Tecnun Universidad de Navarra

Proyecto Fin de Máster

INGENIERIA BIOMEDICA

DEEPSF: A NOVEL DEEP NEURAL NETWORK TO UNVEIL TUMOR TRANSCRIPTOME

Pº Manuel Lardizabal, 13. 20018 Donostia-San Sebastián, Gipuzkoa

Tel. 943 219 877 · Fax 943 311 442 · www.tecnun.es

deepSF: A Novel Deep Neural Network to Unveil Tumor Transcriptome

PROYECTO

presentado para optar

al Título de Máster en Ingeniería Biomédica por

Joseba Sancho Zamora

bajo la supervisión de

Phd. Fernando Carazo

Donostia-San Sebastián, julio de 2022

Tecnun Universidad de Navarra | ESCUELA DE INGENIERÍA INGENIARITZA ESKOLA SCHOOL OF ENGINEERING

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF EQUATIONS

1. THEORETICAL BACKGROUND

Nucleotides are precursors to molecules such as RNA and DNA. A single nucleotide is formed by a sugar, a phosphate group, and a nitrogenous base. The DNA molecule, in turn, is made up of two long chains of nucleotides linked together, forming a double helix, by means of hydrogen bonds between the bases of both chains.

DNA contains in its base sequence the genetic information necessary for protein synthesis. This macromolecule is packed inside the cell nucleus thanks to the chromatin that allows DNA segments to wrap around proteins, called histones, forming octamers. With this organization it is possible to fit much more information into a very small space, but this structure has to be flexible to allow gene transcription. To translate the information from DNA into proteins, first nuclear DNA must be transcribed into messenger RNA (mRNA). The DNA strands open, a polymerase binds to the strand, and the complementary RNA strand is synthesized.

Immature mRNA is contained by exons and introns, but only the exons contain protein-coding sequences. For this reason, when the mRNA leaves the nucleus and reaches the cytoplasm, RNA maturation occurs. This genetic process known as splicing consists of cutting and pasting different parts of the RNA, obtaining different isoforms of mRNA from a primary transcript of RNA. Transcripts from the same gene can be translated in several proteins that can have different or even opposite functions.

Contrary to what might be thought, the expression of a protein is not directly related to the expression of the gene or pre-mRNA, but is proportional to the expression of the transcript that encodes it. A gene with the same expression can give rise to two different proteins.

In turn, the expression of the transcript depends on the expression of the gene (pre-mRNA) and alternative splicing. That is, if the gene is not expressed, neither will its respective transcripts. But if a gene has an expression of 5tpm, it does not mean that all its transcripts are expressed equally, but rather that, depending on the splicing, some will be expressed more and others less.

Figure 1. The protein transcriptome genome relation of dependency.

Alternative splicing is a process that can be modeled by looking at how splicing factors (SFs) are expressed. SFs are RNA-binding proteins (RBPs) that are responsible for regulating splicing. Changes in expression or mutations in SFs can also lead to changes in splicing and result in an unwanted gain or loss of protein function.

Splicing can also vary depending on cell tissue context or pathological conditions such as cancer. Because this process allows a wide variety of transcripts to be created and proteins to be synthesized from a smaller number of genes, characteristics of tumors such as cell immortality or resistance to the immune

system, etc., may be related to aberrant splicing of the genes. This is why knowing how this genetic process is regulated is very important to treat certain diseases (Fernando Carazo, 2019).

The main objective of this project is to build a deep neural network (DNN) that models how the transcriptome of cancer related genes is expressed from the splicing factors and the genome. An autoencoder was also trained and added to the final model architecture to study the cellular context of protein coding genes in different tumor tissues. Finally, we tried to identify which splicing factors regulate which genes by interpreting the parameters of the final model with a tool called Deep Learning of Features (DeepLIFT).

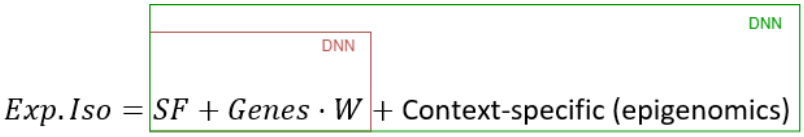


Figure 2. The Deep Neural Networks developed in this project.

