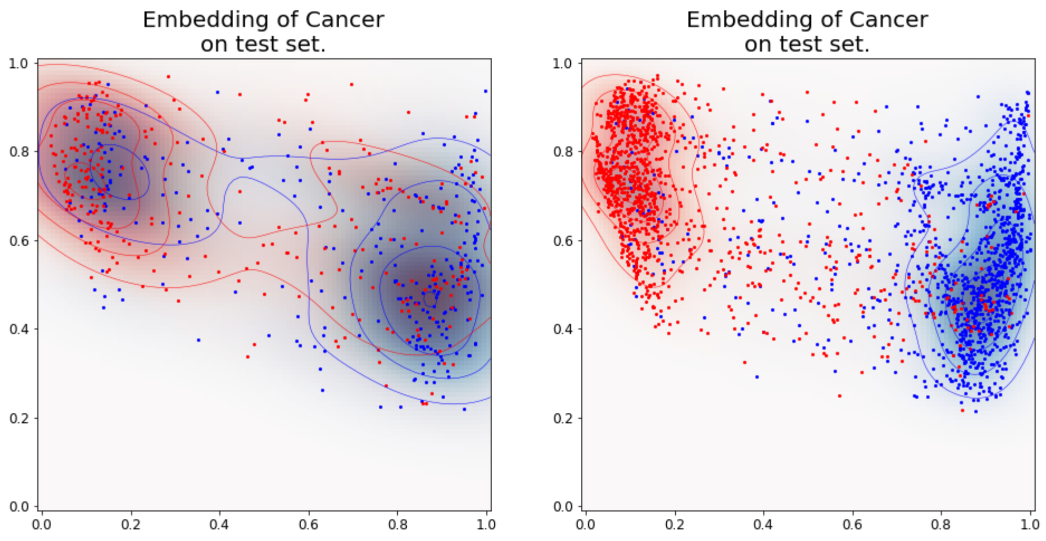


Figure 4.13: Cancer graphs embedded in a two-dimensional space using Spektral + Autoencoder + DNN-UMAP

Figure 4.14 shows the embedding of the data set Leukemia. In particular, the shown embedding obtained an accuracy of 0.69 on the test set and 0.70 on the train set. While the average accuracy of the five experiments is 0.69 on the test set and 0.89 on the train set.

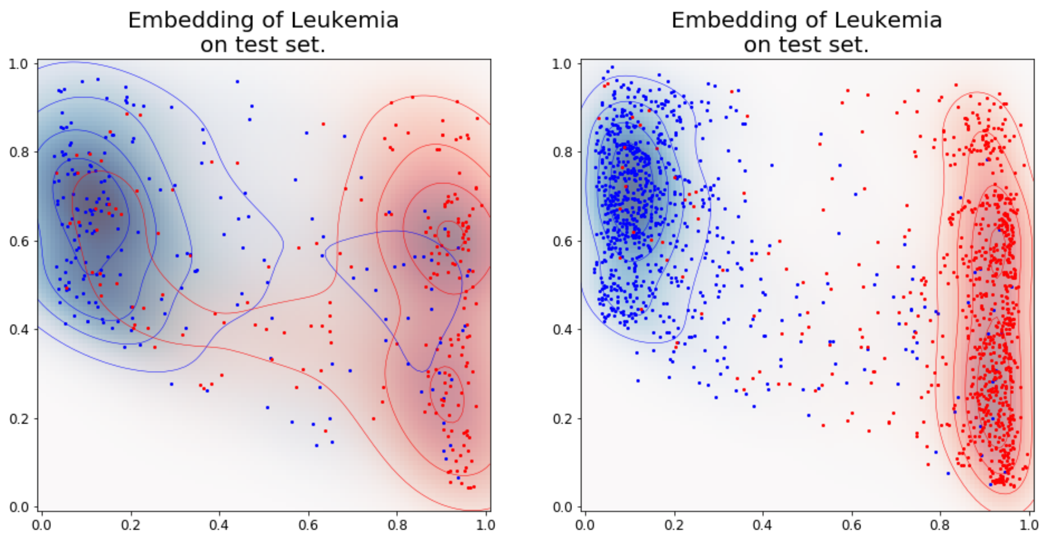


Figure 4.14: Leukemia graphs embedded in a two-dimensional space using Spektral + Autoencoder + DNN-UMAP

Table 4.3 summarize the obtained accuracy from 1-Nearest-Neighbour applied to the embedding of each data set. In particular, can be seen a worsening in the accuracy on the data set AIDS, an improvement on the data sets PROTEINS and Cancer, while the accuracy obtained on DHFR and Leukemia has been held. Moreover, this method can be considered stable because the obtained standard deviation shows a limited divergence from the mean.

Name	**Train**		**Test**		
	Acc	**Std**	**Acc**	**Std**	**Accuracy Loss w.r.t R^{32}**
PROTEINS	0.81	±0.01	0.70	±0.02	+0.01
DHFR	0.99	±0.00	0.99	±0.00	0.00
AIDS	0.99	±0.00	0.99	±0.00	-0.01
Cancer	0.82	±0.00	0.59	±0.01	+0.03
Leukemia	0.89	±0.00	0.69	±0.02	0.00

Table 4.3: The accuracy obtained with 1-Nearest-Neighbour classifier on all the data sets in a real space with 2 dimensions, and loss of accuracy concerning the same classifier on 32 dimensions on the test set.

Finally, figure 4.15 summarizes the obtained accuracy for each tested embedding dimension on all the data set. In general, DNN-UMAP preserves the accuracy from the previous space or even it improve it. While the Autoencoder worsens the performance in four out of five data sets. Note that the Autoencoder reduce the dimension of the input data in an unsupervised manner, for this reason, in the next section, the Autoencoder is replaced by a Supervised Autoencoder.

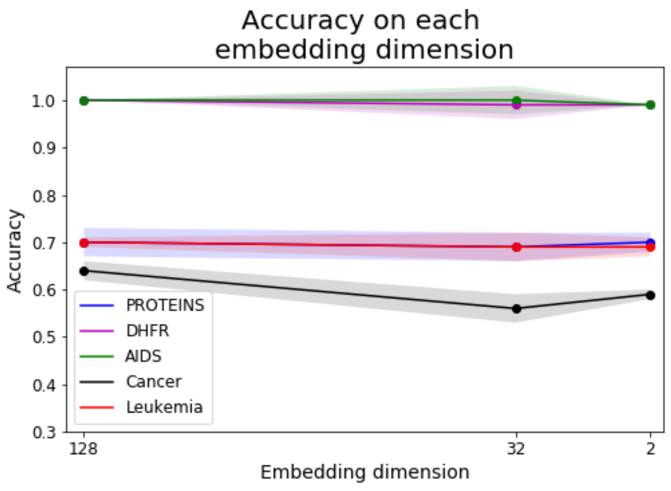
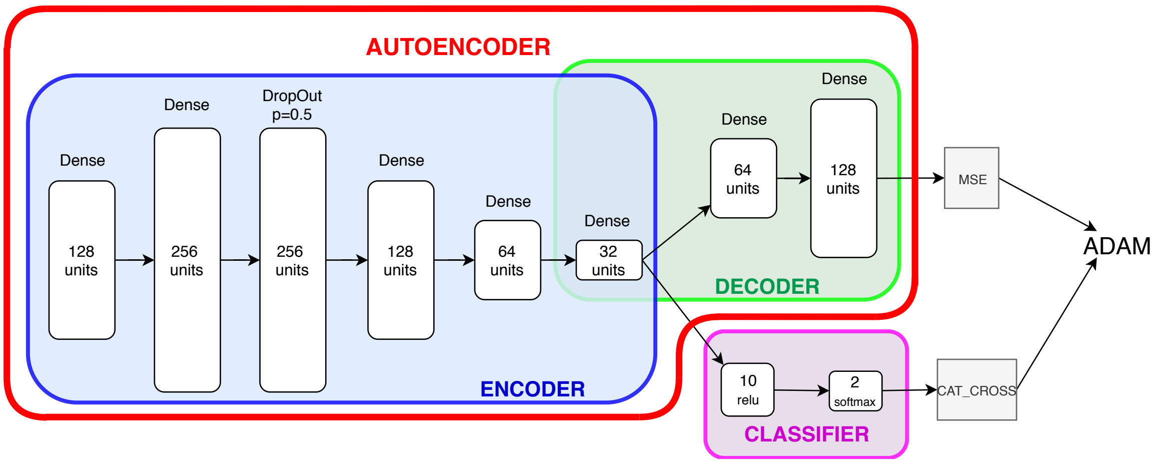


Figure 4.15: The accuracy obtained by 1-Nearest-Neighbour on dimension 128, 32 and 2, in each data set.

4.3 Supervised Autoencoder

In this section, a new architecture, called supervised Autoencoder, is presented. The Supervised Autoencoder aims to reduce the size of the input, as an Autoencoder does, taking under consideration also the belonging class of the sample in input.
The idea behind the model shown in figure 4.16 comes from the fact that a simple Autoencoder does not take any advantages from the label on the training phase. For this reason, we create a Multi-Output Deep Neural Network. Where one output is the reconstruction of the input and the second one is a classification. The model should compress an input taking into consideration its belonging class. Note that, each output has its own loss function, in particular, the Autoencoder has the Mean Square Error, while the Classifier has a Categorical Cross-Entropy loss function. To apply the back-propagation, both the losses are summed and then ADAM optimizer is applied.

Figure 4.16: Supervised Autoencoder.



Note that the loss obtained with Mean Square Error is much bigger than the loss obtained with the categorical cross-entropy. For this reason, If the Supervised Autoencoder would be trained on a generic data set, only the classifier would be optimized.
To circumvent this problem, losses have been balanced, multiplying 0.01 to the Mean Square Error before sum it to the Categorical Cross Entropy.
The accuracy obtained with 1-Nearest-Neighbour classifier, on the embedding of the data set PROTEINS in a 32 dimensions space, is 0.69 on the test set and 0.82 on the train set. On the data set PROTEINS, the supervised Autoencoder maintains the same performance obtained with a simple

Autoencoder on the test set, while on the training, the Supervised Autoencoder obtains an improving of 0.08 on the accuracy. On the DHFR data set, the supervised Autoencoder reduces the performance of 0.02 and 0.01 in the test and train set respectively. The performance previously obtained with the simple Autoencoder has been held with the Supervised one, on the AIDS data set. Interestingly, the test accuracy on Cancer increases of 0.04 and the train accuracy increases of 0.18. Finally, on the Leukemia data set, the supervised Autoencoder increases the gap between train and test accuracy, improving train performance.
Table 4.4 shows the losses obtained with the Supervised Autoencoder, reconstructing the input, on both train and test set. The table shows also the losses obtained with the simple Autoencoder on the test set, and finally, it shows the improvement obtained passing from an Autoencoder to a Supervised Autoencoder.
From the table can be seen a worsening of the performance, in particular, an increase of the losses on the data sets DHFR, Cancer, and Leukemia, while the loss has been held on the AIDS data set, and it has been decreased of 0.07 on the PROTEINS data set.

