ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ
ΤΜΗΜΑ ΦΥΣΙΚΗΣ

Διπλωματική Εργασία
του φοιτητή του Τμήματος Φυσικής του
Πανεπιστημίου Κρήτης

Γεωργούλη Ηλία
Αριθμός Μητρώου: 4410

ΘΕΜΑ

Bio-embeddings construction from microarray gene expression datasets using deep neural networks

Κατασκευή βιο-αναπαραστάσεων από σύνολα δεδομένων μικροσυστοιχιών γονιδιακών εκφράσεων με χρήση βαθιών νευρωνικών δικτύων

Επιβλέπων

Ιωάννης Πανταζής

Αριθμός Διπλωματικής Εργασίας:

Ηράκλειο, Ιούνιος 2022

Abstract

The biotechnology revolution has made possible the generation of a plethora of omics data with an exponential growth pace. The organisation of these data, in publicly available omics libraries like BioDataome [32] and others, provides the opportunity for large-scale studies which, in the past, would be virtually impossible. In this work our aim is to create a low dimensional representation of gene expression data using deep neural networks, which could enable applications, such as batch effect removal or a global classifier for omics data. The high dimensionality of the omics data, along with non-biological factors, that cause changes in the data produced by experiments (batch effect), are two of the major difficulties that occur in the study of omics data. Combining deep learning techniques with the Generalised End to End (GE2E) [53] loss function, we attempt to create a feature latent space, where microarray samples from different studies are pushed towards their study's latent space area and away from the other studies one's. The result is a feature space, in which different studies are separated, but also their relative "topology" in terms of similarity is retained. In this work we experimented with both synthetic and real microarray data. In both cases the separation of the different studies samples both visually and in terms of objective metrics was successful and the classification ability of the algorithm on seen and also unseen by the model samples and studies, was also significantly improved. We also conducted large-scale experiments on the algorithm, with more than 100 studies and 19,000 samples.

Περίληψη

Η επανάσταση της βιοτεχνολογίας έχει καταστήσει δυνατή την παραγωγή πληθώρας ομικών δεδομένων, με εκθετικά αυξανόμενο ρυθμό. Η οργάνωση αυτών των δεδομένων σε δημόσια διαθέσιμες βιβλιοθήκες ομικών δεδομένων, όπως η Biodataome [32] και άλλες, παρέχει την δυνατότητα εκτέλεσης μεγάλης κλίμακας μελετών, που στο παρελθόν θα ήταν σχεδόν αδύνατες. Σε αυτή την εργασία στοχεύουμε στο να δημιουργήσουμε μια χαμηλής διάστασης αναπαράσταση για δεδομένα γονιδιακών εκφράσεων χρησιμοποιώντας βαθιά νευρωνικά δίκτυα, η οποία θα μπορούσε να κάνει δυνατές εφαρμογές, όπως η αφαίρεση επίδρασης δέσμης (batch effect) ή όπως η δημιουργία ενός οικουμενικού ταξινομητή για ομικά δεδομένα. Η μεγάλη διάσταση των ομικών δεδομένων καθώς και μη βιολογικοί παράγοντες που προκαλούν αλλαγές στα δεδομένα που παράγονται από τα πειράματα (επίδραση δέσμης), είναι δύο από τις κυριότερες δυσκολίες που προκύπτουν κατά την μελέτη των ομικών δεδομένων. Συνδυάζοντας τεχνικές βαθιάς μάθησης με την συνάρτηση απώλειας Generelised End to End (GE2E) [53], δοκιμάζουμε να δημιουργήσουμε έναν λανθάνοντα χώρο χαρακτηριστικών, όπου τα δεδομένα μικροσυστοιχειών διαφορετικών μελετών σπρώχνονται προς τα κεντροειδή των μελετών στις οποίες ανήκουν και μακριά από τα κεντροειδή των άλλων μελετών. Το αποτέλεσμα είναι ένας χώρος χαρακτηριστικών, στον οποίο διαφορετικές μελέτες διαχωρίζονται, αλλά, επίσης, διατηρείται η σχετική ¨τοπολογία' των μελετών ανάλογα με την ομοιότητά τους. Σε αυτή την εργασία πειραματιστήκαμε με συνθετικά καθώς και πραγματικά δεδομένα μικροσυστοιχειών. Ο διαχωρισμός των δεδομένων από διαφορετικές μελέτες, ταυτόχρονα οπτικά, καθώς και με βάση αντικειμενικών μετρικών, ήταν επιτυχής. Η δυνατότητα ταξινόμησης δεδομένων και μελετών που έχει δει κατά την εκπαίδευση το νευρωνικό δίκτυο, αλλά και αγνώστων, βελτιώθηκε σημαντικά μετά τη χρήση της μεθόδου. Επιπλέον, διεξήγαμε πειράματα μεγάλης κλίμακας στον αλγόριθμο με πάνω από 100 μελέτες και 19.000 δεδομένα.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Use of Machine Learning in Biology

Machine learning addresses the question of how to build computers that improve automatically through experience. It is one of today's most rapidly growing technical fields, lying at the intersection of computer science and statistics, and at the core of artificial intelligence and data science. Machine learning has demonstrated potential in analyzing large, complex biological data [56]. Figure 1.1 shows some machine learning applications for biological data. In order to better understand complex biological phenomena, such as many human diseases or quantitative traits in animals/plants, massive amounts and multiple types of 'big' data are generated from complicated studies. In the not so distant past, data generation was the bottleneck, but now it is data mining, or the extraction of useful biological insights from large datasets. In the past decade, technological advances in data generation have advanced studies of complex biological phenomena. In particular, next generation sequencing (NGS) technologies [17] have allowed researchers to screen changes at varying biological scales, such as genome-wide genetic variation, gene expression and small RNA abundance, epigenetic modifications, protein binding motifs, and chromosome conformation in a high-throughput and cost-efficient manner. The explosion of data, especially omics data, challenges the long-standing methodologies for data analysis [26],[35].

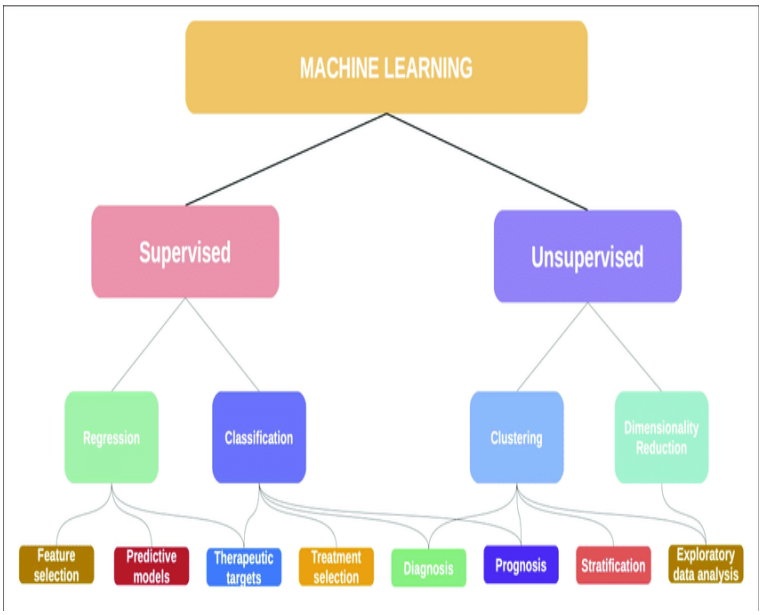

Figure 1.1: Some machine learning applications in cancer and multiomics [3].

1.2 Challenges

Biological systems are complex. Most large-scale studies focus only on one specific aspect of the biological system, for example, genome-wide association studies (GWAS) focus on genetic variants associated with measured phenotypes. However, complex biological phenomena can involve many biological aspects, both intrinsic and extrinsic, and, thus, cannot be fully explained using a single data type. For this reason, the integrated analysis of different data types has been attracting more attention. Integration of different data types should, in theory, lead to a more holistic understanding of complex biological phenomena, but this is difficult due to the challenges of heterogeneous data and the implicitly noisy nature of biological data. Another challenge is data dimensionality, omics data are high resolution, or stated another way, highly dimensional. In biological studies, the number of samples is often limited and much fewer than the number of variables due to costs or available sources (e.g., cancer samples, plant/animal replicates), this is also referred to as the 'curse of dimensionality', which may lead to data sparsity, multicollinearity, multiple testing, and overfitting [2] [26].

The massive and rapid advancements in both biological data generation and machine learning methodologies are promising for the analysis and discovery from complex biological data. The growth of interest in omics technologies in the twenty-first century can be seen in Figure 1.2. However, there are several hurdles. Firstly, interpretation of models derived from some sophisticated machine learning approaches such as deep learning can be difficult if not impossible. In many cases, researchers are more interested in the biological meaning of the predictive model than the predictive accuracy of the model and the 'black box' nature of the model can inhibit interpretation. The information from the model may need further processing and should be carefully interpreted with corresponding biological knowledge. Problems such as sparsity, multicollinearity, and overfitting are difficult to avoid in high-resolution studies such as in omics datasets, although the larger sample size and modern machine learning methods can partially mitigate these problems [2]. To increase the number of samples it may be necessary to combine data from multiple sources, which may be feasible for qualitative data like single-nucleotide polymorphisms (SNPs) but can be hard for quantitative data such as gene expression data due to the many 'hidden' effects such as variation in developing stages or batch effects from experimental methodologies that can confound analyses. It is still an open question how to normalize data from different sources and additional work on data production, sharing, and processing will be necessary [26].

Although improved machine learning methods and the increasing number of available samples show great promise to increase our understanding of complex biological phenomena, building proper machine-learning models can still be challenging due to hidden biological factors such as population structure among samples or evolutionary relationship among genes. Biological datasets should be carefully curated to remove confounders. Without properly accounting for such factors, the models can be overfit, leading to false-positive discovery. To build proper models, the biological and technical factors specific to the modeling scenario need to be taken into account. For example, biological data are often imbalanced, such as the case in some diseases or traits that occur only in a small frac-

tion of a population. It is usually more meaningful to access metrics like precision and recall for the non-major class rather than simple accuracy to evaluate model performance for imbalanced classes in the data [26].

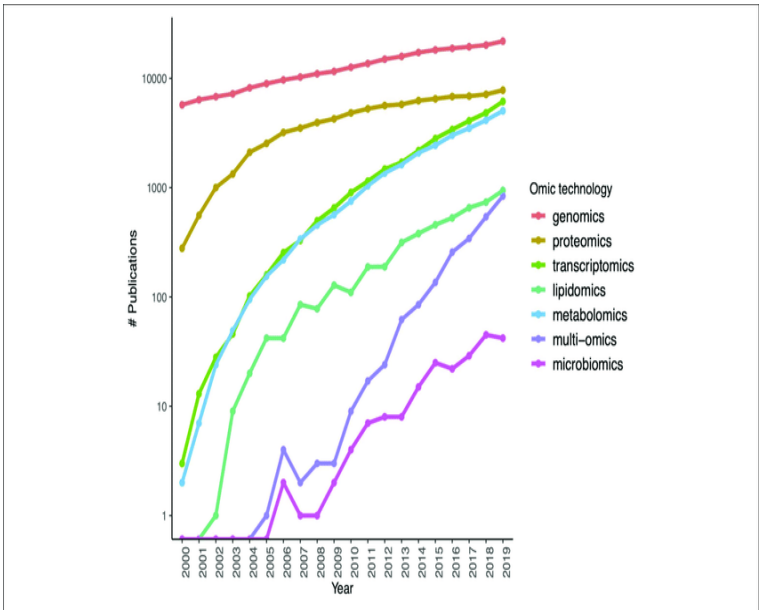

Figure 1.2: Growth of interest in omics technologies in the twenty-first century: the number of Pubmed publications mentioning each omic technology in its title or abstract measured yearly since the year 2000 [3].

1.3 Problem statement and the proposed method

Our goal in this work is to build an algorithm that will be able to produce a low dimensional feature latent space using microarray datasets for gene expression, where individual studies will be distinctly separated from one another, but also their relative "topology" in terms of similarity will be preserved. To achieve this, in this work we propose a neural network with the classical autoencoder architecture in combination with the Generalised End to End (GE2E)[53] loss function, which was first introduced by Google for speaker verification applications. This loss function has the ability

to push the embedding vectors of a dataset's samples towards its centroid and away from the centroids of other datasets.

1.4 Possible applications of the method

Some possible applications that the method could contribute at, are:

Batch effect removal. Batch effects are sub-groups of measurements that have qualitatively different behaviour across conditions and are unrelated to the biological or scientific variables in a study. Non biological factors that can cause batch effects, for example, could be, the laboratory conditions at the time of the experiment or the instruments that have been used to conduct the experiment [27]. The ability of our method to effectively separate the samples from different similar studies could make possible their unification.

The **assessment** of the **similarity** of different datasets in order to unify the ones that are the most similar.

The construction of a **low-dimensional global representation** for gene expression microarray data that would be able to generalise on new, unseen datasets. This low dimensional representation could also prove useful in subsequent classification tasks.

The method's results could also probably **ease the biological interpretation** of transcriptomics data by biologists.

Chapter 2

Artificial Neural Networks

2.1 ANN's Description

Artificial neural networks (ANNs), usually simply called neural networks (NNs), are computing systems inspired by the biological neural networks that constitute animal brains. An interpretation of the structure of biological and artificial neurons can be seen in Figure 2.1. An ANN is based on a collection of connected units or nodes called artificial neurons, which loosely model the neurons in a biological brain. Each connection, like the synapses in a biological brain, can transmit a signal to other neurons. An artificial neuron receives a signal then processes it and can signal neurons connected to it. The "signal" at a connection is a real number, and the output of each neuron is computed by some non-linear function of the sum of its inputs. The connections are called edges. Neurons and edges typically have a weight that adjusts as learning proceeds. The weight increases or decreases the strength of the signal at a connection. Neurons may have a threshold such that a signal is sent only if the aggregate signal crosses that threshold. Typically, neurons are aggregated into layers. Different layers may perform different transformations on their inputs. Signals travel from the first layer (the input layer), to the last layer (the output layer), possibly after traversing the layers multiple times [4][23].

