Predicting Cis-Regulatory Regions through Convolutional Neural Networks

Giorgio Borghetto, Alessio Quercia

Abstract—In this paper we analyse the results obtained using different Convolutional Neural Network models to perform classification in order to predict Cis-Regulatory Elements, given DNA sequences. Several tests have been made using 16 models variations on 5 tasks belonging to 4 different cell lines. From the results of these tests we conclude that performing a multi-class classification is possible, but the accuracy is barely acceptable; whereas implementing a binary classification, predicting whether a DNA sequence is a Cis-Regulatory Element or another one leads to much better results.

1 Introduction

Convolutional Neural Networks (*CNNs*) are a useful machine learning approach to classification tasks. They are widely used in Computer Vision for image classification tasks, given their power to manage grid-like topologies [1]. However, since sequences and time-series can be thought of as 1D grids, CNNs can also be used in Bioinformatics field. Our work is an example of application of CNN models to this field, given that, starting from DNA sequences, we aim to predict *Cis-Regulatory Regions* (*CRRs*).

In the next section a background concerning Cis-Regulatory Regions together with a general description of the main operations that constitute a Convolutional Neural Network is provided.

Section 3 contains a detailed explanation of our work, specifying how the datasets are initially structured and how they are preprocessed in order to be given as input to our models in an appropriate form. Indeed, the data are split in tasks in order to apply binary classification on couples of Cis-Regulatory Elements. Moreover, this section includes the description of how a general CNN architecture is built, varying some parameters in order to obtain slightly different models, and how the learning of these models is accomplished.

Section 4 specifies how the simulations are run over all the tasks with different models, our systems architectures and the implementation details. Whereas in Section 5 the results of these simulations are analyzed.

Giorgio Borghetto, Alessio Quercia, Bioinformatics, A/A 2018-2019, Università degli Studi di Milano, via Celoria 18, Milano, Italy
E-mails: giorgio.borghetto@studenti.unimi.it, alessio.quercia@studenti.unimi.it

2 Background

In this work Convolutional Neural Networks have been used to predict Cis-Regulatory Regions (such as Enhancers and Promoters).

Cis-Regulatory Regions and Convolutional Neural Networks and their basic building blocks are briefly described in the following subsections.

2.1 Cis-Regulatory Regions

Cis-Regulatory Regions (CRRs) are DNA sequences that help specify the formation of diverse cell types and respond to changing physiological conditions. Their main role is to control the gene expression [2], [3]. Promoters and Enhancers are two types of Cis-Regulatory Elements: the former specify and enable the positioning of RNA polymerase process at transcription initiation sites [2], the latter modulate the transcriptions of genes on the same molecule of DNA and can be found near or relatively far away from the genes they regulate. Besides, enhancers are typically 50-1500 bp long and are located either upstream or downstream from the transcription start site of their target genes [4].

These DNA sequences are crucial in the understanding of gene regulatory mechanisms. Indeed, numerous studies have identified cis-regulatory mutations with functionally significant consequences for morphology, physiology and behaviour [5]. Thus, predicting them through machine learning approaches is currently a matter of high interest.

2.2 Convolutional Neural Networks

Convolutional Neural Networks are a category of Neural Networks such as image and video recognition, recommender systems, image classification, medical image analysis, and natural language processing.

There are four main operations in a CNN:

- *Convolution*: mathematical operation on two functions which produces a third function that expresses how the shape of one is modified by the other. It is a linear operation which computes element wise matrix multiplication and addition. In CNN terminology, the matrix used to perform the convolution is called a filter or kernel or feature detector and the matrix formed by sliding the filter over the image and computing the dot product is called the Convolved Feature or Activation Map or the Feature

Map. It is important to note that filters act as feature detectors from the original input image. Sometimes, it is convenient to pad the input matrix with zeros around the border, so that it is possible to apply the filter to bordering elements of our input image matrix.
- *Non Linearity*: since most of the real-world data we would want our CNN to learn would be non-linear, we account for non-linearity by introducing a non-linear activation function like ReLU.
- *Normalization*: operation that can be performed on the outputs of a convolutional layer in order to boost the learning and increase the network stability.
- *Pooling (or Sub Sampling)*: spatial pooling reduces the dimensionality of each feature map but retains the most important information. Spatial Pooling can be of different types: Max, Average, Sum, etc.
- *Classification (Fully Connected Neural Network)*: the Fully Connected Neural Network is a traditional Multi Layer Perceptron that uses an activation function as the Softmax (in the case we want the sum of the classes probabilities to be equal to 1) or the Sigmoid function (if we just need values between 0 and 1) in the output layer.
  The term Fully Connected implies that every neuron in the previous layer is connected to every neuron on the next layer. The output from the convolutional and pooling layers represents high-level features of the input image. The purpose of the Fully Connected Network is to use these features for classifying the input into various classes based on the training dataset.
- *Dropout*: trick that can be used to speed up the learning during the backpropagation, deciding whether or not to apply the weight update on each weight with a certain probability.

These are the basic operations of a Convolutional Neural Network. More details about the implementation and models are provided in the next section.

3 Theoretical Model

3.1 Datasets

The adopted datasets are in FASTA format, that is each example is represented in two-lines. The first line includes the number of the chromosome and the starting and ending chromosome locations of every 200 bp nucleotides sequence; while the second one includes the sequence itself.

There are four datasets, each one collecting data from a different cell line:

- GM12878, a lymphoblastoid cell line.
- HelaS3, an immortalized cell line that was derived from a cervical cancer patient.
- HepG2, a cell line derived from a male patient with liver carcinoma.
- K562, an immortalized cell line produced from a female patient with chronic myelogenous leukemia (CML).

Each dataset has a corresponding label dataset, which contains the labels for the nucleotides sequences. There are seven differnt labels:

- AE, Active Enhancer.
- IE, Inactive Enhancer.
- AP, Active Promoter.
- IP, Inactive Promoter.
- AX, Active Exon.
- IX, Inactive Exon.
- UK, Unknown.

Notice that the original datasets come as a multiclass classification problem, since there are 7 different types of labels. Thus, to perform binary classification on them, some kind of data preprocessing is required.

3.2 Data preprocessing

Each dataset has been preprocessed in order to extract the nucleotides sequences and the corresponding labels. An intermediate file has been created, containing couples of the form $(sequence, label)$, where both the sequence and the label are represented via One Hot Encoding (OHE). Each sequence has been treated like $200 \times 1 \times 4$ image [4], where 200 is the sequence length, 1 is the height and 4 are the different types of nucleotides (Adenine, Cytosine, Thymine, Guanine), which correspond to the OHE; while each label is a vector of seven elements, corresponding to the labels listed above, where only the sequence actual label is set to 1, and the others are set to 0.

For each cell line, the following five tasks have been created based on the identification of the different types of nucleotides sequences:

- AE vs AP.
- AE vs IE.
- AP vs IP.
- IE vs IP.
- AE $\cup$ AP vs REST (IE $\cup$ IP $\cup$ AX $\cup$ IX $\cup$ UK)

For each of the above tasks, starting from the intermediate file related to a specific cell line, a new file made up of couples of the form $(sequence, [0, 1])$ has been created, where the label is either 0, if the sequence refers to the first component of the chosen task, or 1, otherwise. This was done in order to perform binary classification on the given data, which originally was formatted as a multi-class dataset.

Once the preprocessed data is ready to use, it is shuffled and then split into training and test sets, respectively formed by 70% and 30% of the data.

3.3 CNN Architecture

The adopted model is a general Convolutional Neural Network, which can work with sequences. Indeed, the inputs are of the form $200 \times 1 \times 4$. Depending on the network architecture, the length can be reduced (e.g. using max pooling layers or even by not using the zero padding during the convolutions), while the depth doubles at each convolutional layer, given that the kernels number is doubling. The general CNN architecture follows the scheme listed below:

1) N convolutional layers (convolution and activation),
   1.1) Optional batch normalization layers.
   1.2) Optional max pooling layers.

2) M fully connected hidden layers.

2.1) Optional dropout layers.

3) 1 output (sigmoid activation).

