# Bioinformatics Course Project Report

## Prediction of regulatory regions activity through deep learning methods

Gabriele Cerizza
gabriele.cerizza@studenti.unimi.it
https://github.com/gabrielecerizza/bioinformatics_project

## 1 Introduction

The regulation of gene expression is handled by non-coding regions that account for 98% of the DNA. Chief among non-coding regions are those denoted as "enhancers" and "promoters". Promoters provide transcription factors binding sites to which, among other proteins, bind the RNA polymerase, which is the main enzyme responsible for gene transcription. Enhancers aid gene transcription by interacting with promoters. Enhancers can interact with promoters across hundreds of thousands of nucleotides [14], by virtue of the peculiar three-dimensional folding structure of the DNA.

Data-driven prediction of regulatory regions activity is crucial to understand the impact of genetic variations on phenotype, since variations that fall within inactive regulatory regions are less likely to be pathogenic [3]. In this endeavor, an important role is played by deep learning methods, owing to their capability to model complex non-linear functions starting from large quantities of data.

The application of deep learning methods in the context of regulatory regions has been an active field of research in recent years. Feed-Forward Neural Networks (FFNNs) trained on epigenomic data have been employed to distinguish promoters and enhancers, as well as active and inactive regulatory regions [3,19]. Convolutional Neural Networks (CNNs) trained on DNA sequences were used to identify promoters and distal regulatory elements [3, 5, 14, 25, 27] and to predict their activity [3]. Sometimes, CNNs have been trained also on epigenomic data [5, 6].

Novel approaches combining data from different sources are being developed, exploiting several integration strategies. The joint consideration of heterogeneous information within the framework of Multi-Modal Neural Networks (MMNNs) shows improvements in the predictive power of deep learning models [7, 20]. In consideration of the foregoing, we hereby explore different deep learning architectures with two tasks in mind: (i) predicting active versus inac-

| Repetitions | Layer type | No. of units | Activation |
|---|---|---|---|
| 2 | Dense | $[32, 256]$ (step 32) | ReLU |
| $[0, 4]$ | Dense | $[16, 128]$ (step 16) | ReLU |
| 1 | Dense | 1 | Sigmoid |

**Table 1:** General architecture for the FFNN meta-model. Square brackets specify the minimum and maximum values for each interval of hyperparameter values. When not otherwise stated, step is assumed to be 1. For intervals concerning floating-point values, the full range of values is explored. Curly brackets specify sets of values, instead of intervals of values.

tive enhancers (AE vs IE); (ii) predicting active versus inactive promoters (AP vs IP).

In Section 2 we will provide an overview of the models evaluated on the aforementioned tasks, along with the corresponding explored hyperparameter spaces. In Section 3 we will discuss details pertaining to the execution of the experiments and to the data. Finally, in Section 4 we will present the results of the experiments and their implications.

# 2 Models

In this section we will discuss the proposed neural network architectures for the different tasks (Subsection 2.1), the means of hyperparameter optimization adopted (Subsection 2.2) and the comparison models (Subsection 2.3).

## 2.1 Proposed architectures

To tackle the problem of determining whether a regulatory region is active or inactive, we had at our disposal two kinds of data: epigenomic data and DNA sequence data (for details on the data, see Subsection 3.3). We opted for a FFNN when dealing with epigenomic data and for a CNN when dealing with sequence data. A MMNN was then employed to combine both architectures and work simultaneously on both kinds of data.

FFNNs are well-suited for epigenomic data, since the order of the features is inconsequential and this kind of neural network is permutation invariant. On the other hand, CNNs are better suited for data having a grid-like topology, with patterns embedded within sequences [11]. This is precisely the case with the one-hot encoded sequence data.

**FFNN.** We first explored a meta-model for the FFNN, which we named `ffnn_hp`. The architecture was designed as a chain of fully-connected layers having a Rectified Linear Unit (ReLU) activation function and a $L^2$ parameter regularization. The FFNN culminates in an output layer composed of a

| Hyperparameter | Values |
|---|---|
| $L^2$ reg. | $[0.0, 0.1]$ |
| Learning rate | $\{0.0001, 0.01\}$ |
| Optimizer | Nadam |
| Batch size | 256 |
| Max epochs | 1000 |
| Early stopping patience | 3 |

**Table 2:** Further hyperparameter spaces for the FFNN meta-model. The same notation as Table 1 is adopted.

| Repetitions | Layer type | No. of units | Activation |
|---|---|---|---|
| 2 | Dense | 224 | ReLU |
| 2 | Dense | 32 | ReLU |
| 1 | Dense | 1 | Sigmoid |

**Table 3:** General architecture for the FFNN model with manually chosen hyperparameters. The same notation as Table 1 is adopted.

single neuron having a sigmoid activation function. The architecture and the corresponding hyperparameter spaces are illustrated in Tables 1 and 2.

The layers are organized in two levels. The first level consists in two layers with a number of neurons ranging from 32 to 256, using a step of 32. The second level is optional and is composed of up to 4 layers with a number of neurons ranging from 16 to 128, using a step of 16. The range of neurons in the second level is smaller, following the classical tapering architecture that is commonly used in practice.

With regards to the optimization algorithm, we adopted Nadam [9], which is an adaptive learning rate algorithm incorporating Nesterov momentum. This allowed us to avoid setting up a decay schedule for the learning rate of the algorithm, thus freeing resources for hyperparameter tuning. The learning rate is chosen between 0.0001 and 0.01. Empirically, we found that a batch size of 256 provided a fast convergence without compromising the optimization of the metrics. The training of the model, scheduled for a maximum of 1000 epochs, was carried out with the early stopping algorithm, monitoring the loss function on the validation set. The early stopping algorithm was set to wait for 3 epochs before halting the training process. For the loss function, binary cross-entropy is suggested to prevent saturation of the sigmoid activation function [11].

In addition to the meta-model, we designed an architecture for the FFNN, named simply `ffnn`, with hyperparameters chosen through manual trial and error testing. We show in Tables 3 and 4 respectively the architecture and hyperparameter values for this model. With respect to the meta-model, this model presents a total of 4 hidden layers, with the first two having 224 neurons

| Hyperparameter | Values |
|---|---|
| $L^2$ reg. | 0.005 |
| Learning rate | 0.0001 |
| Optimizer | Nadam |
| Batch size | 256 |
| Max epochs | 1000 |
| Early stopping patience | 3 |

**Table 4:** Further hyperparameters for the manually chosen FFNN. The same notation as Table 1 is adopted.

and the last two having 32 neurons. For the $L^2$ parameter regularization, we used a value of 0.005, while the learning rate for the Nadam optimizer was set to 0.0001.

**CNN.** Also for the CNN we considered a meta-model and a manually chosen model. We start with a description of the meta-model architecture, named `cnn_hp`. This architecture is organized as a chain of convolutional layers followed by a flattening layer and a stack of fully-connected layers interspersed with dropout layers, culminating in a single neuron with a sigmoid activation function. Both the convolutional and fully-connected layers use ReLU activation functions.

Th architecture for the part containing convolutional layers can be described as having two levels. In the first level we have two convolutional layers, each containing from 32 to 256 neurons (with step 32) and a kernel size ranging from 5 to 10. Each convolutional layer is followed by a batch normalization layer. In the second level we have up to 6 convolutional layers with batch normalization, but the number of neurons now ranges from 16 to 128 (step 16), while the kernel size spans from 2 to 5. After the first level and for every two consecutive convolutional layers, we have a max pooling layer with a pool size of 2. The architecture is surmised by Tables 5 and 6.

Although deep learning theory purports that batch normalization allows omission of dropout layers for regularization purposes [11], we decided not to restrict the possibility of cumulating the two effects during hyperparameter tuning. This is because (i) models trained on sequence data were prone to overfitting, as shown in Section 4; and (ii) the optimal model may very well defy sensible but not universal notions about architecture design.

For the kernel of the first convolutional layers we selected a size between 5 and 10, which reflects the plausible length of a nucleotide sequence motif [3]. Beyond those layers, however, the sequence assumes a latent, internal representation that, having lost its characterization as a sequence of nucleotides, does not warrant further consideration of the length of the motifs. As such, we adopted more refined kernels with sizes between 2 and 5 for the subsequent convolutional layers.

| Repetitions | Layer type | No. of units | Notes |
|---|---|---|---|
| 2 | Convolutional | $[32, 256]$ (step 32) | kernel $[5, 10]$ |
| | BatchNormalization | - | - |
| | ReLU | - | - |
| 1 | MaxPool | - | size 2 |
| $[0, 6]$ | Convolutional | $[16, 128]$ (step 16) | kernel $[2, 5]$ |
| | BatchNormalization | - | - |
| | ReLU | - | - |
| | MaxPool (every 2 repetitions) | - | size 2 |
| 1 | Flatten | - | - |
| 1 | Dense | 512 | ReLU |
| 1 | Dropout | - | rate $[0.0, 0.5]$ |
| 1 | Dense | 256 | ReLU |
| 1 | Dropout | - | rate $[0.0, 0.5]$ |
| 1 | Dense | 128 | ReLU |
| 1 | Dense | 1 | Sigmoid |

**Table 5:** General architecture for the CNN meta-model. The same notation as Table 1 is adopted.

| Hyperparameter | Values |
|---|---|
| Learning rate | $\{0.0001, 0.01\}$ |
| Optimizer | Nadam |
| Batch size | 256 |
| Max epochs | 1000 |
| Early stopping patience | 3 |

**Table 6:** Further hyperparameter spaces for the CNN meta-model. The same notation as Table 1 is adopted.

| Repetitions | Layer type | No. of units | Notes |
|---|---|---|---|
| 2 | Convolutional | 128 | kernel 10 |
| | BatchNormalization | - | - |
| | ReLU | - | - |
| 1 | MaxPool | - | size 2 |
| 2 | Convolutional | 64 | kernel 8 |
| | BatchNormalization | - | - |
| | ReLU | - | - |
| | MaxPool (after 2 repetitions) | - | size 2 |
| 1 | Flatten | - | - |
| 1 | Dense | 128 | ReLU |
| 1 | Dropout | - | rate 0.6 |
| 1 | Dense | 64 | ReLU |
| 1 | Dense | 1 | Sigmoid |

**Table 7:** General architecture for the manually chosen CNN model. The same notation as Table 1 is adopted.

| Hyperparameter | Values |
|---|---|
| $L^2$ reg. | 0.1 |
| Learning rate | 0.0001 |
| Optimizer | Nadam |
| Batch size | 256 |
| Max epochs | 1000 |
| Early stopping patience | 3 |

**Table 8:** Further hyperparameter values for the manually chosen CNN model. The same notation as Table 1 is adopted.

Again, the Nadam algorithm was used for optimization. The same hyperparameter setting as that described for the FFNN meta-model was also employed regarding learning rate, batch size, maximum number of epochs, early stopping and loss function.