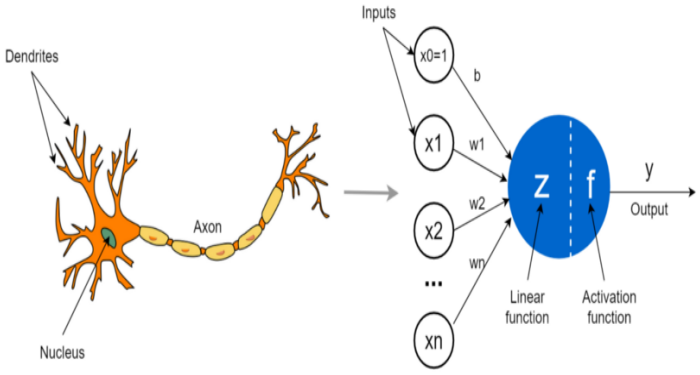

Figure 2.1: Structure interpretation of biological and artificial neurons [45].

2.1.1 Artificial neurons

ANNs are composed of artificial neurons which are conceptually derived from biological neurons. Each artificial neuron has inputs and produces a single output which can be sent to multiple other neurons [36]. The inputs can be the feature values of a sample of external data, such as images or documents, or they can be the outputs of other neurons. The outputs of the final output neurons of the neural net accomplish the task, such as recognizing an object in an image. To find the output of the neuron, we must first take the weighted sum of all the inputs, weighted by the weights of the connections from the inputs to the neuron. We add a bias term to this sum. This weighted sum is sometimes called the activation. This weighted sum is then passed through a (usually nonlinear) activation function to produce the output [4][13].

2.1.2 Organization

The neurons are typically organized into multiple layers, especially in deep learning. Neurons of one layer connect only to neurons of the immediately preceding and immediately following layers. The layer that receives external data is the input layer. The layer that produces the ultimate result is the output layer. In between them are zero or more hidden layers. Single layer and unlayered networks are also used. Between two layers, multiple connection patterns are

possible. They can be 'fully connected', with every neuron in one layer connecting to every neuron in the next layer. They can be pooled, where a group of neurons in one layer connect to a single neuron in the next layer, thereby reducing the number of neurons in that layer. Neurons with such connections form a directed acyclic graph and are known as feedforward networks. Figure 2.2 depicts a schematic of such a network. Alternatively, networks that allow connections between neurons in the same or previous layers are known as recurrent networks [4] [12].

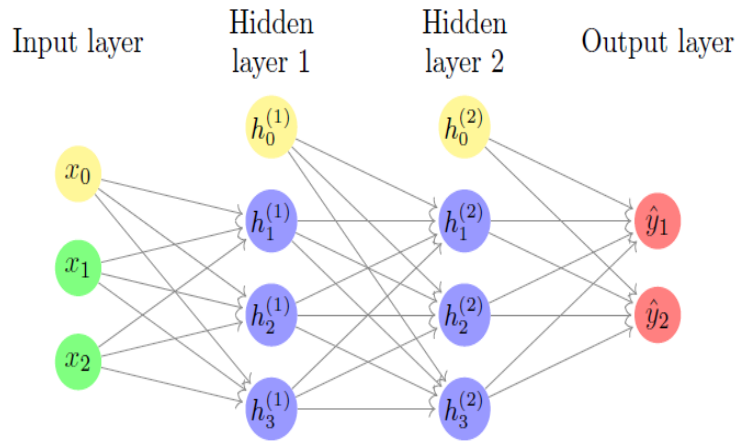

Figure 2.2: Neural network schematic [48].

2.1.3 Cost function

In machine learning, cost functions are used to estimate how badly models are performing. Put simply, a cost function is a measure of how wrong the model is in terms of its ability to estimate the relationship between x and y, where here x represents an input sample and y is the predicted output by the network. This is typically expressed as a difference or distance between the predicted value and the actual value. The cost function (also referred to as loss or error) can be estimated by iteratively running the model to compare estimated predictions against "ground truth" the known values of y [34].

2.1.4 Hyperparameters of an ANN

A hyperparameter is a constant parameter whose value is set before the learning process begins. The values of parameters are derived via learning. Examples of hyperparameters include learning rate, the number of hidden layers and batch size. The values of some hyperparameters can be dependent on those of other hyperparameters. For example, the size of some layers can depend on the overall number of layers [29].

2.2 Deep neural networks

A deep neural network (DNN) is an artificial neural network (ANN) with multiple layers between the input and output layers. DNNs can model complex non-linear relationships. DNN architectures generate compositional models where the object is expressed as a layered composition of primitives. The extra layers enable composition of features from lower layers, potentially modeling complex data with fewer units than a similarly performing shallow network. DNNs can be modeled as feedforward networks in which data flows from the input layer to the output layer without looping back. At first, the DNN creates a map of virtual neurons and assigns random numerical values, or "weights", to connections between them. The weights and inputs are multiplied and return an output between 0 and 1. If the network did not accurately recognize a particular pattern, an algorithm would adjust the weights. That way the algorithm can make certain parameters more influential, until it determines the correct mathematical manipulation to fully process the data [14].

2.3 Learning procedure

Learning is the adaptation of the network's parameters to better handle a task by considering sample observations. Learning involves adjusting the weights (and optional thresholds) of the network to improve the accuracy of the result. This is done by minimizing the observed errors. Even after learning, the error rate typically does not reach 0. If after learning, the error rate is too high, the network typically must be redesigned. Practically this is done by defining a cost function that is evaluated periodically during learning. As

long as its output continues to decline, learning continues. The cost is frequently defined as a statistic whose value can only be approximated. Learning attempts to reduce the total of the differences across the observations. Most learning models can be viewed as a straightforward application of optimization theory and statistical estimation [30].

2.3.1 Training - Backpropagation

Neural networks learn (or are trained) by processing examples, each of which contains a known "input" and "result," forming probability-weighted associations between the two, which are stored within the data structure of the net itself. The training of a neural network from a given example is usually conducted by determining the difference between the processed output of the network (often a prediction) and a target output. This difference is the error. The network then adjusts its weighted associations according to a learning rule and using this error value [4]. An illustration of neural network's training process can be seen in Figure 2.3. Stochastic gradient descent is one of the most prevalent methods for accomplishing this. Stochastic gradient descent (often abbreviated SGD) is an iterative method for optimizing an objective function. It can be regarded as a stochastic approximation of gradient descent optimization, since it replaces the actual gradient (calculated from the entire data set) by an estimate thereof (calculated from a randomly selected subset of the data) [52]. Especially in high-dimensional optimization problems this reduces the very high computational burden, achieving faster iterations in trade for a lower convergence rate [50]. As the weights of a neural network are adjusted, it will produce outputs that are increasingly comparable to the goal outputs. After a sufficient number of these adjustments the training can be terminated based upon certain criteria. This is known as supervised learning [4].

Backpropagation is a method used to adjust the connection weights to compensate for each error found during learning. The error amount is effectively divided among the connections. Technically, backpropagation calculates the gradient (the derivative) of the cost function associated with a given state with respect to the weights. The weight updates can be done via stochastic gradient descent or

other methods, such as Extreme Learning Machines, "No-prop" networks, training without backtracking, "weightless" networks, and non-connectionist neural networks [4].

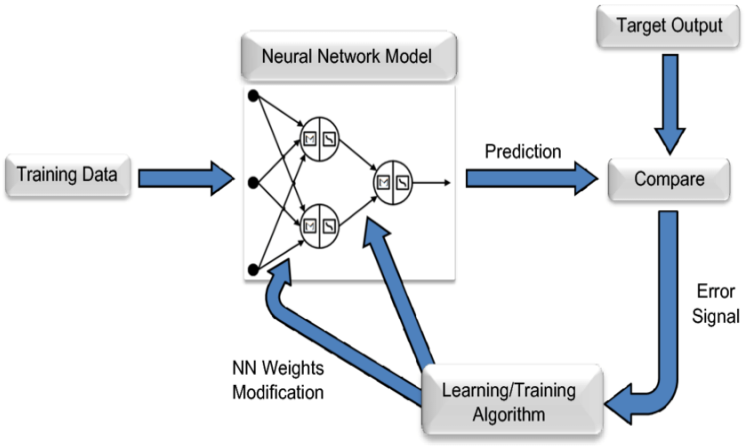

Figure 2.3: Representation of neural network training process [49].

2.4 Learning paradigms

The three major learning paradigms are supervised learning, unsupervised learning and reinforcement learning. They each correspond to a particular learning task

2.4.1 Supervised learning

Supervised learning uses a set of paired inputs and outputs. The learning task is to produce the desired output for each input. In this case the cost function penalizes incorrect deductions. A commonly used cost for regression tasks, is the mean-squared error, which tries to minimize the average squared error between the network's output and the desired output. Tasks suited for supervised learning are pattern recognition (also known as classification) and regression (also known as function approximation). Supervised learning is also applicable to sequential data (e.g., for hand writing, speech and gesture recognition). This can be thought of as learning with a "teacher", in the form of a function that provides continuous feedback on the quality of solutions obtained thus far [4], [41].

2.4.2 Unsupervised learning

In unsupervised learning, input data is given along with the cost function, some function of the data x and the network's output. The cost function is dependent on the task (the model domain) and any a priori assumptions (the implicit properties of the model, its parameters and the observed variables). As a trivial example, consider the model $f(x) = a$ where a is a constant and the cost $C = E[(x - f(x))^2]$. Minimizing this cost produces a value that is equal to the mean of the data. The cost function can be much more complicated. Its form depends on the application: for example, in compression it could be related to the mutual information between x and $f(x)$, whereas in statistical modeling, it could be related to the posterior probability of the model given the data (note that in both of those examples those quantities would be maximized rather than minimized). Tasks that fall within the paradigm of unsupervised learning are in general estimation problems, the applications include clustering, the estimation of statistical distributions, compression and filtering [4].

2.5 Autoencoders

2.5.1 Autoencoders Description

An autoencoder is a type of artificial neural network used to learn efficient codings of unlabeled data (unsupervised learning). The encoding is validated and refined by attempting to regenerate the input from the encoding. The autoencoder learns a representation (encoding) for a set of data, typically for dimensionality reduction, by training the network to ignore insignificant data ("noise") [31]. A representation of the basic structure elements of autoencoders can be seen in Figure 2.4.

Variants exist, aiming to force the learned representations to assume useful properties. Examples are regularized autoencoders (Sparse, Denoising and Contractive), which are effective in learning representations for subsequent classification tasks, and Variational autoencoders, with applications as generative models. Autoencoders are applied to many problems, from facial recognition, feature detection, anomaly detection to acquiring the meaning of words. Autoencoders

are also generative models: they can randomly generate new data that is similar to the input data (training data) [5][31].

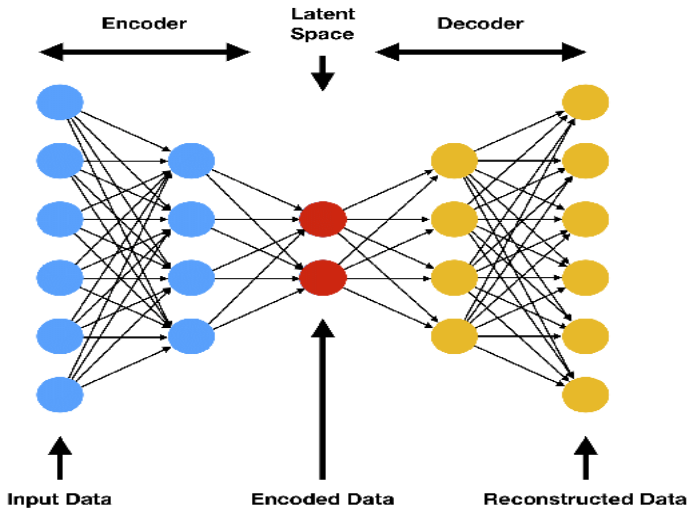

Figure 2.4: Basic structure elements of autoencoders [54].

2.5.2 Basic architecture

An autoencoder has two parts: an encoder that maps the input into the code, and a decoder that maps the code to a reconstruction of the input. The idea of autoencoders has been popular for decades. The first applications date to the 1980s [24]. Their most traditional application was dimensionality reduction or feature learning, but the concept became widely used for learning generative models of data [55]. Some of the most powerful AIs in the 2010s involved autoencoders stacked inside deep neural networks [5].

The simplest way to perform the copying task perfectly would be to duplicate the signal. Instead, autoencoders are typically forced to reconstruct the input approximately, preserving only the most relevant aspects of the data in the copy.

The simplest form of an autoencoder is a feedforward, non-recurrent neural network similar to single layer perceptrons that participate in multilayer perceptrons (MLP) – employing an input layer and an output layer connected by one or more hidden layers. The output layer has the same number of nodes (neurons) as the input layer. Its purpose is to reconstruct its inputs (minimizing the difference between the input and the output) instead of predicting a target value y given inputs x. Therefore, autoencoders learn unsupervised [5].

