TU Delft

Bachelor Project

Deep Learning and identification of Cancer Related Sub-networks

Paul Bakker

Berend Ottervanger

Sam Smulders

Coach
Amin Allahyar, Bioinformatics group, TU Delft

Client
Prof. Marcel Reinders, Bioinformatics group, TU Delft

Bachelor Project Coordinator
Otto Visser, Distributed Systems group, TU Delft

June 29, 2016

Abstract

The breast cancer survival rate has improved significantly between 1975 and 2003. The primary improvement in treatment is due to subtyping, where the high complexity cancers are divided in types of cancer and subgroups. In a study by Dr. Parker it was found that by looking at mRNA only, the intrinsic subtypes of breast cancer could be identified. In our work we will create images out of gene expressions, which we use to train a convolutional neural network to classify the created images on cancer subgroup.

This report describes an analysis of using convolutional neural networks on images created using the correlations between gene expressions. We focused on using deep learning to identify cancer related gene-gene interaction sub-networks. Prior research has shown that using gene expression assays can be used to improve classification of subtypes of cancer [1]. We created two dimensional images, preserving the correlations between genes as distances in the image and used convolutional neural networks to classify the sub-types. These networks can do object recognition, and in the case of our images the object could be build up of sub-networks of genes.

Preface

This report was written as our bachelor project as part of the bachelor in Computer Science at TU Delft. We chose this project out of our interest in deep learning and in the current developments in the usage of DNA in the medical sciences. In this project we looked into a method proposed by our client, to see if it could work, or not. This method should be able to classify cancer subtypes, or more specifically, breast cancer subtypes. First we developed this method, assembling all required modules. After that we tested the method, and tried to explain the results, coming to a conclusion about the viability of the method.

This report was written to explain our research and our conclusions. During this project we have learned about different neural network types and their parameters, about DNA, their expressions and how these are currently observed in research as well as in health-care. We also had to learn more about probabilities and statistics to analyse and gather our data, so that we could make meaningful conclusions. We also had to change how we thought to come up with hypothesis and experiments, for which we would like to thank our coach for supporting us.

We would also like to thank our coach Amin Allahyar for his continuous support and enthusiasm during this project as our coach. He challenged us to come up with new ideas, and helped us focus on the real questions. When we came up with ideas he gave us well founded feedback and guidance during this project. We would also like our thank Professor Marcel Reinders for his contribution, who took over the roll as our client when our original client, Jeroen de Ridder, was no longer involved in the project.

Summary

Cancer treatment is currently based on clinical parameters and pathological markers which dont capture all the complexity needed for giving treatment based on the subtype of cancer. Subtyping based on multi-gene expression assays have previously been shown to be more precise than pathology based techniques for subtyping. Even with this fact it is not widely used with low availability and high cost being the main causes. Our client requested us to create images of gene expression assays using the correlation between the gene expressions and to use these images as input for a convolutional neural network to classify the different breast cancer subtypes. With the expectation that this could then be used to detect malfunctioning gene interaction networks causing the cancer.

The core challenge of this project was using gene expression correlations to create an image interpretable by a convolutional neural network. Given that the results are unknown not all requirements could be determined at the start of the project. We therefore used agile project management which means evolutionary development and early delivery. It also encourages rapid and flexible response to change. During the research phase we looked at tools that are easy to set up to be able to spend as much time explaining the results and providing foundation supporting these claims.

Given the many parameters of a convolutional neural network multiple configurations have been tested. Therefore we developed an application that could run various configurations multiple times with different initialisations to test the accuracy of classification. These results could be expanded upon by adding more configurations without having to retrain the old configurations. We used this application to test the requested method, but it can be used in other projects to test various configurations of neural networks on other data sets. To test the intuition of the image creation methods we developed tests for determining the effect of integrating correlated gene expressions. We also determined the quality of the images of the proposed image creation algorithms. With the developed application and the two image creation algorithms we tested various configurations of convolutional neural networks to find the most promising meta parameters.

Contents

1 Introduction

1.1 Implications of breast cancer

Breast cancer is the most common cancer for women with as many as one in eight women being diagnosed with breast cancer in the United States [2]. Breast cancer, as its name suggests, is cancer that develops in the breast tissue. Many risk factors are associated with the disease including: age, sex, lifestyle such as smoking or exercise and genetics. Thanks to substantial advances that are made in treatment of this disease, there is a substantial improvement in survival rate of these patients. More specifically, the increase of the survival rate from 74.8% in 1975 trough 1977 to 90.3% in 2003 [3], but 450.000 people are still estimated to die from breast cancer annually worldwide [4].

1.2 Subtyping of breast cancer

The primary improvement in the understanding of breast cancer and the resulting improvement in treatment is due to subtyping. Subtyping looks at the high complexity of the cancer and divides the type of cancer (e.g. breast cancer) in subgroups to represent the differences between the groups. These substypes were found to have significantly different incidence, risk factors, prognoses and chemotherapy intensity needed [5][6]. In a study by Parker et al. it was found that by looking at mRNA only, the intrinsic subtypes of breast cancer could be identified. In other words the diversity of the cancer could be determined by looking at the gene expressions [6][7]. Improving the treatment of breast cancer can be done by specializing the treatment based on the specific subtype given the difference discussed. This is called personalized treatment and is based on the understanding of the different aspects of the subtypes.

1.3 Problems in utilizing subtyping

In a pioneering study, Prat et al. [7] look at the effectiveness of current subtyping methodologies and explains how these approaches fails to address the problem of determining the correct subtype. Treatment is currently based on clinical parameters and pathological markers which don't capture all the complexity needed for giving the most effective treatment. Subtyping based on multi-gene expression assays was found to be more precise than pathology based techniques for subtyping [1]. Even with this fact it is not widely used because of low availability and high cost.

1.4 Machine learning

Machine learning is a subfield in computer science which focuses on pattern recognition and artificial intelligence. In recent years the latest developments in this field which gained popularity are focused on Deep Learning. Deep learning methods use multiple processing steps to create high level abstractions of the input data performing successive non-linear transformations on it. Deep learning is already used for many purposes such as handwritten-text recognition, face recognition and transcriptome data analysis (the analysis of gene expression).

Machine learning comes in two varieties: supervised and unsupervised learning. Supervised learning requires input data which is labeled, whereas unsupervised learning does not. Supervised learning techniques try to identify patterns in the data (i.e. features) to explain those labels. After training, a supervised learning algorithm can be used to classify data for which the labels are unknown. Unsupervised learning techniques identify inherent structures in the data. They form representations of the data, this can be thought of as clustering. These clusters can then be interpreted by looking at their features. Classifiers can be constructed when combined with a supervised learning algorithm, using a small amount of labeled data and an extra training round.

1.5 Proposed approach

Our client requested to find out if gene expression correlations could be used to classify subtypes using a convolutional layer. To do this we first represented the gene expressions in a way a convolutional neural network can understand by creating two and one dimensional images to use as input for these networks. We proposed two different algorithms to represent these correlations as a matrix, IME and Clustering. The intuition of the images was putting correlated gene expression next to each other to improve classification. To substantiate this intuition we proposed a test to conclude that using the average of correlated genes compared to a single gene expression increased both the stability of accuracy of classification as well as accuracy of classification itself. To reduce the high-dimensionality of the pairwise correlations of the gene expressions we used the t-SNE algorithm, which will create a two dimensional representation of the correlation between gene expressions. We expect that by creating images this way the convolutional neural network will be more capable of finding useful patterns in the images.

We proposed and developed an application which can automatically run multiple network models on different images from all available images and network configurations. The evaluation of the produced networks are then saved to the disk, which are later collected to generate the results. In order to determine the better convolutional neural network configurations, we used Student's t-test to compare the accuracy of each of the convolutional neural networks. The set of selected genes in this research is not based on prior knowledge of subgroups that are linked to the particular cancer, but just based on the genes with the highest standard deviation. This means that these techniques are more generally applicable than just to breast cancer subtyping. This was required since the goal is to eventually be able to detect the malfunctioning gene interaction networks causing the cancer. The overall structure is given in figure 1.

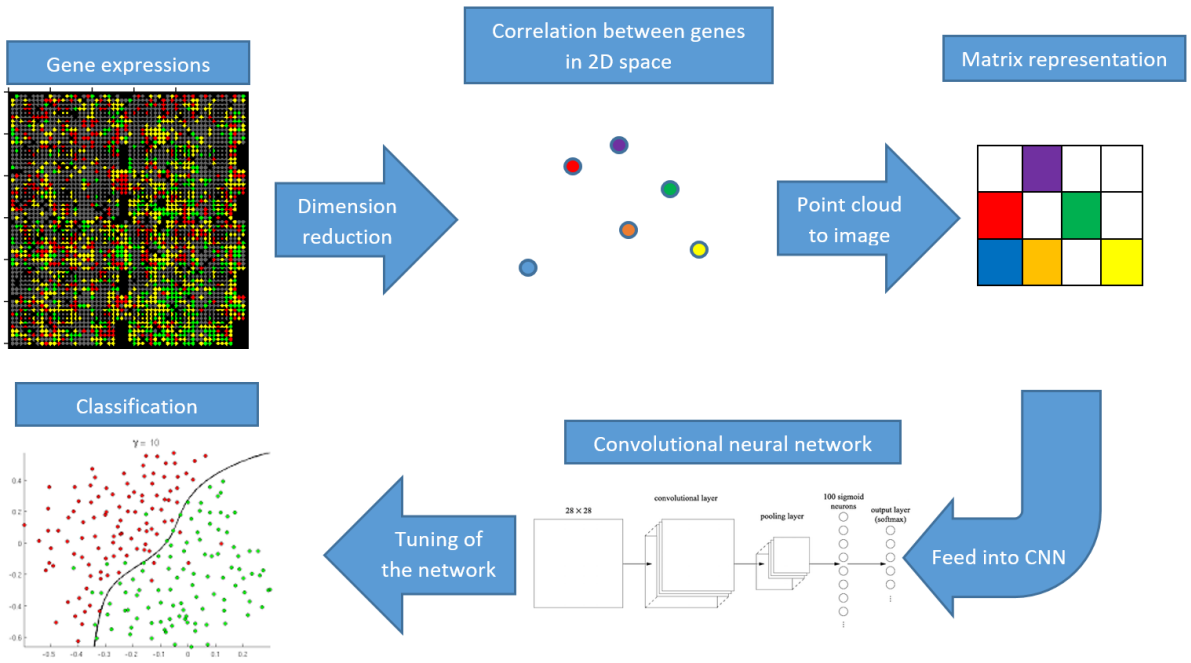


Figure 1: Structure of the total project

1.6 Structure of this report

In the following section is the the overview of the project given, including a description of the project, the plans for quality control, the software development methodology and the requirements defined during the project.

Next we will discuss the methods we used to express the correlation in section 3, the two proposed algorithms for creating images, the proposed application and the baseline neural network where we wish to improve upon. In this section we will also conclude that the intuition of putting correlated gene expressions together increases both the stability of accuracy of classification as well as accuracy of classification itself. For the two proposed algorithms for creating images we implemented a method for comparison to determine how well these perform. We end this section by creating a baseline network configuration, to compare other configurations to later on.

In section 4 we report all the results of the different configurations of convolutional neural networks as well as a linear neural network. We will discuss the results found and look at both the different configurations of the neural network as well as the difference in using the two proposed algorithms for creating images to random generated images. We will discuss why there is no improvement when using a convolutional layer with a proposed test to confirm or disprove the hypothesis.

In section 5 we will give a summary and explanations of all the results and explain our findings. Using our conclusions in this section, we will discuss possible directions for future work in the final section.

2 Project

In this section we explain how this project was addressed. In section 2.1 we describe the project as was discussed with our client, describing the expectations of the client and we describe the success criteria. The nature of the project what follows out of this section influenced the software development method used, which is discussed in section 2.2, which influenced the way we set requirements, which we discuss in section 2.3. After that we describe in section 2.5 how we managed our project and what we did to maintain a certain level of code quality, which makes the developed software during this project open for future usage.

2.1 Project Description

In this project the client proposed a method to detect cancer related gene-gene interaction subnetworks, and asked us, the project team, to test this method and show if this method does or doesn't work. This method is described briefly in section 1.5, and the components of this method are described in detail in section 3. We divided this project into two parts; the first part where we setup the requested method, and the second where we test this method and come up with hypothesis and experiments to conclude if this method does or doesn't work.

Some parts of this method do already exists. This includes the t-SNE, the computation of correlations and the convolutional neural networks. But there are also some holes in the proposed method, which are asked to us to be filled in. This includes the method of converting point clouds produced by t-SNE to matrices or images, and a convolutional neural network configuration that can be used on the generated images. This is also the part of the project where we proposed and implemented new software solutions for the specific problems.

In the second part we will test the performance of this method and try to come up with hypothesis to explain these results. After which we come up with experiments to test these hypotheses. By this progress we will try to give a conclusion, with as goal to explaining why this method does or doesn't work. The client's proposed method should be tested and the performance of this method should be explained. This project is successful when both these parts are addressed.

2.2 Software development methodology

During this project we used an adaptation of the Agile software development. This adaptation was required because of the nature of this project, which is more a research project then a software development project. After each client and/or coach meeting, which we had once a week, we applied the new feedback and requests to our backlog and setting up a new sprint. Each sprint contains a number of goals which we would like to accomplish that week so that we could show and discuss our results during the next meeting. We used this approach because our client during this project frequently had new requests. Which due to a relatively small backlog could be addressed. We worked on the TU Delft campus 8 hours each day with communication done verbally.

2.3 Requirements

During the project we could define the following requirements following the MoSCoW structure:

Must have

- A test must be developed to conclude that by averaging correlating genes we could improve the stability of accuracy of classification.
- Correlation between genes must be represented in two dimensional space by t-SNE.
- An algorithm converting the two dimensional representation created by t-SNE into an two dimensional image must be created.
- A baseline neural network must be created to compare any results with.
- Multiple configurations of a convolutional neural network must be compared to the baseline.
- An analysis of the results must be given.
- None of the steps in the process from gene expression to classification may require external knowledge. This includes the selection of genes.

Should have

- The image creation technique should be tested for the ability to preserve correlations between gene expressions as distance.
- An application should be developed to test multiple configurations of neural networks.
  - The application should accept multiple data-sets to run multiple the configurations on.
  - The application should save the results of the different configurations.
  - New configurations should be able to be added without having to retrain the old configurations.
  - The results should be exported for further analysis.

Could have

- Deep Belief Networks could be tested using our developed application
- Deep Boltzman Machines could be tested using our developed application

Won't have

- The application for testing multiple configurations won't have automatic t-test or effect-size calculation.
- The application won't be a stand-alone .jar

The reason we won't develop automatic t-test or effect-size calculations of the different configurations is that by just returning the results for further analysis in Matlab the application much more analysis can be done than only t-test or effect-size. The implementations of Matlab are also efficient and used throughout the research field. The application can be easily expanded with other neural networks, such as Deep Boltzman Machines and Deep Belief Networks. To make a stand-alone .jar would make these expansions impossible.

2.4 Software Structure

The focus of this project was research. This meant that the software developed served as a tool for our research. This meant it was paramount to set up all necessary code as fast as possible. This was reflected in the requirements by the fact that there were no limitations to the programming language, or libraries used. In order to be as fast as possible we wanted to use a programming language we were all familiar with: *Java*. However, we soon found out that it was easier to use the existing t-SNE implementation in *Matlab*. Therefore we also considered using a *Matlab* library to train the neural networks, to use one programming language for the whole application. Later we decided that using Deep Learning for Java would be faster and easier to get up and running. By this time various algorithms such as IME were already finished.

The requested method by our client, is a deep learning approach. As such, it consists of multiple steps which use data generated by previous processing steps. We implemented this method as a pipeline of independent modules, which can all be run individually. This allows us to switch out modules with relative ease. The pipeline is displayed in the data flow diagram in figure 2, of which some of the components will be discussed in this section.

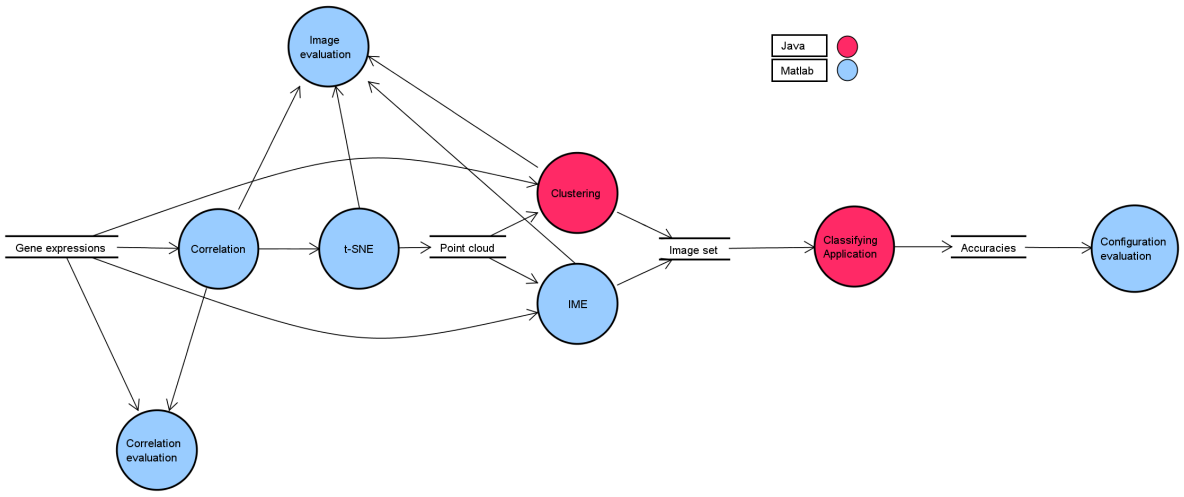


Figure 2: Structure of the pipeline

2.4.1 Image creation

IME was developed within a single *Matlab* function script. The choice for *Matlab* was made because of the fact that *Matlab* works with matrices as its main data structure. The expansion of the matrix in IME was implemented as a concatenation of (sub)matrices. Other programming languages would have required us to build such a complex data structure, and test it ourselves. In section 3.3.1 we will discuss the inner workings of this algorithm.

Clustering uses the implementation of the 'all nearest neighbor' algorithm which we developed in a previous course. This algorithm uses the divide and conquer algorithm design paradigm to return the closest neighbors with the distance for each point. Divide and conquer means that we keep splitting the problem into a smaller sub-problems which are simpler to solve. In this case we will recursively split the point cloud until there are only two point left in each sub-problem. These sub-problems are then combined by looking at a strip with the width being the maximum distance found so far.

This implementation is written in *Java* and the overall Clustering algorithm was therefore chosen also to be written in *Java*. We defined a object *Cluster* with x and y values. This is the mean of the all coordinates of the points in the cluster. A cluster has a list of lower level clusters and when this list is empty the cluster represents a point with the x and y value being the coordinates of the point. The results of the 'all nearest neighbor' algorithm was originally stored in two arrays. But to make sorting of these arrays easier we made an object *Neighbor* containing the points ID, the closest neighbors ID and the distance. This object is a comparable based on the distance for easy sorting. The main class of the Clustering algorithm is *Mapping*, where the point cloud is loaded and clustering of the points using the all nearest neighbor algorithm is done. The results are in this class written to the '.meta', '.lab', '.dat' file. This class can be called with all the necessary parameters using the command line or inside an IDE meaning that no code has to be changed.

2.4.2 Application

Because we where required to get some insight in this method as fast as possible, we wanted to develop an application for automatic testing in a short amount of time, and therefore without too much complexity. For this we used Deeplearning4j as training in this library can be done using the GPU. Another reason is that was relatively easy to understand and use. Given that this library is developed in *Java* we had to use *Java* for further development of this testing application. The detailed implementation with the way we read the input data and the used interfaces can be found in section 3.4.

2.4.3 Different evaluations

Besides the image creation techniques and the application used for training different configurations of a convolutional neural networks we had to evaluate multiple results. These results could be the resulting images, but also the accuracies of the neural networks as well as the proof of the intuition of using correlation to create images. For these test we developed *Matlab* scripts as *Matlab* has many tools built in which are useful for our evaluation. An example is the Student's t-test function and multiple ways to represent the data such as histograms.

2.5 Code quality

In this section we discuss how we maintained a certain level of code quality and how we addressed related flaws which where addressed by inCode and SIG from SIG.eu.

2.5.1 Used tools

One of the methods used to enforce code quality was the usage of checkstyle to find mistakes in the code and lacking documentation. We configured an own checkstyle rule set which addressed and enforced rules that addressed documentation, usage of fields using final on the parameters, required 'this' usage to call fields, and many more details. By doing this we enforces the readability and re usability for later users, using our project.

For code collaboration during this project, as well for version control, we used Git and the Git repository Github. While we used Git to access our repository on Github, we also used the Github web interface to overview and request or accept pull requests from our branches to the master branch. We required all members of the team to agree on a pull request before this was merged to the master, and we commented on the code or pull request. This was done to confirm if the code quality was maintained in the new or adjusted code, and if everything was sufficiently tested.

This was not only done manually, but we also continually and automatically tested the project using Travis and JUnit unit tests. Using tests we prevented several bugs and eased the development. We tested good behavior, as well as bad behavior by catching exceptions. In total, the unit tests written for this project have a test coverage of 92,4%.

These tests can prevent bugs in the software, but can not be used to detect design flaws. To look for these flaws we used inCode. Using this application we noticed a high complexity in the clustering image generation code and a high accessibility of some classes or methods which led to so called feature envy. The high complexity was improved upon by splitting one of the classes and separating the class responsibilities to a single responsibility for each class. The accessibility was reduced by introducing private and protected fields and getters to data classes, reducing its responsibility to only providing the information instead of accessing it.

2.5.2 SIG report

The design flaws where also addressed in the first SIG report analyses on our project, see appendix E. Our project was graded with four out of five stars for an above average maintainability. The highest score wasn't reached because of a lower score in unit size and unit interfacing, thus Dennis Bijlsma.

The unit size was mainly to blame on the method used for the cluster image construction, see section 3.3.3. A large unit size, or in our case large methods, are hard to overview during later development and harder to test. We addressed this problem by refactoring this method to smaller more descriptive methods, as was suggested by SIG. It was also decided to move the 'All nearest neighbor' algorithm to a separate utility class to improve the structure even further.

By unit interfacing is looked at the percentage of code with an above average amount of parameters. A huge amount of parameters indicates a lack of abstraction, and the amount of parameters could lead to confusion by calling these methods. We addressed this by re factoring some methods with a higher amount of parameters. The class Data was explicitly mentioned in the SIG evaluation, using ten parameters in the constructor. The suggested solution by SIG was to group some parameters together as separate classes. An instance of this class is representing a

loaded file, including the gene expression data, labels as well as the meta data. The constructor of this class is also not meant to be used outside of the class itself, but is supposed to be created using a factory method. Because this class is meant for a specific type of data, we decided to create an interface and a loader class for it, and make the constructor of the class itself private. Because of this a future user can use the data class interface to load new data types, but can also use the factory to load the expected data type, without filling in any parameters.

We also decided to rename the class to give it a more descriptive name. The interface is called *GeneExpressionDataBase*, because it is responsible for making data available. The implemented class is calls the *LoadedGeneExpressionDataBase*, because this class makes the data available by loading it from the disk to memory. All non related functions for the providing responsibility, which are all functions to load data into memory, are moved to the Loader class. Finally we added the loader class, called *LoadedGeneExpressionDataBase.Loader*, with only one accessible method for loading a *LoadedGeneExpressionDataBase* instance.

3 Methods

In this section we will discuss the methods developed, which are the parts required for the system described in section 1.5. First we will discuss the data used through this project, and how we limit the size of this data in section 3.1. From the raw data we go to the usage of the correlations between the different gene expressions within the data in section 3.2, which then is used to create a point cloud using t-SNE in section 3.2.2. Then, in section 3.3, we will discuss how this t-SNE point cloud could be used to create images so that the gene expressions on the image close to each other are more correlated then the genes further away. After explaining the progress of generating the images, we propose an automated way of testing configurations in section 3.4, which we then use to set up a baseline network configuration in section 3.5. This baseline will later be used in section 4 to compare other configurations with.

3.1 Data used

The data set used in this project is an aggregation of twelve studies by Staiger et al. [8] to form a data set of 1616 patients. The expression of 12750 genes are contained in the data set for each patient. The data set also contains labels representing the prognoses of the subtype of breast cancer. This label indicates one of the breast cancer subtypes: normal-like, Her2 overexpression, Luminal A, Luminal B and Basal-like. Because these prognoses were not available in all studies, the labels were predicted using PAM50 genes[5].

We chose not to use all the gene expressions for the creation of the two dimensional representation as it would slow our progress down significantly. By limiting the data we could train and test network configurations in minutes, allowing us to test more ideas and to get a feel for the data set. To limit the used data a selection was made by using the 100 genes with the highest standard deviation. These genes included 5 of the PAM50 genes used to generate the label, this might result in unfairly high estimates of classification accuracy. Therefore, we will use a baseline and only look at relative increases and losses in accuracy.

3.2 Using correlation to create image representing gene expressions

In the research report and the introduction we discussed the fact we would like to group correlated genes together for identification of cancer subtypes. The intuition being that putting correlated genes together in the two dimensional representation will create bigger patterns inside the data which machine learning can find more easily, but also reduce the noise that comes with uninteresting mutations. Correlated genes are expected to behave together which means that if these genes are together a considerable part of the two dimensional representation will change between the different subtypes.

We used the *Matlab* function *corr(X)* found in the research phase, which returns a p-by-p matrix containing the pairwise linear correlation coefficient between each pair of columns in the n-by-p matrix X. This is more often called the pairwise similarity matrix which is a efficient way to represent the correlation between the genes. We will first prove that putting correlated genes together in the two dimensional representation will improve the performance by showing that by using the average of four correlated genes we get a more stable accuracy of classification and a improvement in the accuracy itself. We will then look into t-SNE's ability to preserve high-dimensionally correlated genes and explain why we use this technique in the process of expressing the correlation.

3.2.1 Improving stability and accuracy by using average activation of correlated genes

To prove that putting the genes with high correlation next to each other in a matrix can improve the accuracy of a neural network we will select multiple times single gene using a unique seed with its three most correlated genes and take the average activation A_{avg} of these four genes and the activation A_g of that single gene. We then select a subset of A_g and A_{avg} and determine a threshold to make a classification. The optimal threshold will be the one with the highest accuracy of the group of patients. This threshold will be computed by taking the minimum and maximum of A_g in the case of the individual gene and the minimum and maximum of A_{avg} in the case of the average gene expressions. To determine the threshold with the highest accuracy we simply 'walk' with a small step-size from the minimum to the maximum.

Given that with a threshold we could only distinguish between two labels, we choose a random label and its negation, which was label 3 or not label 3 in our case. This process is depicted in figure 3. We will take all these accuracies of these subsets and look at the standard deviation of the total distribution of accuracies. This is done for multiple genes to get two sets of standard deviations, one of the standard deviation of accuracy using a single gene expression and the other using the average of correlated gene expressions, were a lower average standard deviation means a more stable accuracy of classification. To show that using the average activation of correlated genes decreases the standard deviation we use a one tailed t-test, in our case on the right tail. For the accuracy itself we will use the one tailed t-test on the left tailed as we would like to increase the accuracy.

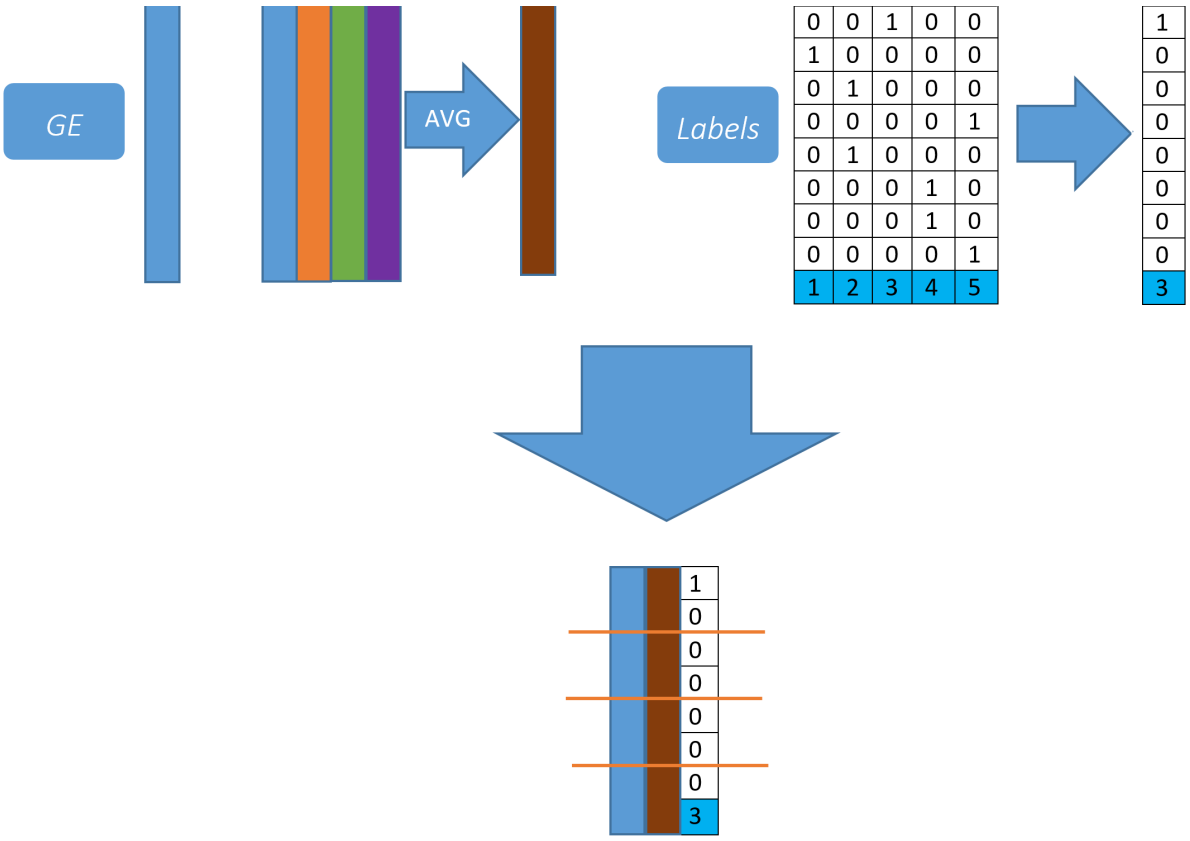


Figure 3: Representation of the selection of a single gene and the integration of the four most correlated genes with a chosen label

We did this for 100 genes each with 101 groups of 16 patients to get two distributions, one containing the standard deviation of A_g for each group, the other the standard deviation of A_{avg} for each group. Using t-test we found that the two distributions where in fact significantly different.

This total algorithm, including the t-test, we ran 100 times (runs) to be certain that every time we created the distributions there was an improvement in both stability and accuracy of classification. The pseudo-code of our test can be found in Appendix B.

Next we will give a overview of the found results starting with the results of the 100 one tailed t-test we ran. These tests will look at the two distributions of standard deviation and proving that the mean standard deviation of distribution of standard deviation of the single gene is significantly greater than the one of the average of four correlated genes. The array h_{array} contained only ones meaning that the null hypothesis of the t-test was rejected every time, from which we can conclude that the average standard deviation, when using a single gene expression, was significantly greater than the average standard deviation when using the average of four correlated gene expressions.

The p-values of the t-test ware given in the array p_{array} and which is represented in figure 4. This boxplot shows the p-value expressed as $-log_{10}(p)$ where a higher means a greater significance. To get the null hypothesis rejected a minimum of $-log_{10}(0.05) \approx 1.3$ has to be achieved, which was always the case as can be seen from figure 4. This indicates that the standard deviation of accuracy using an average activation of correlated genes is smaller than when using the activation of a single gene, meaning an improvement in stability of classification.

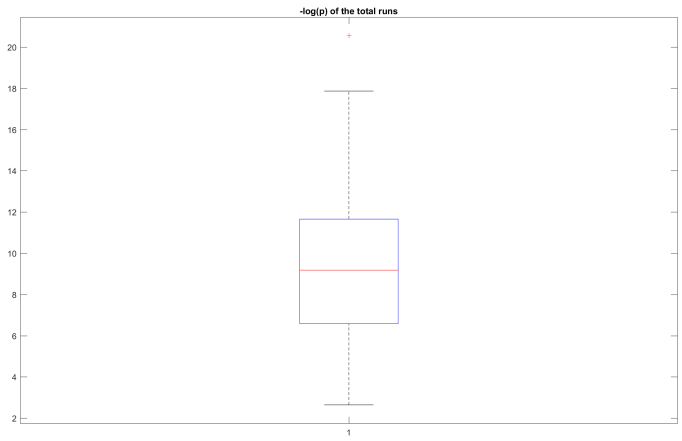


Figure 4: P-values of 100 right tailed t-tests looking at the standard deviation

Next we looked at the average standard deviation of the 100 runs, which we did to show the effect of using an average activation of correlated genes. The total arrays of $totalMean_1$ and $totalMean_{avg}$ are represented in figure 5. Here we see that the distribution of standard deviation of the average is closer to zero meaning an increase in stability in classification and the total average of all these 100 distribution of standard deviation of one genes being 8.94% while the average of the standard deviation of the average of four genes being 8.69%.

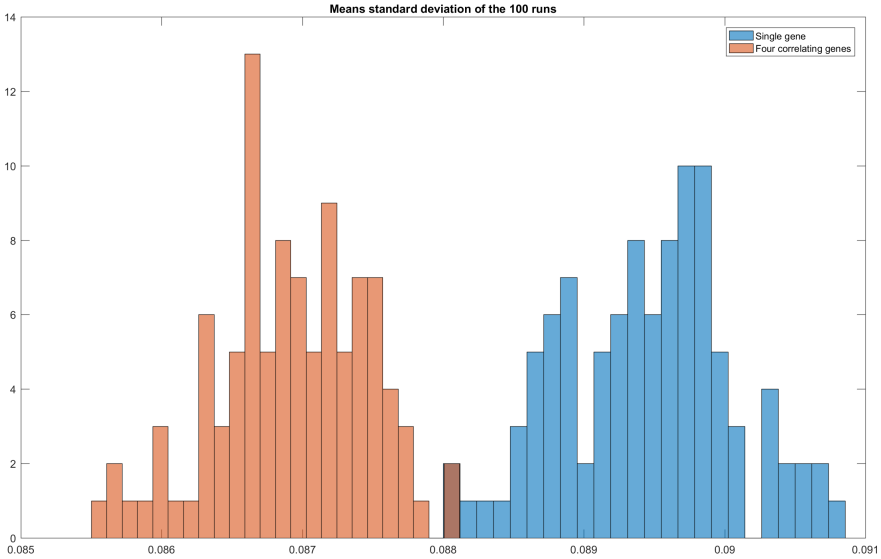


Figure 5: Two distributions of the average standard deviation

We also determined if the accuracies did significantly increase when taking the average of correlated gene expressions improving the classification itself. Here we ran again a t-test each run, but now one tailed to conclude that the the accuracy, when using the average activation of four correlated genes, is significantly improved, when compared to using the activation of a single gene. The result can be found in figure 6. The total accuracies were contained in $totalAcc_1$ and $totalAcc_{avg}$ and are represented in figure 7. Here we can see that by using the average activation of four correlated gene expressions we could improve the accuracy with the average accuracy being or 73.97% instead of the average activation 71.25% with the use of a single gene expression.

