Genome Functional Annotation using Deep Convolutional Neural Network

Ghazaleh Khodabandelou[1], Etienne Routhier[1,2], Julien Mozziconacci[1]

1 Sorbonne Université, CNRS, Laboratoire de Physique Théorique de la Matière Condensée, LPTMC, 75005 Paris, France. **2** ENS Cachan, Cachan, France

* ghazaleh@lptmc.jussieu.fr, mozziconacci@lptmc.jussieu.fr

Abstract

Deep neural network application is today a skyrocketing field in almost all disciplinary domains. In genomics, which deals with DNA sequences, the development of deep neural networks is expected to revolutionize current practice, from fundamental issues such as understanding the evolution of genomes to more specific applications such as the development of personalized medicine. Several approaches have been developed relying on convolution neural networks (CNN) to identify the functional role of sequences such as promoters, enhancers or protein binding sites along genomes. These approaches rely on the generation of sequences batches with known annotations for learning purpose. While they show good performance to predict annotations from a test subset of these batches, they usually work less well when applied genome-wide; i.e. for whole genome annotation. In this paper, we address this issue and propose an optimal strategy to train CNN for this specific application. We use as a case study gene Transcription Start Sites (TSS) and show that a model trained on one organism (e.g. human) can be used to predict TSS in a different specie (e.g. mouse).

Author summary

We propose a method to use deep convolution neural networks in order to label genomes with functional annotations. Functional annotations cover any relevant features which can be associated with specific positions on the genome (e.g. promoters, enhancers, conserved regions). This method is based on a optimized generation of the examples used to train the network in order to deal with the well-known problem of using unbalanced data. When these annotations are known in one species, the trained neural network can be used to predict these annotations in a different specie if the mechanisms used to interpret the genomes are conserved in the two species. We use as a case study gene transcription start sites (TSS) in human and show that the model trained on human TSS can be used to recover a similar information on the mouse genome.

Introduction

In genomics, one of the major goals is to interpret the function of DNA sequences. While some of the sequences in the human genome are simply used to encode the protein sequences of the proteome, most of the sequences do not code for any protein. Deciphering these non-coding sequences function is a challenging task which has been increasingly achieved with the development of next generation sequencing. The 3.2

Billion base pair (bp) long human genome is now annotated with many functional and bio-chemical cues. Interpreting these annotations and their relevance in embryonic development and medicine is a current challenge in human health. Deep neural network techniques [1] is undoubtedly a perfect tool in the field as proven by many studies summarized in [2,3]. It was shown in pioneering studies [4–8] that deep convolutional neural network (CNN) is a state-of-the-art neuronal architecture to reliably interpret genetic sequences. In these studies, the training datasets are obtained in a similar fashion. For instance, Min et al. [4] used a deep convolution network to detect enhancers, specific sequences that regulate gene expression at a distance, in the human genome. They considered genetic sequences as 1D images and collected 300 bp long sequences around enhancers as positive inputs and the same numbers of sequences of the same length from non-enhancers regions, as negative inputs. Predictions were made on 10^{-1}th of the total number of sequences and the method reached very good scores ranking it better than previous state-of-the-art, i.e. support vector machine methods [9,10]. Similar training datasets with balanced data, i.e. with a similar amount of positive and negative examples, were used in other papers aiming at identifying promoters [5] or detecting splicing sites [11]. While these approaches are very competitive when applied on test sets derived from the training sequences, they tend to perform less well when applied on full chromosomes or genome-wide. In this paper, we tackle this issue using a different strategy for training.

We use here as a case study one of the most simple and useful genome feature: the transcription start sites (TSS) of genes in the human genome. TSS play a fundamental role in the regulation of transcription of protein-coding and RNA genes since they identify the exact location of the first nucleic acid that will be transcribed in RNA. The promoter region is defined as a larger regulatory region flanking the TSS within a length of $\sim 100 - 1000$ base pairs (bp) located at upstream (5') of the sense strand of the regulated gene. The motifs within the promoter region can be used by proteins as recognition sites providing specific regulation of gene expression as well as precise location of the initiation of transcription. Therefore, a promoter region contains the information to identify the TSS locations *in silico*. It has been however proven quite hard to conceive such a computational model because of the gene-specific nature of the TSS. Although, several studies have been tackling the identification of promoters regions [12–14], there is still not a trustworthy method for prediction of TSS positions on a genome-wide scale.

We inspire our strategy from Alipanahi et al. [6] and Kelley et al. [8] who used unbalanced datasets, i.e. with more negative than positive training examples to predict respectively DNA and RNA-binding sites for proteins [6] and genome accessibility [8]. In this paper, we propose to optimize the ratio between positive and negative examples in order to obtain the highest prediction scores for identifying TSS. Furthermore, unlike the state-of-the-art approaches we do not take into account prediction scores obtained from test sets as a quality measure. These subsets are randomly pulled from the initial training set. We rather assess the ability of our model to find potential TSS in full chromosomes. A direct advantage of this method is that we can now train our network on a dataset corresponding to one organism and then use it for prediction on the genome of a related organism. As a proof of principle, we show here that a CNN trained on human TSS is able to recover TSS in the mouse genome and vice versa.

The program is available to run at https://github.com/StudyTSS/DeepTSS/.

Results

Training models for genome annotation

The problem of detecting TSS using deep neural networks has been already tackled by Umarov et al. [5]. To detect sequence features, convolutional neural networks have proven the most efficient [2,3]. We use here our own architecture, comprising 3 convolution layers followed by one fully connected layer (Fig 1, see Convolution Neural Network (CNN) for details). In order to construct our training set, we use a similar protocol as Umarov et al. [5] (see Convolution Neural Network (CNN)). We train and validate our model on an equal number of 299 bp long positively/negatively labeled input sequences and test it on a test set of 15% of the input data (see Materials and methods). We reach scores of AUROC=0.984 and AUPRC=0.988 comparable to results of [5]. Nevertheless, in order to develop a practical tool for detecting TSS on a genome-wide scale, we apply our trained model over all sequences along chromosome 21 (which is withdrawn from the training set) with a rolling window. This leads to a very noisy output making the model an imprecise tool as depicted on Fig 2 by blue signal. The model trained on the balanced data is applied over the full human chromosome 21 (with a total of 480 TSS). Fig 2 illustrates the predictions of the CNN model over a typical region of 300 kbp containing 7 TSS. Although the predictions present higher scores over TSS positions (peaks), they also present high predictions scores over many non-TSS positions. This phenomenon makes it difficult for a predictive model to differentiate between true and false TSS. Indeed, a predictive model trained on the balanced data and applied on imbalanced data tends to misclassifying the majority class. This is due to the fact that the reality is biased in training phase, since the CNN model learns to attribute equal weights to the positive and the negative classes. This leads to the emergence of many false positives in a context in which the ratio between the positive and the negative class is very different. The model, facing extreme imbalances in new examples such as chromosome 21, fails to generalize inductive rules over the examples. The imbalances hinder learning process because of the difficultly to make conjunctions over the high degree of features with limited examples [15]. Instead, the model learns the inductive rules over the small examples leading to over-fitting, hence the false positives issue.

In regards to address this issue and effectively learning from such dataset we propose a heuristic approach. This approach consists in adding progressively more sparsity (negative examples) into the balanced dataset to alleviating the importance of positive class in training phase and allocating more weight to the negative class. We call such datasets limited imbalanced datasets. This method, detailed in Materials and methods, is expected to mitigate the impact of the extreme imbalanced data on learning process. In order to denote the ratio between positive and negative training examples, we use the Q^* labeling where Q is the corresponding ratio. For instance, on Fig 2 the model trained on the balanced data yielding to blue signal predictions is denoted by 1^*.

To set up a limited imbalanced dataset, first we take the same number of TSS positions as in the balanced dataset, building the positive class. We then select randomly for $Q = 100$ the non-TSS positions, building the negative class. We denote the dataset thereby generated by 100*. Finally, we apply the CNN model on the dataset 100* and assess the efficiency of the trained model. As depicted on Fig 2 by a red signal, the predictions for this model displays a much higher signal to noise ratio, with high peaks over each of the 7 TSS (C21orf54, IFNAR2, IL10RB, IFNAR1, IFNGR2, TMEM50B, DNAJC28) and a much weaker signal between these sites. Predicting TSS using the 100* model is thus much efficient, generating no false positive signal.

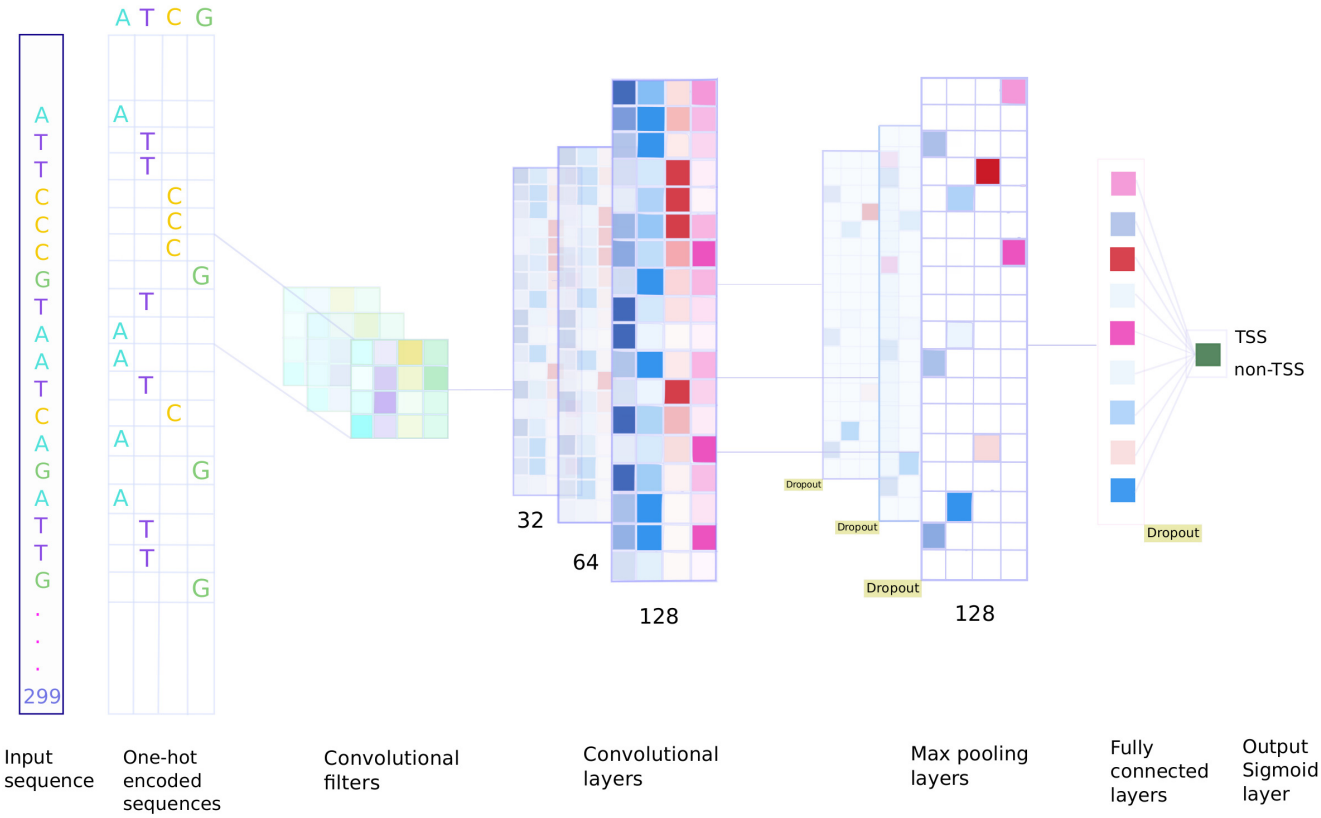


Fig 1. Overview of the CNN model. Nucleotide sequences of size 299 bp long is one hot encoded forming a matrix of shape 4×299. The first convolutional layer performs the convolution operation on successive input matrices to recognize relevant patterns. The next convolutional layers model the correlation among patterns in previous layers and form high-level features. Max-pooling layers reduce the input patterns to a lower dimension resulting in less computational cost and promote high-level features detection. Dropout layers discard randomly some outputs of previous layers to avoid over-fitting. Fully connected layer deals with linear and non-linear combination of high-level features arising from the convolutional layers in order to make the final decisions.