Concerning the manually chosen CNN model, named simply `cnn`, we show in Tables 7 and 8 its relevant features. With respect to the CNN meta-model, this model has a total of 4 convolutional layers, with 128 units for the first two and 64 units for the last two. Kernel sizes are respectively 10 and 8 for the two levels. Again, a max pooling layer with pool size 2 was added after any two consecutive convolutional layers.

Most notably, this model presents only two fully-connected layers, respectively having 128 and 64 neurons, and only one dropout layer. Due to the reduced number of dropout layers, we added $L^2$ parameter regularization to the fully-connected layers, which was not present in the meta-model. Moreover, we

employed a high dropout rate (0.6), as a means to counterbalance the already mentioned overfitting of the models.

While searching for possible kernel sizes, we also explored high values in the range of 50 and 100 for the first convolutional layers. While these values showed some promise, the higher computational burden and severe GPU over-heating ultimately lead us to the aforementioned lower values.

**MMNN.** For the MMNN we applied an intermediate integration strategy [20], in which the last hidden layers of the FFNN and the CNN are joined in a concatenation layer. This new layer is followed by a small stack of fully-connected layers with ReLU activation functions, terminating in a final neuron with a sigmoid activation function. An alternative intermediate integration strategy, having also regularization properties, is illustrated in [7].

We show in Figure 1 the layout of the MMNN, noting that no automated hyperparameter tuning has been performed on this neural network. No additional regularization methods were used in the final part of the MMNN.

The remaining hyperparameters, involving the optimization algorithm and the training of the neural network, were set to the same values already described for the FFNN and CNN. In particular, we used a learning rate of 0.001, which was the default in the TensorFlow implementation of Nadam.

## 2.2 Hyperparameter optimization

Deep learning algorithms possess a large number of hyperparameters. Manually searching for the optimal combination of hyperparameter values that minimizes the generalization error can be a daunting task. For this reason, well-known model selection algorithms such as grid search or random search are often employed to automatically explore hyperparameter spaces.

To achieve good results while keeping an acceptable computational cost, new hyperparameter optimization algorithms have been recently developed. We opted to use Hyperband (2018), which treats hyperparameter optimization as an infinite-armed bandit problem [18]: having a fixed amount of resources (e.g., time) and a large number of hyperparameter configurations, we have to decide whether to allocate a small quantity of resources to train each of a large number of configurations or to allocate a large quantity of resources to train models longer, but considering fewer configurations.

Hyperband builds upon the Successive Halving algorithm idea of iteratively allocating a fixed amount of resources to a set of configurations, evaluating the configurations and then throwing out the worst half until a single configuration remains. In practice, Hyperband works by organizing subsequent brackets. In the first bracket a large number of configurations is explored, allocating a small amount of resources to each configuration (i.e., a small number of epochs for training). The best configurations are kept through the next bracket and allowed an increased amount of resources. In the last bracket, each configuration is allocated the maximum amount of resources and the best configuration is then proposed as the solution.

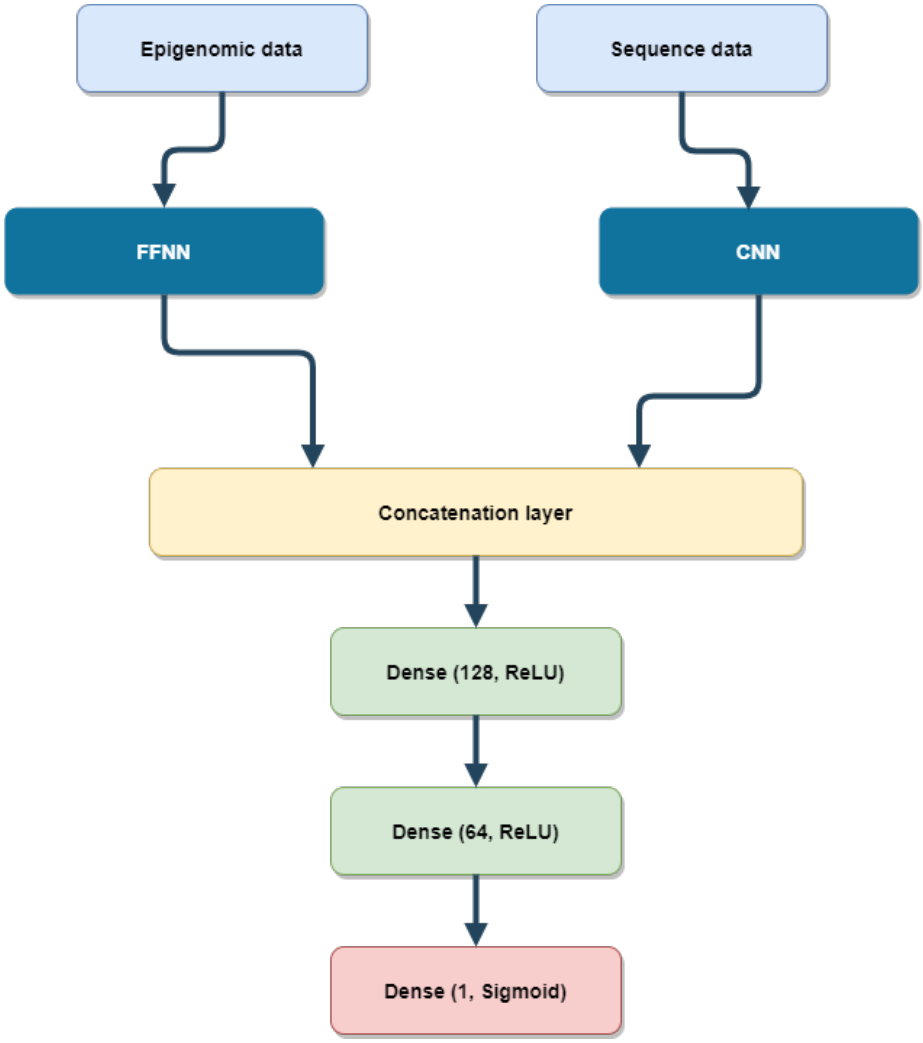


**Figure 1:** MMNN architecture representation.

According to the researchers who developed Hyperband, this novel algorithm is more robust and up to 30 times faster than Bayesian optimization (BO), though possibly achieving slightly inferior solutions than some BO variants.

Ultimately, apart from speed, what motivated our decision to adopt Hyperband was the fact that this algorithm natively supports conditional hyperparameters in the search space. Although studies are being conducted in this direction also for BO [24], popular libraries have yet to include BO support for conditional hyperparameters, whereas this feature is already integrated in the Hyperband implementation for Keras and TensorFlow [15].

The ability to exploit conditional hyperparameters is crucial to save time when, for instance, the second level of the FFNN or CNN architecture does not have any layer in a given configuration.

## 2.3 Comparison models

The models briefly listed below have been used as comparison models to gauge the performances of the FFNN, CNN and MMNN models proposed in this report. Some of these comparison models, in the papers that introduced them, were intended to be used with input of different shape, or with different data, or for different tasks than those considered here. As such, these comparison models should be regarded more as reference architectures rather than actual published models. This should be kept in mind when comparing and contrasting the results obtained from these models.

When available, these comparison models were adapted from the code sources provided by the respective Authors, using the hyperparameter values either mentioned in the papers or set in the corresponding implementation.

**DeepEnhancer.** This is a CNN described in [25] with the purpose of distinguishing enhancers from background genomic sequences, using only one-hot encoded sequence data. The best variation of this CNN, labeled "4conv2pool4norm" by the Authors, consists of 4 convolutional layers followed by batch normalization, with a max pooling layer every two consecutive convolutional layers. The network ends with two fully-connected layers interleaved with a dropout layer and a single neuron with a sigmoid activation function on top.

**Bayesian-FFNN and Bayesian-CNN.** These networks were proposed in [3] for the same tasks we are considering in this report, although human genome "HG19" instead of "HG38" was used. Nonetheless, the FFNN was trained on epigenomic data, while the CNN was trained on sequence data.