Name	**Train**		**Test**				
	Supervised Autoencoder				**Autoencoder**		**Improvement**
	Avg	**Std**	**Avg**	**Std**	**Avg**	**Std**	
PROTEINS	0.07	±0.01	0.06	±0.01	0.14	±0.06	-0.07
DHFR	0.06	±0.01	0.06	±0.01	0.05	±0.01	+0.01
AIDS	0.12	±0.03	0.12	±0.02	0.12	±0.03	0.00
Cancer	0.05	±0.00	0.05	±0.01	0.02	±0.01	+0.03
Leukemia	0.05	±0.01	0.05	±0.01	0.03	±0.01	+0.02

Table 4.4: Reconstruction error obtained by the decoder, reconstructing the input, with both Supervised Autoencoder and simple Autoencoder, in a real space of 32 dimensions.

4.3.1 Sinusoidal Callback

As previously explained, the supervised Autoencoder allows deciding the weights of the Classifier and the Autoencoder losses, before they are added and the results are considerate as the global loss to be minimized. Formally:

$$L_{global} = w_1 * L_{AUTO} + w_2 * L_{CLASS} \tag{4.2}$$

Where, L_{AUTO} is the loss obtained with the Mean Square Error between the original input and the reconstructed one, while L_{CLASS} is the loss obtained with Categorical Cross Entropy between the target class and the predicted one. The parameter w_1 and w_2 are the weights of the Autoencoder and the Classifier.
In the previous section, the parameters are set equal to 1 for the Autoencoder and 0.01 for the Classifier. However, such fixed section seams to give more relevance to the Classifier, and this could explain the worsening on the losses shown in figure 4.4.
For this reason, a specific callback has been developed. In particular, the callback changes the weights accordingly to a sinusoidal function. The idea is to increase the weights of the loss of the Autoencoder and decrease the weights of the Classifier loss, successively, reverse the process, decreasing the weights of the Autoencoder and increase the weights of the loss of the Classifier.
Sinusoidal Callback has been developed using the Keras API with TensorFlow as backend, it takes in input the following five parameters:

- *Weight_autoencoder*
- *Weight_classifier*
- *N_epochs*
- *Plateau*
- *N_periods*

Weight_decoder and *Weight_classifier* are a reference to the TensorFlow back-end, and they represent the weights of the Autoencoder and Classifier. The parameter *N_epochs* represent the number of epochs used to train the network, while the parameters *Plateau* and *N_periods* are used to build the shape of the function that is used to obtain the weight. In particular, the parameter *Plateau* specify for how many epochs the weight stays at the equal to 1, while the parameter *N_periods* specify the number of periods of the sinusoidal. Figure 4.17 shows four examples of sinusoidal produced with different parameter. In the following experiments, the sinusoidal has been generated using 4 periods

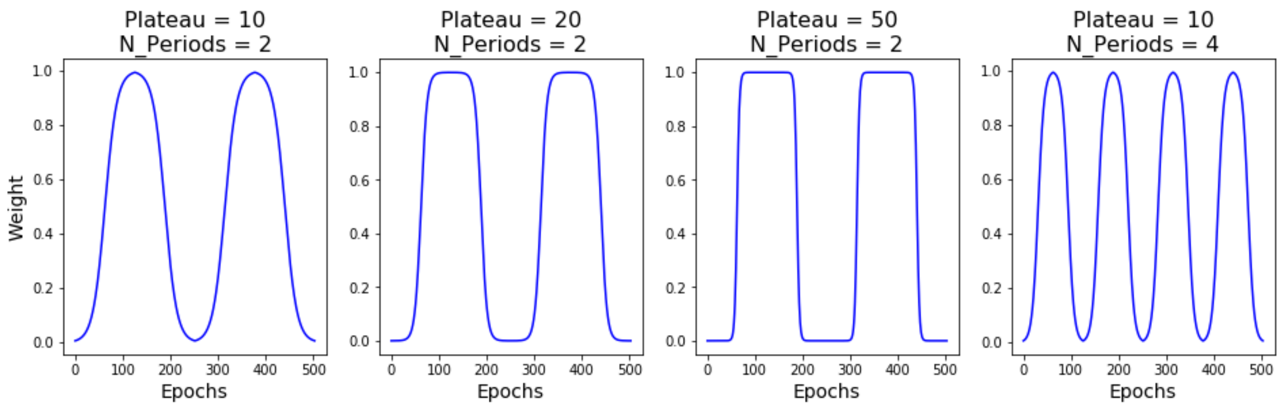


Figure 4.17: Example of sinusoidal shapes with different parameter.

and a *plateau* equal to 4. In particular, Sinusoidal callback uses two sinusoidal, one for the Classifier and one for the Autoencoder. The Idea is to deactivate on loss, decreasing its weight up to zero, and activate the other, increasing its weights up to one. Figure 4.18 shows the used weights for each epoch of both Classifier loss and Autoencoder loss. In particular, the abscissa axes shows the weights, while the epochs are shown on the ordinate axes. Red and blue points represent the used weights for the Autoencoder loss and Classifier loss, respectively.

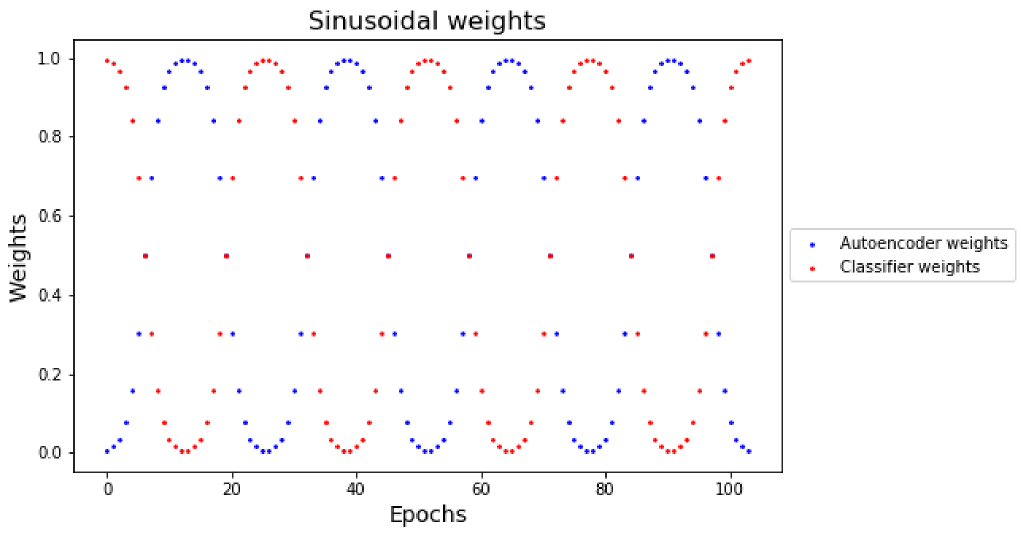


Figure 4.18: Weights used by the Autoencoder (red), and by the Classifier (blue).

The supervised Autoencoder has been executed on each data set with 100 epochs and the previously explained setting for the sinusoidal callback. In the following, the supervised Autoencoder with the sinusoidal callback is called SASin, while the supervised Autoencoder without the sinusoidal callback is called SA. On the data set PROTEINS, the SASin obtains a loss of 0.06 in both train and test set, improving the reconstruction of the input of 0.01 concerning SA, and 0.08 for the simple Autoencoder. Regarding to the accuracy obtained with 1-Nearest-Neighbour classifier on the embedded space, the embedding produced by SASin obtains an accuracy of 0.70 on the test set, and 0.82 on the train set, improving the performance on the test set of 0.01 with respect the results obtained with SA and holding the performance obtained on the embedded data set in 128 dimensions. On the data set DHFR, SASin obtains a loss of 0.04 and on both train and test set, reducing the loss of 0.01 concerning SA, while 1-Nearest-Neighbour obtains an accuracy of 0.98 on the produced embedding, in both test and train set. On the data set AIDS, SASin reduced the losses of 0.03 in both test and train set, passing from 0.12 obtained with SA to 0.09 with SASin. while, the embedded data set with SASin obtains the same accuracy of SA, in both train and test set. Unfortunately, on the data set Cancer and Leukemia, the losses are increased of 0.02 and 0.01 on both train and test set. However, the accuracy obtained with 1-Nearest-Neighbour on the embedding of the data set Cancer is increased of 0.04 on the test set, and 0.19 on the train set, while on the data set Leukemia, SASin produces an embedding that obtains 0.73 on the test set and 0.89 on the train set, 0.04 more than the results obtained with the simple Autoencoder.

Name	**Train**		**Test**				
	Sup. Auto. + Sin. Callback				**Autoencoder**		**Improvement**
	Avg	**Std**	**Avg**	**Std**	**Avg**	**Std**	
PROTEINS	0.06	±0.01	0.06	±0.01	0.14	±0.06	-0.08
DHFR	0.04	±0.00	0.04	±0.00	0.05	±0.01	-0.01
AIDS	0.09	±0.02	0.09	±0.03	0.12	±0.03	-0.03
Cancer	0.04	±0.00	0.04	±0.01	0.02	±0.01	+0.02
Leukemia	0.04	±0.01	0.04	±0.01	0.03	±0.01	+0.01

Table 4.5: Reconstruction error obtained by the decoder, reconstructing the input, with both Supervised Autoencoder plus Sinusoidal Callback and simple Autoencoder, in a real space of 32 dimensions.

Table 4.6 compares the accuracy obtained with 1-Nearest-Neighbour classifier, on the test set, for each Autoencoder presented. In particular, the embedding space has 32 dimensions. Supervised Autoencoder with sinusoidal callback achieve the best results on four data sets, moreover, such results have a small standard deviation. For this reason, the Supervised Autoencoder with sinusoidal callback is used to reduce the input space from 128 up to 32.

Name	**Autoencoder**		**Sup. Autoencoder**		**Sup. Autoencoder + sinusoidal callback**	
	Avg	**Std**	**Avg**	**Std**	**Avg**	**Std**
PROTEINS	0.69	±0.03	0.69	±0.02	**0.70**	±0.01
DHFR	0.99	±0.01	0.97	±0.00	**0.99**	±0.00
AIDS	**1.00**	±0.00	0.99	±0.00	0.99	±0.00
Cancer	0.56	±0.01	0.59	±0.01	**0.60**	±0.01
Leukemia	0.69	±0.01	0.68	±0.00	**0.72**	±0.01

Table 4.6: Accuracy of 1-Nearest-Neighbour on the test set obtained on the embedding produced by the three Autoencoders.