Investigating the effect of random selection of the negative examples over predictions

Since the negative examples are randomly picked out of the genome, the performance of the model in different regions of chromosome 21 could vary for different sets of training examples. To investigate this variation, we setup 30 balanced datasets and train them separately using CNN. We then apply the 30 models over human chromosome 21 to study the fluctuation of predictions. The variation of 30 predictions is depicted in Fig3. The maximum and minimum predictions for each genomic site are highlighted by black and red colors, respectively. The other 28 predictions are colored in gray. The first observation is that almost all predictions peak over the gene DIP2A. However, it is not the case for the DIP2A-IT1 gene, which reflects the fact that trained models using different training sets can yield different prediction scores even over the TSS positions. Indeed, the significant gap between the minimum and maximum predictions demonstrates the variability of prediction obtained with different balanced datasets. While some of these predictions report weak scores others represent high scores. This shows the model uncertainty when predicting new observations, i.e. chromosome 21. As explained in the previous subsection, adding more negative examples to the training set

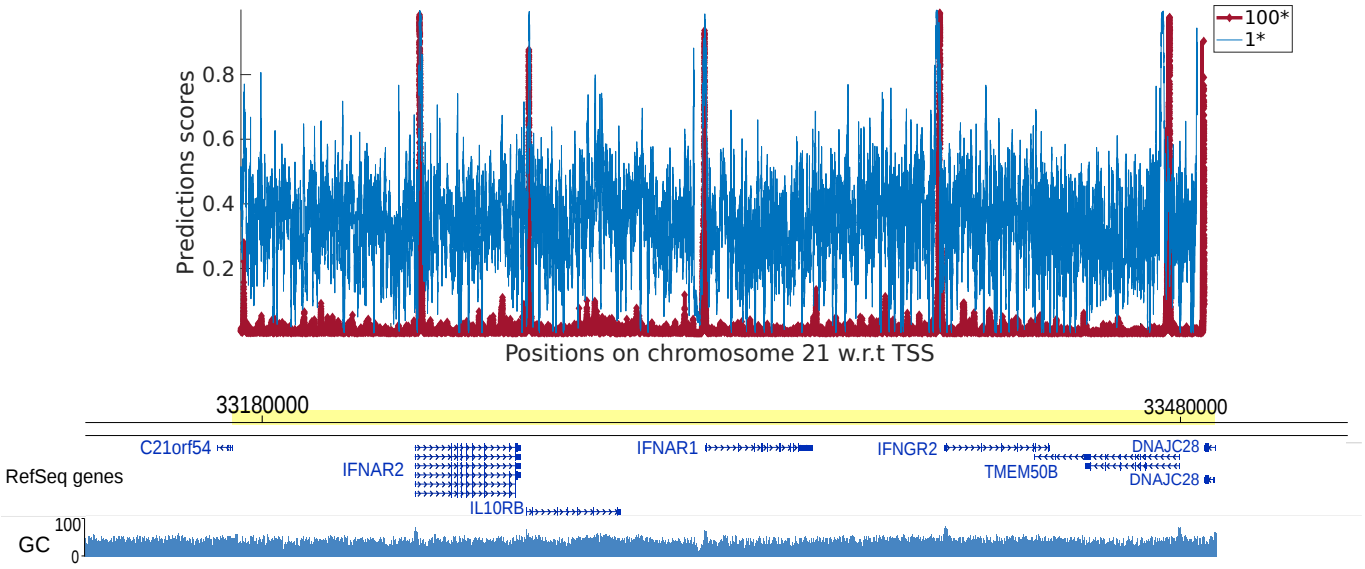


Fig 2. Application of a neural network model trained to find TSS over a 300 kbp region of chromosome 21. Prediction scores for 1* and 100* models, respectively in blue and red. The position of genes is indicated below. The GC content is indicated as blue bars below the genes. Both models detect effectively seven TSS positions over a fragment of chromosome 21, still the model 1* returns many false positives. Adding more negative examples into the balanced data using the model 100* mitigates obviously the false positives while preserving the high scores over TSS.

gives the model a better chance to identify false positive when applied genome-wide.

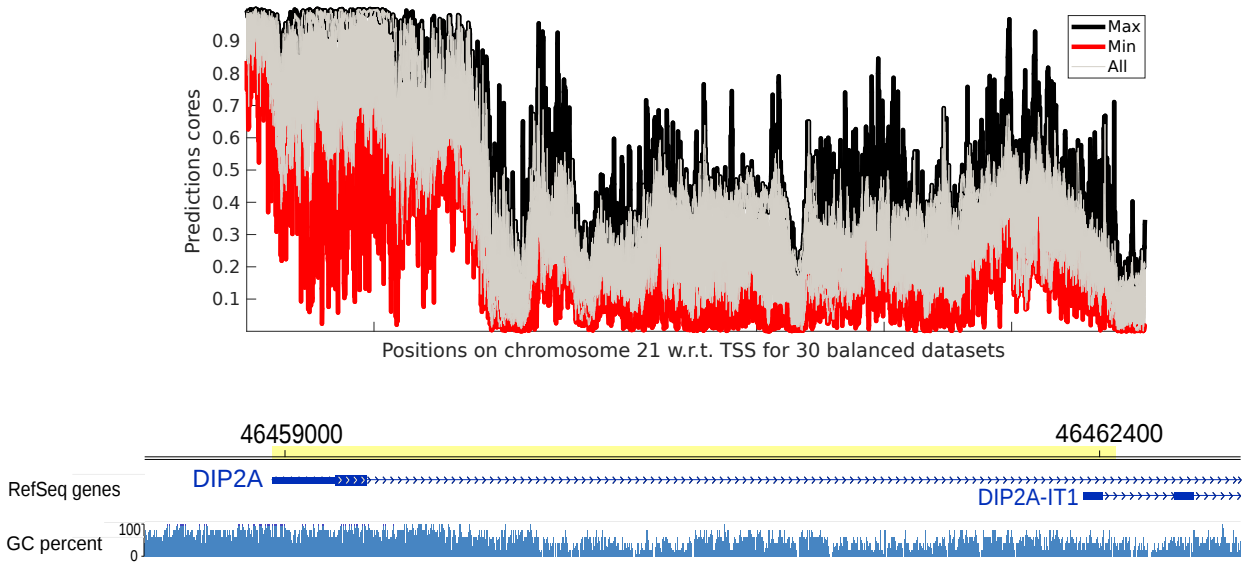


Fig 3. Application of 30 deep convolutional neural network models trained to find TSS over a 3.2 kbp region of chromosome 21. All the predictions are plotted in gray. At each site, the min and the max of the predictions are indicated respectively in red and black.

Comparing 1* and 100* models over all TSS of chromosome 21

In the previous subsection, we showed that training the CNN with more negative examples improves its performance for genome-wide prediction. We use the trained model over the full chromosome 21 and present the results in a heat-map form. On Fig 5 (a) and (b) each horizontal line corresponds to the standard score of the predictions computed over ±5000 bp of each TSS of chromosome 21 for the models 1* and 100*, respectively. The central axis indicates the exact position of the TSS and positions around the axis indicate the neighboring regions. While the model 1* Fig 5 (a) presents a noisier signal around TSS positions, the model 100* Fig 5 (c) displays a higher signal to noise ratio. Fig 5 (c) and (d) illustrate the standard score curves averaged over all the TSS of chromosome 21 for the model 1* and 100*, respectively, allowing us to quantify the average signal to noise ratio. We call the areas under the curves λ score. This definition implies the fact that a larger λ score corresponds to a better signal to noise ratio. In this particular case, we find a λ score of 5 and 10 for the 1* and 100* model, respectively. We could vary the ratio between positive and negative examples used for training from 10* to 100* and λ scores showed almost no variations (see Fig 4 (a) and (b)).

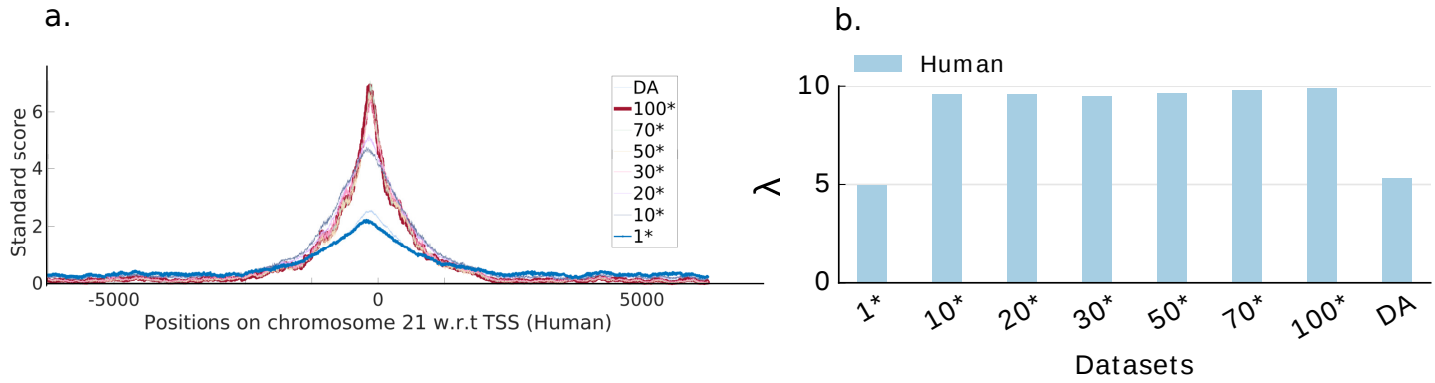


Fig 4. Standard score and λ values for different datasets. For the purpose of genome-wide annotation, the robustness of a model cannot be determined only based on test set performance, since test set contains the same distribution as training set along with much lower samples size. Therefore, a trained model could produce high performance applied on test set but it might fail to fit new observations and to generalize the trend and inductive rules out of training set. To investigate the performance of the model on the balanced data on the genome-wide scale, we apply the trained models on an unseen part of the genome (excluded from training phase), i.e. chromosome 21 for human, chromosome X for mouse. In this perspective, the costume metric λ is used to evaluate the quality of the models. (a) standard scores averaged over 5000 bp flanking the TSS of chromosome 21 for the models 1*, 10*, 20*, 30*, 50*, 70*, 100* and DA. The limited imbalanced data scores double up the balanced data score. While the limited imbalanced data scores variate slightly, the highest and the lowest standard scores belong to models 100* and 1*, respectively. The area under the standard score curve λ follows the same tendency. (b) The λ reaches at the maximum values for model 100*. It should be pointed out that, since the performance of models 30* and 100* varies slightly, it is possible to use the 30* model instead of 100* to perform less costly and less time-consuming experimentations.