An autoencoder consists of two parts, the encoder and the decoder, which can be defined as transitions ϕ and ψ, such that:

$$\phi : \mathcal{X} \rightarrow \mathcal{F}$$

$$\psi : \mathcal{F} \rightarrow \mathcal{X}$$

$$\phi, \psi = \arg\min \|\mathcal{X} - (\psi \circ \phi)\mathcal{X}\|^2$$

In the simplest case, with no hidden layers, the encoder stage of an autoencoder takes the input $\mathbf{x} \in R^d = \mathcal{X}$ and maps it to $\mathbf{h} \in R^p = \mathcal{F}$:

$$\mathbf{h} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$

The vector h is usually referred to as code, latent variables, or a latent representation. σ is an element-wise activation function such as a sigmoid function or a rectified linear unit. $\mathbf{W}$ is a weight matrix and $\mathbf{b}$ is a bias vector. Weights and biases are usually initialized randomly, and then updated iteratively during training through backpropagation. After that, the decoder stage of the autoencoder maps $\mathbf{h}$ to the reconstruction $\mathbf{x}'$ of the same shape as $\mathbf{x}$:

$$\mathbf{x}' = \sigma'(\mathbf{W}'\mathbf{h} + \mathbf{b}')$$

where $\sigma', \mathbf{W}'$,and $\mathbf{b}'$ for the decoder may be unrelated to the corresponding σ ,$\mathbf{W}$, and $\mathbf{b}$ for the encoder.

Autoencoders are trained to minimise reconstruction errors (such as squared errors), often referred to as the "loss":

$$\mathcal{L}(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|^2 = \|\mathbf{x} - \sigma'(\mathbf{W}'(\sigma(\mathbf{W}\mathbf{x} + \mathbf{b})) + \mathbf{b}')\|^2$$

where $\mathbf{x}$ is averaged over the training set.

As mentioned before, autoencoder training is performed through backpropagation of the error, just like other feedforward neural networks. Should the feature space $\mathcal{F}$ have lower dimensionality than the input space $\mathcal{X}$, the feature vector $\phi(x)$ can be regarded as a compressed representation of the input x [5]. This is the case of undercomplete autoencoders. If the hidden layers are larger than (overcomplete), or equal to, the input layer, or the hidden units are given enough capacity, an autoencoder can potentially learn the identity function and become useless. However, experimental results found that overcomplete autoencoders might still learn useful features [6].

Chapter 3

Biological Data

In this work we used data that have been acquired by the BioDataome database and specifically microarray gene expression data that have been produced by the GPL570 platform. BioDataome is a database of uniformly preprocessed and disease-annotated omics data with the aim to promote and accelerate the reuse of public data. Currently, BioDataome includes 7320 datasets, 317433 samples spanning 847 diseases and can be easily used in large-scale massive experiments and meta-analysis. There are data aquired by 6 different platforms, namely: GPL570, GPL6244, GPL96, GPL11154, GPL1261 and GPL13534 for two species Homo sapiens and Mus musculus. All datasets are publicly available for querying and downloading via BioDataome web application [32]. In Figure 3.1 can be seen pie charts of the disease distribution, for two different platforms that exist in the biodataome database.

3.1 Genes and gene expression

In biology, a gene is a basic unit of heredity and a sequence of nucleotides in DNA that encodes the synthesis of a gene product, either RNA or protein [15] .

During gene expression, the DNA is first copied into RNA. The RNA can be directly functional or be the intermediate template for a protein that performs a function. The transmission of genes

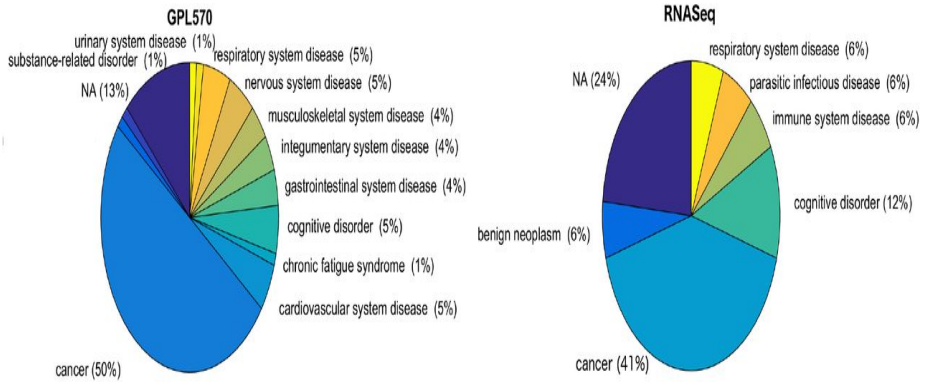

Figure 3.1: Pie charts with the disease/state categories of the test sets for two platforms that exist in the Biodataome database. Approximately half of the human gene expression datasets study cancerous samples [32].

to an organism's offspring is the basis of the inheritance of phenotypic traits. These genes make different DNA sequences called genotypes. Genotypes along with environmental and developmental factors determine what the phenotypes will be. Most biological traits are under the influence of polygenes (many different genes) as well as gene–environment interactions. In all organisms, two steps are required to read the information encoded in a gene's DNA and produce the protein it specifies. First, the gene's DNA is transcribed to messenger RNA (mRNA). Second, that mRNA is translated to protein. RNA-coding genes must still go through the first step, but are not translated into protein [10]. The process of producing a biologically functional molecule of either RNA or protein is called gene expression, and the resulting molecule is called a gene product [19].

3.2 Microarrays

A microarray is a laboratory tool used to detect the expression of thousands of genes at the same time. DNA microarrays are microscope slides that are printed with thousands of tiny spots in defined positions, with each spot containing a known DNA sequence or gene. Often, these slides are referred to as gene chips or DNA chips. The DNA molecules attached to each slide act as probes to detect gene expression, which is also known as the transcriptome or the set of messenger RNA (mRNA) transcripts expressed by a group of genes [38].

To perform a microarray analysis, mRNA molecules are typically collected from both an experimental sample and a reference sample. For example, the reference sample could be collected from a healthy individual, and the experimental sample could be collected from an individual with a disease like cancer. The two mRNA samples are then converted into complementary DNA (cDNA), and each sample is labeled with a fluorescent probe of a different color. For instance, the experimental cDNA sample may be labeled with a red fluorescent dye, whereas the reference cDNA may be labeled with a green fluorescent dye. The two samples are then mixed together and allowed to bind to the microarray slide. The process in which the cDNA molecules bind to the DNA probes on the slide is called hybridization. Following hybridization, the microarray is scanned to measure the expression of each gene printed on the slide. If the expression of a particular gene is higher in the experimental sample than in the reference sample, then the corresponding spot on the microarray appears red. In contrast, if the expression in the experimental sample is lower than in the reference sample, then the spot appears green. Finally, if there is equal expression in the two samples, then the spot appears yellow. The data gathered through microarrays can be used to create gene expression profiles, which show simultaneous changes in the expression of many genes in response to a particular condition or treatment [38].

3.2.1 Affymetrix GeneChip

In this work we used omics data that have been produced using the Affymetrix U133 Plus 2.0 Array (NCBI GEO ID: GPL570) platform.

The Affymetrix GeneChip system is a commercial microarray platform that allows whole genome gene expression analysis for a wide variety of experimental organisms. This system has three major advantages over other array systems, it is easy to get rapid results, it has the capability to monitor the expression of every gene in the genome and it is the most widely used commercial microarray platform [21].

GeneChip® Human Genome U133 Plus 2.0 Array (GPL570)

The GeneChip® Human Genome U133 Plus 2.0 Array offers a comprehensive analysis of genome-wide expression on a single array. It analyzes the relative expression level of more than 47,000 transcripts and variants, including more than 38,500 well characterized genes and UniGenes. It offers an additional 9,900 probe sets, representing approximately 6,500 new genes, compared to the previous generation HG-U133 Set [18]. Every sample that is taken using this platform is an array of 45675 values. Every value represents how much the certain gene is expressed.

Chapter 4

Generalized End-to-End GE2E Loss

The Generalized End-to-End Loss (GE2E) has been proposed by Li Wan, Quan Wang and others in the paper "GENERALIZED END-TO-END LOSS FOR SPEAKER VERIFICATION" [53]. The authors' purpose in this work is to make the training of speaker verification models more efficient and for this reason they propose the GE2E loss. In our work, we use the GE2E in a significantly different manner as long as in a completely different field of study. The main distinction in our application of the method, is that we want to reconstruct the created by the method embeddings in order for them to contain the important biological information. As a result, we use an autoencoder network structure (which mean that we add an additional loss to the method, the reconstruction loss), as opposed to the study cited above, in which the authors solely use the GE2E loss for classification purposes and do not care about reconstructing the reduced audio samples.

As it is explained by the authors [53] Generalized end-to-end (GE2E) training is based on processing a large number of samples at once, in the form of a batch that contains N datasets-studies, and M samples from each dataset in average.

4.1 Description of the method

The first requirement of the method is to have data batches, that are a mixture of data samples, from the different datasets that are used for the training of the network.

For $\mathbf{N} \times \mathbf{M}$ samples where N is the number of studies and M the number of samples of each study $\mathbf{x_{ji}}$ ($1 \leq j \leq N$ and $1 \leq i \leq M$) represents the ith sample of the jth study.

The output of the neural network application on the $\mathbf{x_{ji}}$ sample is denoted as $f(\mathbf{x_{ji}}; \mathbf{w})$ where $\mathbf{w}$ represents all parameters of the neural network. The embedding vector (d-vector) is defined as the L2 normalization of the network output [53]:

$$\mathbf{e}_{ji} = \frac{f(x_{ji}; w)}{\|f(x_{ji}; w)\|^2} \tag{4.1}$$

Here $\mathbf{e}_{ji}$ represents the embedding vector of the jth dataset's ith sample. The centroid of the embedding vectors from the kth dataset $[\mathbf{e}_{j1}, \ldots, \mathbf{e}_{jM}]$ is defined as $\mathbf{c}_k$ via the following equation:

$$\mathbf{c}_k = \frac{1}{M} \sum_{m=1}^{M} \mathbf{e_{km}} \tag{4.2}$$

The similarity matrix $\mathbf{S}_{ji,k}$ is a $N \times M \times N$ matrix, which is defined as the scaled cosine similarities between each embedding vector $\mathbf{e}_{ji}$ to all centroids $\mathbf{c}_k$ ($1 \leq j, k \leq N$, and $1 \leq i \leq M$):

$$\mathbf{S}_{ji,k} = w \cdot cos(\mathbf{e}_{ji}, \mathbf{c}_k) + b \tag{4.3}$$

where w and b are two learnable parameters. We constrain the weight to be positive $w > 0$ because we want the similarity to be larger when cosine similarity is larger [53].

During the training, we want the embedding of each sample to be similar to the centroid of all that dataset's embeddings, while at

the same time, far from other studies' centroids. As shown in the similarity matrix in Figure 4.1, we want the similarity values of colored areas to be large, and the values of gray areas to be small. Figure 4.2 illustrates the same concept in a different way: we want the blue embedding vector to be close to its own dataset's centroid (blue triangle), and far from the others centroids (red and purple triangles),especially the closest one (red triangle).

To implement this concept, given an embedding vector $\mathbf{e}_{ji}$, all centroids $\mathbf{c}_k$, and the corresponding similarity matrix $\mathbf{S}_{ji,k}$, we put a softmax on $\mathbf{S}_{ji,k}$ for k = 1, . . . , N that makes the output equal to 1 if k = j, otherwise makes the output equal to 0. Thus, the loss on each embedding vector $\mathbf{e}_{ji}$ could be defined as:

$$L(\mathbf{e}_{ji}) = -\mathbf{S}_{ji,j} + log \sum_{k=1}^{N} exp(\mathbf{S}_{ji,k}) \tag{4.4}$$

This loss function means that we push each embedding vector close to its centroid and pull it away from all other centroids [53].

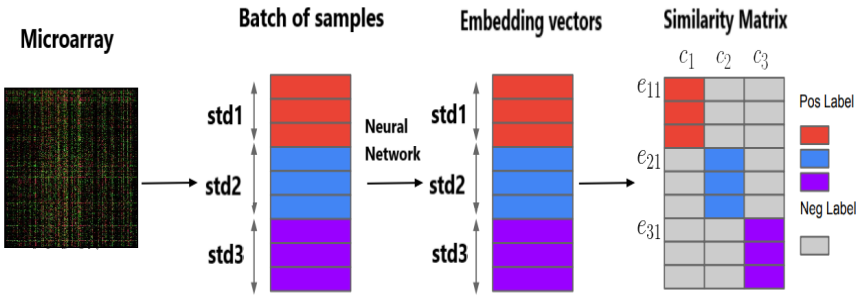

Figure 4.1: System overview. Different colors indicate samples/embeddings from different speakers [53].

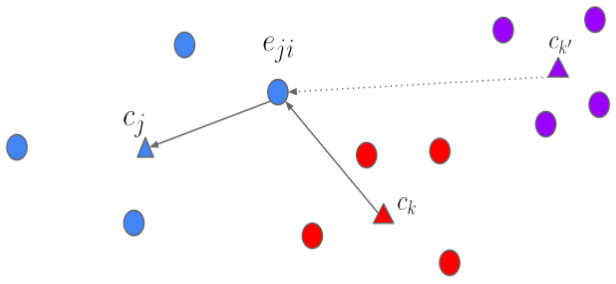

Figure 4.2: GE2E loss pushes the embedding towards the centroid of the true dataset, and away from the centroid of the most similar different dataset [53].

Chapter 5

Model Definition & Implementation

The deep neural network that was built for the task is an autoencoder, created using the python library Pytorch. Pytorch is an open source optimized tensor library primarily used for Deep Learning applications using GPUs and CPUs. The use of tensors on Graphics processing units (GPUs) can dramatically speed up computational processes for deep learning.

5.1 Model characteristics

5.1.1 Data loading

Unlike most common machine learning applications, we have to deal with multiple datasets on each experiment. We want every batch that is fed as input to the neural network to be consisted of random samples from all the studies that we are going to use. To do that, we use a custom data loader that we created. In each model iteration, the loader creates a batch of a given number of samples from all the different datasets. The loader also makes sure that all the available samples are seen by the network after some iterations.

5.1.2 Optimizer

In machine learning, we use optimizers in order to improve the speed and performance of our models. An optimizer is a function or an

algorithm that modifies the attributes of the neural network, such as weights and learning rate. Thus, it helps in reducing the overall loss and improve the accuracy [22].