4.3.2 DNN-UMAP

As has been done before, DNN-UMAP is used on each data set to reduce the dimensionality from 32 up to 2. Each experiment has been repeated five times. However, the following plots show the embedding of one of the five execution, in particular, figures show the embedding such that obtained an accuracy closer to the mean.

Figure 4.19 shows the embedding of the data set PROTEINS obtained with DNN-UMAP, in particular, DNN-UMAP achieved an average accuracy of 0.71 with a standard deviation of 0.01 on the test set, while it achieved an average accuracy of 0.83 and a standard deviation of 0.01 on the train test.

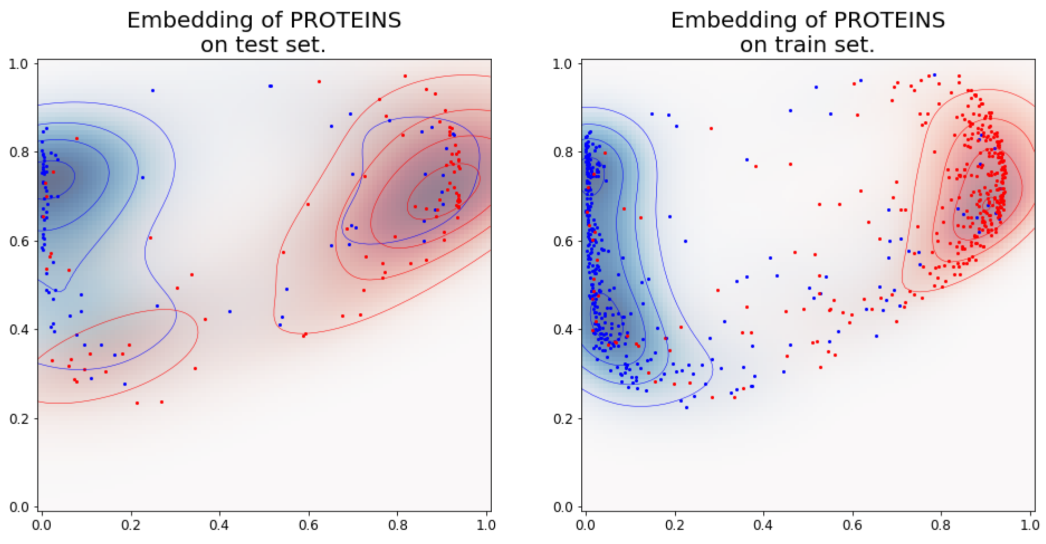


Figure 4.19: Proteins graphs embedded in a two dimensional space using Spektral + Supervised Autoencoder + DNN-UMAP

On the data set DHFR, DNN-UMAP produces an embedding that obtains 0.98 accuracy in train and test set, with a standard deviation of 0.001 in both test and train set. Such embedding is shown in figure 4.20.

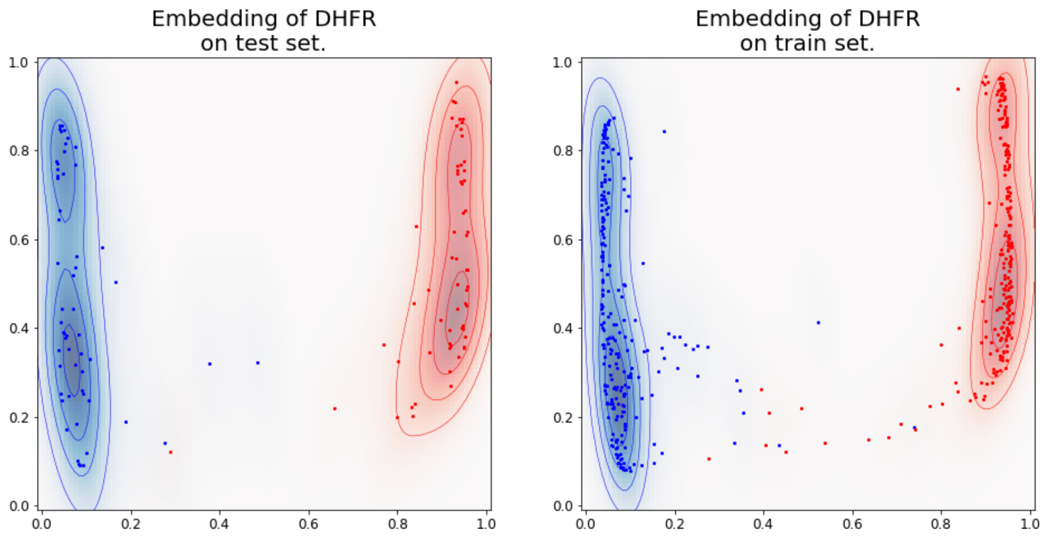


Figure 4.20: DHFR graphs embedded in a two dimensional space using Spektral + Supervised Autoencoder + DNN-UMAP

One of the five embeddings produced by the experiments, using DNN-UMAP on the AIDS data set is shown in figure 4.21. In particular, the DNN-UMAP obtains an accuracy of 0.99 with 0 standard deviation on both train and test sets.

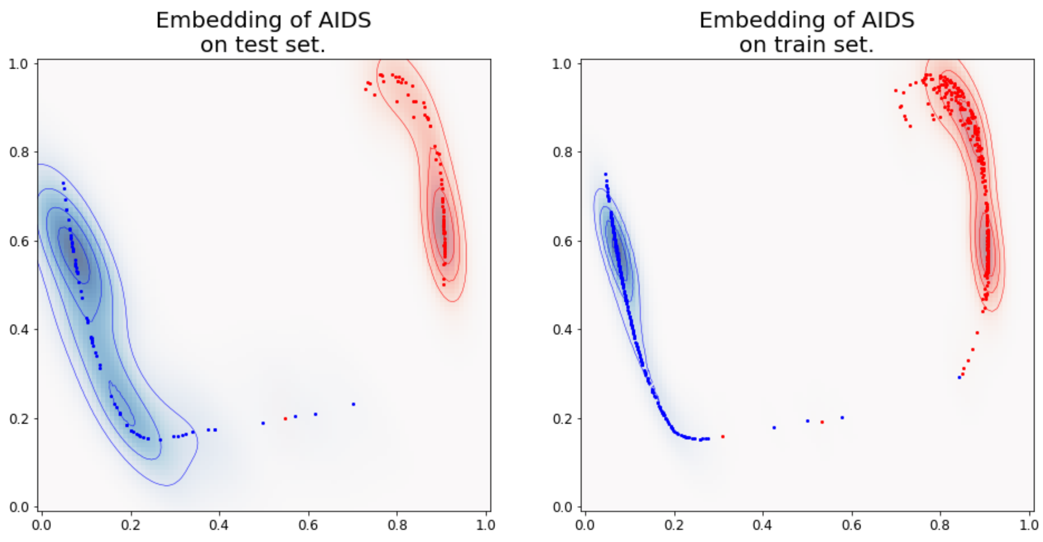


Figure 4.21: AIDS graphs embedded in a two dimensional space using Spektral + Supervised Autoencoder + DNN-UMAP

On the Cancer data set, DNN-UMAP obtains an accuracy of 0.59 with 0.01 standard deviation on the test set, while it obtains an accuracy of 0.83 with 0 standard deviation on the train set. One of the five produced embeddings is shown in figure 4.22.

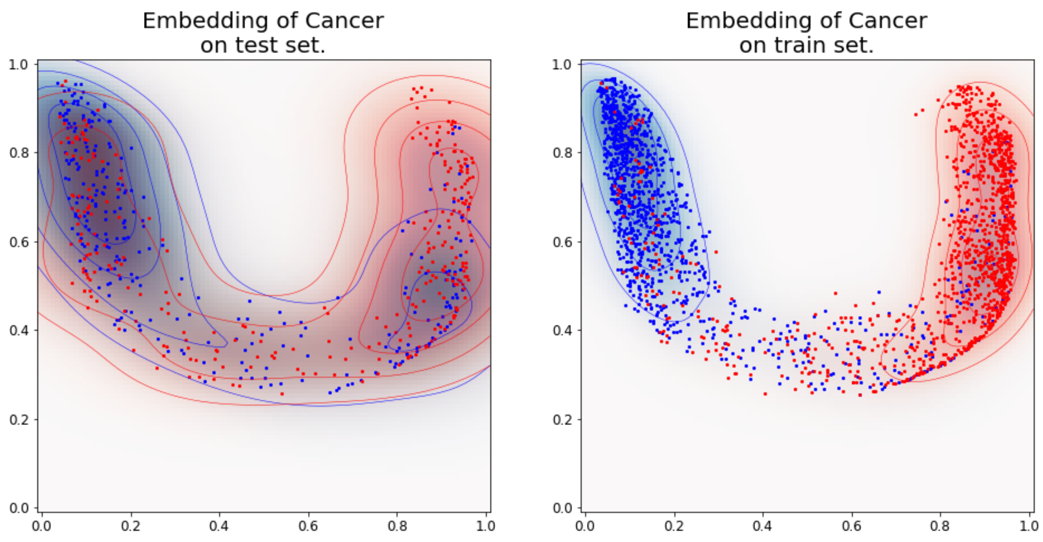


Figure 4.22: Cancer graphs embedded in a two dimensional space using Spektral + Supervised Autoencoder + DNN-UMAP

Figure 4.23 shows the results obtained on the Leukemia data set. In particular, DNN-UMAP produced an embedding that obtains an accuracy of 0.72 with 0.01 standard deviation on the test set, when evaluated with 1-Nearest-Neighbour classifier, and 0.89 with 0 standard deviation on the train set.