In order to illustrate the prediction scores on given examples, we pick four representatives TSS along the chromosome. The first arbitrary selected TSS corresponds to the gene CXADR, shown in Fig 5 (e). While the prediction of model 1* results in poor averaged standard scores over all positions, the averaged standard score of model 100* strongly peak around the TSS positions and shows low variations over non-TSS positions. Fig 5 (g) depicts the second selected TSS corresponding to a specific gene KRTAP19-2. Fig 5. This gene is part of a cluster of similar genes belonging to the family of Keratin Associated Proteins (highlighted by a yellow rectangle on Fig 5 (a)

and (c)). For this particular cluster, the predictions are poor for both 1* and 100*, probably reflecting a specific TSS signature for these proteins. Another example of gene cluster with a poor prediction score for TSS is the t-RNA cluster, highlighted in green. Fig 5 (h) and (i) displays the predictions around the TSS of the SCAF4 and PCNT, C21ORF58 genes, respectively. On these more typical TSS the 100* model shows a higher signal to noise ratio than the 1*, as a consequence of this, TSS are faithfully detected.

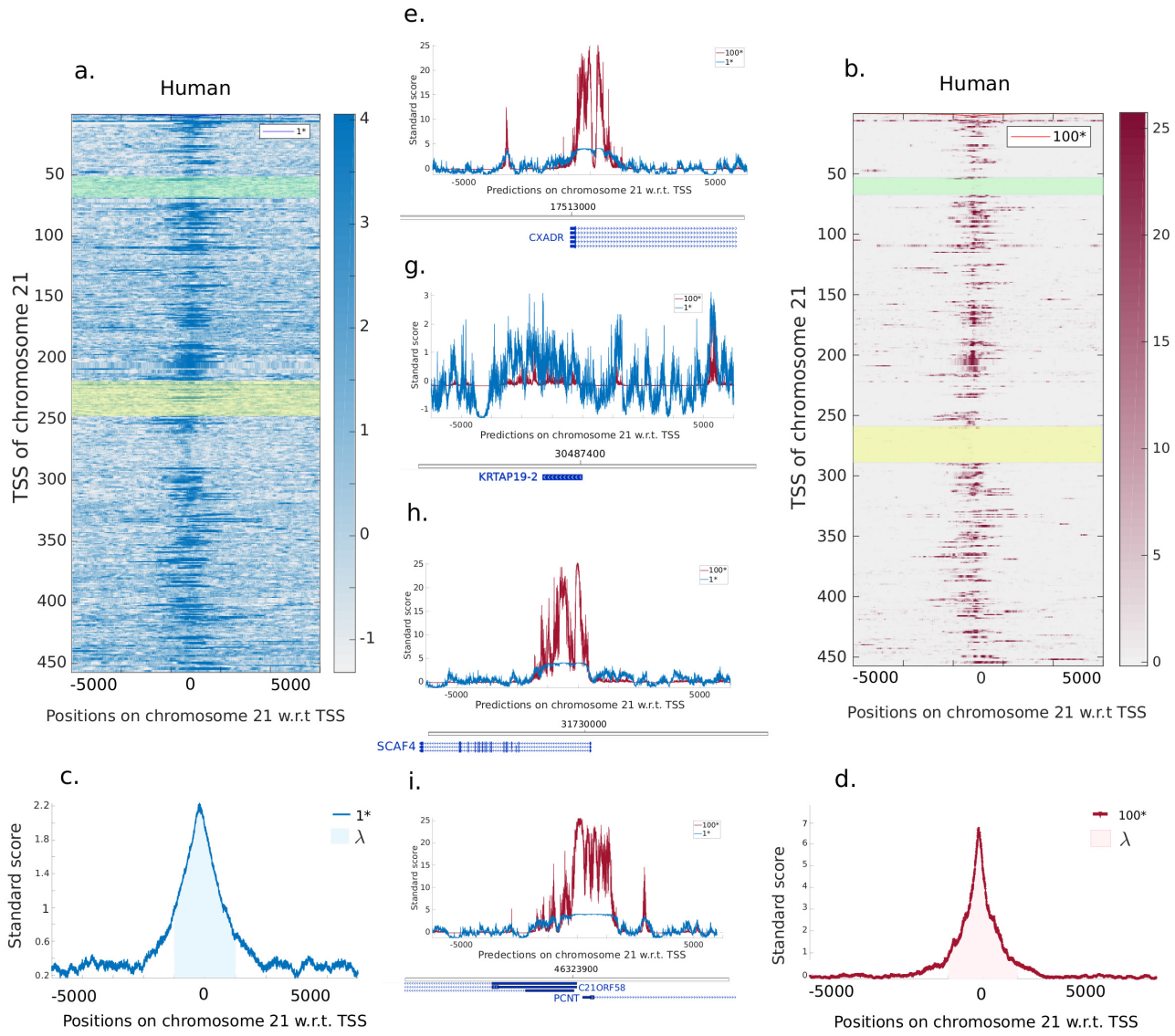


Fig 5. Comparison of the models 1* and 100* predictions over chromosome 21. (a,b) Heat maps depict the standard score of the prediction for respectively the 1* and 100* models flanking 5000 bp around each TSS of chromosome 21. (c,d) Averaged standard score of the predictions over each TSS of chromosome 21. (e-j) Zoom on regions around selected TSS. Genes are indicated at the bottom of each plot.

Learning TSS in a cross-species context: human and mouse

In order to demonstrate generalizability of the proposed method to predict effectively TSS positions, we replicate our TSS analysis in mouse spices using the mm10 genome assembly. The 1* and 100* models trained on mouse are applied over the mouse chromosome X to assess the model performance, see Fig 6 (a), (d) and (g). Note that the chromosome X is discarded from learning set. The averaged standard score reaches maximum values of 2.8 and 7.5 respectively for the 1* and 100* models in quantitative agreement to the model performance observed in human genome.

We then test the possibility for our method to assess TSS positions in one organism when trained on a different albeit related organisms. We thus apply the mouse trained model on human chromosome X and the human trained model on mouse chromosome X. The two chromosomes carry homologous genes, the number of annotated TSS varies with a total of 4,968 TSS in human and only 2,005 TSS in mouse. While the model trained and applied on mouse shows a better signal to noise ratio, the same model applied to human chromosome X still captures most of the TSS and gives a maximum averaged standard score of 5.9 for the 100* model, see Fig 6 (b), (e) and (h). Similarly, the models trained over human capture most of TSS on the mouse X chromosome 6 (c), (f) and (i) and reaches a maximum averaged standard score of 6.8 for the 100* model. In all cases, the signal to noise ratio is improved in the models 100*.

All results for the cross-species comparison on chromosome X are summed up in Fig 7. Overall, the scores show comparable results for both 1* and 100* models when trained over human and applied on human chromosome 21 (H-H), trained over mouse and applied on mouse chromosome X (M-M), trained over mouse and applied on human chromosome X (M-H) and trained over human and applied on mouse chromosome X (H-M). The human model applied on human provides the highest scores for both 1* and 100* models, possibly due to the fact that the TSS annotation is more complete.

Conventional measures for model performance assessment do not reflect their genome-wide quality

We next investigate the scores obtained for Q^* models (with $Q = 1$, 10, 20, 30, 50, 70, 100) as well as a data augmentation strategy (DA, see Materials and methods) using conventional metrics as explained in Performance measures. To evaluate the performance of the models trained by supervised learning techniques, they are applied over test sets. Conventionally, these sets are pulled out of the learning examples in a random manner. As reported in Fig 8, the CNN model applied on the balanced data (1*) yields the best performance on such a test set regarding Receiver Operating Characteristic curve (ROC) and precision/recall curve (PR) with respect to other Q^* models. In the balanced data case, the ROC curves ascent much faster towards to top-left corner comparing to the limited imbalanced datasets (Q^*) suggesting the model can achieve relatively high true positive rate at relatively low false positive rate. The results also suggest the superiority of the model applied on the balanced data. Counter intuitively, the model which gives the best scores on a conventional test set yields the poorest predictions when used genome-wide. Both metrics indicate the highest score for DA model and the lowest for the genome model. All 299 bp no-overlapping windows that do not contain a TSS are taken as negative examples. Although, the DA outperforms the other models scores, its performance is biased by the fact that similar sequences, albeit with different offsets with respect to the TSS positions, are found in both training and test sets. Once again, the model giving the best scores on a conventional test set yields the poorest predictions when used on the genome-wide scale.