Our model uses the AdamW optimizer, which is an improved version of Adam optimizer. Adaptive Moment Estimation is an algorithm for optimization technique for gradient descent. The method is really efficient when working with large problems involving a lot of data or parameters [1]. It requires less memory and is efficient. Adam optimizer involves a combination of two gradient descent methodologies:

Momentum:

This algorithm is used to accelerate the gradient descent algorithm by taking into consideration the 'exponentially weighted average' of the gradients. Using averages makes the algorithm converge towards the minima in a faster pace.

Root Mean Square Propagation (RMSP):

Root mean square prop or RMSprop is an adaptive learning algorithm that tries to improve AdaGrad. Instead of taking the cumulative sum of squared gradients like in AdaGrad, it takes the 'exponential moving average'.

Adam Optimizer inherits the strengths or the positive attributes of the above two methods and builds upon them to give a more optimized gradient descent. AdamW modifies the typical implementation of weight decay in Adam to combat Adam's known convergence problems by decoupling the weight decay from the gradient updates [1].

5.1.3 Activation and loss functions

Activation functions

Activation function decides, whether a neuron should be activated or not by calculating weighted sum and further adding bias with it. In Deep learning, a neural network without an activation function is just a linear regression model as these functions actually do the non-linear computations to the input of a neural network making it capable to learn and perform more complex tasks [42].

In our model we use ReLU (Rectified Linear Unit) and SiLU (Sigmoid Linear Units) as our activation functions.

ReLU:

The rectified linear activation function or ReLU for short is a piecewise linear function that will output the input directly if it is positive, otherwise, it will output zero. It has become the default activation function for many types of neural networks because a model that uses it is easier to train and often achieves better performance [9]. As a formula:

$$\begin{aligned} R(x) &= x \quad , \quad x > 0 \\ R(x) &= 0 \quad , \quad x <= 0 \end{aligned} \tag{5.1}$$

SiLU:

Sigmoid Linear Unit (SiLU) also known as Sigmoid-Weighted Linear Unit or Swish, is an activation function that uses the sigmoid function with multiplication itself. In other words, the activation of the SiLU is computed by the sigmoid function multiplied by its input [51].

$$R(x) = x * \sigma(x) \tag{5.2}$$

where $\sigma(x)$ is the sigmoid function:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{5.3}$$

Like ReLU, Swish is bounded below (meaning as x approaches negative infinity, y approaches some constant value) but unbounded above (meaning as x approaches positive infinity, y approaches infinity). However, unlike ReLU, Swish is smooth (it does not have sudden changes of motion or a vertex). Additionally, Swish is non-monotonic, meaning that there is not always a singularly and continually positive (or negative) derivative throughout the entire function [57].

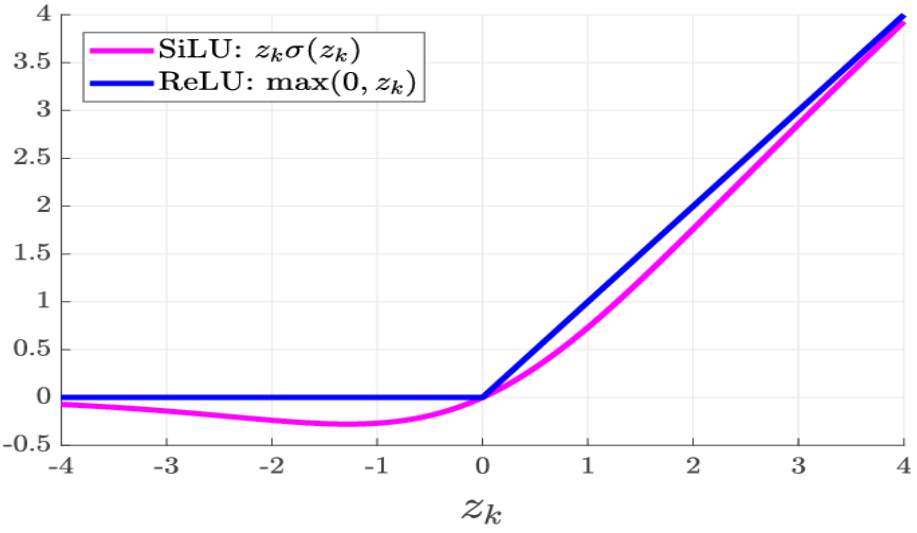

Figure 5.1: SiLU and ReLU activation functions comparison [16].

Loss Functions

A loss function is used to optimize the parameter values in a neural network model. Loss functions map a set of parameter values for the network onto a scalar value that indicates how well those parameter accomplish the task the network is intended to do [33].

For the needs of our model we used Mean Squared Error (MSE) and Cross-Entropy loss functions.

Mean Squared Error Mean squared error (MSE) is the most commonly used loss function for regression. The loss is the mean

overseen data of the squared differences between true and predicted values, or writing it as a formula:

$$L(y, \hat{y}) = \frac{1}{N} \sum_{i=0}^{N} (y - \hat{y}_i)^2 \tag{5.4}$$

where y is the true value $\hat{y}$ is the predicted value and N is the total number of data [37].

Cross-Entropy Loss

Cross entropy loss is a metric used to measure how well a classification model in machine learning performs. The loss (or error) is measured as a number between 0 and 1, with 0 being a perfect model. The goal is generally to get your model as close to 0 as possible. Cross entropy loss is often considered interchangeable with logistic loss. Cross entropy loss measures the difference between the discovered probability distribution of a machine learning classification model and the predicted distribution [7].

5.1.4 Dropouts

Dropout is a regularization method that approximates training a large number of neural networks with different architectures in parallel. During training, some number of layer outputs are randomly ignored or "dropped out." This has the effect of making the layer look-like and be treated-like a layer with a different number of nodes and connectivity to the prior layer. In effect, each update to a layer during training is performed with a different "view" of the configured layer. Dropout has the effect of making the training process noisy, forcing nodes within a layer to probabilistically take on more or less responsibility for the inputs. This conceptualization suggests that perhaps dropout breaks-up situations where network layers co-adapt to correct mistakes from prior layers, in turn making the model more robust. Thus, dropouts are simple way to prevent neural networks from overfitting [8]. The use of dropouts has been proved essential for the stability of our model.

5.2 Model architecture

As mentioned before, the model follows the classic autoencoder architecture. A three layer encoder is followed by an identical reverse in structure three layer decoder. All layers are linear and fully connected with the following ones. The encoder layers decrease in size as we progress deeper in the network, the reverse happens with the decoder.

The first layer is followed by a 10% dropout as long as a ReLU function. Then follows the second layer along with a 5% dropout after it. Lastly, the third layer is followed by a SiLU function. The use of SiLU function in the end of the encoder has significantly improved the stability of the model. The decoder has the same architecture but reverse in order. A visual representation of the model can be seen at figure 6.2.

5.3 The training algorithm

In this section, we are going to describe the main actions that are being performed on every training iteration.

5.3.1 Data loading and normalisation

On every iteration a data batch is created by the loader, which is described at subsection 6.1.1 . The batch contains a predetermined number of data from each dataset that is going to be used. Then, using torch command nan_to_num, nan values that may occur are replaced with zeros. After that, L2 normalisation is performed on the batch.

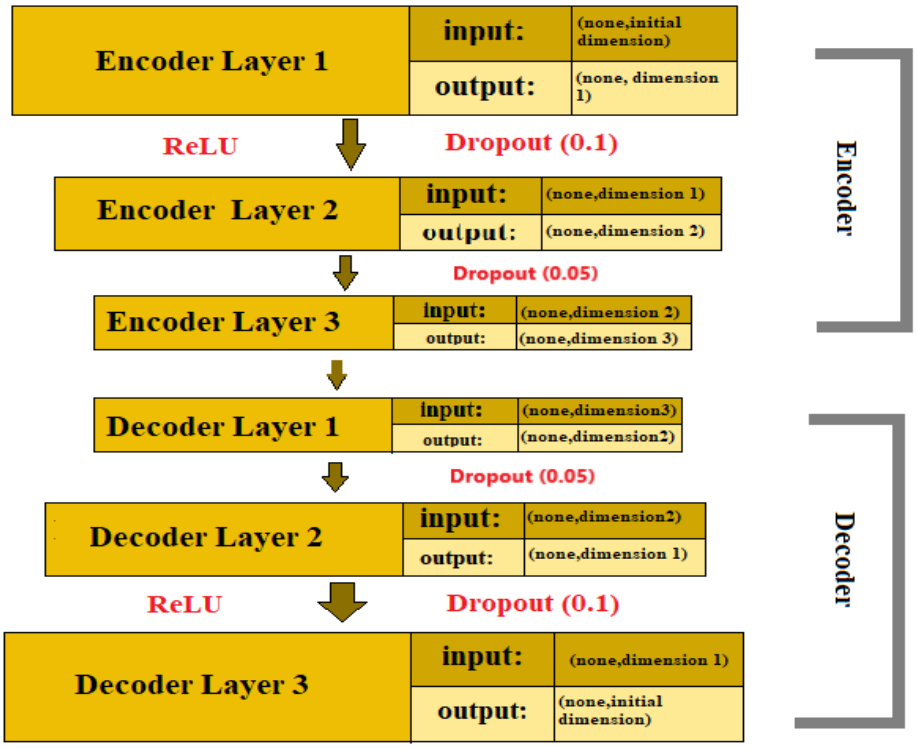

Figure 5.2: Neural Network summary graph

5.3.2 Reconstruction and similarity loss calculation

Reconstruction loss: After the data batch passes through the model, the reconstruction loss is calculated by measuring the mean squared error between each element of the input batch and the reconstructed ones. The mean value of all elements, mean squared error, is our final reconstruction loss. As an equation the reconstruction loss for a given sample x_{ji} is:

$$L_{rec}(x_{ji}) = \parallel x_{ji} - \hat{x_{ji}} \parallel_2^2 \tag{5.5}$$

where x_{ji} is the initial sample and $\hat{x_{ji}}$ is the reconstructed sample.

Similarity loss: In every iteration the similarity matrix $S_{ji,k}$ of the batch is calculated from the output of the last encoder layer which is our embeddings vector matrix. The similarity matrix is calculated as it is discussed in chapter 4. The similarity matrix is an $N \times M$ matrix, where N is the number of samples and M the number of studies on every batch. Every sample is represented by M values on the similarity matrix, which are the scaled cosine similarities between each embedding vector to all studies centroids on the embedding vector space. On every iteration, the similarity matrix of the batch is compared with a target matrix using CrossEntropyLoss. The target matrix values are unique integers for every different study sample. The outcome of CrossEntropyLoss applied on these two matrices is our similarity loss. As an equation the similarity loss for a given embedding vector e_{ji} is:

$$L_{sim}(\mathbf{e}_{ji}) = -\mathbf{S}_{ji,j} + log \sum_{k=1}^{N} exp(\mathbf{S}_{ji,k}) \tag{5.6}$$

Total loss: Our total loss on every iteration is the combination of these two losses. Also, a weight factor can be added that has proven to help training speed. As an equation the total loss on every training iteration is:

$$L_{total} = \frac{1}{N \cdot M}(a \cdot \sum_{ji} L_{sim}(e_{ji}) + \sum_{ji} L_{rec}(x_{ji})) \tag{5.7}$$

where N is the number of studies, M is the number of samples of every study and a is a weight factor.

5.4 Dimensionality reduction and visualisation tools

In this chapter, the basic principles of our visualisation and dimensionality reduction tools are being presented. In our work, we used Principal component analysis (PCA) in order to visualise our results and also to perform an initial reduction to our microarray data dimension. Also, t-distributed stochastic neighbor embedding (t-SNE) was used for more sophisticated visualisation of the results.

5.4.1 Principal component analysis (PCA)

Principal Components Analysis, also known as PCA, is a technique commonly used for reducing the dimensionality of data while preserving as much as possible of the information contained in the original data. PCA achieves this goal by projecting data onto a lower-dimensional subspace that retains most of the variance among the data points [44].

In data analysis, the first principal component of a set of p variables, presumed to be jointly normally distributed, is the derived variable formed as a linear combination of the original variables

that explains the most variance. The second principal component explains the most variance in what is left once the effect of the first component is removed, and we may proceed through p iterations until all the variance is explained. PCA is most commonly used when many of the variables are highly correlated with each other and it is desirable to reduce their number to an independent set. It can be shown that the principal components are eigenvectors of the data's covariance matrix. Thus, the principal components are often computed by eigendecomposition of the data covariance matrix or singular value decomposition of the data matrix. PCA is the simplest of the true eigenvector-based multivariate analyses and is closely related to factor analysis [44] [46].

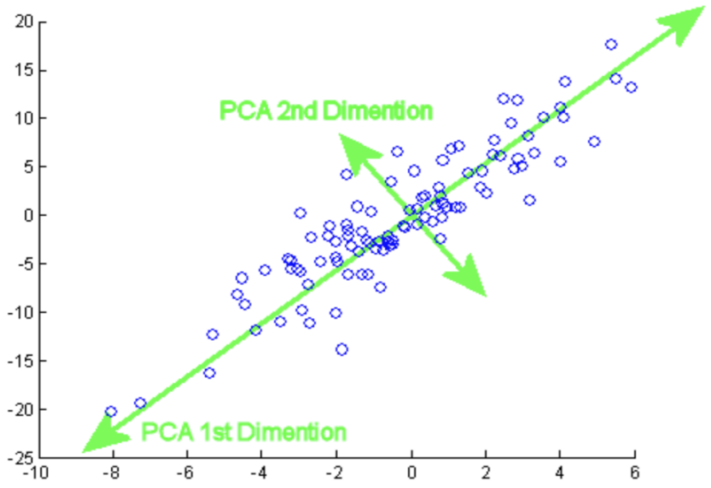

Figure 5.3: Example graph of the first two principal components of a distribution in the two dimensional space [44].