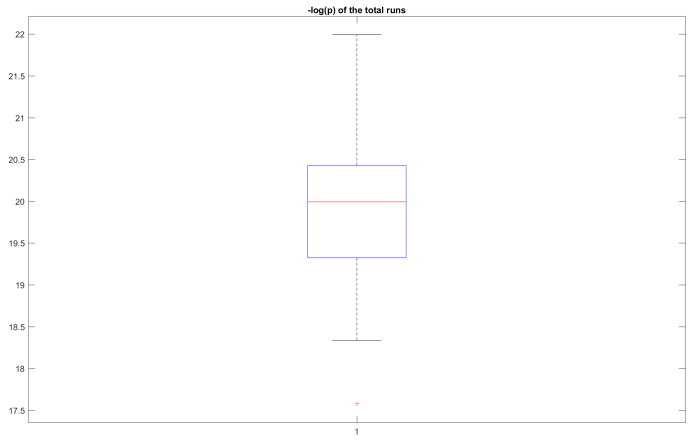


Figure 6: P-values of 100 left tailed t-tests looking at the accuracy

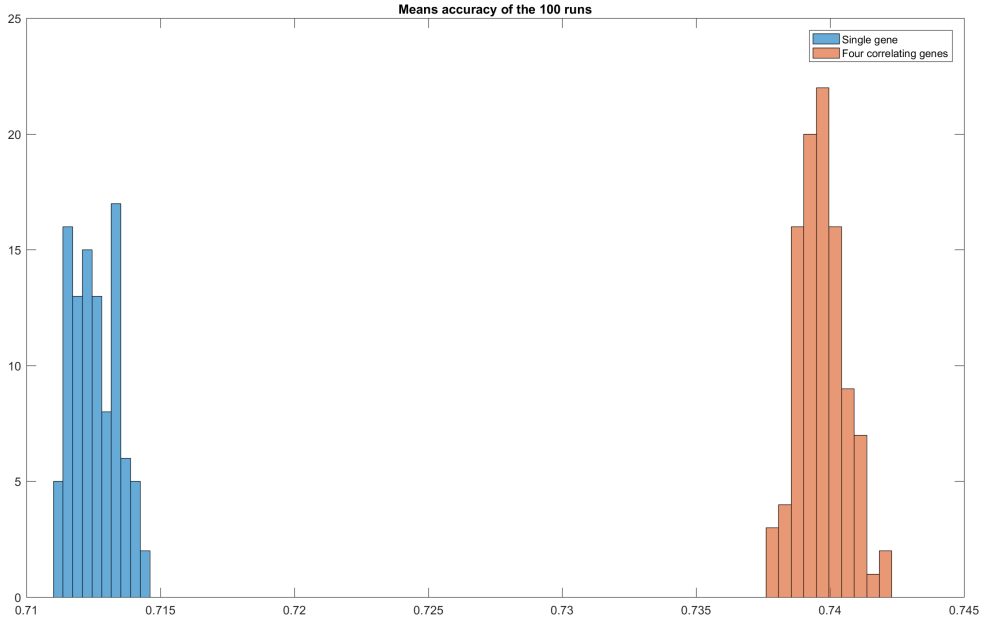


Figure 7: Two distributions representing the accuracies

With these results we can conclude that using the average of four correlated gene expressions could improve the stability of the classification as well as the classification itself in the case of subtype Luminal A (label 3). Given this conclusion we looked at selecting more than four correlated genes for determining the average activation to see if we could improve the results even more. In section B.2.1 the four figures of each these selections are depicted. The mean accuracy kept improving by selecting more correlated genes to determine the average expression and the standard deviation kept decreasing. The overview of results can be found in table 1 containing the mean standard deviation and table 2 containing the mean accuracy. These tables also contain in the last column in how many runs there was a significant improvement.

Table 1: Mean standard deviation using a single gene or an average of correlated genes.

Number of correlated genes	Single gene	Average	t-test
4	8.94%	8.69%	100%
9	8.93%	8.56%	100%
16	8.95%	8.56%	100%
25	8.94%	8.49%	100%
36	8.94%	8.44%	100%
49	8.94%	8.41%	100%

Table 2: Mean accuracy using a single gene or an average of correlated genes.

Number of correlated genes	Single gene	Average	t-test
4	71.25%	73.97%	100%
9	71.25%	76.12%	100%
16	71.24%	77.76%	100%
25	71.25%	79.26%	100%
36	71.24%	80.12%	100%
49	71.25%	80.64%	100%

These tests were then done again but now with a random label and its negation every time we select a gene in the algorithm found in Appendix B. In figure 3 is a representation given of single selection of a gene with its three most correlated gene expressions. Here is also a label selected to test against, which in this case is randomly chosen instead of choosing only label 3. The results of these tests are in table 3 and 4, with the detailed plots being in section B.2.2. Here we can see that the overall improvement is smaller but still significant. The same was done with a bigger group-size, see tables 5 and 6 and section B.2.3 for the detailed results. With these results we have shown that by using the average of correlated gene expressions, even with smaller amount of correlation, we could increase both the stability of accuracy of classification as well as accuracy of classification itself.

Table 3: Mean standard deviation using a single gene or an average of correlated genes with a random label each time we select a gene.

Number of correlated genes	Single gene	Average	t-test
4	8.21%	8.11%	95%
9	8.20%	8.06%	100%
16	8.22%	8.06%	100%
25	8.20%	8.02%	100%
36	8.20%	8.00%	100%
49	8.20%	7.99%	100%

Table 4: Mean accuracy using a single gene or an average of correlated genes with a random label each time we select a gene.

Number of correlated genes	Single gene	Average	t-test
4	79.15%	79.51%	93%
9	79.19%	79.70%	95%
16	79.09%	79.64%	96%
25	79.16%	79.74%	96%
36	79.14%	79.72%	96%
49	79.11%	79.77%	97%

Table 5: Mean standard deviation using a single gene or an average of correlated genes with a random label each time we select a gene and a group-size 100 instead of 16.

Number of correlated genes	Single gene	Average	t-test
4	3.49%	3.45%	98%
9	3.48%	3.42%	100%
16	3.47%	3.41%	100%
25	3.48%	3.42%	100%
36	3.48%	3.41%	100%
49	3.48%	3.41%	100%

Table 6: Mean accuracy using a single gene or an average of correlated genes with a random label each time we select a gene and a group-size of 100 instead of 16.

Number of correlated genes	Single gene	Average	t-test
4	80.19%	80.64%	100%
9	80.31%	80.95%	100%
16	80.39%	81.16%	100%
25	80.28%	81.16%	100%
36	80.33%	81.27%	100%
49	80.32%	81.35%	100%

3.2.2 t-SNE

t-Distributed Stochastic Neighbor Embedding (t-SNE) is a technique for dimensionality reduction that is particularly well suited for the visualization of high-dimensional data sets. The technique is a variation of Stochastic Neighbor Embedding but using a Student's t-distribution rather than a normal distribution, allowing it to work better in higher dimensions [9]. Given our high-dimensional representation of the gene expression in the pairwise similarity matrix this technique provides a way to reduce our data to a two dimensional image of points representing the gene expressions where the correlation between the genes is maintained. In the case of our 100 genes we have 100 dimensions as we calculated all pairwise correlations. The t-SNE algorithm we used will first reduce this to 30 dimensions using principal component analysis. Then the actual t-SNE algorithm will reduce the dimensionality further to just two, keeping correlated genes together. After using t-SNE on our data we got the two dimensional representation depicted in figure 8

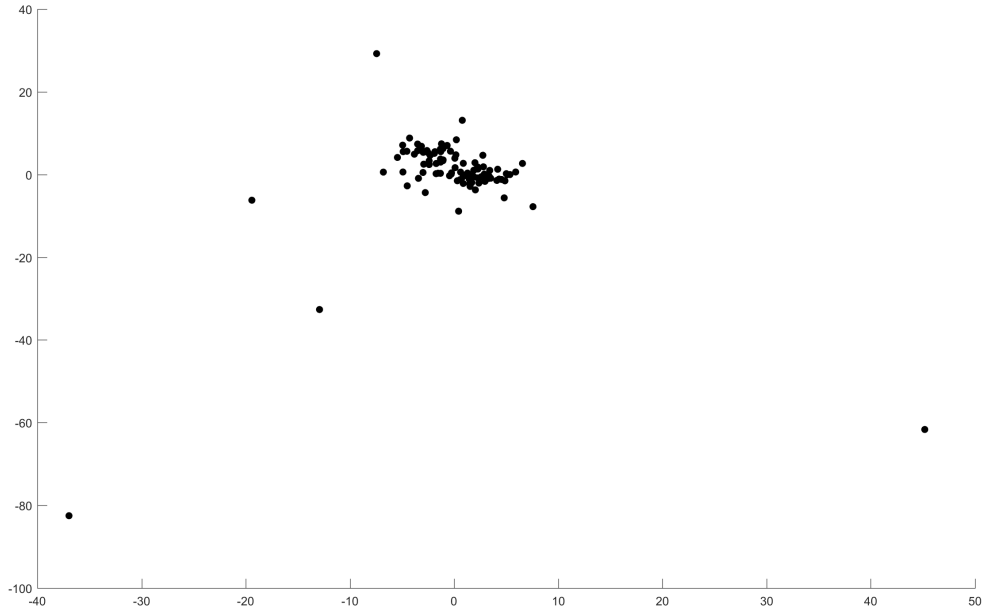


Figure 8: Gene correlation representation using t-SNE.

3.3 Creation of the images

In the research phase it was found that convolutional neural networks are primarily used on two-dimensional images. In the research report we discussed multiple techniques to construct this image from the t-SNE representation. We proposed a method for creating the image, and called it Iterative Matrix Expansion (IME). First we implemented the IME algorithm, but found some downsides. After that we implemented the Clustering algorithm. Here we will discuss the advantages and disadvantages of these algorithms. We will explain how well each implementation preserves the correlations between genes as distance between pixels as well as image size. The number of genes used in this case are the 100 genes with the highest standard deviation for reasons described in the research report.

3.3.1 Iterative matrix expansion

In the research report we proposed a way of converting the t-SNE result into a matrix. We developed this algorithm which we called Iterative Matrix Expansion (IME). IME works by iteratively adding points to a matrix M, starting with $M = 1 \times 1$. If two points are mapped to the same index, the points are split along the most discriminating axis, x or y, and a new row or column is added respectively. See also the pictorial in figure 9.

The image created by running IME on using 100 genes is given in figure 10. We used the implementation of IME as described by the algorithm 4 in Appendix C for generating the image. In doing so we found that this algorithm has several downsides. Firstly the running time, which is $O(n^3)$ where n is the amount of points or genes in our case. This is because every iteration using a new point can result in expanding the matrix, worse yet: it can result in expanding the matrix in the direction in which it is smallest, resulting in the largest possible matrix (i.e. $\lceil {}^{n}/_{2} \rceil \times \lfloor {}^{n}/_{2} \rfloor + 1 = O(n^2)$). Since the matrix has to be copied each time it is expanded this results in $O(n^2)$ running time per iteration of the outer loop. It is rather unlikely that t-SNE

will return a diagonal line, so this worst-case will rarely occur. Nevertheless the matrix will generally be much larger than needed to contain all data points. This is because t-SNE is tries to put neighbouring points close together. However, if they are, this algorithm usually increases the size of M.

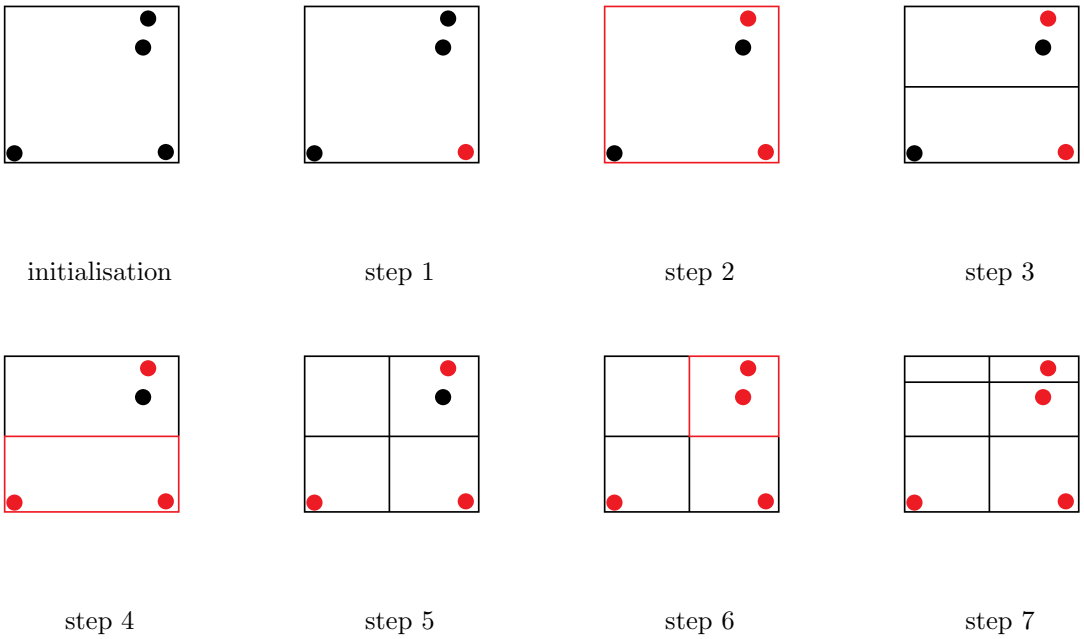
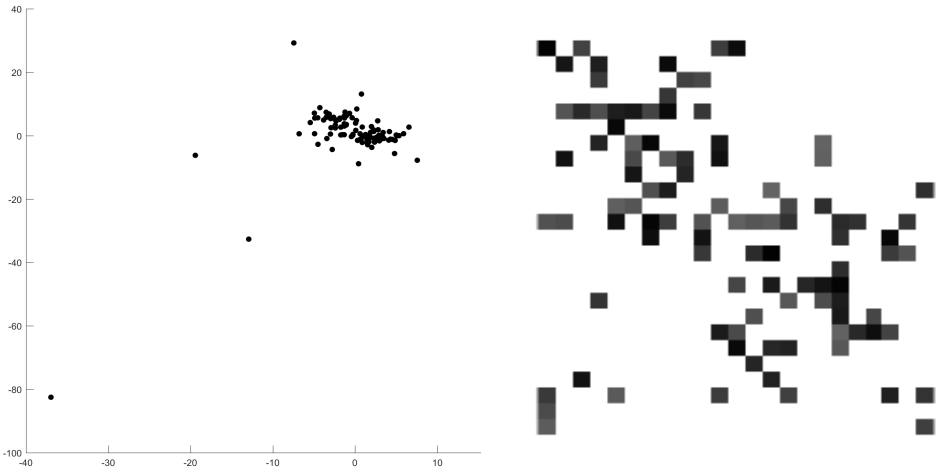


Figure 9: IME randomly selects unmapped points to add to the mapping, and adds either a row or column to the image if there is more that one point mapped to a pixel. Red dots represent mapped points, black dots represent unmapped ones. The lines represent the pixel borders and conflicts are highlighted in red.



(a) Representation of 100 genes by running t-SNE on the pairwise correlation matrix.

(b) Image created by running IME on the result of t-SNE depicted in 10a

Figure 10: A matrix representation created by IME

3.3.2 Evaluating the images

We suspect that IME does not do well at preserving the pairwise distances as provided by t-SNE. As such it would also not correctly put correlated genes together in the image. In order to try to confirm this, we tried various methods of comparing the matrix to the points returned by the t-SNE run it was based on. None of these were satisfactory. Therefore we propose to evaluate the data models based on the Jaccard similarity between the most correlated genes and the same amount of locations near the representation in the data model. This gives an indication as to how well correlated genes are kept together in the image.

The Jaccard similarity is a set similarity, calculated by taking the common elements and dividing by the total of all elements. In other words the Jaccard similarity of A and B is defined as:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

For each gene, we took the Jaccard similarity of the set of N most correlated genes, and the set of N genes that is closest in the image. We can't choose N either too large or too small since in both cases the similarity will be meaningless. With N too large it will always be 1 and with N too low it will be down to luck if we get either a 1 or a 0. So we chose $N = 4 \dots 25$ and took the mean of all results. This gives us a value between 0 and 1, where 1 means the high dimensional space was perfectly preserved and 0 means all highly correlated genes are out of the scope of our imaginary kernel. Realistically, the highest value achievable for any method relying on t-SNE is t-SNE's score itself. Over 20 runs, t-SNE scored a mean similarity of 0.34. IME was then run on each of the t-SNE outputs which scored it a value of 0.21. Images that were filled randomly scored a value of about 0.082. This did not change much over 1000 runs, however there are very many ways ($100! \approx 9 \times 10^{157}$) to fill a 10×10 matrix randomly, therefore this value might not be representative.

3.3.3 Clustering the points

Because of the way convolutional neural networks work and for speed of processing we want to avoid creating unused pixels. Given the large amount of unused space inside the image created by the IME algorithm (section 3.3.1) we proposed an algorithm based on a preexisting 'all nearest neighbor' method. Given that the distance between points represent the correlation, we try to cluster the highest correlated points together for the image creation. The algorithm is an implementation of a mean linkage clustering [10]. As input we take the output of t-SNE and the gene expression of all the patients corresponding to these points. With this algorithm we can create the tree as described in section A.3.1.

Here the nodes of the tree are clusters which can both represent clusters and points. The distinction is that a cluster with no set within it represents a point, while in all other cases representing a cluster. The algorithm starts at the lowest layer with a list of only points and uses hierarchical clustering to cluster these points, and in subsequent layers clusters. This is done until there is only one cluster left containing all the points in its branches. This cluster is the root of the tree created. The exact implementation is found in appendix D.

The clustering in each iteration is done using an existing implementation of the algorithm 'all nearest neighbors', which computes for each point, or in higher layers; 'for each cluster', the closest neighbor to that point (or in higher layers; 'cluster'), and the corresponding distance. This algorithm is used every iteration when we cluster a layer into a higher level layer to find the closest neighbor for each cluster. The clustering in each layer is illustrated in the simple point

cloud in figure 11 and the tree representation of these clusters is depicted in figure 12. Here a layer is the group of nodes with the same height in the tree.

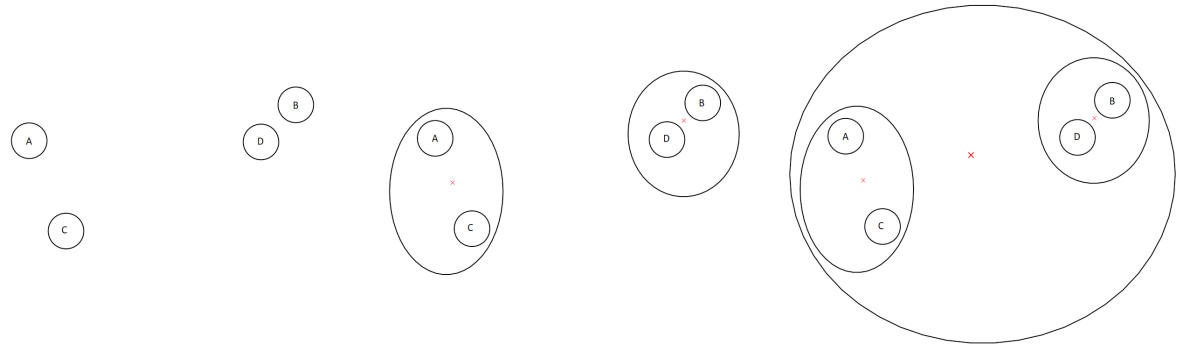


Figure 11: Clustering a point cloud of four points The red crosses are the means of the clusters, the larger one being the mean of the higher level cluster.

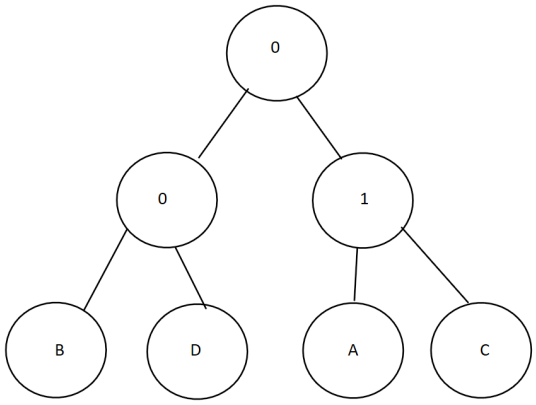


Figure 12: Tree representing the clusters of figure 11

To illustrate what the 'All nearest neighbor' algorithm does, we will give the output of this algorithm the first time we cluster the data in table 7. This is used in Algorithm 5 to find the point we wish to cluster with. Using table 7 the algorithm makes a random selection of a point, if this is B we add it to a cluster. Then we just look at the closest neighbor, in this case D. If this point isn't already in a cluster it is also added to the cluster.

Table 7: Example result of the 'All nearest neighbor' algorithm

point	A	B	C	D
neighbor	C	D	A	B
distance	3.4	1.3	3.4	1.3

We now have a tree where each node represents a cluster of points. To create a matrix we could simply put each leaf from left to right into the matrix row for row, given that leaves close to each other in the tree have a higher correlation then leaves far apart from each other, as discussed

in appendix A.3.1. For our image we used the 100 gene expressions with the highest standard deviation of the 1616 patients, to fill a 100 by 1 matrix. This is a one dimensional representation of the two dimensional representation of correlation generated by t-SNE. The correlation is only one dimensional because the points are put into the lowest layer of the tree and a layer only has one dimension; from left to right.

We extended our implementation of filling the matrix by making it possible to not only fill a matrix with the leaves of the tree but this matrix can also be filled with layers. These layers contain only clusters, so to determine the value to use in the matrix we simply take the average of the points inside the cluster. Our implementation takes a integer representing the minimum amount of cells in the matrix. Then it looks at the first layer, starting at the root of the tree, that has at least that amount of clusters. So if we give the total amount of points, 100 in the case of the data we use, as the minimum we have recreated our simple algorithm we have discussed at the top of this section. This is comparable to the Average Pooling layer which is often used in a convolutional neural network. By averaging the clusters the average is always taken of highly correlated gene expressions, which may improve the accuracy as we have seen in section 3.2.1. With this implementation we can also decide how big the matrix will be and compare results of the different matrices.

Another way we tried improving the results is by sorting the input list of points or clusters and its distance to its closest neighbor created by the 'All nearest neighbor' algorithm based on the distance to its closest neighbor. This improvement makes the algorithm behave like the UPGMA algorithm by Sokal and Michener [11], but with the addition described above. With the addition of sorting the algorithm will create more pairs as clusters as the distance of the first point will always be the same as the distance of the second point. This is because the distances are between two points and that we sort on the shortest distance. The algorithm gives the closest neighbor for each point, were the two shortest distances are always a pair of closest points, thus creating a cluster with only those two points. This also means that the most correlated points are always together in the lowest layer of the tree, which could improve the identification by the neural network. The implications can be explained best by figure 13. In this figure we can see that by sorting based on the distance we maintain the fact that point B and C are closest to each other in the tree by being in a distinct cluster. In the original implementation we can't see that the distance between point A and B is greater than the distance between B and C after the first time we cluster.

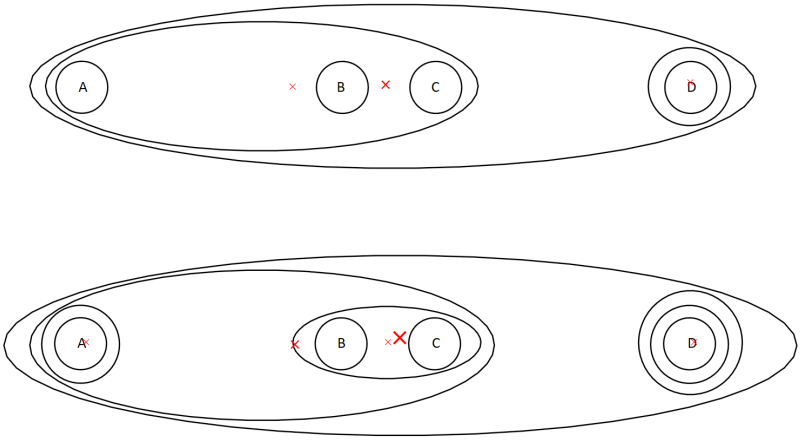


Figure 13: Up: Choose as first point A Down : Sort on distance

To conclude, with this algorithm we can create images without unused cells or 'pixels' while creating a one dimensional representation of the, by t-SNE converted, two dimensional representation of correlations. To get good identification by the convolutional neural network it has to be adapted for the one dimensional correlation. The stride and kernel size of the convolutional layer could for example be only one dimensional and a pooling layer could be excluded as averaging is already done in image creation. The advantage of using this algorithm is a speed up of the convolutional neural network as there are less layers inside the network and given that averages are taken in the image creation resulting in smaller input images.

3.3.4 Evaluating the images

We wanted to see if there is an improvement to the ability to preserve the correlations when compared to images created by IME. Therefore we calculated our metric for fitness of the image (section 3.3.2). As stated before, we ran t-SNE 20 times and used it as input to generate both the IME and the cluster matrix. The cluster matrix had a similarity rating of 0.28, meaning that it did improve upon the IME image. So, we should expect to find better results from the convolutional neural networks running on the cluster data model, than those using IME's data. An overview is given in table 8.

Table 8: preservation of higher dimensional relationships

random	IME	cluster	t-SNE
0.0819	0.2027	0.2758	0.3418

3.4 Comparing networks and data sets

In section 3.3 we describe two ways of converting our two dimensional t-SNE output data into a two dimensional image. These images should be tested on different neural network configurations. A neural network, and specifically a convolutional neural network has many parameters which can be changed. The impact of changing these parameters can change the overall results and for different inputs different configuration produce the best result. To compare and test different networks and images, we proposed and developed an application which can automatically run multiple network models on different images, and save its results for later usage.

3.4.1 Dealing with randomized initialized weights

Because the results of the network configurations are dependent of their randomized weight initialization, they might not perform the same under similar conditions. Therefore, to compute a more reliable average accuracy, the application runs each network data set combination multiple times, each time with a predetermined initialization seed to initialise the starting weights of the neurons. This so that each network-image combination runs the same number of iterations on the same seeds. Other parameters than the initialisation seed are left unchanged, including the training and test set. While we compute the average, we do save the individual results, so that the number of computed results can be expanded later on.

When the application is used to extend the number of computations, the application will use a set seed to compute the seeds for the network initialisations, so that for each run the application will use the same unique seed for each iteration. Because the seed guaranties the same result on every time or device, this application can skip the first several iterations when these are already run and written on the disk. These seeds don't have to be recomputed, because the result will

be exactly the same when it is ran on the same initialization seed on every computer. This also makes it possible to distribute the computation tasks among multiple computers, or to continue on an application run on an other device.

3.4.2 Network configuration evaluation

This application uses evaluation class that is build into the DL4J library, to evaluate the networks. This class is called *org.deeplearning4j.eval.Evaluation* and can be used to compute the accuracy, precision, recall and the F1 score, which is computed after counting all TN (true negatives), FN (false negatives), TP (true positives) and FP (false positives). Because of the usage of softmax in the output layer of networks that are used, the Evaluation instance counts one class as TP and all other classes as TN when the classification was correct. And when a classification is incorrect, the Evaluation instance counts one class as FP, one class as FN and all other classes as TN.

Because we classify more than two classes on a softmax layer, the standard method for computing the accuracy, $(TP + TN)/(TP + FP + FN + TN)$, could distort the interpretation since there will always be some TN on every result. The mentioned evaluation class however, computes the accuracy as TP divided by the number of classified samples instead. Which computes the percentage of successful classifications where every classifier is predicted correct. For this reason, this last method is used in this work to compute the accuracy. Because the evaluations are saved entirely, we can later still access the TP, TN, FP and FN counts without recomputing the network data set combinations, and therefore easily compute other averages with this implementation.

The evaluation files are stored on the disk so that they can later be reused without having to recompute these results. Saving these files makes it also possible to distributed the testing of configurations, using this application, among other computers. Also, if an user wants to extend the number of iterations on the data, using an other device, an user has to run the same network configurations. Therefore the network configuration is also saved to the disk. This also makes it possible to add a new image, which can easily be compared with the old images by running this application on the user its own device, which will then only compute the missing data.

These networks are saved to the disk on initialization of a network configuration by the Java code, as a list of their properties as a JSON string. This string can be read and understood by humans, as well by the program using a JSON object reader found in the *com.fasterxml.jackson* package. Because the application will automatically find all available network configurations in the network folder, the user is able to add and run new configurations by hand without having to edit the Java code.

3.4.3 Image file format

As for the network configurations, to prevent mixing of images, the application requires consistent data which does not change over different devices or over time. These image files follow the following set of rules, which are used to read as well as to create the images. We store these images into three separate files, all in a shared folder. This folder contains one '.dat' file, '.lab' file and '.meta' file. In the '.dat' file the images are stored separated by new lines as a list of numbers, which express the gene expressions. In the '.lab' file the labels of the images are stored, each separated by a new line, so that the label its line number matches the line number of the image it belongs to. Finally there is the '.meta' file, which describes the image's properties which can't be described in the '.dat' or '.lab' file, like the width and height of the image.

One important feature of the meta file is the time stamp, or image set version number. This time stamp functions as a unique identifier of the image set, so that a device using this image set won't put results of other image sets in the same result folder. This time stamp is created on creation of the '.dat', '.lab' and '.meta' file. This time stamp is defined in milliseconds as the difference between the current time and midnight, January 1, 1970 UTC, which is the Unix epoch. Therefore it is unlikely that multiple users generate their own data set on their own device with the same time stamp.

Because of this given rule set, it is possible to inject new image sets for training new or existing convolutional neural networks, without changing any code. However, this data loading class and its database class are also replaceable by own implementations, when using the *GeneExpressionDatabase* interface, which enables users to extend this application using other data formats.

3.4.4 Gene expression database

The *GeneExpressionDatabase* interface is responsible for providing some meta data as the width and height of the image, but is also responsible for providing the data itself in the form of a subsets. This selection of subsets uses a given training set percentage, which determine what percentage of the data should be used to train the neural network models on. What is left is then used to test the performance of the network configuration on.

Unfortunately the samples are not equally distributed among different classifications, causing some labels being over overabundant in the training or testing set, while having just a few samples in the other set. This could cause the network models to train on different labels then are used to test. Therefore this class first collects the samples in separate lists for each label type before taking subsets to feed the network model. When an instance of this class is requested to give a subset of its data, it gives the requested percentage of each of these lists, enforcing that each label type is relatively equally presented in the training set as the test set.

3.4.5 Result extraction

The evaluation files stored on the disk are only accessible by the Java library, because of the evaluation file's format. Therefore we developed two different methods to extract this information for further analysis using Matlab or other programs. One method converts all available data into a table, with the network configurations as rows and image sets as columns. The other method extracts a single detail about the data, and writes these down as lists.

The table generator iterates over each image- network-configuration combination, applying a table filler to fill the empty cells within the table. The filler used is let open for extension, so that user can develop their own filler to fulfil their own needs. The filled class used for some of the early observations is the average accuracy filler, which fills in the empty cells within the table with the average accuracy measured for each image network configuration combination.

In a similar manner the other method of extraction, the data listing method, generates the data for each image network configuration combination. But instead of filling a table, the listing method generates separate files. This method is also left open for extension. During this project a single listing method was used, which lists all measured accuracies of the generated evaluation files, for each network-image combination in a separate file.

3.4.6 Future usage of this application

This application is capable of automatically running network configurations using the DL4J library, using the available image sets as the train and test data. It is designed to run different image-sets/data-sets or network configurations to compare and optimise. It saves its results and used network configurations and therefore prevents the re-computation of existing results, which enables the user to continue the application or extend the results later on without having to recompute everything. There are multiple methods for data extraction, and it is easily expand upon. Therefore this application is not only useful during the experiments during this project, but it is also reusable by other projects involving neural network deep learning methods.

3.5 Baseline neural network configuration

To design new networks, we would like to have a baseline network to compare these networks with. For our baseline we will start with a standard single hidden layer fully connected network. This network contains a input layer with 100 'input neurons', h 'hidden neurons' in the hidden layer with a *relu* activation and 5 'output neurons' with a softmax activation.

For the number of neurons in the hidden layer, we expect a number between the number of input and output neurons. We expect a better performance with more neurons in the layer, because with each hidden neuron the chance of a neuron to reach a good configuration increases, increasing the likelihood of better configuration. However, because there are more parameters to learn, this also increases the chance of over-fitting. We decided to test multiple networks with a different number of neurons in the hidden layer. We tested for $h = 5, 10, 20, 40, 80$ and 160 neurons in the hidden layer.

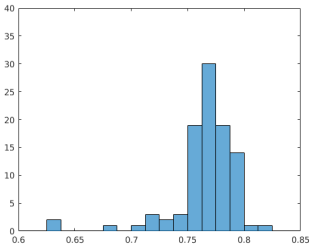
In figures 14a, 14b, 14c, 14d, 14e and 14f we see the distributions representing the accuracy of these six of these configurations, each with a sample size of 100 runs. As can be seen in this figure, the accuracy increases slightly for each increase in neurons in the hidden layer. And we also see the deviation shrink with more neurons, giving a more consistent result. table 9 confirms our observation, where we see the average accuracy increase when more neurons are used in the hidden layer. We conclude using more neurons in the hidden layer produces a more fit network.

In table 9 we can also observe the diminishing effect of adding more neurons, which could be explained because of the chance of getting a better configured neuron only increases with one over the already added neurons for each neuron added. We assume this increase in performance and reduction of the deviation will continue by increasing the number of neurons in the hidden layer. To confirm this we ran the same network for 320 hidden neurons and 640 hidden neurons, which continued the trend (see table 9 and figures 14g and 14h)

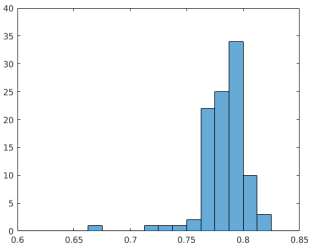
Because the training speed of the network is highly dependent on the number of neurons it has to train, a cost-benefit consideration should be made. We decide not to use more neurons in the hidden layer, than in the input and output layer are used. In our case the biggest layer is the input layer with a neuron count of 100. Therefore we will use a hidden layer neuron size of 100.

Table 9: Average accuracies of different network configurations over a hundred iterations.

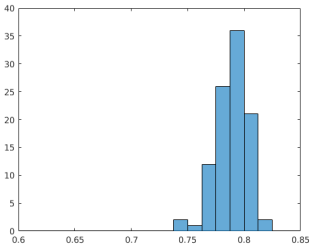
Hidden layer size	Average
5	0.7506
10	0.7835
20	0.7894
40	0.7910
80	0.7956
160	0.7967
320	0.7990
640	0.7993



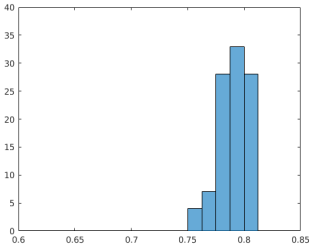
(a) m = 5



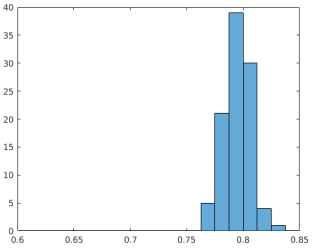
(b) m = 10



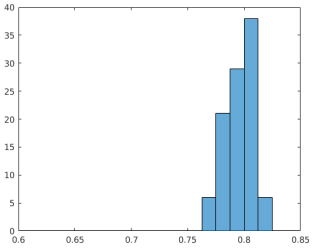
(c) m = 20



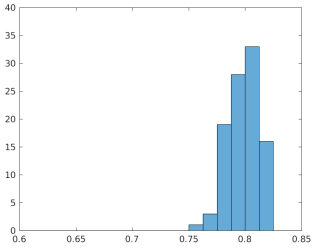
(d) m = 40



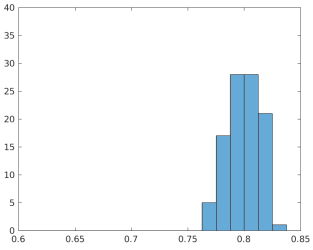
(e) m = 80



(f) m = 160



(g) m = 320



(h) m = 640

Figure 14: Accuracy distributions for different hidden layer configurations over a hundred iterations.

4 Results of using a convolutional neural network

The next step is to convert this baseline network found in section 3.5 into a convolutional network. In section 3.2.1 we have shown that combining gene expressions of highly correlated genes has higher accuracy when used to classify a single class compared to a single gene expression, therefore adding a convolutional layer to the neural network could give us a more accurate classification. A convolutional neural network does however have many parameters as discussed in section 3.4 so to conclude if a convolution neural network can improve upon the baseline, multiple configurations are trained.

4.1 Configurations and test procedure

First we determined what the effect of using a convolutional layer in a neural network is by testing different kernel-sizes and number of kernel using our program proposed in section 3.4. This means that for every model we create of a network configuration and image representation we train 20 times with different initialization. This creates a distribution of accuracies on which we can determine the effect using t-test and effect-size. We will use a one tailed t-test which gives a measure of significance that the mean of one distribution, in our case the accuracy of a convolutional neural network, is greater than that of another distribution, in our case the baseline of one kernel.

Effect-size is a simple way of quantifying the difference between two groups. In our case there is a improvement of accuracy in the case of values greater than zero, and decrease of the accuracy in the case of values less than zero. By using the effect-size we look not only at the mean of the two distributions we would like to compare but also at the standard deviation, which is a standardised mean difference. For example, an effect size of 0.9 means that the average accuracy of the experimental configuration is 0.9 standard deviations above the average accuracy of the baseline to which we compare the experimental configuration. The equation used to calculate effect-size is defined below (Glass' Δ).

$$\Delta = \frac{mean(experiment) - mean(baseline)}{std(baseline)}$$

We also give the average accuracy of the 20 runs of each configuration and look if we can improve the accuracy by adding three extra features; stride of 2×2 rather than 1×1 on the convolutional layer, average pooling and maximum pooling with again 20 runs on each configuration. With these different configurations we will test the most common configurations of a convolutional neural network.