Deep learning models has already been used to identify gene expression level (Lirong Zhang, 2022), discover disease-relevant modules of genes (Sanjiv K. Dwivedi, 2020), inferring transcription factor (TF) binding sites (Peter K.Koo, 2020), predict TFs from protein sequences (Gi Bae Kim, 2020) or predict transcriptomic profiles from TF expression (Rasmus Magnusson, 2022). However, the use of the gene expression of known SFs to unveil the transcriptome regulation is a novel approach.

2. STATE-OF-THE-ART

2.1. DEEP NEURAL NETWORKS

During the model development different architectures have been tested, using different Deep Learning libraries such as Keras and Pytorch. The details of the processing and training will be discussed in the materials and methods section, but the final architectures of the models developed using Pytorch are shown below:

2.1.1. DeepSF2hidden with gene expression

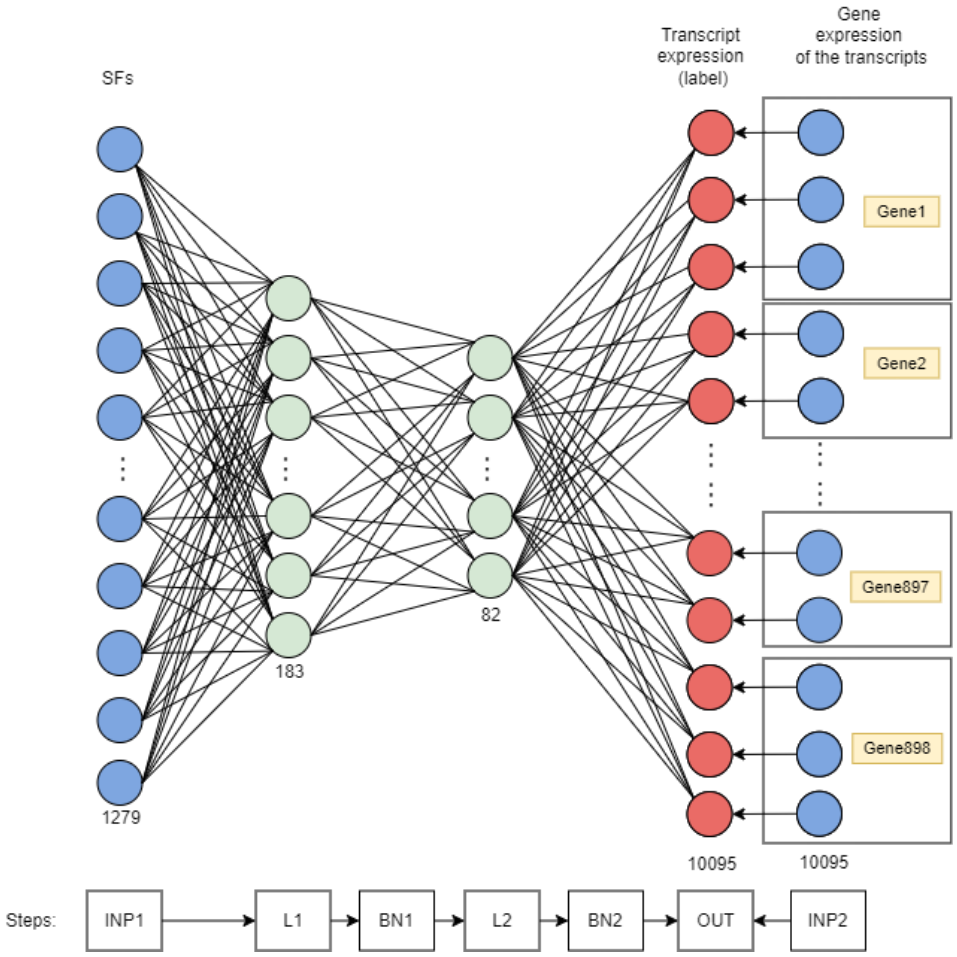


Figure 3. Architecture of the model with two hidden layers with the addition of the gene expression for each isoform developed with PyTorch.

This model has two inputs colored in blue: the expression of the SFs that are the main input of the model and the gene expression of each one of the isoforms. Between the input with the SFs expression and the output there are 2 hidden layers (L1 and L2) with 183 nodes and 82 nodes respectively. All nodes are densely connected by linear functions with ReLU activation function. After L1 and L2, two batch normalizations were performed and the gene expression for each isoform was added to the last linear function between L2 and the output.

