Bioinformatics
Active regulatory regions prediction using deep neural networks

Paolo Cortis – 09539A

Introduction

In this project, deep supervised learning methods are applied to identify the position of active/inactive cis-regulatory regions (CRRs), including both enhancers and promoters, across the human genome.

These cis-regulatory regions are part of the non-coding DNA regions, which include 98% of the human genome, hence they do not encode for structural proteins and enzymes. CRRs are instead involved in the development of different cell tissues and play a crucial role in controlling the timing and intensity of gene expression. Through interactions with proteins such as histones and DNA binding transcription factors, there regions help specify the formation of diverse cell types and change physiological conditions. In fact, these specialized transcription factors indicate genes to RNA polymerase, by promoting or blocking his attachment to a particular coding region. Promoters specify and enable the positioning of RNA polymerase machinery at transcription initiation sites; they stand before a sequence to be coded by RNA polymerase to indicate a starting point. Enhancers modulate the activity of close promoters from linearly distal locations away from transcript initiation sites, increasing the likelihood that transcription of a particular gene will occur; hence, they can be far away in terms of base pairs but close in the 3D structure of the DNA.

The relevance in generating informative data for the detection of these regions is immense since it is essential to dissect the mechanisms underlying the modulation of gene expression and to understand the functional impact of genetic variants on human diseases. In fact, the effect of genetic variants in non-coding regions is strongly related to the prediction of active regulatory regions. If a genetic variant, even if potentially deleterious, is located in an inactive DNA region, it is less likely to be pathogenic. To support this, the Genome-wide association studies discovered thousands of variants associated with diseases and traits enriched in non-coding sequences, and several lines of research show that genetic variants in regulatory regions might be deleterious or directly involved in genetic diseases.

Task and Dataset

The tasks considered are predicting active promoters VS inactive promoters and active enhancers VS inactive enhancers. In practise, starting from data regarding a sequence of nucleotides in a genomic window of a promoter or enhancer, the goal is to classify it as active or inactive. The data considered regards *HG38*, cell line *HepG2* and was extracted from the following sources:

- ENCODE: *epigenomic data*. Real valued vector indicating the grade of histone modification of parts of the genome (float values representing the amount of interaction measured in that region) and TF binding data. This type of data was used to train feed forward neural networks.
- UCSC: *genomic sequence data*. Each genomic region is represented by a sequence of one-hot encoded nucleotides (A, C, G, T). To extract these sequences the bed coordinates were retrieved from the epigenomic data and used to get the genome portion for the HepG2 cell line. This type of data was processed with convolutional neural networks. It was also combined with the epigenomic data, wrapping together the two types of data, and used to train multi modal neural networks.
- FANTOM: *data labels*. FANTOM (Functional ANnoTation Of the Mammalian genome) is a worldwide collaborative project aiming at identifying all functional elements in mammalian genomes. Genomic regions are annotated with an activation value. These real valued labels were converted into discrete binary values to treat the problem as a binary classification task. Hence, a threshold was selected to label regions as active or inactive. This binarization is still an open problem and the literature doesn't provide a clear method to perform it nor a specific cut-off value. In this project the choice of the threshold was guided by the necessity to reach a reasonable balance between the classes and the will to not discard any piece of data, resulting in a promoter threshold equal to 1.0 and an enhancer threshold equal to 0.09.

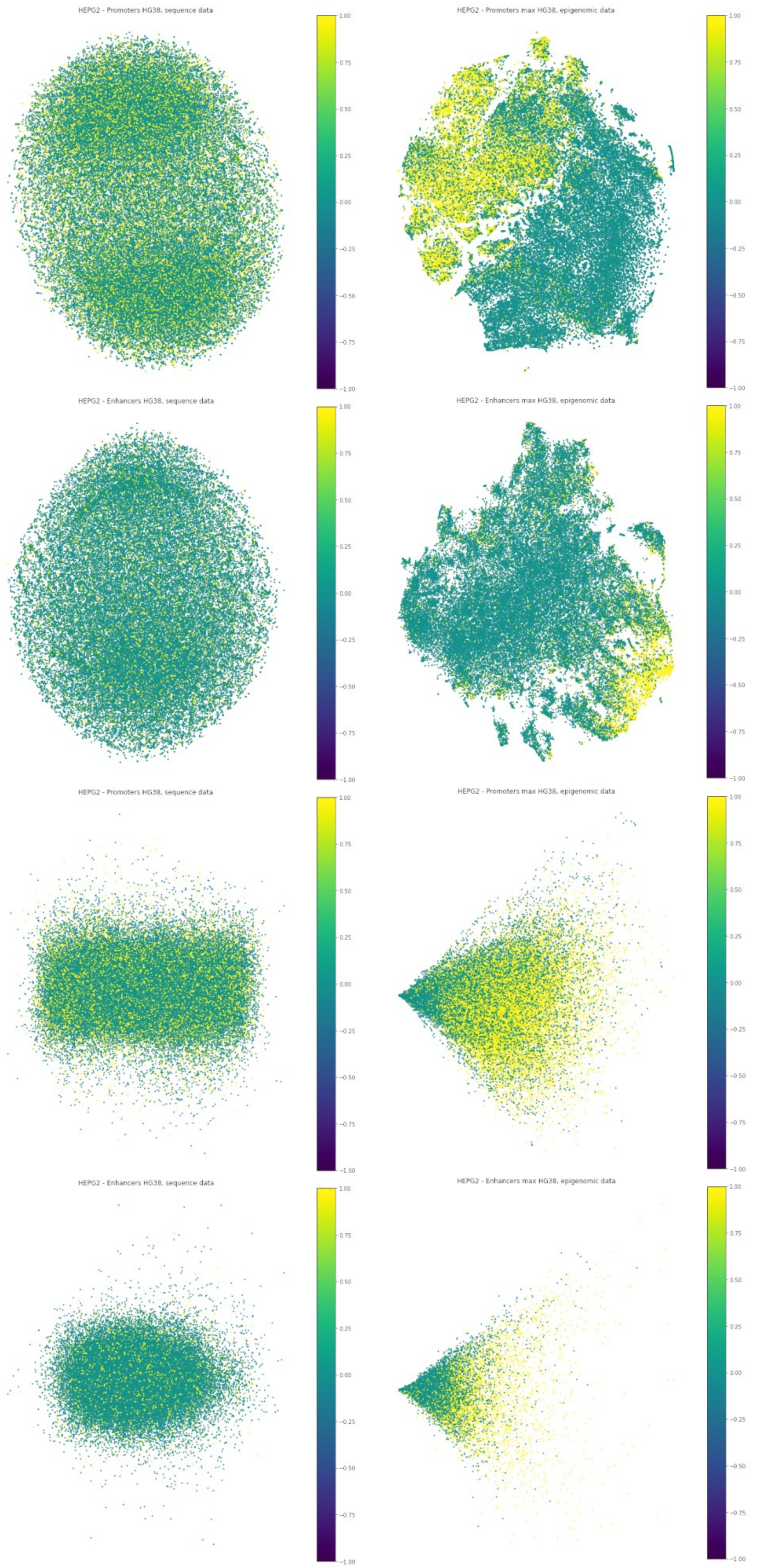
Data visualization

To visualize the feature space and the dataset distribution the *PCA* and *t-SNE* data decomposition algorithms were considered. This method refers to the application of decomposition methods to reduce an high-dimensionality dataset, meaning a lesser number of features are found to represent it. These techniques are here used for visualizing the dataset in a two-dimensional space.