4.2 Number of kernels inside the convolutional layer

We determined the optimum number of kernels to use in the convolutional layer by running configuration with different number of kernels and measure the accuracies. This is done to assess whether the accuracy of the baseline could be improved upon by using a proper kernel-size. With methods discussed in section 4.1 we determined if there was indeed an improvement upon the baseline. The results of the t-tests and corresponding effect-sizes can be found in table 11. In these configurations we used for the 2 dimensional configurations a kernel-size of 3 by 3 and for the one dimensional configurations a kernel-size of 4 by 1, reason being that the correlation in the image created in section 3.3.1 is represented in two dimensional space, while the algorithm in section 3.3.3 represents the correlation in one dimensional space.

Table 10: Results of the t-test, the corresponding effect-size and accuracy for comparing different number of kernels inside the convolutional layer for IME with the baseline.

Kernels inside convolutional layer	H-value	P-value	Effect-size	Average accuracy
1 (baseline)	NaN	NaN	0	79.07%
5	0	0.8055	0.2296	79.30%
10	0	0.8097	0.2396	79.31%
15	0	0.9014	0.3594	79.44%
20	0	0.9601	0.4293	79.51%
25	0	0.9531	0.4393	79.52%
30	0	0.8762	0.2995	79.38%

Table 11: Results of the t-test, the corresponding effect-size and accuracy for comparing different number of kernels inside the convolutional layer for Clustering with the baseline.

Kernels inside convolutional layer	H-value	P-value	Effect-size	Average accuracy
1 (baseline)	NaN	NaN	0	78.71%
5	1	0.0175	0.3997	79.35%
10	1	0.0033	0.5711	79.63%
15	1	5.0170 E-4	0.7107	79.86%
20	1	7.8268 E-4	0.7233	79.88%
25	1	7.7742 E-4	0.6916	79.82%
30	1	5.2936 E-4	0.6409	79.74%

With these results found in tables 10 and 11 we can see a difference in improvement between the two data models used. By choosing more kernels up to 20 we could significantly increase the accuracy in the case that clustering is used in the image creation step. Adding more than 20 kernels caused the accuracy to start dropping again. In the case of using IME in the image creation step the p-values are less than 0.05, meaning not a significant improvement, but overall adding kernels up to 20 would increase the accuracy as well. The reason for the decrease in accuracy when using more than a discussed above is most likely caused by overfitting.

Overfitting occurs when a model is excessively complex, such as having too many features relative to the number of observations. This can cause the network to recognise noise in the data as signal, causing it to lose predictive power for data lacking the same noise (such as the test set). We chose to select 20 kernels in the case when training on Clustering images as well as in the case of using IME images as the optimal number of kernels in the convolutional layer.

4.3 Kernel-size of the convolutional layer

We wanted to test the impact of different kernel-sizes in the convolutional layer to see if an improvement upon the baseline could be made. In testing we created different configurations of convolutional neural networks for each of the data models used to represent the gene expressions, reason being that the correlation in the image created in section 3.3.1 is represented in two dimensional space, while the algorithm in section 3.3.3 represents the correlation in one dimensional space.

We compared the current baseline with multiple other networks with different kernel-sizes. Three of these are convolutional networks, which read from a two dimensional data set, which in our case is the matrices or images created using the IME algorithm (section 3.3.1). The other

convolutional networks read from one dimensional data set created using the clustering algorithm (section 3.3.3).

The following kernel sizes where tested: $2 \times 2, 3 \times 3$ and 4×4 on the two dimensional data set and 2×1, 3×1, 4×1, 5×1, 9×1 and 16×1 on the one dimensional data set. We first determined with a one tailed t-test if there was a configuration better than the baseline. This baseline has a convolutional layer but by choosing a kernel-size of one by one we are essentially saying low level features are per-pixel. These features don't affect neighbouring pixel at all, so we should apply the same operation to all pixels.

Given that the pixels in our case are gene expression this causes the neural network to not use multiple correlated gene expression in the kernels of the convolutional layer. The stride of this layer was one by one meaning that we only moved one cell at the time over the images. The results can be found in table 12 and 13. For each of the configurations we will note the h and p-values of the t-test as well as the effect-size and the average accuracy of the 20 iterations of each configuration.

Table 12: Results of the t-test, the corresponding effect-size and accuracy for comparing different kernel-sizes in the convolutional layer with the baseline. All the images were created by IME.

Kernel-size convolutional layer	H-value	P-value	Effect-size	Average accuracy
1×1 (baseline)	NaN	NaN	0	79.57%
2×2	0	0.8629	-0.1892	79.33%
3×3	0	0.6287	-0.0493	79.51%
4×4	0	0.9628	-0.2632	79.24%

Table 13: Results of the t-test, the corresponding effect-size and accuracy for comparing different kernel-sizes in the convolutional layer with the baseline. All the images were created by Clustering.

Kernel-size convolutional layer	H-value	P-value	Effect-size	Average accuracy
1×1 (baseline)	NaN	NaN	0	80.17%
2×1	0	0.9953	-0.3135	79.73%
3×1	0	0.9901	-0.2770	79.73%
4×1	0	0.9584	-0.2114	79.88%
5×1	0	0.9791	-0.3062	79.74%
9×1	0	0.9193	-0.4010	79.61%
16×1	0	0.9997	-0.6270	79.30%

These results suggest that a convolutional layer in the neural network has a negative impact on the accuracy of classification when looking at low level features in multiple correlated gene expressions. This is the case for both images creation techniques used with images created by IME having less decrease in accuracy when increasing the kernel-size when compared to images created by Clustering. The best kernel-size of the convolutional layer is in this case one by one.

To correctly interpret these results we must look at the underlying data. If we didn't we'd assume IME is better when using a convolutional neural network. However this is not necessarily the case. The IME data model contains many zeros, while the cluster data model does not. It is therefore not fair to compare 16×1 and 4×4, since the effective kernel size is smaller for IME.

Noteworthy is the fact that the best convolutional neural network for both data models, is not a convolutional neural network at all, as choosing a kernel-size of one by one we are essentially

saying low level features are per-pixel, and they don't affect neighbouring pixel at all, and that we should apply the same operation to all pixels. This can mean several things. One possibility is again overfitting. However another possible explanation is that the data does not contain the kind of structures convolutional neural networks are designed to recognise. In the creation of both data sets we put highly correlated genes together. That does not mean however, that the convolutional neural network can use this. convolutional neural networks find local spatial correlation patterns. These are patterns of pixels contrasting in some way with the pixels in their immediate surroundings. These patterns are found, no matter where in the image they occur. However, the way we fed data into these convolutional neural networks, the location does matter, and the local spatial correlations between pixels may vary.

To explain these results it would be beneficial to look into the actual weights of the convolutional layer. However due to the complexity of this analysis and the limited time of the project we didn't investigated further, but we can speculate as to why there is an decrease in accuracy when increasing the kernel-size. It is most likely that the kernels did not recognise the functional units of genes in the image. If they did the accuracy would arguably be greater than the baseline's. This only results in the outputs from the convolutional layer to be confounded with values of neighbouring pixels. This in turn leads to the classic NN (which does all the work in the baseline) to perform increasingly worse as the kernel size is increased, which is exactly what we see.

4.4 Stride & Pooling

To speed up training time of the convolutional neural network one could use a stride in the convolutional layer or a pooling layer. To test the impact of these features on our accuracy we tested for Clustering images and IME images convolutional networks with a stride of 2×2 inside the convolutional layer, an average pooling layer and a max pooling layer. In the cases of adding a pooling layer we put the stride of the convolutional layer back to the standard 1×1. We again used t-test and effect-size to see if an improvement could be made on the results in table 14 and 15.

Table 14: Results of the t-test, the corresponding effect-size and accuracy for comparing extra features in the convolutional layer with the baseline. All the images were created by IME.

Extra features	H-value	P-value	Effect-size	Average accuracy
No features (baseline)	NaN	NaN	0	79.51%
Stride 2×2	0	0.9960	-0.6707	78.82%
Max pooling	0	1	-2.5026	76.95%
Average pooling	0	1	-3.2134	76.22%

Table 15: Results of the t-test, the corresponding effect-size and accuracy for comparing extra features in the convolutional layer with the baseline. All the images were created by Clustering.

Extra features	H-value	P-value	Effect-size	Average accuracy
No features (baseline)	NaN	NaN	0	79.88%
Stride 2×1	0	0.3980	0.0345	79.93%
Max pooling	0	1	-2.6391	75.96%
Average pooling	0	1	-2.8394	75.67%

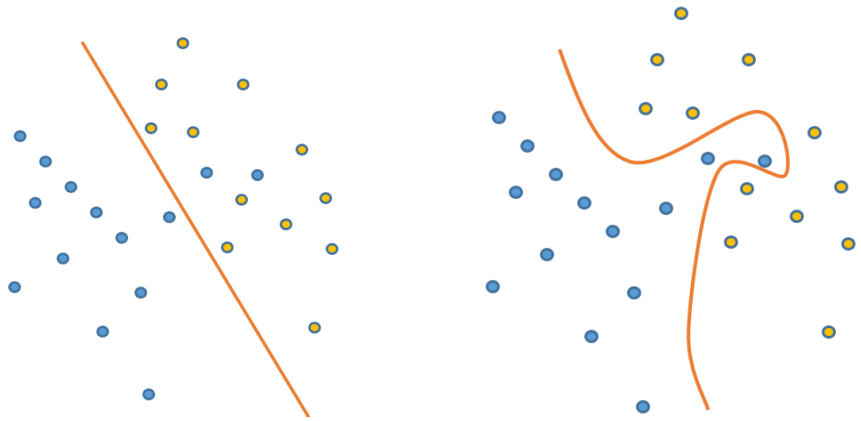
With the results found in table 14 and 15 we can see that adding a pooling layer decreases the accuracy for both representations of gene expressions. We expected some loss, however the loss

in accuracy is much more severe than anticipated. Therefore we can conclude that as we pool data to improve speed of training we lose important information. This most likely means that, rather than representing a single feature of the image, the individual values in one feature in our convolutional neural network represent different attributes of the image. Max pooling in a convolutional neural network keeps only the most interesting value for one feature. With the data we fed into the network however max pooling might keep only one attribute, resulting in a loss in accuracy.

After testing the different configurations of a convolutional neural network the best configuration was found to be one with a convolutional layer with a kernel-size of 1×1, a number of kernels of 20 without the addition of a pooling layer. This is not a configuration we wish to use as this is equivalent to repeating the input data 20 times with different initializations of the weights and taking best result found. This does result in a better accuracy but doesn't use the correlated gene expressions, as discussed in section 4.3.

4.5 Linear classifier

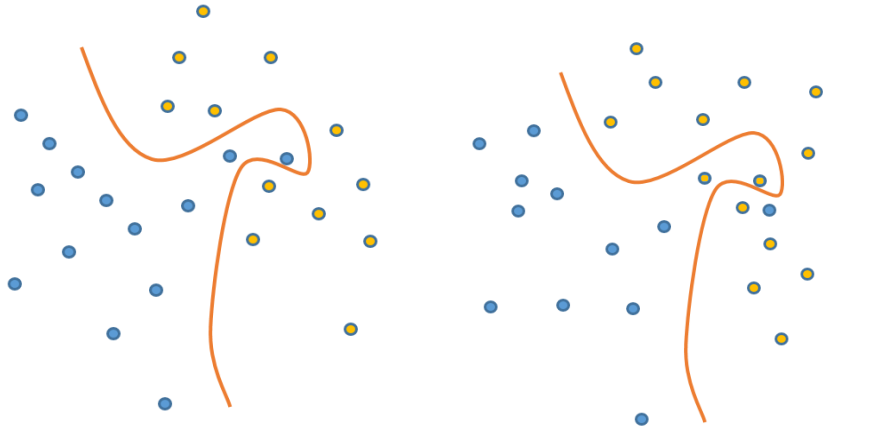
In the previous sections we tried to using networks with a hidden layer. However, the data might be too complex and we might not have enough data to train a hidden layer well, therefore we tried the same network without a hidden layer. Removing the hidden layer makes the classification linear instead of non-linear which we used in the previous configurations. The network uses linear combinations of the feature values to classify the data on a yes/no basis. One for each of the possible classes. Five of these are combined to give the actual classification, choosing the one with the highest confidence. The difference can be explained most easily with figure 15. Here the point represent the patients, where the location can be determined by combinations of multiple gene expressions in both the x and y value.



(a) Linear classifier (b) Non-linear classifier

Figure 15: Difference between a linear and non-linear classifier

By removing the hidden layer we can speed-up the classifications significantly. Given the high complexity of our data which may causes the lack of any meaningful structures inside the images it can also cause a higher accuracy. The data set used is also relatively small with only 1616 patients, which can cause the neural network to overfit based on only a few exceptions in the data. This is also represented in figure 15, given that the non-linear classifier's 'bubble' is based on only two exceptions in the data. When training on such a small training set it could be that such an exception in the train set can cause an incorrect classification in the case of the test set example found in figure 16.



(a) Non-linear classifier created by the train set

(b) Non-linear classifier used on a test set

Figure 16: Potential problem of overfitting caused by a small train set

Another reason that linear could perform better is the possible noise in the data set used [12]. This noise could cause many exceptions, causing there to be less distinction between different classes. An example is given in figure 17. If in the test set there is only a small movement of the points inside the bell curves it can cause an incorrect classification.

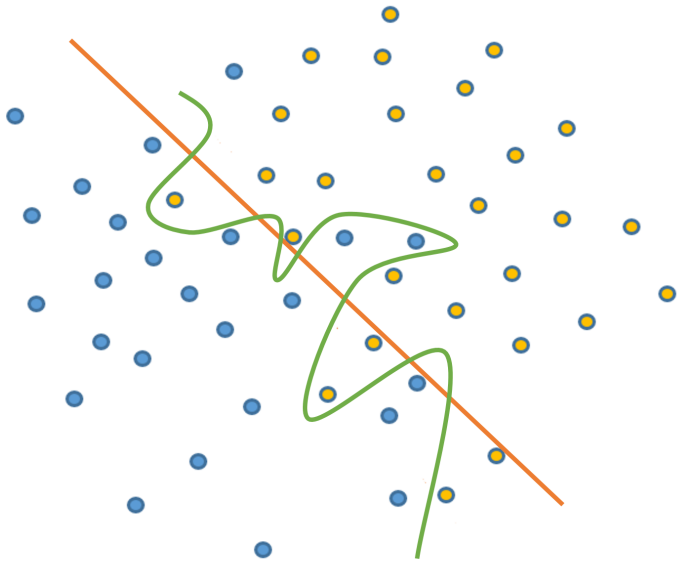


Figure 17: Orange: Linear classifier, Green: Non-linear classifier

By using a linear classification we do however no longer try to look at an image, because we are just putting the inputs, in our case the gene expression, directly onto the output layer. This without intervention of a convolutional layer. The results of using a linear classifier can be found in table 16 when using images created by IME and in table 17 when using images created by Clustering.

4.6 Best configuration of the neural network

We will finally compare the best convolutional neural network with the baseline and with the linear configuration discussed in section 4.5. The baseline configuration is the neural network found in section 3.5 and the best configuration of a convolutional neural network is one with a convolutional layer with kernel-size 1×1 and a number of kernels of 20, which strictly isn't a convolutional neural network as discussed in section 4.3. The linear configuration as only the one hundred input neurons and 5 output neurons with a softmax function to classify.

Table 16: Results of the t-test, the corresponding effect-size and accuracy for comparing the linear neural network to the baseline of section 3.5 and the best configuration of a convolutional neural network when using IME.

Configurations	H-value	P-value	Effect-size	Average accuracy
baseline	NaN	NaN	0	79.40%
best CNN	0	0.2319	0.1676	79.57%
linear	1	9.9554 E-04	0.9185	80.33%

Table 17: Results of the t-test, the corresponding effect-size and accuracy for comparing the linear neural network to the baseline of section 3.5 and the best configuration of a convolutional neural network when using Clustering.

Configurations	H-value	P-value	Effect-size	Average accuracy
baseline	NaN	NaN	0	79.13%
best CNN	1	0.0025	0.6473	80.17%
linear	1	0.0291	0.5457	80.01%

When comparing the neural network using linear classifiers to the best configuration found in previous sections we found that when using data created by IME that the linear classifier performed better. This is obviously no longer interpreted as an image by the linear classifier. To confirm this we ran a one tailed t-test to compare the best 'convolutional' neural network found and the linear configuration and found there to be a significant improvement when using the linear configuration with a p-value of 0.0026. When using data created by Clustering it was found that the difference between the best configuration found in previous sections and the linear configuration isn't significant with a p-value of 0.7376. The most likely reason for difference in significance of improvement when using a convolutional neural network is the fact that the input data is just put 20 times into the network with different initialization of the weights, as was discussed in 4.3, causing the distribution of the accuracy to narrow with a mean slightly higher.

The linear configuration is similar in accuracy when compared to the best configuration found in previous section, but can train much faster given the significant decrease of neurons in the neural network. The speed-up in training makes it easier to use more gene expressions to train while still being faster than complex non-linear configurations such as a convolutional neural network. In a linear configuration it is also easier to find which gene is interesting for classification, as there is no hidden layer with activations. Therefore a linear configuration has high interpretability. So overall the better configuration of a neural network is found to be a neural network without a hidden layer causing it to use linear classifiers in the case of our images.

4.7 Random data producing better results

After comparing the different configurations of the convolutional neural network we also compared the performance of classification between the images created. Reason being that we would like to see if the images created by us were useful for structure detection by a convolutional neural network. To compare we looked at the same configurations of the neural networks with only changing the images used. To test the significance of the difference we used a two tailed t-test, the effect-size and a one tail t-test to make sure it is significantly worse compared to a random image in the case of a right tailed (RT) t-test or significantly better in the case of a left tailed ed t-test (LT).

4.7.1 IME versus Random

Table 18: Results of the t-test, the corresponding effect-size and accuracy for comparing the IME matrix creation with a image with gene expression put random into the matrix

Kernel-size	Avg IME	Avg Random	H-value	P-value	Effect-size	H-value RT
1×1	79.57%	80.11%	1	0.0086	-0.4210	1
2×2	79.33%	80.67%	1	6.7946 E-06	-1.0797	1
3×3	79.51%	80.00%	1	0.0177	-0.4675	1
4×4	79.24%	79.99%	1	9.3327 E-04	-0.6187	1

Table 19: Results of the t-test, the corresponding effect-size and accuracy for comparing the IME matrix creation with a matrix with gene expression put random into it

Kernels	Avg IME	Avg Random	H-value	P-value	Effect-size	H-value RT
1	79.07%	78.15%	1	0.0287	0.8548	0
5	79.30%	79.40%	0	0.7204	-0.0770	0
10	79.31%	79.61%	0	0.1401	-0.2485	0
15	79.44%	80.12%	1	0.0037	-0.6391	1
20	79.51%	80.00%	1	0.0177	-0.4675	1
25	79.52%	79.92%	0	0.0625	-0.4372	1
30	79.38%	80.06%	1	3.8563 E-04	-0.7392	1

With all the configurations with only a convolutional layer we can see in table 18 and 19 that an image created by randomly putting our 100 gene expression into the image performed better than images created using IME. The reason most likely being that the same as in section 4.3, which is that the data does not contain the kind of structures convolutional neural networks are designed to recognise.

Table 20: Results of the t-test, the corresponding effect-size and accuracy for comparing the IME matrix creation with a matrix with gene expression put random into it

Feature	Avg IME	Avg Random	H-value	P-value	Effect-size	H-value LT
Stride 2×2	78.82%	78.88%	0	0.8329	-0.0538	0
Max pooling	76.95%	73.97%	1	1.0617 E-10	2.4896	1
Avg pooling	76.22%	74.02%	1	2.7463 E-05	1.7448	1

When comparing the image creation technique IME with a random image, on a convolutional neural network with a pooling layer, we saw a big difference in accuracy as found in table 20. Here the images created by IME outperformed the random generated images significantly, as shown by a one tailed t-test in the last column. We came up with the following hypothesis; taking the average of four correlated gene expressions produces more stable accuracy as well as accuracy itself when comparing to the average of four random genes. This is also the case when taking the max gene expression instead of the average. The reason being that an average pooling layer takes the average of several, in our case four values, to decrease the amount of cells inside the images to speed up training. We already concluded in section 3.2.1 that putting correlated genes together would improve classification.

There is however a difference with the test in section 3.2.1 in the fact that the average pooling layer takes the average of four random gene expressions instead of only using a single gene expression. To find if our hypothesis holds true we ran the test in section 3.2.1 again, but instead of a single gene expression we take the average of four random genes. We will only test four correlated gene expressions versus the four random gene expression, as this is what is done inside the pooling layer. The results can be found in the figures 18, 19, 20 and 21.

Of the 100 runs of t-test, in 80 cases the null hypothesis was dismissed out of the 87 cases that the accuracy had improved. This means that the stability was significantly improved when using correlated genes. The null hypothesis wasn't dismissed every time, which may be caused by the fact that the random selected genes sometimes were correlated. But even with this fact 87% of the time there was a significant improvement when selecting four correlated gene expression, causing the hypothesis to hold true in the case of taking the average of the expressions.

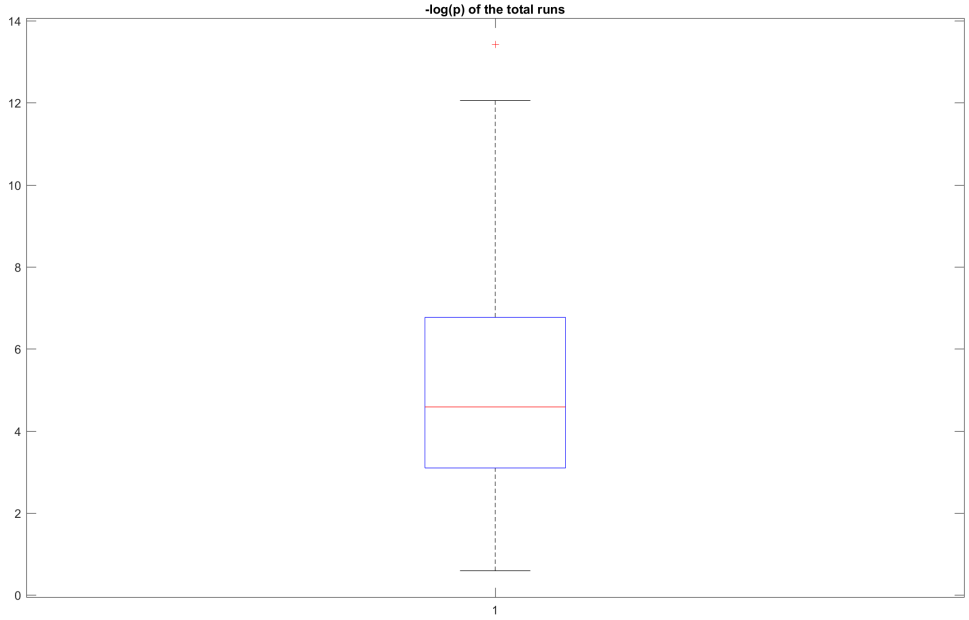


Figure 18: P-values of 100 right tailed t-tests looking at the standard deviation comparing average of four correlated gene expression versus the average of four random gene expressions

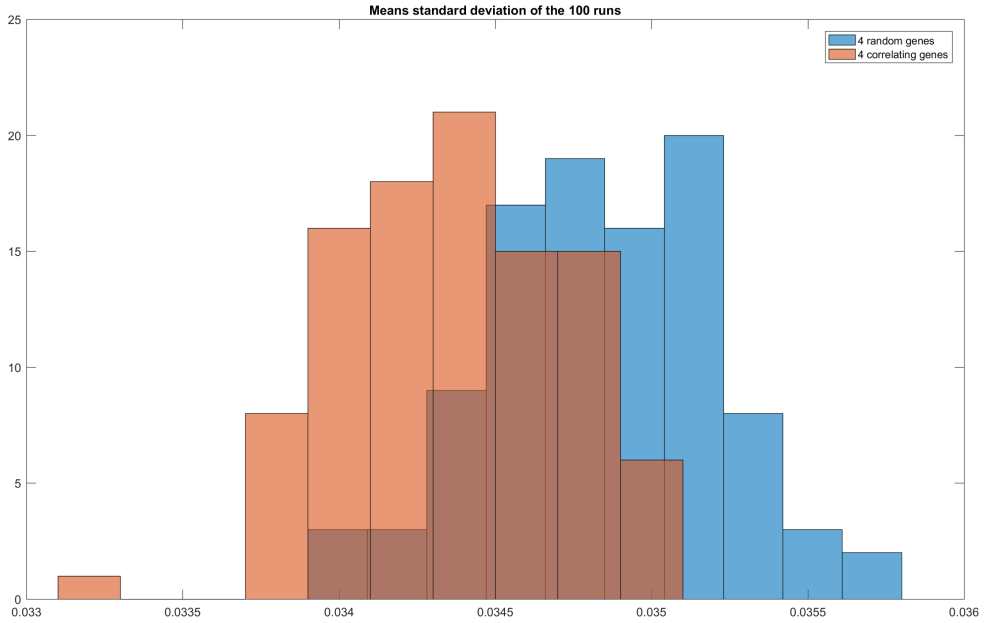


Figure 19: Two distributions of the average standard deviation of the 100 runs

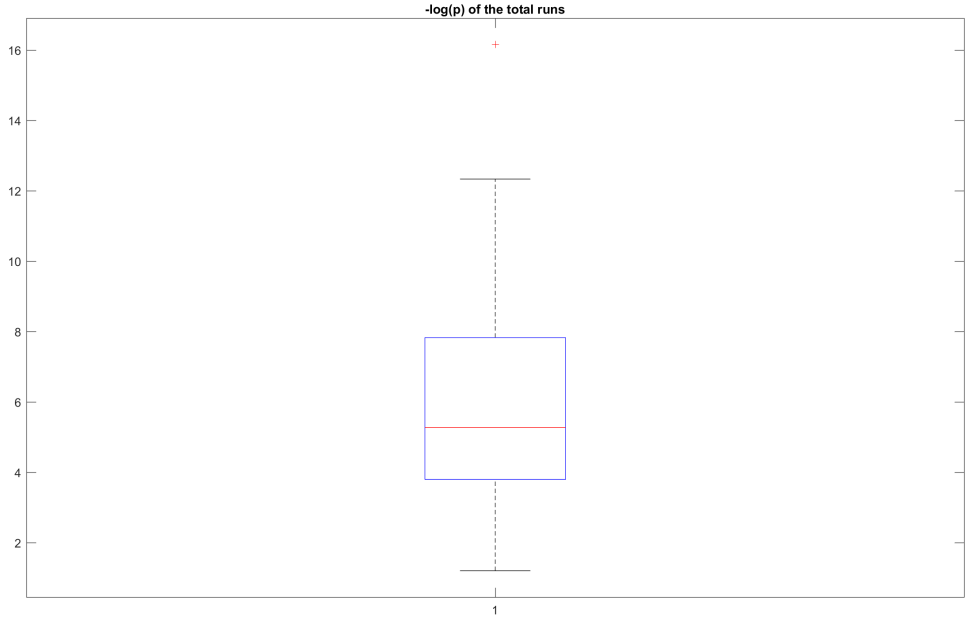


Figure 20: P-values of 100 left tailed t-tests looking at the accuracy comparing average of four correlated gene expression versus the average of four random gene expressions

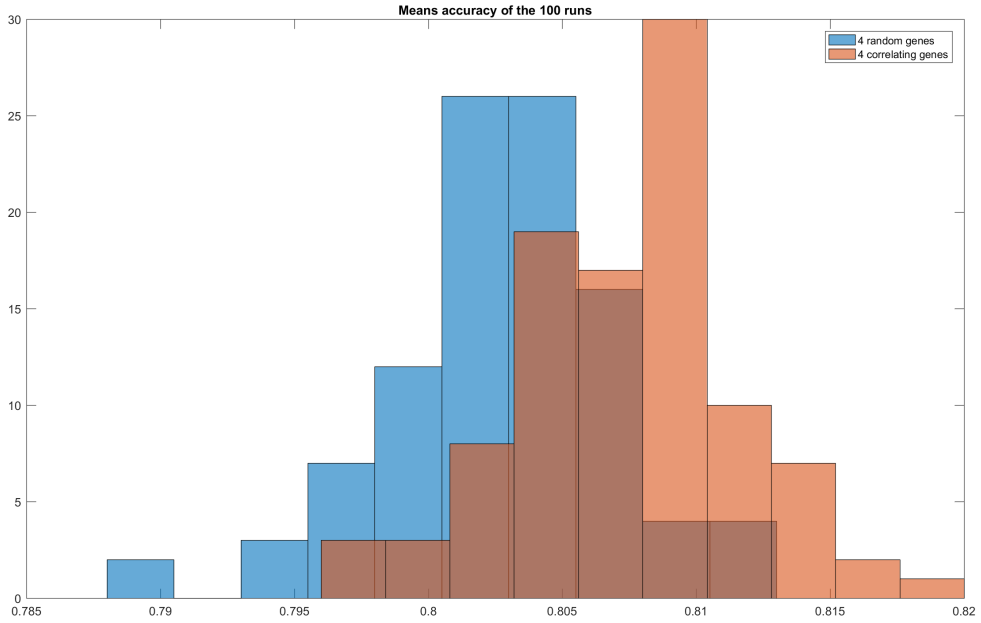


Figure 21: Two distributions representing the average accuracies of the 100 runs

We performed the same test to conclude that this also caused the increased the accuracy when using images created using IME with a convolutional neural network with a max pooling layer, see figures 22, 23, 24 and 25. The test did use the maximum expression instead of averages of the four genes. Here t-test dismissed the null-hypothesis 98 times of the 100 runs meaning that there was an significant improvement in both stability of accuracy as well as accuracy itself

when using correlated gene expressions over the random placement of gene expressions. So the hypothesis does also holds true in the case of using the max of four correlated gene expressions versus the max of four random gene expressions.

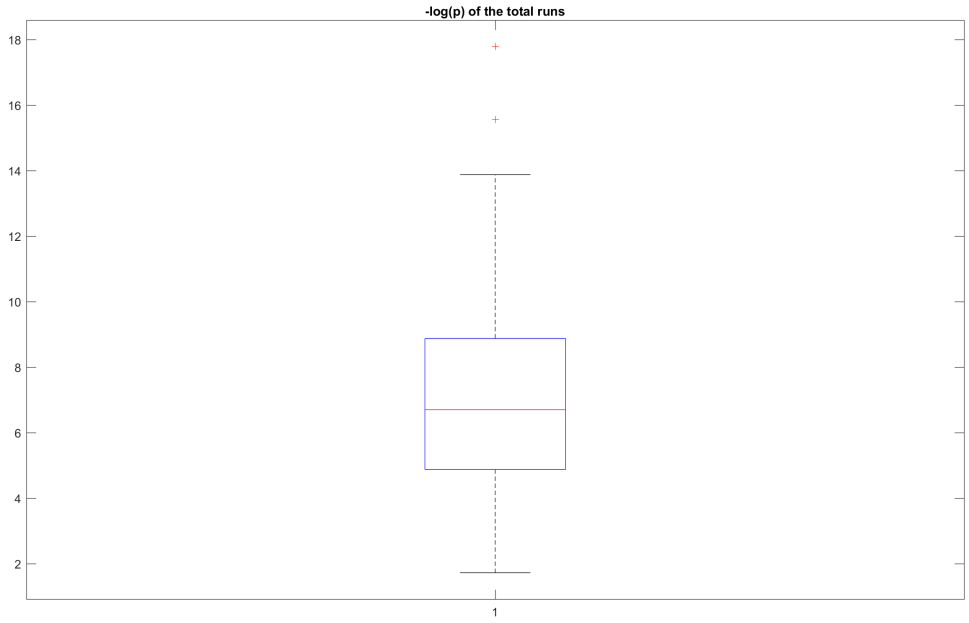


Figure 22: P-values of 100 right tailed t-tests looking at the standard deviation comparing max of four correlated gene expression versus the max of four random gene expressions

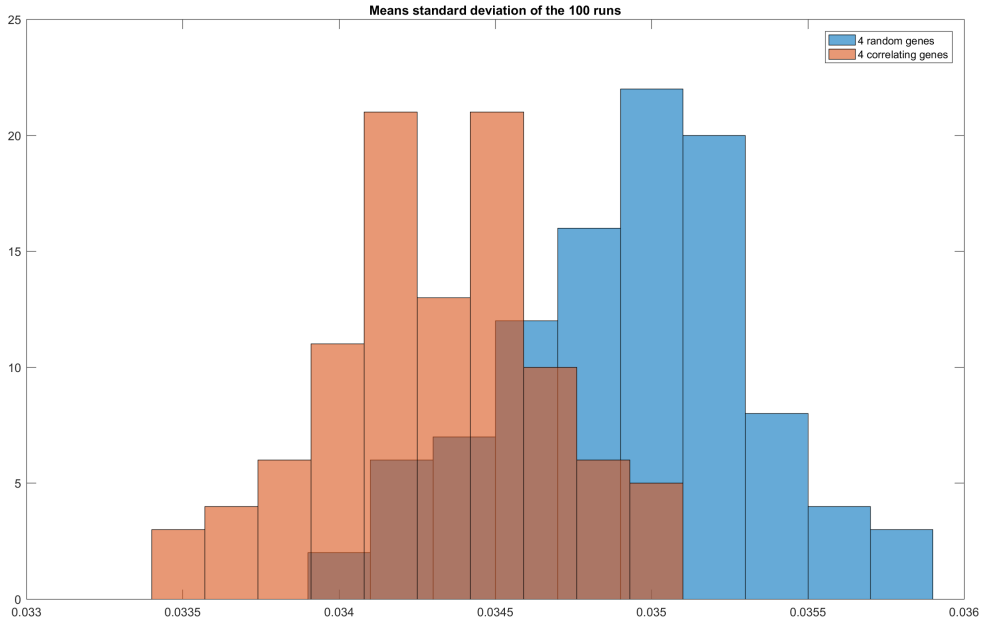


Figure 23: Two distributions of the average standard deviation of the 100 runs

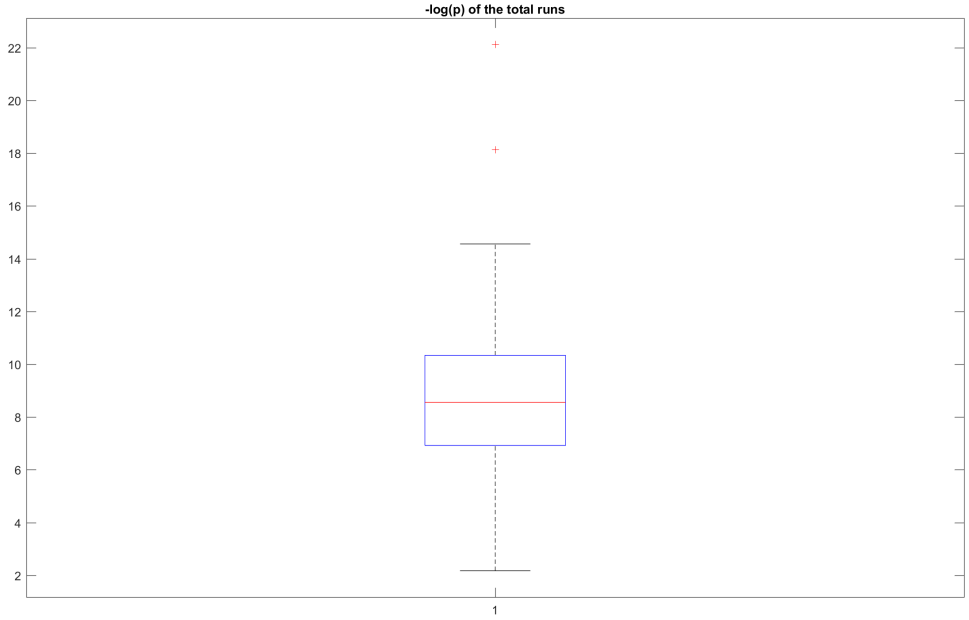


Figure 24: P-values of 100 left tailed t-tests looking at the accuracy comparing max of four correlated gene expression versus the max of four random gene expressions

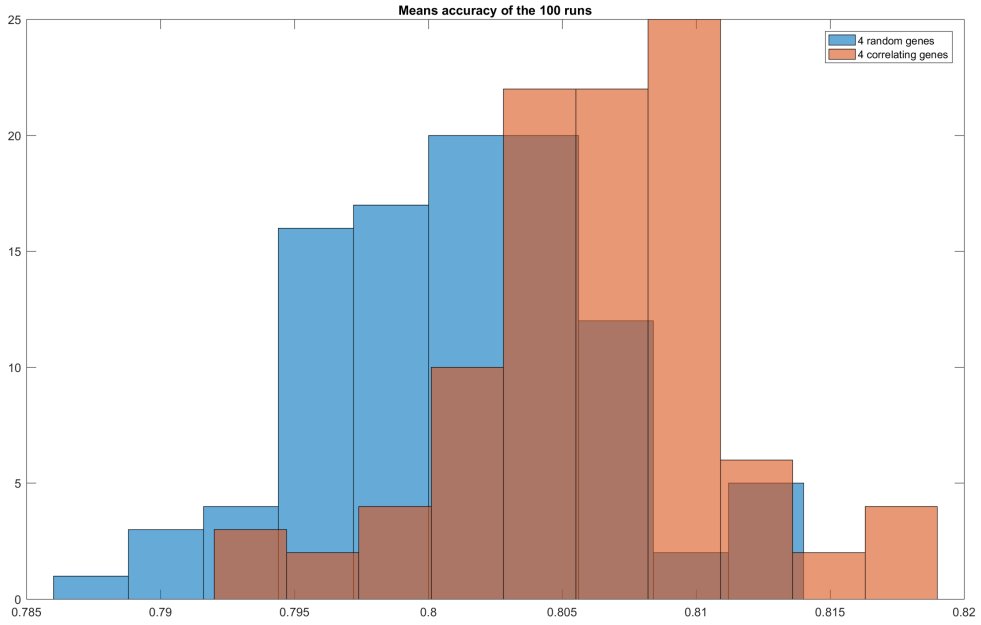


Figure 25: Two distributions representing the average accuracies of the 100 runs

4.7.2 Clustering versus Random

Table 21: Results of the t-test, the corresponding effect-size and accuracy for comparing the Clustering matrix creation with a image with gene expression put random into the matrix

Kernel-size	Avg Clustering	Avg Random	H-value	P-value	Effect-size	P-value RT
1×1	80.17%	80.11%	0	0.5770	0.0477	0
2×1	79.73%	80.14%	1	0.0094	-0.3114	1
3×1	79.78%	80.08%	0	0.1069	-0.2184	0
4×1	79.88%	80.16%	0	0.1236	-0.2272	0
5×1	79.74%	79.98%	0	0.1679	-0.1823	0
9×1	79.61%	79.60%	0	0.9786	0.0106	0
16×1	79.29%	79.57%	0	0.2419	-0.1959	0

Table 22: Results of the t-test, the corresponding effect-size and accuracy for comparing the Clustering matrix creation with a matrix with gene expression put random into it

Kernels	Avg Clustering	Avg Random	H-value	P-value	Effect-size	P-value RT
1	78.71%	79.16%	0	0.1929	-0.2875	0
5	79.35%	79.56%	0	0.1799	-0.1565	0
10	79.63%	80.07%	1	0.0121	-0.3643	1
15	79.86%	79.82%	0	0.8020	0.0314	0
20	79.88%	80.16%	0	0.1236	-0.2272	0
25	79.83%	79.83%	0	1	0	0
30	79.74%	79.99%	0	0.1893	-0.1806	0

Table 23: Results of the t-test, the corresponding effect-size and accuracy for comparing the Clustering matrix creation with a matrix with gene expression put random into it

Feature	Avg Clustering	Avg Random	H-value	P-value	Effect-size	P-value LT
Stride 2×2	79.93%	78.81%	0	0.5454	0.0970	0
Max pooling	75.96%	73.80%	1	9.5832 E-06	1.6610	1
Avg pooling	75.67%	73.68%	1	2.3159 E-06	1.5228	1

When comparing the results, found in tables 21 and 22 we saw that the image created by the Clustering image versus a randomly generated image, on all configurations with a convolutional layer, did not give a significant difference between the accuracies. The reason most likely being that convolutional layer isn't able to find meaningful structures inside the images as discussed in section 4.3 in which case Clustering images can be seen as randomly generated images. The difference between the images created by IME is that the images created by Clustering doesn't contain empty spaces, which cases that Clustering isn't significantly worse in accuracy when compared to randomly generated images. The Clustering images did also start to perform better in the case of a pooling layer as shown in table 23, with the most likely reason being discussed in section 4.7.1.