The Bayesian-FFNN consists of 3 fully-connected layers, with a ReLU activation function, stacked on top of each other and terminating in a single neuron with a sigmoid activation function. Additionally, $L^2$ parameter regularization is enforced for each hidden layer. To minimize the loss function, the Authors employed Stochastic Gradient Descent (SGD) with a learning rate decay schedule.

The Bayesian-CNN consists of 4 convolutional layers followed by batch normalization, with a max pooling layer after the third and fourth layers. Then 2 fully-connected layers followed by dropout layers are added, with a final neuron having a sigmoid activation function. Here the Nadam algorithm is used for optimization.

**DeepCAPE.** DeepCAPE is a complex network described in [5] with the purpose of predicting enhancers. The network incorporates two different CNNs trained on sequence data and on DNase-seq data. Owing to the small number of replicates in DNase-seq data and differences between cell lines, an autoencoder antecedes the CNN dealing with this kind of data. Both CNNs are then joined in a concatenation layer, culminating in a stack of 3 fully-connected layers with a dropout layer. The Authors mentioned using the Adam [16] optimizer.

The peculiarity of this network is represented by its intensive use of shortcut (or skip) connections reminiscent of those employed in the renowned ResNet neural network for image classification [13]. With these connections, many of the intermediate layers from both CNNs are directly tied to the concatenation layer. Such connections, among other properties, counteract the vanishing gradient problem, allowing for deeper networks that can be trained faster.

For the sake of making a fair comparison with other architectures without feeding different kinds of data to the models, in our experiments we used only the CNN trained on sequence data, while keeping the final fully-connected layers and the corresponding connections when possible.

**DECODE.** DECODE is a CNN illustrated in [6] aimed at predicting enhancers. However, this CNN is not trained on sequence data, but on epigenetic features extracted from STARR-seq, DNase-seq, ATAC-seq and ChIP-seq data.

The architecture displays 4 convolutional layers followed by a max pooling layer, then other 3 convolutional layers followed by another max pooling layer, and finally a fully-connected layer followed by a single neuron with sigmoid activation function. Optimization is carried out by the Adam algorithm with a learning rate of 5e-5.

Another characteristic of DECODE is that each convolutional layer is endowed with a sqeeze-and-excitation block, consisting of a global average pooling layer and two fully-connected layers acting as a gate which determines how many of the features are kept.

**Random Forest.** To gain insights into the performance of alternative machine learning algorithms, we also employed Random Forest [2] as a comparison model. The implementation of this algorithm was taken from `scikit-learn` (hereinafter also `sklearn`) [28]. Random Forest was trained only on epigenomic data.

# 3 Experimental setup

In this section we will provide the relevant specifics of the machine used in the experiments (Subsection 3.1), we will explain in greater detail what the carried out tasks entail (Subsection 3.2), the type of data and their sources (Subsection 3.3), the data pre-processing pipeline (Subsection 3.4), the ratios between samples/features and class labels (Subsection 3.5), what correlations and distributions exist within the data (Subsection 3.6), what feature selection algorithm was adopted (Subsection 3.7), how the data can be visualized (Subsection 3.8), how the holdouts have been planned (Subsection 3.9), what metrics have been used to evaluate the models (Subsection 3.10).

## 3.1 Machine specifics

The experiments were run on a machine with an Intel(R) Core(TM) i7-9700K 3.60GHz 8 core processor and 16 GB of RAM. To speed up the neural networks training, a NVIDIA(R) GeForce(R) RTX 2070 SUPER(TM) was employed.

The code was written in the Python language, using Python 3.9. To implement the neural networks, we used TensorFlow 2.5.0 [1].

## 3.2 Tasks description

Starting from the epigenomic and sequence data detailed in Subsection 3.3, we performed two tasks. The task referred to as "AE vs IE" pursued the objective of predicting whether a regulatory region marked as "enhancer" was associated or not with regulatory activity within a given cell line. In other terms, this was a supervised binary classification task aimed at distinguishing active enhancers from inactive enhancers in a given cell line.

The second task, "AP vs IP", was related to an identical supervised binary classification on the same cell line, but this time the objective was to infer from the data whether a regulatory region marked as "promoter" was active or inactive.

Ultimately, the goal of these tasks was to provide a Machine Learning (ML) method that would be able to reliably classify as active or inactive those regulatory regions for which, currently, we do not have any label.

## 3.3 Data types and sources

The data sources and types reflect those used in [3, 19]. The epigenomic data were downloaded from the ENCODE Consortium [8,10], with genomic assembly "HG38", cell line "HepG2" and a window size of 256. These data were extracted from histone modification and TF binding ChIP-seq, DNase-seq, ATAC-seq and WGBS-seq assays marked with the "released" status on the ENCODE portal.

The window size refers to the number of base pairs considered for each regulatory region. For enhancers, the window is centered on the activation peak for the given region. For promoters, we picked the window that starts from the

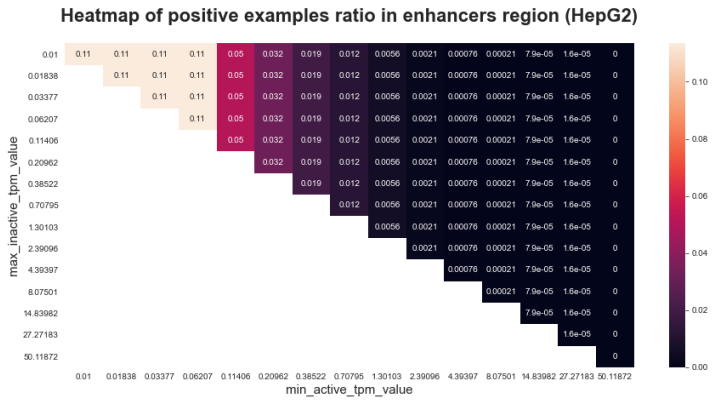


**Figure 2:** Heatmap showing how TPM threshold values affect the positive examples ratio in the enhancers data set for cell line "HepG2".

peak and then covers the sequence of nucleotides in the opposite direction of the gene, which may be on the left or on the right of the peak depending on the DNA strand we are considering.

The genomic sequence data were downloaded from the UCSC Genome Browser database [26], considering again genomic assembly "HG38". The data were then one-hot encoded.

The ground truth labels identifying each region as an active or inactive enhancer or promoter were retrieved from the FANTOM Consortium [22, 23], considering genomic assembly "HG38".

In particular, the FANTOM5 project used cap analysis of gene expression (CAGE) to identify transcriptionally active regions. From CAGE, FANTOM researchers obtained real values measured in Transcripts Per Kilobase Million (TPM). Two thresholds are then set: the minimum TPM value to consider a region active and the maximum TPM value to consider a region inactive. Obviously, the minimum TPM value for active regions should be greater than the maximum TPM value for inactive regions. Values between these two thresholds are deemed as "gray zone values" and consequently dropped.

Determining the optimal threshold values is an active field of research. In Figures 2 and 3 we produce heatmaps showing how different threshold values impact the ratio between positive and negative examples in the enhancers and promoters data sets for cell line "HepG2". The threshold values were sampled from a log scale evenly spaced interval of 15 values, with a minimum value of $10^{-2}$ and a maximum value of $10^{1.7}$.

In Figures 4 and 5 we can glimpse at the behavior of the positive examples ratio across all the available cell lines, depending on the minimum TPM value to consider a region as active. Clearly, for all the cell lines the ratio decreases monotonically as the threshold increases. Since the function is monotonic, no hints are given as to what the optimal threshold value could be. In other contexts,

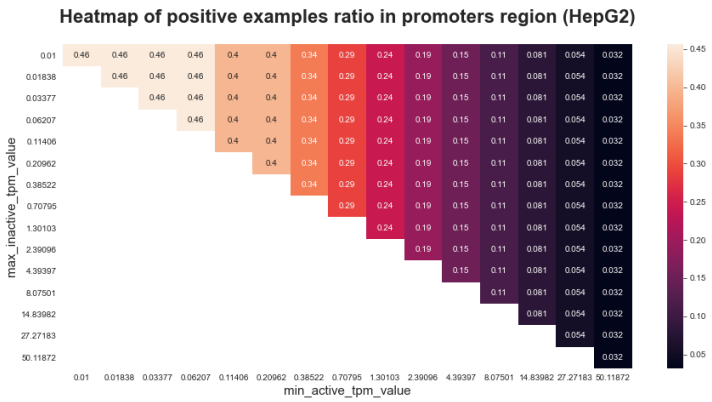


**Figure 3:** Heatmap showing how TPM threshold values affect the positive examples ratio in the promoters data set for cell line "HepG2".

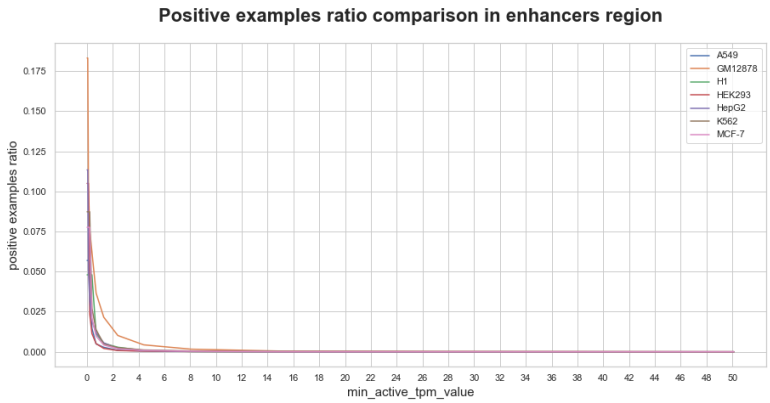


**Figure 4:** Line plot showing how the minimum TPM value to consider an enhancer region as active affects the positive examples ratio across all the available cell lines.