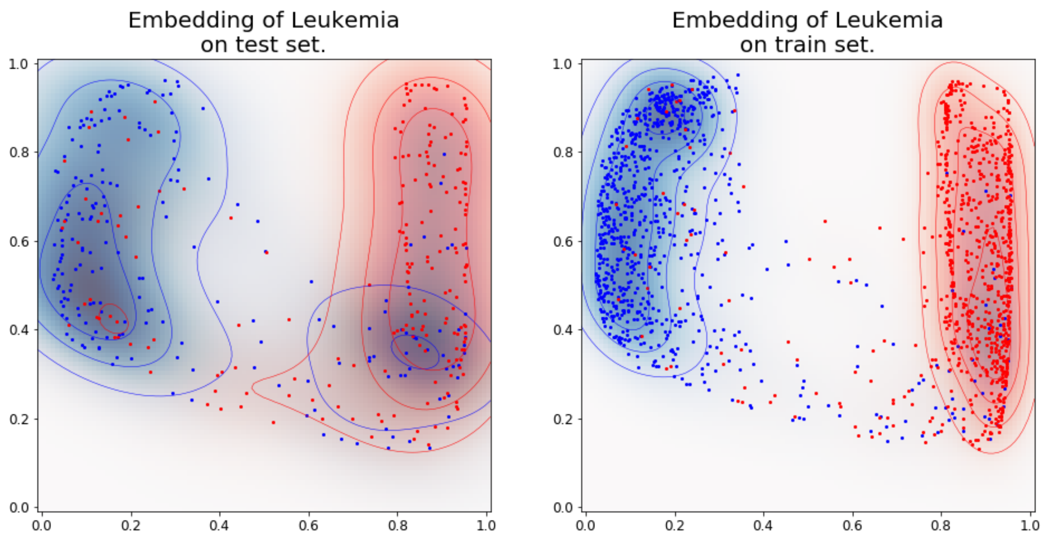


Figure 4.23: Leukemia graphs embedded in a two dimensional space using Spektral + Supervised Autoencoder + DNN-UMAP

Table 4.7 summarizes the results obtained with 1-Nearest-Neighbour classifier in the test set on the embedding produced by the technique explained so far. In particular, The second end third columns show the average accuracy and its standard deviation on the embedding produced by the second-last layer of the Deep Neural Network in a real space with 128 dimensions. The fourth and fifth columns show the average accuracy on the 32 dimension embedding produced by the simple Autoencoder explained in section 4.2, taking as input the output of the Deep Neural Network, while the sixth and seventh columns show the results obtained with DNN-UMAP in a 2 dimension embedding, taking as input the output of the simple Autoencoder. Finally, the last-four columns shows accuracy and standard deviation in a 32 dimensions space and in a 2 dimensions space, produced respectively by Supervised Autoencoder with Sinusoidal callback (SASin), where the input of SASin is the output of the Deep Neural Network, and DNN-UMAP, where the input of DNN-UMAP is the output of SASin. Interestingly, the combination Deep Neural Network plus Supervised Autoencoder with Sinusoidal callback plus DNN-UMAP produced and embedding that achieves the best results on four data set out of five. Moreover, in each data set there is an improvement of the performance passing from $\mathbf{R}^{128}$ to $\mathbf{R}^2$, in particular, on the PROTEINS data set the improvement, obtained reducing the space dimension, is 0.01, on the DHFR data set the improvement is from 0.80 to 0.99, on the AIDS the improvement is 0.01. Also on the Cancer data set the improvement is 0.01, finally, on Leukemia the accuracy obtained in a 128 dimensions real space is 0.61, while in a 2 dimensions real space is 0.72.

Name	**Spektral** $\mathbf{R}^{128}$		**Autoencoder** $\mathbf{R}^{32}$		**DNN-UMAP** $\mathbf{R}^{2}$		**SASin** $\mathbf{R}^{32}$		**DNN-UMAP** $\mathbf{R}^{2}$	
	Avg	Std	Avg	Std	Avg	Std	Avg	Std	Avg	Std
PROTEINS	0.70	±0.03	0.69	±0.03	0.70	±0.03	0.70	±0.01	**0.71**	±0.01
DHFR	0.80	±0.02	0.99	±0.01	**0.99**	±0.01	0.99	±0.00	0.98	±0.00
AIDS	0.98	±0.01	1.00	±0.00	0.99	±0.00	0.99	±0.00	**0.99**	±0.00
Cancer	0.58	±0.01	0.56	±0.01	0.59	±0.01	0.60	±0.01	**0.59**	±0.01
Leukemia	0.61	±0.02	0.69	±0.01	0.69	±0.01	0.72	±0.01	**0.72**	±0.01

Table 4.7: Accuracy of 1-Nearest-Neighbour on the embedding produced by different method on different dimension spaces.

4.4 Limitations

In this chapter, the input dimension of the embedded graphs has been reduced, maximizing the accuracy of 1-Nearest-Neighbour classifier. However, the Supervised Autoencoder allow to investigate both the accuracy and the reconstruction error, for this reason, in the following chapter, the output of the Supervised Autoencoder with Sinusoidal Callback is reduced using another Supervised Autoencoder, aiming to find a trade-off between accuracy and reconstruction error.

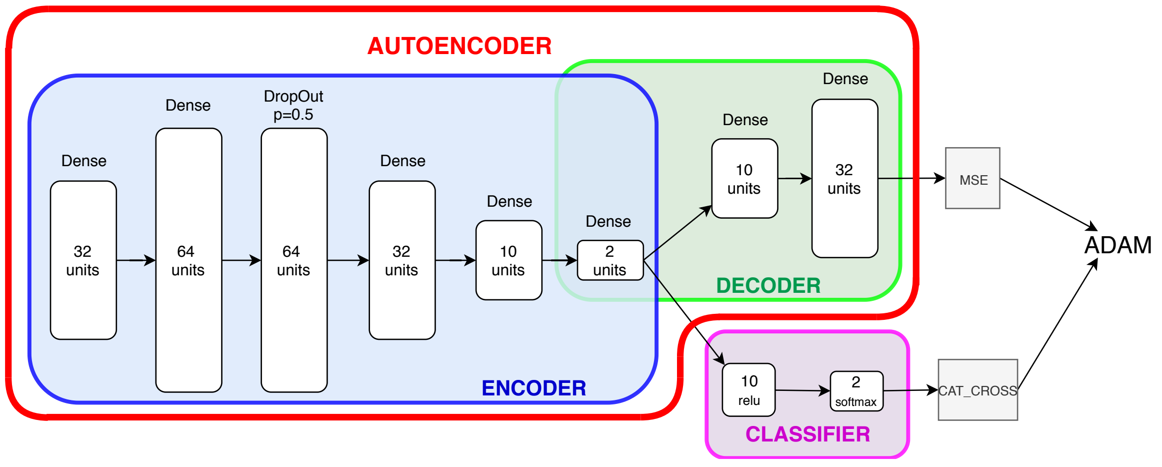
5 From High to Low Dimensionality Vectors: Quality of the Distance Preserving Embeddings

In this chapter, the aim is to increase the correlation among distances of different embeddings, that is, the correlation of the distances among graphs embedded in high dimensional space, and the distances among the graphs in a low dimensional space. Note that the thesis aims to embed a set of graphs in a low dimension space, to use a technique such as density estimation and visualization, so ideally, best results can be achieved obtaining a correlation equal to 1 between distances in high and low dimension. Firstly, the correlation is evaluated on the embedding produced by the model explained in the previous chapter, successively, part of the model is changed to improve the correlation. Then, a third model based on the Vectorizer is developed, and finally, the model is optimized to improve the correlation, creating a fourth model.

5.1 Recursive Supervised Autoencoder

The Supervised Autoencoder can be used to replace DNN-UMAP, in particular, instead of using DNN-UMAP another Supervised Autoencoder is used to reduce the dimension from 32 to 2. The second Supervised Autoencoder is called *Recursive Supervised Autoencoder*(RSA).The figure 5.1 shows the structure of the used Autoencoder. The input layer has 32 units, then there are a dense and a dropout

Figure 5.1: Recursive Supervised Autoencoder.



layer with 64 hidden units, successively, the dimension of the layers decreases to 2, passing through 32,10 and 2 hidden units. The decoder rebuilds the input from 2 hidden units to 32 hidden units, passing through a dense layer of 10 units, while the classifier increases the size up to 10 and finally the output layer has 2 units, that are equal to the number of class. The data set is divided into test and train set, respectively, 80% and 20% of the whole data set. The network is trained with 500 epochs and a batch size of 32. The Sinusoidal Callback explained in subsection 4.3.1, is used with a strong weight on the decoder.

5.2 Measurement

In order to evaluate the performance of the model, Spearman correlation is used, in particular, Kokoska et al. [11] define the Spearman correlation as follow.

$$\rho_s = 1 - \frac{6 \sum d_i^2}{n(n^2 - 1)} \tag{5.1}$$

Where d_i is the difference between the ranks of corresponding variables and N is the number of observations. Spearman correlation does not assume that data are normally distributed.
The correlation is computed among distances in different embedding space, with distances in a high space. Such high space is a real space with 4096 dimensions and the embedding is produced using Vectorizer with 12 bits and the parameter complexity is fixed equal to 2.
The input data sets are binary, and most of them represent molecules, usually molecules that are positive against a disease share some properties, while molecules that are not effective against diseases are characterized by different properties.
For example, suppose to visualize a 2D embedding of molecules, then the positive samples tend to stay in a unique portion of the space, creating a unique island of points, while negative samples may create many islands of points, and each island should embed the properties of the molecules. For this reason, the correlation among distances of positive samples will be shown.

5.3 Embedding models

This section explains three models and their parameters, in particular, two models are based on Spektral and one is based on Vectorizer.

5.3.1 Spektral based

The first model is called SPK128-SA32-DNN-UMAP2, where the embedding is done by the second-last layer of Spektral, where each node of the input graph is modified, adding the node embedding as node attributes (as explained in section 3.4), each graph is embedded in a real vector space of 128 dimensions, successively the dimension of the vector is reduced up to 32 using a Supervised Autoencoder, where the weight of the decoder is equal to 0.1 while the weight of the classifier is equal to 0.0001. Finally, to reduce the dimension from 32 to 2, DNN-UMAP is used.
The second model is still based on Spektral and it is similar to the previous one, the only differences relay on the last part of the model, in fact, this model instead of use the DNN-UMAP it uses another Supervised Autoencoder with a much lower weights on the classifier, in particular it is equal to 0.0000001. Such a model is called SPK128-SA32-SA2.

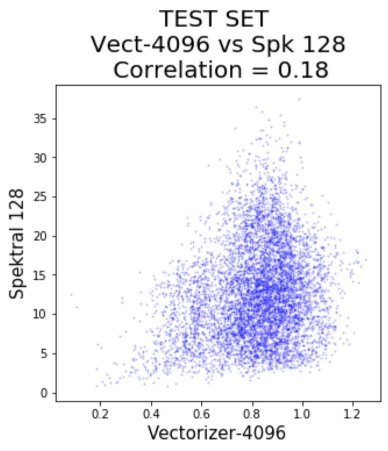
5.3.2 Vectorizer based

The third model, called VECT4096-PCA128-SA32-SA2, is based on Vectorizer, in fact, the embedding is done through the Vectorizer with nbits equal to 12 and the complexity parameter is fixed to 2, thus the graph embedding lies in a real vector space of 4096 dimensions. Successively, the reduction up to 128 is done by PCA, then from 128 to 32 with a Supervised Autoencoder and from 32 up to 2 with another Supervised Autoencoder. Interestingly, the reduction made by PCA preserves the correlation concerning the high dimension real space, in fact later such dimensionality will be analyzed in more details.