4.7.3 Correlated images versus Random

In our previous results in section 4.7.1 and 4.7.2 we observe the random images outperforming the other images where highly correlated gene expressions are put near each other. In section 4.3

we also saw that when looking at low level features in multiple correlated gene expressions the accuracy decreased. We therefore proposed the following hypothesis; the positioning of highly correlated genes close to each other has a negative effect on the network its accuracy.

To test this hypothesis we set up an experiment, where we run one network on two different images. The network consists out of a single convolutional layer, with a kernel-size of two by one, and a single output layer with five output neurons with a softmax activation. The network is trained with 40 training iterations over each data set.

The images are build from two pairs of gene expressions. Each pair with one randomly selected gene expression, which we will call A and B, and its most correlated gene expression, which we will call a and b. We then check if all gene expression are different, if they are not, we try it again.

These two pairs are then put in a two by two matrix to be used by the convolutional neural network. The two matrices are representing the two images we would like to create. We call them the 'good'-image, where the highly correlated genes are placed next to each other, and the 'bad'-image, where the highly correlated genes are placed so that there is less of a correlation relation between the neighbours.

In the 'good'-image the gene expressions are placed as: A, a on the first row and b, B on the second, see table 24, so that the most correlated genes are put together and so that they fit in one kernel its input for the convolutional neural network. For the second data set the gene expressions are placed as: A, B on the first row and b, a on the second row, see table 25, so that the correlation relation between the genes are minimal with their neighbours.

Table 24: The 'good'-image

A	a
b	B

Table 25: The 'bad'-image

A	B
b	a

If the correlated data set performs better at classifying the data, then the placing of correlated genes close to each other is valuable, and our hypotheses is rejected. Otherwise, when the uncorrelated data set performs better classifying the data, our hypotheses is accepted.

We generated 1000 images of each type in the way described, and we made sure that each of the images is different. We ran the model on each image once to compute a total of 2000 measured accuracies for the two image types. Using a one tailed t-test on the accuracies, see table 26, we can conclude that the second difference in performance is nil. Also the effect size and the histogram, see figure 26, suggest the 'bad'-image to perform similar to the 'good'-image. Our hypothesis can not be proven or rejected with these results, but we can conclude that placing the correlated genes together does not significantly improve the accuracy of the networks, because the two image types generated give a similar accuracy distribution.

Table 26: Right tailed t-test on the accuracies of the two image types with its effect size.

H-value	P-value	Effect-size
0	0.7778	0.0324

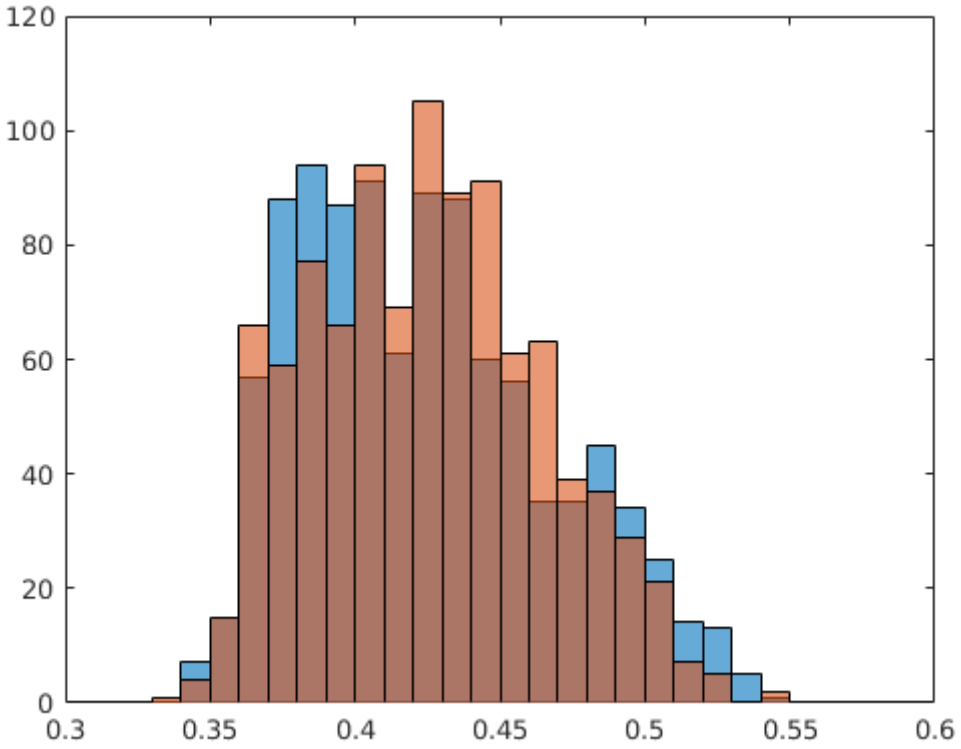


Figure 26: In blue the accuracy distribution of the 'good'-images, see section 4.7.3, in orange the accuracy distribution of the 'bad'-images. With the x-axis the representing the accuracy.

In our previous finding in section 3.2.1 we found that taking averages of correlated gene expressions will reduce the noise, and therefore improve the performance. But with this experiment it can be concluded that putting these gene expressions next to each other does not have the same effect. This may be because the noise of the gene expressions is still there when putting these genes next to each other.

5 Conclusions

We tried predicting breast cancer subtype labels using 100 genes with the highest standard deviation. This set of genes contains 5 of the PAM50 genes used to generate that same label[5]. This most likely means that our results showed higher accuracies than would have been possible with real-world data. However, since we used a baseline and only looked at relative increases and losses in accuracy, this should not affect our results much.

We have shown in section 3.2.1 that by using the average activation of correlated genes we could improve both the overall stability of accuracy of classification as well as accuracy of classification itself, independent of group-size, subtype and the number of correlated genes. With this conclusion we looked into creating images of the gene expression which maintained the correlation to use as input for a convolutional neural network. To represent the correlation in two dimensional space we used t-SNE. Given the large amount of gene expressions the data used in this project were the 100 gene expression with the highest standard deviation.

We developed two techniques to create an image of this representation for use as input in a neural network. The first image creation algorithm we called IME. It adds precision to the image as needed, but introduces many unused pixels. Given the large number of empty pixels we also looked into another way to create the image, which we called Clustering. This algorithm clusters point together based on their closest neighbor which is sorted on the distance to said neighbor each time we create a higher level cluster. This technique creates a one dimensional image of the two dimensional representation generated by t-SNE without empty pixels. While looking at the capability to preserve correlation we found Clustering to perform better than IME. Both techniques were however used for testing different configurations of the neural networks as the dimensionality of the images was different.

We tested the accuracy of classification using the two image creation techniques by creating a simple baseline neural network with a single hidden layer. To optimize the baseline we looked at the optimal number of neurons in this hidden layer. To look at the effectiveness of a convolutional neural network we tested different configurations of such network to determine if an improvement upon the created baseline was possible. To do this we proposed and developed an application which can automatically run multiple network models on different images from all available images and network configurations. The evaluation of the produced networks are then saved to the disk, which are later collected to collect the results.

We used our developed application to look at the impact of changing the number of kernels inside the convolutional layer, changing the the kernel-size, adding a stride of 2 by 2 to the convolutional layer and adding a max or average pooling layer. It was found that a convolutional neural network using our image creations techniques does improve the accuracy only when using a kernel-size of one by one, which causes it not to be an convolutional neural network as described in section 4.3. Here it was discussed a possibility that the data does not contain the kind of structures convolutional neural networks are designed to recognise. The addition of kernels keeps improving the accuracy up to twenty kernels, where adding more kernels after this will most likely cause overfitting. Adding a stride of two by two or a pooling layer to the convolutional neural network was found to cause a decrease in the accuracy of classification. A possible reason for this is that the amount of samples available was rather small, only 1616 samples. Using artificial data synthesis might give more conclusive results, since the neural networks can be trained more thoroughly.

When comparing our image creation techniques to a randomly generated image it was shown that IME performed worse while Clustering performed similarly when using a convolutional layer. Only with the addition of a pooling layer we saw that the two image creation techniques

performed better. The reason for improvement in accuracy when using a pooling layer was shown in section 4.7.1. Here it was found that taking the average of correlated gene expressions produces more stable accuracy as well as increased accuracy when comparing to the average of four random genes. This was also found when taking the max gene expression instead of the average.

In section 4.7.3 we concluded that placing these correlated genes together by the convolutional layer does not improve the accuracy when compared with putting non correlated genes together. Therefore we can conclude that putting highly correlated gene expressions close to each other in the images, might not be beneficial for the accuracy of the convolutional neural network. We also used a linear classifier as configuration of a neural network. This network performed similarly or better than a convolutional neural network on both data representations used while delivering a significant reduction in the amount on neurons.

The goal of this project was finding cancer related gene interaction networks. From our findings we can conclude that method proposed by our client might not work. Mostly because the created images are unlikely to be interpretable by a convolutional neural network. Either another way of creating the images needs to be found, or the use of convolutional neural networks has to be abandoned entirely. There are two main challenges with such images: The images need to contain structures of spatially local correlations that represent gene interaction networks. This might be hard to achieve in the low dimensional spaces convolutional neural networks usually operate on. In addition to this these structures cannot be too similar, since the translational invariant filters of the convolutional neural network will recognise them as being the same. Because of these constraints on the image, we don't see a way to make convolutional neural networks work on this data.

6 Future work

As discussed, the amount of data used might not have been to make clear statements. Therefore it might be interesting to use some artificial data synthesis methods to increase the amount of data and retrain the convolutional neural networks. Doing this could give a more definitive answer as to whether the described methods work or not.

However, using our findings as an indication, we advise using different (deep learning) methods to detect cancer related subnetworks, such as deep belief networks and deep Boltzmann machines. These are examples of unsupervised learning. As such, they create their own representations of the data. This might be a more appropriate approach than fixing the representation of the data in a manner similar to the way we did in this project. Then, using these learned representations, we can again use a neural network to see which clusters are cancer related. It might also be interesting to use only restricted Boltzmann machines, then perform classification with a linear classifier on those. Although high level abstractions are not created, and therefore interactions could be missed, this should be more easily interpretable, since the clusters directly correspond with groups of genes.

References

[1] Aron Goldhirsch et al. "Personalizing the treatment of women with early breast cancer: highlights of the St Gallen International Expert Consensus on the Primary Therapy of Early Breast Cancer 2013." In: *Annals of Oncology* 24.9 (2013), pp. 2206–2223.

[2] MJ Horner et al. *SEER Cancer Statistics Review, 1975-2006, National Cancer Institute. Bethesda, MD*. 2009.

[3] Carol E DeSantis et al. "Cancer treatment and survivorship statistics, 2014." In: *CA: a cancer journal for clinicians* 64.4 (2014), pp. 252–271.

[4] C La Vecchia et al. "Cancer mortality in Europe, 2000–2004, and an overview of trends since 1975." In: *Annals of Oncology* (2009), mdp530.

[5] A Prat et al. "PAM50 assay and the three-gene model for identifying the major and clinically relevant molecular subtypes of breast cancer." In: *Breast cancer research and treatment* 135.1 (2012), pp. 301–306.

[6] Joel S Parker et al. "Supervised risk predictor of breast cancer based on intrinsic subtypes." In: *Journal of clinical oncology* 27.8 (2009), pp. 1160–1167.

[7] Aleix Prat et al. "Clinical implications of the intrinsic molecular subtypes of breast cancer." In: *The Breast* 24 (2015), S26–S35.

[8] Christine Staiger et al. "Current composite-feature classification methods do not outperform simple single-genes classifiers in breast cancer prognosis." In: *Quantitative Assessment and Validation of Network Inference Methods in Bioinformatics* (2015), p. 84.

[9] Laurens Van der Maaten and Geoffrey Hinton. "Visualizing data using t-SNE." In: *Journal of Machine Learning Research* 9.2579-2605 (2008), p. 85.

[10] Pavel Berkhin. "A survey of clustering data mining techniques." In: *Grouping multidimensional data*. Springer, 2006, pp. 25–71.

[11] Mark A Larkin et al. "Clustal W and Clustal X version 2.0." In: *Bioinformatics* 23.21 (2007), pp. 2947–2948.

[12] Michael Gamon. "Sentiment classification on customer feedback data: noisy data, large feature vectors, and the role of linguistic analysis." In: *Proceedings of the 20th international conference on Computational Linguistics*. Association for Computational Linguistics. 2004, p. 841.

[13] Motoo Kimura and George H Weiss. "The stepping stone model of population structure and the decrease of genetic correlation with distance." In: *Genetics* 49.4 (1964), p. 561.

[14] indico. *Visualizing with t-SNE*. 2015. URL: `https://indico.io/blog/visualizing-with-t-sne/` (visited on 05/02/2016).

[15] scikit-learn developers. *Neural Network Models (supervised)*. 2016. URL: `http://scikit-learn.org/dev/modules/neural_networks_supervised.html` (visited on 05/02/2016).

[16] Michael A Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015. URL: `neuralnetworksanddeeplearning.com`.

[17] Geoffrey E Hinton. "Training products of experts by minimizing contrastive divergence." In: *Neural computation* 14.8 (2002), pp. 1771–1800.

[18] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. "A fast learning algorithm for deep belief nets." In: *Neural computation* 18.7 (2006), pp. 1527–1554.

[19] Ruslan Salakhutdinov and Geoffrey E Hinton. "Deep boltzmann machines." In: *International conference on artificial intelligence and statistics*. 2009, pp. 448–455.

[20] Tomas Mikolov et al. “Distributed representations of words and phrases and their compositionality.” In: *Advances in neural information processing systems*. 2013, pp. 3111–3119.

[21] Ehsaneddin Asgari and Mohammad RK Mofrad. “Continuous Distributed Representation of Biological Sequences for Deep Proteomics and Genomics.” In: *PloS one* 10.11 (2015), e0141287.

Appendices

A Research report

A.1 Introduction

In this research report we will discuss techniques we consider using for the identification of cancer related sub-networks of genes. These techniques can be divided up in three distinct classes, namely techniques for representing correlation, techniques for expressing these correlations into a matrix and techniques for deep learning. Our client requested us to use t-SNE and Convolutional Neural Networks to classify cancer types from correlation data between genes and labeled human genome data.

t-SNE will be used to generate a two dimensional representation of the n-dimensional correlation data. The dimensionality reduced data should then be cast into a matrix, so it can be used to train and predict with a Convolutional Neural Network. Our assignment is to find an effective conversion from a point-cloud into a matrix, while keeping as much data as possible.

First, in chapter 3.2.1 we will discuss how the correlation can be expressed in two dimensions, where we mainly discuss what t-SNE is and does for our project. In chapter A.3 we discuss several ideas about the conversion from the point-cloud data to matrices. We need this data in matrix form so that we can use a Convolutional Neural Network as discussed in chapter A.4.2. Here we discuss what a Convolutional Neural Network is, how it could be used and our doubts about its effectiveness. Because of these doubts we will also explore other neural network types in chapter A.4, searching for alternatives.

A.2 Representing correlation

About the functioning of human genes a lot is still unknown. We do however know that there are correlations between certain genes close to each other [13] which means that if the productivity of one gene decreases, so does the productivity of the correlated genes.
The genetic data that was provided to us is contained in a Matlab database. Matlab itself has many build in function specialized on matrix operations. The function $corr(X)$ returns a p-by-p matrix containing the pairwise linear correlation coefficient between each pair of columns in the n-by-p matrix X. This is more often called the pairwise similarity matrix which is a efficient way to represent the correlation between the genes. Other techniques we could use meant first exporting our given data to a different environment causing a greater overhead so we choose to use the build-in function of Matlab. We also chose not select all the genes to use for the creation of this matrix as it would slow our system down significantly by looking through uninteresting data. We selected the 100 genes with the highest standard deviation as starting point because these genes are behaving abnormal and therefore we suspect they are interesting for classification.

t-Distributed Stochastic Neighbor Embedding (t-SNE) is a technique for dimensionality reduction that is particularly well suited for the visualization of high-dimensional datasets. The technique is a variation of Stochastic Neighbor Embedding but improved in producing a single map of high dimensional data and visualization of this data [9]. Given our high-dimensional representation of the DNA in the pairwise similarity matrix this technique provides a way to reduce our data to a 2 dimensional image of points representing the gene expressions where the correlation between the genes is maintained. Figure 27 shows the result of using t-SNE on the MNIST database which gives a two dimensional representation.

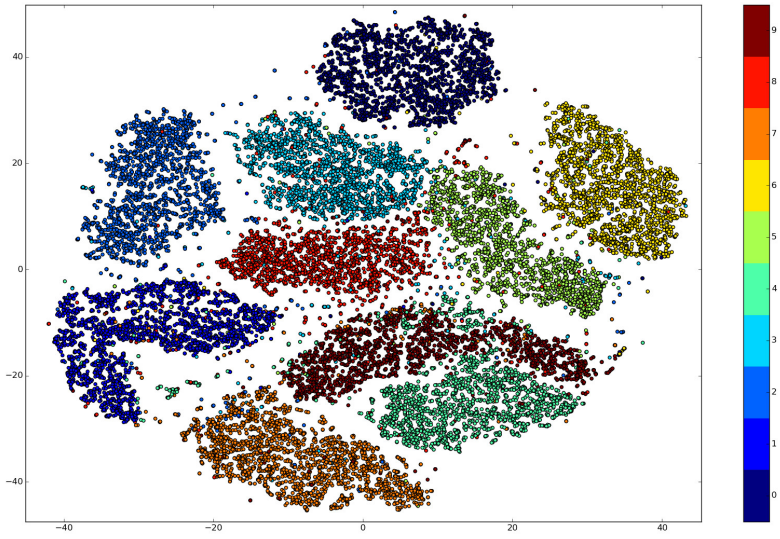


Figure 27: Representation of the MNIST database [14]

Using an existing Matlab implementation, we get a set of two dimensional points where distances between points represent the correlation. t-SNE creates this two dimensional representation of correlations by preserving local distances. Therefore, we can make no assumptions on correlations of points far away from each other. This also means the triangle inequality is not preserved. If for example two points are both 10 centimeter away from another point which has neighbors within 1 centimeter we cannot conclude that the correlation between the two points and the other point are the same. The existing t-SNE implementation accepts a pairwise similarity matrix, however we have computed correlations. Since a highly negatively correlated pair of variables might be interesting to look at together, we don't want t-SNE to pull them apart. Therefore we might want to take the absolute value of the correlation and run t-SNE on that.

The implementations of t-SNE in Java were found to be not suitable for this project as it lacked the support of taking a pairwise similarity matrix as an input. This was also the case for the implementation in *Python*. Another reason for choosing the Matlab implementation that further processing of the data is needed. We want to to use Matlab for transforming the dimensionality reduced data to a 2×2 matrix, suitable for interpretation by a CNN. Having the t-SNE implementation in Matlab means that intermediate exporting the data isn't necessary and all the calculations to create the image used for deep learning could be done in one single environment. The gene correlation of the first 100 genes obtained by the technique found in section 3.2.1 is represented on the next page.

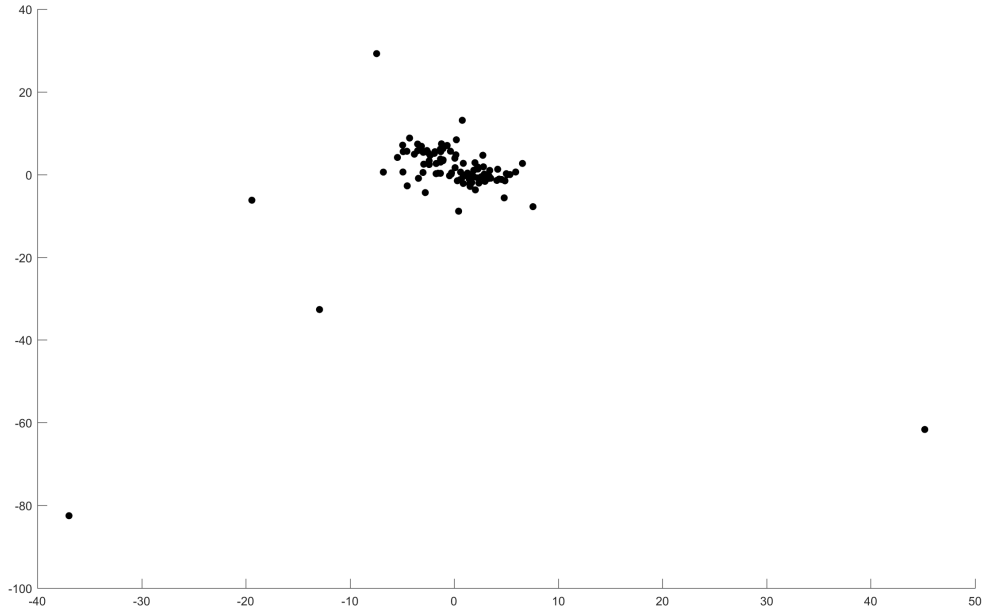


Figure 28: Gene correlation representation using t-SNE.

A.3 Expressing correlations as matrix

We want to run a convolutional neural network on the results given by the t-SNE algorithm. In order to do this, we need to define which inputs are given to each neuron. Unfortunately this is not trivial for the result given by t-SNE. Convolutional neural networks require images as input. However, t-SNE gives a set of points in 2D space. So, after using t-SNE to get a 2D representation of the correlations, we have to put that into a matrix form. Ideally this image would capture all spatial relationships between the points in the t-SNE result. Practically, however, this results in images that are too large to process. This and the fact that t-SNE only preserves probabilities of points being close together, suggests that we should only care about putting together points that are close together in the t-SNE result regardless of orientation and relative positions on larger scales.

Because our ideal image is not practical several alternatives of constructing this matrix are considered. One of these methods could be called 'the average method'. The average method would distribute the points over a fixed reasonable size matrix, so that multiple points will share a matrix cell. Where this happens, the mean of these points should be used instead of the value of a single point.

t-SNE generates big compact clusters with the given correlation data, which could lead to a huge loss of information when these clusters are mostly averaged. The quantity of data lost is related to the matrix size. By increasing the matrix size, the input size is increased, but the chance for points to be positioned in the same cell is reduced. If we would like to use this method, we should weigh the information loss against the required training time. By using the average method a cell could lose important information about the more rare inactivity or activity of a gene. When a minimum or maximum method is applied on the same data, one of the extremes can be preserved. Both can be interesting for the cancer classification. However, it is unknown for each cell what value is the most valuable, as this could differ from cell to cell and for each type of cancer.

One possible solution could be creating an image of both where points collide, making our data three dimensional. This extra dimension should have a fixed size because of the later Convolutional Layer requires a fixed amount of weights. So we can't stack all points in a cell. It is, however, possible to put some important points in there, as the maximum, minimum and average value.

Another option entitled iterative matrix expansion (IME), would be to insert columns and rows in the matrix after the initialisation, depending on the amount of points mapped to the same location. The most simple way of doing this would be iteratively adding points to a matrix M, starting with $M = 1 \times 1$. Adding a new row or column for every pair of data points mapped to the same place in M. The points are split along the most discriminating axis and a new row or column is added. This is a method that, as described above tries to preserve most spatial relationships. Again, as stated above it returns a large sparse matrix of 600+ $\times$ 600+, which is not practical since we only have 12750 values of gene activation per person. Even if we select the first 100 genes by the technique found in section 3.2.1 we still get a matrix of 20+ by 20+, meaning four times as large as necessary. This could give a matrix as found in Figure 29.

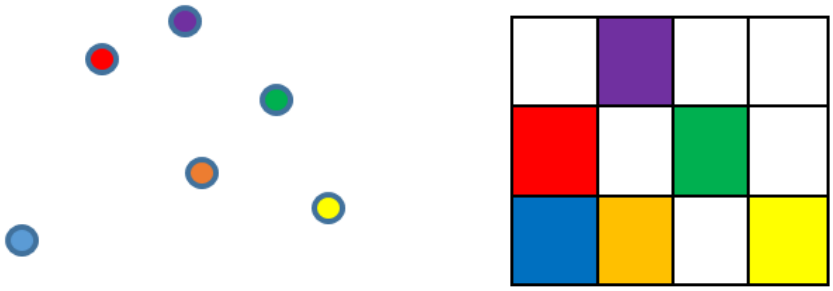


Figure 29: An example matrix representation of the gene correlation from the t-SNE point cloud

A.3.1 The Nearest Neighbours Method

In this method a matrix is build by unpacking a tree, where nodes in this tree are held together as much as possible. This tree is build from the two dimensional representation produced by t-SNE, by clustering points in the point-cloud and clustering these clusters over and over again until a single cluster remains. By doing so we hope to save as much information as possible.

Because only the local distances are are maintained within the two dimensional representation generated by t-SNE, we only have to protect the short distances within the two dimensional representation. Therefore we can put all points within a cluster in any order, as long points close together are within the same cluster. The clustering of larger clusters is therefore also no problem, because there clusters should already be further apart of each other, which implies no correlation between the clusters. Therefore we can create a dense matrix with a dimension of two times the square root of the number of points.

This method contains two parts; the building of the tree, where the point cloud is converted into a tree, and the filling of the matrix, where the tree is used to fill the matrix with the given points/genes.

In algorithm 1 the pseudo code of the first half of this algorithm is displayed, where the tree is build. In this pseudo code we call the clusters of points groups, which contain all points of the groups it contains. When our initial groups are set, we loop until there is only a single group left, and therefore a single tree. In this loop it computes the closest groups for each group, after which this is used to put the groups in buckets, clustering the close groups together. At the end of each iteration, these buckets are emptied into new groups, starting this algorithm over again. This technique is similar to linkage clustering and below we will discuss how we plan to use this technique with additions fit to our use case.

One of the biggest cost within the while loop is the computation of all closest points within the group, which can be computed with a time complexity of $O(n^2 log(n))$. We should consider computing these distance as distance between the group centers or between the closest points within the group. Computing the closest points withing the group could give us the most accurate tree, saving more information then the other method. However, this could lead to large clusters absorbing all the small clusters that could unbalance the tree. When the method of computing the distance from the group center is used, larger groups will create a lot of distance between itself and other groups, and therefore creating room for smaller groups to combine.

In this pseudo code a possible threshold is mentioned, which could be used to exclude distant groups from clustering, so that groups won't expand too fast. This could be beneficial when using the earlier described center distance computation, preventing the group its center from jumping when there are nearby groups to merge with. The usage of a threshold is required if we want to add multiple groups within the same bucket, because we could get a single big bucket in the worst case. This threshold could be computed as the average smallest distance for all groups times a scalar.

An other option would be to only combine pairs of groups. By starting with the closest pair of points this option will group the points that have the highest correlation always together, making these two groups will be placed exactly next to each other in the next part of this algorithm. In the case of combining group we should look at the mean of the points in the group to avoid chaining groups together with ever increasing distance between them.

The idea of combining the groups in pairs could be extended by combining the groups in other numbers by limiting the bucket size. This will force creating smaller groups, preventing all points from gathering in a single group on the first iteration. Smaller groups will have the same advantages of information preservation as the pairs, but possibly in a smaller extend.

The filling of the matrix can be done by simply recursively taking each leaf of the tree and row for row fill the matrix. A downside of using this implementation will be that when we change the row two possibly higly correlating points will be taken apart. To combat this we will implement two techniques to fix this problem. The first will be taking averages of groups to fill the matrix. This will make the matrix smaller as a single cell will represent multiple genes, but the higly correlating genes are still together in this case. Another solution would be creating a one dimensional matrix, to will preserve all the correlation between points but means that the convolutional neural network can not use a pooling layer in two dimensions.

Algorithm 1 Building the tree

```
define an empty group list
make a group for each point in the point cloud add them to the group list
while there is more than one group do
    for all groups g_i compute the closest group g_j to group g_i;
    set a threshold;
    for each group g_i do
        if g_i its closest group g_j its distance is less or equal then the threshold then
            if g_i and g_j are in the same bucket then
                continue
            else if g_i xor g_j are in a bucket then
                add g_i and g_j to that bucket
            else if g_i and g_j are in different buckets then
                combine the two buckets
            else
                create a new bucket and add both groups
        elsecontinue
    empty the group list
    for each bucket do
        combine all groups within the bucket into a new group and add them to the group list
```

A.3.1.1 Filling the matrix

Algorithm 2 describes the second part of this algorithm which computes a matrix from the computed tree. It is a recursive algorithm that produces a matrix by dividing the available space within the matrix, positioning the points in the matrix once it reaches the leaves of the tree. The point position could be taken into account during the reserving of space while splitting and placing of points once it reaches the leaves. Doing this could make sure that closely correlated groups are located next to its most correlated groups, in an attempt to retain more information.

Algorithm 2 Matrix Filling

```
INPUT: a tree of clusters with as leafs points
C ← tree
M ← a matrix with the smallest size that fits all points
divideSpace(C, M)
return M

function DIVIDESPACE(Tree, Matrix)
    INPUT: the tree of clusters to be added to the matrix,
    the (sub)matrix to add the points to
    if a cluster is a leaf then
        draw the point on its space on the matrix
    else
        divide the matrix so that each inner cluster in the cluster has enough space
        for all the points it contains
        for all inner clusters within the cluster do
            divideSpace(inner_cluster)
```

A.3.2 Matrix Neighbour Embedding

Another way of making a matrix out of the result given by t-SNE is m-NE. The name we lent from t-SNE itself. Given a list of points, m-NE tries to embed each in the same relative location within a matrix.

m-NE starts by placing some point inside a matrix. It then takes the closest point to that initial point (or those points already added) in the original data, and places it as close as possible in the matrix. At first this involves only putting the second point in either of $M_{i-1\,j}, M_{i+1\,j}, M_{i\,j-1}, M_{i\,j+1}$ if $M_{i\,j}$ is the initial point. In general we can say that the next point to be added, or in other words, the closest non-mapped point to the mapped cluster should be placed in the matrix only in one of $M_{i-1\,j}, M_{i+1\,j}, M_{i\,j-1}, M_{i\,j+1}$ if $M_{i\,j}$ is a filled entry in the matrix. This reduces the search space for embedding the point inside the matrix. We can then take all of these viable spaces and define a fitness function for them.

$$f(p, p_M, C, M) = \sum_{i=1}^{|M|=|C|} d(p_M, M_i) \cdot \frac{\sum_{c \in C} d(p, c)}{d(p, C_i)} \tag{1}$$

In this equation p represents the point to be added to the matrix, p_M its potential representation in M. C is the cluster of visited points, M is the matrix and M_i is the representation of C_i (the ith visited point) in M. With this equation we scale the distances in M by the 'closeness' in the t-SNE data; that is, closer points are scaled up more than farther points. The sum of scaled distances determines the fitness of the potential location, lower being better. So to select the right p_M we have to minimise over their values of f.

$$m(p, C, M) = \min_{\forall p_M \in M} \left\{ \sum_{i=1}^{|M|=|C|} d(p_M, M_i) \cdot \frac{\sum_{c \in C} d(p, c)}{d(p, C_i)} \right\} = \min_{\forall p_M \in M} \{f(p, p_M, C, M)\} \tag{2}$$

A.4 Neural Networks

The created image of the gene expression will be used in a neural network for training. Given the high complexity of the gene expression and the experimental state of the use of machine learning for identification on this kind of image we will consider many different Neural Networks to test which will be the give the most optimal identification. We will describe the networks we would like to use and which we think will produce the best result. Given that we will use libraries for the creation of neural networks we would like to find the most promising and not focus on one single network. The choice of programming language and/or library can be different for each network, as we will look at the most efficient existing implementation each time.

A.4.1 Single Hidden Layer Neural Network

To get a first result even before the use of a convolutional neural network we will first fed the created matrix into a single hidden layer neural network. This simple neural network has the same number of input neurons as the number of cells in the matrix. These input neurons will be the activation of each of the gene and each input neuron will be connected with each neuron in the following hidden layer. This hidden layer has as number of neurons the number as the number of labels. By setting weights on each of the incoming edge the network tries to classify the data. The activation function of these neuron was the hyperbolic tangent function, which determines when the neuron should be activated giving the prediction of the label. The final output layer of our network has a single neuron and used the softmax function to classify the matrices. This function will look at which neuron in the hidden layer is activated most, meaning that this neuron that represents a label has highest change of being the correct label. In the image n would be the number of genes and k the number of labels associated with the patients.

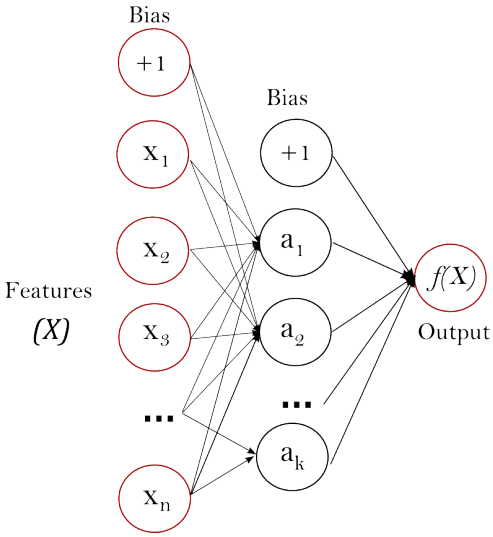


Figure 30: Simple Neural Network [15]

Given this simple network a image with the correlations of the genes isn't as important, so the image we will create can have each kernel representing a random gene. This is easily created in Matlab and will give us a first indication about how well machine learning can be used for identification. The most efficient implementation found to be the Deeplearning4j library with can be found at deeplearning4j.org. This library has support for GPU acceleration, many existing networks as example and easy evaluation tools regarding precision of the classification.