Principal component analysis (*PCA*) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components. It is used to decompose a multivariate dataset in a set of successive orthogonal components that explain a maximum amount of the variance. This transformation is defined in such a way that the first principal component has the largest possible variance and each succeeding component in turn has the highest variance possible under the constraint that it is orthogonal to the preceding components.

T-distributed Stochastic Neighbour Embedding (*t-SNE*) is a nonlinear dimensionality reduction technique, based on Stochastic Neighbour Embedding, well-suited for embedding high-dimensional data in a low-dimensional space of two dimensions for visualization.

From these visualization it is evident how much more explicative and informative epigenomic data is for the task with respect to sequence data. Especially when looking at the t-SNE decomposition for epigenomic data, these graphs suggest that a good machine learning model could tackle the classification task.



From top to bottom:
Promoters sequence data (t-SNE)
Enhancers sequence data (t-SNE)
Promoters sequence data (PCA)
Enhancers sequence data (PCA)

Promoters epigenomic data (t-SNE)
Enhancers epigenomic data (t-SNE)
Promoters epigenomic data (PCA)
Enhancers epigenomic data (PCA)

Data pre-processing and feature selection

After the data retrieval phase, it is crucial, for all types of machine learning applications, to check thoroughly the dataset and apply some kind of *pre-processing*, in fact, even the best model cannot learn much from poor, noisy and inconsistent data.
The following methods have been applied to *epigenomic data only*, since for sequence data pre-processing and feature selection hardly applies.

A first check is the ratio between the number of features and the number of samples, if close or less than one, the model would behave erratically and overfit. Afterwards a check for Nan values was performed, as these values have a detrimental impact on the performance of the model. If a particular sample or feature shows an abundance of Nan values, dropping that row or column completely could be considered, in this case the approach was imputation. There are numerous approaches to reconstruct Nan values; in this case a K-nearest neighbour logic was applied, meaning that a missing value is imputed with the average value of the K-nearest datapoints to it. Data normalization is proven to improve the performance of models and was considered as the next pre-processing step using a robust scaler. The robust scaler operates normalization by subtracting the median and dividing by the standard deviation between the interquartile range. This results in a z-scoring while being robust to outliers.

Feature selection implies the choice of a subset of the original pool of features present in the dataset. This is a relevant step as data often contains some features that are either redundant or irrelevant and can negatively affect the model performance; hence, feature selection reduces noise, improves model accuracy, and reduces training time by cutting the total number of features without incurring in much loss of information. To this end, identifying *feature correlation* with the output and correlation between features is helpful. This is because features not correlated with the output do not carry relevant information for the model to exploit for the task and can thus be removed; the same goes for features correlated with other features, these carry the same kind of information and result redundant. The correlation tests considered in this project to perform feature selection are:

- Pearson test: identifies linear correlation between each feature and the output. This test was also considered between features to identify those correlated with each other.
- Spearman test: identifies monotonic correlation between each feature and the output.
- Maximal information coefficient (Mic) test: identifies non-linear correlation between each feature and the output. Because of its computational complexity, this test was applied only to confirm the correlations found by the two previous experiments.
- Boruta algorithm: a wrapper built around the random forest classification algorithm, it gives a numerical estimate of the feature importance, allowing to separate relevant features from irrelevant ones. The algorithm categorizes features in those to keep, those to discard, and *tentative*, meaning that there is uncertainty regarding their actual correlation with the output. In this project, both uncorrelated and tentative features were discarded. Experiments were

performed when discarding only strictly uncorrelated features, the results, however, were found to be similar.

The feature selection step cannot be applied once to the entire dataset, as this would introduce *bias* into the following models due to *data leakage*. Instead, these tests were reserved for the *training main* loop and ran, for each holdout, over the training portion of the data only, maintaining test data unseen, and saving it for actual testing.

Models

The models developed in this project are *Feed Forward Neural Networks* (FFNN) trained on epigenetic data, one-dimensional *Convolutional Neural Networks* (CNN) trained on DNA sequence data and *Multi Modal Neural Networks* (MMNN) to combine the previous two models and predict using both epigenomic and sequence data.

The feed forward neural network consists of one input and output layer with a certain number of hidden densely connected layers in between. The activation function of these layers is the *Relu* function; this is the most commonly used for hidden layers since it avoids the vanishing gradient problem during training, while a sigmoid function was employed for the output layer. To avoid *overfitting* of the model (learning the training set up to the noise without being able to generalize over unseen data) the regularization technique of *dropout* was implemented. The idea is to partially limit the network capabilities during training by “shutting off” some neurons. This forces the rest of the network to learn different aspects of the input and pick up more robust features, which in turn helps generalization. To implement dropout within the model, dropout layers need to be added within the architecture.
For this type of architecture, experiments were carried out *with and without Boruta feature selection* to inspect the possible impact on the performance.

A convolutional neural network is great at handling fixed size sequence data, in this case the genome sequence. This type of network is based around the operation of convolution, consisting in the application of a filter to the input to create a feature map that identifies patterns and relevant features within the data. The *Relu* activation function is used for these layers as well, to keep only the portion of the input where the kernel is responsive, negative values are set to zero. This allows clear identification of patterns and reduction of noise, making learning simpler. After each *convolutional* layer a *max pooling* layer is necessary to down sample, keeping only the maximum values within the feature map region, thus, the output is a feature map consisting of the most prominent portions of the previous one. Regarding the application of dropout to convolutional neural networks there is a debate regarding his application over convolutional layers. In fact, the effect of a dropout layer is well defined over dense layers but less understandable over convolutional ones, in this project it was applied as it was deemed to have a positive impact on the model performance. The resulting building block of convolutional neural network consists of a convolutional layer, followed by a max-pooling layer; multiple blocks of this kind compose the full network. When defining a convolutional layer, the *number of filters* applied must be specified, as well as the *size of the kernel*, *padding* is also used to avoid changing the shape of the data when operating the convolution. The results of the convolutional portion of the network are then passed to a set of dense layers to obtain the final prediction.