3.3.1 Convolutional Layers

The first convolutional layer takes as input the preprocessed data described above and performs on them the convolution operation, which is:

$$S(i,j) = (K*I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n) \quad (1)$$

where I and K are respectively the two-dimensional input and kernel used by the convolution operation. In our case, the initial input has shape $200 \times 1 \times 4$, while the kernel has shape $8 \times 1 \times 4$. Moreover, the number of kernels is variable and doubles at each convolutional layer. This means that every convolutional layer produces an output with depth equal to the number of kernels applied. During the convolution operation, the kernel scans the input moving itself by a quantity described by the stride parameter, which was set to 1 by default, since we are analysing sequences. Given that the convolution cannot be applied along the borders of the input, either zero-padding is applied to let the kernels stride properly, by adding an appropriate number of zeroes at the end of the sequence, or the output of the convolution will have reduced length, resulting in a sequence length reduction by a quantity corresponding to the kernel length reduced by one. Once the output is computed, an activation function is applied over it. In our case, the adopted activation functions are the Rectified Linear Function (or ReLU), which introduces non linearity, and the Hyperbolic Tangent (or TanH).

Optionally, in the generalized model, it is possible to specify the application of a batch normalization layer, after each convolutional layer. The batch normalization is used to speed up the learning and increase the neural network stability, normalizing the output of the previous activation layer by subtracting the batch mean and dividing by the batch standard deviation:

$$\hat{x_i} = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \quad (2)$$

where x_i are the previous layer outputs, μ_B is the batch mean and $\sqrt{\sigma_B^2 + \epsilon}$ is the batch standard deviation. After the normalization, the normalized outputs $\hat{x_i}$ are scaled and shifted as follows:

$$y_i = \gamma \hat{x}_i + \beta \quad (3)$$

where γ and β are respectively the scaling and shifting parameters.

Furthermore, an optional max pooling layer can be applied in order to decrease the spatial dimensionality. In our case, a max pooling of $(2, 1)$ has been applied to halve the length.

An iteration over a variable number N is performed to decide how many convolutional layers (and optionally batch normalization and max pooling layers) form the Convolutional Neural Network architecture. By adding more layers, the depth doubles, the spatial dimensionality decreases and the computational complexity increases.

3.3.2 Fully Connected Layers

The output of the last convolutional layer is flattened in order to obtain the required 1D vector input for the subsequent Fully Connected Neural Network, which is composed by a variable number M of hidden layers, made up by an arbitrary number of units and by a single output unit. This Fully Connected Neural Network acts as a linear classifier, which computes the outputs as follows:

$$y = Wx + b \quad (4)$$

where W is the matrix of the model weights, x is the input vector and b is the bias vector. The sigmoid activation function is applied over the output y to squash it between 0 and 1, so that the outputs represent probabilities and binary cross entropy loss function can be applied to compute the error between the predictions and the actual labels:

$$\sigma(y) = \frac{1}{1 + e^{-y}}. \quad (5)$$

An optional dropout layer (in our case with probability 0.5) can be used after each hidden layer to reduce the model overfitting during the training phase.

3.4 Training, Validating and Testing

Given the complexity of the model and the low hardware specifications, each different model is trained and validated for 10 epochs, which are enough to produce good results. Indeed, the models often produce their best outputs after 4 or 5 epochs and then start overfitting, since there are few data for each task, often with unbalanced numbers of samples per class. At each epoch, the model is trained using Adam optimizer and a batch size of 32 on the training set. After each training step, the model is validated on the test set, carrying out the validation error and accuracy for that epoch. Once the model has been trained, it is evaluated on the test set and the obtained predictions are used to compute the True Positives Rates and False Positive Rates (used to plot the ROC curve), and Precision and Recall values (used to plot the PR curve).

3.4.1 Adam Optimizer

In order to create the models, an optimizer is chosen from the Keras library: Adam.

Adam is an adaptive extension of the classic Gradient Descent algorithm. This last one relies on a single learning rate value to update the weights and this value doesn't change throughout the process; on the other hand, Adam manages multiple learning rates and adapts them during the training.

3.4.2 Loss Function and Metrics

The adopted loss function used to obtain the prediction error is the binary cross-entropy, which can be computed as follows:

$$l(\hat{y}, y) = -(y \log(\hat{y}) + (1-y)\log(1-\hat{y})). \quad (6)$$

In order to analyze the results, two metrics are cosidered: Area Under Receiver Operating Characteristic Curve (AUROC) and the Area Under Precision Recall Curve (AUPRC). These metrics are computed evaluating the data in the

confusion matrix, which is a 2×2 matrix containing the following values:

- *True Positive* (*TP*): positive records classified by the model as positive.
- *True Negative* (*TN*): negative records classified by the model as negative.
- *False Positive* (*FP*): negative records classified by the model as positive.
- *False Negative* (*FN*): positive records classified by the model as negative.

The ROC curve is defined through the relation of the *True Positive Rates* (*TPR*) on the y-axis and the *False Positive Rates* (*FPR*) on the x-axis, where:

$$TPR = \frac{TP}{TP + FN} \tag{7}$$

$$FPR = \frac{FP}{FP + TN}. \tag{8}$$

The PR curve is defined through the relation of the *Precision* (*P*) on the y-axis and the *Recall* (*R*) on the x-axis. The first is the percentage of true positive predictions with respect to the total positive labels, whereas the second one is the percentage of true positive predictions with respect to the total positive predictions. They can be computed as follows:

$$P = \frac{TP}{TP + FP} \tag{9}$$

$$R = \frac{TP}{TP + FN}. \tag{10}$$

The AUROC is the measure of the area under the ROC curve, while the AUPRC is the measure of the area under the PR curve.

4 Simulations and Experiments

The algorithm running all the simulations is described by the pseudocode shown in Algorithm 1.

Algorithm 1 CNN architectures

```
for task in TASKS do
  for a in ACTIVATIONS do
    for cl in CONV_LAYERS do
      for hl in HID_LAYERS do
        for un in UNIT_NUMBERS do
          model = cnn_model(a, cl, hl, un)
          model.compile(adam, binary_crossentropy)
          model.fit(tr_X, tr_Y, te_X, te_Y, epochs=10)
          model.predict(te_X)
```

As shown in the pseudocode, there is a loop over all the possible tasks (which are 20, considering all the cell lines), and, for each of these tasks, 16 different models are created, trained and tested. Furthermore, there are other 4 nested loops handling the model creation, by varying the activation function (ReLU or TanH), the number of convolutional layers (2 or 3), the number of hidden layers (2 or 3) and the number of hidden neurons per hidden layers (32 or 64).

Since the number of tasks is considerably large, we were forced to limit the range of the possible varying parameters and, therefore, the number of different possible models.

For the same reason, several parameters were kept constant: the batch normalization was not performed, the max pooling was applied after each convolutional layer, the number of kernels was set to 32 at the beginning and then doubling at each convolutional layer, and the kernel length was set to 8.

Depending on the task, the algorithm execution time varied between 1 hour, for small tasks, and 3 hours, for the bigger ones.

4.1 System architecture

The simulations were carried out on two laptops, characterized by:

- CPU Intel Core i7-6700HQ, GPU nVidia Geforce GTX 950M and 8Gb RAM.
- CPU Intel i7-7700HQ, GPU nVidia GeForce GTX 1060 and 16Gb RAM.

4.2 Implementation details

The whole project was developed using Python 3.7. A first implementation was built relying on *Tensorflow (gpu)* library, while the last one was based on *Keras* library. Both the implementations were developed using *numpy*, to handle the data structures, *scikit-learn*, for the evaluating metrics, and *matplotlib*, in order to plot the ROC and PR curves.

5 Results

Since the original dataset contained more than two classes, a first attempt was made by performing multi-class classification over it. However, the model results were barely acceptable, given that the accuracy was about 60%.

On the other hand, the second attempt carried out better results, since it was performed on the binary classification tasks described in Section 3.2.

In the following subsections, the results obtained by the different models are described for each task.

5.1 AE vs AP

The best model considering this task is the one that refers to the GM12878 cell line (Figure 1) composed by 3 convolutional layers, 2 hidden layers and 64 units per hidden layer and using Relu activation function. The worst results were given by three different models on the K562 cell line (Figure 4), having as structures 3 convolutional layers, 2/3 hidden layers and 32/64 units per hidden layer, and using the hyperbolic tangent activation function. Except for these 3 models, all the others produced acceptable results.