Following this observation, we verify whether this holds also for other metrics commonly used to evaluate the performance of the CNN models over test sets. Fig 9 (a)

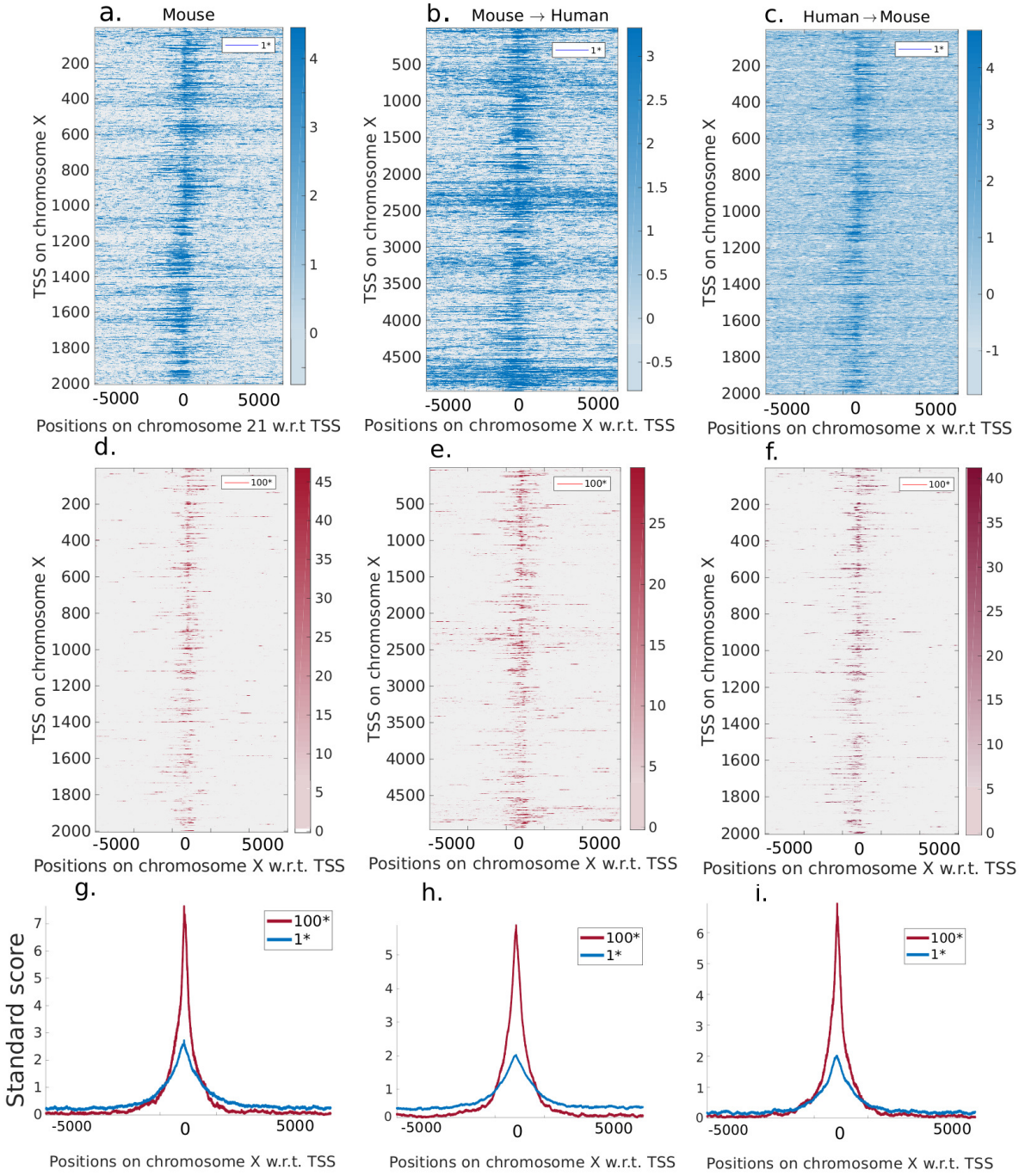


Fig 6. Overview of human and mouse models performances over the chromosome X. (a) and (c) Heat maps depict the standard score of the predictions for respectively the 1* model trained on mouse and applied on mouse (a), human (b) and for the 1* model trained on human and applied on mouse (c). (e) and (g) Similar to (a) and (c) with the 100* model. (h) and (j) Averaged standard score of the predictions over of the heat maps over all TSS, for the models 1* and 100* similar to (a) and (c).

recapitulates the results presented in Fig 8. The Area Under Precision/Recall Curve 203

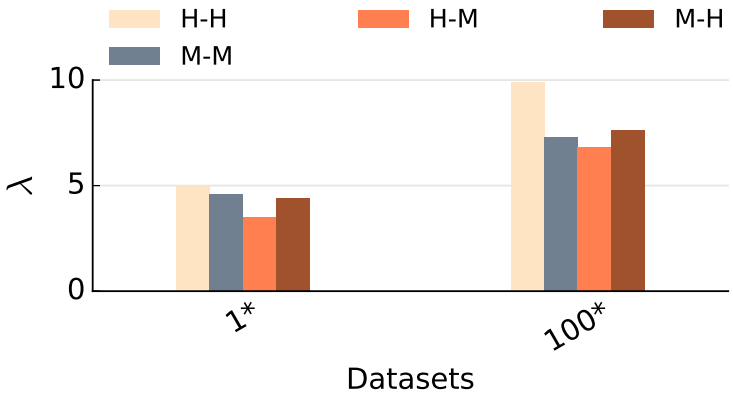


Fig 7. Cross-species comparison of λ scores for the models 1* and 100* for human and mouse-trained models applied on human and mouse chromosome X. The model 100* returns higher scores for all experiments comparing to model 1*. The highest score is provided by model 100* (H-H) and the lowest one by model 1* (H-M).

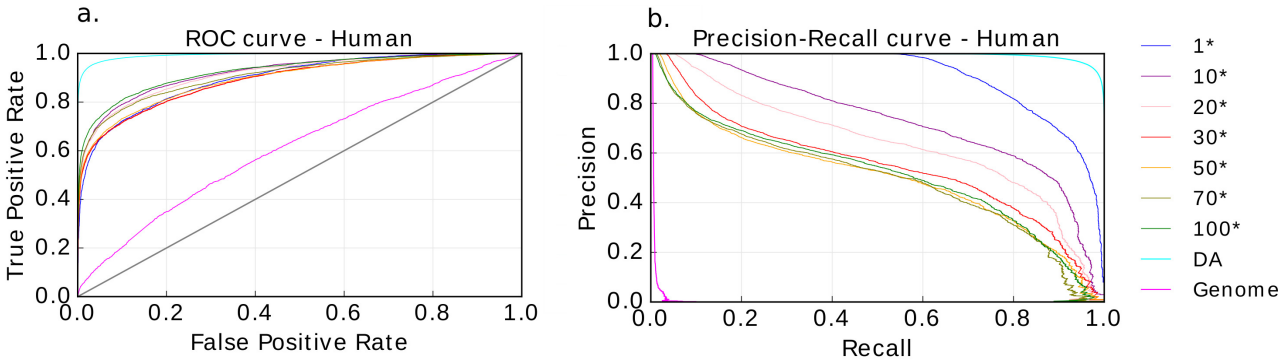


Fig 8. Results on ROC and PR obtained over test sets. The model 1* represents the balanced data. The model applied over augmented data (DA) reports the highest score on both metrics while the model applied over genome wide shows the lowest scores.

(AUPRC) reveals an uppermost score for the balanced dataset but it deteriorates across the limited imbalanced datasets. The Area Under an ROC Curve (AUROC) on the other hand presents stationary scores across all models. Given that there are many more true negatives than true positives within imbalanced datasets, PR is considered as a trustworthy measure because it does not take into account the true negatives. Indeed, AUPRC curve is misleading when applied to strongly imbalanced datasets, because the false positive rate (FP/total real negatives) does not decrease drastically when the total real negatives is huge. Whereas AUPRC is highly sensitive to FP, it is not impacted by a large total real negative denominator. In Fig 9 (b), F1-score reports a weighted average between precision and recall per class. While, the F1-score enhances for non-TSS class across the datasets an opposite trend is observed for TSS class. This signifies the more negative samples are introduced in the datasets, the more the model has the difficulty to return efficient predictions for TSS class. We observe the same trend where two classes are combined. Fig 9 (c) shows the scores for binary cross entropy (equation 2), MCC (equation 5) and accuracy measures for all models. Binary

Cross Entropy is the loss function computed by back-propagation algorithm during training process. Cross entropy loss decreases as the predicted probability converges to the ground truth data. Cross entropy penalizes both false positive and false negative. However, it especially penalizes those false predications having a high confident score. A perfect model would have a 0 log loss. As shown in Fig 2 the balanced data produces plenty of false positives which are penalized by cross entropy. This justifies the high value of the metric regarding the balanced data. Adding up more negative examples into the balanced dataset reduces the cross entropy score across the limited imbalanced datasets. Regarding the accuracy scores, they reach their maximum when it comes to the limited imbalanced datasets. In the imbalanced data scenario, accuracy is not any more a reliable measure. As a matter of fact, machine learning algorithms are usually designed to improve accuracy by reducing the error. Thus, facing imbalanced datasets, they produce inadequate predictions, since they do not consider the class distribution. This leads to achieving high overall accuracy, while it only reflects the accuracy of the majority class. The issue with accuracy arises from a deeper problem. De facto, accuracy assigns equal intensity to all errors in classification. In the context of a predictive model, it fails to be sensitive to the probabilistic nature of classification by measuring hard class assignments trueness while learnings from data are probabilistic.

Discussion

With the thriving DNA sequencing technologies, billiards of sequences are generated everyday. It has not escape the notice of many computational biologists that deep neural networks will be a key tool to harvest the fruits from this gigantic and still increasing amount of data. One of the practical issues when applying deep CNN on genomic sequences is the imbalanced data, a well-known issue in the machine learning literature [16], [17], [15]. In the present paper, we address this problem using human TSS as a case study. Indeed, the TSS occupy only a few locations on the genome (31,037 TSS for human), which leads to extreme imbalances in dataset ($\sim 10^{-5}$ is the ratio of TSS-containing 299 bp windows to non-TSS in the human genome). This issue is usually referred to as a rare events or rare samples. Imbalance due to rare samples is representative of domains where minority class samples are very limited. In this scenario, the lack of representative data deteriorates learning process as predictive models using conventional machine learning algorithms do not accurately measure model performance and could be biased and inaccurate. To deal with this disparity we adopt a sub-sampling strategy to decrease the frequency of the majority class samples (non-TSS) improving thereby identification of the rare minority class samples (TSS). We show that this method achieves very good performance for CNN models in predicting TSS and non-TSS sequences genome-wide. It does not require knowledge of any specific TSS features. Since the convolution filters are able to automatically capture sequence motifs and other significant characteristics of biological and genomic sequences, this approach can be easily extended to identify other functional regions in any annotated genomes.