Data usually comes with different modalities, which carry different information, like in this case where to predict the activation of cis-regulatory regions both epigenomic and sequence data carry useful information. Multimodal learning attempts to model the combination of different modalities of data, in this case via a multi-modal neural network, a model that is able to jointly represent and exploit the information of both data modalities (epigenomic data and genomic sequence data). To shape this network the layers of a FFNN and a CNN are concatenated, each receiving their specific input, the outputs are then combined into a final dense layer and output layer.
For this type of architecture experiments were carried out with *fixed* FFNN and CNN architectures, as well as with *Bayesian optimized* FFNN and CNN architectures, to inspect the possible impact on the performance. To train the model the approach used was to train the FFNN and the CNN at the same time as a whole MMNN.

Each model is constructed using the *Tensor Flow Keras* library and compiled using the same training loss function and optimizer. In this project the *binary cross-entropy* is used, a standard loss function for the binary classification task. Equally important is the optimizer, which influences the weight update of the network; in this case *nadam* is used, an extended version of stochastic gradient descent, which is a standard choice for deep learning. Other relevant training parameters are the *batch size* and the *number of epochs*. The batch size defines the number of samples the model passes through before updating the internal parameters; it was set to 256. An epoch is a full pass over the entire training set, and the number of epochs defines the number of times the algorithm goes through it during the training procedure, in the project it was set to 1000. This is a purposely high value that makes sense when paired with the *early stopping* call-back. Even with regularization techniques in place when the number of epochs start to increase overfitting becomes inevitable, hence early stopping monitors the model test loss and stops training if the performance gets worst.

Hyperparameter tuning

When designing a neural network, the choice of the architectural shape (number of layers, neurons, activation functions) and setting of learning hyperparameters (optimizer algorithm, batch size, learning rate) are critical for achieving reliable and high performances. At the state of the art, there is no unified method for finding the appropriate hyperparameters for a given task, and model selection is generally performed by relying on experience, involving empirical tests, or applying automatic methods, which explore the hyperparameter space in a bounded domain.

Bayesian optimization is a technique that learns from the observed performance of previously tried hyperparameter settings on the current task. This knowledge then helps to build a *meta-model* that can be used to predict which unseen configurations may work best on the task. It is a sequential design strategy for global optimization of black-box functions, and it is usually employed to optimize expensive-to-evaluate functions, in this case the training of a neural network. This method has proven to be an effective and cost-efficient solution to hyperparameter optimization and is the hyperparameter tuning method chosen for this project.

The hyper model is defined by specifying, for each of the critical hypermeters, a range of possible values to explore for a set number of trails. The procedure then trains this meta-model using training data and tests it with validation data. Subsequently it returns the set of hyperparameters found that optimized the *accuracy*, *AUROC* and *AUPRC* metrics (deemed to be the most relevant for the task at hand). When choosing the parameters to evaluate and their ranges, which define the search space, the goal was to strike a balance between the coverage of the hyperparameter space and the computational time required. Enlarging further the spectrum of the optimized learning parameters, as well as the range of exploration, may further improve the model performance.

For the FFNN the hyperparameters to set were the number of hidden layers (1-5), the number of neurons per layer (8-136) and the amount of dropout per layer (0.3, 0.4 or 0.5). It is interesting to see how, for most holdouts, the hyper model opted for an architecture with a single hidden layer of 72-136 neurons.
This result does not really follow the accepted concept of having more “slim” layers rather than few “fat” ones, it is unclear if this is the actual best configuration for this specific problem and dataset or if the algorithm simply found an easy exploit to have good but inconsistent results on the specific holdout configuration, still the results are respectable.

For CNN, since the computationally expensive Boruta algorithm was not part of the loop, a more complex hyperparameter space was defined, allowing more trials to explore the configuration space. In this case, after some experimenting, the value ranges were chosen to accommodate the preference of the hyper model during the different tests. The hyperparameters to set were the number of convolutional layers (1-6), the number of filters per layer (8-160), the kernel size per layer (6, 8, 12 or 16), the convolutional dropout rate per layer (0.0, 0.1 or 0.2), as well as the number of the following dense layers (1-4), the number of neurons per dense layer (16-128) and the dense dropout rate (0.3, 0.4 or 0.5).
In this case, the optimization results confirmed the general guidelines in terms of CNN design, a number of convolutional layers ranging from 3-6 with a progressively decreasing number of filters.

When designing the multi modal neural network architecture a fixed FFNN and CNN, as well as the Bayesian optimized FFNN and CNN for each holdout were combined.

Model training and evaluation methods

In order to train a model, tune his hyperparameters with a meta-model, and evaluate his performance, the dataset must be split into *training*, *validation* and *testing* set.

To have a statistically sound estimate of an architecture performance, multiple models are built and trained, each with the same architecture, over different portions of the data (*holdouts*) and, the average performance of those, is considered as an estimate of the overall performance of the architecture. In this project, the technique used to generate the holdouts was the *stratified Monte-Carlo* method, producing each time a different arrangement of the training, validation and testing set, while keeping roughly the same *class balance*. 10 holdouts were considered to train and evaluate each model, 20% of the whole dataset was used for testing, 80% for training where 20% of it was reserved for the validation of the meta-model. The same could be done for hyperparameter tuning, meaning hyperparameters are tuned different times for different splits of the training and validation set, and the best possible hyperparameters of the model are found as the average of those sets. This nested-fashion procedure means that the optimal set of hyperparameters found depends only on the specific training set considered at each holdout and may very well be different at each iteration of the loop. However, to avoid increasing excessively the computational complexity of the main loop, the hyperparameter tuning step was performed, for all holdouts, considering a single split of the training set in a training and validation portion.

Evaluation procedure notes:

- Both the removal of uncorrelated features and the hyperparameter tuning step are performed over the training set only, so to never have *data leakage* and introduce *bias* into the model.
- Both the uncorrelated features and the optimal set of hyperparameters found depend on the *specific* training set considered at each holdout and may be different at each iteration of the main loop.
- The final performance estimation is that of the model trained across different portions of the data (10 holdouts) when the removal of uncorrelated features and the tuning of hyperparameters is performed, at each holdout, over the specific training set.

The main training loop:

- Tune hyperparameters with the meta-model based on Bayesian optimization using the training set (split into train and validation) for 1 holdout. The result is the best set of hyperparameters found using that specific training set. For MMNN this step was also avoided to assess the fixed FFNN and CNN performances.
- For epigenomic data, remove features uncorrelated with the output performing the Pearson and Spearman correlation tests, the Mic correlation test to confirm the choice, remove features correlated with each other applying the Pearson correlation, and the Boruta feature selection algorithm using the training set. For FFNN the Boruta algorithm was also avoided to assess the eventual difference in the performances. For CNNs and MMNNs the Boruta feature selection algorithm was not considered.