5.2 AE vs IE

Due to the heavy displacement of the dataset concerning this task, it is the task on which our models performed the worst. Indeed, the best model produced an AUROC value of about 0.6 (on the HepG2 cell line - Figure 7), which is 0.3 less than the previous task mean results. This model has 2 convolutional layers, 2 hidden layers, 64 units per hidden layer and uses ReLU activation function.

5.3 AP vs IP

In this task, the best performing model was obtained on the K562 cell line (Figure 16), it has 3 convolutional layers, 3 hidden layers, 64 units per hidden layer and uses ReLU as activation function. From the GM12878 cell line (Figure 13) the model with 3 convolutional layers, 2 hidden layers, 64 units per hidden layer and which uses tanh activation function, carried out the worst values of AUROC and AUPRC for this task, deviating from all the other models, that have an average good performance.

5.4 IE vs IP

The models built over all the cell lines had a very similar behavior, producing small variations among their results. Particularly, the models executed on HepG2 cell line (Figure 11) had overall slightly better results. Nevertheless, these results don't exceed the ones obtained in the other tasks.

5.5 AE $\cup$ AP vs REST

This task considers all the sequences (and labels) inside each cell line, differently from the others. Indeed, its datasets are way bigger than the previous ones, which only contain two CRRs. Due to this fact, it was the most time-consuming task (approximately 3 hours per cell line). The best model for this task worked over the K562 cell line (Figure 20) and has 3 convolutional layers, 2 hidden layers, 32 units per hidden layer and uses ReLU activation function. Besides, at least one model per cell line failed to give good results, probably because of the heavy complexity of the structures (each having 3 convolutional layers).

6 Conclusions

The results obtained by running all the simulations clearly tell us that not only it is possible to predict whether a nucleotides sequence acts as a cis-regulatory region or another one, but that a properly built Convolutional Neural Network can perform very well on this kind of task. Predicting any of the cis-regulatory regions given a nucleotides sequence is not an easy task. Indeed, our fisrt attempt to multi-class classification on the original dataset did not meet our expectations, producing an accuracy of about 60%, as already mentioned in Section 5.

Using the AUROC and AUPRC metrics to evaluate the results, it is possible to identify the best model, which is the one that refers to the GM12878 cell line running on the AE vs AP task, and it is composed by 3 convolutional layers, 2 hidden layers and 64 units per hidden layer and using ReLU activation function. This means that the AE and AP classes are highly separable.

On the contrary, the worst models refer to the AE vs IE task, since, as already mentioned in Section 5.2, the dataset is heavily displaced.

In general, all the models obtained during the simulations have AUROC values ranging from 0.8 and 0.9, and AUPRC values of about 0.9.

Further development and experiments could be done, considering more and different models architectures (varying the parameters). Moreover, wider and more balanced datasets can lead to better results.