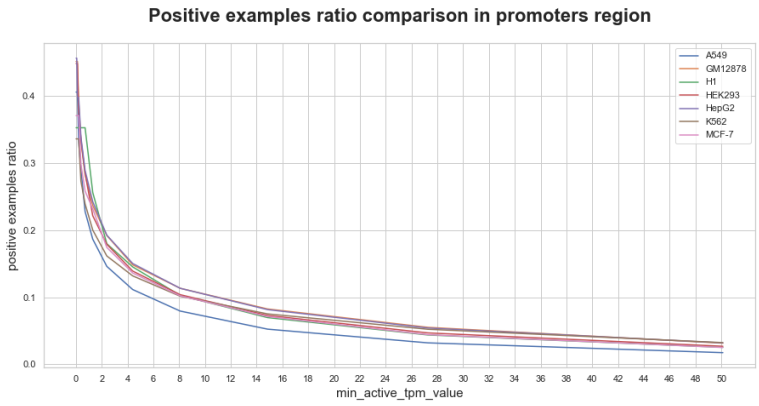


**Figure 5:** Line plot showing how the minimum TPM value to consider a promoter region as active affects the positive examples ratio across all the available cell lines.

such as metrics used to evaluate unsupervised clustering based on monotonic functions, like the RMSSTD, R-Squared and Modified Huber Γ metrics, it is considered optimal the point where the bend of the curve is most steep, which is referred to as "knee" or "elbow" in literature [12, 21]. Here we note that such elbow appears to be located at around 0.5 and 1 for enhancers and around 5 and 7 for promoters.

In [19], the Authors, using genomic assembly "HG19", chose minimum active values of 0 and 5, respectively for enhancers and promoters; and considered as inactive enhancers and promoters having a TPM equal to 0. FANTOM researchers suggest using an active threshold of 1 TPM for both enhancers and promoters, dropping values between 0 and 1. We adopted this last solution for our experiments.

Useful tools to automate the process of retrieving the data, using the Python language, are provided by the following `pip` packages: `epigenomic_dataset`[1] and `ucsc_genomes_downloader`[2].

## 3.4 Data pre-processing

The epigenomic data for the enhancers contained 19 NaN values out of 35566170 total values, while the epigenomic data for the promoters contained 359 NaN values out of 56133122 total values. In both cases, the WGSB-seq assay accounted for most of the NaN values.

Given the limited amount of NaN values, imputation of the missing values in the epigenomic data was feasible. To accomplish this task, we opted for the `KNNImputer` class made available by `sklearn`. It is worth to note that

[1] `https://github.com/AnacletoLAB/epigenomic_dataset`
[2] `https://github.com/LucaCappelletti94/ucsc_genomes_downloader`

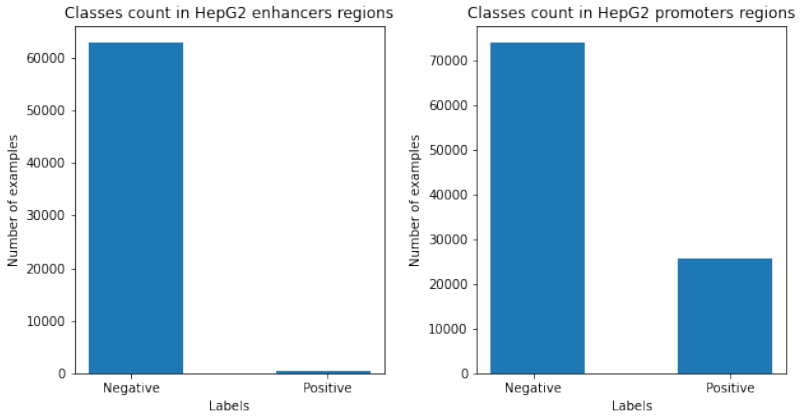


**Figure 6:** Class imbalance in the enhancers and promoters data sets.

the `KNNImputer` was fitted using only the training set to avoid any possible data leakage. The fitted imputer was then applied to transform the training, validation and test sets.

Upon the epigenomic data we also performed normalization by means of the `RobustScaler` class of `sklearn`. In this way we standardized the dataset by centering the values to the median and then scaling according to the interquartile range, so that possible outliers were not taken into account during the process. Again, the scaler was fitted only on the training set.

As for the sequence data, only one nucleotide remained unidentified for enhancers, while 4 nucleotides remained unidentified for promoters. In this case, the solution was to use 0.25 as a filler value when one-hot encoding the unknown nucleotides.

## 3.5 Samples/features and class labels ratios

In this subsection we briefly comment on the ratios between features and samples and on the imbalance of the class labels.

**Samples/features ratio.** Considering the epigenomic data, the ratio between samples and features was $\approx 112.606$ for enhancers and $\approx 177.724$ for promoters. With data having a strongly imbalanced ratio between samples and features, it is possible to find a single feature that, purely by chance, is able to correctly discriminate the classes of the data. If that was the case, the predictions made by the models would not be reliable on different data sets. In this case it is safe to assume that the ratios are within a reasonable range.

**Class imbalance.** The classes of the data set, for both enhancers and promoters, are not balanced, as shown in Figure 6. More precisely, $\approx 0.992\%$ of the

enhancers are inactive, while $\approx 0.008\%$ are active; concerning the promoters, $\approx 0.742\%$ are inactive, while $\approx 0.258\%$ are active. Possible strategies to address the class imbalance issue include:

- over-sampling, which balances the classes by increasing the number of instances of the under-represented class (for example, using the SMOTE algorithm [4]);
- under-sampling, which balances the classes by decreasing the number of instances of the over-represented class;
- specific class weights for the neural networks, so that errors on the under-represented class have more impact on the loss function[3].

We put some of these strategies into practice during the experiments (see the Results *infra*, Section 4).

### 3.6 Data correlations and distributions

In this subsection we focus on what features are the least correlated with the output, what features are the most correlated with each other and what is the distribution of the most different features.

**Output correlation.** In the context of epigenomic data, we computed Pearson correlation coefficient, Spearman correlation coefficient and Maximal information coefficient to identify the features least correlated with the output. The features we found are listed below.

- Enhancers: MAFG, SNRNP70, EZH2, NBN, ZNF382, CEBPZ, ARNT, ATM.
- Promoters: ZNF207, ZNF737.

**Feature correlation.** In Figures 7 and 8 we show the pairwise scatter plots of the features with the highest correlation. Correlation is particularly notable for SOX13-SOX5 and SAP130-ARID4B in the enhancers data set; and for ZNF580-ERF, ZBTB25-ZNF639, KLF16-ZGPAT, KLF16-ERF, ZNF274-ZNF883, SAP130-ARID4B, ZNF407-ZNF883 and NFYB-NFYC in the promoters data set.

[3]To compute the weights, we used the formulas found at `https://www.tensorflow.org/tutorials/structured_data/imbalanced_data#class_weights`