We also show that our method can be efficiently used trans-genome-wide, that is training the model in one genome and applying it to other genomes. We used mouse and human TSS as case study and apply both models on chromosome X of each organism. While the sequence of this chromosome has evolved differently in both species, many genes are homologous. The fact that we can recover TSS in mouse/human with a model trained in the other organism suggest that the machinery able to recognize TSS in both organism is largely conserved. While genome sequence conservation can be computed directly from DNA sequences, our method provides a new tool to address the conservation of the whole nuclear machinery that interprets the sequences. We expect that such a tool will be used more and more in the future both by evolutionary biologists

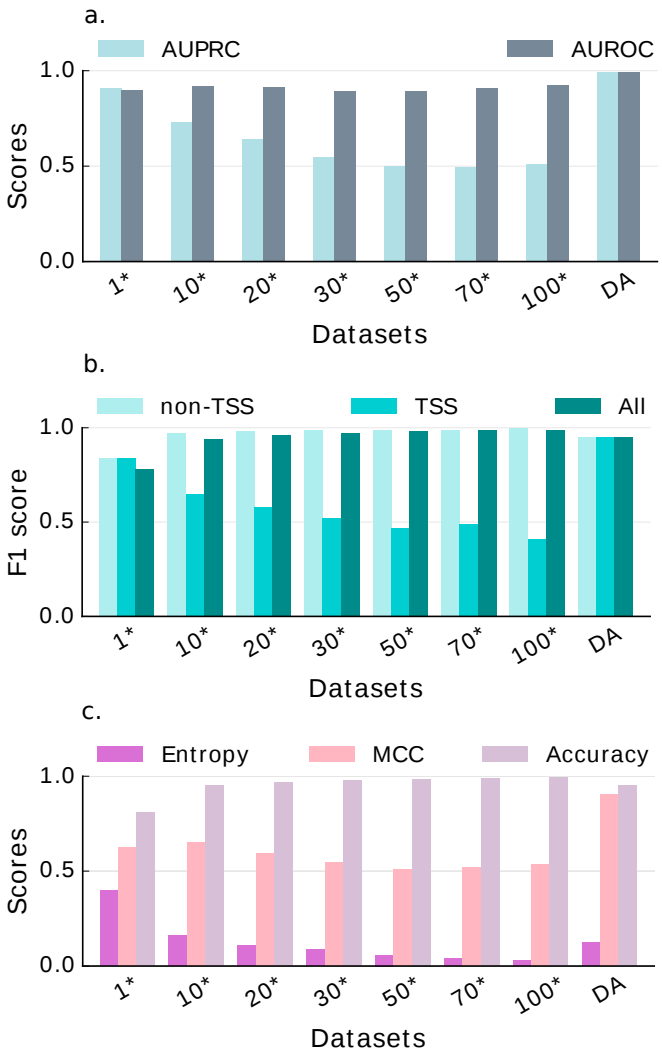


Fig 9. Results on different metrics for evaluating the performance of CNN model applied over Q^* models. The results reports the performance of the models over test sets. (a) AUPRC is Area Under Precision-Recall Curve and AUROC is Area Under an ROC Curve. (b) F1-score reports a weighted average between precision and recall per class (TSS and non-TSS). (c) Variation of binary cross entropy, MCC and accuracy measures for all Q^* models.

and by molecular biologists interested in the evolution of chromatin metabolism.

Materials and methods

Input Generation

We collect TSS positions from the reference genomes for human (hg38) and mouse (mm10) species. Genome sequences are downloaded at http://hgdownload.soe.ucsc.edu/goldenPath/hg38/bigZips/ and http://hgdownload.soe.ucsc.edu/goldenPath/mm10/bigZips/, respectively. TSS

positions over the entire human and mouse genomes are downloaded at http://epigenomegateway.wustl.edu/browser/ and http://egg.wustl.edu/d/mm10/refGene.gz, respectively.

The input generation consists of two strategies: we first generate balanced and imbalanced datasets; we then apply a data augmentation strategy to increase the examples size.

Generating balanced and imbalanced datasets

We use as positive input sequences regions of 299 bp flanking TSS, i.e. ±149 bp around the TSS, which are supposed to contain multiple sequence signals indicating the presence of a TSS to the transcription machinery of the cell. Overall, 31,037 TSS positions are extracted on both DNA strands, eventually generating the complementary sequence of the human genome (15,798 for positive strand and 15,239 for negative strand). In a similar fashion, we extract 25,698 TSS positions out of the mouse genome (12,938 for positive strand and 12,760 for negative strand). In order to generate the negative class, we perform a sub-sampling strategy on both genomes to set up a balanced dataset. To do this, we select $Q \times 31,037$ random positions 299 bp long regions flanking those random selected positions, insuring that the regions do not contain a TSS. For $Q = 1$, this strategy avoids the problem of imbalanced data, since it equalizes the ratio of negative class examples (non-TSS positions) to positive class (TSS positions). To introduce more negative examples into balanced dataset, we use different values of Q ranging from 1 to 100.

Data augmentation

A constraint of deep neural networks is the need of a large amount of training examples. In our balanced dataset, we only dispose of 62,074 (for human) and 51,396 (for mouse) training examples for both negative and positive classes. The limited samples size leads to over-fitting issues as the model learns only the restrictive rules and becomes too specific. In order to address this issue, we also opt for a data augmentation strategy to increase the training set volume. We slide a window of size 299 bo along the genome with stride 1 bp around the TSS positions, and take every sequence overlapping with the original one to obtain augmented sequences. We also augmented the number of examples from non-TSS regions using the same strategy, i.e. 1 bp sliding around positions randomly chosen in the genome. The number of training examples increases from 62,074 to about 2 million examples. The same as sub-sampling strategy, the augmented sequences from TSS regions are labeled as positive, and those from non-TSS regions are labeled as negative. For the sake of brevity, in our study the data augmentation is realized only in human.

Convolution Neural Network (CNN)

A deep convolutional neural network typically has an input shape as a tensor with dimension of $c \times b \times l$. We have implemented a CNN to implement a prediction model capable of predicting the presence of a TSS out of a DNA sequences of size 299 bp. In our model architecture, summarized in Fig 1, the input can be thought of as a tensor where c is the array size for a single nucleotide, b is the length of the input sequence and l is the depth of the tensor. The nucleotide sequences are one hot encoded forming 4 channels as A (1,0,0,0), T (0,1,0,0), G (0,0,1,0), and C (0,0,0,1). Thus, the value of c is equal to 4 and the input sequences b are of length of 299 bins with a depth l equals to 1. A single DNA sequence example is of shape $(4 \times 299 \times 1)$ in training set. The training set contains N samples of labeled-pair of form $(X^{(n)}, y^{(n)})$, for $n \in \{1, \cdots, N\}$, where $X^{(n)}$

are matrices of size $c(=4) \times b(=299)$ and $y^{(n)} \in \{0,1\}$. Each sample in the matrices $X^{(n)}$ is labeled by $y^{(n)} = 1$ when it corresponds to TSS position and $y^{(n)} = 0$ otherwise. The convolutional layers carry out a series of sliding windows operation (convolution) using k kernels each of size s to scan motifs all over positions p. The first layer kernels perform convolution on s successive input sequences positions $p \in \{1, \cdots, (b-s+1)\}$ to recognize relevant patterns (motifs) and updating themselves during the training phase. The next convolutional layers scan low-level features in the previous layers in order to model high-level features. This filtering operation generates an output feature map of size $k \times (b-s+1)$. For an input sample X of size $c \times b$. The feature map $\mathcal{M} = f_{conv}(X)$ resulting from the convolution operation is computed as follows:

$$\mathcal{M}_{p,i} = \sum_{j}^{c} \sum_{r=1}^{s} \mathcal{W}_{i,j,r} X_{p+r-1,j} + \mathcal{B}_i, \quad i \in \{1, \cdots, k\} \tag{1}$$

where $\mathcal{W}$ denotes the weights with size $(k \times c \times s)$, which are updated via a back-propagation process so that the loss decreases and $\mathcal{B}$ denotes the bias with size $(k \times 1)$. This step is followed by applying a non-linear function, here a Rectified Linear Unit (ReLU). This activation function computes $f_{ReLU}(\mathcal{M}) = max(0, \mathcal{M})$ to incorporate nonlinearity by transforming all negative values to zero. In order to reduce the input dimensionality and provide an abstract form of the representation, we apply max-pooling process with pool size m over the output of $f_{ReLU}(\mathcal{M})$. Max-pooling reduces the input parameters to a lower dimension and provides basic translation invariance to the inner representation resulting in less computational cost and promotes high-level patterns detection. Using fewer parameters avoids also over-fitting. In our model, max-pooling layer reduces a large region into a smaller representation by taking the maximum values of the range in pooling size. We apply a max-pooling layer $f_{maxpool}(\mathcal{M})$ over an input $\mathcal{M}$ of size $k \times (b-s+1)$. One of the important issues of any learning algorithm is over-fitting. In order to deal with this issue we use a regularization parameter, called dropout. In training step, to avoid over-fitting some outputs of previous layer are randomly discarded and then remaining information are fed as the inputs to the next layer.

The final step is using a fully connected layer to learn a model to map DNA sequences $X^{(n)}$ onto TSS positions $y^{(n)}$. While convolutional layers learn high-level features, the fully connected layer deals with linear and non-linear combinations of high-level features arising from the convolutional layers in order to make the final decision. Indeed, the output of the previous layer determines which features correlate the most to a particular class. In order to learn all hidden features, our network has several hidden layers and the last layer is the output layer, which through a sigmoid function ($\phi = \frac{1}{1+e^{-x}}$) generates the class scores for all input sequences. The optimization function in a supervised learning problem attempts to minimize a loss function during training phase by measuring the compatibility between a prediction and the ground truth label. The loss function computes an average over the losses for every individual sample. The loss function calculates the gap between the predicted output and the ground truth. We use binary Cross Entropy (log loss), which is commonly-used to measures the performance of a classification model whose output is a probability value between 0 and 1. It is computed as:

$$\mathcal{L} = -1/N \sum_{i=1}^{N} [y^{(n)} log(\hat{y}^{(n)}) + (1 - y^{(n)}) \times log(1 - \hat{y}^{(n)})] \tag{2}$$

where $\hat{y}^{(n)}$ is the estimated scores for the input samples $X^{(n)}$. To minimize $\mathcal{L}$ gradient descent via a back-propagation process updates all learning parameters of the model Θ, in the opposite direction:

$$\nabla\Theta = -\gamma\frac{\delta\mathcal{L}}{\delta\Theta} \tag{3}$$

where γ stands for the learning rate, which determines the size of the steps to reach a local minimum.

Hyperparameter tuning

We implement CNN using Keras library and Tensorflow [18] as back-end. For a faster training a GTX 1070 Ti GPU is used. We use Adaptive Moment Estimation (Adam) [19] to compute adaptive learning rates for each parameter. Adam optimizer is an algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments. Our CNN architecture (see Fig1) consists of three convolutional layers of size 32, 64 and 128 kernels each of shape (4×4) with sliding step 1. The output from convolutional is entered into a max-pooling layer with pooling size (1×2). After each max-pooling layer a ReLU layer is used following by a dropout with a ratio of 0.2. Finally, the output of the last pooling layer is flattened to 1D via a Flatten layer and passed through a fully connected (Dense) layer with 128 neurons. The output layer contains a single neuron. It uses a sigmoid activation function to make predictions by producing a probability output of being TSS or non-TSS. The models are trained for 150 epochs and they mostly converge early (around 30-35 epochs). This step is a k-fold cross validation, which helps tuning parameters. The network architecture is detailed in Table 1. For training a CNN model

Table 1. Network architecture of the CNN model.