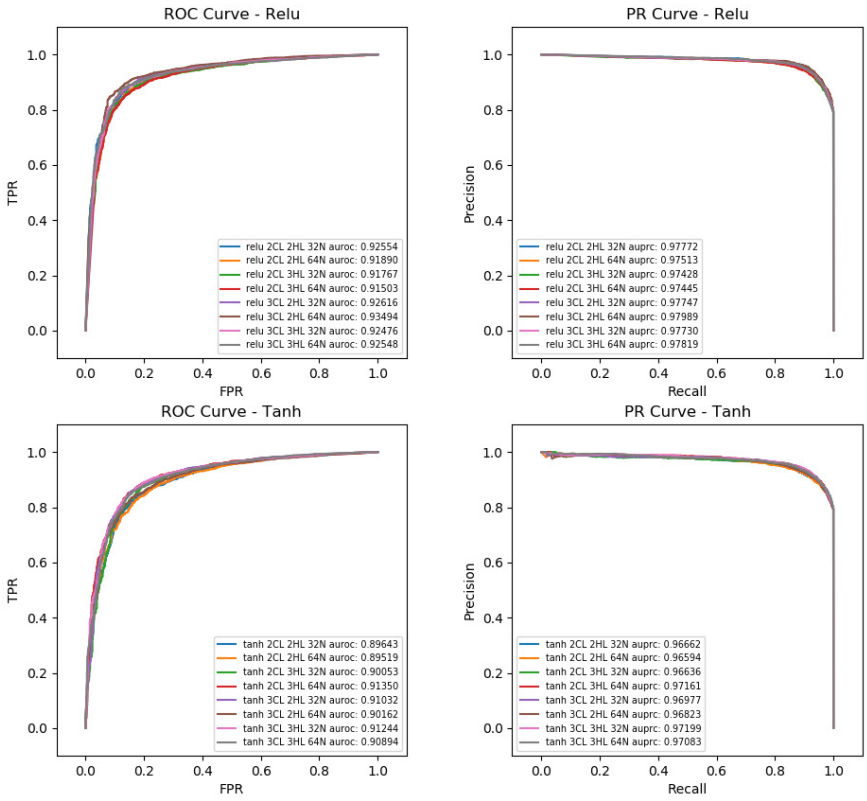

Fig. 1: GM12878 - Active Enhancer vs Active Promoter

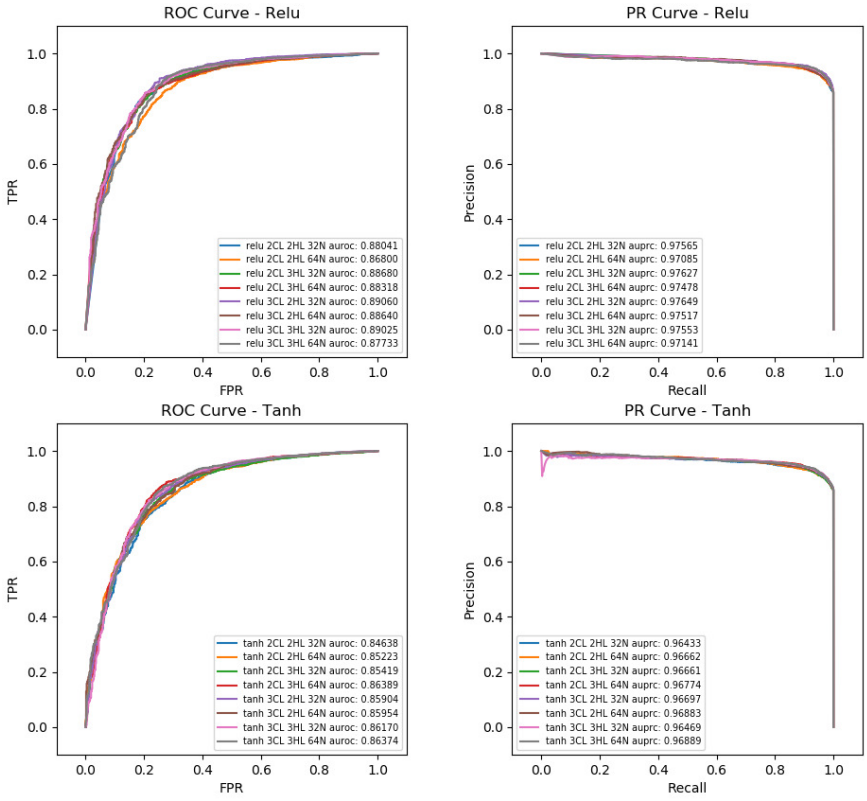

Fig. 2: HelaS3 - Active Enhancer vs Active Promoter

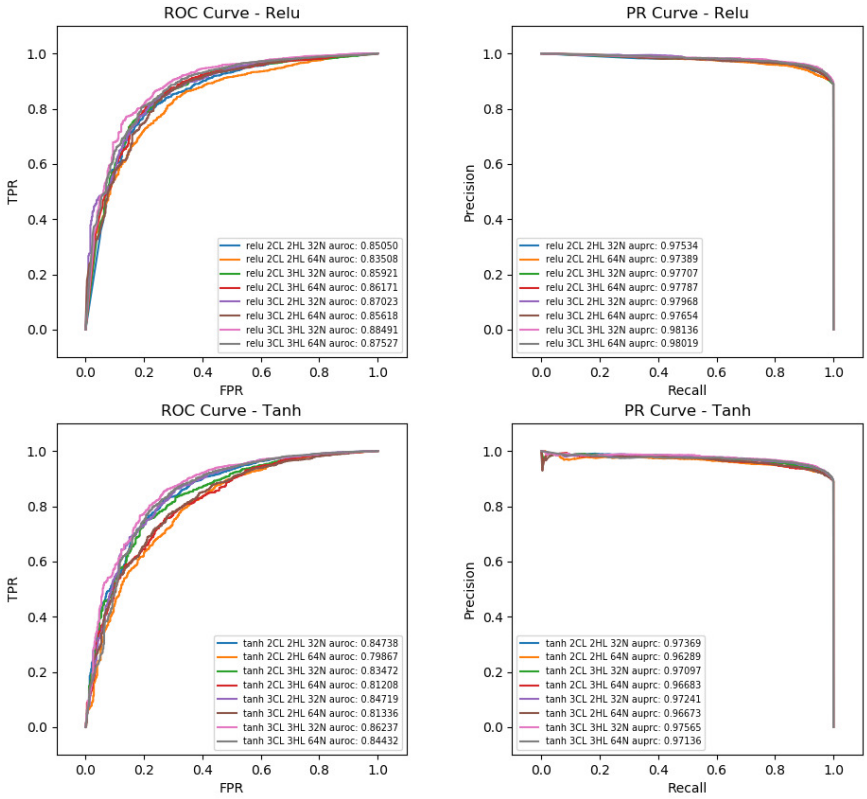

Fig. 3: HepG2 - Active Enhancer vs Active Promoter

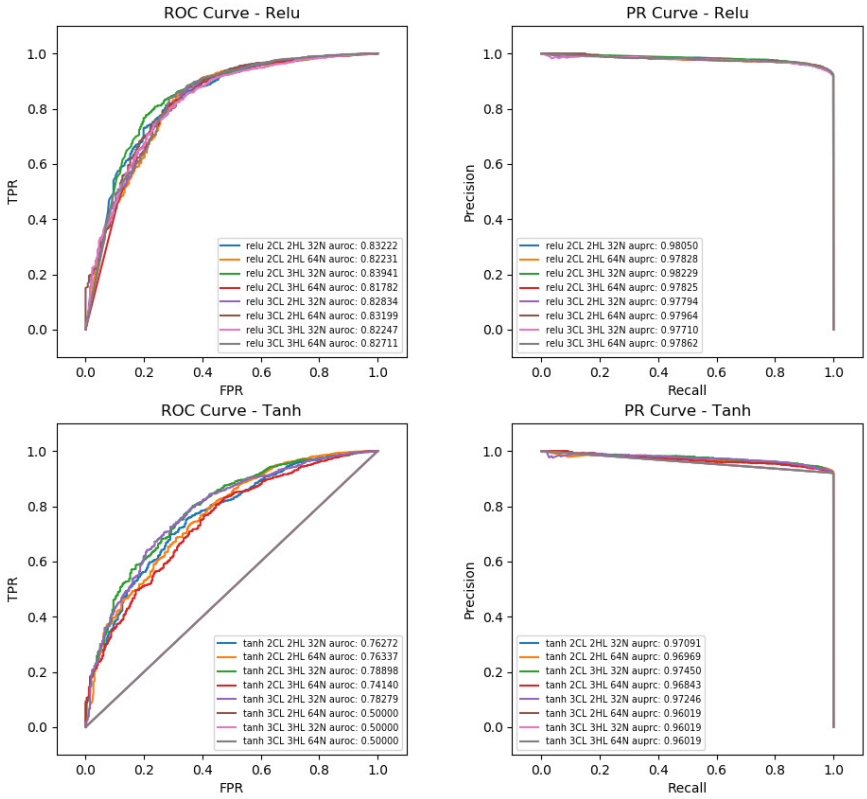

Fig. 4: K562 - Active Enhancer vs Active Promoter

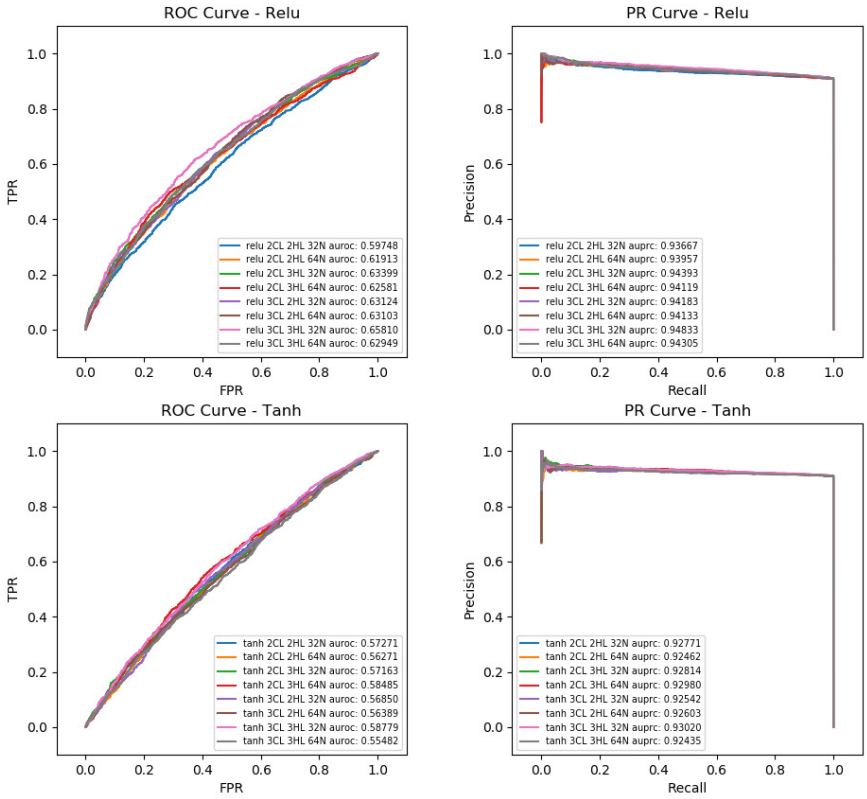

Fig. 5: GM12878 - Active Enhancer vs Inactive Enhancer

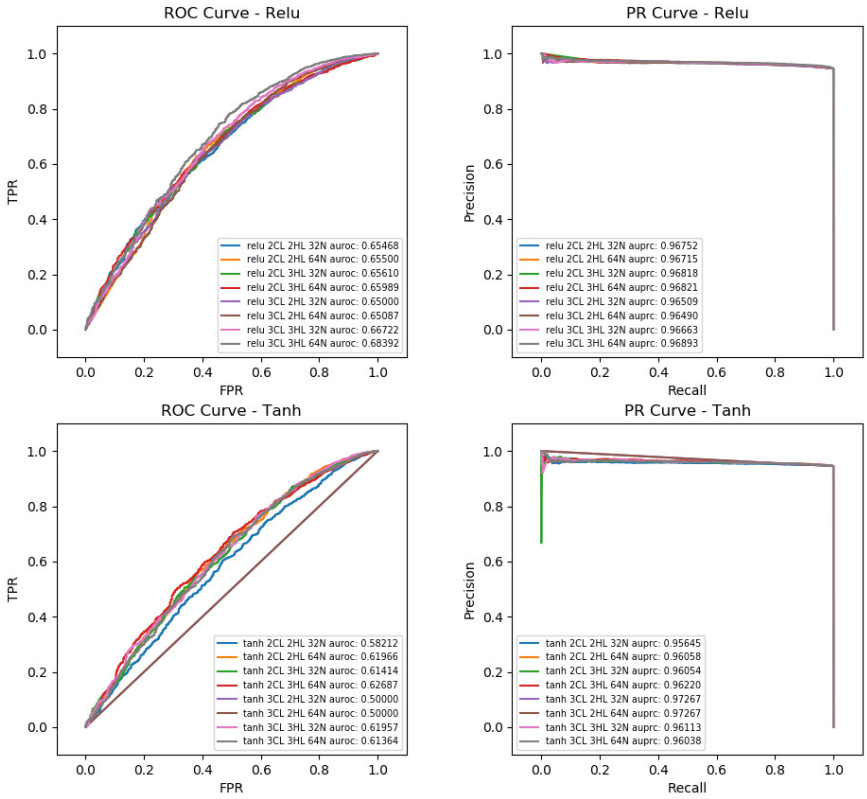