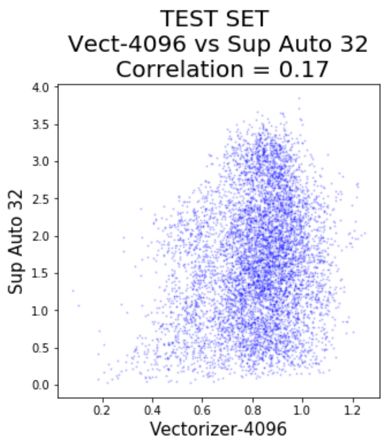
5.4 Results

Here the results obtained with the previous three models are shown. In particular, the results are shown only on the data set DHFR because on the other data set, the results have similar behaviour.

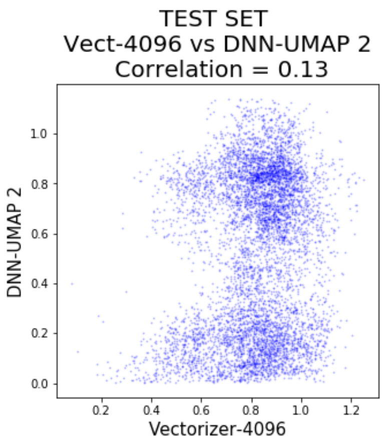
Model 1: SPK128-SA32-DNN-UMAP2 The first model tested is the same used in the previous chapter where the aim was to maximise the accuracy. Each plot reported on figure 5.2 shows on the abscissas axis the distances between points in the embedding produced by the Vectorizer, while on the ordinate axis there are distances computed in different spaces. From the figure can be seen that the correlation is low in all dimensions and also the correlation in among positive sample is still low. Table 5.1 report the correlation obtained in each dimension for both all samples and only positive samples. In particular, the first row shows the correlation on the test set among all samples, starting from a correlation of 0.202 concerning the embedding in $\mathbf{R}^{128}$ up to a correlation of 0.13 concerning the embedding in $\mathbf{R}^2$. Thus the reduction of 126 dimensions lies in a decreasing of the correlation of 0.072 points. On the other hand, the second row shows the correlations between distances of positive samples, and it starts from 0.339 in $\mathbf{R}^{128}$ up to 0.319 in $\mathbf{R}^2$, here the loss is equal to 0.02.



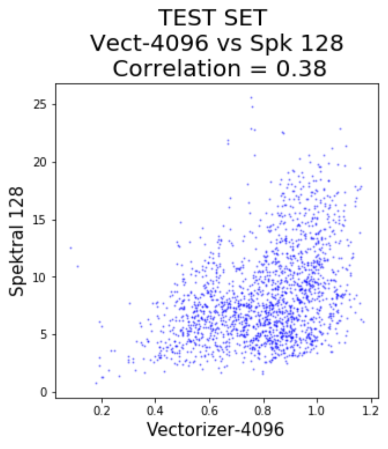
(a)



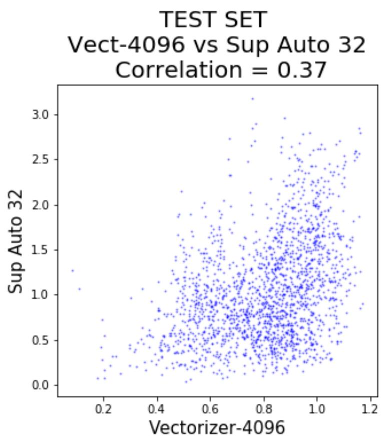
(b)



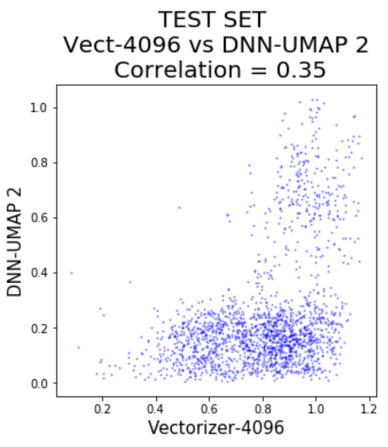
(c)



(d)



(e)



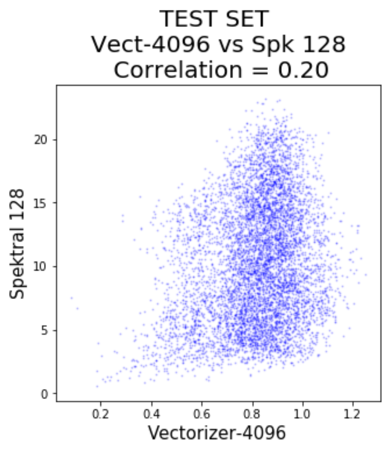
(f)

Figure 5.2: On the abscissas axis, there are distances computed on the real space with 4096 dimensions, while on the ordinate axis there are distances computed in different spaces. The first row shows the correlation among embedding of graphs, positive and negative samples on the test set, while the second row shows the correlation among embedding of positive samples. In Particular, figure 5.2a and 5.2d shows the correlation with the embedding in 128 dimensions produced by the second-last layer of Spektral, figure 5.2b and 5.2e shows the correlation with the embedding in 32 dimensions, and finally figure 5.2c and 5.2f shows the correlation with the embedding in 2 dimensions.

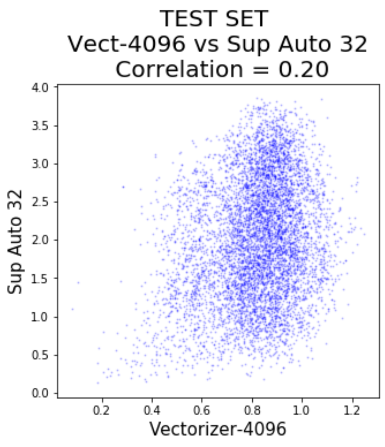
	Spektral $\mathbf{R}^{128}$	Super. Auto. $\mathbf{R}^{32}$	DNN-UMAP $\mathbf{R}^{2}$
All samples	0.18	0.17	0.13
Positive samples	0.38	0.37	0.35

Table 5.1: Correlation of distances on the data set DHFR in different embedding spaces.

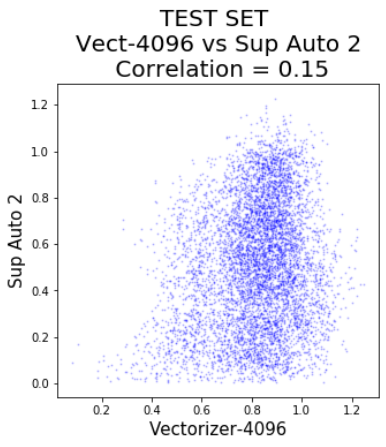
Model 2: SPK128-SA32-SA2 The second model, not only replaces the DNN-UMAP with another supervised Autoencoder, but it also has a much stronger weight on the decoder. Again, the results are showed through figure 5.3 and table 5.2. In particular, from table 5.2 can be seen that a stronger weight on the decoder lies in an improvement of the correlation in low dimension, in fact, the lose of correlation from $\mathbf{R}^{128}$ to $\mathbf{R}^2$ is equal to 0.042 on all samples, and 0.01 on the positive samples.



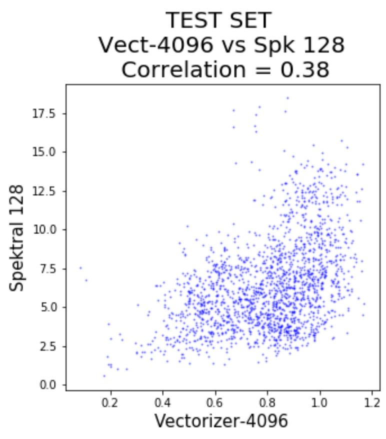
(a)



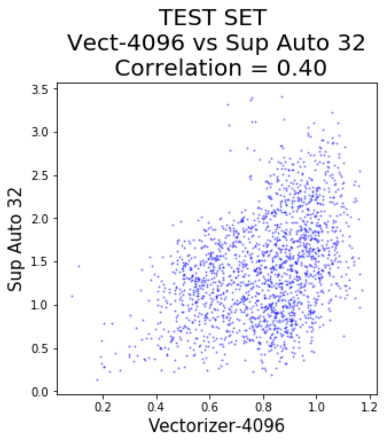
(b)



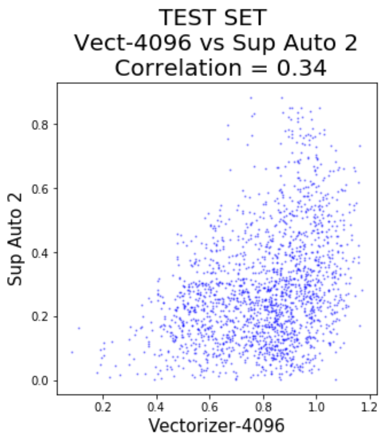
(c)



(d)



(e)



(f)

Figure 5.3: On the abscissas axis, there are distances computed on the real space with 4096 dimensions, while on the ordinate axis there are distances computed in different spaces. The first row shows the correlation among embedding of graphs, positive and negative samples on the test set, while the second row shows the correlation among embedding of positive samples. In Particular, figure 5.3a and 5.3d shows the correlation with the embedding in 128 dimensions, figure 5.3b and 5.3e shows the correlation with the embedding in 32 dimensions, and finally figure 5.3c and 5.3f shows the correlation with the embedding in 2 dimensions produced by the Supervised Autoencoder.

	Spektral $\mathbf{R}^{128}$	Super. Auto. $\mathbf{R}^{32}$	Super. Auto. $\mathbf{R}^{2}$
All samples	0.20	0.20	0.15
Positive samples	0.38	0.40	0.34

Table 5.2: Correlation of distances on the data set DHFR in different embedding spaces.

Looking at the results obtained with Model 1 and Model 2, the biggest loss of correlation is given by Spektral, because the Supervised Autoencoder seems to be able to maintain the correlations, reducing the dimensionality of the embedding. For this reason, model 3 uses the Vectorizer instead of Spektral to compute the embedding of the graphs.

Model 3: VECT4096-PCA128-SA32-SA2 The third model uses the Vectorizer to embed the graphs in a high dimensional space, then PCA to reduce the dimension up to 128 and finally it uses the same Supervised Autoencoders as Model2 does. Interestingly, PCA maintains a high correlation among distances passing from 4096 dimensions to 128 dimensions. In particular, from table 5.3 can be seen that the correlation of distances in $\mathbf{R}^{128}$ is high in both all samples and positive samples. The correlation in 2D decreases significantly up to 0.506 on all samples, and up to 0.603 on positive samples.