- Build and train the model over the entire training set with the optimal set of hyperparameters found by the meta-model and test it over the test set. For MMNN the fixed FFNN and CNN were also considered.

This is considered to be enough for the experimental nature of this project. In case an actual model was to be developed, an idea would be to deploy the best performing one across the 10 generated.

When evaluating each model performance, a plethora of metrics are available, in this project, the *accuracy*, the *AUROC* (Area Under Receiver Operating Characteristic) and the *AUPRC* (Area Under the Precision-Recall Curve) were deemed to be the most relevant. The accuracy returns the number of correct predictions over all samples considered. The AUROC is the area under the curve where the x axis is the false positive rate and the y axis is the true positive rate. The AUPRC, especially relevant for unbalanced class problems, is the area under the curve where the x axis is the recall and the y axis is the precision. These metrics were computed for both the training and testing portion of the data at each holdout. After the stratified holdout procedure is complete, an *average* of these values is performed, returning the *10-holdout performance estimate* of the architecture. An historical report of these parameters is also kept during the training of each model, this permit plotting the evolution of these metrics over the epochs, ultimately allowing a better evaluation of the model behaviour.

Results and considerations

The *Wilcoxon* signed-rank test is a non-parametric statistical hypothesis test, which checks the null hypothesis that two related paired samples come from the same distribution. In the scope of this project, this test was used to determine whether there was any *statistically significant* difference in the results produced by two different model architectures. The comparison carried, and the related conclusions, are hereby summarized:

- FFNN – Promoters and Enhancers – *Boruta model performances VS Non-Boruta model performances*: the two results are statistically indistinguishable; hence, even if the performances are slightly better when using Boruta feature selection, this computationally expensive algorithm is of no particular use in this case.

```
FFNN - Promoters - Wilcoxon test - Boruta model performances VS Non-Boruta
model performances:
Accuracy: WilcoxonResult(statistic=15.0, pvalue=0.232421875)
AUROC: WilcoxonResult(statistic=14.0, pvalue=0.193359375)
AUPRC: WilcoxonResult(statistic=21.0, pvalue=0.556640625)

FFNN - Enhancers - Wilcoxon test - Boruta model performances VS Non-Boruta
model performances:
Accuracy: WilcoxonResult(statistic=21.0, pvalue=0.556640625)
```

```
AUROC: WilcoxonResult(statistic=3.0, pvalue=0.009765625)
AUPRC: WilcoxonResult(statistic=19.0, pvalue=0.431640625)
```

- Promoters and Enhancers – *FFNN performances VS CNN performances*: the two results are found to be statistically different; this follows the literature consensus, as it is known that sequence data, used to train CNNs, is less effective than epigenomic data, used to train FFNNs, for the given task.

```
Promoters - Wilcoxon test - FFNN VS CNN:
Accuracy: WilcoxonResult(statistic=0.0, pvalue=0.001953125)
AUROC: WilcoxonResult(statistic=0.0, pvalue=0.001953125)
AUPRC: WilcoxonResult(statistic=0.0, pvalue=0.001953125)

Enhancers - Wilcoxon test - FFNN VS CNN:
Accuracy: WilcoxonResult(statistic=2.0, pvalue=0.005859375)
AUROC: WilcoxonResult(statistic=0.0, pvalue=0.001953125)
AUPRC: WilcoxonResult(statistic=0.0, pvalue=0.001953125)
```

- Promoters and Enhancers – *MMNN with fixed FFNN and CNN VS MMNN with Bayesian optimized FFNN and CNN*: the two results are statistically indistinguishable; hence, even if the performances are slightly better when using Bayesian optimized networks, there is not any statistical difference with the fixed networks in this case.

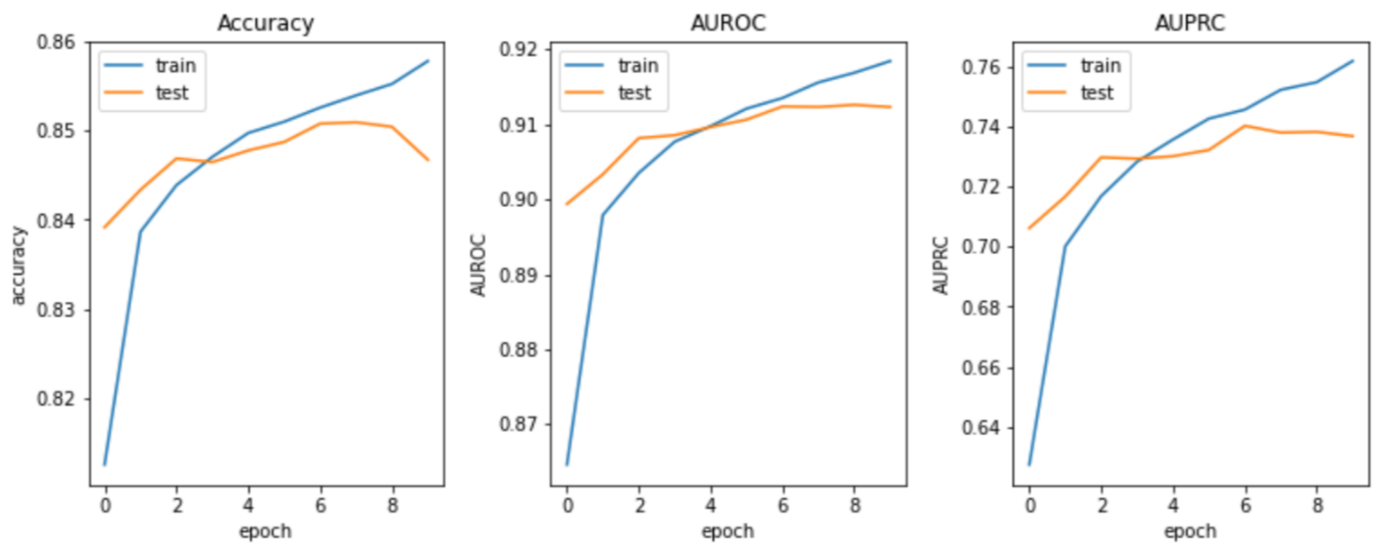
```
MMNN - Promoters - Wilcoxon test - fixed FFNN and CNN performances VS
optimized MLP and CNN performances:
Accuracy: WilcoxonResult(statistic=25.5, pvalue=0.921875)
AUROC: WilcoxonResult(statistic=10.0, pvalue=0.083984375)
AUPRC: WilcoxonResult(statistic=7.0, pvalue=0.037109375)

MMNN - Enhancers - Wilcoxon test - fixed FFNN and CNN performances VS
optimized MLP and CNN performances:
Accuracy: WilcoxonResult(statistic=16.0, pvalue=0.275390625)
AUROC: WilcoxonResult(statistic=11.0, pvalue=0.10546875)
AUPRC: WilcoxonResult(statistic=21.0, pvalue=0.556640625)
```