Fig. 6: HelaS3 - Active Enhancer vs Inactive Enhancer

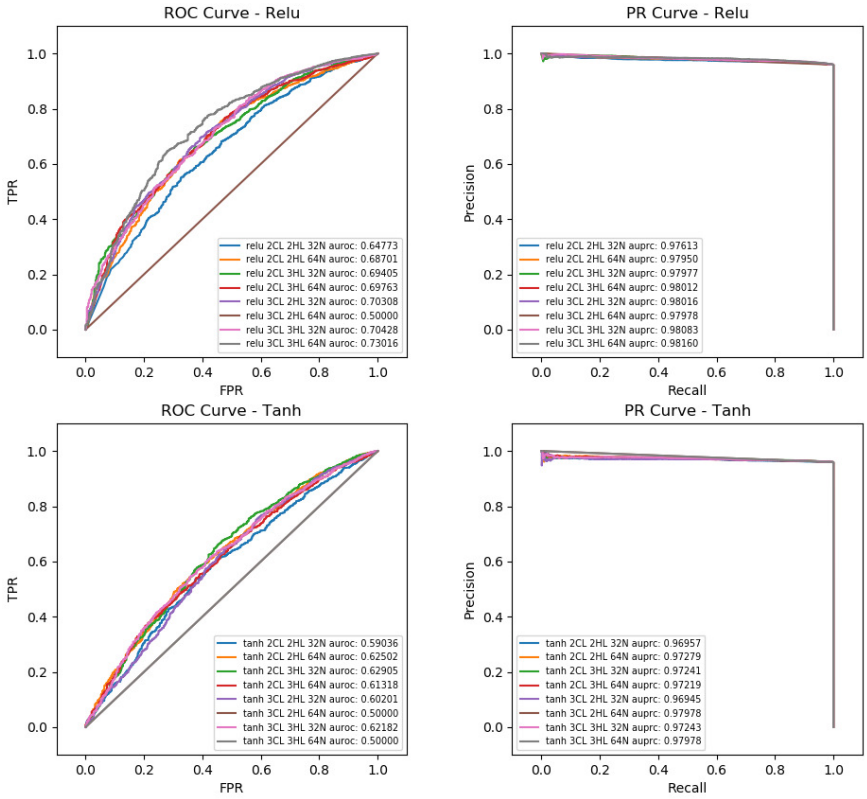

Fig. 7: HepG2 - Active Enhancer vs Inactive Enhancer

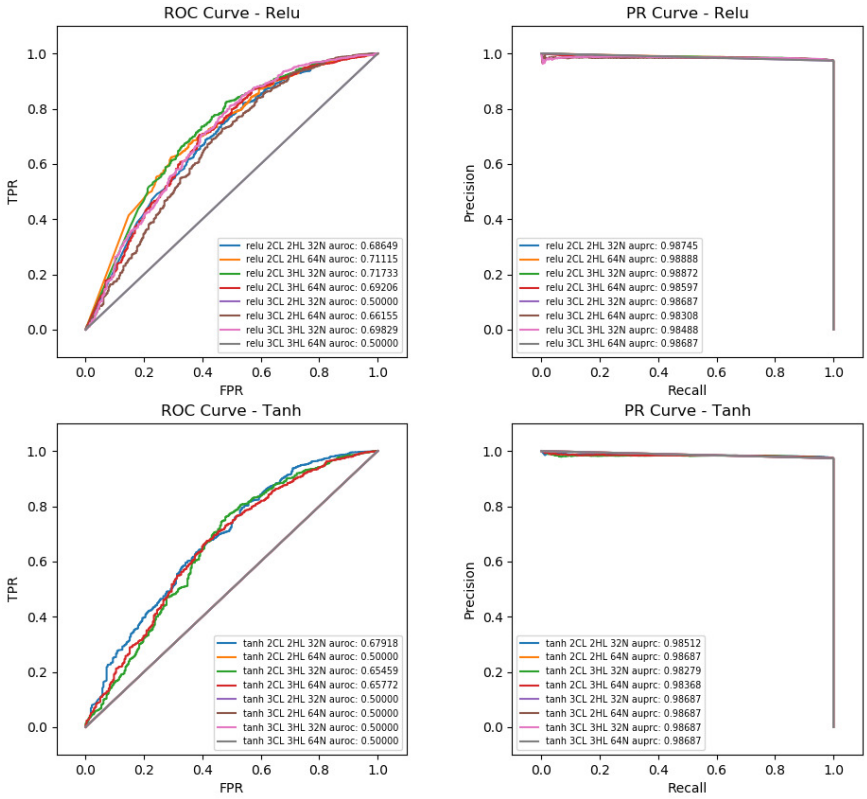

Fig. 8: K562 - Active Enhancer vs Inactive Enhancer

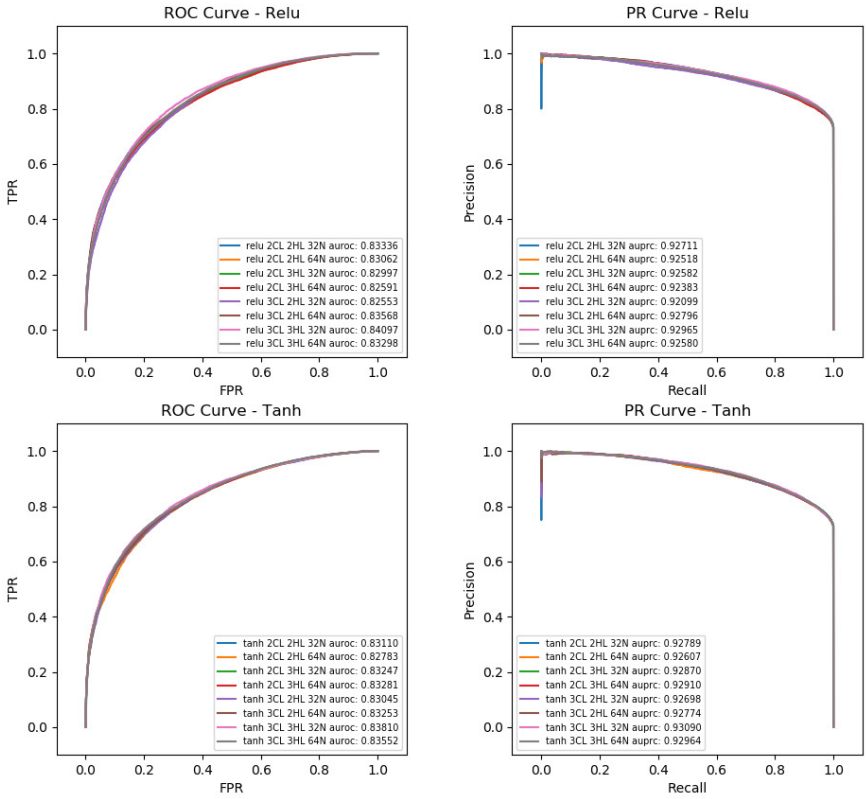

Fig. 9: GM12878 - Inactive Enhancer vs Inactive Promoter

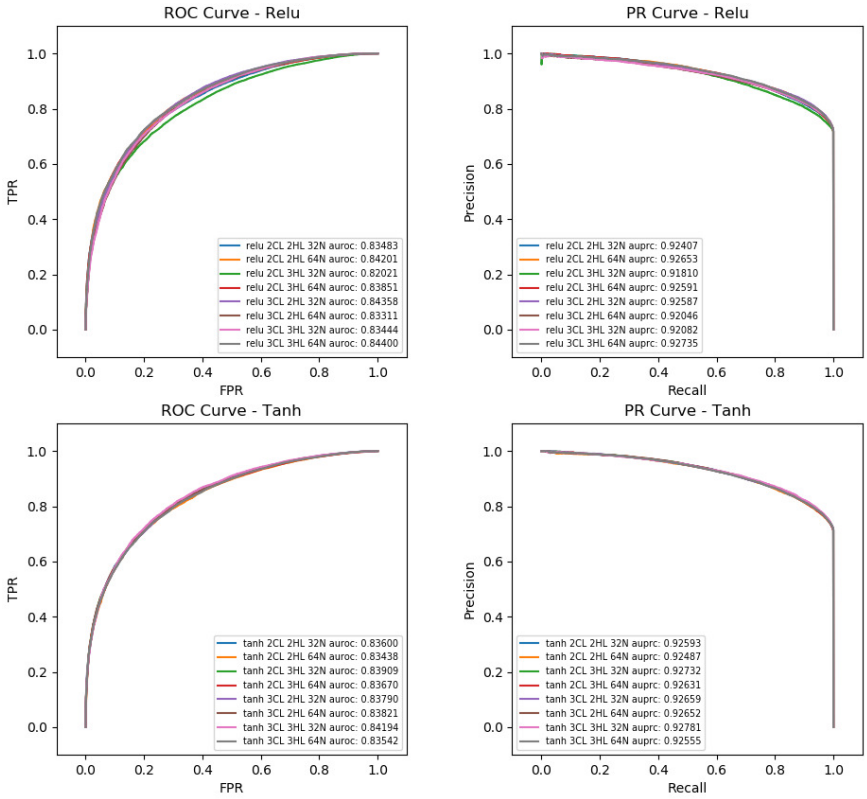

Fig. 10: HelaS3 - Inactive Enhancer vs Inactive Promoter

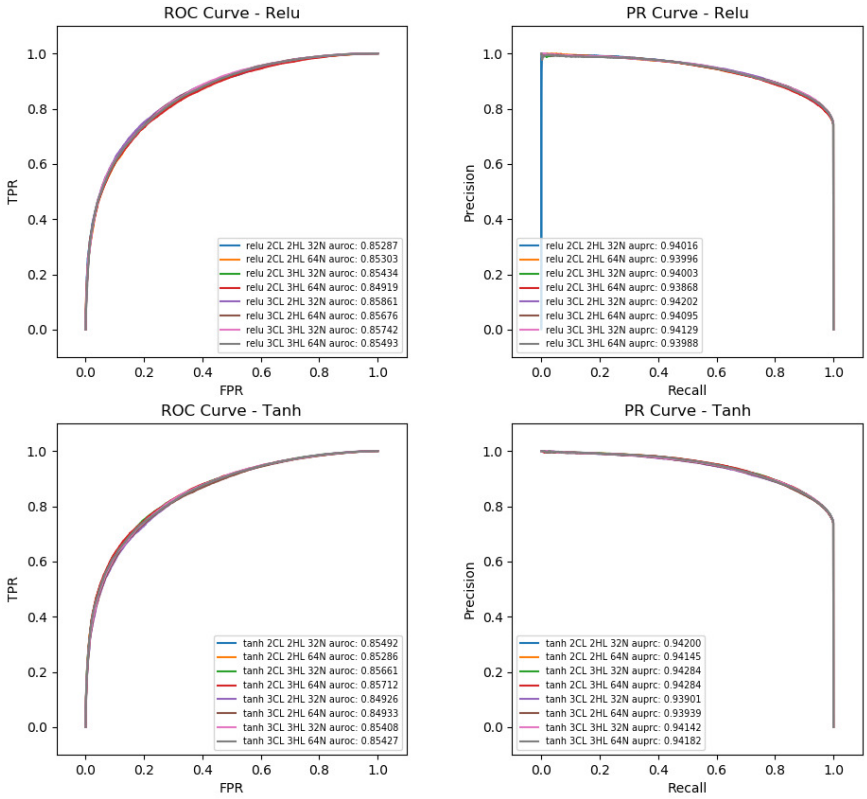

Fig. 11: HepG2 - Inactive Enhancer vs Inactive Promoter

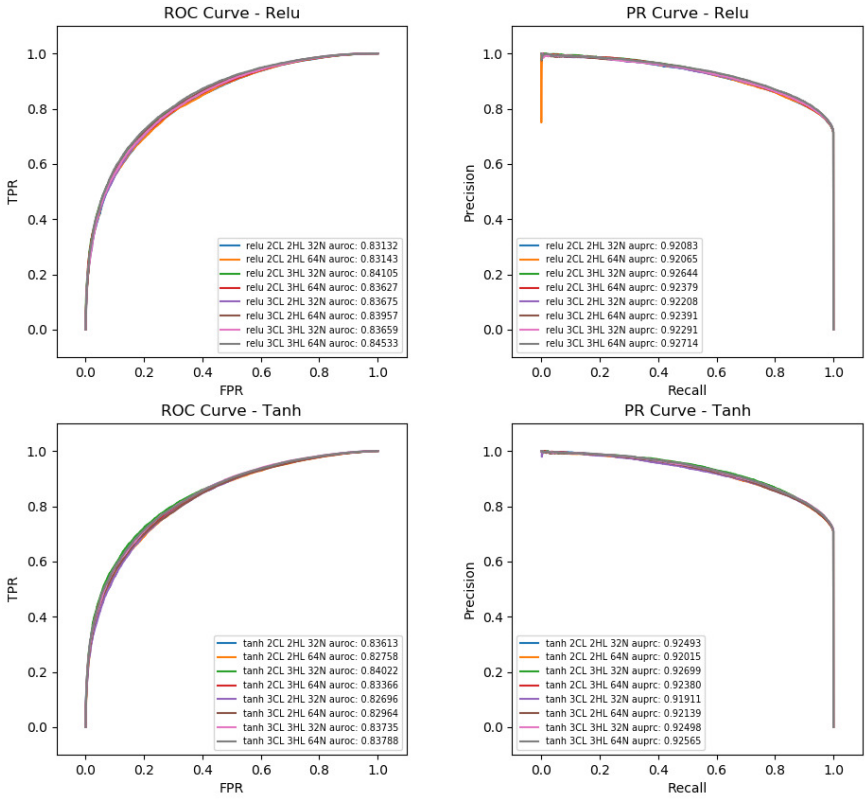