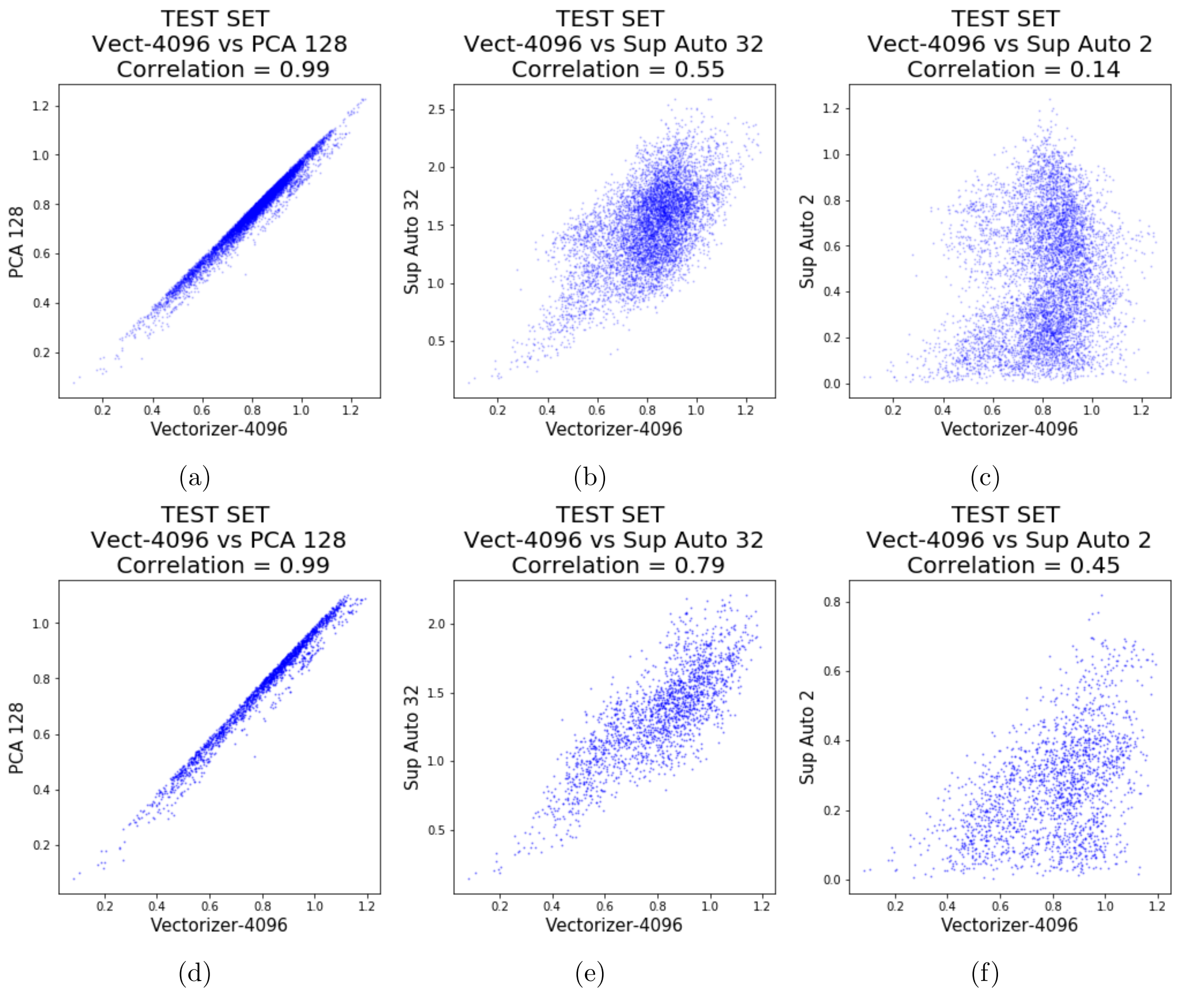


Figure 5.4: On the abscissas axis, there are distances computed on the real space with 4096 dimensions, while on the ordinate axis there are distances computed in different spaces. The first row shows the correlation among embedding of graphs, positive and negative samples on the test set, while the second row shows the correlation among embedding of positive samples. In Particular, figure 5.4a and 5.4d shows the correlation with the embedding in 128 dimensions produced by Vectorizer and PCA, figure 5.4b and 5.4e shows the correlation with the embedding in 32 dimensions, and finally figure 5.4c and 5.4f shows the correlation with the embedding in 2 dimensions produced by the Supervised Autoencoder.

	PCA $\mathbf{R}^{128}$	Super. Auto. $\mathbf{R}^{32}$	Super. Auto $\mathbf{R}^{2}$
All samples	0.99	0.55	0.14
Positive samples	0.99	0.79	0.45

Table 5.3: Correlation of distances on the data set DHFR in different embedding spaces.

Table 5.4 compares the results obtained so far, and it is easy to see that the best results are obtained using the third model. In particular, the model based on the Vectorizer obtains approximately 0.35 more correlation in 2D on all samples and 0.3 on the positive samples. Such pore results in models based on Spektral are because Spektral itself lose all the information concerning distances, while the good results obtained with the model based on Vectorizer bring as investigate in more details the middle dimensions of the embeddings.

	All samples			Positive samples		
	$\mathbf{R}^{128}$	$\mathbf{R}^{32}$	$\mathbf{R}^{2}$	$\mathbf{R}^{128}$	$\mathbf{R}^{32}$	$\mathbf{R}^{2}$
Model 1	0.18	0.17	0.13	0.38	0.37	0.35
Model 2	0.20	0.20	**0.15**	0.38	0.40	0.34
Model 3	**0.99**	**0.55**	0.14	**0.99**	**0.79**	**0.45**

Table 5.4: Correlation among distances in real space of 4096 dimensions with distances on real spaces of lower dimensions, calculated in both all samples, and only positive samples. The first two models are based on Spektral, in particular, model 1 is Spektral, Supervised Autoencoder and DNN-UMAP, while, model 2 is Spektral and two Supervised Autoencoders. Model 3 is composed by the Vectorizer, PCA and two Supervised Autoencoders.

5.5 Improving the Vectorizer based model

In this section, the embedding dimensions of the model VECT4096-PCA128-SA32-SA2 are analyzed in more details. In particular, each middle dimension of the model is investigated to improve the final correlation between distances in a 4096-dimensional real space and a 2-dimensional real space.

5.5.1 High dimension analysis

QUESTION: *Given in input a vector of 4096 elements, what is a suitable dimension that allows PCA to maintain a good correlation among distances in the space of 4096 dimensions and the target space?*

From the results shown in table 5.4 it is easy to see that PCA reduces the dimensionality from 4096 to 128 maintaining an high correlation among distances, for this reason here a more detailed analysis of the target dimension is done. In particular, PCA is used to reduce the dimension from 4096 to the following dimensions: $[10, 20, 30, 35, 40, 45, 50, 55, 60, 100, 128]$. Then, for each reduction, the correlation of the distances and the accuracy of 1-Nearest-Neighbour classifier are computed, in particular, the classifier is computed in 10-fold cross validation, in order to have a standard deviation. Such results are shown in figure 5.5, where the plot in the left shows the correlation on the abscissa axis and the embedding dimension on the ordinate. The plot shows both test and train set in blue and red respectively. The plot to the right of the figure 5.5 shows the accuracy obtained in by 1-Nearest-Neighbour classifier when the embedding size increases.

The plot to the left of the figure 5.5 shows that the correlation increases rapidly when the size of the embedding is small, for this reason, the target dimension after PCA is equal to 35. On the other hand, the accuracy on the test set has a larger variance concerning the test. However, the choice of an embedding dimension of 35 does not affect too much the accuracy obtained in an embedding of 128 dimensions.

In particular, the reduced dimension of PCA obtains an accuracy of 0.87 with 0.11 of standard deviation on the test set, and an accuracy of 0.95 with 0.03 of standard deviation on the train test. The correlation between the distances in $\mathbf{R}^{4096}$ and $\mathbf{R}^{35}$ is equal to 0.97 on the test and train set.

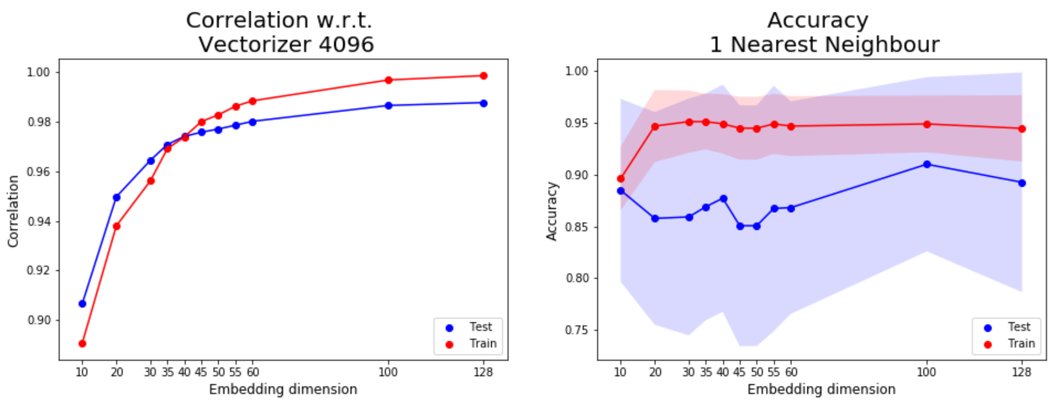


Figure 5.5: The correlation of distances and 1-Nearest-Neighbour accuracy with the increasing size of the embedding produced by PCA.

ANSWER: *PCA can reduce up to 35 dimensions the input, without a drastically decreasing in the correlation among distances.*

5.5.2 Medium dimension analysis

QUESTION: *What is the best medium dimension produced by the Supervised Autoencoder, such that maximise the correlation among distances in a 4096-dimensional space and distances in the target space?*

In this subsection, the Supervised Autoencoder is used to reduce the dimension of the embedded graphs from space in $\mathbf{R}^{35}$ up to one of the following dimensions $[5, 7, 10, 15, 20, 25]$. In particular, figure 5.6 shows the behaviour of the correlation (on the left) and ones of the 1-Nearest-Neighbour classifier (on the right). The accuracy has a trend that reflects the intuition when the embedding dimension increases the obtained accuracy increases as well. On the other hand, the correlation seems to be stable up to 20 and then it drastically decreases, such behaviour may be due to the architecture of the network, in particular from the bottleneck (the interjection of the encoder and the decoder) is greater than the second-last layer of the decoder, it means that the decoder decrease the size of the input while it is trying to reconstruct it. However, the Supervised Autoencoder reduces the input from a 35 dimensional vector up to a 5 dimensional vector, with an accuracy equal to 0.75 and a standard deviation equal to 0.12 on the test set, and 0.76 with standard deviation equal to 0.01 on the train set, with a loss of 0.12 and 0.19 with respect to $\mathbf{R}^{35}$ in train and test set, respectively. The correlation of distances in $\mathbf{R}^{4096}$ and distances in $\mathbf{R}^{5}$ is equal to 0.71 in both train and test set.