2.1.2. DeepAE

In (Sanjiv K. Dwivedi, 2020) the researchers were able to discover disease-relevant modules of genes by training a Deep autoencoder from large transcriptional data.

An autoencoder is a neural network (NN) model that is used to capture the most relevant information from the input data (encodings). The input data and the output to be predicted is the same, which is why it is considered unsupervised learning. Since the prediction is not compared to a label but to the input data itself. In the end, what is obtained with this NN is a representation of the input data in a lower dimension (Bandyopadhyay, 2022).

In our case, in order to obtain a model capable of learning particular expression characteristics of each tumor tissue, an autoencoder was trained based on the [paper] architecture with 3 hidden layers and 512 nodes per layer (with which they managed to capture around 95% of the variance of the gene expression). To do so the expression of 19,594 protein coding genes was used as an input. Two batch normalizations were also introduced in the autoencoder architecture.

The trained model with its weights was saved to be used as transfer learning when designing the final model.

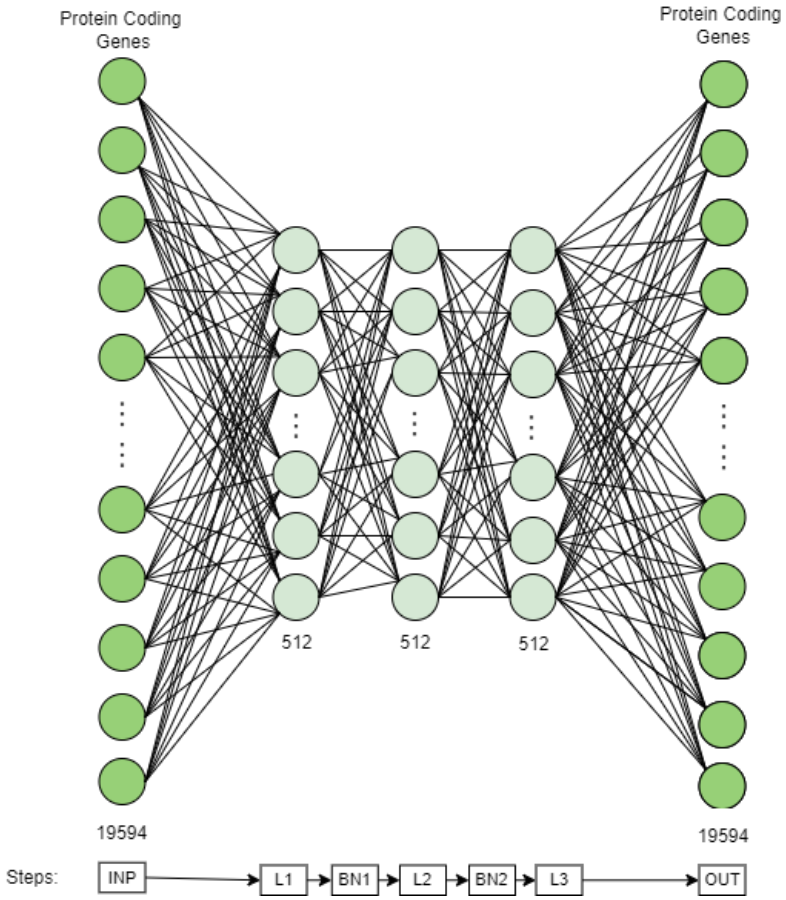


Figure 4. Architecture of the autoencoder with three hidden layers using the protein coding genes expression developed with PyTorch based on (Sanjiv K. Dwivedi, 2020).

2.1.3. DeepSF&AEensemble

Finally, a model was developed that joined the 2 hidden layers of the first model with the knowledge acquired in the training of the autoencoder. To achieve this, the last layer of the autoencoder was removed and it was also connected to the last linear layer of the model.