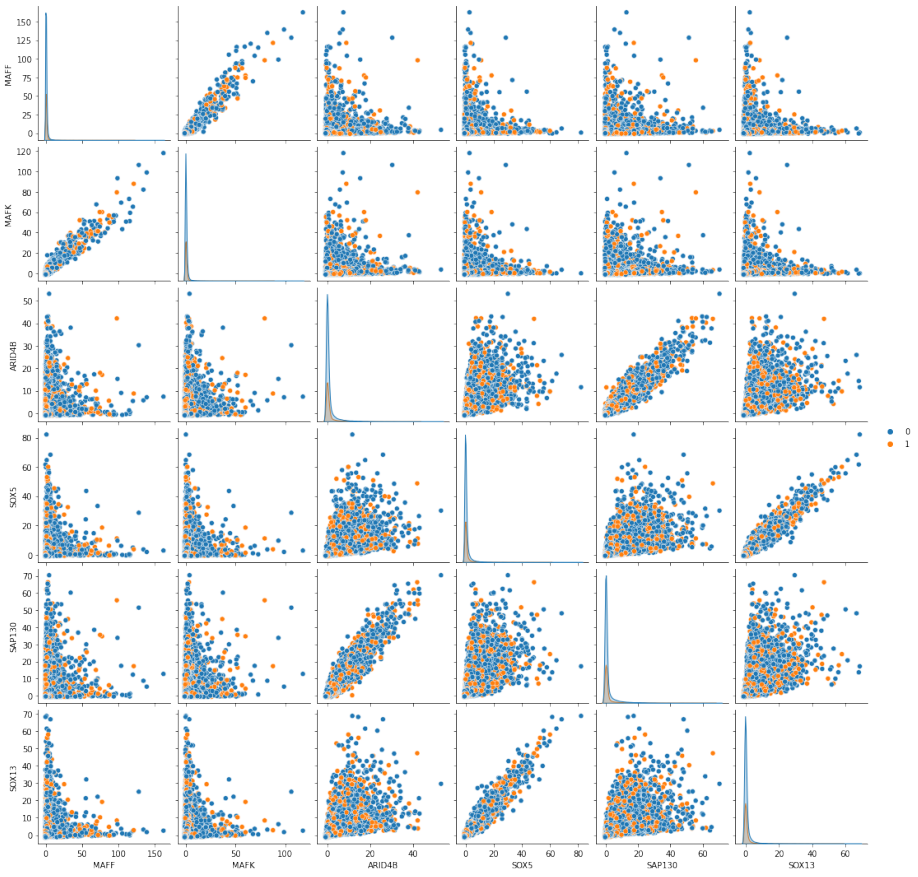


**Figure 7:** Pairwise scatter plot of the most correlated features in the enhancers data set.

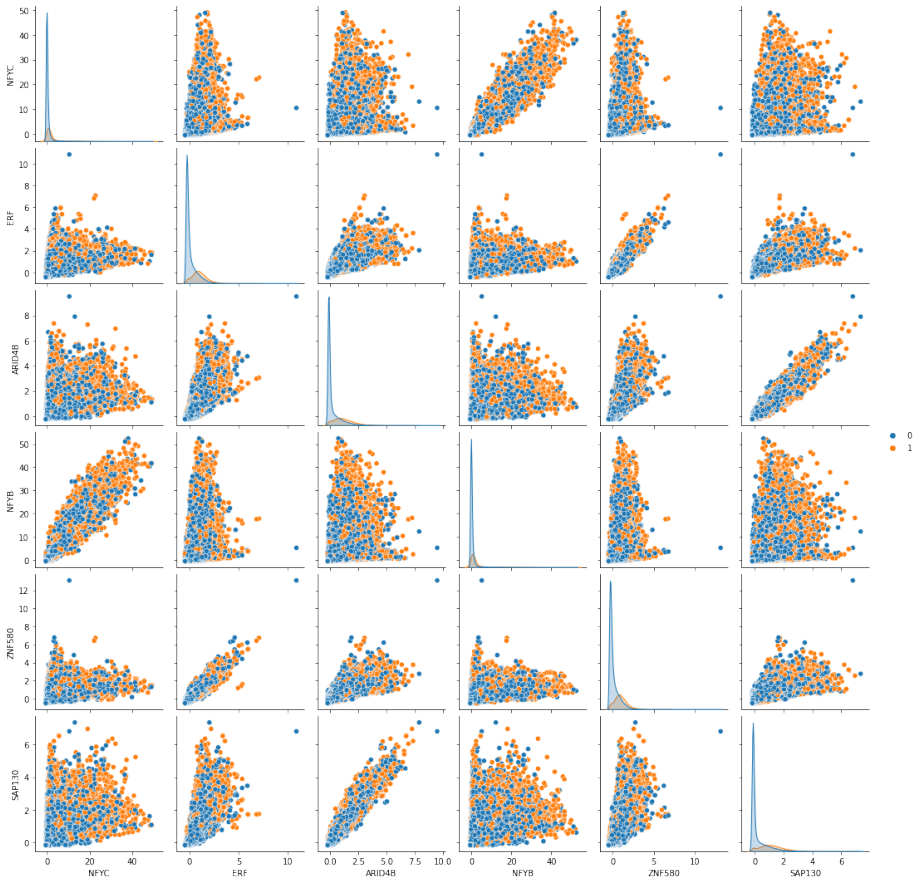


**Figure 8:** Pairwise scatter plot of the most correlated features in the promoters data set.

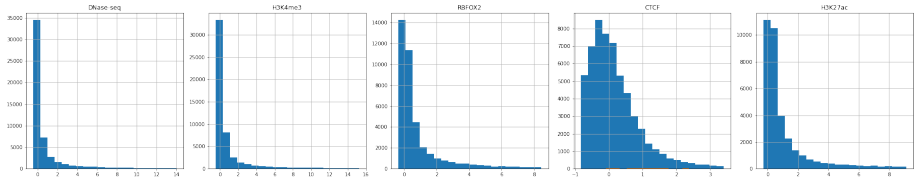


**Figure 9:** Distributions of the most different features in the enhancers data set.

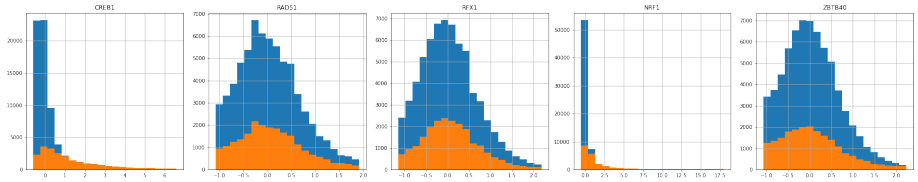


**Figure 10:** Distributions of the most different features in the promoters data set.

**Feature distribution.** In Figures 9 and 10 we show the distributions of the five most different features, where the difference was measured by the Euclidean distance between each feature vector.

From the figures we can argue that the task of classifying active and inactive enhancers/promoters cannot be tackled with a simple linear classifier, seeing that, especially for promoters, both labels assume a similar distribution, thus making separation more difficult. Generally the distributions of the two labels are both right-skewed and either Gaussian or exponential. In the case of the promoters, we also have a spike located mainly in the same position for the two labels. In the enhancers data set, the small number of positive examples makes it difficult to attain useful insights on the distributions of the labels.

## 3.7 Feature selection

The epigenomic data are composed of 562 features. To reduce the number of features, keeping only the most relevant ones and removing any possible noise, we performed feature selection by means of the Boruta algorithm [17]. We availed ourselves of the Python implementation offered by the `BorutaPy` package[4].

Boruta wraps a Random Forest classifier, which provides a ranking of the importance of the features with respect to the loss of the accuracy of classification. Boruta then adds to the data set "shadow" features obtained by shuffling the values of the original features. The importance of these shadow features is used to determine which features are truly relevant.

Concerning Boruta's hyperparameters, we built a Random Forest classifier from `sklearn` with a maximum depth of 5 and a balanced subsample class weight. Boruta was run for 100 iterations with the alpha hyperparameter set to 0.05 and the number of estimators set to "auto".

Like we did for imputation and normalization, Boruta was fitted only on the training set.

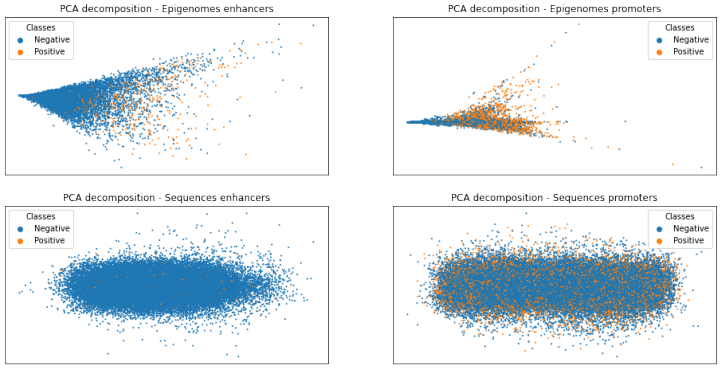


**Figure 11:** Data visualization using PCA for dimensionality reduction.

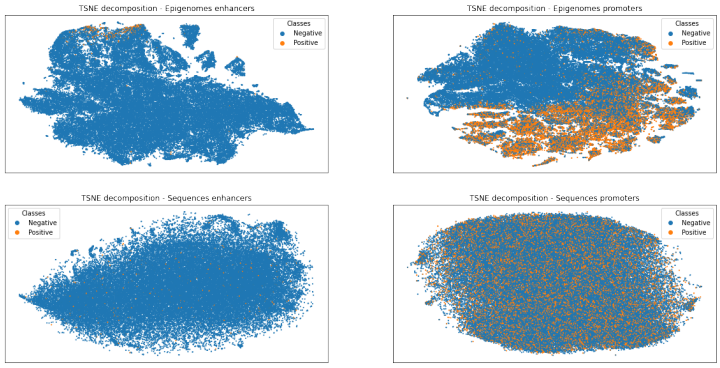


**Figure 12:** Data visualization using t-SNE for dimensionality reduction, with perplexity set to 50.

### 3.8 Data visualization

Figures 11 and 12 respectively visualize data by applying two well-known dimensionality reduction algorithms: Principal Component Analysis (PCA), in the implementation provided by `sklearn`; and t-distributed Stochastic Neighbor Embedding (t-SNE), in the implementation provided by the `MulticoreTSNE` package[5]. Before applying the algorithms, we performed imputation and normalization on the whole data sets. A form of data scaling is especially useful to obtain a meaningful representation from PCA.

Both the PCA and t-SNE plots support the conclusion that, while the epigenomic data sets offer some degree of separation between the two classes, the sequence data sets make hardly any distinction possible. This consolidates the claim that classification based on sequence data is a more difficult task. However, it should be kept in mind that this conclusion is drawn from two-dimensional data, which may or may not serve as a reliable indication of the separability of the two classes in the original high-dimensional space.

### 3.9 Holdouts

In order to evaluate the models we executed 10 holdouts, each of which, after shuffling the data, split the data sets in a training set containing 80% of the data and in a test set containing the remaining 20% of the data. In doing so, we preserved the stratification of the classes in both the training and test sets.

Within each holdout, we also extracted a validation set from the training set, following the same procedure described above. So we ended up with a training set containing 80% of the data of the original training set and a validation set containing the remaining 20% of the data of the original training set. The validation set was then employed during the model selection for the FFNN and CNN meta-models and during the training of the neural networks, playing an important role within the early stopping algorithm.

The importance of having a validation set could not be overstated. While having a validation set for model selection is required to avoid an overly optimistic evaluation of the chosen model, it should be stressed that a validation set should be employed also during the training of the chosen model, when the early stopping algorithm is in place. Indeed, early stopping is part of the learning machine and should not have access to the test set before the final evaluation, if we want to keep our models from becoming biased.