Layer name	Layer shape	Output shape
Input	-	$4 \times 299 \times 1$
Conv2D	$32 \times 4 \times (4 \times 4)$	$32 \times 284 \times 1$
Max-pooling	1×2	$32 \times 142 \times 1$
Dropout	-	$32 \times 142 \times 1$
Conv2D	$64 \times 32 \times (4 \times 4)$	$64 \times 127 \times 1$
Max-pooling	1×2	$64 \times 63 \times 1$
Dropout	-	$64 \times 63 \times 1$
Conv2D	$128 \times 64 \times (4 \times 4)$	$128 \times 48 \times 1$
Max-pooling	1×2	$128 \times 24 \times 1$
Dropout	-	$128 \times 24 \times 1$
Dense	128	128
Dropout	-	128
Dense (sigmoid)	1	1

The first column depicts the different layers used consecutively in the network. The layer shape column reports shape of the convolutional kernels, the max-pooling windows and the fully connected layers. The output shape column represents the variation of layers shape at each step.

on the balanced dataset, the sample sets of total 62,074 (for human) and 51,396 (for mouse) are divided into 3 separate sets of training (70%), test (15%) and validation (15%) sets. The training set is used to find the optimal weights using the back-propagation algorithm by pairing the input with expected output. The validation set helps to tune the hyperparameters and provides information about how well the model has been trained. It returns the model performance scores for each epoch and helps to find the optimum number of hidden units or determine a stopping point for the

back-propagation algorithm to avoid over-fitting. The test set helps to assess the quality of the fully-trained model over unseen samples. As here we are facing the imbalanced data issue, during the training phase the model may reach at a local optimum predicting always non-TSS and making it hard to further improve the model. In order to deal with this issue, the model has to make a trade-off between precision and recall of the predictions during the training phase. In other words, both TSS and non-TSS classes must be regarded as equally important. To incorporate this trade-off into the training process of the imbalanced datasets, the samples belonging to both classes have to contribute to the loss with different weight distribution. This can be achieved by assigning more importance to TSS class by multiplying its samples by a CW weight:

$$CW = w \times \frac{\text{number of non-TSS}}{\text{number of TSS}}, \qquad w \in [0, 1] \tag{4}$$

when $w = 1$ more importance is attributed to recall, when $w = 0$ more importance is attributed to precision and when $w = 0.5$ both classes are regarded as equally important. We take here a value of $w = 0.5$.

To investigate the impact of different architectures of our CNN model on the network performance quality, we tune some hyperparameters. The different architectures permit to explore the effect of the kernel size s, to study the influence of the pool size m or to explore the impact of the network depth over validation set reliability. Table 2 displays the impact of varying s, m and network depth on validation set across the balanced data with regard to accuracy score. As reported in the table, performance of different architectures did not vary significantly from each other. However, the architecture related to the highest accuracy score is selected as hyperparameters of our final CNN model, see Table 1.

Table 2. Results on hyperparameters tuning.

Kernel size	Pool size	Depth	Accuracy
(4x4, 4x4, 4x4)	(1x2, 1x2, 1x2)	3	0.829
(3x4, 4x4, 2x4)	(2x2, 2x2, 2x2)	3	0.807
(2x4, 3x4, 4x4,4 x4)	(1x2, 1x3, 1x2, 1x2)	4	0.791
(1x4, 1x4, 1x4, 1x4)	(1x2, 1x2, 1x2, 1x2)	4	0.812
(3x4, 4x4, 2x4, 4x4, 4x4)	(2x2, 1x2, 1x2, 2x2, 1x2)	5	0.802
(2x4, 2x3, 2X4, 1x4, 4x4)	(1x2, 1x2, 1x2, 1x2, 1x2)	5	0.785

The impact of different combinations of kernel size s, pool size m and network depth on accuracy score over validation set of the balanced data. The first column represents different kernel sizes for 3 convolutional layers. The second column shows different pool sizes used after each convolutional layer. The third column corresponds to the depth of the network. s is capable of recognizing relevant patterns within local neighborhood and m reduces input patterns to a lower dimension by combining important representations within the region. The first row values representing the highest accuracy score are selected as hyperparameters of our CNN model.

Performance measures

Several conventional measures are used to evaluate the performance of the CNN models. Our predictive models can be thought of a binary classifier by assigning a decisive threshold over their probabilistic outputs. Traditional measures for binary classification task are precision, recall, F1-score and accuracy. We define true positives (TP) as correctly predicted TSS, false positives (FP) as TSS wrongly classified as non-TSS, false

negative (FN) as non-TSS wrongly classified as TSS and true negative (TN) as TSS wrongly classified as non-TSS.

Precision is the ratio of correctly predicted TSS to the total predicted positive observations: $Pr = \frac{TP}{TP+FP}$. Thus, the high precision relates to the low false positive rate. Recall (or sensitivity) is the ratio of correctly predicted TSS to the all observations in positive class: $Re = \frac{TP}{TP+FN}$.

F1-score is the weighted average of precision and recall, which takes equally both false positives and false negatives into account: F1-score $= 2 \times \frac{Re \times Pr)}{(Re+Pr)}$. Accuracy is the most intuitive performance measure and it is defined by a ratio of correctly predicted observations (true positives and true negatives) to the total positive and negative observations: Accuracy $= \frac{TP+TN}{TP+FP+FN+TN}$. However, accuracy is not a reliable measure to assess model performance for datasets with inequivalent distributed classes such as the unequal proportion of TSS and non-TSS samples. F1-score measure is usually more adequate than accuracy in uneven class distribution. Matthews Correlation Coefficient (MCC) is widely used in Bioinformatics as a performance metric. According to [20] MCC is more reliable than the other measures for imbalanced data. This measure takes into account the imbalance in the classes (in binary classification) and is defined as follows:

$$\text{MCC} = \frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}} \tag{5}$$

The MCC is a correlation coefficient value between -1 and $+1$, where a coefficient of $+1$ signifies an ideal prediction, -1 an inverse prediction and 0 an average random prediction.

All aforementioned measures assess the performance of the model applied on the test sets. On the one hand, they are no longer state of the art measures to evaluate the performance of the trained models when applied over new imbalanced inputs such as on the genome-wide sequence. This is due to the fact that the predictive models tend to be biased towards the majority class and hence perform poorly on the minority class. Thus, predictive models using conventional machine learning algorithms do not accurately measure model performance and could be biased and inaccurate. On the other hand, having a high performance on test set does not imply the generability of a model, since test set is only a subset of learning set and it may not be representative of a much larger data (e.g. whole genome or a full chromosome) with different properties and data distribution (see Fig 4 for more details). Thus, we have to use an adequate metric to assess how the trained models generalize over the unseen chromosomes, i.e. chromosomes 21 for human and chromosomes X for mouse.

We therefore define a custom metric called λ. To compute λ, we first compute the genome-wide standard score $\mathcal{Z}$ from the predictions X_i:

$$\mathcal{Z} = \frac{X_i - \bar{\mu}}{\sigma}, \qquad i \in \{1, \cdots, N\} \tag{6}$$

where $\bar{\mu}$ and σ stand for the predictions mean and standard deviation, respectively. X_i represents all predictions scores. We denote w as all windows selected over $\mathcal{Z}$ scores within 10 kbp regions flanking each TSS position. We then compute $\mathcal{S}_v$ the average of $\mathcal{Z}$ over w (see ig 4 (a)):

$$\mathcal{S}_v = \frac{1}{||w||} \sum_{\alpha \in w} \mathcal{Z}_v(\alpha) \tag{7}$$

where $v \in [-5000, +5000]$ are the positions within each window and $\mathcal{Z}_v(\alpha)$ is the standard score for a given window α at position v. We define the area under the curve

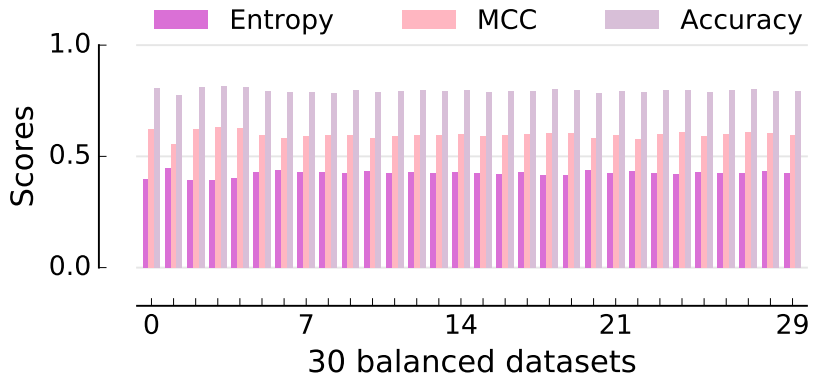
$\mathcal{S}_v$ on a region of $r = 600$ bp centered over the TSS as λ:

$$\lambda = \frac{1}{r} \sum_{v=-r/2}^{+r/2} \mathcal{S}_v \tag{8}$$

A higher value of λ corresponds to a higher signal to noise ratio around TSS regions.

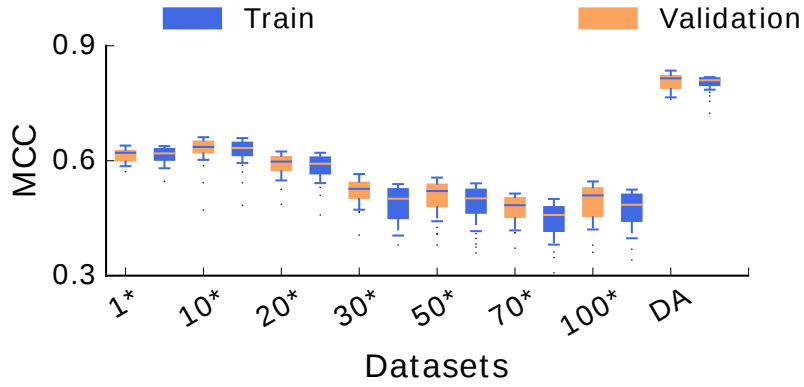
Supporting information

S1 Fig. The performance of the CNN models for test sets applied separately over 30 balanced datasets.



We generate up to 30 random balanced datasets from non-TSS regions and use the 1* model to separately train them. The metrics express almost no variation throughout the 30 models scores for test sets.

S2 Fig. The variation of MCC scores for training and validation sets within training phase.



In regards to the MCC, the test sets scores decreasing through the datasets. The same trend is observed in training and validation sets scores. The MCC metric takes into account the unequal distribution of classes within the datasets.

Acknowledgments

We would like to thank Léopold Carron for helping us with datasets. This research was supported by funding from the Agence Nationale pour la Recherche (HiResBac ANR-15-CE11-0023-03 to J.M.).