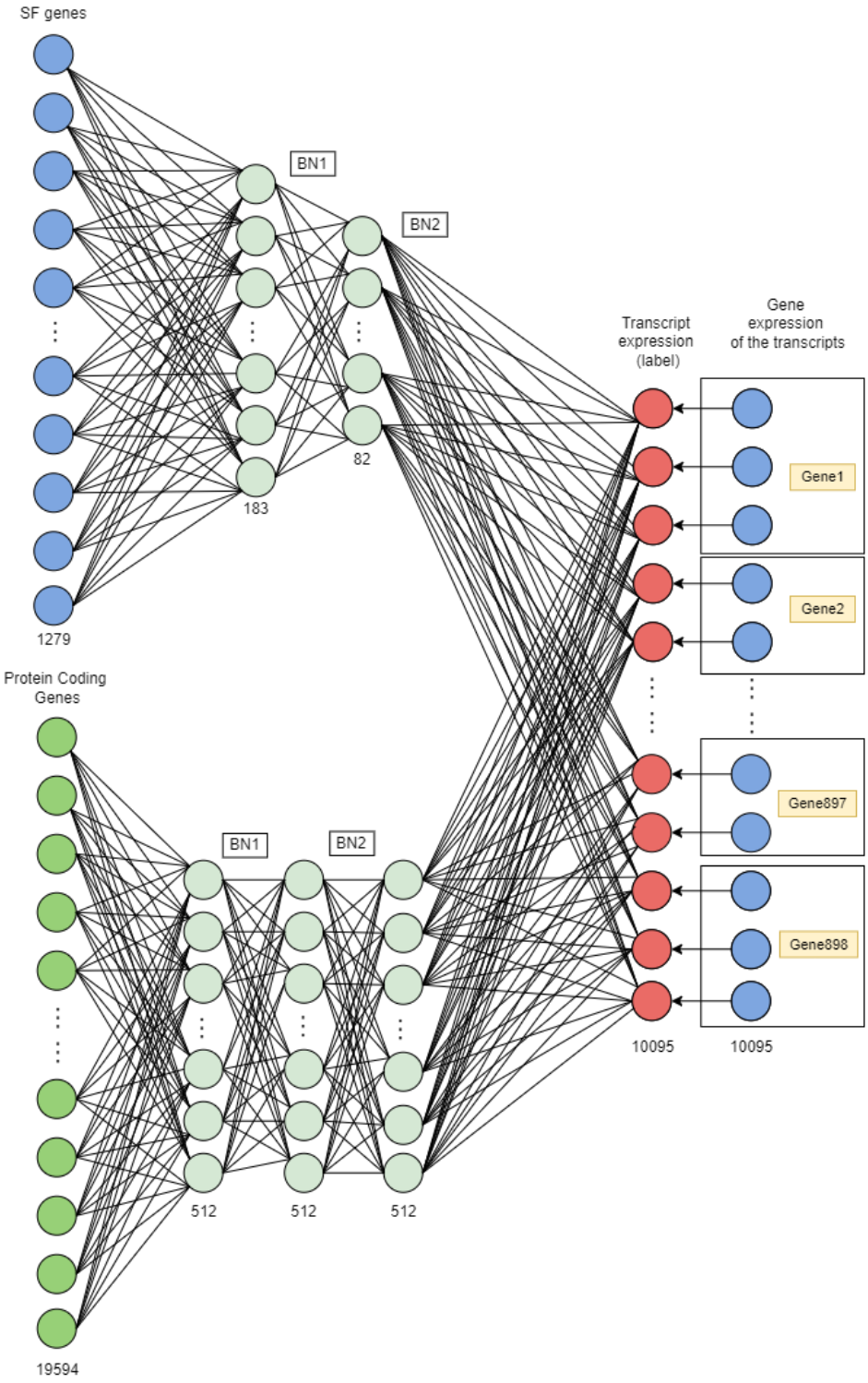


Figure 5. Architecture of the DeepSF&AEensemble model developed with PyTorch.

The model inputs referred to the SFs and gene expression are colored in blue and the expression of the protein coding genes, which are the input for a previously trained autoencoder based in *(Sanjiv K. Dwivedi, 2020)*, are colored in green. The target to predict, i.e, the expression of the transcripts is colored in red.

2.2. LOSS FUNCTIONS

Different loss functions have been used to train and evaluate the performance of the models created, comparing the theoretical expression values of the isoforms with those predicted by the model. Among the loss functions used in this project we find the mean absolute error (MAE), the mean square error (MSE) and the root mean squared error (RMSE).