A.4.2 Convolutional Neural Network

Our customer and coach requested the usage of a CNN (Convolutional Neural Network). A CNN is often used to classify multiple objects within a larger image. In this project we will try to visualizes gene expressions with t-SNE, which will create a large image and train a CNN on it to recognize the different cancer types with it. CNN's work with multiple layer types: The convolutional layer, the optional pooling layer, the optional ReLU layer and the fully connected layer, see figure 31.

The first layer in the CNN, the layer that connects the input data with the network, is the convolutional layer. This layer can be represented as a cube of neurons, with a limited width and height which cover a part of the input data, with one input source to on neuron. This cube is used on multiple positions on the input data, using the same weights. In the depth of this cube are multiple filters, also called kernels, that are trained to recognize different patterns or features. This layer is trained to recognize different patterns within the data, with the number of kernels as the number of different patterns. This could be used, for example, to recognize corners, edges and plains.

This layer is often connected with a pooling layer, which summarizes the results of the previous layer. Often the max function is used in the pooling layer, which determines if a feature is found or not within the range of a individual pooling neuron. The often used relu layer is 1:1

connected with the pooling layer and applies a activation function on its input data, increasing its nonlinear properties of the decision. The fully connected layer These layers are often stacked multiple times, but finally ends in the last layer of the CNN, which is a fully connected layer. The fully connected layer connects all outputs of the previous layer, and may contain several hidden layers of its own.

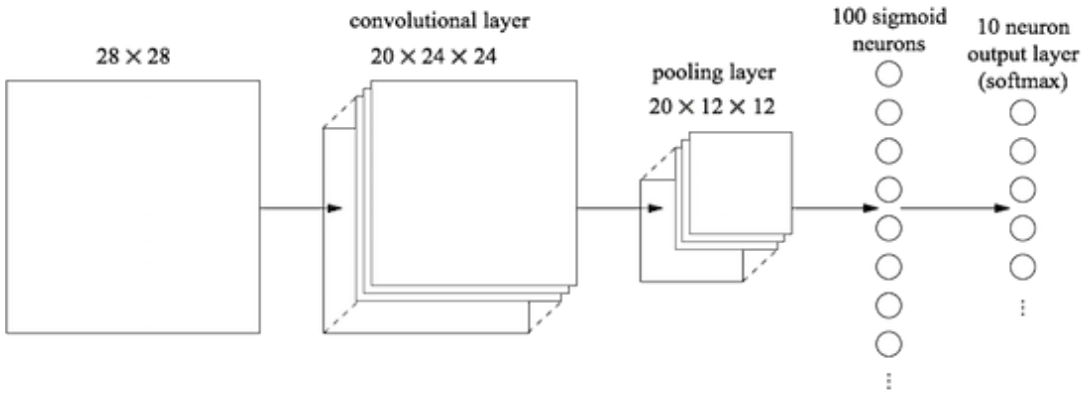


Figure 31: A diagram of a simple convolutional network [16]

A.4.3 Deep Belief Network

We also want to look into using other deep learning networks. The first interesting network found was a Deep Belief Network. This network uses Restricted Boltzmann machines in its layers which uses contrastive divergence learning procedure to train [17]. Restrictive Boltzmann machines have no connections from hidden layers to hidden layers or from visible layer to visible layers [17]. The hidden layers can then be trained by using the outcome of one RBM as the input for training of a higher-level RBM [18]. This greedy, layer-for-layer technique of learning doesn't result in a multilayer Boltzmann machine [18][19], but in the Deep Belief Network we would like to use first for identification.

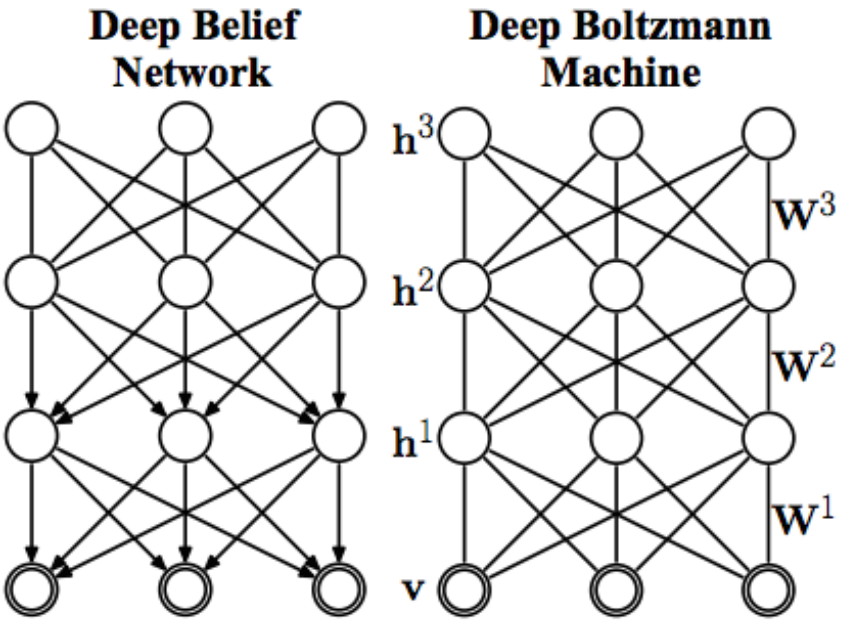
This network, just as a convolutional neural network, can be trained in an unsupervised way, meaning that the input is unlabeled, to reconstruct certain features in the data. After training on this subset of data the network can be further trained in a supervised way to perform classification [18]. We think this network will produce higher precision than the convolutional neural network but there is a slight adjustment of this network given even higher efficiency we will also use [19].

A.4.4 Deep Boltzmann machine

A variation of the Deep Belief Network discussed earlier is the Deep Boltzmann machine. This technique still uses restricted Boltzmann machines whom are multilayered but has some important differences. First some key similarities between the two; both can learn to reconstruct certain features in the input data, both can train in an unsupervised way and both can than further trained in a supervised way to perform classification [19].

The difference between a Deep Belief Network and a Deep Boltzmann machine is that the approximate inference procedure and the initial bottom-up pass can incorporate top-down feedback allowing a more robust way to allow ambiguous input [19]. The use of this network on the MNIST dataset was found to be more precise than Deep Belief Networks [19] making it a

promising network for classifying our data. The top-down feedback is represented in the image below by the fact that the edges from the hidden layers are undirected.



[19]

Figure 32: A Deep Belief Network and a Deep Boltmann Machine compared

A.5 GeneVec

A totally different way to try identify cancer related sub-networks of genes is to first represent gene sequences as vectors. This is comparable to the Word2vec technique used by many. This technique is not a deep neural net as there are no hidden layers. The word2vec neural net makes from each word a vector which a deep neural network can use [20]. The deep neural network is trained to find words that are similar to each other, which has many use cases such as auto-correct used when typing. Given that these vectors can result in better classification of families [21] they could be used as input data for the t-SNE technique. These vectors are defined just recently in 2015 so there isn't a lot of sample data to use. This means we will have to create these vectors ourselves causing it to take a lot of time to implement. This is why we will only look into this technique if we have all the results of the other techniques described above.

B Correlation

B.1 Algorithm

Algorithm 3 Algorithm to prove a more stable accuracy with averages

$GE \leftarrow$ gene expressions of all the patients
$lab \leftarrow (label == 3)$
$numRuns \leftarrow 100$
$numGenes \leftarrow 100$
$numGroups \leftarrow 101$
$numPatients \leftarrow 16$
$h_{array} \leftarrow$ array of size $numRuns$
$p_{array} \leftarrow$ array of size $numRuns$
$totalMean_1 \leftarrow$ array of size $numRuns$
$totalMean_{avg} \leftarrow$ array of size $numRuns$
$totalAcc_1 \leftarrow$ array of size $numRuns$
$totalAcc_{avg} \leftarrow$ array of size $numRuns$
for 1 to $numRuns$ **do**
    $std_1 \leftarrow$ array containing all the standard deviations of one gene
    $std_{avg} \leftarrow$ array containing all the standard deviations of the average
    $acc_1 \leftarrow$ array containing all the accuracies of one gene
    $acc_{avg} \leftarrow$ array containing all the accuracies of the average
    for 1 to $numGenes$ **do**
        $g1_{act} \leftarrow$ gene expression of one random gene
        $gAVG_{act} \leftarrow$ gene expression average of four correlated gene with $g1$.
        $group1_{acc} \leftarrow$ array of accuracies of each group of one gene
        $groupAVG_{acc} \leftarrow$ array of accuracies of each group of the average
        for 1 to $numGroups$ **do**
            $group1_{act} \leftarrow$ subset of $g1_{act}$ of size $numPatients$
            $groupAVG_{act} \leftarrow$ subset of $gAVG_{act}$ of size $numPatients$
            Calculate highest accuracy for both subsets
            add highest accuracy of $group1_{act}$ to $group1_{acc}$
            add highest accuracy of $groupAVG_{act}$ to $groupAVG_{acc}$
        add standard deviation of $group1_{acc}$ to std_1
        add standard deviation of $groupAVG_{acc}$ to std_{avg}.
        add the mean of $group1_{acc}$ to acc_1
        add the mean of $groupAVG_{acc}$ to acc_{avg}.
    Run t-test on std_1 and std_{avg} and add the resulting h to h_{array} and p to p_{array}
    add std_1 to $totalMean_1$
    add std_{avg} to $totalMean_{avg}$
    add acc_1 to $totalAcc_1$
    add acc_{avg} to $totalAcc_{avg}$
return $sum(h_{array})/size(h_{array})$ and $mean(totalMean_1)$ and $mean(totalMean_{avg})$

B.2 Results

B.2.1 Difference in accuracy by selecting more correlated genes to calculate the average activation