The 10-holdouts estimates of all the architectures tested, and the related conclusions, are hereby summarized:

```
FFNN - Promoters - withOUT BORUTA feature selection - 10-holdouts estimate:
Accuracy: train - 0.8585627753303966 -- test - 0.8457426039945937
AUROC: train - 0.9228826231988739 -- test - 0.908225762344436
AUPRC: train - 0.7754590575471727 -- test - 0.730489182316224

FFNN - Promoters - with BORUTA feature selection - 10-holdouts estimate:
Accuracy: train - 0.8610082098518221 -- test - 0.8482404765480303
AUROC: train - 0.926005039875637 -- test - 0.9115797167326514
AUPRC: train - 0.782136794629508 -- test - 0.7343833693474171
```

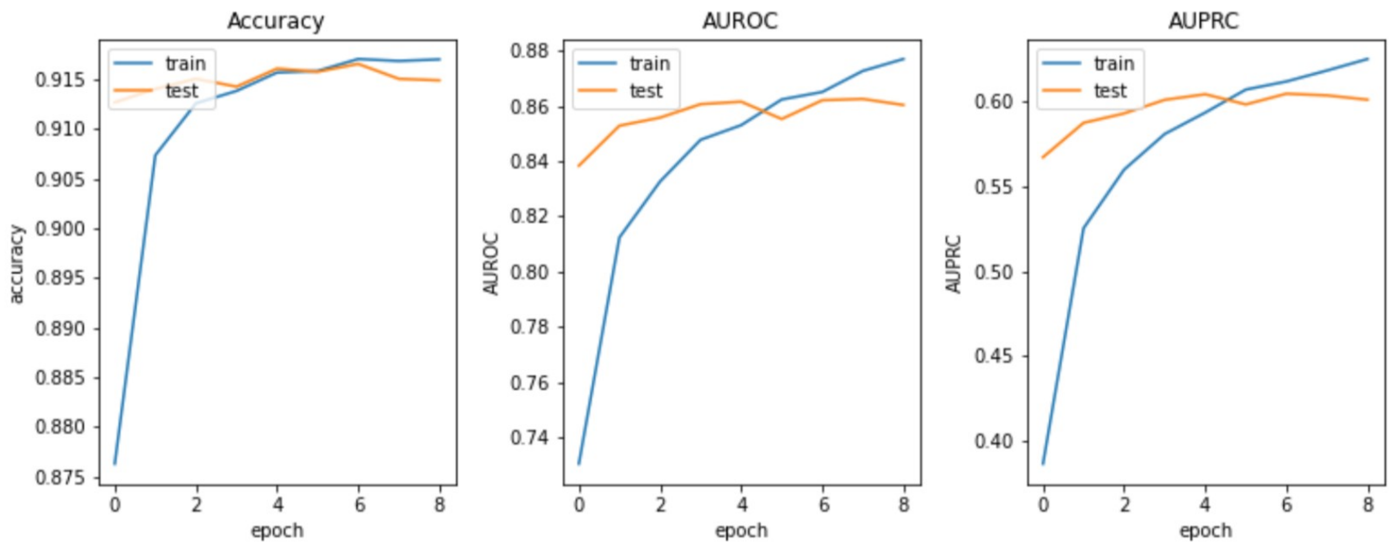


1 - FFNN - Promoters - Training History

Graphs referring to a single holdout, graphs from other holdouts show a similar trend

```
FFNN - Enhancers - withOUT BORUTA feature selection - 10-holdouts estimate:
Accuracy: train - 0.9105297463853992 -- test - 0.9092043928261042
AUROC: train - 0.8975376189283605 -- test - 0.8528177944393699
AUPRC: train - 0.6178753437988747 -- test - 0.5708571083326558

FFNN - Enhancers - with BORUTA feature selection - 10-holdouts estimate:
Accuracy: train - 0.9148712175080982 -- test - 0.9137078296594769
AUROC: train - 0.88410107552469 -- test - 0.8576130727534329
AUPRC: train - 0.62724966027985 -- test - 0.5990892730926193
```

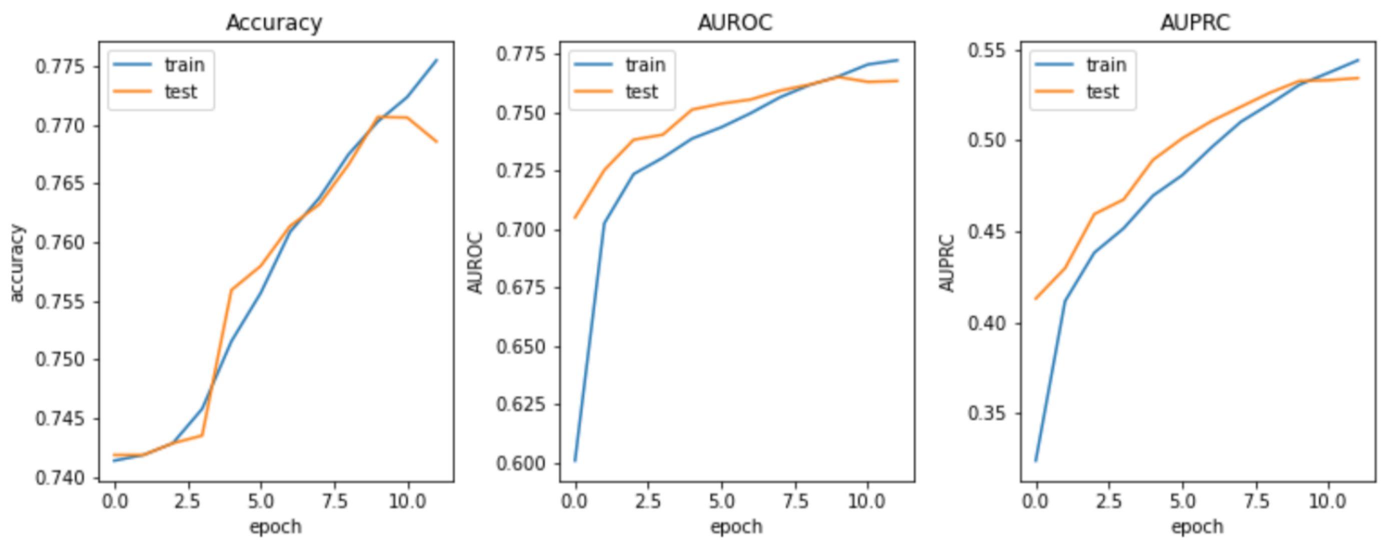


2 - FFNN - Enhancers - Training history

Graphs referring to a single holdout, graphs from other holdouts show a similar trend

The *FFNN* trained over epigenomic data achieves an accuracy of about 84.6%, an AUROC of about 0.91 and an AUPRC of about 0.73 over the testing set for promoters, and an accuracy of about 91%, an AUROC of about 0.85 and an AUPRC of about 0.58 over the testing set for enhancers.
These results are respectable, even if the training history graphs show some signs of overfitting, proving that a FFNN is indeed a good fit for the task when epigenomic data is considered. These results also prove what was already clear from the Wilcoxon test: Boruta feature selection does not appear to improve the results in any significant manner.