Author Contributions

GK and JM conceived and designed the methodology. GK performed the experiments. GK and ER analyzed the data. GK wrote the paper. JM revised the manuscript.

References

1. Goodfellow I, Bengio Y, Courville A. Deep Learning. MIT Press; 2016.
2. DeepLearning Biology;. `https://github.com/hussius/deeplearning-biology`.
3. Ching T, Himmelstein DS, Beaulieu-Jones BK, Kalinin AA, Do BT, Way GP, et al. Opportunities and obstacles for deep learning in biology and medicine. bioRxiv. 2018; p. 142760.
4. Min X, Chen N, Chen T, Jiang R. DeepEnhancer: Predicting enhancers by convolutional neural networks. In: Bioinformatics and Biomedicine (BIBM), 2016 IEEE International Conference on. IEEE; 2016. p. 637–644.
5. Umarov RK, Solovyev VV. Recognition of prokaryotic and eukaryotic promoters using convolutional deep learning neural networks. PloS one. 2017;12(2):e0171410.
6. Alipanahi B, Delong A, Weirauch MT, Frey BJ. Predicting the sequence specificities of DNA-and RNA-binding proteins by deep learning. Nature biotechnology. 2015;33(8):831.
7. Zhou J, Troyanskaya OG. Predicting effects of noncoding variants with deep learning–based sequence model. Nature methods. 2015;12(10):931.
8. Kelley DR, Snoek J, Rinn JL. Basset: learning the regulatory code of the accessible genome with deep convolutional neural networks. Genome research. 2016;26(7):990–999.
9. Lee D, Karchin R, Beer MA. Discriminative prediction of mammalian enhancers from DNA sequence. Genome research. 2011;21(12):2167–2180.
10. Ghandi M, Lee D, Mohammad-Noori M, Beer MA. Enhanced regulatory sequence prediction using gapped k-mer features. PLoS computational biology. 2014;10(7):e1003711.
11. Leung MK, Xiong HY, Lee LJ, Frey BJ. Deep learning of the tissue-regulated splicing code. Bioinformatics. 2014;30(12):i121–i129.
12. Picardi E, Pesole G. Computational methods for ab initio and comparative gene finding. In: Data mining techniques for the life sciences. Springer; 2010. p. 269–284.
13. McHardy AC. Finding genes in genome sequence. In: Bioinformatics. Springer; 2008. p. 163–177.

14. Sonnenburg S, Zien A, Rätsch G. ARTS: accurate recognition of transcription starts in human. Bioinformatics. 2006;22(14):e472–e480.

15. He H, Garcia EA. Learning from imbalanced data. IEEE Transactions on knowledge and data engineering. 2009;21(9):1263–1284.

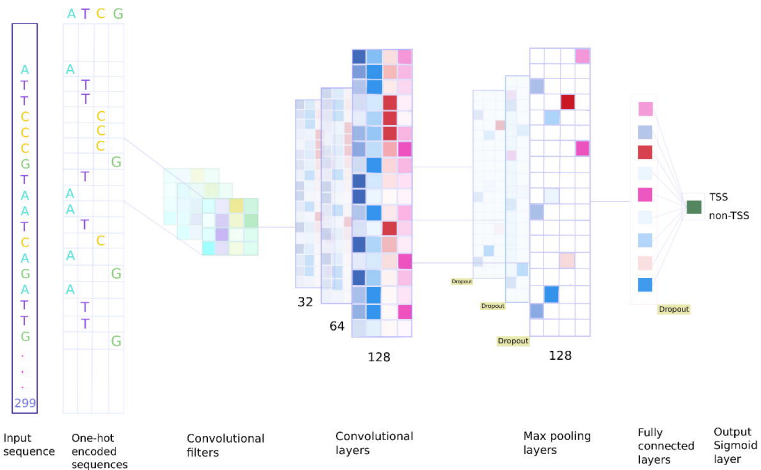
16. Chawla NV, Japkowicz N, Kotcz A. Special issue on learning from imbalanced data sets. ACM Sigkdd Explorations Newsletter. 2004;6(1):1–6.

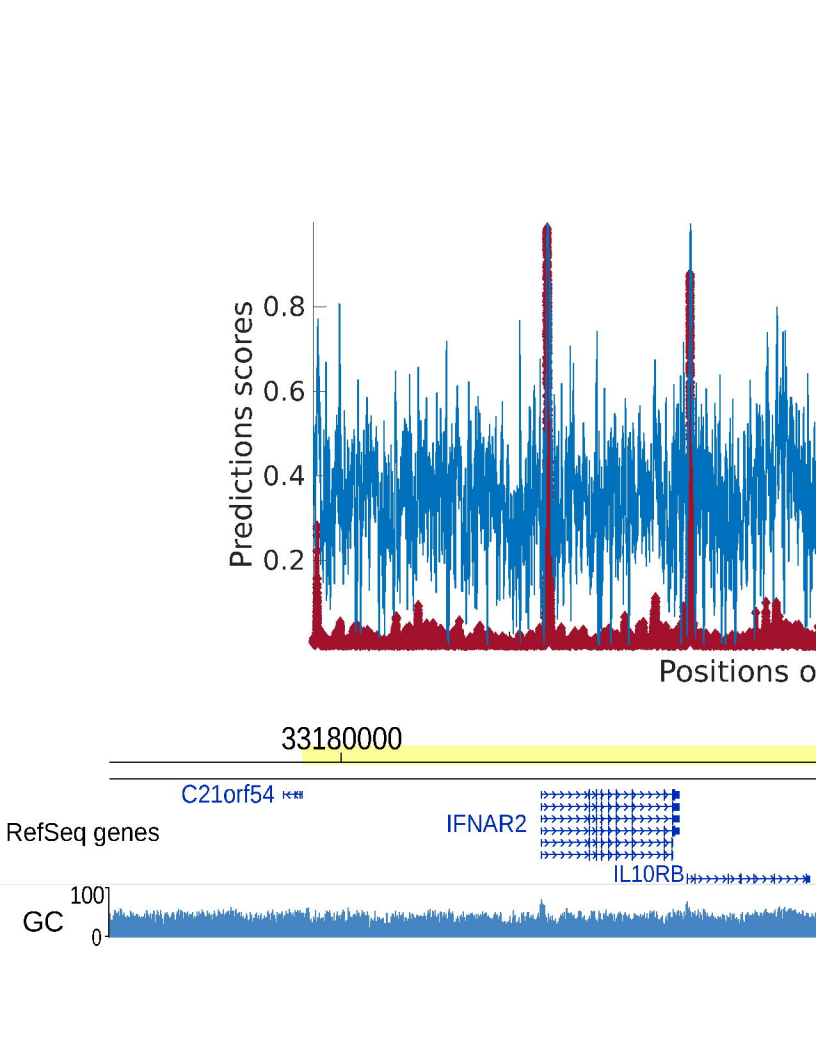
17. Batista GE, Prati RC, Monard MC. A study of the behavior of several methods for balancing machine learning training data. ACM SIGKDD explorations newsletter. 2004;6(1):20–29.

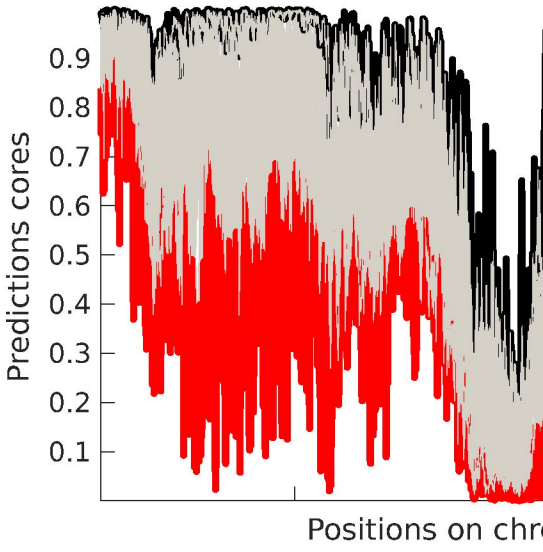
18. Abadi M, Agarwal A, Barham P, Brevdo E, Chen Z, Citro C, et al.. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems; 2015. Available from: `https://www.tensorflow.org/`.

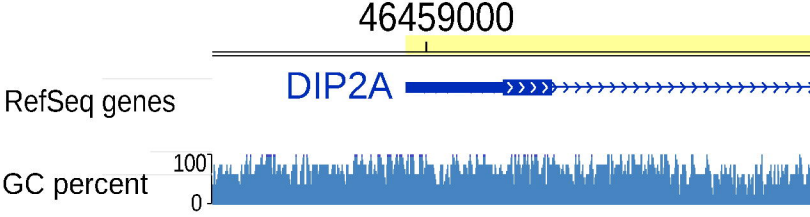
19. Kingma DP, Ba J. Adam: A method for stochastic optimization. arXiv preprint arXiv:14126980. 2014;.

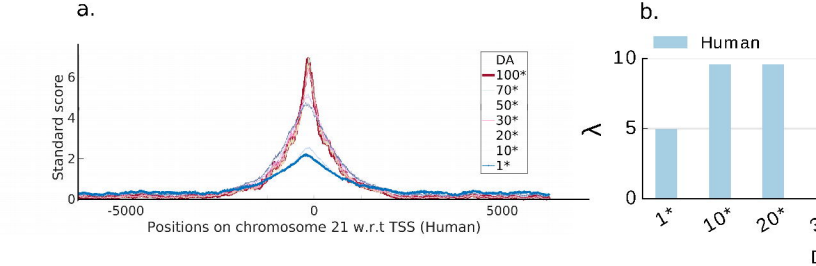
20. Boughorbel S, Jarray F, El-Anbari M. Optimal classifier for imbalanced data using Matthews Correlation Coefficient metric. PloS one. 2017;12(6):e0177678.