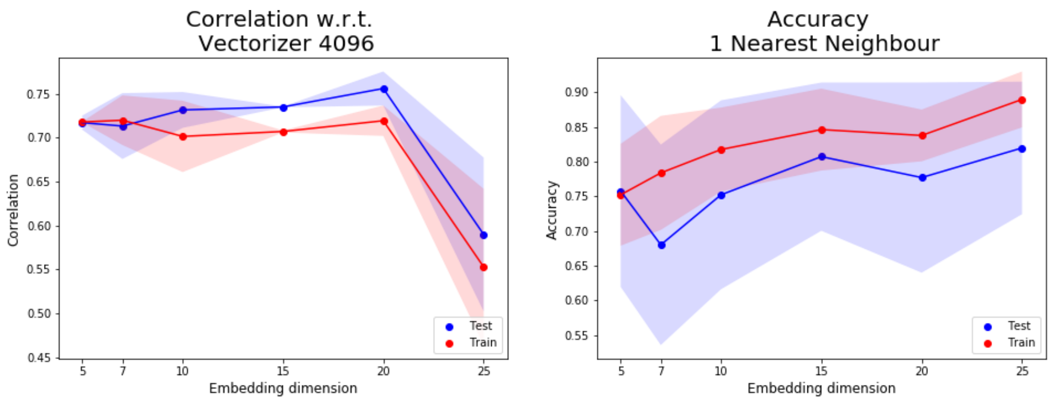


Figure 5.6: The correlation of distances and 1-Nearest-Neighbour accuracy with the increasing size of the embedding produced by the Supervised Autoencoder.

ANSWER: *The Supervised Autoencoder can reduce up to 5 dimensions the input vector, decreasing the correlation of 0.24 points concerning the embedding in a real space with 35 dimensions.*

5.5.3 Low dimension analysis

QUESTION: *What are the values of the Sinusoidal Callback's parameters that maximise the correlation among distances in a 4096-dimensional space and distances a 2D space?*

To reduce the dimensions form $\mathbf{R}^5$ to $\mathbf{R}^2$ the structure of the Supervised Autoencoder has to be changed. In particular, the second-last layer of the classifier and the decoder (shown on figure 5.1) is reduced from 10 units up to 5 units. Moreover, the network has been trained whit 700 epochs instead of 500. Finally, the parameters *N_periods* of the Sinusoidal Callback (explained in section 4.3.1) is increased from 4 to 6.
Successively, the parameters $Weight\_autoencoder$ and $Weight\_classifier$ of the Sinusoidal Callback have been analyzed in more details, changing their values and monitoring decoder loss, 1-Nearest-Neighbour accuracy and Spearman correlation among distances in $\mathbf{R}^{4096}$ and distances in $\mathbf{R}^2$. The plot to the right of the figure 5.7 shows on the abscissa axis the $1-Accuracy$, and on the ordinate axis the decoder loss. The plot in the middle of the figure shows on the abscissa axis the $1 - Accuracy$, while on the ordinate axis the $1 - Correlation$, and finally, the plot to the right shows on the abscissa axis the decoder loss and on the ordinate axis the $1-Correlation$. Each point in the figure identifies a specific configuration of the weights for the Autoencoder and of the classifier. In particular, two weights are tested for the Autoencoder, that are 1 and 0.1, while for the classifier four values are tested, that are 0.1, 0.001, 1e$-$7 and 1e$-$12, for a total of 8 configurations, each configuration is tested five times, and the reported result is the average. The blue dashed line shows the so-called Pareto Front, that is a set of configurations that minimizes both the objective functions on the abscissa and ordinate. The Pareto Fronts identify three sets of points, one for each multi-objective optimization problem, and interestingly, only one configuration lies in all the sets. Such configuration is *W.Dec:0.1 W.Cla:0.01*, that is a weight equal to 0.1 for the Autoencoder and a weight equal to 0.01 for the classifier.
The found configuration of the Supervise Autoencoder produce an embedding in $\mathbf{R}^2$ that obtains an accuracy of 0.59 on the test set, and a Spearman correlation of 0.59 among distances in $\mathbf{R}^{4096}$ and $\mathbf{R}^2$, that is 0.084 more with respect the best algorithm found so far.

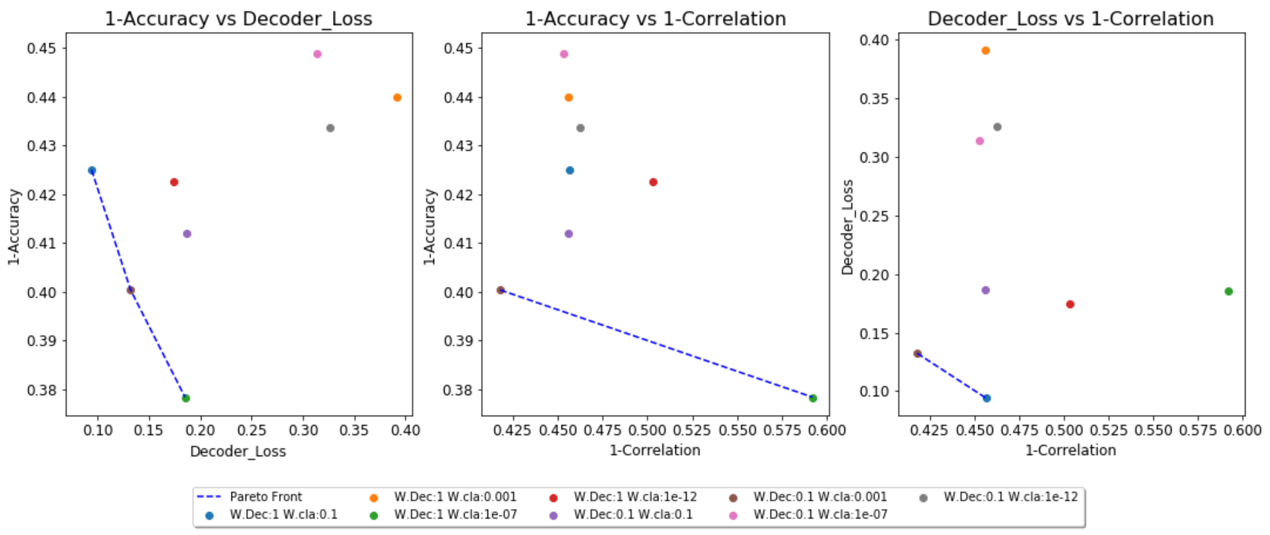


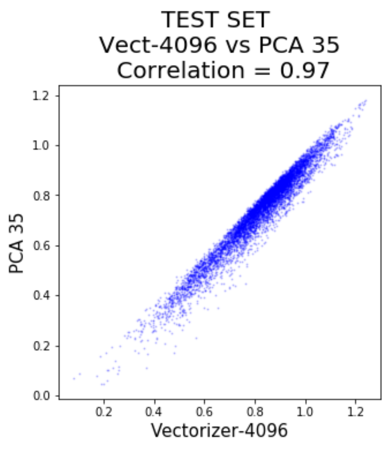
Figure 5.7: Pareto optimality of 1-Nearest-Neighbour classifier, Decoder loss and correlation among distances in $\mathbf{R}^{4096}$ and $\mathbf{R}^2$

ANSWER: *The best results are achieved using Weight_Autoencoder equal to 0.1 and Weight_Classifier equal to 0.01.*

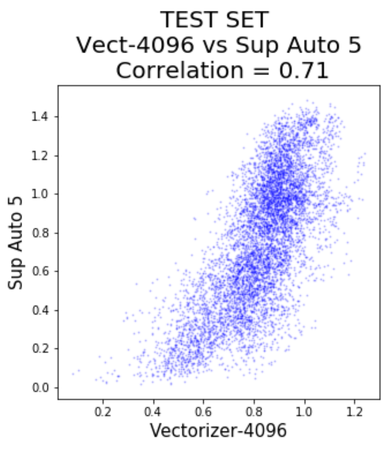
After the previous analysis, a fourth model can be defined as follow:

- Embed graphs using the Vectorizer with *n_bits* equal to 12.
- Reduce the dimensions of the space up to a 35-dimensional space, using PCA.
- Use the Supervised Autoencoder explained in section 5.5.2 to reduce the dimension up to 5.
- Use the Supervised Autoencoder explained in section 5.5.3 to reduce the dimension up to 2.

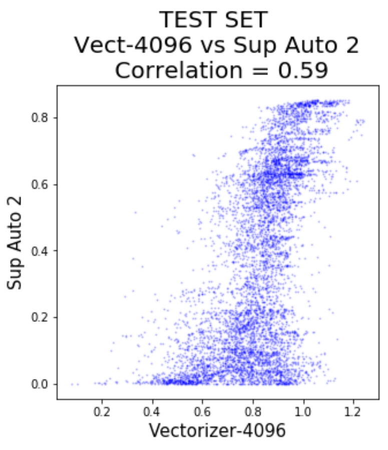
As previously, figure 5.8 shows the correlation for each embedding dimension in both cases: all samples and only positive samples.



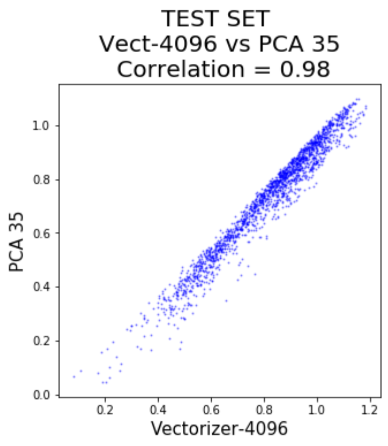
(a)



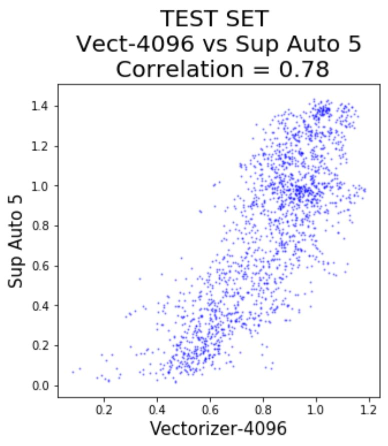
(b)



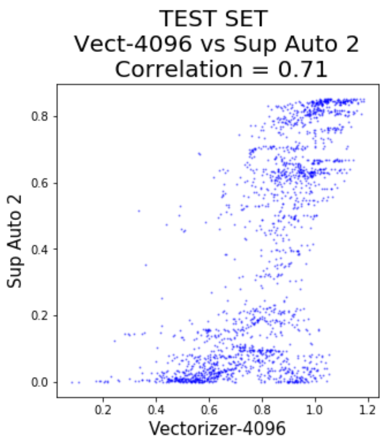
(c)



(d)



(e)



(f)

Figure 5.8: On the abscissas axis, there are distances computed on the real space with 4096 dimensions, while on the ordinate axis there are distances computed in different spaces. The first row shows the correlation among embedding of graphs, positive and negative samples on the test set, while the second row shows the correlation among embedding of positive samples. In Particular, figure 5.8a and 5.8d shows the correlation with the embedding in 35 dimensions produced by Vectorizer and PCA, figure 5.8b and 5.8e shows the correlation with the embedding in 5 dimensions, and finally figure 5.8c and 5.8f shows the correlation with the embedding in 2 dimensions produced by the Supervised Autoencoder.