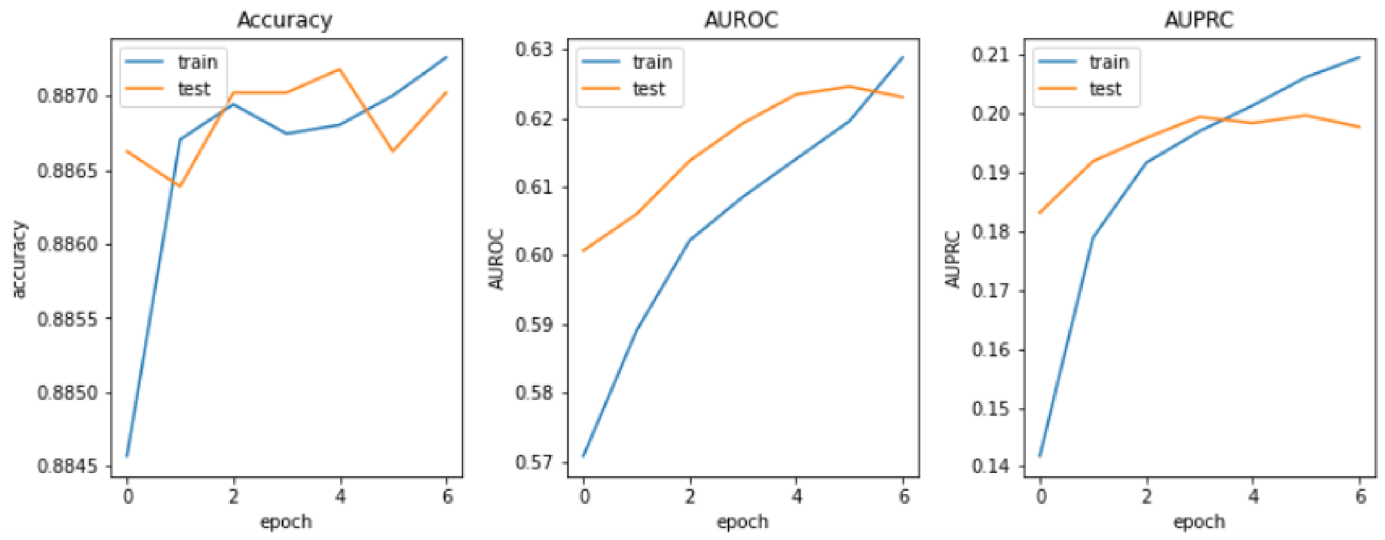
```
CNN - Promoters - 10-holdouts estimate:
Accuracy: train - 0.8364537358283997 -- test - 0.771271961927414
AUROC: train - 0.865492069721222 -- test - 0.765926045179367
AUPRC: train - 0.727402257919312 -- test - 0.542338216304779
```



3 - CNN - Promoters - Training history

Graphs referring to a single holdout, graphs from other holdouts show a similar trend

```
CNN - Enhancers - 10-holdouts estimate:
Accuracy: train - 0.8868017852306366 -- test - 0.8867424964904785
AUROC: train - 0.6695534765720368 -- test - 0.626373231410980
AUPRC: train - 0.2362850889563561 -- test - 0.207713982462883
```

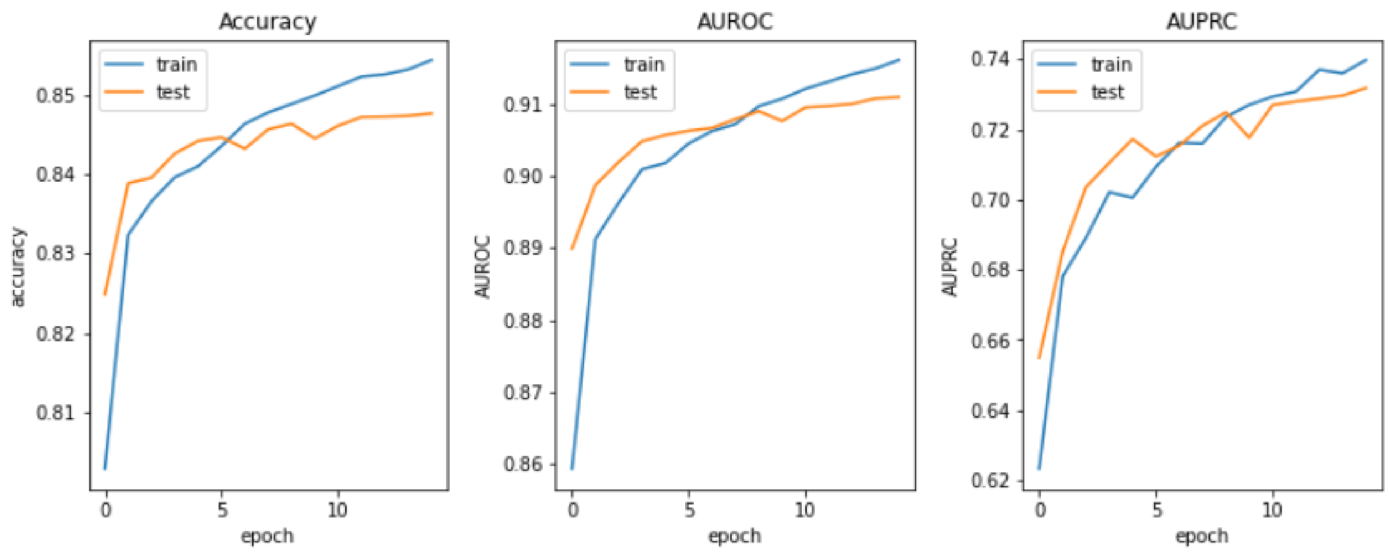


4 - CNN - Enhancers - Training history

Graphs referring to a single holdout, graphs from other holdouts show a similar trend

The *CNN* trained over genomic sequence data achieves an accuracy of about 77.1%, an AUROC of about 0.77 and an AUPRC of about 0.54 over the testing set for promoters, and an accuracy of about 88.7%, an AUROC of about 0.63 and an AUPRC of about 0.21 over the testing set for enhancers.
These results are a lot worse than those of the FFNN, as suggested by the Wilcoxon test, proving that sequence data is indeed much harder to interpret and less significant when it comes to the task at hand. This is especially true for enhancers the results, particularly considering AUPRC, are abysmal, further experiments proved that this has little to do with the architecture and the model capabilities, it is more likely due to the enhancers class imbalance and the way the binarization was performed.