5.4.2 t-distributed stochastic neighbor embedding (t-SNE)

t-SNE is a machine learning technique for dimensionality reduction that helps to identify relevant patterns. The main advantage of t-SNE is the ability to preserve local structure. This means, roughly, that points which are close to one another in the high-dimensional

data set will tend to be close to one another in the chart.

The t-SNE algorithm models the probability distribution of neighbors around each point. Here, the term neighbors refers to the set of points which are closest to each point. In the original, high-dimensional space this is modeled as a Gaussian distribution. In the 2-dimensional output space this is modeled as a t-distribution. The goal of the procedure is to find a mapping onto the 2-dimensional space that minimizes the differences between these two distributions over all points. The fatter tails of a t-distribution compared to a Gaussian help to spread the points more evenly in the 2-dimensional space [25].

The main parameter controlling the fitting is called perplexity. Perplexity is roughly equivalent to the number of nearest neighbors considered when matching the original and fitted distributions for each point. A low perplexity means we care about local scale and focus on the closest other points. High perplexity takes more of a "big picture" approach.

Because the distributions are distance based, all the data must be numeric. You should convert categorical variables to numeric ones by binary encoding or a similar method. It is also often useful to normalize the data, so each variable is on the same scale. This avoids variables with a larger numeric range dominating the analysis [25].

Chapter 6

Results on synthetic data

Initially, we tested the method on synthetic datasets that emulate microarray data.

6.1 Synthetic data generation

In order to create datasets of data that are not too far apart in their high dimensional space (so the effectiveness of the method can be tested), these datasets must be created simultaneously and have some correlation. In the following table, the synthetic data's characteristics are presented.

Table 6.1: Synthetic data characteristics

Synthetic data characteristics	
Variable Name	Description
k	Data's inner dimension
W	Percentage of control samples
P	Data's final dimension
a	Noise coefficient
cl	Number of classes per dataset
n	Number of dataset's samples

In order for our data to emulate real microarray data, they have to be characterised of an inner dimension (k) much smaller than the final one (P). Studies have shown [43] that omics data can be greatly reduced in dimension without valuable biological information to be lost, as can also be seen in figure 6.1.

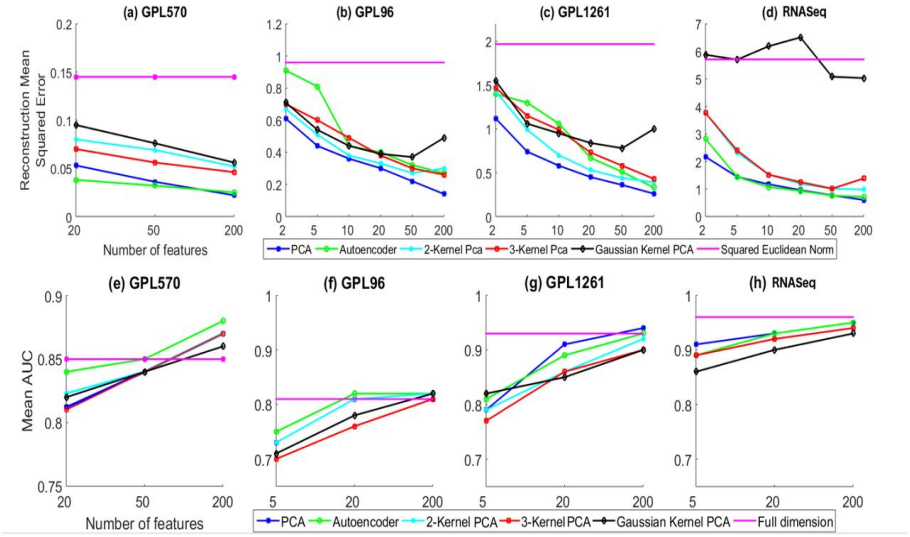

Figure 6.1: Performance assessment of dimensionality reduction techniques on datasets from different platforms both in terms of reconstruction error (upper row of panels) and classification through mean AUC (lower row of panels) [43].

Our data also have a percentage of control samples (W). Control samples in general are samples of a given dataset that are well known and are used to ensure that analyses are properly performed and results are reliable. In the case of omics data, control samples typically are samples from healthy individuals in terms of the condition that is studied.

In order to produce a dataset that contain samples from two different classes (e.g. healthy and unhealthy samples), initially we create two $k \times k$ matrices, from two different random normal distributions. Then these matrices are multiplied by their transposed version and Cholesky decomposition is performed on this product. Let's call the matrices that occur A and B. The number of the control samples w is then randomly calculated (restricted by a

lower bound that we choose) and two matrices of dimensions $w \times k$ and $(n - w) \times k$ are also created by two different random uniform distributions. We then, multiply them by pairs with the initial A and B matrices and combine those products into one nxk matrix which we name C.

Also, a $k \times p$ matrix is created by a random uniform distribution. We call it projection matrix P. The $n \times p$ product of P with C is calculated and white noise is added. This is the first dataset. In order to create the next dataset, we slightly perturb the projection matrix P and we follow the same method.

6.2 One class per dataset example

In this example, 5 synthetic datasets that we produced were used for the model's training. In this example, every dataset has one class of samples. Some important values about the synthetic data of the example are being presented in table 6.2.

Table 6.2: Synthetic data's parameter values

Synthetic data's parameter values	
Variable Name	Value
Dimension (P)	1000
Inner dimension (k)	50
Number of classes (cl)	1
Control samples percentage (W)	20%
Noise coefficient (a)	0.0001
Dataset 1,2	200 samples
Dataset 3,4	300 samples
Dataset 5	500 samples

In table 6.3 some important values about the model are being presented.

Table 6.3: Important model's values

Important model values	
Name	Value
Layer 1 size	2048
Layer 2 size	1024
Layer 3 size	256
Studies pre batch	5
Samples per study	200
Initial data's dimension	1000
Initial learning rate	1e-5
k of kNN	25

In order to test our model's classification capabilities, we used k-nearest neighbors algorithm (knn). The k-nearest neighbors algorithm, is a non-parametric, supervised learning classifier, which uses proximity to make classifications or predictions about the grouping of an individual data point. While it can be used for either regression or classification problems, it is typically used as a classification algorithm, working off the assumption that similar points can be found near one another [28].

The training set, of the knn algorithm, is a batch of equal in number samples from all the datasets used for the training. The batch goes through the model, thus the embeddings of the samples are created and then they are normalised. These embeddings serve as the knn's training set. The test set is compromised by samples that the network hasn't seen during the training.

In order to validate our model, we compare the results of the knn algorithm on the test set, before (initial dimension) and after the model appliance (embedding vectors). The knn algorithm's accuracy on the raw data was 19.7%. The accuracy using the created embeddings improved to 55.9%. The relative improvement in this case is 183.8% .

In figure 6.1 can be seen a visual representation of our data on the two-dimensional space created using PCA and another one created using t-SNE . Every color represents a different dataset and every point a different sample.

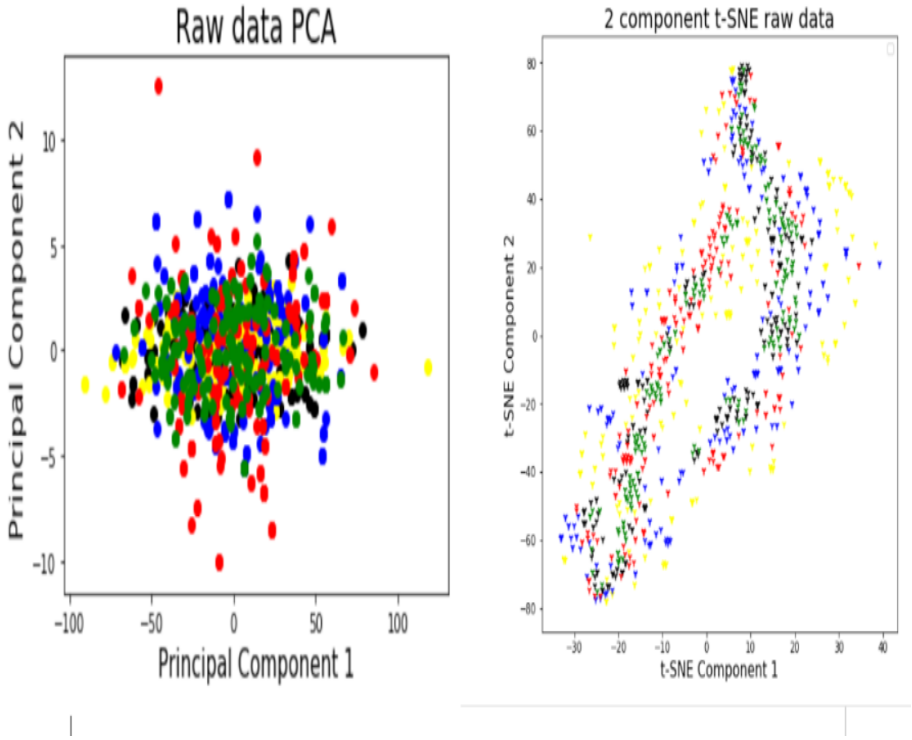

Figure 6.2: Visual representation of the data on the two-dimensional space. On the left using t-SNE and on the right using PCA.

In figure 6.2 the separation progress is presented for different epochs of the training. It can be seen that after 1300 training epochs the different datasets have been completely separated.

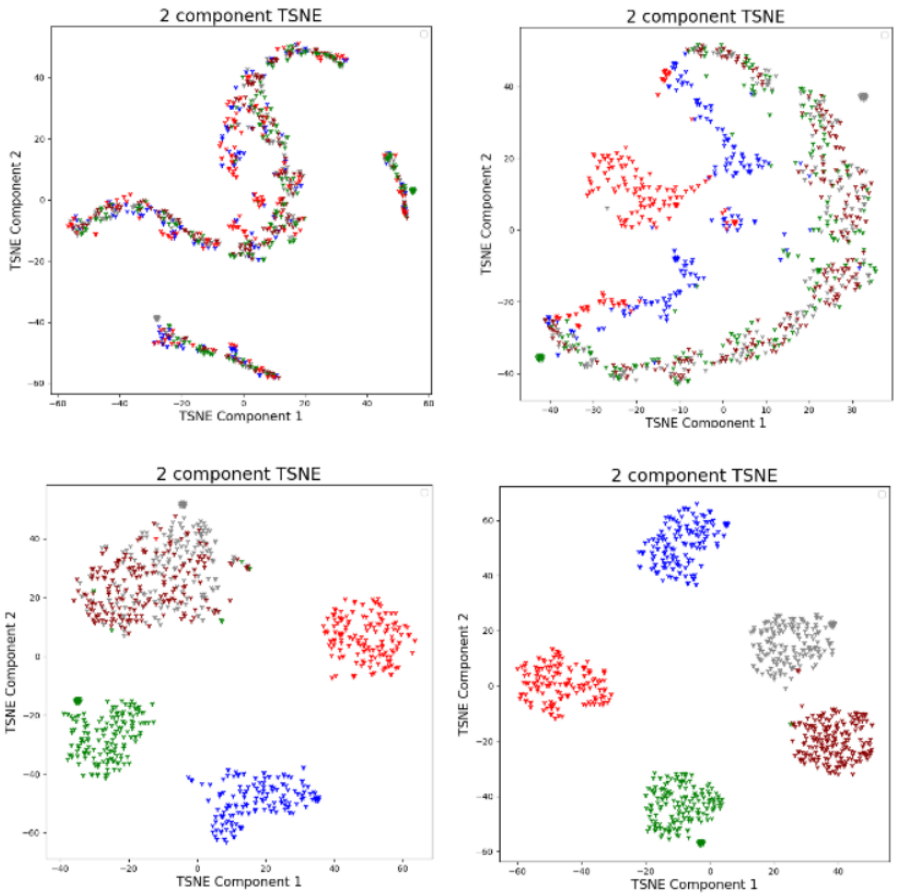

Figure 6.3: Two-dimensional representation using t-SNE of the model results after different number of training epochs. The first plot on the left has been created after 500 epochs, the one on the right after 600, the bottom left after 1000 epochs and the bottom right one after 1300 epochs.

6.3 Two class per dataset example

In this example, we perform the same tests as we did for the previous example, but this time for the case of datasets that are a combination of two different classes. All the other variables and hyperparameters remain the same and have been already presented in tables 6.1 and 6.2.

Following the same procedure that has been described in the one class per study example, the knn algorithm's accuracy on the test set improved from 32.3% to 86.1%. The relative improvement in this case is 166.6% .

In figure 6.4 can be seen a visual representation of our data on the two-dimensional space created using PCA and another one created using t-SNE . Every color represents a different dataset and every point a different sample.

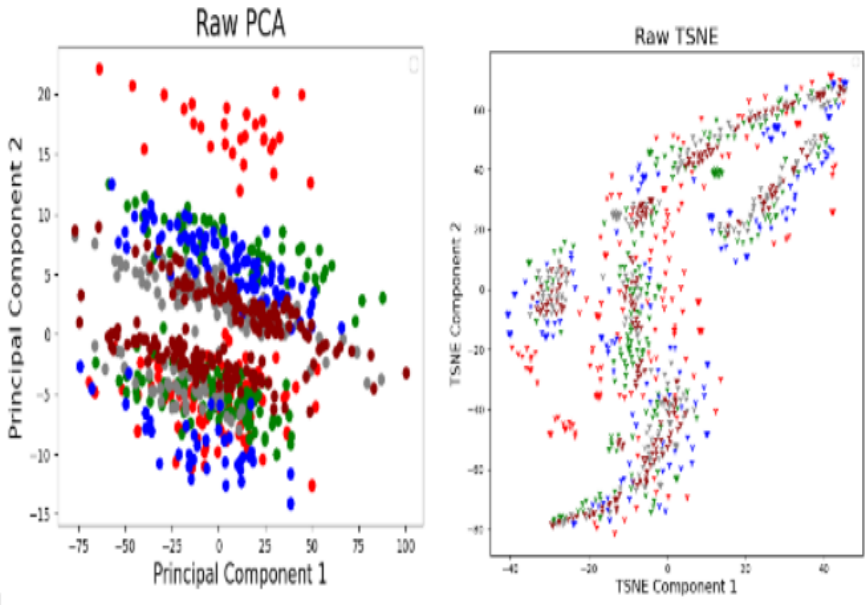

Figure 6.4: Visual representation of the data on the two-dimensional space. On the right using t-SNE and on the left using PCA.