Inside the holdout loops, we performed imputation, normalization and feature selection on the epigenomic data. We reiterate that in all three occasions the algorithms were fitted using only the training sets.

For the FFNN and CNN proposed meta-models, a round of hyperparameter optimization by means of Hyperband was performed for each holdout, limiting the total number of epochs to 200 and the maximum number of epochs for each configuration to 15. We pursued maximization of the area under the precision

[4] `https://github.com/scikit-learn-contrib/boruta_py`
[5] `https://github.com/DmitryUlyanov/Multicore-TSNE`

recall curve (AUPRC) on the validation set to select the best configuration. To curb the computation time required by the hyperparameter optimization, the same optimized model obtained from the data resulting from feature selection was used with the data of the corresponding holdout for which no feature selection was performed. For each meta-model, in each holdout, the model selection phase took from 30 to 45 minutes.

Finally, it is worth mentioning that, to feed the batches to the networks and to lazily compute the one-hot encoding of the sequence data in a memory-efficient way, we relied upon the following packages: `keras_bed_sequence`[6] and `keras_mixed_sequence`[7].

### 3.10 Metrics

Each model was evaluated according to mainly three metrics: area under the precision recall curve (AUPRC), area under the receiver operating characteristic curve (AUROC) and accuracy. Given the imbalance of the classes, the mean AUPRC over the test sets was considered the most informative predictor of the quality of the model.

To gauge the existence of a statistical difference between models under comparison, we used the Wilcoxon signed-rank test. We considered a p-value lower than 0.01 as a sensible threshold to claim that the compared models were statistically different.

## 4 Results

**Naming conventions.** In the figures presented in this section, the evaluated models are referred to with the following naming conventions:

- FFNN: the manually chosen feed-forward neural network;
- FFNN HP: the feed-forward neural network resulting from model selection on the FFNN meta-model;
- Cnn1d: the manually chosen convolutional neural network;
- Cnn1d us: the manually chosen convolutional neural network, with random under-sampling applied to the training set;
- Cnn1d os: the manually chosen convolutional neural network, with SMOTE over-sampling applied to the training set;
- Cnn1d HP: the convolutional neural network resulting from model selection on the CNN meta-model;
- MMNN: the multi-modal neural network trained from scratch;

[6] `https://github.com/LucaCappelletti94/keras_bed_sequence`
[7] `https://github.com/LucaCappelletti94/keras_mixed_sequence`