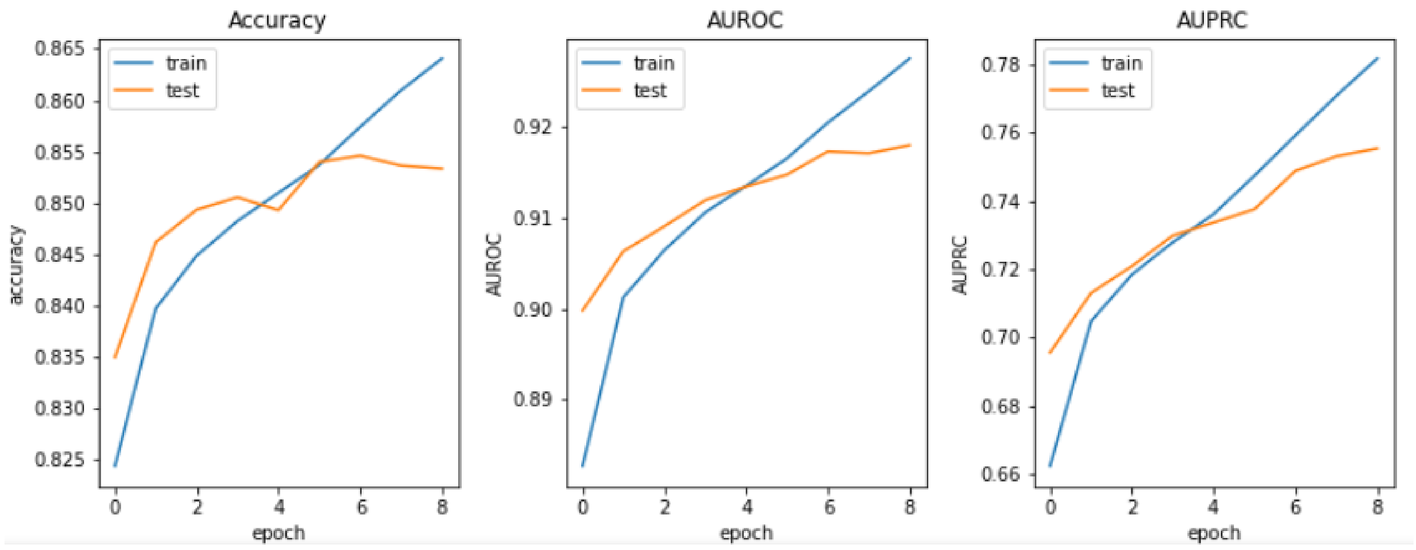
```
MMNN - Promoters - with Fixed FFNN and CNN - 10-holdouts estimate:
Accuracy: train - 0.8591259539127349 -- test - 0.8498673617839814
AUROC: train - 0.922665506601334 -- test - 0.9122659921646118
AUPRC: train - 0.757349681854248 -- test - 0.7347532510757446
```



5 - MMNN – Promoters - Fixed - Training history

Graphs referring to a single holdout, graphs from other holdouts show a similar trend

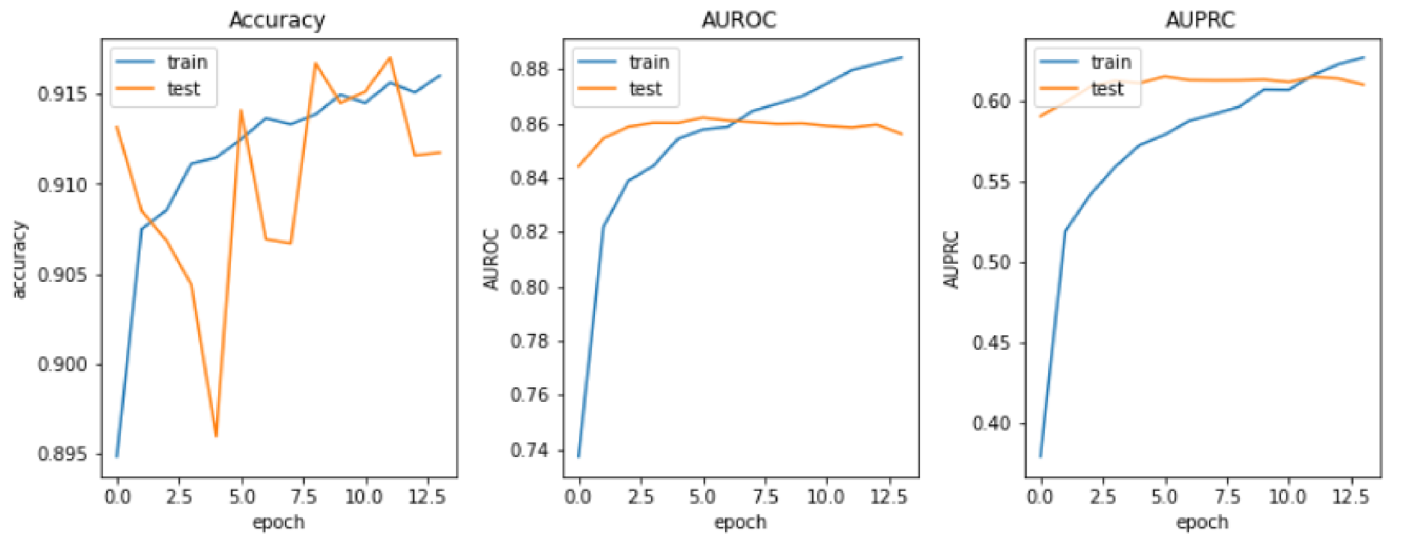
```
MMNN - Promoters - with Bayesian optimized hyperparameters FFNN and CNN -
10-holdouts estimate:
Accuracy: train - 0.8814577519893646 -- test - 0.8502327740192414
AUROC: train - 0.9431495547294617 -- test - 0.9146451532840729
AUPRC: train - 0.8385738432407379 -- test - 0.7485358357429505
```



6 - MMNN – Promoters - Optimized - Training history

Graphs referring to a single holdout, graphs from other holdouts show a similar trend