B.2.1.1 Average of 9 correlated genes

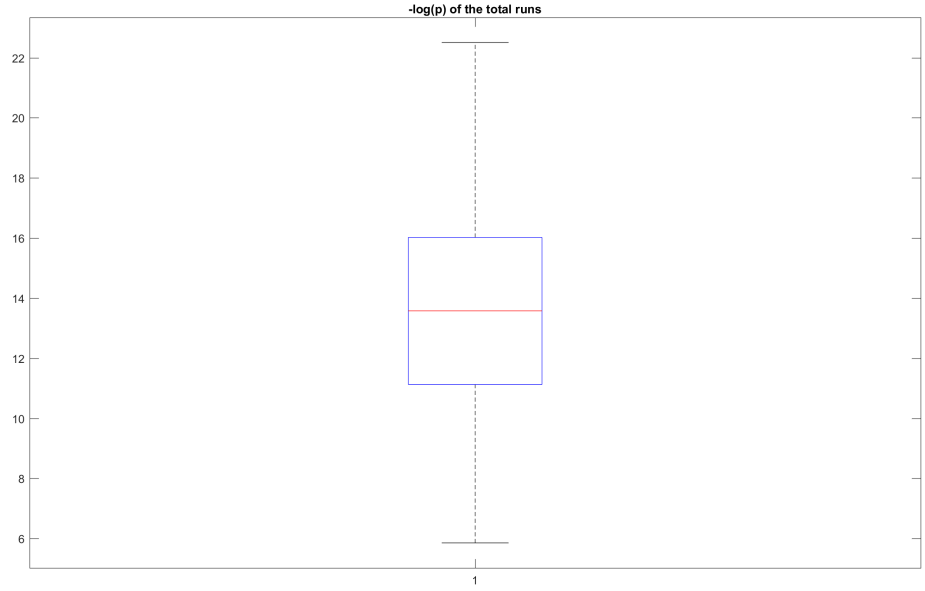


Figure 33: P-values of 100 right tail t-tests looking at the standard deviation

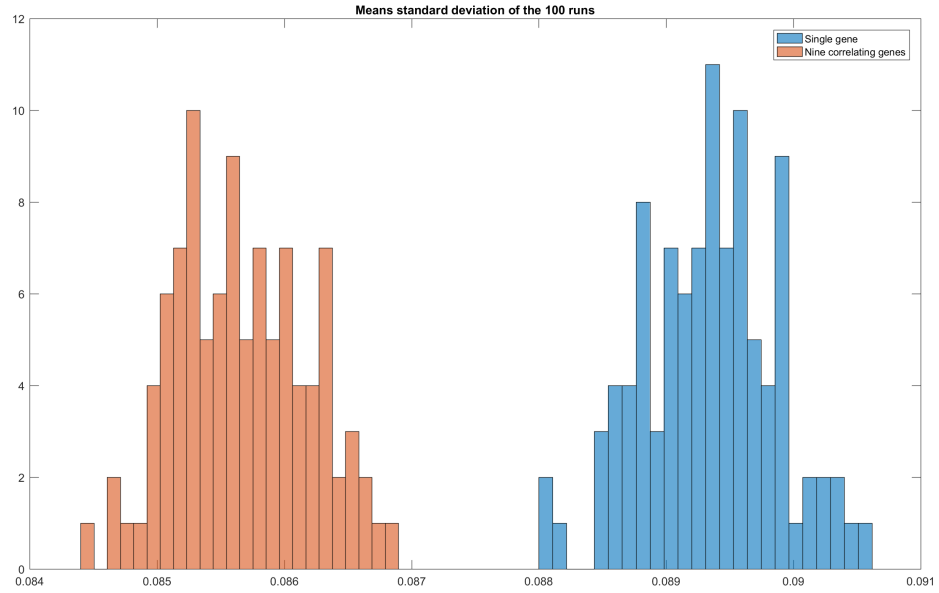


Figure 34: Two distributions of the average standard deviation

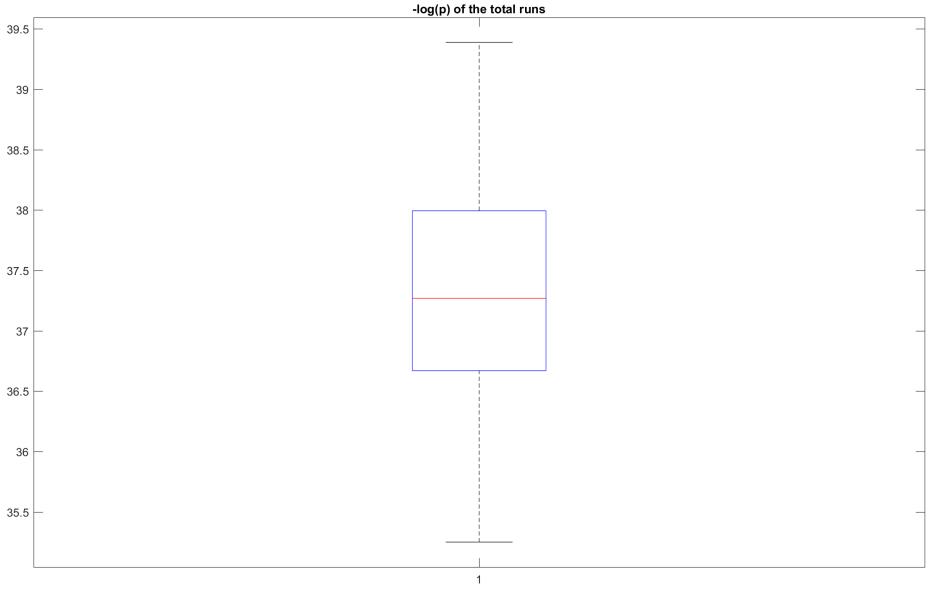


Figure 35: P-values of 100 left tail t-tests looking at the accuracy

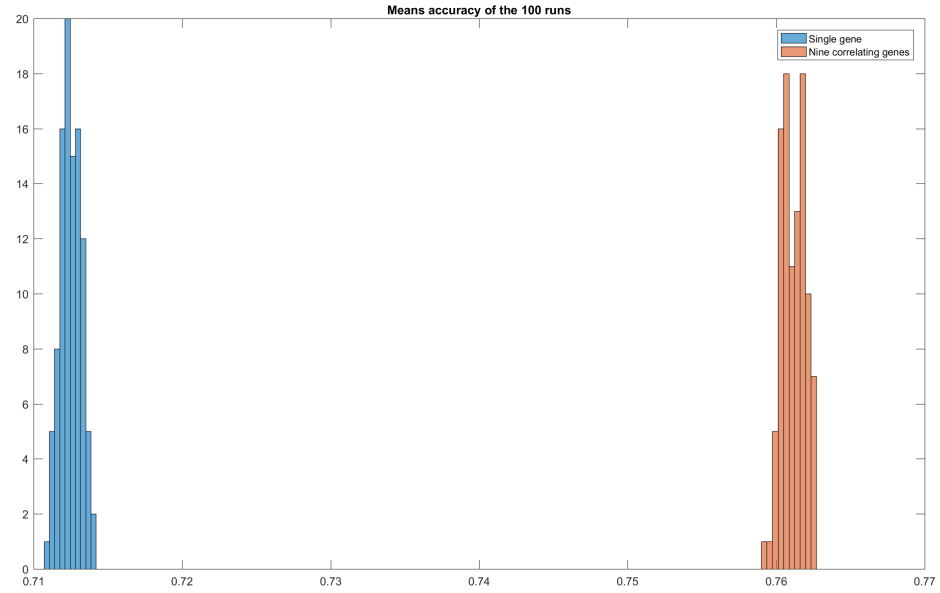


Figure 36: Two distributions representing the accuracies

B.2.1.2 Average of 16 correlated genes

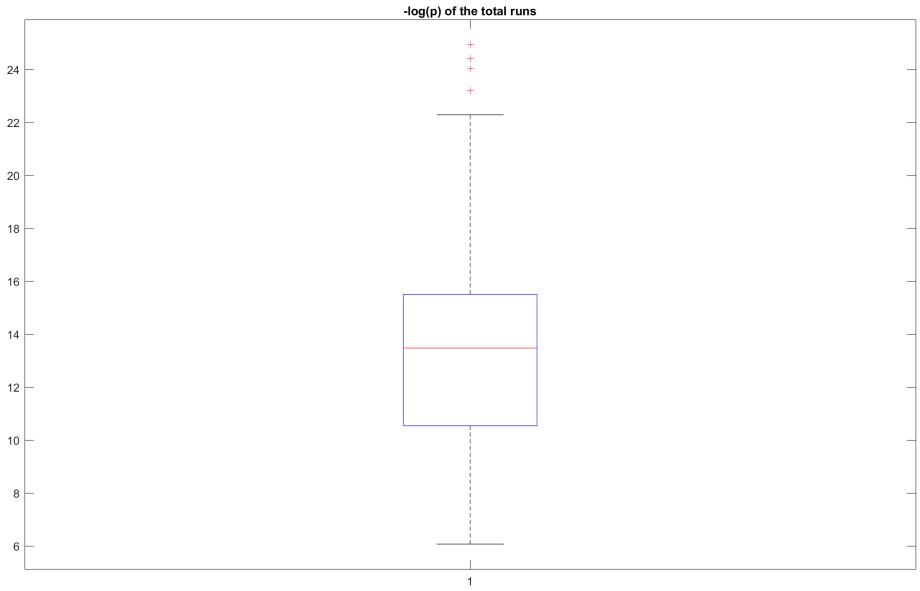


Figure 37: P-values of 100 right tail t-tests looking at the standard deviation

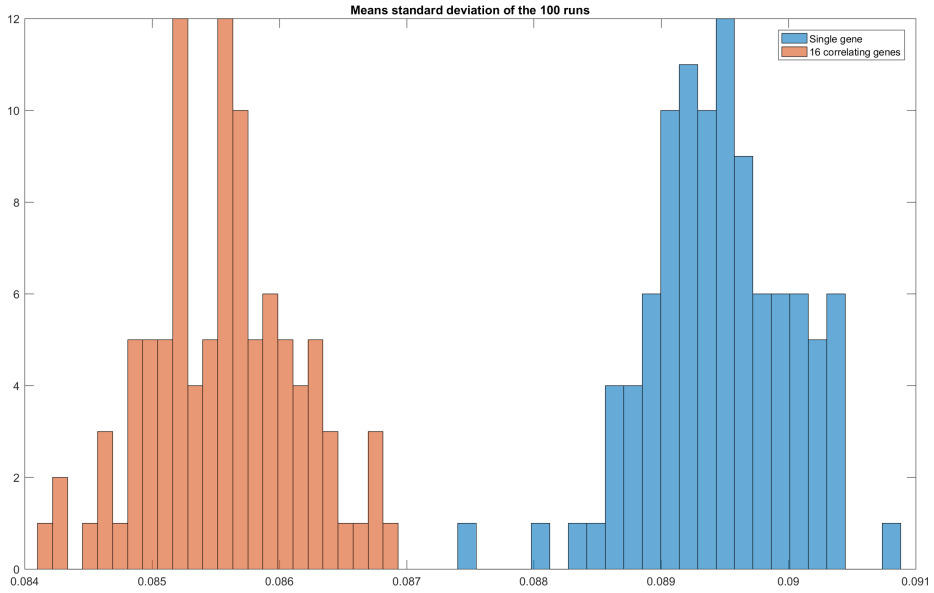


Figure 38: Two distributions of the average standard deviation

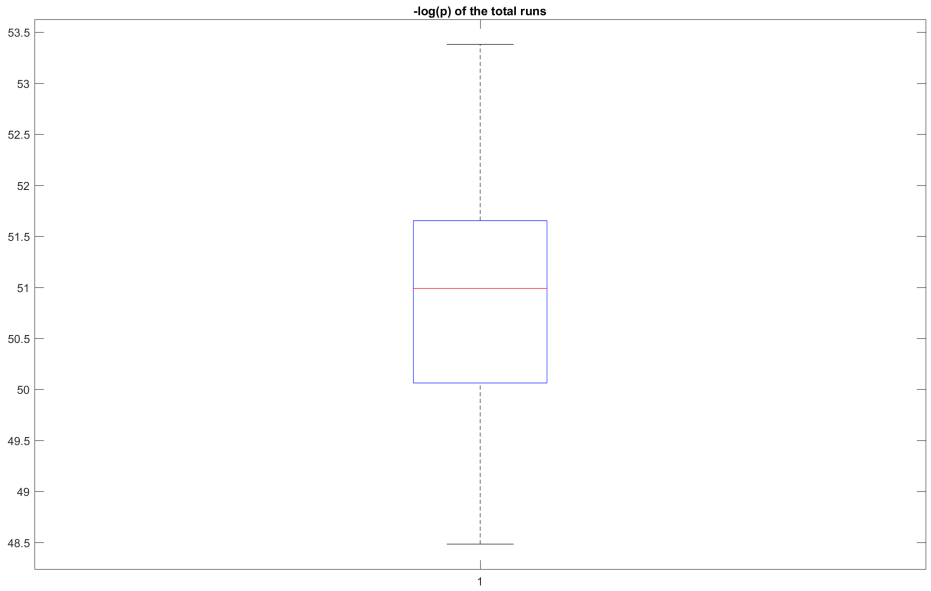


Figure 39: P-values of 100 left tail t-tests looking at the accuracy

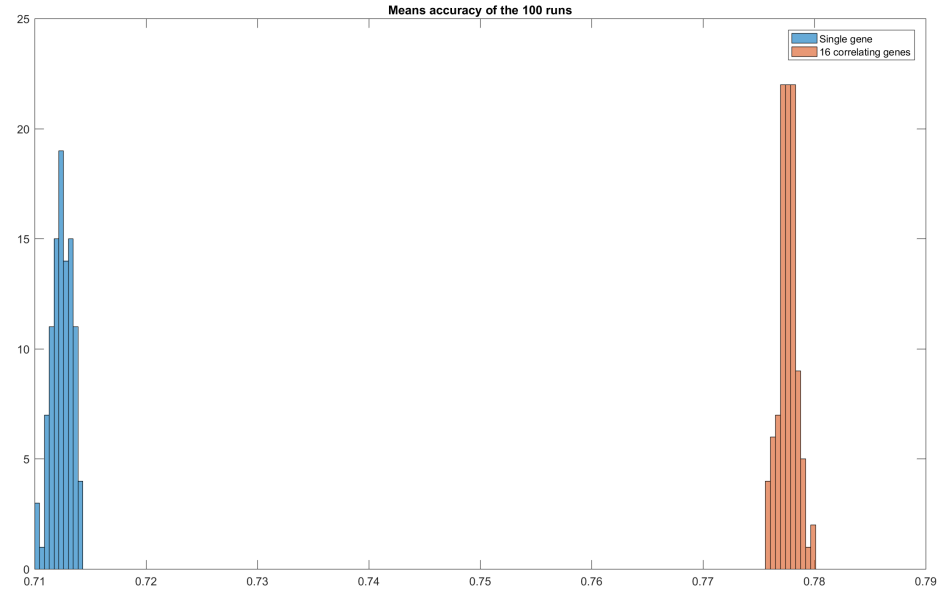


Figure 40: Two distributions representing the accuracies

B.2.1.3 Average of 25 correlated genes

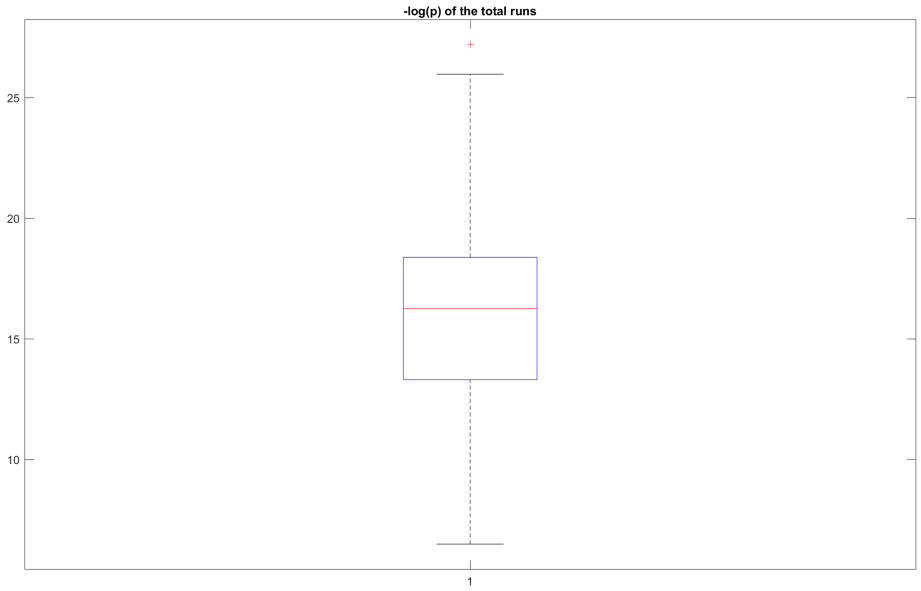


Figure 41: P-values of 100 right tail t-tests looking at the standard deviation

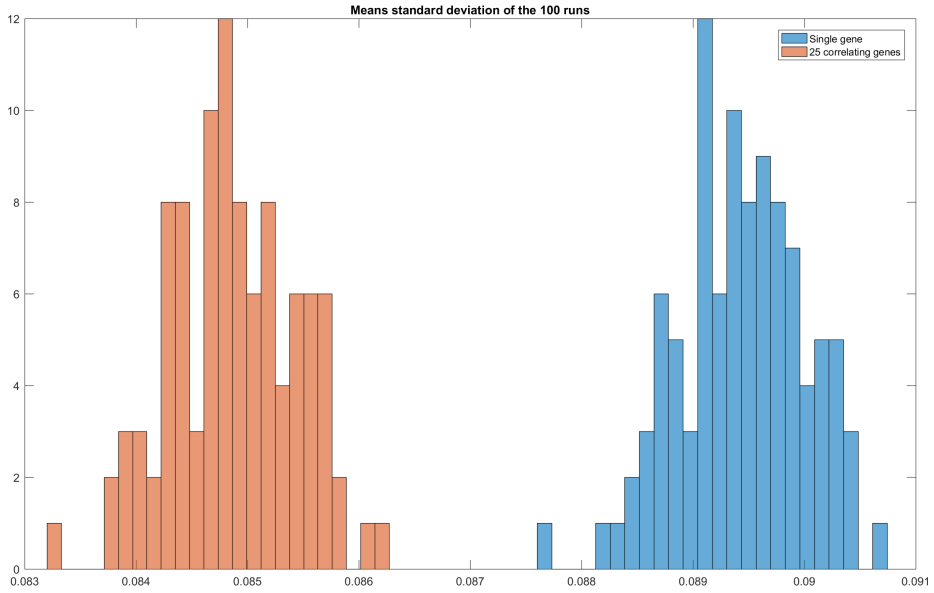


Figure 42: Two distributions of the average standard deviation

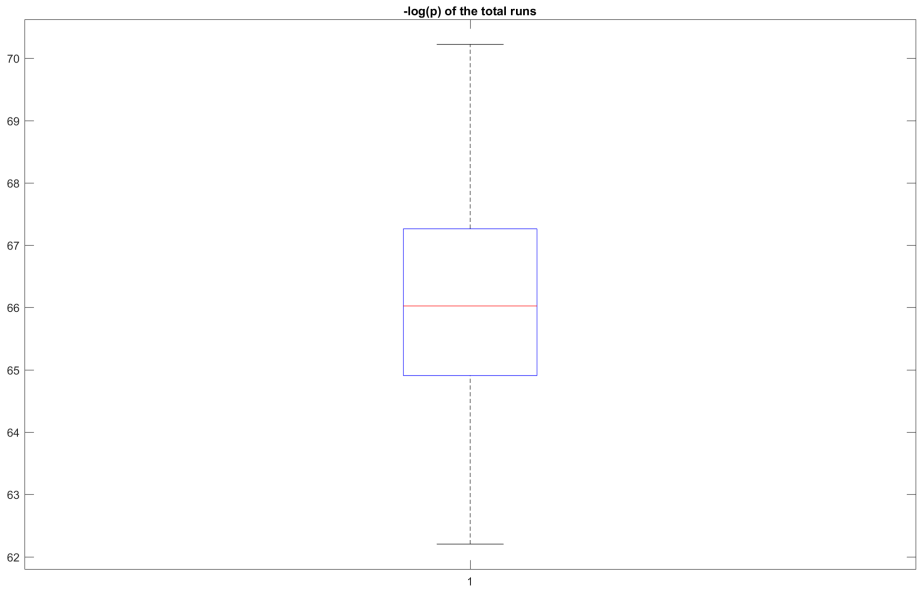


Figure 43: P-values of 100 left tail t-tests looking at the accuracy

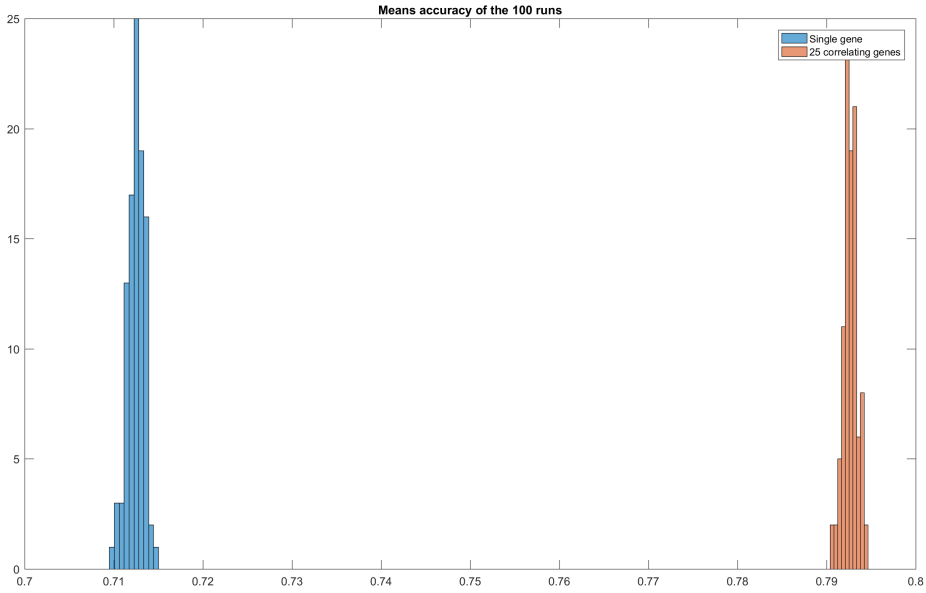


Figure 44: Two distributions representing the accuracies

B.2.1.4 Average of 36 correlated genes

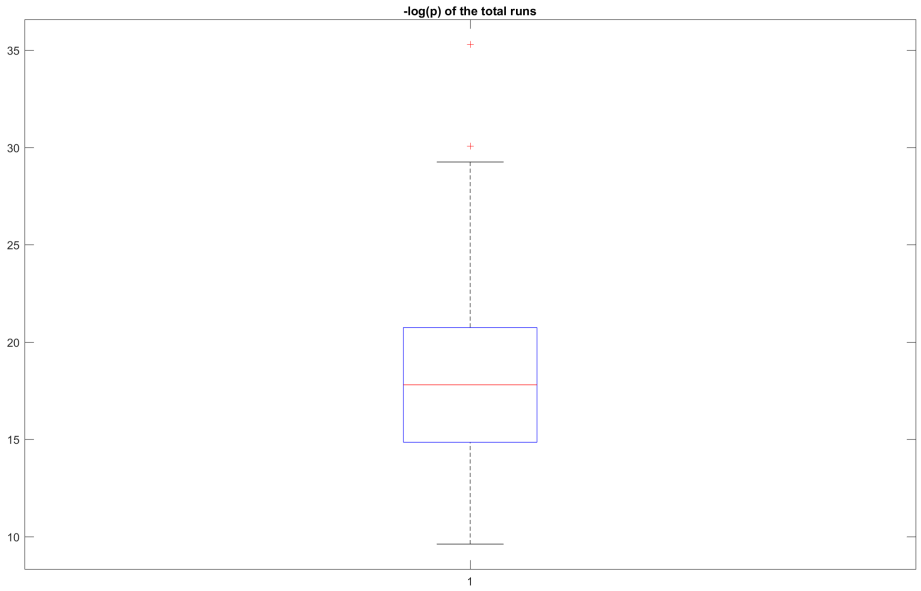


Figure 45: P-values of 100 right tail t-tests looking at the standard deviation

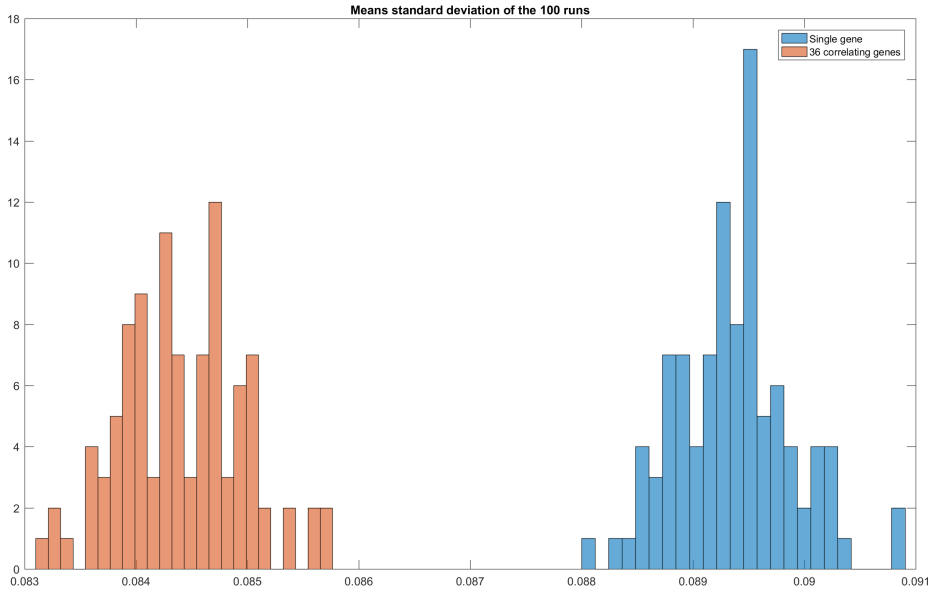


Figure 46: Two distributions of the average standard deviation

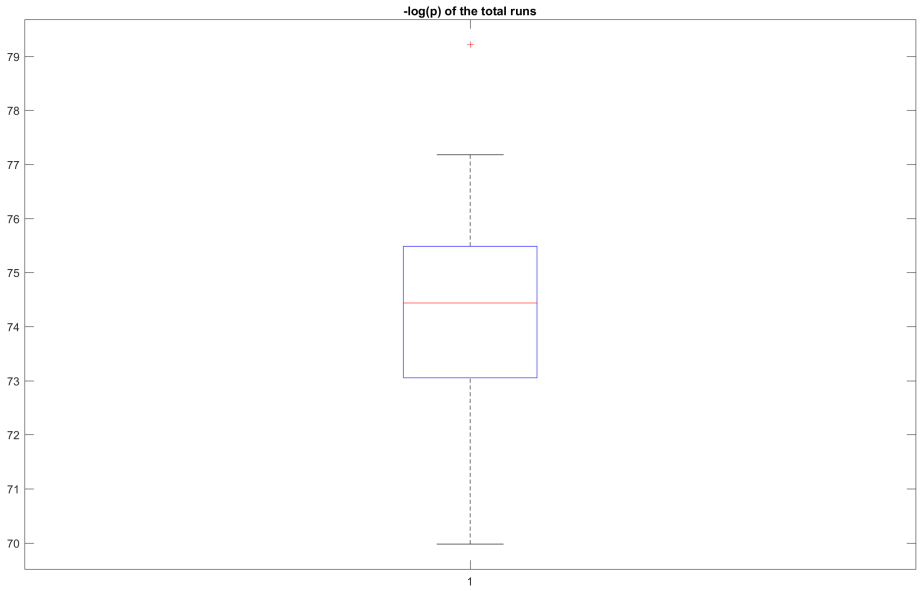


Figure 47: P-values of 100 left tail t-tests looking at the accuracy

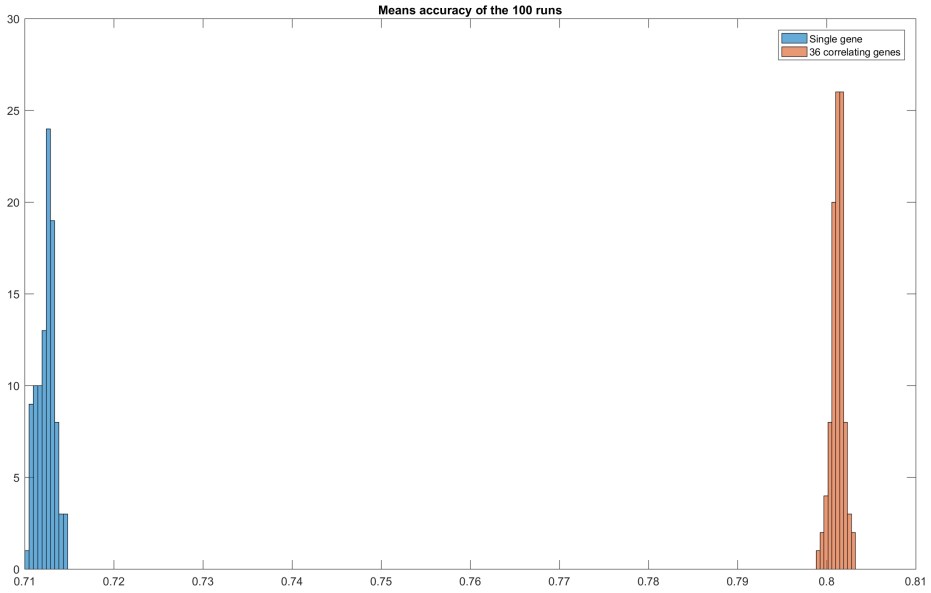


Figure 48: Two distributions representing the accuracies

B.2.1.5 Average of 49 correlated genes

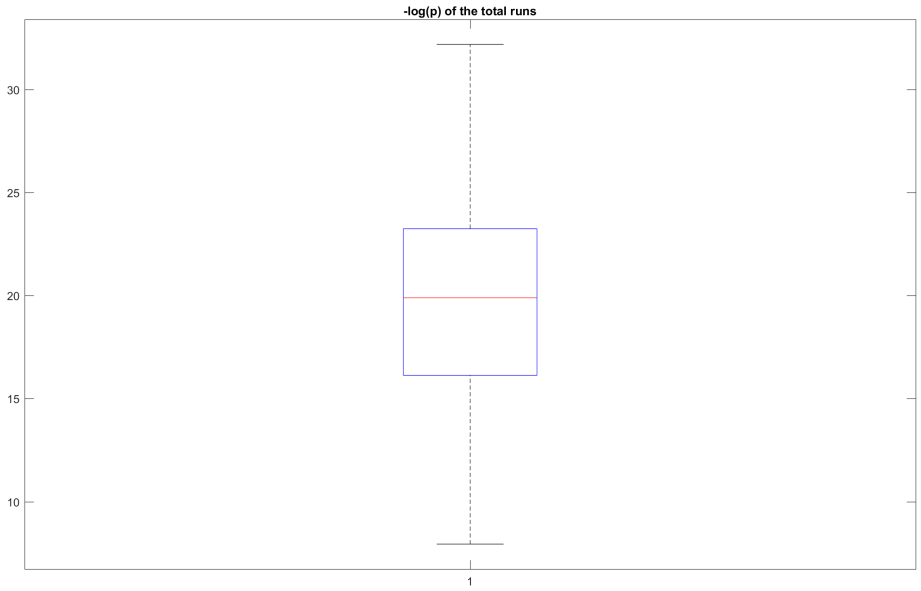


Figure 49: P-values of 100 right tail t-tests looking at the standard deviation

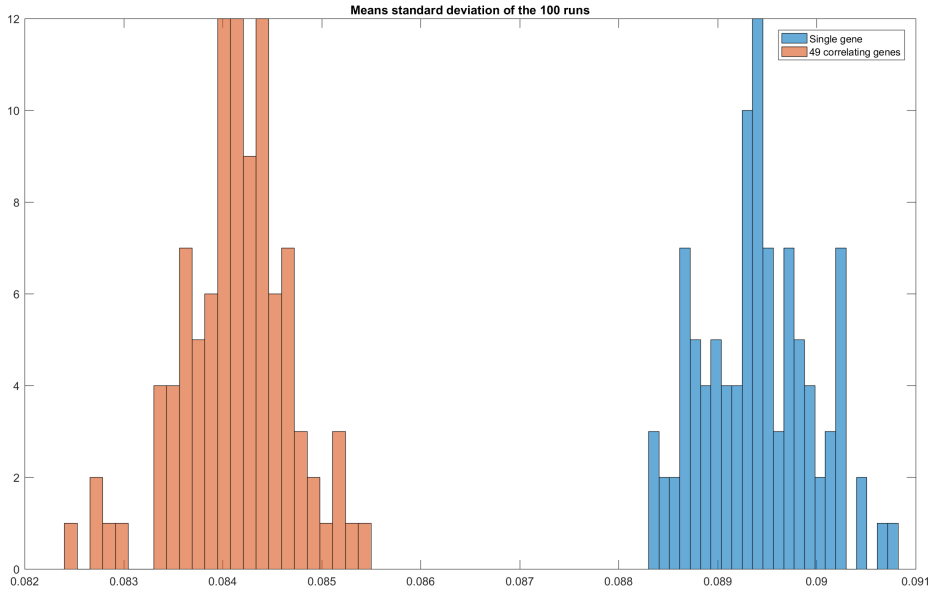


Figure 50: Two distributions of the average standard deviation

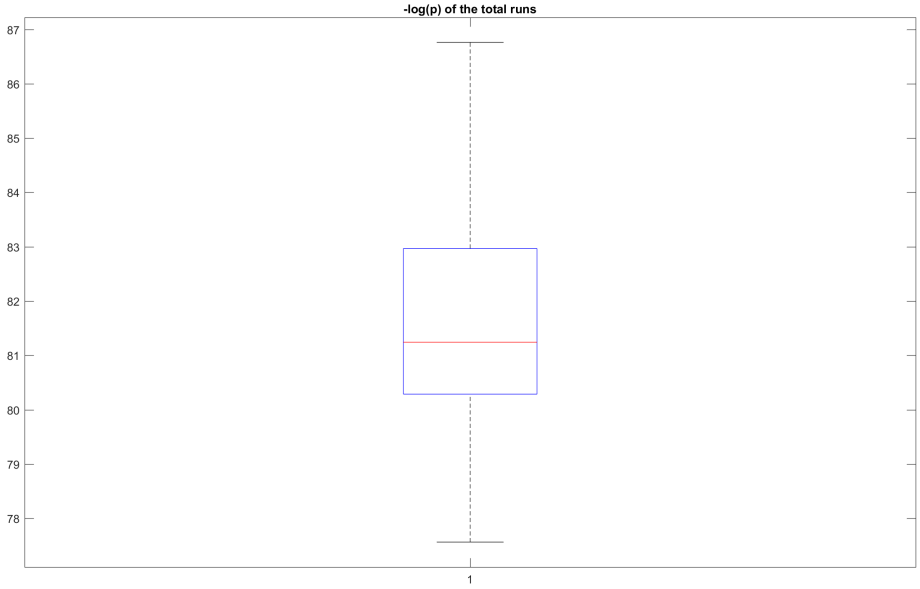


Figure 51: P-values of 100 left tail t-tests looking at the accuracy

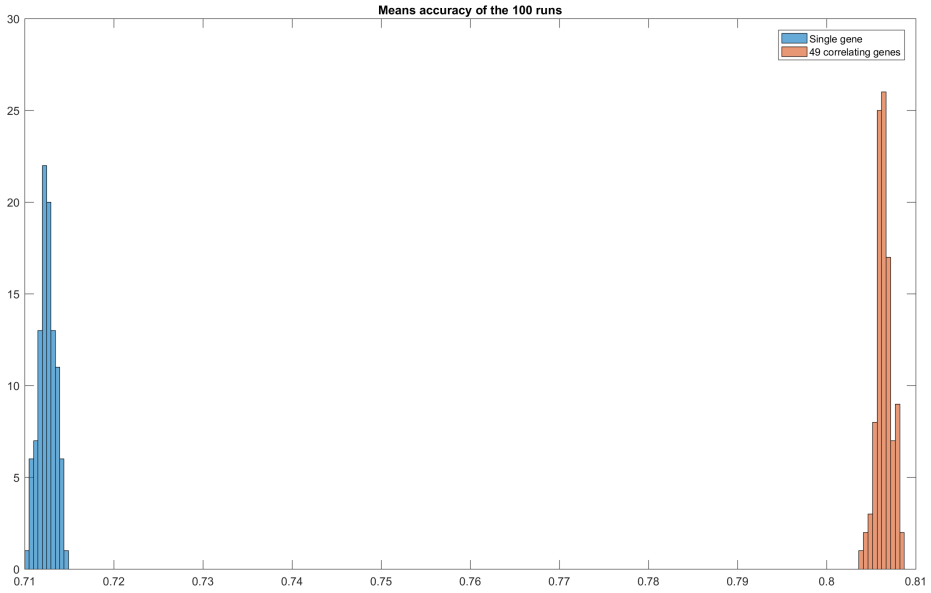


Figure 52: Two distributions representing the accuracies

B.2.2 Difference in accuracy by selecting more correlated genes to calculate the average activation with random label