In figure 6.5 the separation progress is presented for different epochs of the training. It can be seen that after 800 training epochs the different datasets have been completely separated and also that the two classes are also distinguishable.

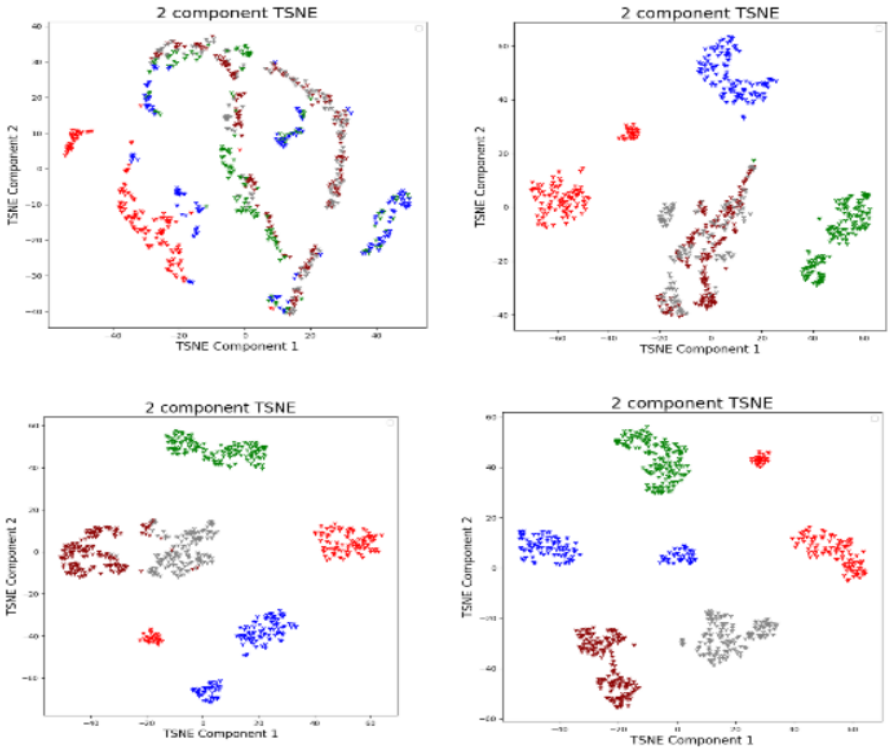

Figure 6.5: Two-dimensional representation using t-SNE of the model results after different number of training epochs. The first plot on the left has been created after 200 epochs, the one on the right after 400, the bottom left one after 600 epochs and the bottom right one after 800 epochs.

It has to be noted that the classification accuracy increases as the epochs increase, even after the different datasets are separated visually.

Chapter 7

Implementation on real microarray data

In this chapter, the results of the algorithm in real microarray data are presented. In this work, we used datasets created with the GPL570 platform. Every sample taken using this platform, is an one dimensional array that consists of 45675 gene expression values. In order to make our data usable but also keep the useful biological information, we perform an initial reduction on the data's dimension using Principal Component Analysis. The data that are used to validate our results were not part of the principal components calculation.

7.1 Multiple studies per disease-dataset example

In this example, for the model training 12 datasets have been used. Every datasets contains samples from multiple studies on a given disease. In table 7.1 can be seen a list of the diseases under examination, as long as the number of samples of each one. In table 7.2 some important values about the example are being presented.

Table 7.1: Diseases and samples number per disease

Diseases and samples number per disease	
Disease Name	Samples number
Acquired immunodeficiency syndrome	370
Asthma	1436
Breast cancer	451
Cancer	673
Chronic lymphocytic leukemia	776
Lung cancer	989
Melanoma	506
Multiple myeloma	1400
Obesity	626
Ovarian cancer	661
Olcerative colitis	400
Total	8879

Table 7.2: Important model's values

Important model's values	
Name	Value
Layer 1 size	2048
Layer 2 size	1024
Layer 3 size	256
Studies pre batch	12
Samples per study	200
Initial data's dimension	500
Initial learning rate	1e-5
k of kNN	25

In order to test our model's classification capabilities, we used k-nearest neighbors algorithm (knn). The training set, for the knn algorithm, is a batch of equal in number samples from all the diseases used for the training. The batch goes through the model, thus the embeddings of the samples are created, and then they are normalised. These embeddings serve as the knn algorithm's training

set. The test set is compromised by samples that the network hasn't seen during the training and goes through the same process as the training set of the knn algorithm. In order to validate our model, we compare the results of the knn algorithm before (initial dimension) and after the model appliance (embedding vectors) on the test set. The knn algorithm's accuracy on the raw data of the test set was 62.0% . The accuracy using the created embeddings improved to 98.2% . The relative improvement in this case is 58.4% .

In figure 7.1 can be seen a visual representation of our data on the two-dimensional space created using PCA and another one created using t-SNE. Every color represents a different disease and every point a different sample.

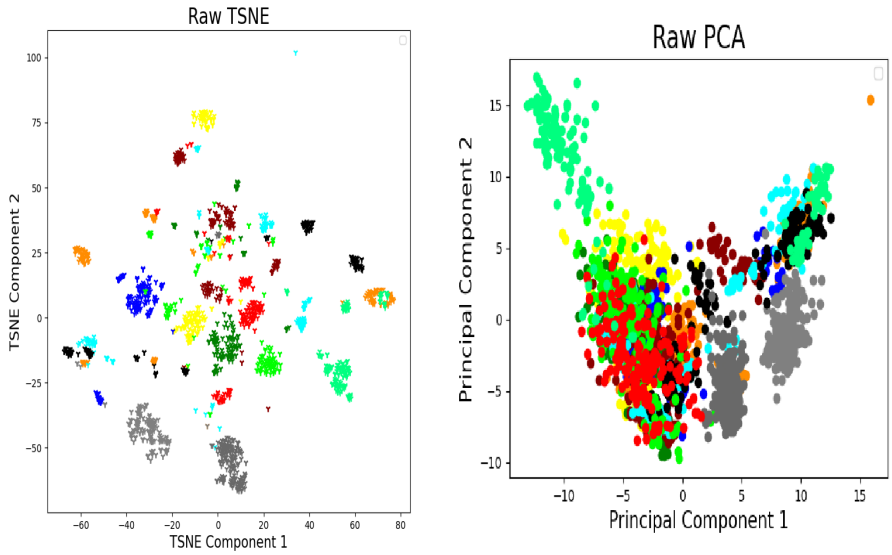

Figure 7.1: Visual representation of the data on the two-dimensional space. On the right using t-SNE and on the left using PCA.

In figure 7.2 the separation progress is presented for different epochs of the training. It can be seen that at 400 training epochs the different diseases have been completely separated.

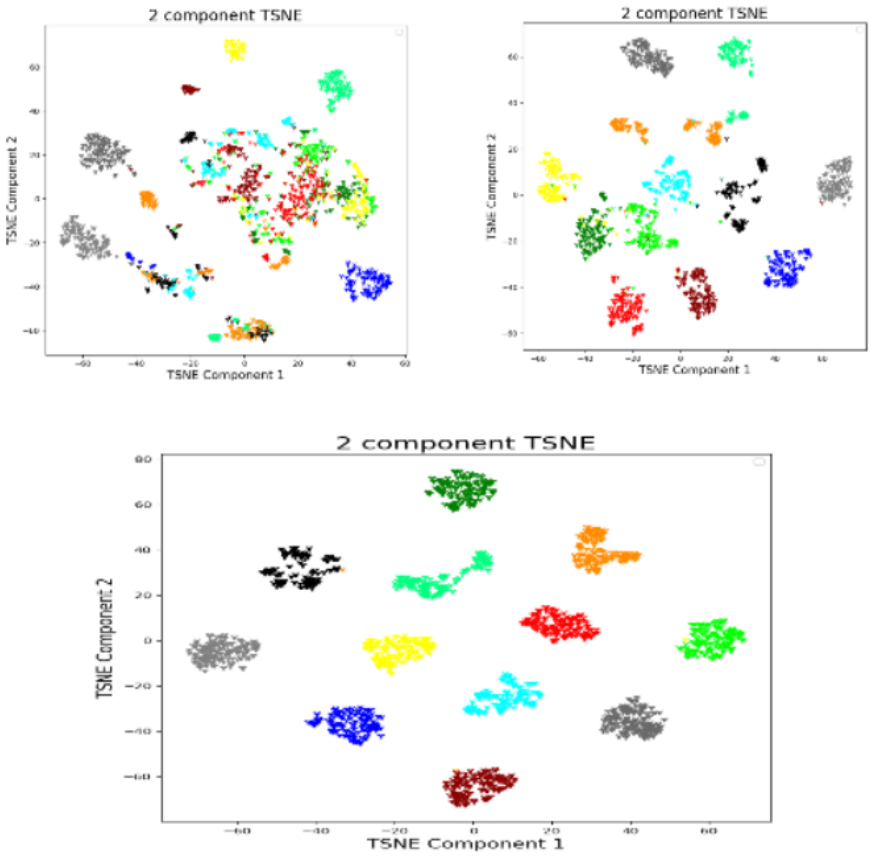

Figure 7.2: Two-dimensional representation using t-SNE of the model results after different number of training epochs. The first plot on the left has been created after 100 epochs, the one on the right after 200 and the bottom one after 400 epochs.

It has to be noted that the classification accuracy increases as the epochs increase, even after the different datasets are separated visually.

Another test that we performed to validate our results, was to plot samples from the test set together with samples from the train set, in order to examine if they are projected at the expected locations in the two-dimensional representation of the embedding's space. As it can be seen in figure 7.3, almost all the test samples were plotted in the right areas.

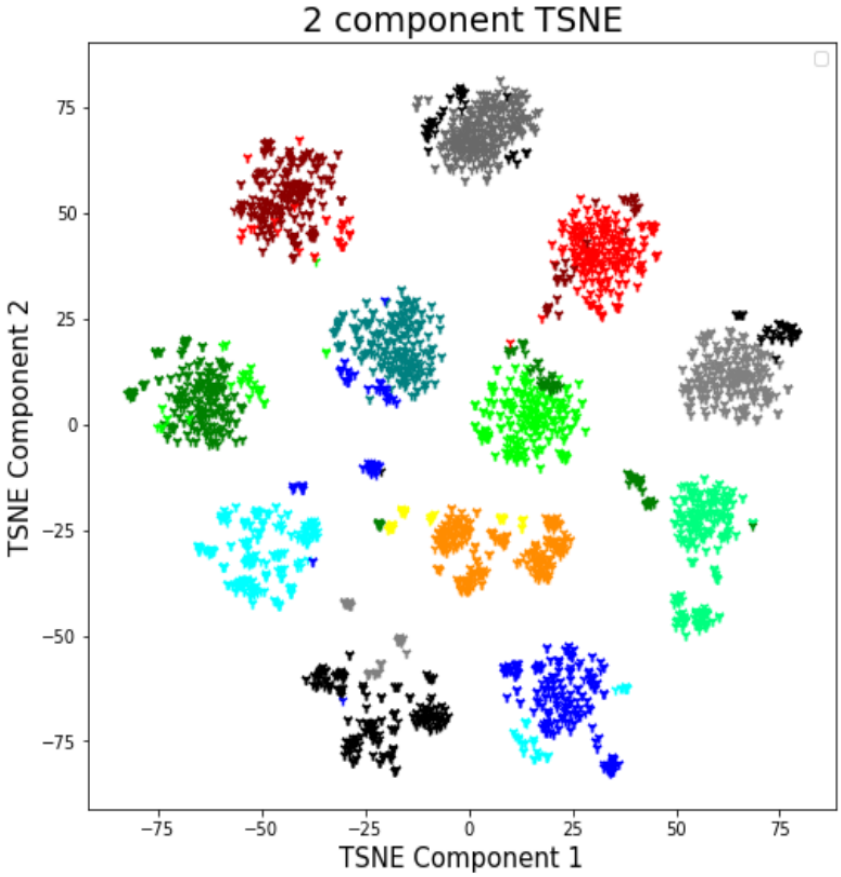

Figure 7.3: t-SNE two-dimensional representation of the embedding's space both for samples from the training set as long as the test set. The test samples of a disease are plotted with a different shade of their disease's chosen color.

7.2 Large-scale example

In this section, a large scale example is presented. A total of 120 datasets with more than 19000 samples were downloaded by the BioDataome database. From every of the 12 different diseases, 10 datasets with at least 60 samples were picked randomly from the database. The diseases as long as the number of samples per disease are presented in table 7.3.

Table 7.3: Diseases and number of samples per disease

Diseases and number of samples per disease	
Disease Name	Samples number
Acute myeloid leukemia	2419
Breast cancer	1062
Chronic lymphocytic leukemia	1246
Chronic obstructive pulmonary disease	1569
Colon cancer	1998
Colorectal cancer	1156
Hepatocellular carcinoma	1209
Lung cancer	1430
Lymphoblastic leukemia	1638
Multiple myeloma	2262
Neuroectodermal tumor	1560
Psoriasis	1885
Total	19434

In table 7.4 some important values about the example are being presented.