Fig. 12: K562 - Inactive Enhancer vs Inactive Promoter

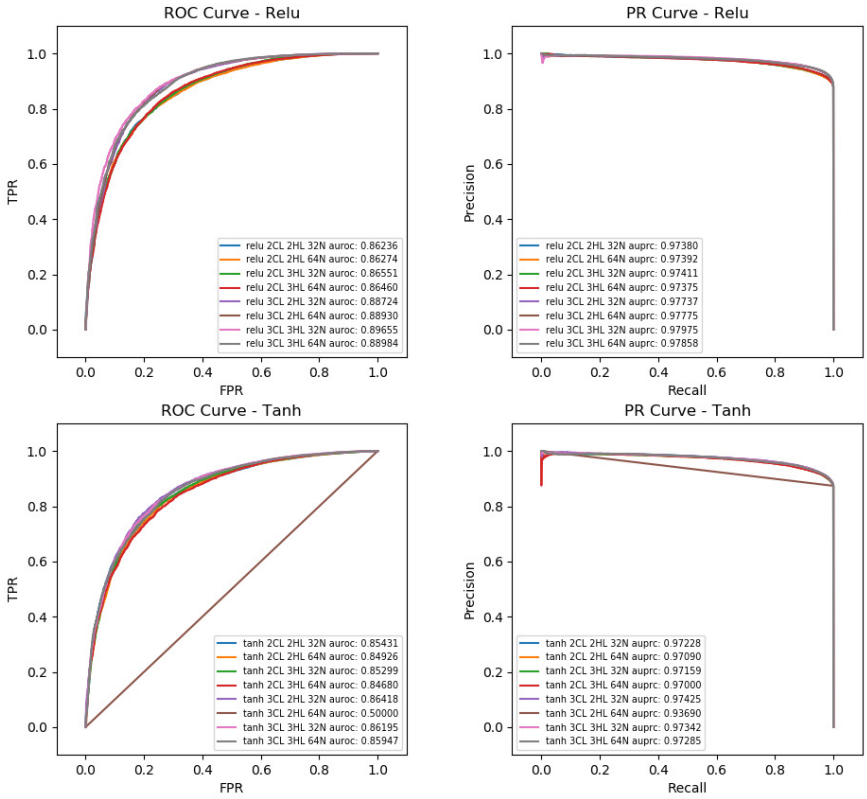

Fig. 13: GM12878 - Active Promoter vs Inactive Promoter

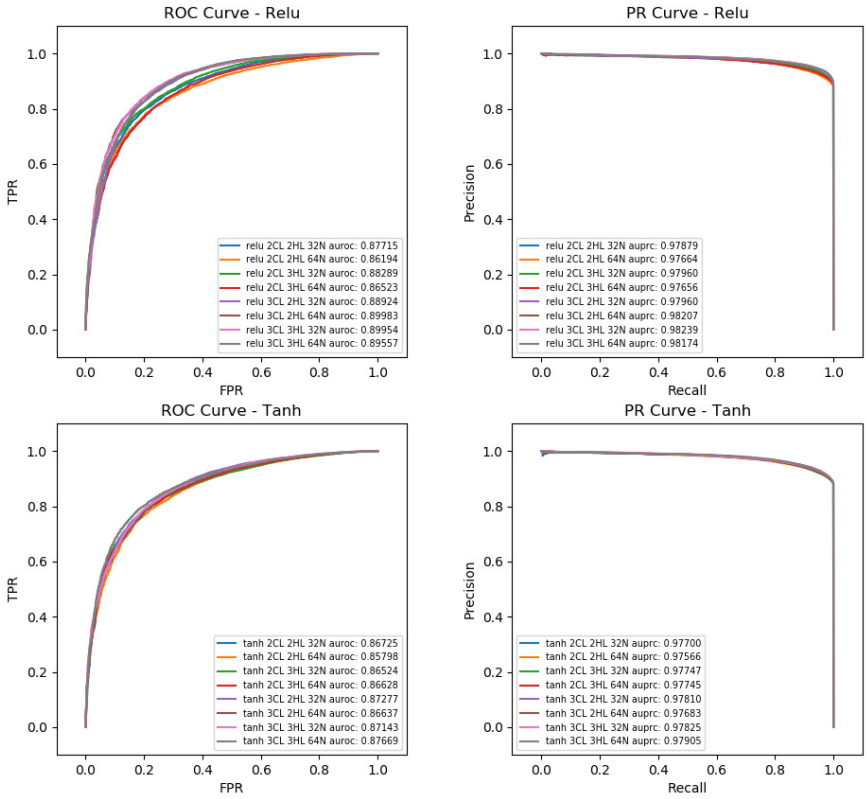

Fig. 14: HelaS3 - Active Promoter vs Inactive Promoter

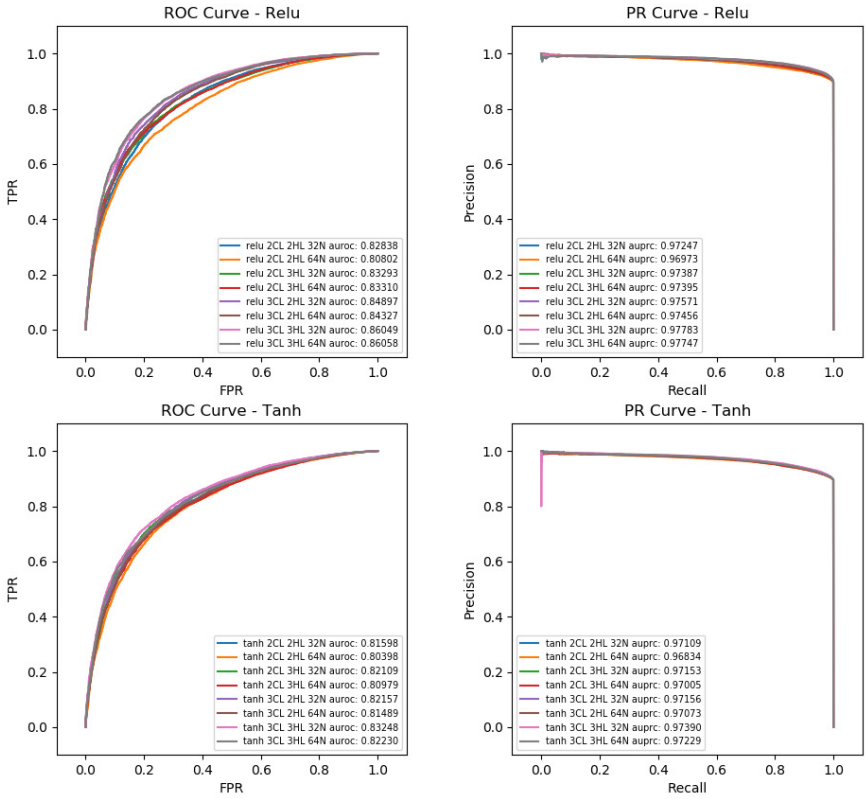

Fig. 15: HepG2 - Active Promoter vs Inactive Promoter

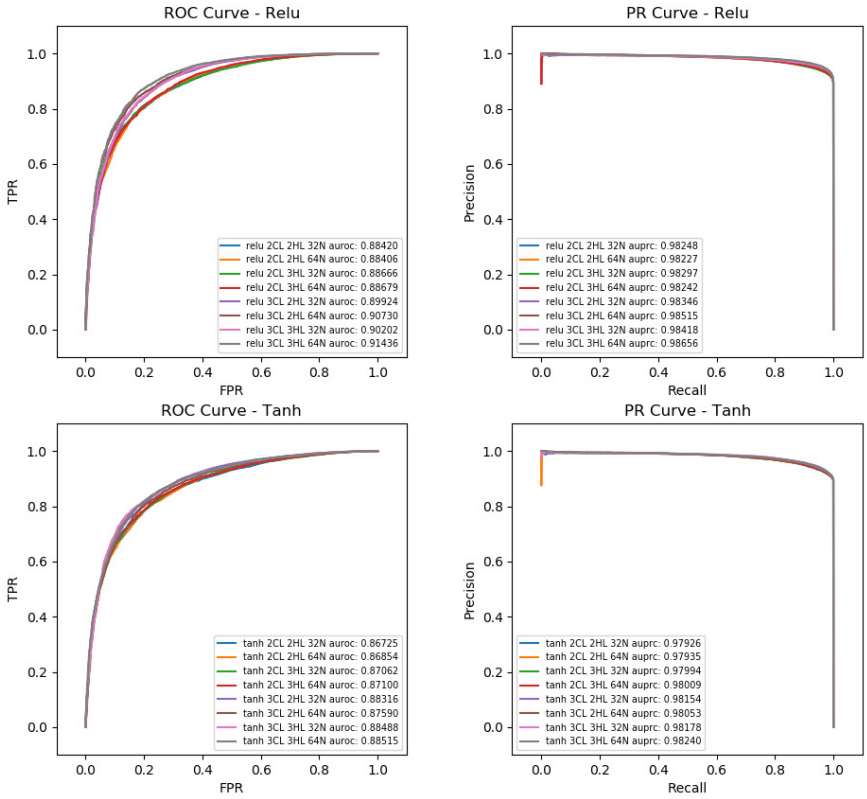

Fig. 16: K562 - Active Promoter vs Inactive Promoter

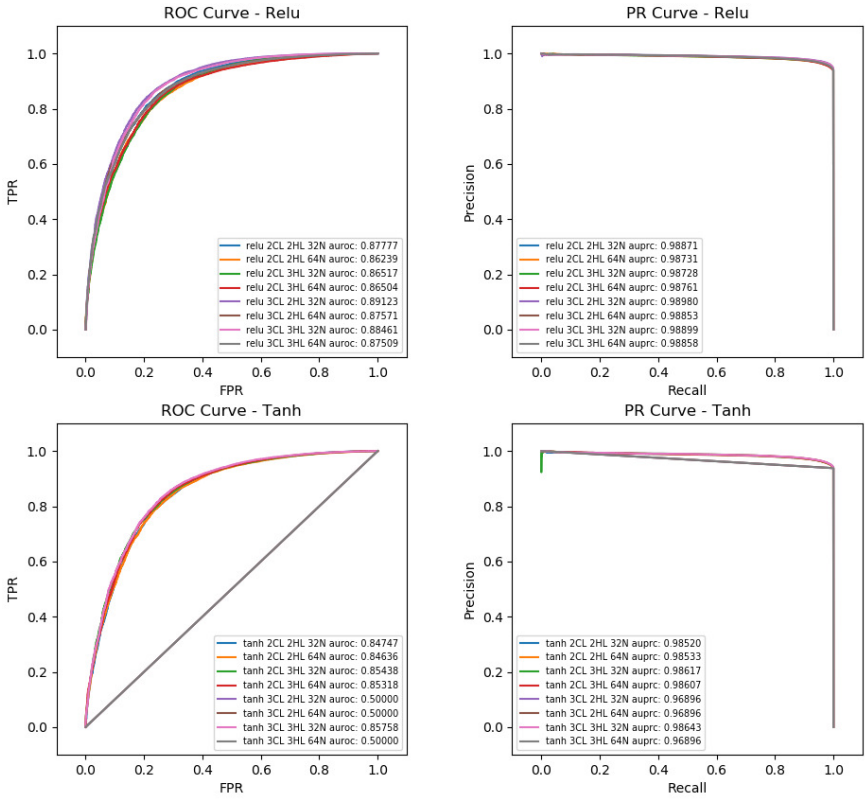

Fig. 17: GM12878 - Active Promoter/Enhancer vs Rest