B.2.2.1 Average of 4 correlated genes

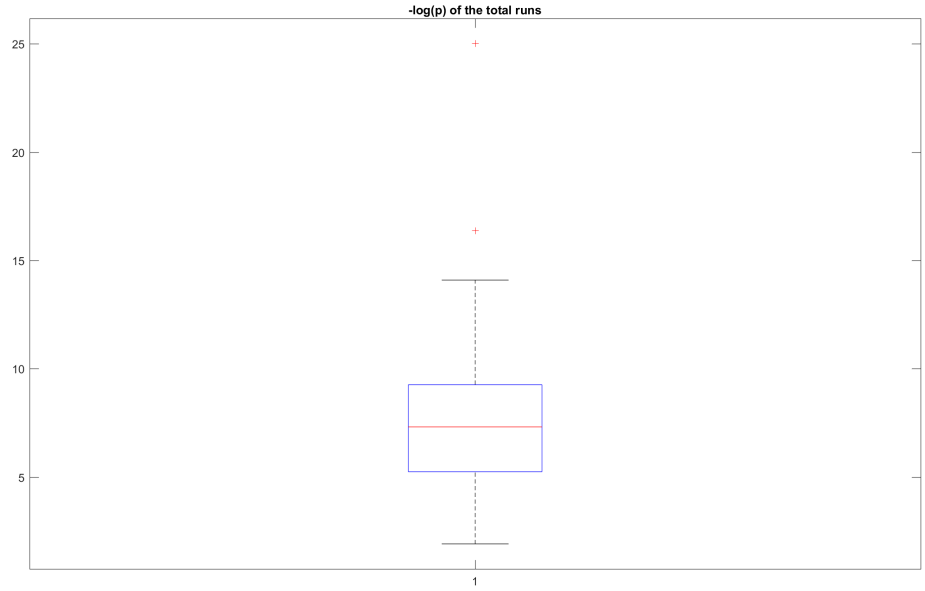


Figure 53: P-values of 100 right tail t-tests looking at the standard deviation

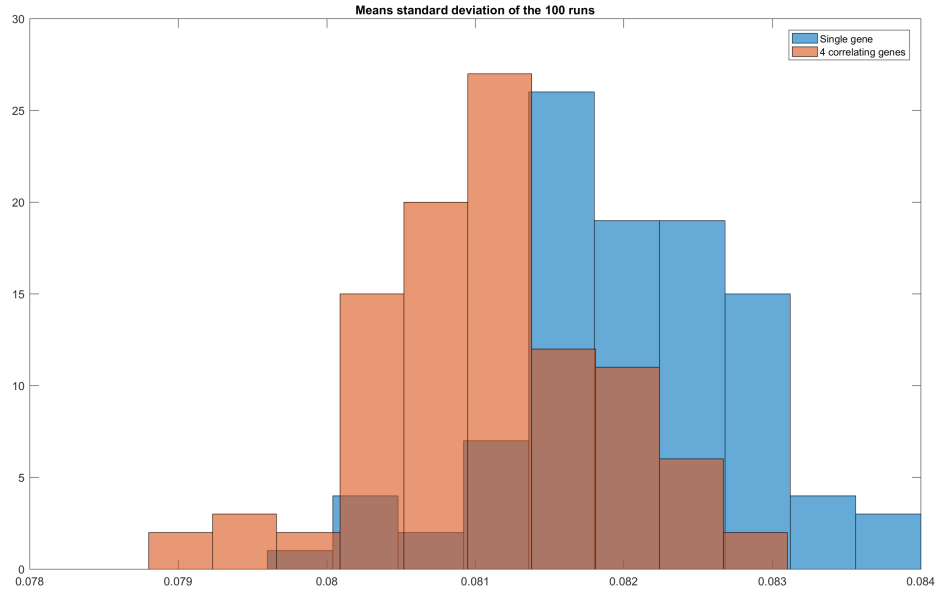


Figure 54: Two distributions of the average standard deviation

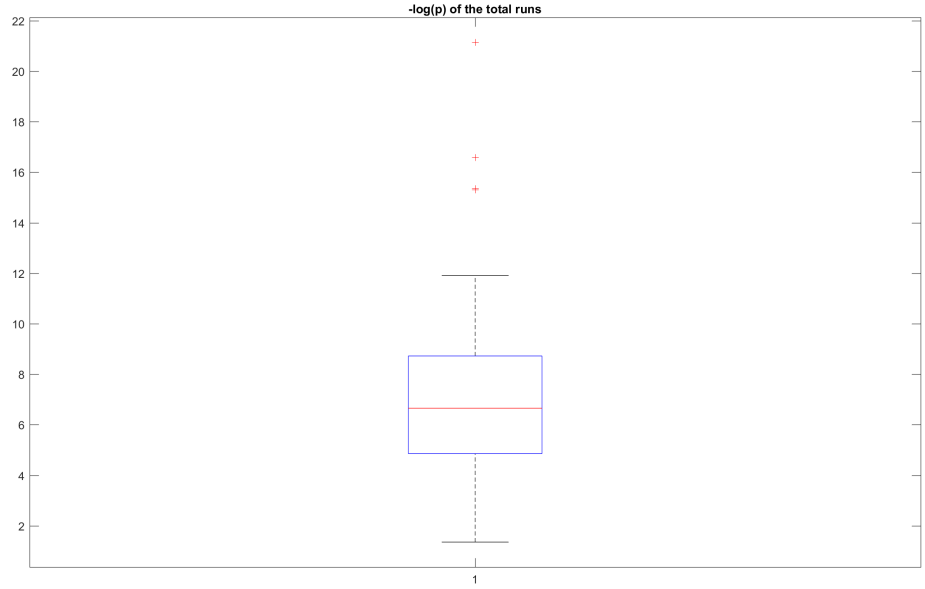


Figure 55: P-values of 100 left tail t-tests looking at the accuracy

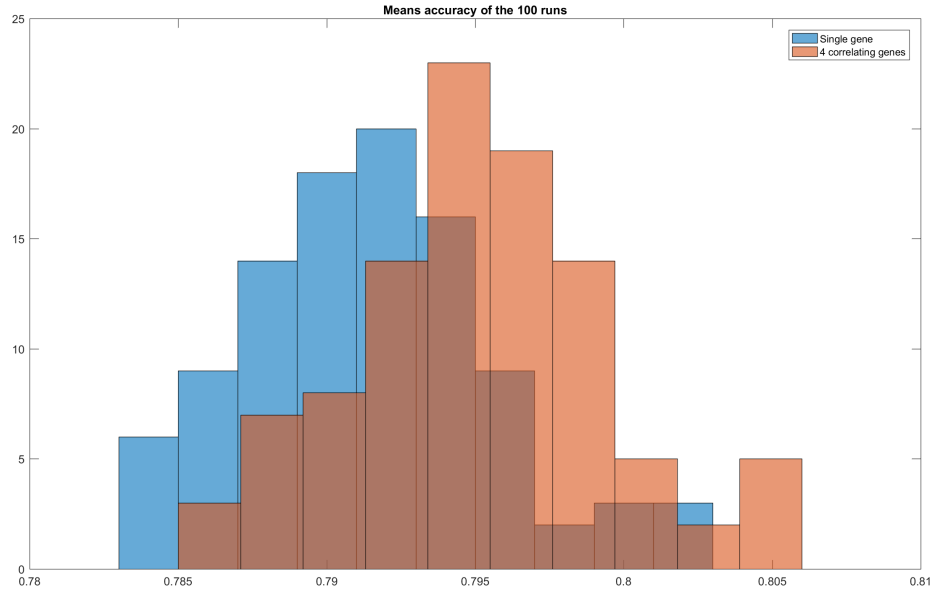


Figure 56: Two distributions representing the accuracies

B.2.2.2 Average of 9 correlated genes

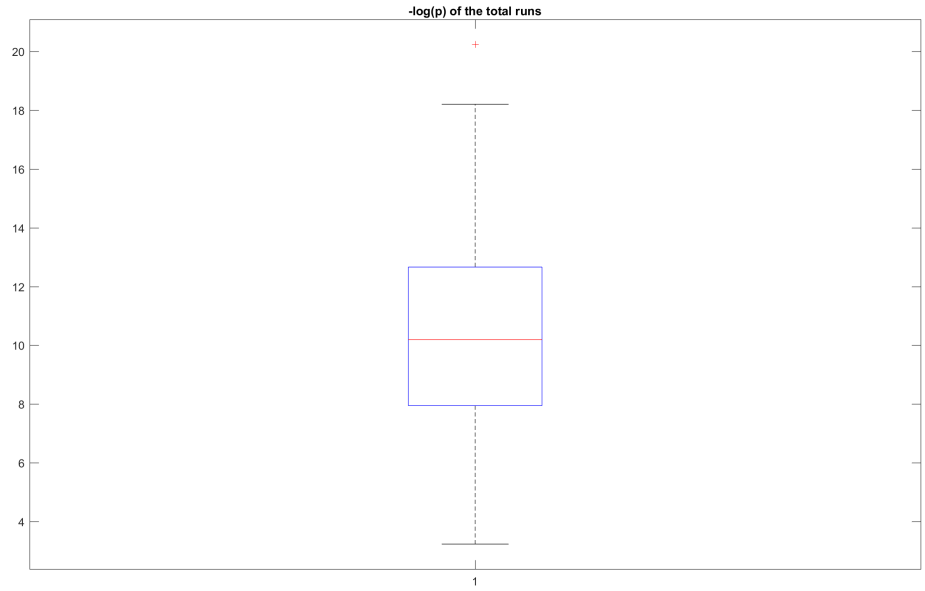


Figure 57: P-values of 100 right tail t-tests looking at the standard deviation

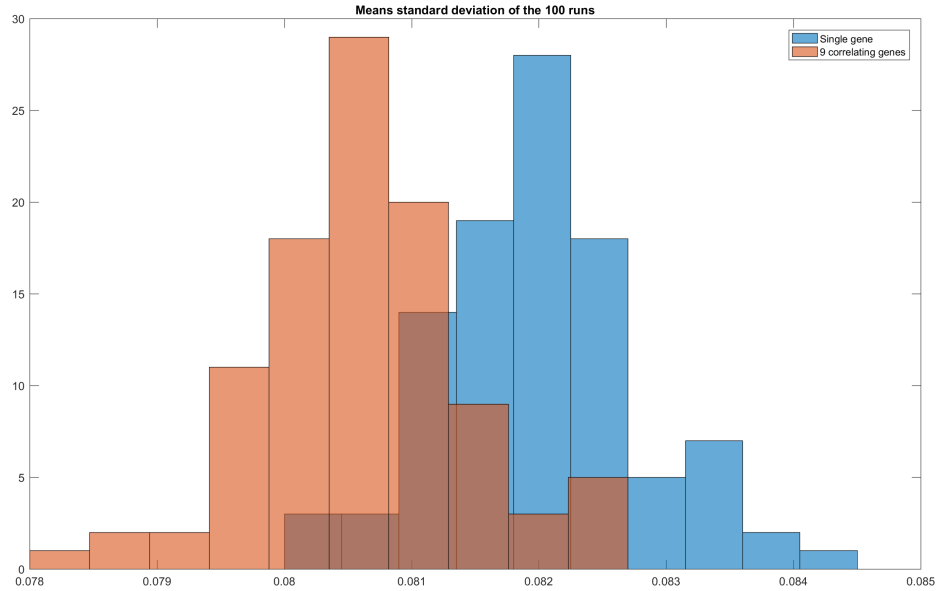


Figure 58: Two distributions of the average standard deviation

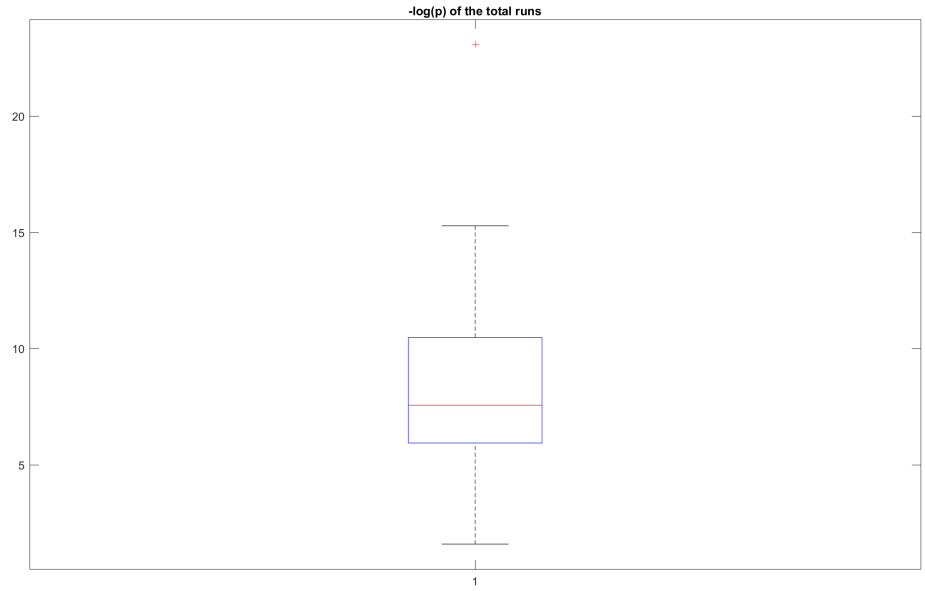


Figure 59: P-values of 100 left tail t-tests looking at the accuracy

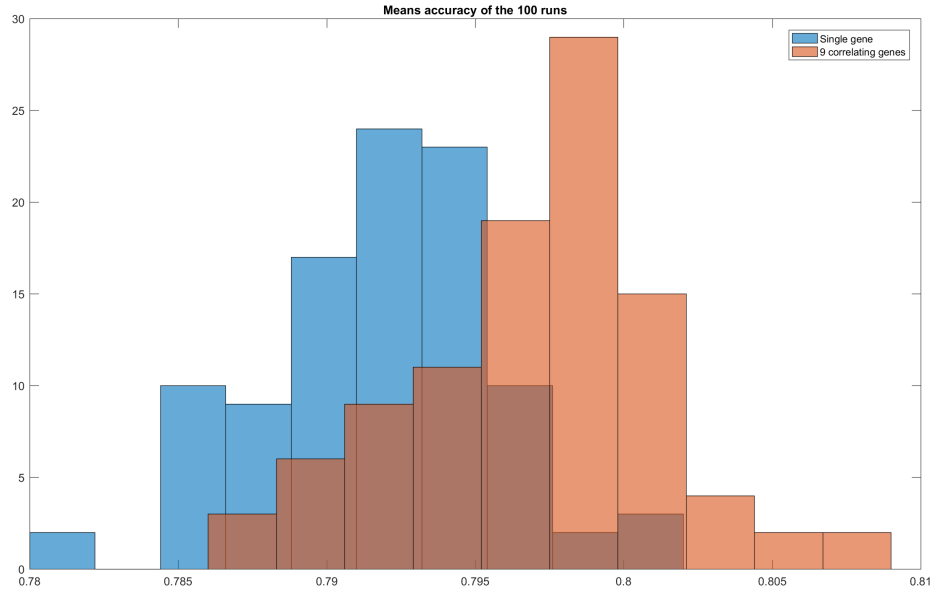


Figure 60: Two distributions representing the accuracies

B.2.2.3 Average of 16 correlated genes

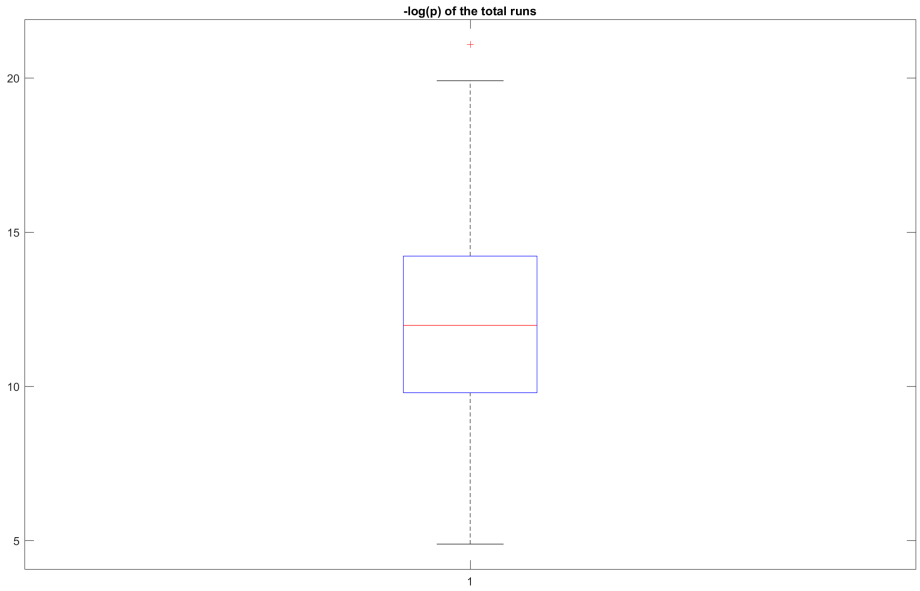


Figure 61: P-values of 100 right tail t-tests looking at the standard deviation

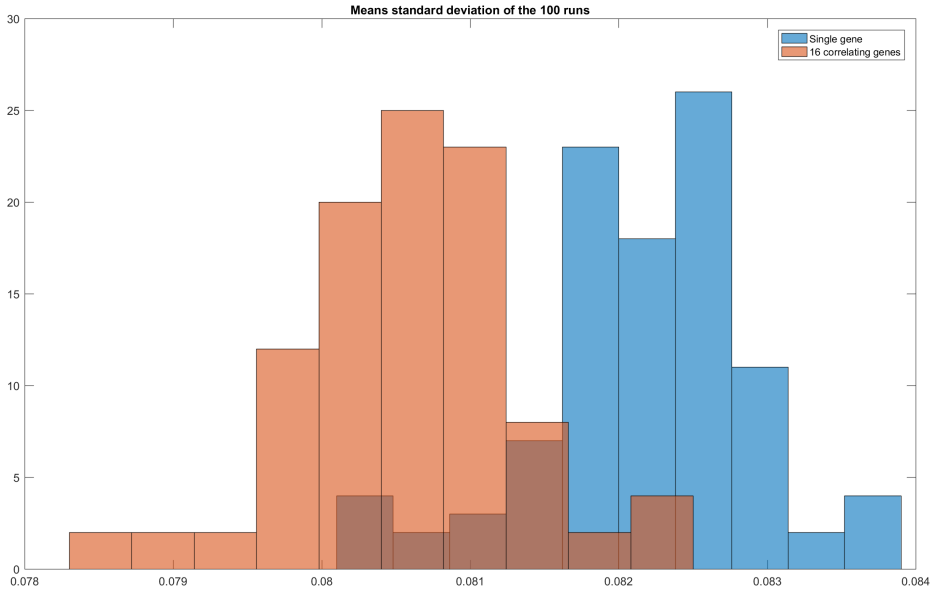


Figure 62: Two distributions of the average standard deviation

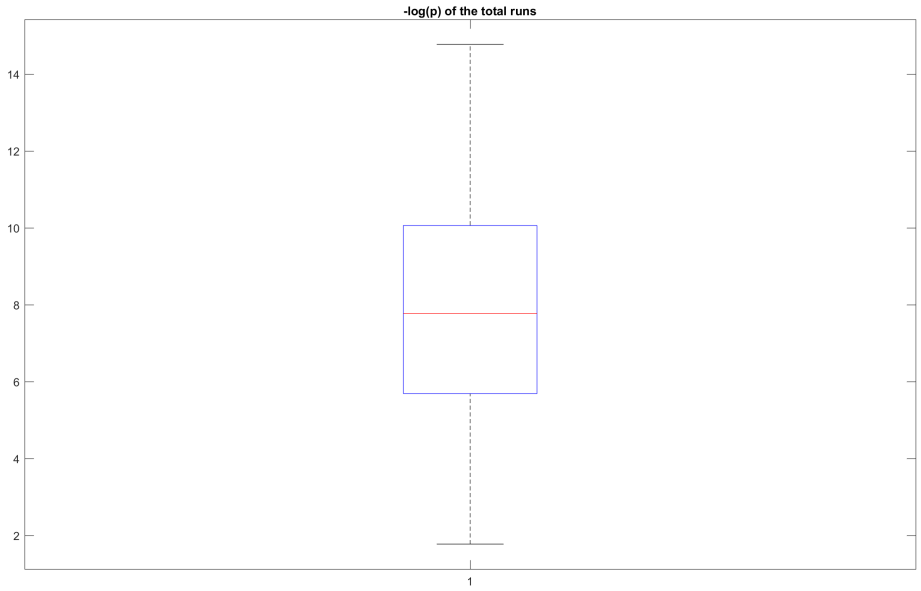


Figure 63: P-values of 100 left tail t-tests looking at the accuracy

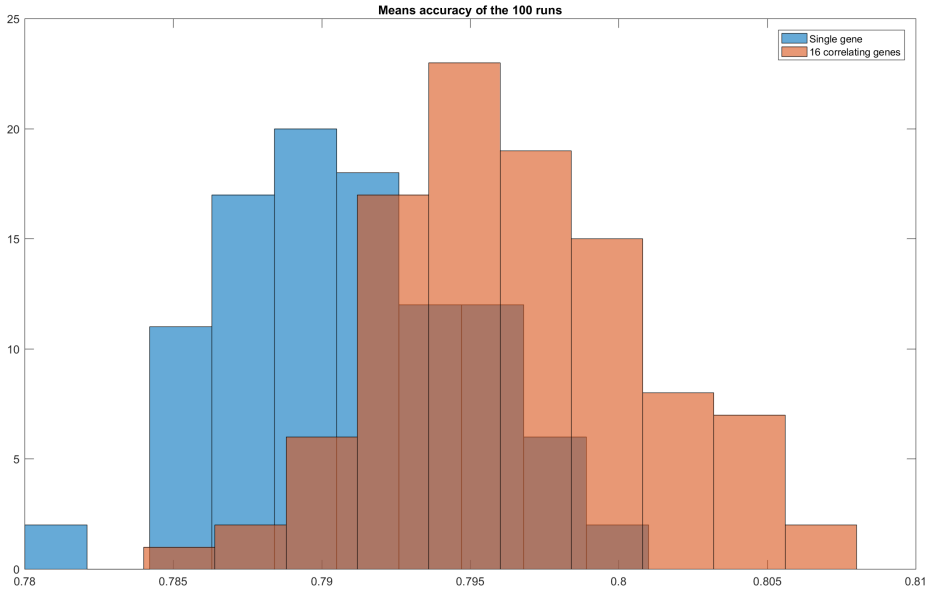


Figure 64: Two distributions representing the accuracies

B.2.2.4 Average of 25 correlated genes

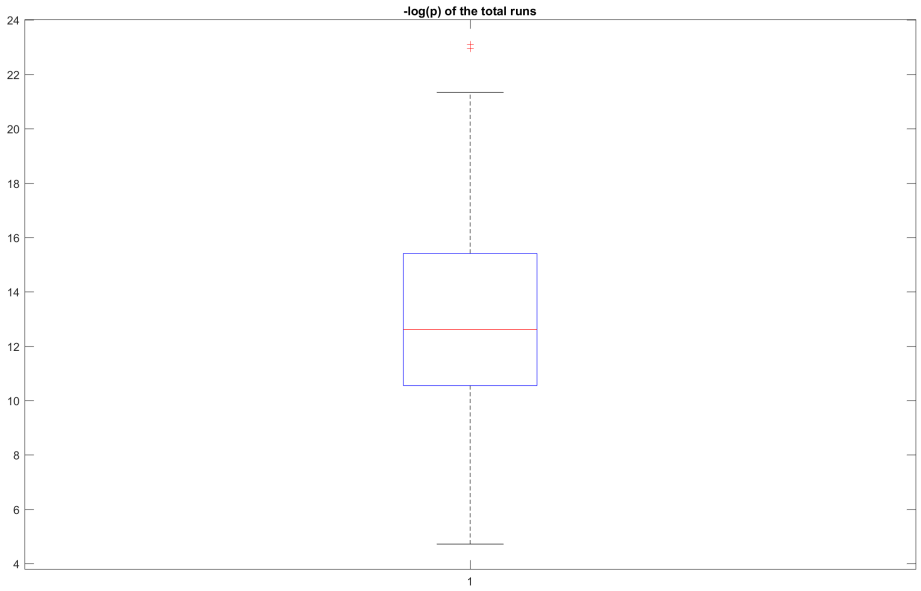


Figure 65: P-values of 100 right tail t-tests looking at the standard deviation

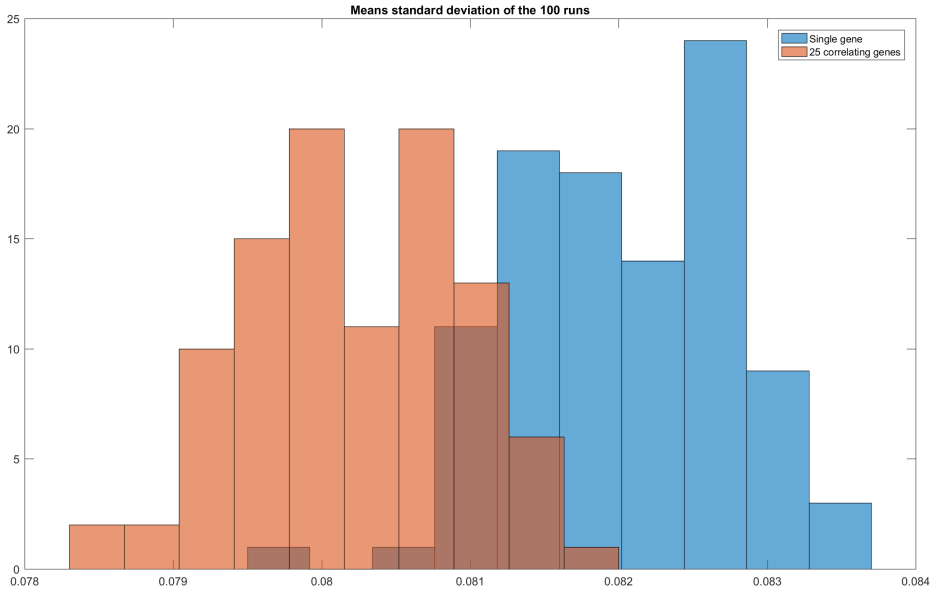


Figure 66: Two distributions of the average standard deviation

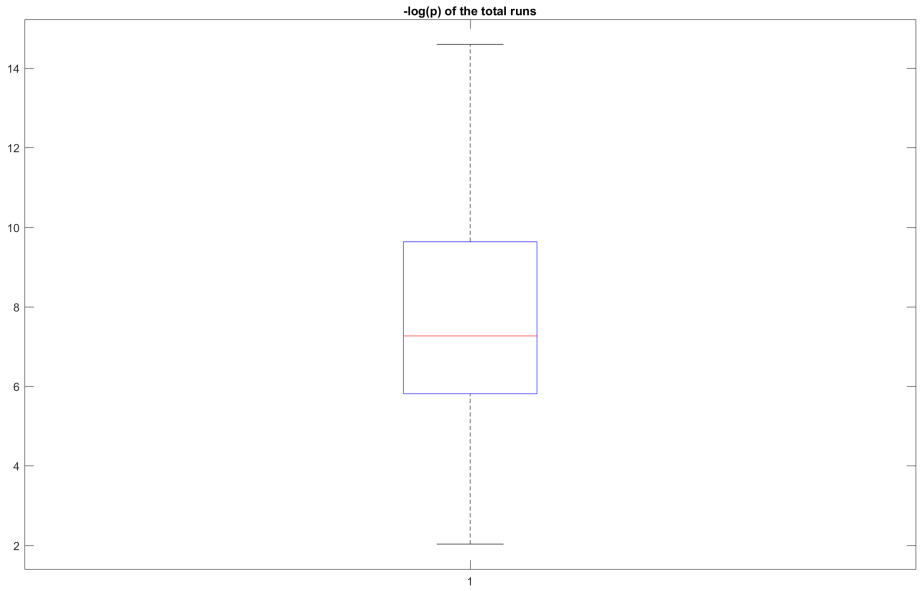


Figure 67: P-values of 100 left tail t-tests looking at the accuracy

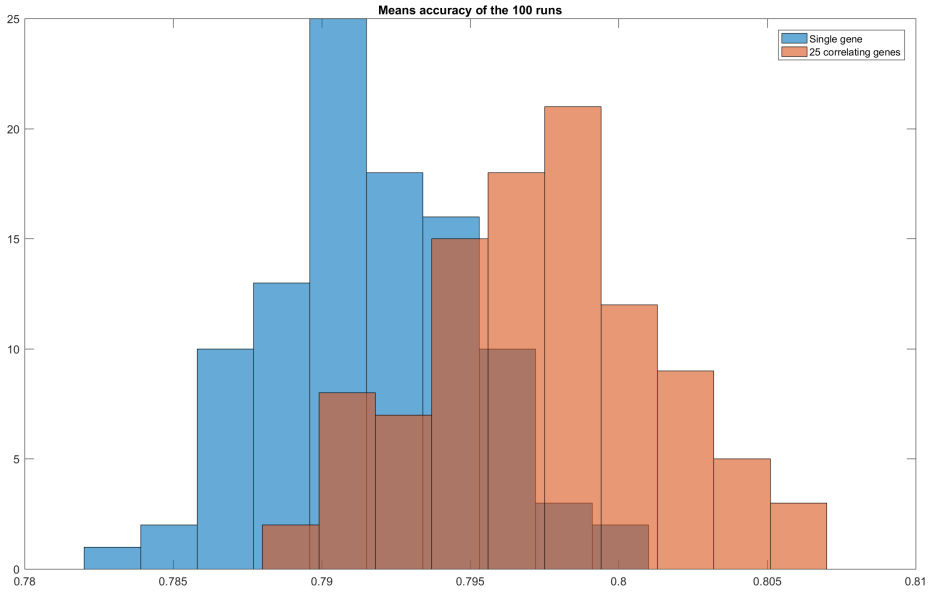


Figure 68: Two distributions representing the accuracies

B.2.2.5 Average of 36 correlated genes

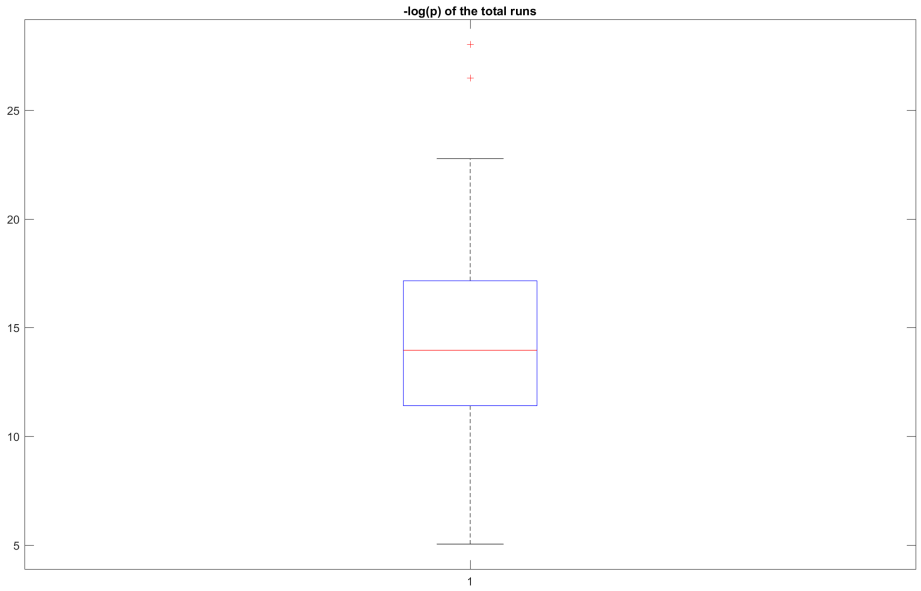


Figure 69: P-values of 100 right tail t-tests looking at the standard deviation

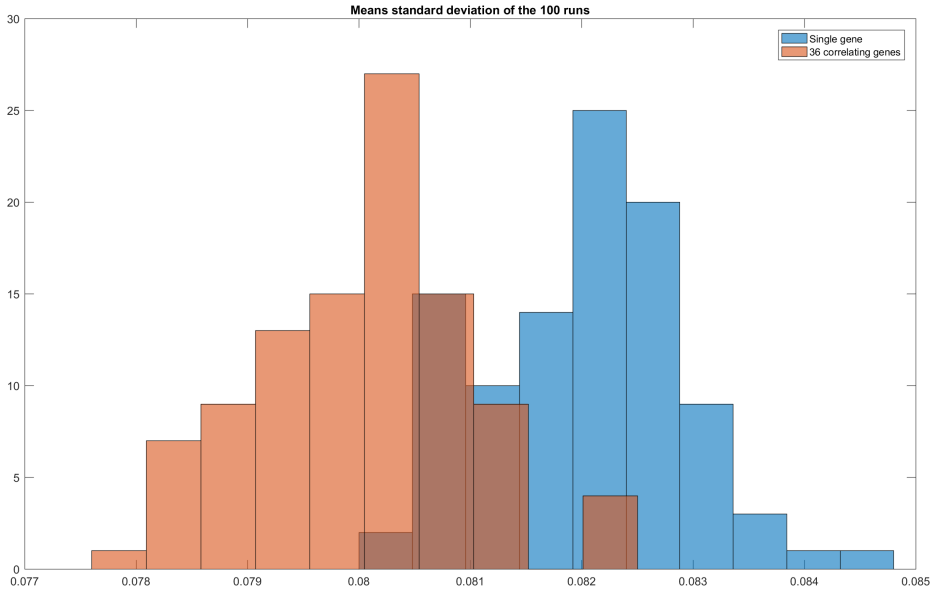


Figure 70: Two distributions of the average standard deviation

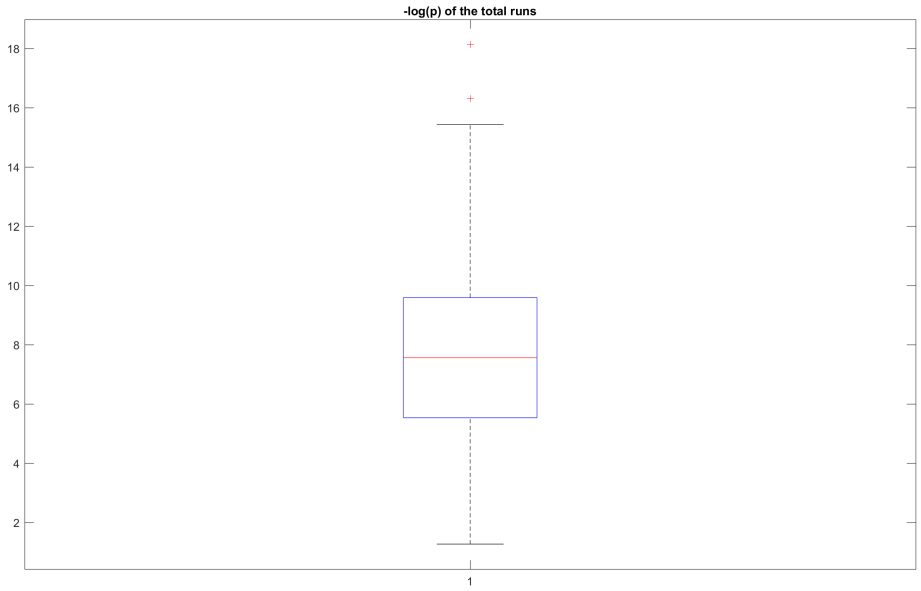


Figure 71: P-values of 100 left tail t-tests looking at the accuracy

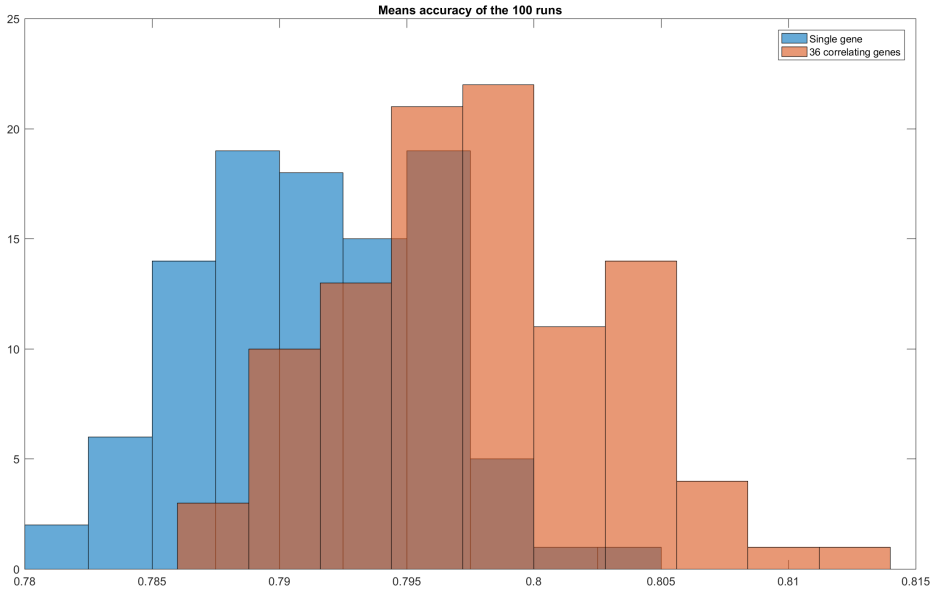


Figure 72: Two distributions representing the accuracies

B.2.2.6 Average of 49 correlated genes

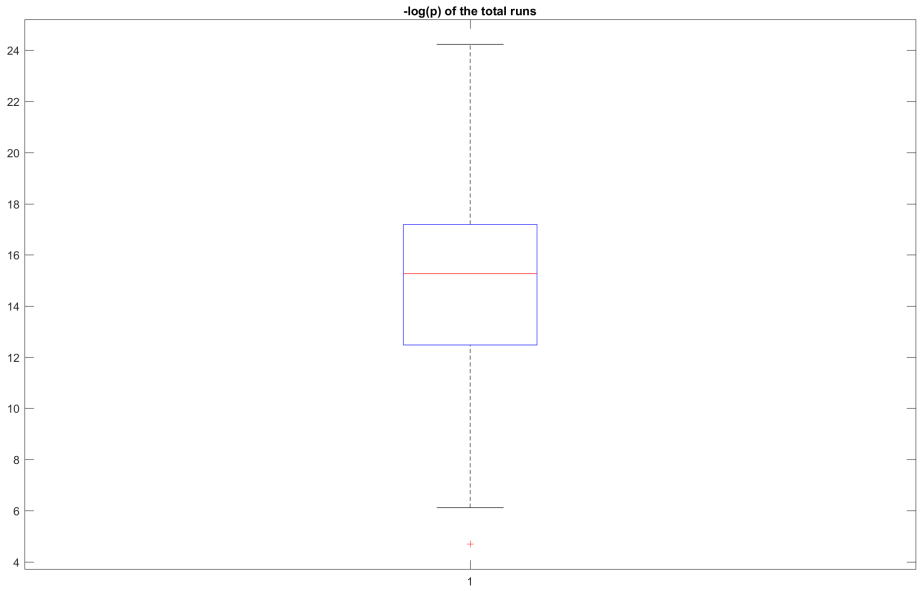


Figure 73: P-values of 100 right tail t-tests looking at the standard deviation

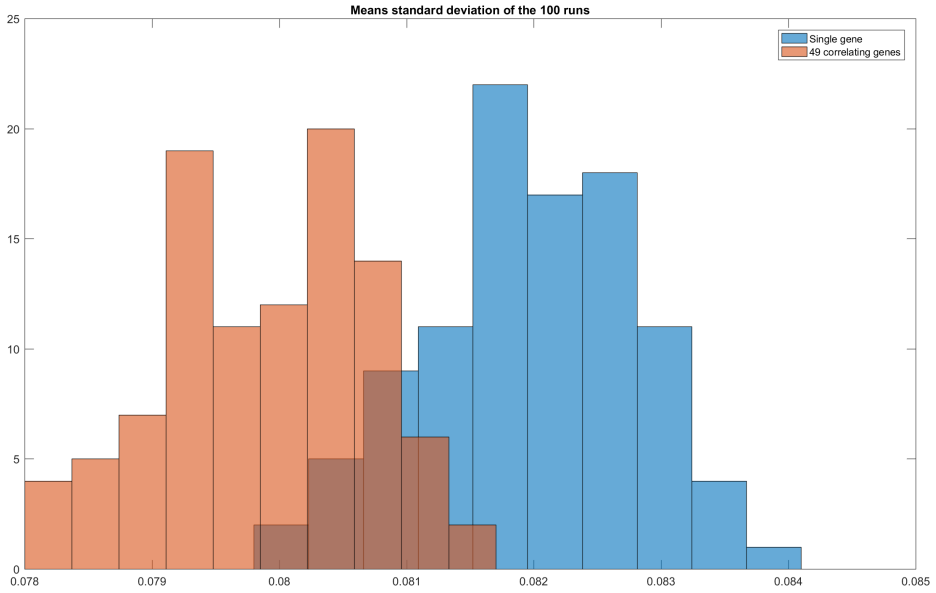


Figure 74: Two distributions of the average standard deviation

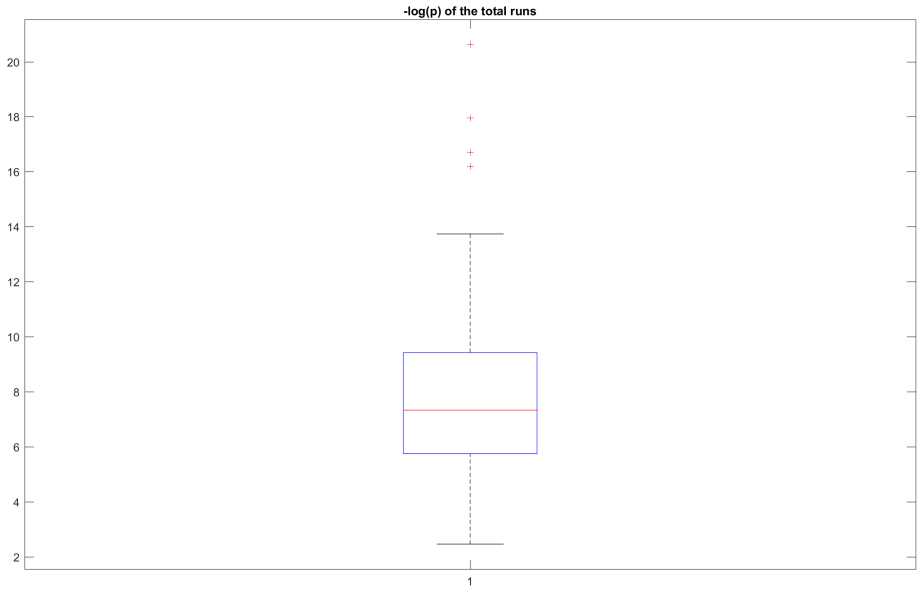


Figure 75: P-values of 100 left tail t-tests looking at the accuracy

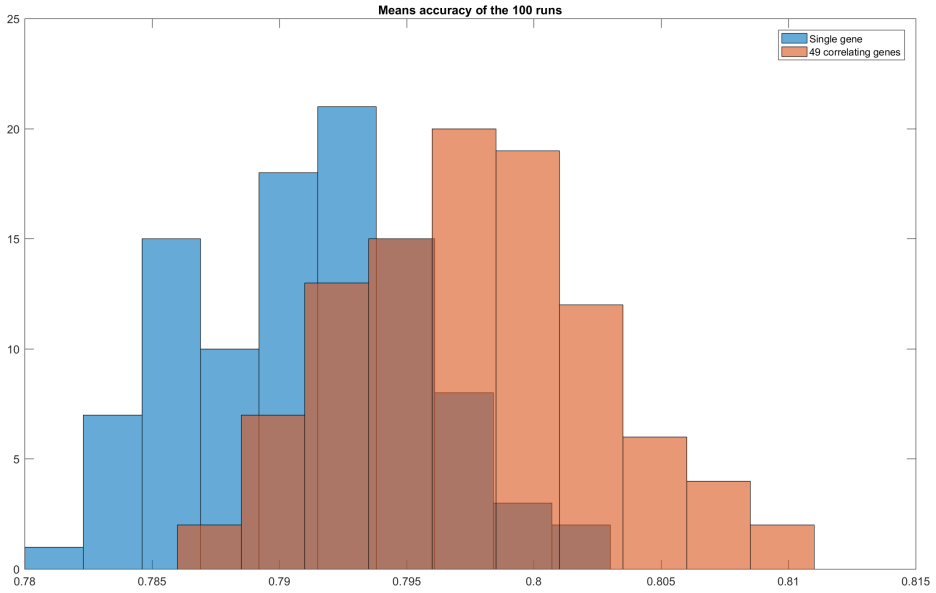


Figure 76: Two distributions representing the accuracies

B.2.3 Increasing the group-size

B.2.3.1 Average of 4 correlated genes

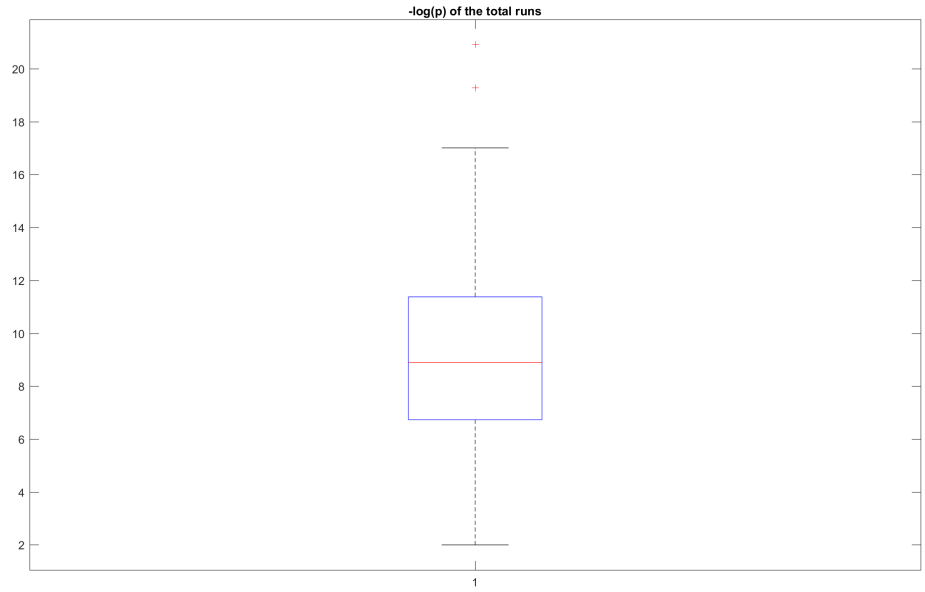


Figure 77: P-values of 100 right tail t-tests looking at the standard deviation

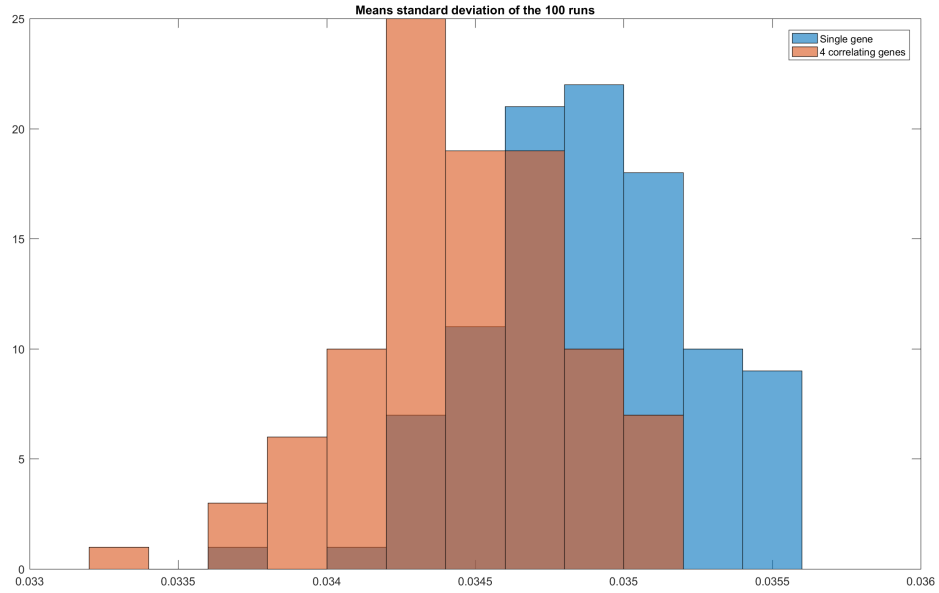


Figure 78: Two distributions of the average standard deviation

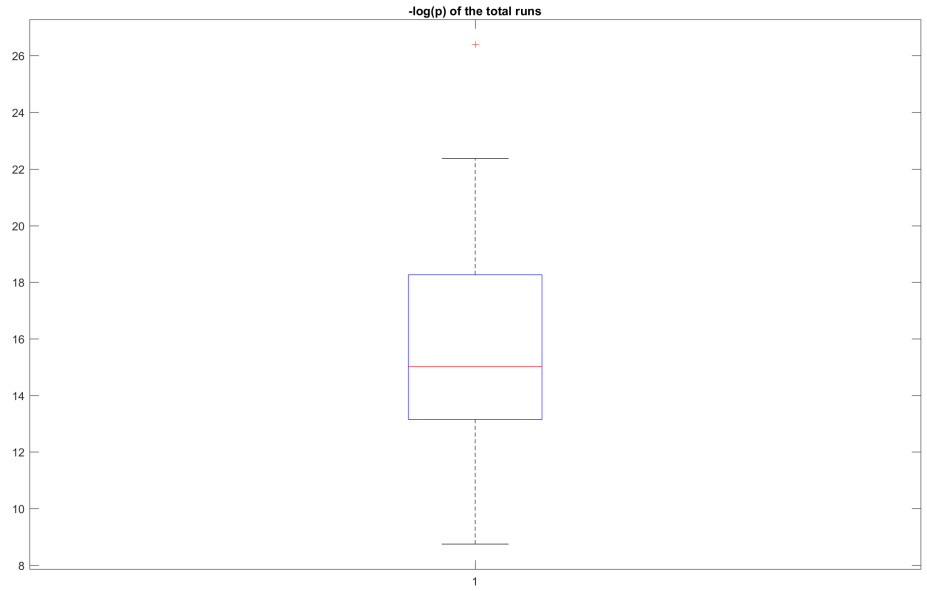


Figure 79: P-values of 100 left tail t-tests looking at the accuracy

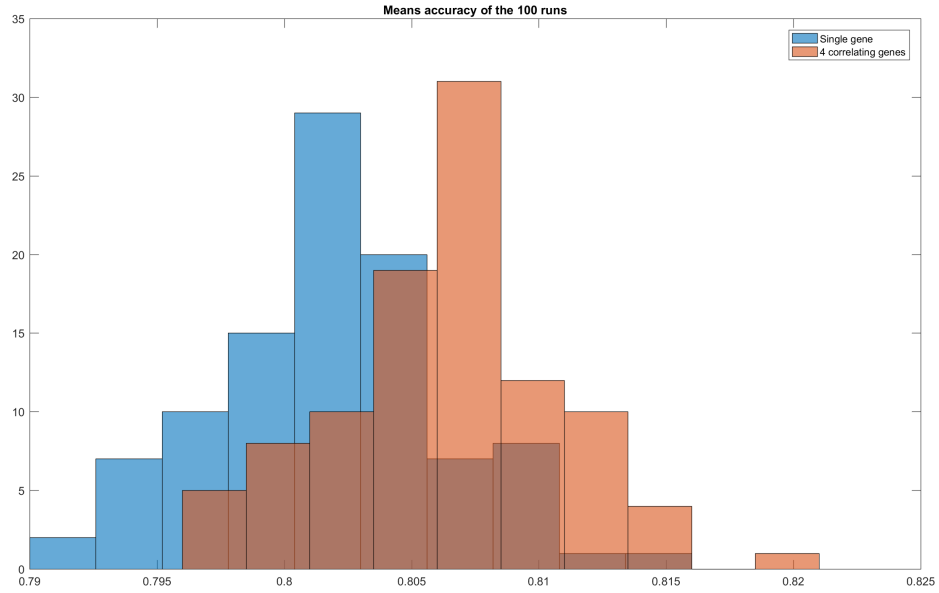


Figure 80: Two distributions representing the accuracies

B.2.3.2 Average of 9 correlated genes

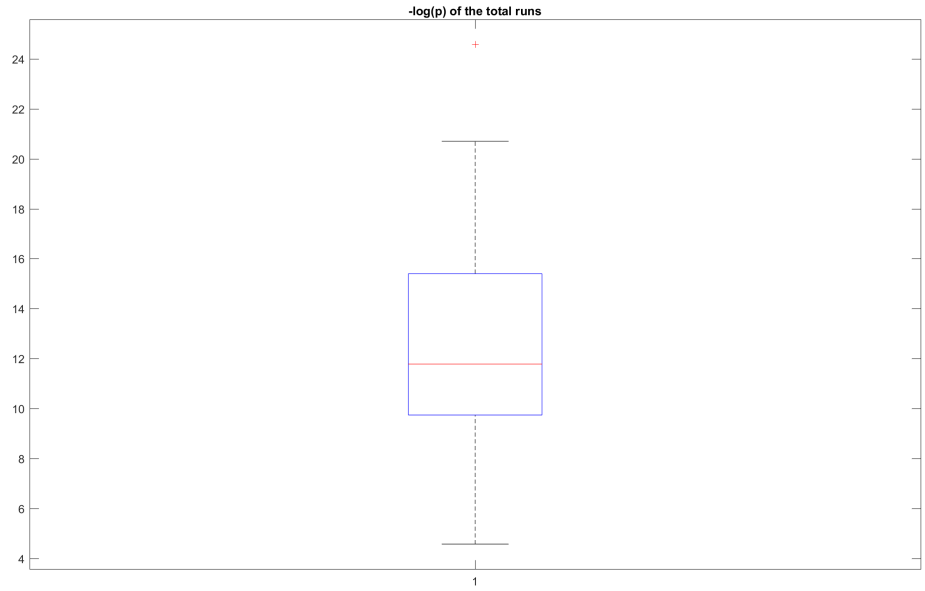


Figure 81: P-values of 100 right tail t-tests looking at the standard deviation

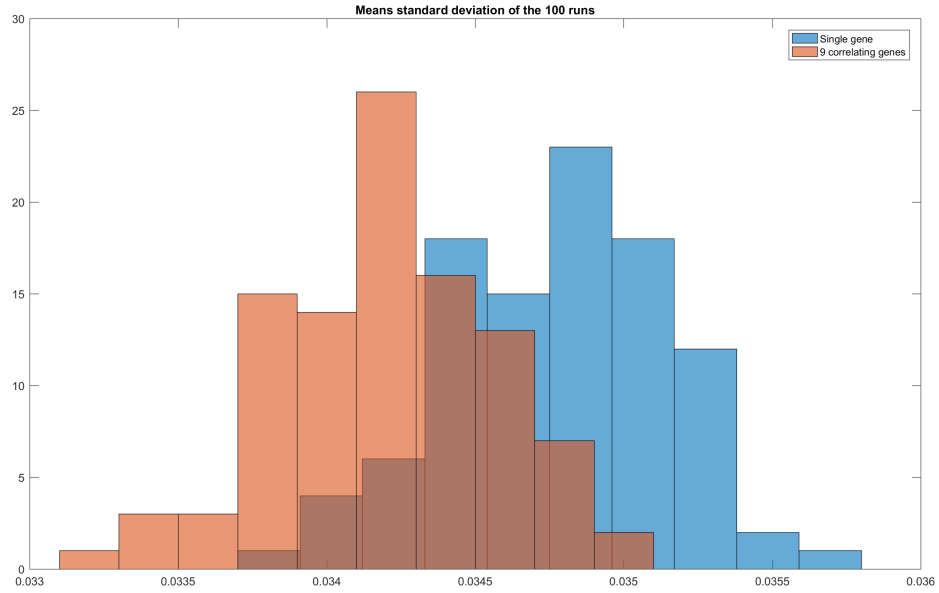


Figure 82: Two distributions of the average standard deviation

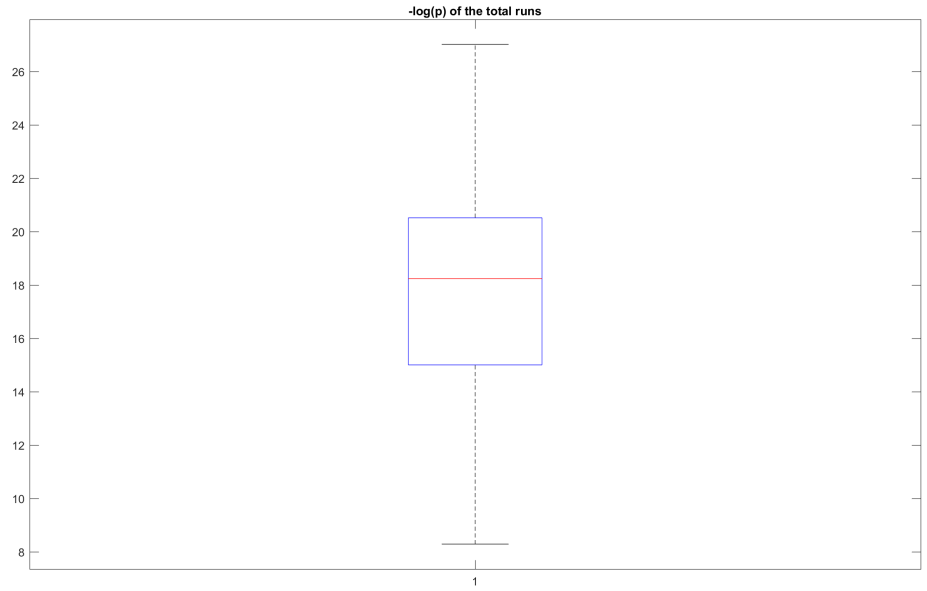


Figure 83: P-values of 100 left tail t-tests looking at the accuracy

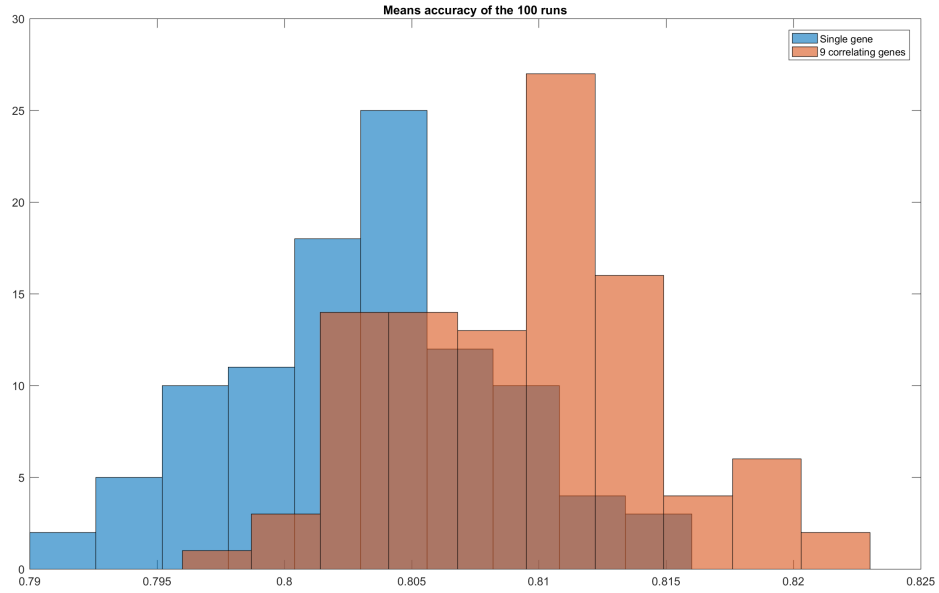


Figure 84: Two distributions representing the accuracies

B.2.3.3 Average of 16 correlated genes

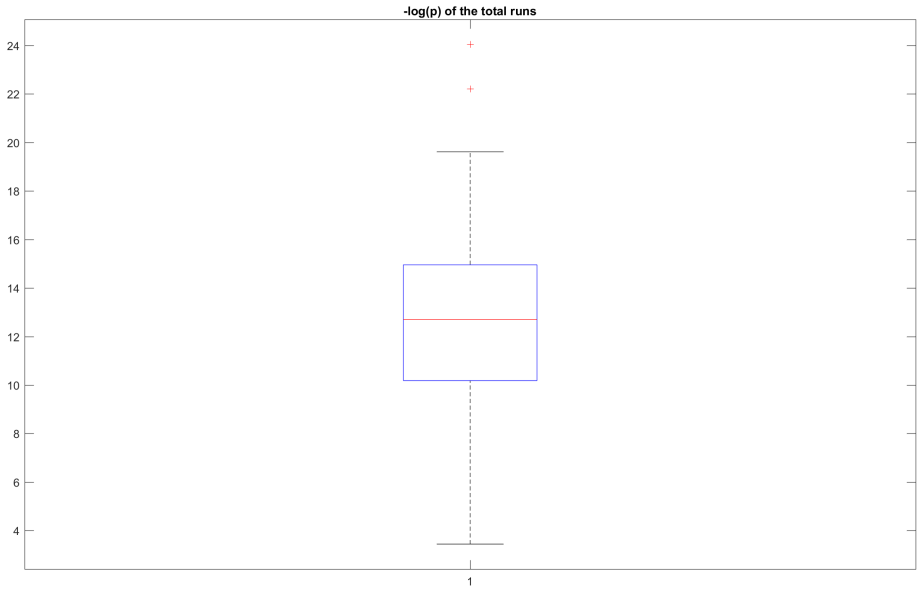


Figure 85: P-values of 100 right tail t-tests looking at the standard deviation

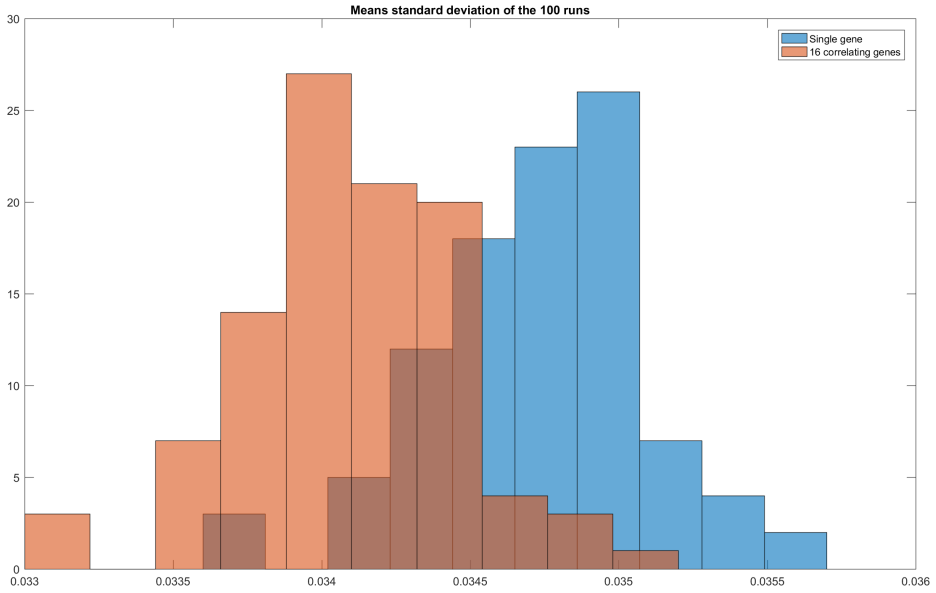


Figure 86: Two distributions of the average standard deviation

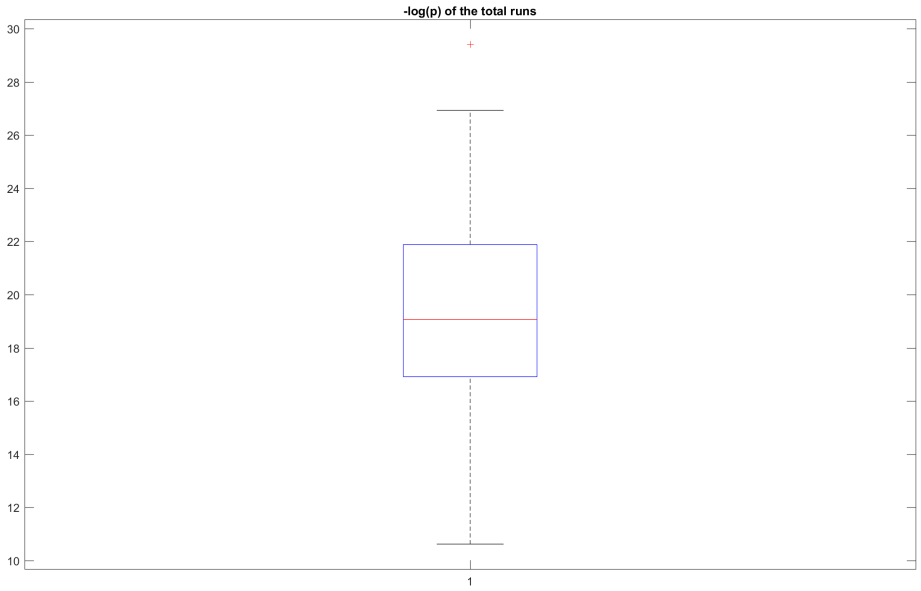


Figure 87: P-values of 100 left tail t-tests looking at the accuracy

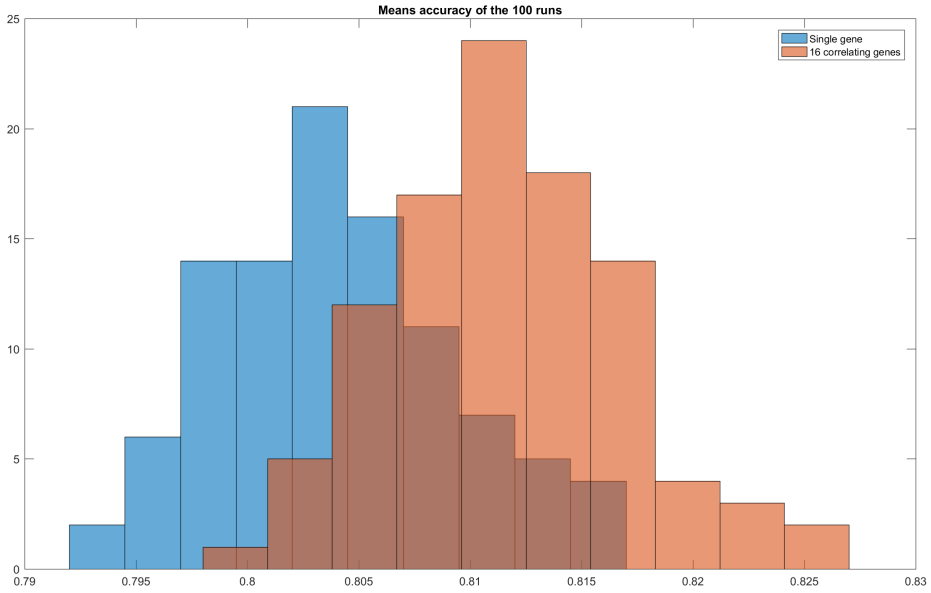


Figure 88: Two distributions representing the accuracies

B.2.3.4 Average of 25 correlated genes

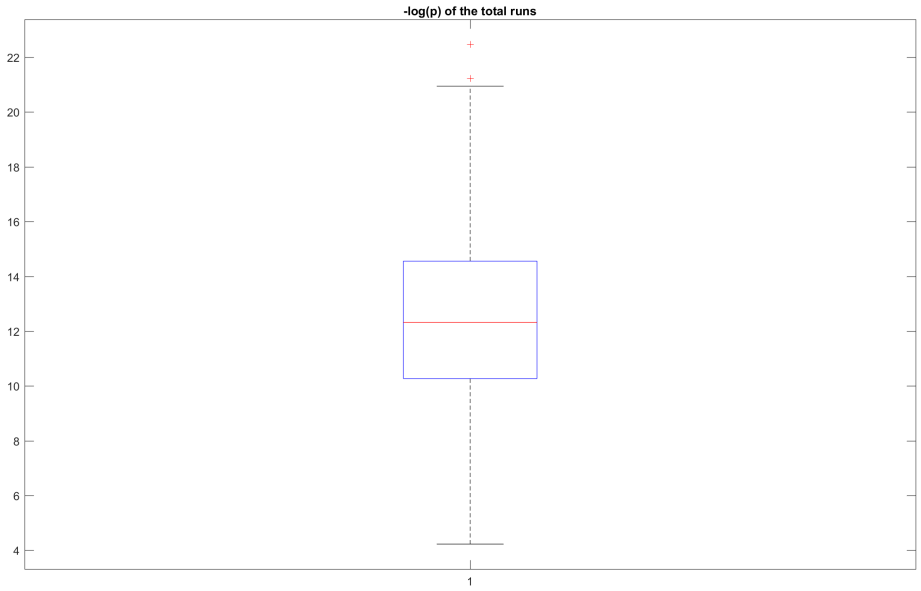


Figure 89: P-values of 100 right tail t-tests looking at the standard deviation

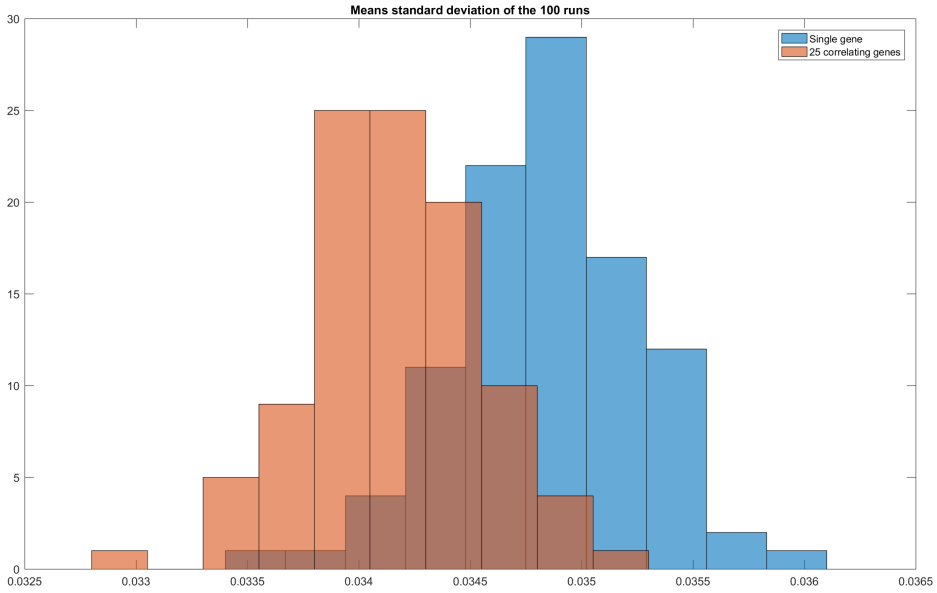


Figure 90: Two distributions of the average standard deviation

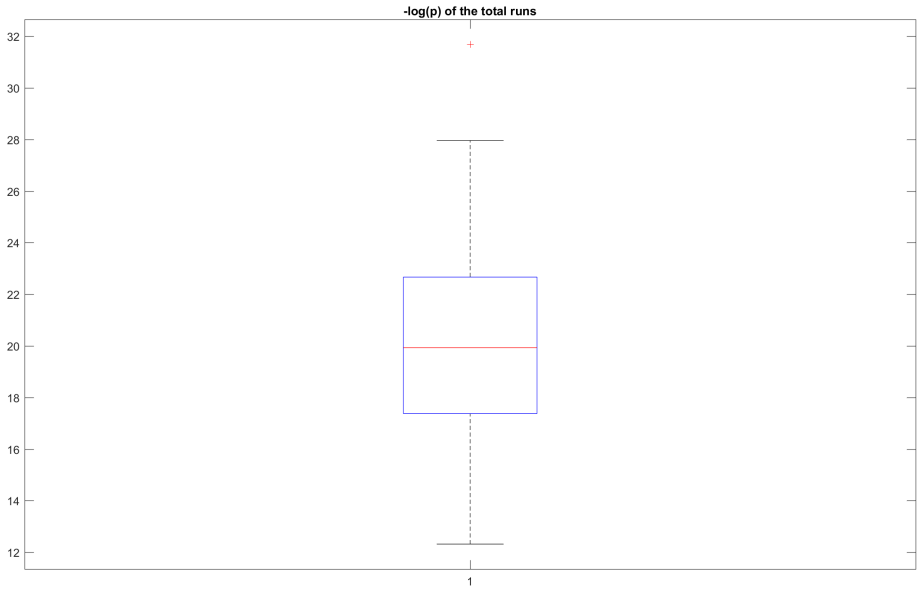


Figure 91: P-values of 100 left tail t-tests looking at the accuracy

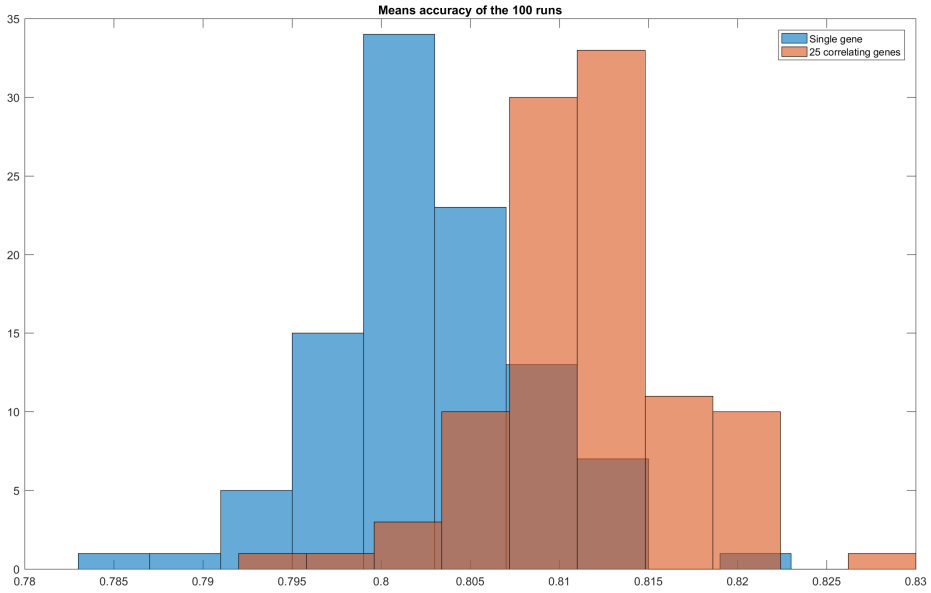


Figure 92: Two distributions representing the accuracies

B.2.3.5 Average of 36 correlated genes

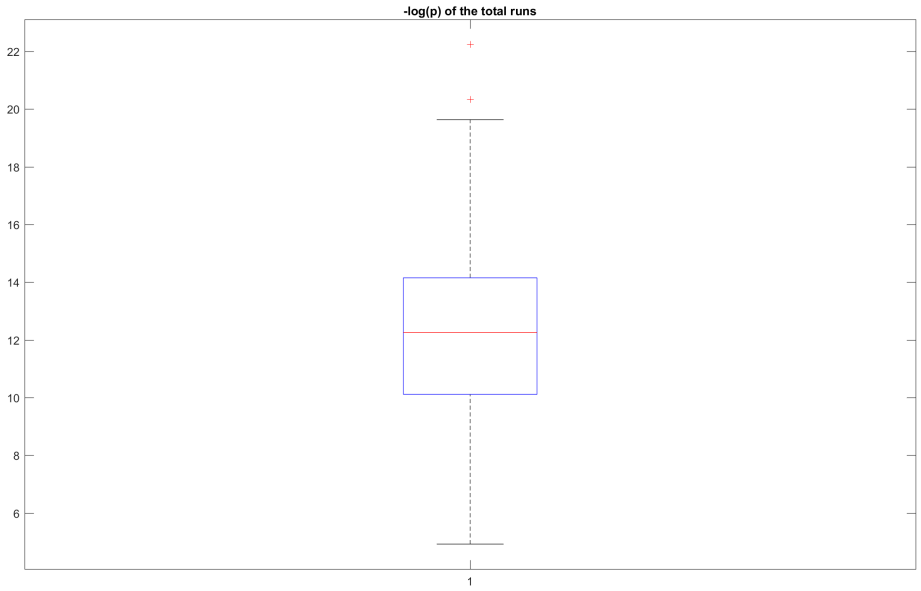


Figure 93: P-values of 100 right tail t-tests looking at the standard deviation

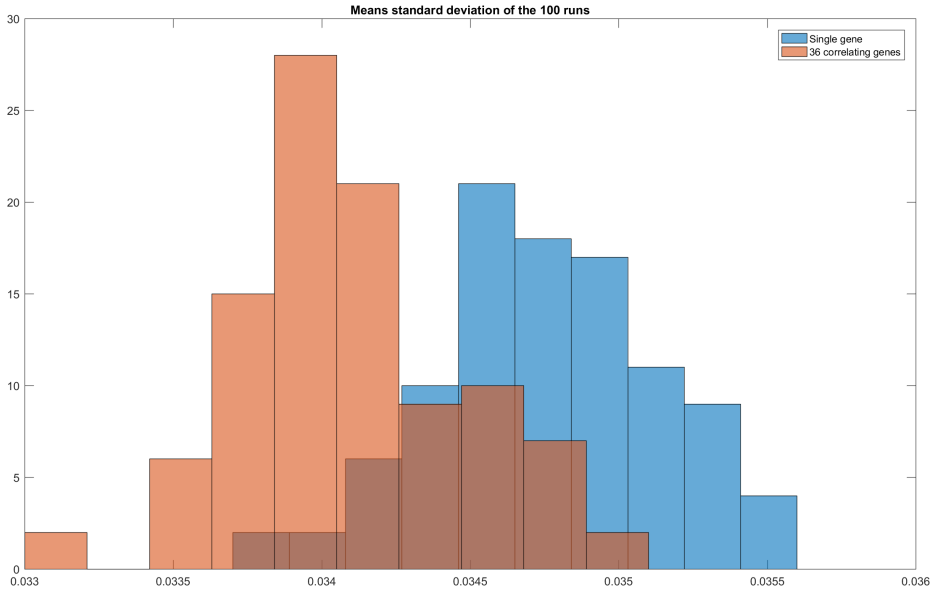


Figure 94: Two distributions of the average standard deviation

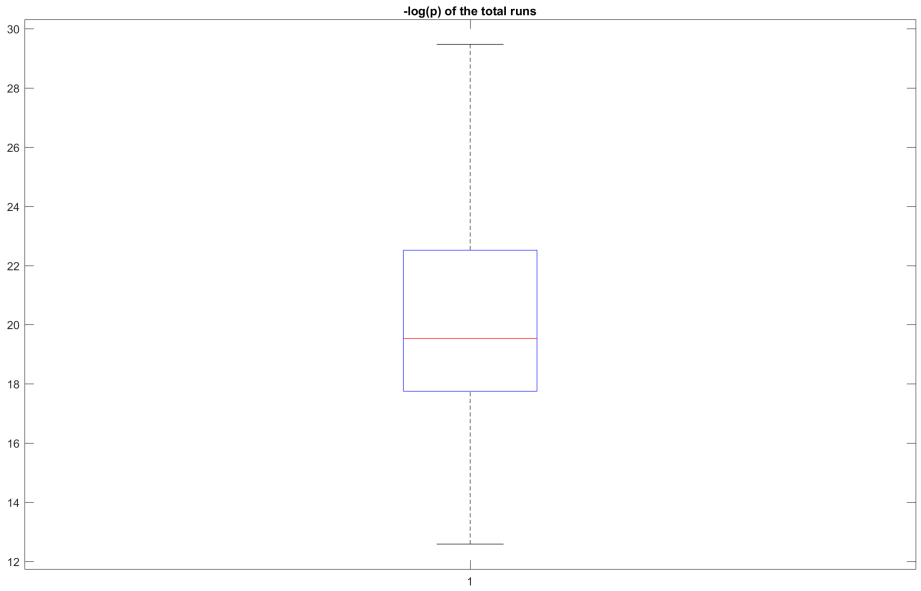


Figure 95: P-values of 100 left tail t-tests looking at the accuracy

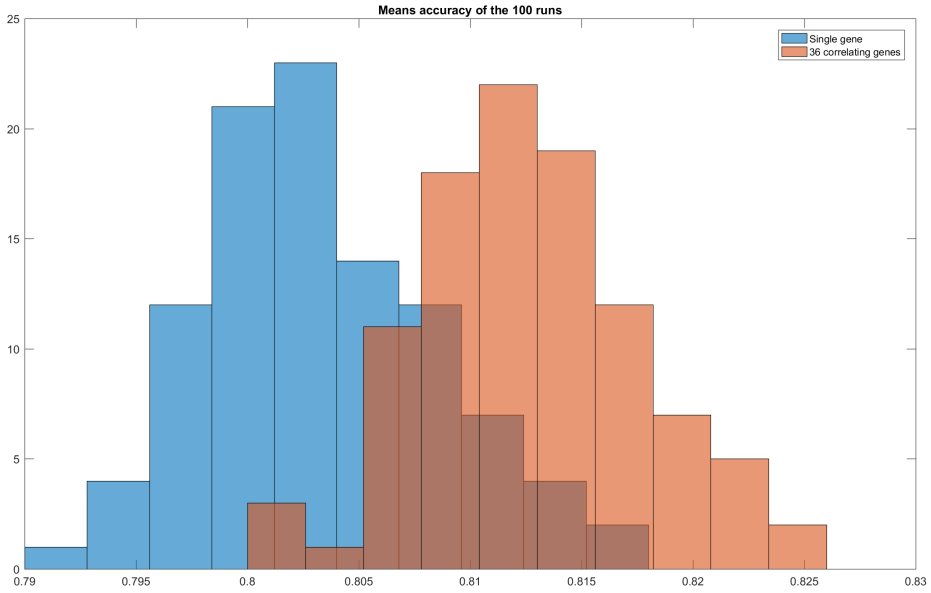


Figure 96: Two distributions representing the accuracies

B.2.3.6 Average of 49 correlated genes

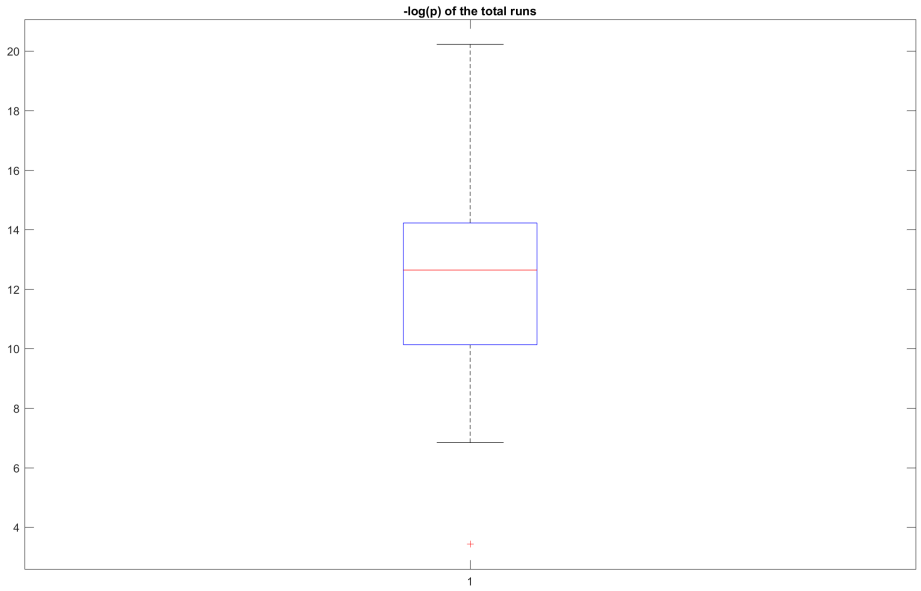


Figure 97: P-values of 100 right tail t-tests looking at the standard deviation

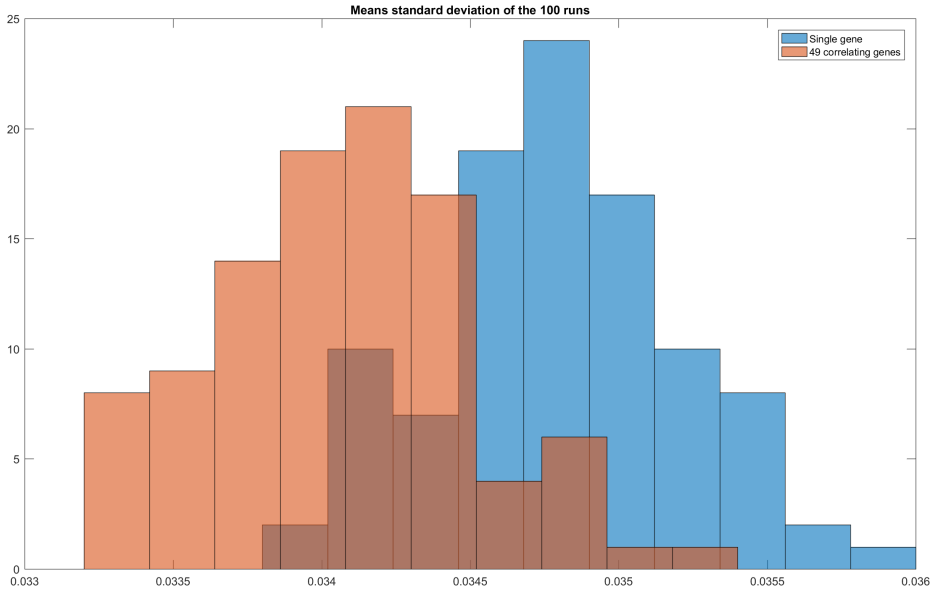


Figure 98: Two distributions of the average standard deviation

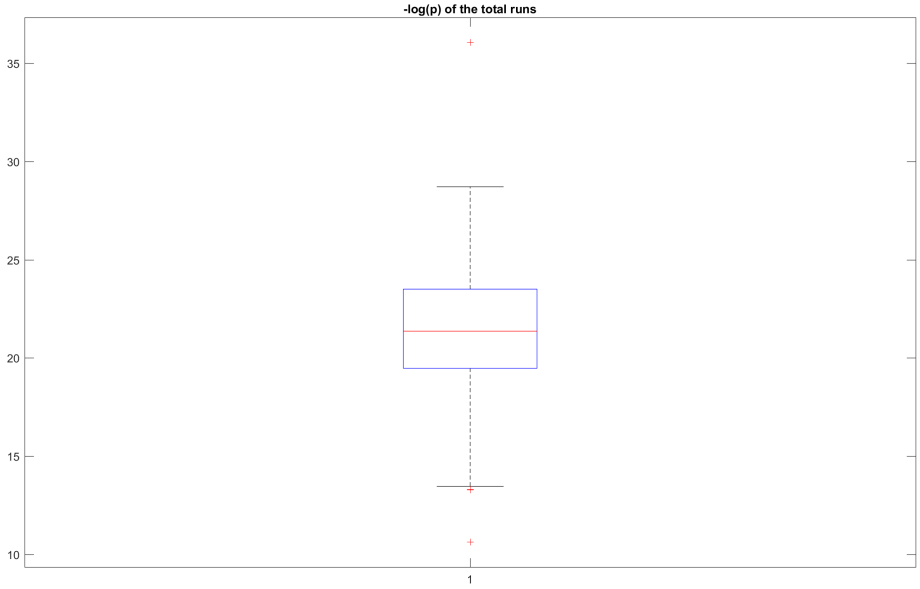


Figure 99: P-values of 100 left tail t-tests looking at the accuracy

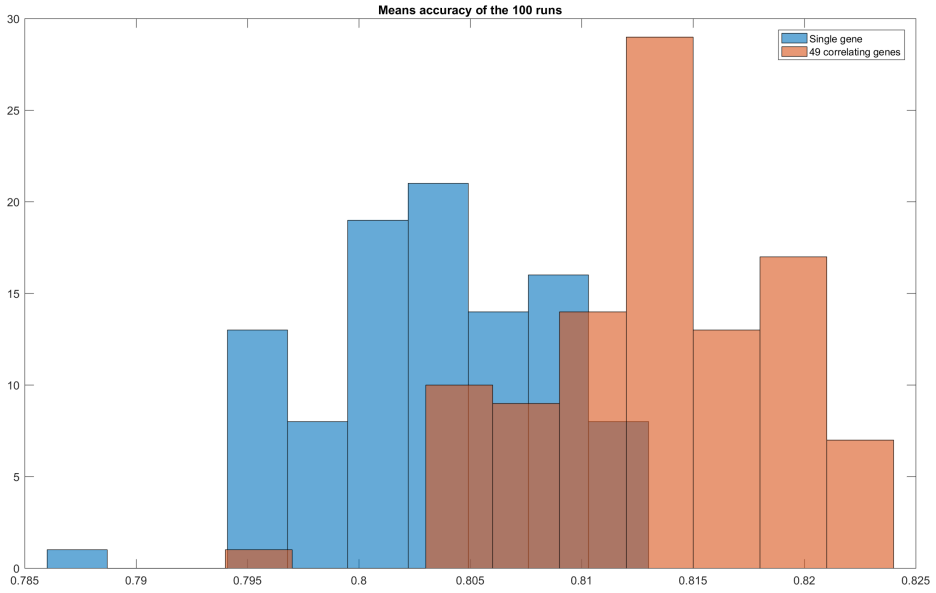


Figure 100: Two distributions representing the accuracies

C IME pseudo-code

Algorithm 4 iterative matrix expansion

INPUT: Points: 2d points (t-SNE result)

▷ the index corresponds to the index of the gene in the database.

OUTPUT: A matrix M representing the input points

$M \leftarrow [null]$

$P \leftarrow Points$

$xs \leftarrow$ lowest x value of points in P

$ys \leftarrow$ lowest y value of points in P

▷ Every point is at or to the upper right side of (xs_1, ys_1)

while $P \neq \emptyset$ **do**

$p \leftarrow$ a randomly chosen point in P

$x \leftarrow$ x index in the matrix

$y \leftarrow$ y index in the matrix

$$\triangleright\ x \leftarrow \max_{xs_i \in x} \{xs_i\},\ xs_i < p.x$$

$$\triangleright\ y \leftarrow \max_{ys_j \in y} \{ys_j\},\ ys_j < p.y$$

if $M_{yx} \neq null$ **then**

$dist \leftarrow$ displacement vector between p and $Points[M_{yx}]$

$mid \leftarrow$ midway point between p and $Points[M_{yx}]$

if $dist = [0, 0]$ **then**

Take the mean.

else if $dist.x > dist.y$ **then**

$extra \leftarrow (null, \ldots, null)^T$ ▷ $extra.size = y.size$

for $i \leftarrow 1 \ldots extra.size$ **do**

if $Points[M_{ix}] > mid.x$ **then**

$extra[i] \leftarrow M_{ix}$

$M_{ix} \leftarrow null$

$xs \leftarrow [x_1 \ldots x_x, mid.x, x_{x+1} \ldots x_{x.size}]$

$$M \leftarrow \begin{bmatrix} M_{11} & \cdots & M_{1x} & extra[1] & M_{1x+1} & \cdots & M_{1n} \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ M_{m1} & \cdots & M_{mx} & extra[m] & M_{mx+1} & \cdots & M_{mn} \end{bmatrix}$$

else

▷ The transpose of the above

else

$M_{yx} \leftarrow Points.indexOf(p)$

$P \leftarrow P \setminus p$

D Clustering pseudo-code

Algorithm 5 Clustering one layer

input list P containing all the points or clusters we wish to cluster.
define an list $inCluster$ of booleans keeping track if a cluster is already in a high level cluster
list $neighbors$ containing the closest neighbor of each point.
create a temporary one layer higher cluster c.
$index \leftarrow 0$
$id \leftarrow 0$
$res \leftarrow$ empty list of clusters
while $inCluster$ contains $false$ **do**
    if $inCluster[index] == false$ **then**
        add $P[index]$ to c
        $inCluster[index] \leftarrow true$
        $index \leftarrow neighbors[index]$
    else
        add c to res
        $id \leftarrow id + 1$
        create new cluster c with id as id
        $index \leftarrow$ index of first $false$ in $inCluster$.
return res

E First SIG Analysis