Table 7.4: Important values about the example

Important example's values	
Name	Value
Layer 1 size	2048
Layer 2 size	1024
Layer 3 size	256
Studies per batch	114
Samples per study	80
Initial data's dimension	500
Initial learning rate	1e-5
k of kNN	25

In this example, we are examining our algorithm's classification abilities both in terms of what disease the samples came from but also in terms of the particular study that the samples originate from. Following the same procedure as we did in the first example, the classification accuracy of the knn algorithm on the test set, in terms of the disease that the samples came from, increased from 91.7% to 96.9% a relative improvement of 5.7% and for what particular study they originate from, increased from 54.5% to 65.3% a relative improvement of 19.8%.

Also, we want to see how the embedding's dimension affects the classification accuracy. In table 7.5 the knn's results for different embedding dimensions are being presented.

Table 7.5: Effect of embeddings dimension on the classification accuracy of the knn algorithm.

Effect of embeddings dimension		
Layer size	Disease accuracy (%)	Study accuracy (%)
Initial dimension (500)	91.7	54.5
256	96.9	65.3
128	94.7	61.3
64	93.4	60.0
32	88.0	52.4
16	79.1	41.0
8	55.6	23.7

As expected, the accuracy of the knn algorithm on the test set decreases as the embeddings size decreases, but a significant part of the information is preserved even after drastically reducing the embeddings dimension.

In figure 7.4 and 7.5 can be seen a visual representation of our data on the two-dimensional space created using PCA and another one created using t-SNE . Every color represents a different disease and every point a different sample.

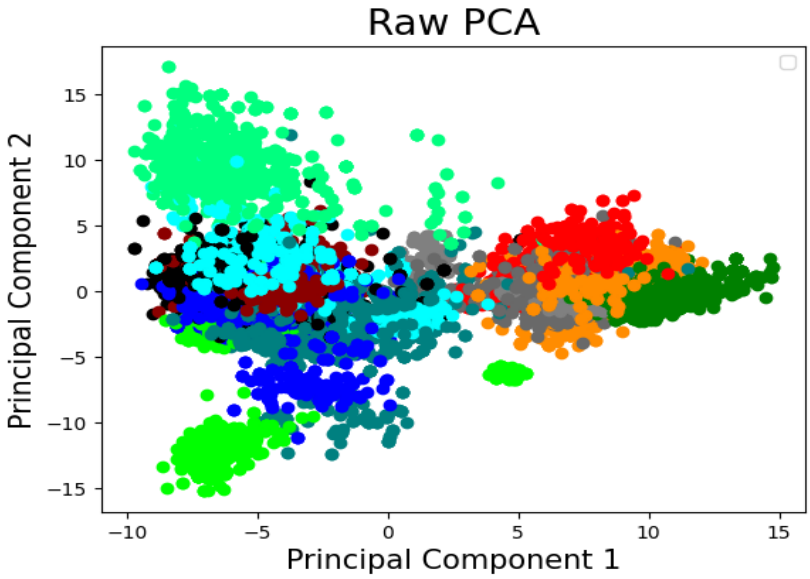

Figure 7.4: Visual representation of the data on the two-dimensional space using PCA.

In this example, for every disease there are 10 different studies. The ideal scenario in this case would be the separation of the individual studies, but also the relative topology in terms of similarity between studies and diseases to be kept. In figure 7.6 the separation progress is presented for different epochs of the training.

In figure 7.8 some of the test samples are plotted along with samples from the training set in order to find out if they are plotted at the expected areas.

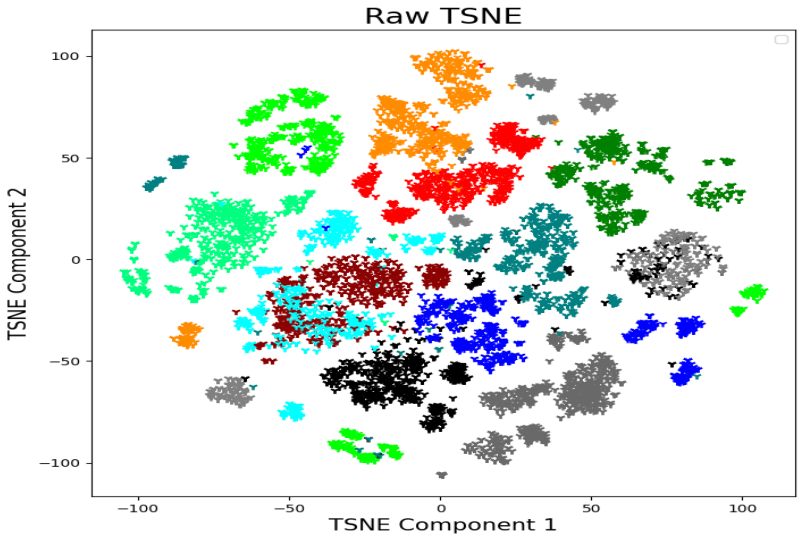

Figure 7.5: Visual representation of the data on the two dimensional space using t-SNE.

It can be seen that as the training progresses, the different studies begin to separate in most cases, but they remain close to the other studies of the same disease (same color). Also, comparing the two-dimensional representations before and after the use of the method, it can be seen that the relative location of the different diseases is retained. Additionally, in figure 7.8 is visible that, the unseen during the training of the network samples of the test set, are plotted at the expected areas in terms of what disease they originate from (similar color) and across the different studies of each disease (different "clouds" of the same color-disease).

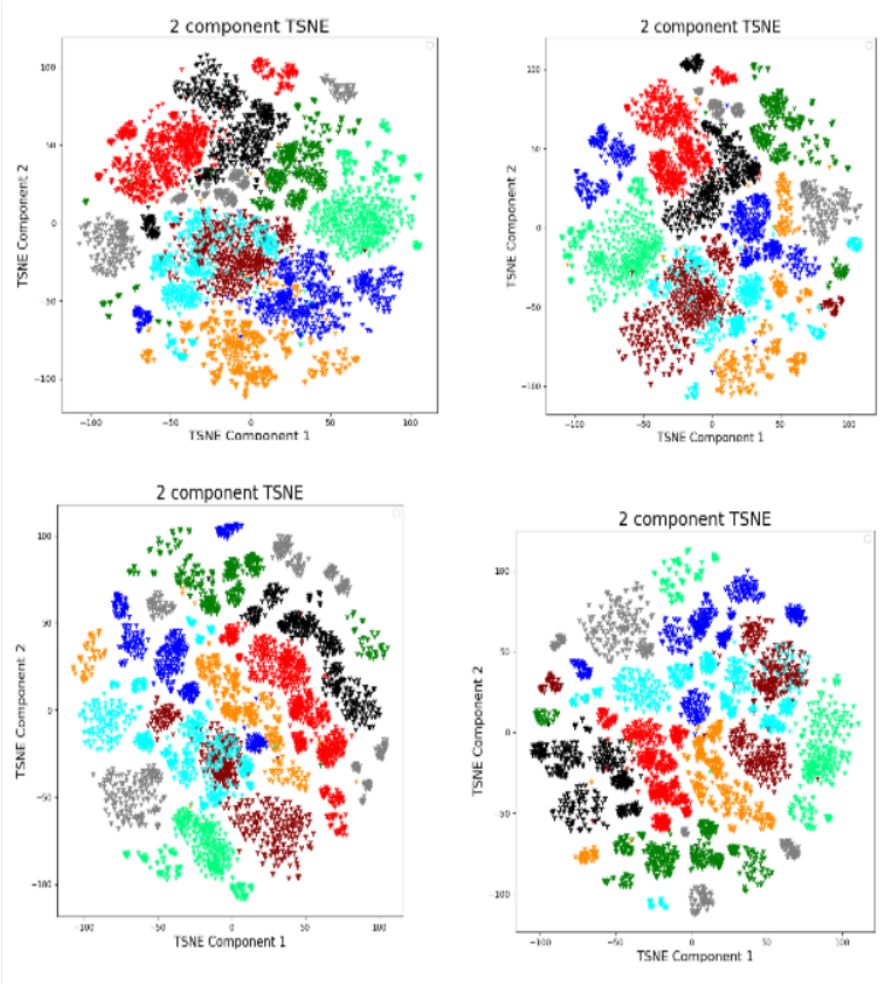

Figure 7.6: Two-dimensional representation using t-SNE of the model results after different number of training epochs. The first plot on the left has been created after 80 epochs, the one on the right after 160 the bottom left one after 320 and the bottom left after 400 epochs.

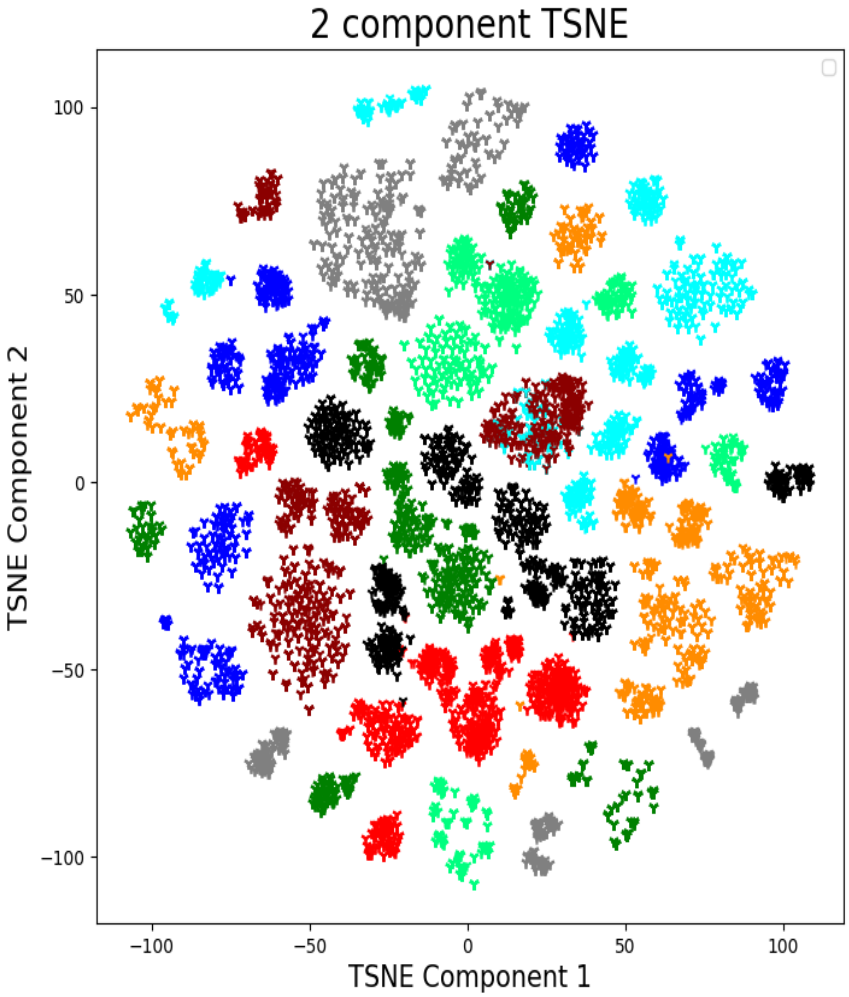

Figure 7.7: Two-dimensional representation using t-SNE of the model results after 800 training iterations.

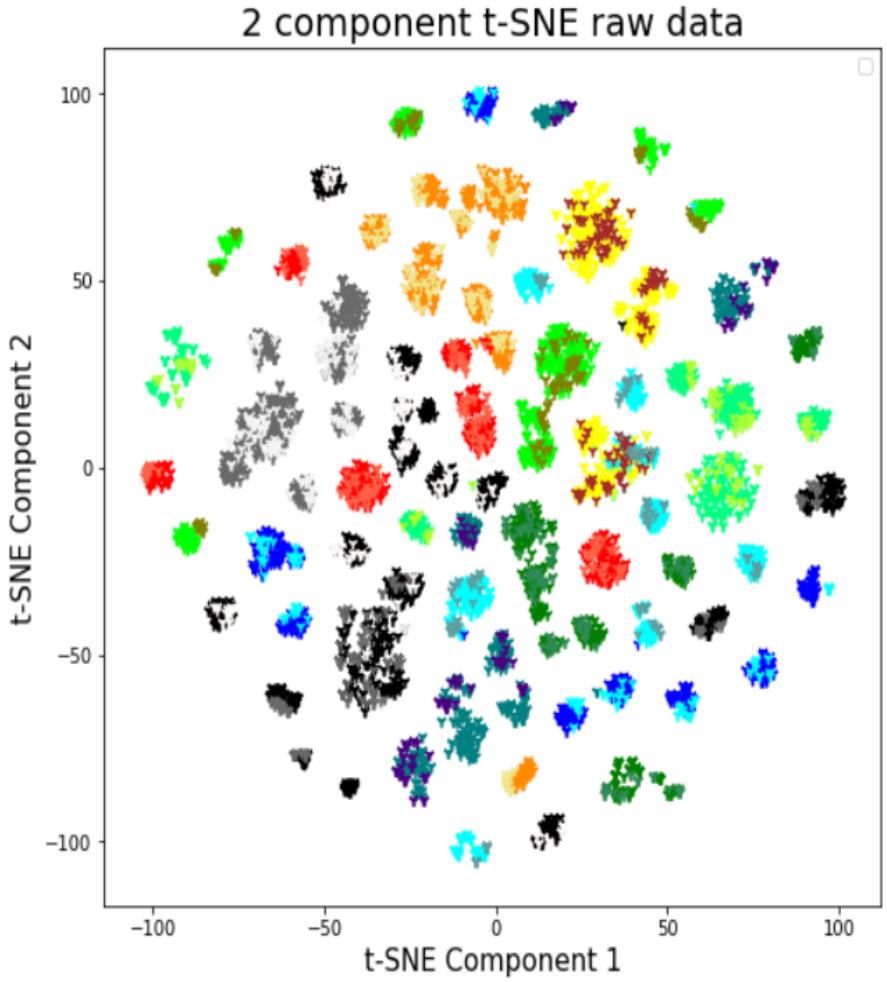

Figure 7.8: t-SNE two-dimensional representation of the embedding space both for samples from the training set as long as the test set. The test samples of a disease are plotted with a different shade of their disease's chosen color or with a similar one.