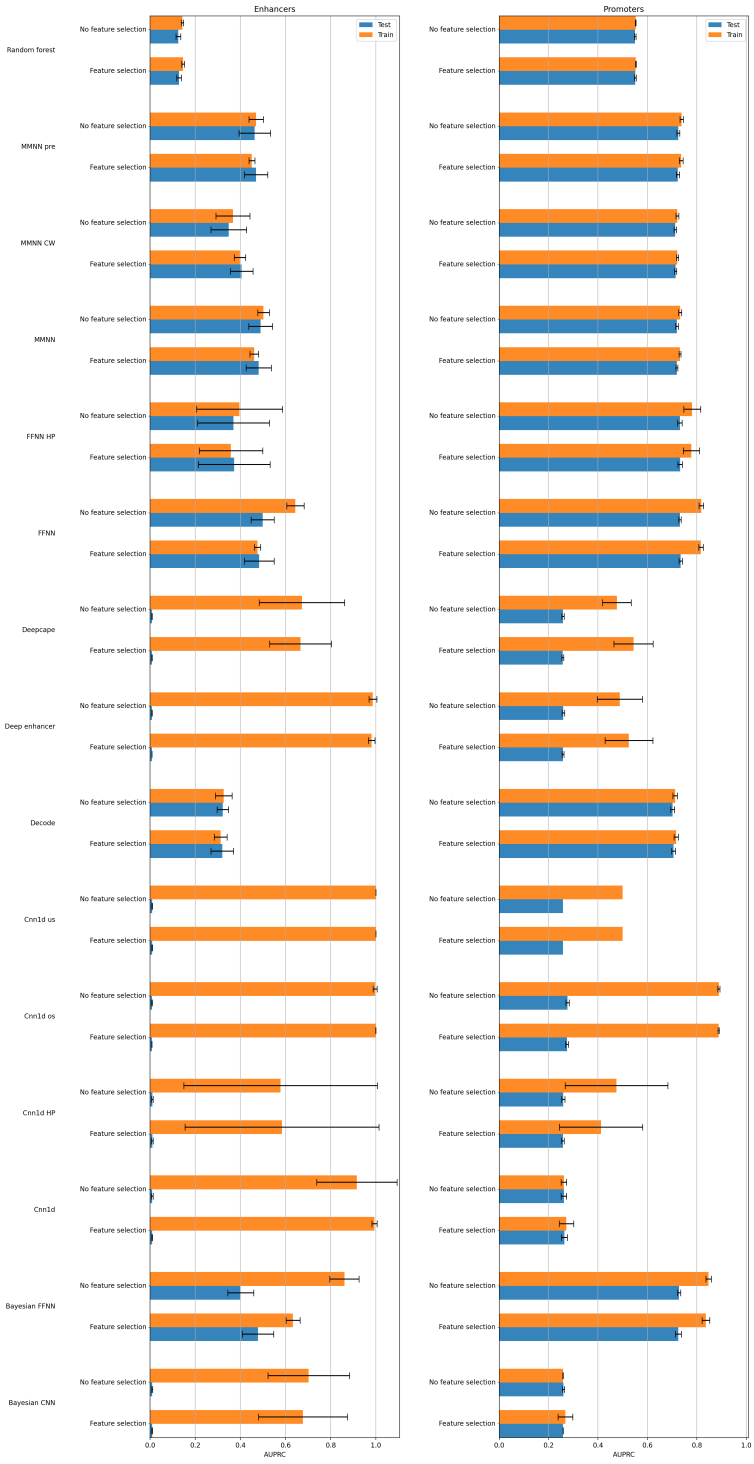


**Figure 13:** Results of the models with the AUPRC metric.

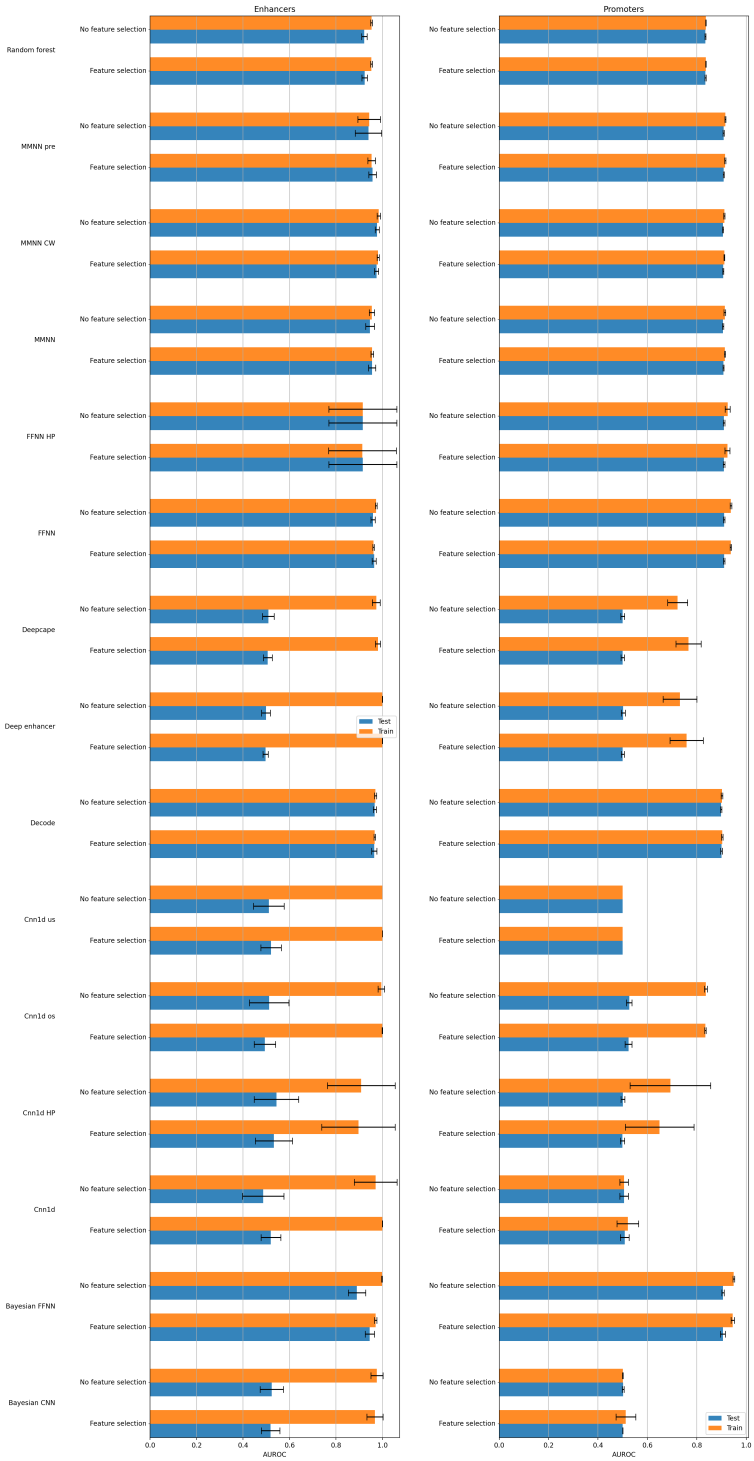


**Figure 14:** Results of the models with the AUROC metric.

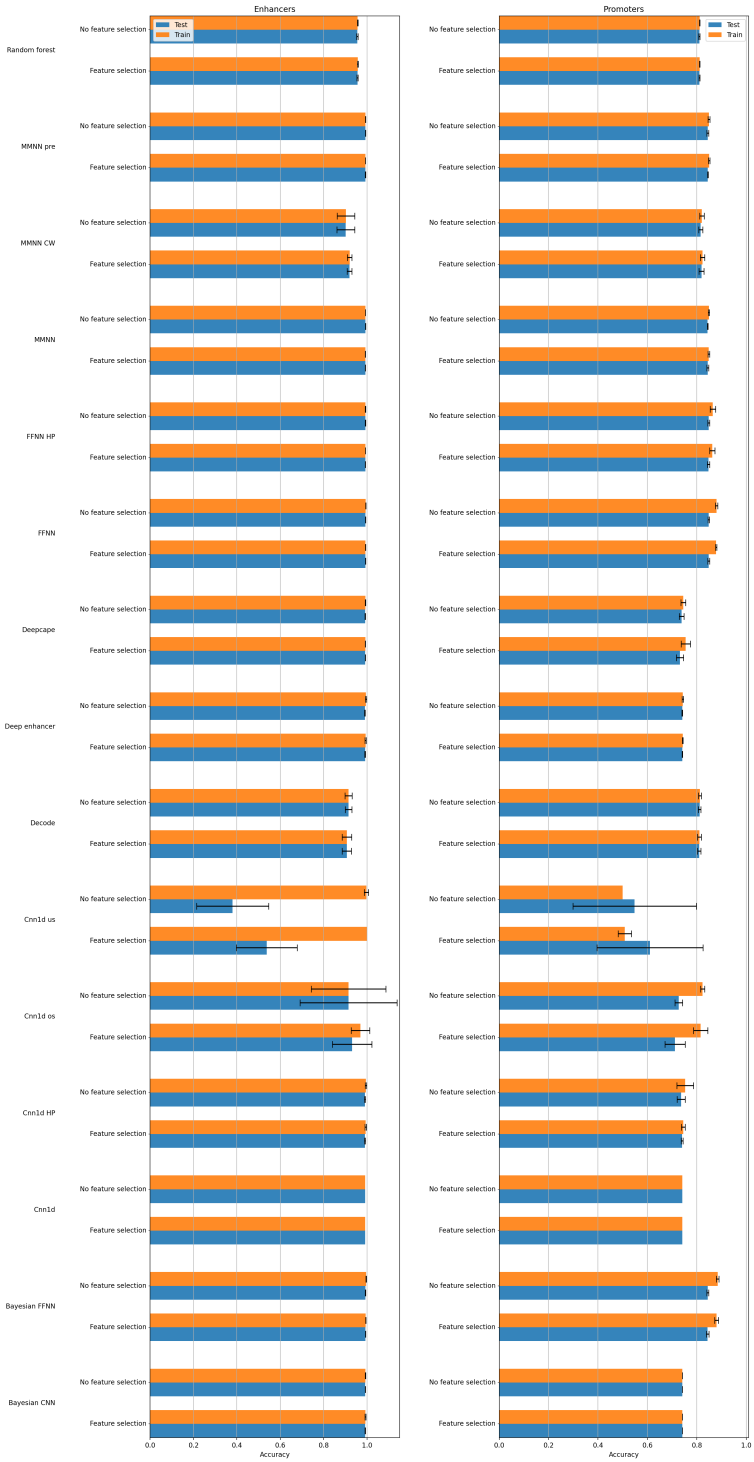


**Figure 15:** Results of the models with the accuracy metric.

| region | model_name | tp/t | | fp/t | | tn/t | | fn/t | |
|---|---|---|---|---|---|---|---|---|---|
| | | mean | std | mean | std | mean | std | mean | std |
| enhancers | bayesian_cnn | 0.000008 | 0.000025 | 0.000482 | 0.001524 | 0.991538 | 0.001524 | 0.007972 | 0.000025 |
| enhancers | cnn1d | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.992020 | 0.000000 | 0.007980 | 0.000000 |
| enhancers | cnn1d_hp | 0.000000 | 0.000000 | 0.001256 | 0.002040 | 0.990764 | 0.002040 | 0.007980 | 0.000000 |
| enhancers | cnn1d_os | 0.000395 | 0.000592 | 0.061263 | 0.091349 | 0.930758 | 0.091349 | 0.007585 | 0.000592 |
| enhancers | cnn1d_us | 0.003919 | 0.001050 | 0.458181 | 0.141164 | 0.533839 | 0.141164 | 0.004061 | 0.001050 |
| enhancers | deep_enhancer | 0.000000 | 0.000000 | 0.000980 | 0.001833 | 0.991041 | 0.001833 | 0.007980 | 0.000000 |
| enhancers | deepcape | 0.000000 | 0.000000 | 0.000166 | 0.000405 | 0.991854 | 0.000405 | 0.007980 | 0.000000 |
| promoters | bayesian_cnn | 0.000005 | 0.000016 | 0.000000 | 0.000000 | 0.741853 | 0.000000 | 0.258142 | 0.000016 |
| promoters | cnn1d | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.741853 | 0.000000 | 0.258147 | 0.000000 |
| promoters | cnn1d_hp | 0.000636 | 0.001517 | 0.002017 | 0.005029 | 0.739836 | 0.005029 | 0.257511 | 0.001517 |
| promoters | cnn1d_os | 0.019748 | 0.024212 | 0.049582 | 0.064890 | 0.692271 | 0.064890 | 0.238399 | 0.024212 |
| promoters | cnn1d_us | 0.069019 | 0.113473 | 0.200200 | 0.328045 | 0.541653 | 0.328045 | 0.189128 | 0.113473 |
| promoters | deep_enhancer | 0.000240 | 0.000377 | 0.000726 | 0.000935 | 0.741127 | 0.000935 | 0.257907 | 0.000377 |
| promoters | deepcape | 0.005241 | 0.007660 | 0.015398 | 0.021746 | 0.726455 | 0.021746 | 0.252906 | 0.007660 |

**Table 9:** True positive, false positive, true negative and false negative ratios of the test set predictions made by models trained only on sequence data with feature selection.

- MMNN pre: the multi-modal neural network trained using pre-trained FFNN and CNN (FFNN and CNN layers were not frozen);
- MMNN cw: the multi-modal neural network trained from scratch, but with class weights applied to the loss function, to give more emphasis to the minority class.

In Figures 13, 14 and 15 we show the results obtained respectively with the AUPRC, AUROC and accuracy metrics.

**Feature selection relevance.** Feature selection with Boruta marked a statistically significant difference in the model performances only with the Bayesian FFNN model, over the enhancers task (p-value 0.003 for AUPRC and AUROC). Therefore, we can claim that the feature selection relevance, using Boruta, was overall negligible. In the next paragraphs, for the sake of simplicity, we will consider only the results obtained with feature selection.

**Enhancers.** Distinguishing active and inactive enhancers was the most difficult task. Due to the class imbalance, we can consider the AUPRC scores as the most informative scores. It is evident from the figures that models trained on epigenomic data obtained the best results. In particular, FFNN, MMNN, MMNN pre and Bayesian FFNN obtained the best results, with a mean AUPRC around 0.48, and were statistically indistinguishable.

The models trained on sequence data learned next to nothing, as it is evident from the AUPRC scores. This claim is confirmed by the AUROC scores, which are invariably set around 0.5, signifying performances comparable with that of a random classifier. Indeed, these models, which were affected by overfitting despite the regularization tools employed, in most cases achieved no true positive prediction on the test set over the 10 holdouts, as shown in Table 9. Only the CNNs aided by over-sampling and under-sampling managed to recognize some

positive examples, but at the price of many positive misclassifications, evinced by the ratio of false positive examples.

**Promoters.** For the active promoters vs inactive promoters task, the best models were FFNN and FFNN HP, whose mean AUPRC reached a value around 0.73. These two models were statistically indistinguishable on mean AUPRC.

Again, models trained on epigenomic data performed better than models trained only on sequence data. Models trained on sequence data also showed poor learning, as seen from the true positive ratios in Table 9, albeit some models achieved slightly better results than those obtained in the enhancers task. The lack of learning is also apparent from the mean AUROC scores, again around 0.5. For the models trained on sequence data, the higher AUPRC scores registered in the promoters task, with respect to the enhancers task, were mostly due to the less severe imbalance between the classes.

**MMNN relevance.** From the results obtained on mean AUPRC, we can say that using MMNN did not improve performance with respect to the isolated FFNN models. However, MMNN achieved better results than isolated CNN models.

Using pre-trained FFNN and CNN in MMNN pre did not modify the overall AUPRC performance with respect to the basic MMNN (p-value 0.019 for enhancers and p-value 0.013 for promoters). Surprisingly, using class weights in MMNN cw worsened the overall AUPRC performance with respect to the basic MMNN, and this difference was statistically significant (p-value 0.0019 for enhancers and p-value 0.0039 for promoters).

**Hyperparameter optimization.** Results obtained by the Cnn1d HP meta-model suffered from high variability, as evidenced by the standard deviation in Figures 13 and 14. Indeed, the chosen hyperparameter values differed much between the holdouts, running the gamut of the hyperparameter spaces. Increasing the amount of resources allocated for model selection and reducing the dimension of the hyperparameter spaces might help in alleviating this problem.