This is the SIG report send to us by Dennis Bijlsma on 03/06/2016 after submitting our code at SIG on 27/05/2016.

De code van het systeem scoort 4 sterren op ons onderhoudbaarheidsmodel, wat betekent dat de code bovengemiddeld onderhoudbaar is. De hoogste score is niet behaald door een lagere score voor Unit Size en Unit Interfacing.

Voor Unit Size wordt er gekeken naar het percentage code dat bovengemiddeld lang is. Het opsplitsen van dit soort methodes in kleinere stukken zorgt ervoor dat elk onderdeel makkelijker te begrijpen, te testen en daardoor eenvoudiger te onderhouden wordt. Binnen de langere methodes in dit systeem, is bijvoorbeeld de 'Mapping.closestClusters'-methode. Hier zijn aparte stukken functionaliteit te vinden welke ge-refactored kunnen worden naar aparte methodes. Het is aan te raden kritisch te kijken naar de langere methodes binnen dit systeem en deze waar mogelijk op te splitsen.

Voor Unit Interfacing wordt er gekeken naar het percentage code in units met een bovengemiddeld aantal parameters. Doorgaans duidt een bovengemiddeld aantal parameters op een gebrek aan abstractie. Daarnaast leidt een groot aantal parameters nogal eens tot verwarring in het aanroepen van de methode en in de meeste gevallen ook tot langere en complexere methoden. Voorbeelden in dit project voor methodes met hoog aantal parameters zijn de constructor van de klasse Data of de methode resultUtil.getTTest. Het aantal parameters kan worden verminderd door een nieuwe specifieke data type te introduceren. Bijvoorbeeld een type Dimensie kan de parameters width en height vervangen.

Het is goed om te zien dat jullie testcode hebben geschreven. De verhouding tussen de hoeveelheid test-code en de hoeveelheid code voor productie is op dit moment redelijk goed, 1:2. Hopelijk zal het volume van de test-code ook groeien op het moment dat er nieuwe functionaliteit toegevoegd wordt.

Over het algemeen scoort de code bovengemiddeld, hopelijk lukt het om dit niveau te behouden tijdens de rest van de ontwikkelfase.

F Second SIG Analysis

This is the SIG report send to us by Dennis Bijlsma on 28/06/2016 after submitting our code at SIG on 17/06/2016.

[Hermeting]

In de tweede upload zien we dat het codevolume is gegroeid, terwijl de score voor onderhoudbaarheid ongeveer gelijk is gebleven.

Bij zowel Unit Size als Unit Interfacing zien we dat jullie de genoemde voorbeelden hebben aangepast, maar het is jammer dat er in de nieuwe code weer een paar wat langere methodes toegevoegd, zoals bijvoorbeeld NearestNeighbors.calculateDistancesStrip. Hierdoor worden de verbeteringen in de bestaande code weer voor een klein deel ongedaan gemaakt.

Wel is het goed om te zien dat jullie door zijn gegaan met het schrijven van tests. De verhouding tussen testcode en productiecode is in vergelijking met de eerste upload wel iets achteruit gegaan, maar is nog steeds redelijk in orde.

Uit deze observaties kunnen we concluderen dat de aanbevelingen van de vorige evaluatie zijn meegenomen in het ontwikkeltraject.

G Original project description

Cancer is known to be caused by abnormal changes in DNA (i.e. mutations). Substantial efforts have been made to identify the regions in the DNA (i.e. genes) that cause cancer upon alteration. Variety of techniques can be used to recognize these deviations from healthy DNA. For example in gene expression measurement, the productivity of a gene in a cancerous cell is compare to its identical gene in a normal cell. A patient who has a DNA with many malfunctioning genes have a high risk of developing cancer.

It is known that each function in cell is carried out by collaboration of several genes and error in any of these functionally related genes might have a similar failure effect (i.e. cancer development). There are many resources available that illustrate gene-gene interactions in cell. However, not all deficiencies cause cancer. In this project, we are aiming to detect these functionally related genes that play an essential role in cancer progression.

In Bioinformatics and especially in this project, we extensively use artificial intelligence methods to facilitate the identification of cancerous groups of genes. To this end, we intend to employ the latest method in the field of artificial intelligence called deep learning. This revolutionary concept could exhibited a substantial improvement in the predictive power and performance over already existing models. For instance, using a particular type of deep learning algorithm called Convolutional Neural Networks (CNN), Google image search engine can know distinguish between objects with an accuracy close to a human.

Company description

This project will be supervised by one assistant professor (Dr. Jeroen de Ridder) and one PhD student (Amin Allahyar) that will meet weekly with the students to guard to overall progress and level of the project. For daily supervision, students can call upon the PhD student to discuss any encountered difficulty during the project.

Auxiliary information

(Jeroen Ridder) / Michel Reinders is the Client for this project, and Amin Allahyar is the TU Coach.

Info sheet

Title of the project: Deep Learning and identification of Cancer Related Sub-networks
Name of the client organization: Bioinformatics group, TU Delft
Date of the final presentation: June 24, 2016
Final report & code repository: https://github.com/Delfil/bep

Description

Cancer treatment is currently based on clinical parameters and pathological markers which don't capture all the complexity needed for giving treament based on the subtype of cancer. Subtyping based on multi-gene expression assays have previously been shown to be more precise than pathology based techniques for subtyping. Even with this fact it is not widely used with low availability and high cost being the main causes.
Our client requested us to create images of gene expression assays using the correlation between the gene expressions and to use these images as input for a convolutional neural network to classify the different breast cancer subtypes. The core challenge of this project was using gene expression correlations to create an image interpretable by a convolutional neural network. Given that the results are unknown not all requirements could be determined at the start of the project. We therefore used agile project management which means evolutionary development and early delivery. It also encourages rapid and flexible response to change. During the research phase we looked at tools that are easy to set up to be able to spend as much time explaining the results and providing proofs supporting these claims. Given the many parameters of a convolutional neural network multiple configurations have been tested. Therefore we developed an application that could run various configurations multiple times to test the accuracy of classification. These results could be expanded upon by adding more configurations without having to retrain the old configurations. We used this application to test the requested method, but it can be used in other projects to test various configurations of neural networks on other datasets. To test the intuition of the image creation methods we developed tests for determining the effect of integrating correlated gene expressions. We also determined the quality of the images of the proposed image creation algorithms. With the developed application and the two image creation algorithms we tested various configurations of convolutional neural networks to find the most optimal meta parameters.

Members of the project team

Name: Berend Ottervanger
Main contributions: Image creation & data analysis

Name: Sam Smulders
Main contributions: Application creation & data analysis

Name: Paul Bakker
Main contributions: Image creation & data analysis

Client & coach:

Name and affiliation of the client: Prof. Marcel Reinders, Bioinformatics group, TU Delft
Name and affiliation of the coach: Amin Allahyar, Bioinformatics group, TU Delft

Contacts:

Contact person 1: Berend Ottervanger, berendottervanger@gmail.com
Contact person 2: Sam Smulders, sam_smulders@live.nl
Contact person 3: Paul Bakker, paulbakker65@gmail.com
Contact person 4: Amin Allahyar, a.allahyar@tudelft.nl