We also tested our model's ability to generalise to completely unseen studies, from the same diseases that exist in the training set. In order to do that, we create a batch of samples from datasets that were used during the training and also samples from unseen by the model studies. The created batch then passes through the model and the embeddings are created. The batch of the sample's embeddings then is used as training set for the knn algorithm. The test set is compromised of samples from the unseen studies by the model during training, samples that the knn algorithm also hasn't seen during its training. Finally, the accuracy of the knn algorithm on the test set is calculated and compared with the accuracy of the algorithm when it's trained by the raw data (initial dimension). The accuracy of the knn algorithm was improved from 68.7% to 77.0% in terms of what individual study the samples came from, a relative improvement of 12.1% . In terms of what disease the samples came from, the accuracy improved from 92.2% to 97.1% a relative improvement of 5.3% .

In the interest of examining the effect of the training set size (in terms of studies number per disease) on our results, we conducted a test which the results of can be seen in table 7.6. We started with an initial number of 24 studies, which means 2 different studies for the 12 diseases. In the first instance, the test set is compromised of samples from those first 24 data sets. Then we increase the number of studies used in the training, keeping the same test set. It can be seen that both in terms of what disease the test samples came from, as long as what particular study they did, the results in terms of accuracy were improved. That's the case "seen study" in table 7.6. In the second case, the test set is a combination of studies that the network did not see during the training, case "unseen study" in table 7.6.

Table 7.6: Effect of the training set size on the knn algorithm's accuracy in both seen and unseen studies by the network.

Effect of the training set size					
Studies number	Init. dim.	24	60	96	114
Disease accuracy %					
Seen study	92.7	94.6	95.3	95.1	95.9
Unseen study	96.3	97.4	97.5	98.8	99.1
Study accuracy %					
Seen study	76.6	82.6	83.8	83.7	84.8
Unseen study	90.1	89.0	91.8	93.7	94.2

It can be seen that in both cases (seen and unseen studies) the knn algoritm's accuracy increases as we add more studies in the training of the network. That happens both in terms of what disease the samples come from, but also in terms of what particular study they originate from. It has to be noted that the results of the "seen study" case and "unseen study" case are not comparable. In the case of "seen study" there are 24 different labels in contrast with the "unseen study" case where there are only 6 different labels.

Chapter 8

Summary

The proposed algorithm has been proved to work after multiple tests. The algorithm was tested both using synthetic data that we produced (which emulate microarray data) and real microarray data. In order to validate our algorithm we tested its classification capabilities using objective measures as well as visual inspection. Specifically, the accuracy of the k-nearest neighbors algorithm which has been used for the classification tests, on unseen by the neural network samples has been improved after the model appliance. The accuracy of the knn algorithm has also been improved for completely unseen studies-datasets by the model. We also carried out several different visual tests that ensured that the method worked properly. For example, we tested if unseen by the model data, are projected at the expected areas of the two dimensional representation of the embeddings space or if similar datasets (perhaps of the same disease) remain close in the embeddings space after the training. Finally, we tested the algorithm's performance on a large scale experiment in order to find out if the method can generalise to more complex tasks. In this experiment more than 19.000 samples from 120 different studies and 12 diseases, have been used for the model training. The method was proven capable of generalising on the larger scale experiment.

Bibliography

[1] *Adam Optimizer*. https://www.geeksforgeeks.org/intuition-of-adam-optimizer/.

[2] Krzywinski M. Altman N. "The curse(s) of dimensionality". In: *Nature Methods* 15 (2018). https://doi.org/10.1038/s41592-018-0019-x, pp. 399–400.

[3] Guillermo de Anda-Jáuregui and Enrique Hernández-Lemus. "Computational Oncology in the Multi-Omics Era: State of the Art". In: *Frontiers in Oncology* 10 (Apr. 2020). figure 8 , figure 3. DOI: 10.3389/fonc.2020.00423.

[4] *Artificial neural network*. https://en.wikipedia.org/wiki/Artificial_neural_network.

[5] *Autoencoders*. https://en.wikipedia.org/wiki/Autoencoder.

[6] Yoshua Bengio. "Learning Deep Architectures for AI". In: *Foundations and Trends in Machine Learning* 2.8 (2009). doi:10.1561/2200000006, pp. 1795–1797.

[7] Jason Brownlee. "A Gentle Introduction to Cross-Entropy for Machine Learning". In: *Probability* (Oct 21, 2019). https://machinelearningmastery.com/cross-entropy-for-machine-learning/.

[8] Jason Brownlee. "A Gentle Introduction to Dropout for Regularizing Deep Neural Networks". In: *Deep Learning Performance* (Dec 3, 2018). https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks.

[9] Jason Brownlee. "A Gentle Introduction to the Rectified Linear Unit (ReLU)". In: *Deep Learning Performance* (Jan,2019). https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/.

[10] Julian Lewis Bruce Alberts Alexander Johnson. *Molecular Biology of the Cell, 4th edition*. New York: Garland Science, 2002.

[11] Y Chen et al. "Gene expression inference with deep learning". In: *Bioinformatics* 32.12 (2016), pp. 1832–1839.

[12] Ueli Meier Ciresan Dan. "Flexible, High Performance Convolutional Neural Networks for Image Classification". In: *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence Volume Two* 2 (2011), pp. 1237–1242.

[13] Robert Wilby Cristian W. Dawson. "An artificial neural network approach to rainfall-runoff modelling". In: *Hydrological Sciences Journal* 43 (1998). https://doi.org/10.1016/j.juro.2007.05.122, pp. 47–66.

[14] *Deep learning*. https://en.wikipedia.org/wiki/Deep_learning.

[15] Jr.; Joseph F Kett; James Trefil E D Hirsch. *The new dictionary of cultural literacy*. Boston : Houghton Mifflin, 2002.

[16] Stefan Elfwing, Eiji Uchibe, and Kenji Doya. *Unbounded Output Networks for Classification*. figure 1. July 2018.

[17] S. Couce F. Marmiesse A. Gouveia. "NGS Technologies as a Turning Point in Rare Disease Research , Diagnosis and Treatment". In: *Current Medicinal Chemistry* 25.3 (2018), pp. 404–432.

[18] *GeneChip® Human Genome U133 Plus 2.0 Array*. https://www.affymetrix.com/support/technical/datasheets/human_datasheet.pdf.

[19] *Genes*. https://en.wikipedia.org/wiki/Gene.

[20] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. https://doi.org/10.1186/s13059-019-1689-0. MIT Press, 2016.

[21] *GPL570*. https://www.bumc.bu.edu/microarray/services/affymetrix-genechip/.

[22] Ayush Gupta. "A Comprehensive Guide on Deep Learning Optimizers". In: *www.analyticsvidhya.com* (Oct,2021). https://www.analyticsvidhya.com/blog/2021/10/a-comprehensive-guide-on-deep-learning-optimizers/.

[23] Larry Hardesty. "Explained: Neural networks". In: *MIT News Office* (). https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414.

[24] Zemel R. S. Hinton G. E. "Autoencoders, minimum description length and Helmholtz free energy". In: *Advances in neural information processing systems* 6 (1994), pp. 3–10.

[25] Jake Hoare. "How t-SNE works and Dimensionality Reduction". In: *www.displayr.com* (). https://www.displayr.com/using-t-sne-to-visualize-data-before-prediction.

[26] Chunming Xu Scott A. Jackson. "Machine learning and complex biological data". In: *Genome Biology volume* 20.76 (2019). https://doi.org/10.1186/s13059-019-1689-0, pp. 115–133.

[27] Héctor Corrada Bravo Jeffrey T. Leek Robert B. Scharpf. "Tackling the widespread and critical impact of batch effects in high-throughput data". In: *Nature Reviews Genetics* 11 (2010). https://doi.org/10.1038/nrg2825, pp. 733–739.

[28] *K-Nearest Neighbors Algorithm*. https://www.ibm.com/topics/knn.

[29] Mac Namee Kelleher John D. "A Walkthrough of Convolutional Neural Network – Hyperparameter Tuning". In: *Towards Data Science* (2020). https://towardsdatascience.com/a-walkthrough-of-convolutional-neural-network-7f474f91d7bd.

[30] Mac Namee Kelleher John D. "Fundamentals of machine learning for predictive data analytics: algorithms, worked examples, and case studies". In: (2020). ISBN978-0-262-36110-1.OCLC1162184998, pp. 7–8.

[31] Mark A. Kramer. "Nonlinear principal component analysis using autoassociative neural networks". In: *AIChE Journal* 37 (1991). doi:10.1002/aic.690370209, pp. 233–243.

[32] K Lakiotaki et al. "BioDataome: a collection of uniformly preprocessed and automatically annotated datasets for data-driven biology". In: *Database* (2018).

[33] *Loss functions*. https://theanets.readthedocs.io/en/stable/api/losses.html.

[34] Conor M. "Machine learning fundamentals (I): Cost functions and gradient descent". In: *towardsdatascience.com* (Nov 27,2017). https://towardsdatascience.com/machine-learning-fundamentals-via-linear-regression-41a5d11f5220.

[35] T. M. MITCHELL M. I. JORDAN. "Machine learning: Trends, perspectives, and prospects". In: *Science* 349.6245 (2015). DOI:10.1126/science.aaa8415, pp. 255–260.

[36] James W.F. Catto Maysam F. Abbod. "Application of Artificial Intelligence to the Management of Urological Cancer". In: *The Journal of Urology* (). https://doi.org/10.1016/j.juro.2007.05.122.

[37] *Mean squared error*. https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/loss-functions/mean-squared-error.

[38] *Microaarays*. https://www.nature.com/scitable/definition/microarray-202/.

[39] S Min, B Lee, and S Yoon. "Deep learning in bioinformatics". In: *Briefings in Bioinformatics* 18.5 (2017), pp. 851–869.

[40] Mariana Hagberg Niklas Markus Gericke. "Definition of historical models of gene function and their relation to students' understanding of genetics". In: *Science Education* 16 (2007), pp. 849–881.

[41] Varun Kumar Ojha, Ajith Abraham, and Václav Snášel. "Metaheuristic design of feedforward neural networks: A review of two decades of research". In: *Engineering Applications of Artificial Intelligence* 60 (2017), pp. 97–116. ISSN: 0952-1976. DOI: https://doi.org/10.1016/j.engappai.2017.01.013. URL: https://www.sciencedirect.com/science/article/pii/S0952197617300234.

[42] Lakshmi Panneerselvam. "Activation Functions and their Derivatives – A Quick Complete Guide". In: *www.analyticsvidhya.com* (Apr,2021). https://www.analyticsvidhya.com/blog/2021/04/activation-functions-and-their-derivatives-a-quick-complete-guide/.

[43] Yannis Pantazis et al. *Latent Feature Representations for Human Gene Expression Data Improve Phenotypic Predictions*. Oct. 2020. DOI: 10.1101/2020.10.15.340802.

[44] *PCA*. https://programmathically.com/principal-components-analysis-explained-for-dummies.

[45] Rukshan Pramoditha. "The Concept of Artificial Neurons (Perceptrons) in Neural Networks". In: *towardsdatascience.com* (Dec. 2021). https://towardsdatascience.com/the-concept-of-artificial-neurons-perceptrons-in-neural-networks-fab22249cbfc.

[46] *Principal component analysis*. https://en.wikipedia.org/wiki/Principal_component_analysis.

[47] Jeyakumar Natarajan Saravanakumar Selvaraj. "Microarray Data Analysis and Mining Tools". In: *Bioinformation* 6.3 (2011), pp. 95–99.

[48] Nirav Vasant Shah. "Artificial Neural Network and Data-driven techniques: Scientific Computing in the era of emerging technologies". In: (Nov. 2020). https://www.romsoc.eu/artificial-neural-network-and-data-driven-techniques-scientific-computing-in-the-era-of-emerging-technologies/.

[49] Syariful Shamsudin. "The Development of Neural Network Based System Identification and Adaptive Flight Control for an Autonomous Helicopter System". figure 2.6. PhD thesis. Aug. 2013. DOI: 10.13140/RG.2.1.1342.3761.

[50] S. Sra, S. Nowozin, and S.J. Wright. *Optimization for Machine Learning*. Neural information processing series. MIT Press, 2012. ISBN: 9780262016469. URL: https://books.google.gr/books?id=JPQx7s2L1A8C.

[51] Kenji Doya Stefan Elfwing Eiji Uchibe. "Sigmoid-Weighted Linear Units for Neural Network Function Approximation in Reinforcement Learning". In: *arXiv:1702.03118* (). https://doi.org/10.48550/arXiv.1702.03118.

[52] *Stochastic Gradient Descent*. https://www.engati.com/glossary/stochastic-gradient-descent.

[53] Li Wan et al. “Generalized End-to-End Loss for Speaker Verification”. In: (2017). https://arxiv.org/abs/1710.10467.

[54] Zihao Wang et al. *Economic Recession Prediction Using Deep Neural Network*. figure 3. July 2021.

[55] Diederik P Welling Max; Kingma. “An Introduction to Variational Autoencoders”. In: *Foundations and Trends in Machine Learning* 12 (2019). arXiv:1906.02691.Bibcode:2019arXiv190602691K.doi:10.1561/2200000056.S2CID174802445., pp. 307–392.

[56] Daniel B. Forger William Gilpin Yitong Huang. “Learning dynamics from large biological data sets: Machine learning meets systems biology”. In: *Current Opinion in Systems Biology* 22 (2020), pp. 1–7. DOI: https://doi.org/10.1016/j.coisb.2020.07.009. URL: https://www.sciencedirect.com/science/article/pii/S2452310020300147.

[57] Andre Ye. “Swish: Booting ReLU from the Activation Function Throne”. In: *Towards Data Science* (Mar 3,2020). https://towardsdatascience.com/swish-booting-relu-from-the-activation-function-throne-78f87e5ab6eb.