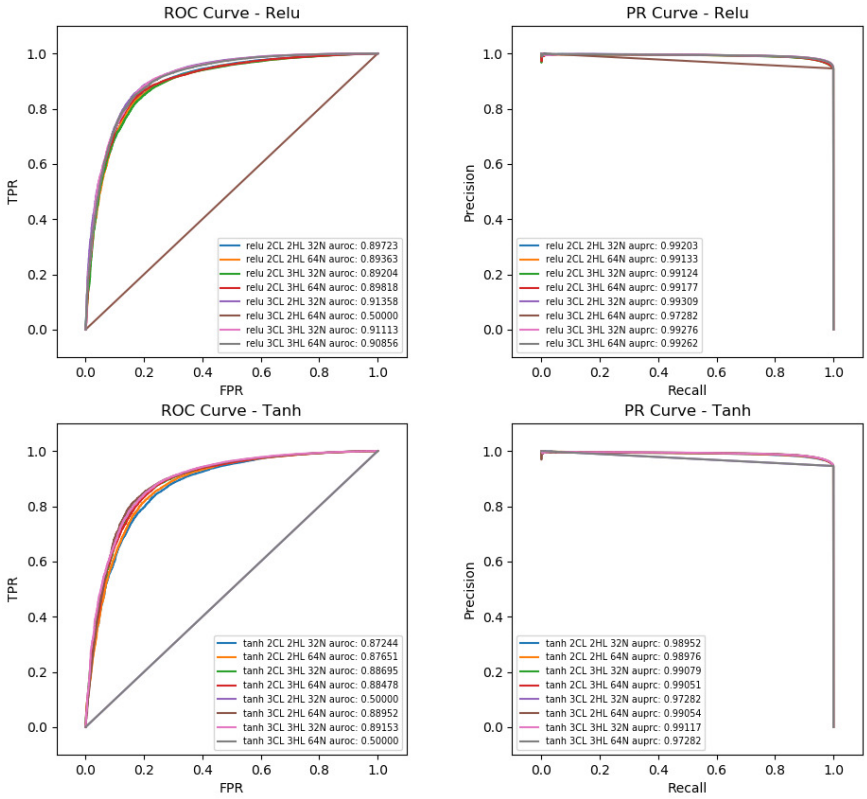

Fig. 18: HelaS3 - Active Promoter/Enhancer vs Rest

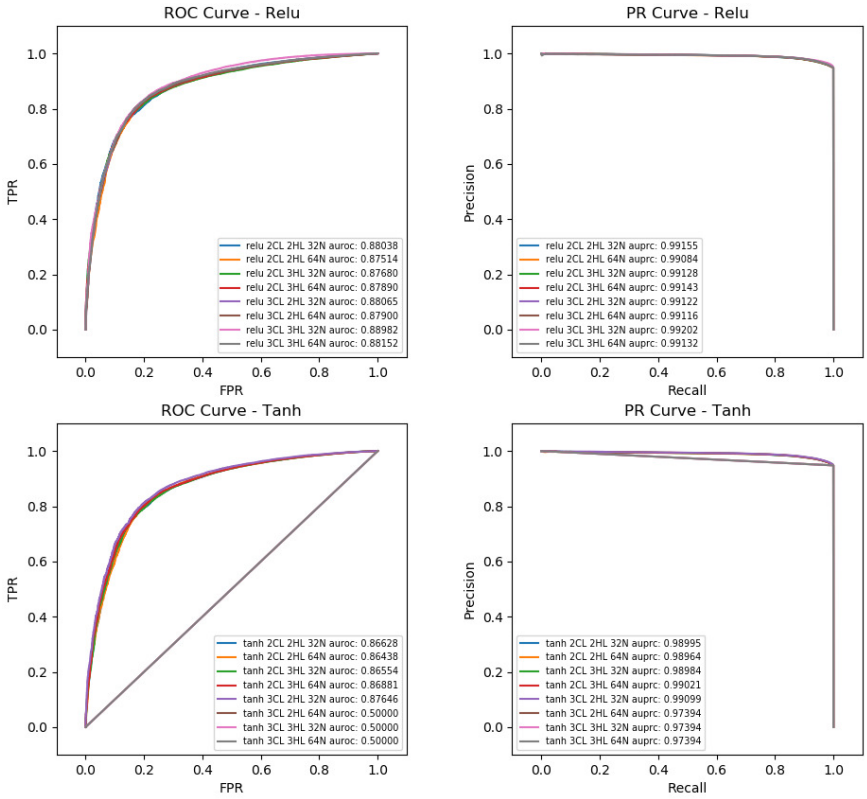

Fig. 19: HepG2 - Active Promoter/Enhancer vs Rest

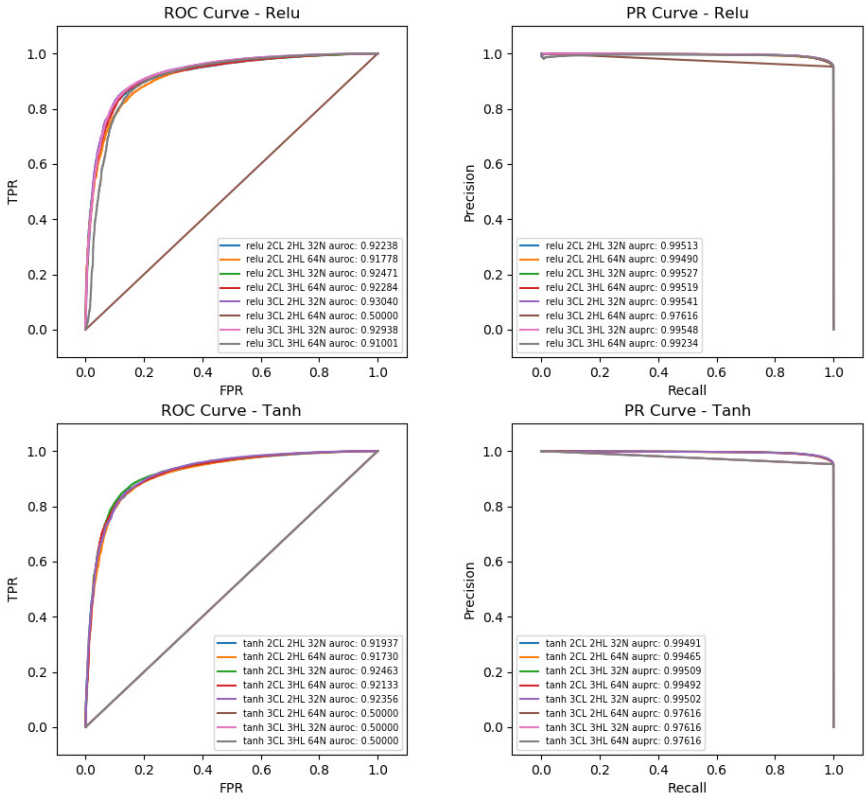

Fig. 20: K562 - Active Promoter/Enhancer vs Rest

REFERENCES

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, p. 436, 2015.

[2] Y. Li, W. Shi, and W. W. Wasserman, "Genome-wide prediction of cis-regulatory regions using supervised deep learning methods," *BMC bioinformatics*, vol. 19, no. 1, p. 202, 2018.

[3] B. Yang, F. Liu, C. Ren, Z. Ouyang, Z. Xie, X. Bo, and W. Shu, "Biren: predicting enhancers with a deep-learning-based model using the dna sequence alone," *Bioinformatics*, vol. 33, no. 13, pp. 1930–1936, 2017.

[4] X. Min, W. Zeng, S. Chen, N. Chen, T. Chen, and R. Jiang, "Predicting enhancers with deep convolutional neural networks," *BMC bioinformatics*, vol. 18, no. 13, p. 478, 2017.

[5] G. A. Wray, "The evolutionary significance of cis-regulatory mutations," *Nature reviews genetics*, vol. 8, no. 3, p. 206, 2007.