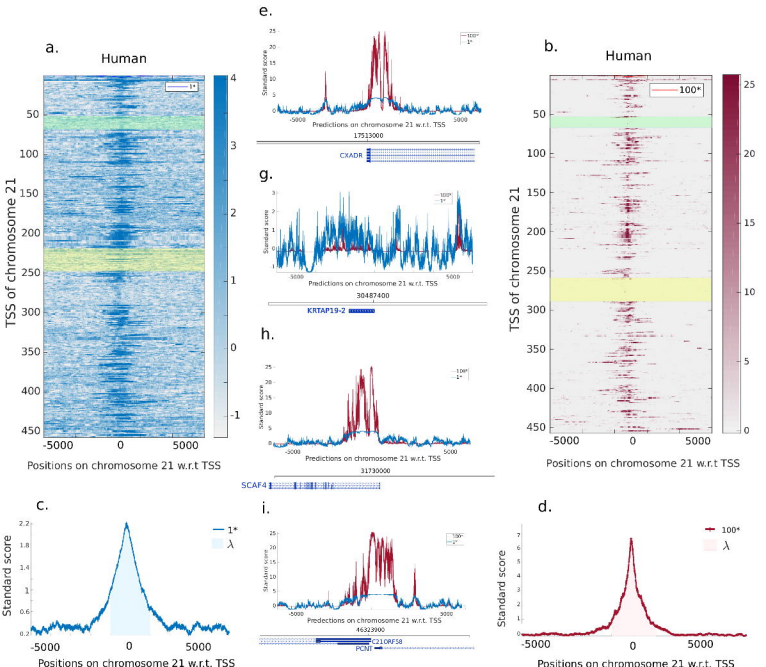

A T C G
299
32
64
128
128
Dropout
Dropout
Dropout
Dropout
TSS
non-TSS
Input sequence
One-hot encoded sequences
Convolutional filters
Convolutional layers
Max pooling layers
Fully connected layers
Output Sigmoid layer

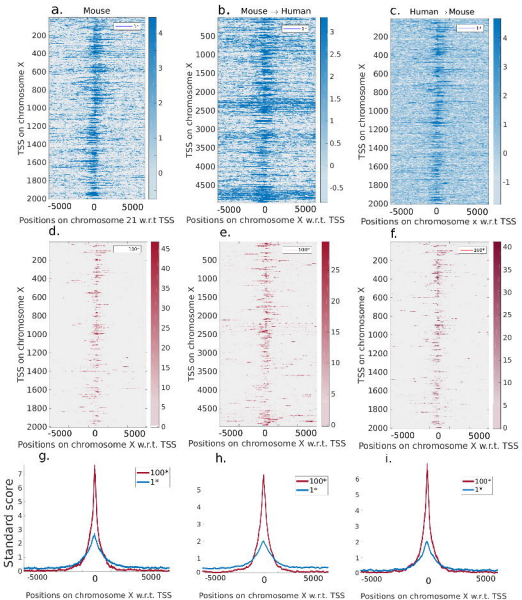

Predictions scores
0.8
0.6
0.4
0.2
Positions o
33180000
C21orf54
IFNAR2
IL10RB
RefSeq genes
GC
100
0

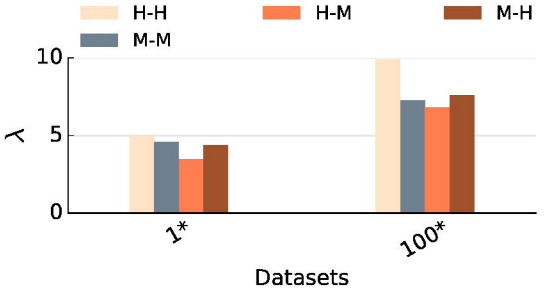

Predictions cores
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
Positions on chr

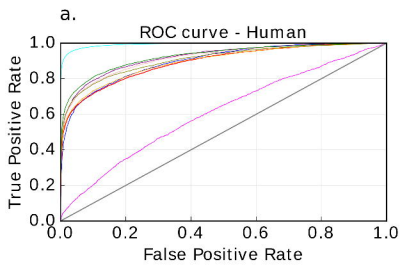

46459000
RefSeq genes
DIP2A
GC percent
100
0

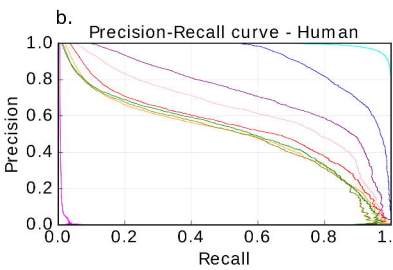

a.
b.
Standard score
Positions on chromosome 21 w.r.t TSS (Human)
-5000
0
5000
DA
100*
70*
50*
30*
20*
10*
1*
Human
λ
10
5
0
1*
10*
20*

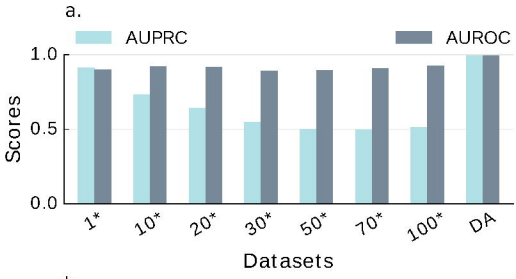

a.
Human
TSS of chromosome 21
Positions on chromosome 21 w.r.t TSS
b.
Human
TSS of chromosome 21
Positions on chromosome 21 w.r.t TSS
c.
Standard score
Positions on chromosome 21 w.r.t. TSS
d.
Standard score
Positions on chromosome 21 w.r.t. TSS
e.
Predictions on chromosome 21 w.r.t. TSS
17513000
CXADR
g.
Predictions on chromosome 21 w.r.t. TSS
30487400
KRTAP19-2
h.
Predictions on chromosome 21 w.r.t. TSS
31730000
SCAF4
i.
Predictions on chromosome 21 w.r.t. TSS
46322900
C21ORF58
PCNT

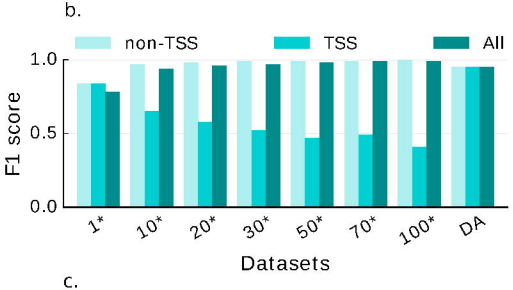

a.
Mouse
b.
Mouse → Human
c.
Human → Mouse
TSS on chromosome X
Positions on chromosome 21 w.r.t TSS
Positions on chromosome X w.r.t. TSS
Positions on chromosome x w.r.t TSS
d.
e.
f.
g.
h.
i.
Standard score
100*
1*

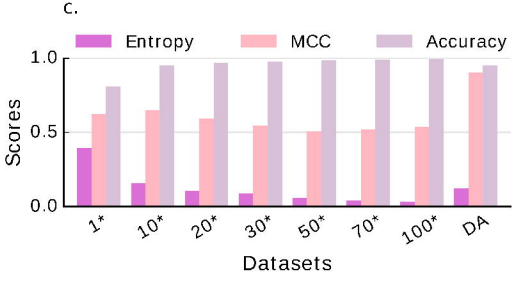

H-H
H-M
M-H
M-M
10
5
0
λ
1*
100*
Datasets

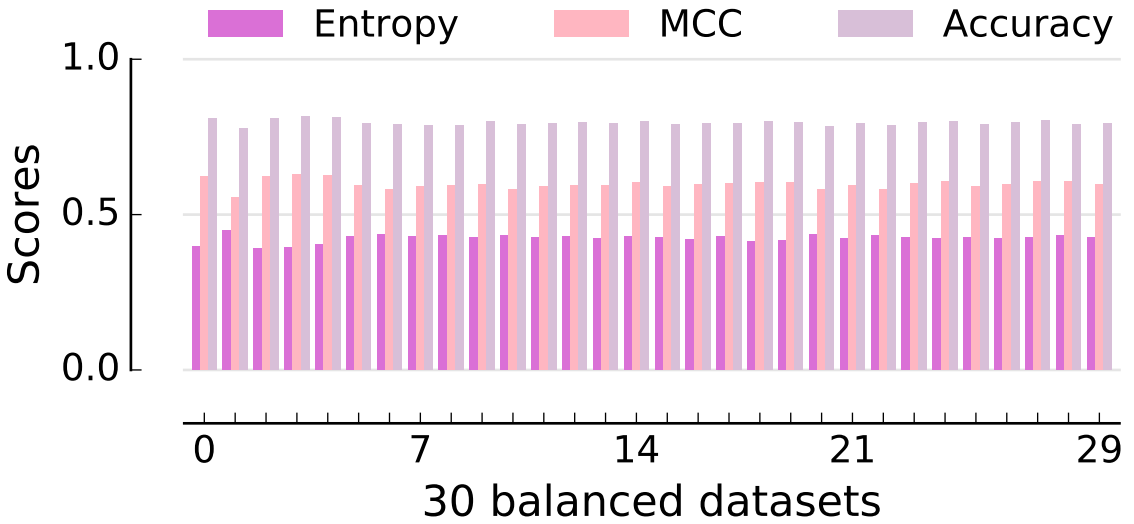

a.
ROC curve - Human
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0

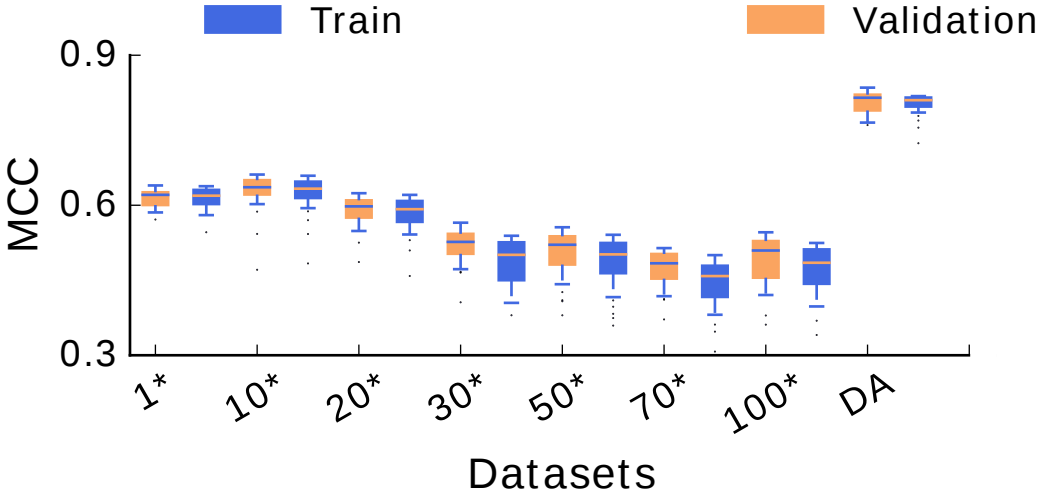

b.
Precision-Recall curve - Human
Precision
Recall
0.0
0.2
0.4
0.6
0.8
1.0


a.
AUPRC
AUROC
1.0
0.5
0.0
Scores
1*
10*
20*
30*
50*
70*
100*
DA
Datasets


b.
non-TSS
TSS
All
1.0
0.5
0.0
F1 score
1*
10*
20*
30*
50*
70*
100*
DA
Datasets


c.
Entropy
MCC
Accuracy
1.0
0.5
0.0
Scores
1*
10*
20*
30*
50*
70*
100*
DA
Datasets


Entropy
MCC
Accuracy
1.0
0.5
0.0
Scores
0
7
14
21
29
30 balanced datasets


Train
Validation
0.9
0.6
0.3
MCC
1*
10*
20*
30*
50*
70*
100*
DA
Datasets