Table 5.5 shows the correlations obtained with the fourth model. And it is easy to see that the biggest loss of correlation is due to the Supervised Autoencoder that reduces the dimension from 35 to 5.

Recalling that the aim of this chapter is to maximize the correlation among distances in different spaces, it is important to mentions that even if model 4 achieve a good correlations among distances in $\mathbf{R}^{4096}$ and $\mathbf{R}^2$ with respect to model 1. Model 4 achieve an accuracy of 1-Nearest-Neighbour equal to 0.59, while, model 1 obtains an accuracy of 0.98.

	PCA $\mathbf{R}^{35}$	Super. Auto. $\mathbf{R}^{5}$	Super. Auto. $\mathbf{R}^{2}$
All samples	0.97	0.71	0.59
Positive samples	0.98	0.78	0.71

Table 5.5: Correlation of distances on the data set DHFR in different embedding spaces.

To conclude this chapter figure 5.9 show the correlation of distances in $\mathbf{R}^{4096}$ and each middle dimension up to 2, for each model. Each colour represents a model, the continuous lines show the correlation among all samples, while the dashed lines show the correlation among positive samples. It is easy to see that models based on Spektral (model 1 and 2) obtain poor results since the beginning (128 dimensions). On the other hand, Vectorizer based models achieve better results, in particular, the fourth model is the one that obtains better results.

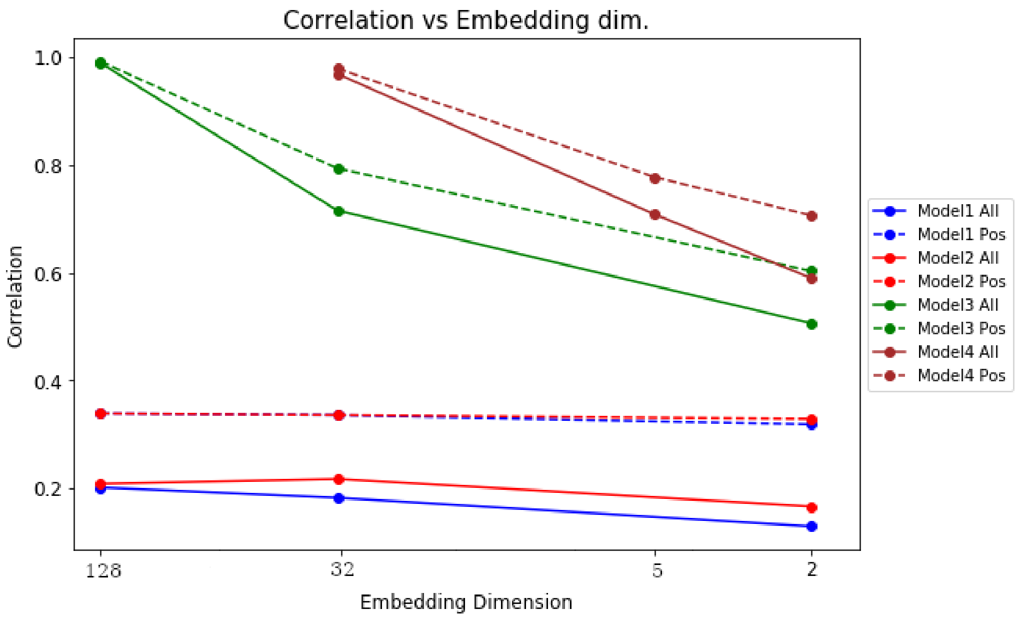


Figure 5.9: Correlation obtained by each model when the size of the embedding decrease.

6 Conclusions and Future Work

This dissertation proposes several approaches to embed graphs in low dimension, with two conflicting objectives. Firstly, the embedding has been done maximizing the accuracy of 1-Nearest-Neighbour classifier. Secondly, the correlation among distances of graphs in high and low dimension spaces has been maximized.

As shown in table 6.1, the model that obtains the better accuracy embeds graphs in a 128-dimensional space using the second-last layer of Spektral. In this solution, each node has been manipulated by adding the embedding of the node itself as an attribute. Then, the dimension is reduced to 32 using a Supervised Autoencoder with a Sinusoidal Callback. Finally, the reduction from 32 dimensions to 2 is obtained using UMAP.

	Spektral plus Vectorizer $\mathbf{R}^{128}$	Supervised Autoencoder $\mathbf{R}^{32}$	DNN-UMAP $\mathbf{R}^{2}$
PROTEINS	0.70	0.70	0.71
DHFR	1	0.99	0.98
AIDS	1	0.99	0.99
Cancer	0.59	0.60	0.59
Leukemia	0.69	0.72	0.72

Table 6.1: Accuracy of 1-Nearest-Neighbour classifier in different embedding spaces.

Concerning the model that maximizes the correlation among distances, it embeds the graphs in a 4096-dimensional space using the Vectorizer. The dimension is then reduced to 35 using PCA. Finally, the dimension is reduced to 5 and then to 2, using two cascaded Supervised Autoencoders. The following table shows the results obtained on the data set DHFR.

	PCA $\mathbf{R}^{35}$	Supervised Autoencoder $\mathbf{R}^{5}$	Supervised Autoencoder $\mathbf{R}^{2}$
DHFR	0.97	0.71	0.59

Table 6.2: Spearman correlation among distances computed in a space with 4096 dimensions and spaces with lower dimensionality.

In conclusion, the first model discussed achieve good results in performing the classification task, while the later obtains significant outcomes in terms of correlation among distances in high and low dimension spaces.

Regarding sequel developments, the Sinusoidal Callback used by the Supervised Autoencoder has several parameters that are fixed beforehand. An interesting improvement is about learning such parameters during the training phase of the Network.
Moreover, In this work, most of the input data set represent molecules. The models can be tested using other data set and also using other graph embedders in high dimensions.
Furthermore, In the second chapter, the combination of the Vectorizer and Spektral lies to an evident improvement in the classification task. This combination may be analyzed in more details, investigating several Graph Neural Network and several node embedders.

Bibliography

[1] Hongyun Cai, Vincent W Zheng, and Kevin Chen-Chuan Chang. A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE Transactions on Knowledge and Data Engineering*, 30(9):1616–1637, 2018.

[2] Fabrizio Costa and Kurt De Grave. Fast neighborhood subgraph pairwise distance kernel. In *Proceedings of the 26th International Conference on Machine Learning*, pages 255–262. Omnipress; Madison, WI, USA, 2010.

[3] Paul D Dobson and Andrew J Doig. Distinguishing enzyme structures from non-enzymes without alignments. *Journal of molecular biology*, 330(4):771–783, 2003.

[4] Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11(Feb):625–660, 2010.

[5] Daniele Grattarola Filippo Maria Bianchi and Cesare Alippi. Mincut pooling in graph neural networks. 2019.

[6] Xue Geng, Hanwang Zhang, Jingwen Bian, and Tat-Seng Chua. Learning image and user features for recommendation in social networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4274–4282, 2015.

[7] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning.* MIT press, 2016.

[8] Marco Gori, Gabriele Monfardini, and Franco Scarselli. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, pages 729–734. IEEE, 2005.

[9] David Haussler. Convolution kernels on discrete structures. Technical report, Technical report, Department of Computer Science, University of California . . . , 1999.

[10] Kristian Kersting, Nils M. Kriege, Christopher Morris, Petra Mutzel, and Marion Neumann. Benchmark data sets for graph kernels, 2016.

[11] Stephen Kokoska and Daniel Zwillinger. *CRC standard probability and statistics tables and formulae.* Crc Press, 2000.

[12] Danai Koutra, Ankur Parikh, Aaditya Ramdas, and Jing Xiang. Algorithms for graph similarity and subgraph matching. In *Proc. Ecol. Inference Conf*, volume 17, 2011.

[13] Nils M Kriege, Fredrik D Johansson, and Christopher Morris. A survey on graph kernels. *arXiv preprint arXiv:1903.11835*, 2019.

[14] Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018.

[15] Alessio Micheli. Neural network for graphs: A contextual constructive approach. *IEEE Transactions on Neural Networks*, 20(3):498–511, 2009.

[16] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[17] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[18] Giannis Nikolentzos, Polykarpos Meladianos, and Michalis Vazirgiannis. Matching node embeddings for graph similarity. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[19] Kaspar Riesen and Horst Bunke. Iam graph database repository for graph based pattern recognition and machine learning. In *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*, pages 287–297. Springer, 2008.

[20] Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas. The earth mover's distance as a metric for image retrieval. *International journal of computer vision*, 40(2):99–121, 2000.

[21] Nino Shervashidze, SVN Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten Borgwardt. Efficient graphlet kernels for large graph comparison. In *Artificial Intelligence and Statistics*, pages 488–495, 2009.

[22] Alessandro Sperduti and Antonina Starita. Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks*, 8(3):714–735, 1997.

[23] Jeffrey J Sutherland, Lee A O'brien, and Donald F Weaver. Spline-fitting with a genetic algorithm: A method for developing classification structure- activity relationships. *Journal of chemical information and computer sciences*, 43(6):1906–1915, 2003.

[24] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th international conference on world wide web*, pages 1067–1077. International World Wide Web Conferences Steering Committee, 2015.

[25] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[26] Daniele Zambon, Lorenzo Livi, and Cesare Alippi. Detecting changes in sequences of attributed graphs. In *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–7. IEEE, 2017.

[27] Xiaohan Zhao, Adelbert Chang, Atish Das Sarma, Haitao Zheng, and Ben Y Zhao. On the embeddability of random walk distances. *Proceedings of the VLDB Endowment*, 6(14):1690–1701, 2013.

[28] Yu Zhao, Zhiyuan Liu, and Maosong Sun. Representation learning for measuring entity relatedness with rich information. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.