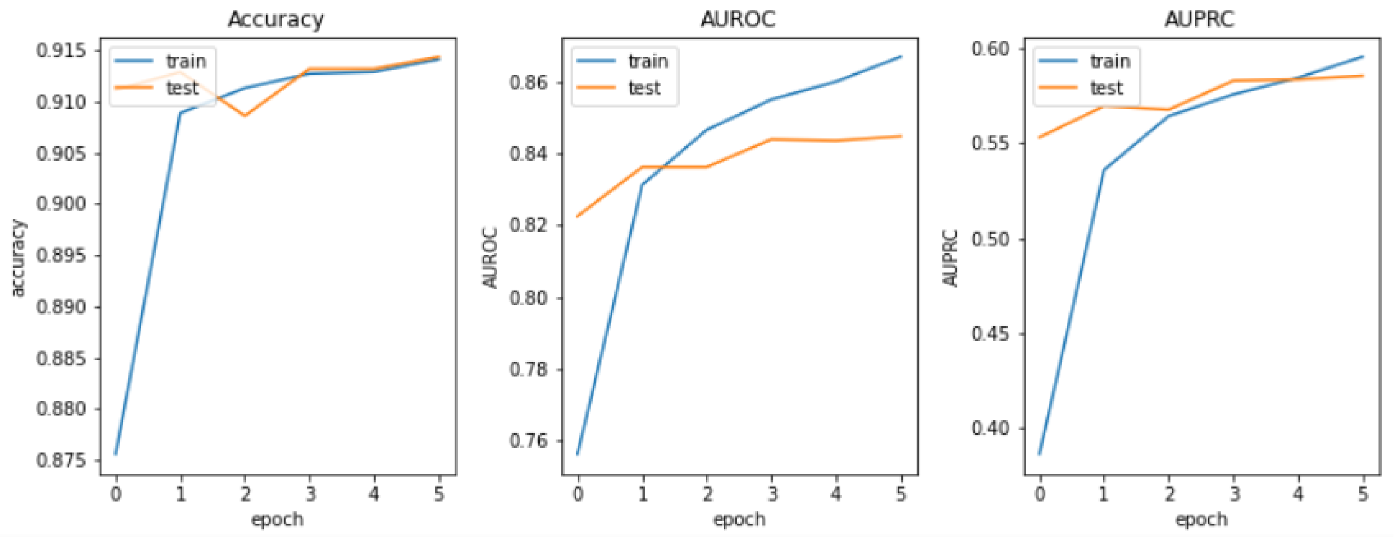
```
**MMNN - Enhancers** - with fixed FFNN and CNN - 10-holdouts estimate:
Accuracy: train - 0.9213261425495147 -- test - 0.9121987879276275
AUROC: train - 0.9278244793415069 -- test - 0.8479498088359833
AUPRC: train - 0.7284751772880554 -- test - 0.5957026779651642
```



7 - MMNN – Enhancers - Fixed - Training history

Graphs referring to a single holdout, graphs from other holdouts show a similar trend

```
MMNN - Enhancers - with Bayesian optimized hyperparameters FFNN and CNN -
10-holdouts estimate:
Accuracy: train - 0.9104132056236267  --  test - 0.9092122912406921
AUROC: train - 0.8816489458084107  --  test - 0.8556916177272796
AUPRC: train - 0.6161971867084504  --  test - 0.5873279452323914
```



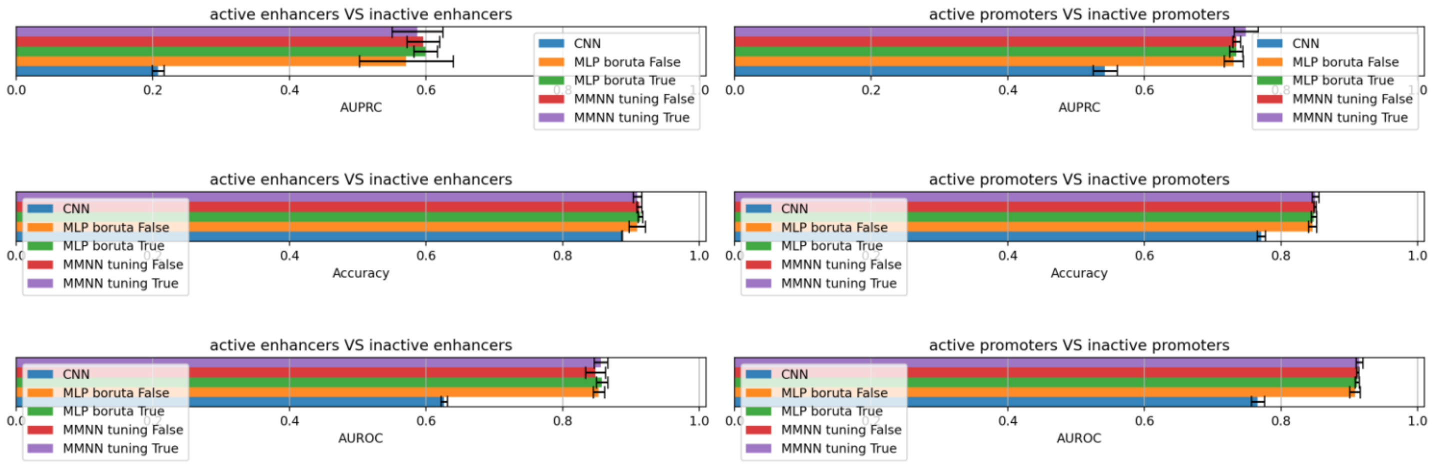
8- MMNN – Enhancers - Optimized - Training history

Graphs referring to a single holdout, graphs from other holdouts show a similar trend

The *MMNN* trained over epigenomic data and genomic sequence data achieves an accuracy of about 91.2%, an AUROC of about 0.85 and an AUPRC of about 0.60 over the testing set for promoters, and an accuracy of about 91%, an AUROC of about 0.86 and an AUPRC of about 0.59 over the testing set for enhancers.

These results are respectable, proving that combining the two modalities slightly improved the performance, especially for promoters, even if the effects were a bit underwhelming for enhancers. A possible cause can be deduced from the training history graphs that show clear signs of overfitting, suggesting that further regularization techniques might improve the current results.

The bar plots of all the architectures' performances are hereby presented:



9 - Test accuracy, AUROC, AUPRC of all architectures

References

L. Cappelletti, A. Petrini, J. Gliozzo, E. Casiraghi, M. Schubach, M. Kircher, and G. Valentini. Boosting tissue-specific prediction of active cis-regulatory regions through deep learning and bayesian optimization techniques. BMC Bioinformatics, 2022.

Yifeng Li, Wenqiang Shi, and Wyeth W. Wasserman. Genome-wide prediction of cis-regulatory regions using supervised deep learning methods. BMC Bioinformatics, 19(1):202, 5 2018.