The FFNN HP meta-model incurred less variability, but the results were still worse or on par with that of other models, such as the manually chosen FFNN model.

**Most relevant features.** During feature selection on the epigenomic data, Boruta returned for each holdout the most relevant features. Across all the holdouts, a total of 131 features, out of the original 562, were chosen at least once for the enhancers data set. The features selected most times are shown in Table 10. For the promoters dataset, a total of 508 feature, out of the original 562, were chosen at least once. The list of features for the promoters dataset is omitted for space sake.

We can conclude that, for the enhancers task, most epigenomic features are considered irrelevant by Boruta, with only a small set of features being

| | count |
|---|---|
| HOXA3 | 10 |
| MED1 | 10 |
| THRA | 10 |
| ZGPAT | 10 |
| PHF8 | 10 |
| ZNF574 | 10 |
| TSC22D2 | 10 |
| TFE3 | 10 |
| NONO | 10 |
| IKZF5 | 10 |
| SAP130 | 10 |
| KDM1A | 10 |
| POLR2G | 10 |
| TFDP2 | 10 |
| MYBL2 | 10 |
| MAX | 10 |
| TBP | 10 |
| KMT2A | 10 |
| TAF1 | 10 |
| ERF | 10 |
| YEATS4 | 10 |
| ARID4B | 10 |
| KLF6 | 10 |
| RBPJ | 10 |
| DRAP1 | 10 |
| HDAC1 | 10 |
| DNase-seq | 10 |
| ZSCAN9 | 10 |
| THAP11 | 10 |
| AFF4 | 10 |
| MXD4 | 10 |
| TFDP1 | 10 |
| GATA4 | 10 |
| SP5 | 10 |
| HDAC2 | 10 |
| H3K9ac | 10 |
| ZBED4 | 10 |
| GATAD1 | 10 |
| ARID4A | 10 |
| KAT7 | 10 |
| POLR2A | 10 |
| ZFX | 10 |
| RBFOX2 | 10 |
| ZNF580 | 10 |
| ZNF331 | 10 |
| IRF2 | 10 |
| H3K27ac | 10 |
| PATZ1 | 10 |
| ZBTB25 | 10 |
| ZBTB7B | 10 |
| SMAD4 | 10 |
| POLR2AphosphoS5 | 10 |
| ASH2L | 10 |
| REST | 10 |
| LCOR | 10 |
| KAT8 | 10 |
| KMT2B | 10 |
| DMAP1 | 10 |
| KLF16 | 10 |
| TFAP4 | 10 |
| RXRB | 10 |
| ELF3 | 9 |

**Table 10:** Number of times the features of the enhancers data set were chosen by Boruta over the 10 holdouts. Only the most selected features are shown.

consistently chosen by the feature selection algorithm. On the other hand, for the promoters task almost all the features are considered relevant at least once.

# References


[1] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Breiman, L. Random forests. *Mach. Learn. 45*, 1 (Oct. 2001), 5–32.

[3] Cappelletti, L., Petrini, A., Gliozzo, J., Casiraghi, E., Schubach, M., Kircher, M., and Valentini, G. Bayesian optimization improves tissue-specific prediction of active regulatory regions with deep neural networks. In *Bioinformatics and Biomedical Engineering - 8th International Work-Conference, IWBBIO 2020, Proceedings* (Jan. 2020), I. Rojas, O. Valenzuela, F. Rojas, L. Herrera, and F. Ortuño, Eds., Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Springer, pp. 600–612. 8th International Work-Conference on Bioinformatics and Biomedical Engineering, IWBBIO 2020 ; Conference date: 06-05-2020 Through 08-05-2020.

[4] Chawla, N. V., Bowyer, K. W., Hall, L. O., and Kegelmeyer, W. P. Smote: Synthetic minority over-sampling technique. *J. Artif. Int. Res. 16*, 1 (June 2002), 321–357.

[5] Chen, S., Gan, M., Lv, H., and Jiang, R. Deepcape: A deep convolutional neural network for the accurate prediction of enhancers. *Genomics, Proteomics & Bioinformatics* (2021).

[6] Chen, Z., Zhang, J., Liu, J., Dai, Y., Lee, D., Min, M. R., Xu, M., and Gerstein, M. Decode: A deep-learning framework for condensing enhancers and refining boundaries with large-scale functional assays. *bioRxiv* (2021).

[7] Choi, J.-H., and Lee, J.-S. Embracenet: A robust deep learning architecture for multimodal classification. *Information Fusion 51* (2019), 259–270.

[8] Davis, C. A., Hitz, B. C., Sloan, C. A., Chan, E. T., Davidson, J. M., Gabdank, I., Hilton, J. A., Jain, K., Baymuradov, U. K., Narayanan, A. K., Onate, K. C., Graham, K., Miyasato, S. R., Dreszer, T. R., Strattan, J., Jolanki, O., Tanaka, F. Y., and Cherry, J. The Encyclopedia of DNA elements (ENCODE): data portal update. *Nucleic Acids Research 46*, D1 (11 2017), D794–D801.

[9] Dozat, T. Incorporating nesterov momentum into adam. In *ICLR Workshop* (2016).

[10] Dunham, I., Kundaje, A., Aldred, S., Collins, P., Davis, C., Doyle, F., Epstein, C., Frietze, S., Harrow, J., Kaul, R., Khatun, J., Lajoie, B., Landt, S., Lee, B.-K., Pauli Behn, F., Rosenbloom, K., Sabo, P., Safi, A., Sanyal, A., and Birney, E. The encode project consortium: An integrated encyclopedia of dna elements in the human genome. 2012. nature 489: 57–74. *Nature 489* (09 2012), 57–74.

[11] Goodfellow, I., Bengio, Y., and Courville, A. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[12] Halkidi, M., Batistakis, Y., and Vazirgiannis, M. On Clustering Validation Techniques. *Journal of Intelligent Information Systems 17* (2001), 107–145.

[13] He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition, 2015.

[14] Kelley, D., Reshef, Y., Bileschi, M., Belanger, D., McLean, C., and Snoek, J. Sequential regulatory activity prediction across chromosomes with convolutional neural networks. *Genome Research 28* (03 2018), 739–750.

[15] Keras-Tuner. `https://keras-team.github.io/keras-tuner`.

[16] Kingma, D. P., and Ba, J. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings* (2015), Y. Bengio and Y. LeCun, Eds.

[17] Kursa, M. B., and Rudnicki, W. R. Feature selection with the boruta package. *Journal of Statistical Software, Articles 36*, 11 (2010), 1–13.

[18] Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., and Talwalkar, A. Hyperband: A novel bandit-based approach to hyperparameter optimization, 2018.

[19] Li, Y., Shi, W., and Wasserman, W. W. Genome-wide prediction of cis-regulatory regions using supervised deep learning methods. *BMC Bioinformatics 19*, 202 (2018).

[20] Li, Y., Wu, F.-X., and Ngom, A. A review on machine learning principles for multi-view biological data integration. *Briefings in Bioinformatics 19*, 2 (12 2016), 325–340.

[21] Liu, Y., Li, Z., Xiong, H., Gao, X., Wu, J., and Wu, S. Understanding and Enhancement of Internal Clustering Validation Measures. *IEEE Transactions on Cybernetics 43*, 3 (2013), 982–994.

[22] Lizio, M., Abugessaisa, I., Noguchi, S., Kondo, A., Hasegawa, A., Hon, C. C., de Hoon, M., Severin, J., Oki, S., Hayashizaki, Y., Carninci, P., Kasukawa, T., and Kawaji, H. Update of the FANTOM web resource: expansion to provide additional transcriptome atlases. *Nucleic Acids Research 47*, D1 (11 2018), D752–D758.

[23] Lizio, M., Harshbarger, J., Shimoji, H., Severin, J., Kasukawa, T., Sahin, S., Abugessaisa, I., Fukuda, S., Hori, F., Kato, S., Mungall, C., Arner, E., Baillie, K., Bertin, N., Bono, H., de Hoon, M., Diehl, A., Dimont, E., Freeman, T., and Kawaji, H. Gateways to the fantom5 promoter level mammalian expression atlas. *Genome Biology 16* (01 2015).

[24] Lévesque, J.-C., Durand, A., Gagné, C., and Sabourin, R. Bayesian optimization for conditional hyperparameter spaces. In *2017 International Joint Conference on Neural Networks (IJCNN)* (2017), pp. 286–293.

[25] Min, X., Chen, N., Chen, T., and Jiang, R. Deepenhancer: Predicting enhancers by convolutional neural networks. *BMC Bioinformatics 18*, 478 (2017).

[26] Navarro Gonzalez, J., Zweig, A. S., Speir, M. L., Schmelter, D., Rosenbloom, K., Raney, B. J., Powell, C. C., Nassar, L. R., Maulding, N., Lee, C. M., Lee, B. T., Hinrichs, A., Fyfe, A., Fernandes, J., Diekhans, M., Clawson, H., Casper, J., Benet-Pagès, A., Barber, G. P., Haussler, D., Kuhn, R., Haeussler, M., and Kent, W. The UCSC Genome Browser database: 2021 update. *Nucleic Acids Research 49*, D1 (11 2020), D1046–D1057.

[27] Onimaru, K., Nishimura, O., and Kuraku, S. Predicting gene regulatory regions with a convolutional neural network for processing double-strand genome sequence information. *PLOS ONE 15*, 7 (07 2020), 1–17.

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