The MAE measures the average of the absolute difference between the prediction and the theoretical value, giving the same relative weight to all errors. The mean squared error (MSE) is a quadratic function that also calculates the mean magnitude of the error. However, these errors are squared before calculating the average. This loss function gives a relatively high weight to large errors, which can be interesting when we are not interested in having large error outliers in our predictions. The two loss functions are indifferent to the direction of the error and the goal is that it must be as small as possible. The root mean squared error is the error obtained by the square root of MSE.

The RMSE goes up with the increase in the variance of the error frequency distribution of error magnitudes and not with the increase in the variance of the error itself. In addition, the value of MSE will always be greater than or at least equal to the value of MAE. Moreover, when more samples are added the value of RMSE tends to increase compared to MAE, which can be conflicting when comparing results obtained with different sample sizes (JJ, 2016).

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Equation 1. Mathematical expression of the Mean Absolute Error.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Equation 2. Mathematical expression of the Mean Squared Error.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

Equation 3. Mathematical expression of the Root Mean Squared Error.

2.3. OPTIMIZERS

Deep Learning optimizers are algorithms that allow us to minimize the loss function by updating the model parameters for each batch during backpropagation. Again, during the design part of the model, different algorithms have been tested to see with which we could obtain the best results, and they will be explained in the following lines:

2.3.1. Stochastic gradient descent (SGD)

Stochastic gradient descent is the most basic method of neural network optimization. With this algorithm, each model parameter is updated by adding to each one the result of the gradient of the loss function with respect to the parameter, multiplied by a learning rate. This last hyperparameter tells us how big is the step we are going to do in the learning.

$$\theta_{t+1} = \theta_t - \gamma \frac{\partial J}{\partial \theta}$$

Equation 4. Stochastic Gradient Descent (SGD) formula

2.3.2. Average Stochastic Gradient Descent (ASGD)

With this optimizer, once the gradient descent algorithm has been applied to the different parameters, for each iteration the average of all of them is done and this is the final result introduced into the model. It is an useful technique when data is noisy .

$$\theta_{t+1} = \frac{1}{N} \sum_{t=1}^{N} \theta_t$$

Equation 5. Average Stochastic Gradient Descent (ASGD) formula

2.3.3. SGD with momentum

One way to reduce the oscillations when looking for an appropriate minimum of the loss function during training with SGD optimizer and converge more quickly is by using momentum (β). In this case, for each iteration, instead of taking only the gradient at the current point, an exponential weighted average of the previous gradients (b_t) is performed. For each iteration the importance of the gradients of the previous iterations is reduced exponentially. A widely used and robust value of β is equal to 0.9 (Cantoral, 2021).

The number of previous gradients taken into account to perform the update can be approximated using the following mathematical expression:

$$\frac{1}{1-\beta}$$

Equation 6. Formula to calculate the number of previous gradients to be considered when updating the parameters using momentum.

Thus, at a higher β value we are taking into account a greater number of previous gradients. Therefore, if a beta value equal to 0.5 has been used, the number of iterations that will be taken into account will be two, with a beta of 0.7 around three, and with a beta of 0.9, the previous iterations considered will be ten. These three values for beta were tried during model designing.

Next, a pseudocode is presented explaining how the model parameters have been updated for each batch using SGD with momentum (PyTorch, s.f.):

$$
\begin{aligned}
&\boldsymbol{input}: \gamma(\mathrm{lr}), \theta(\mathrm{params}), \mathrm{f}(\theta)(\mathrm{objective}), \lambda(\mathrm{weight\ decay}), \beta(\mathrm{momentum}) \\
&\boldsymbol{for}\ t = 1\ \boldsymbol{to}\ num\ iterations\ \boldsymbol{do} \\
&\quad g_t \leftarrow \nabla_\theta f_t(\theta_{t-1}) \\
&\quad \boldsymbol{if}\ \lambda \neq 0 \\
&\quad\quad g_t \leftarrow g_t + \lambda\theta_{t-1} \\
&\quad \boldsymbol{if}\ \beta \neq 0 \\
&\quad\quad \boldsymbol{if}\ t > 1 \\
&\quad\quad\quad b_t \leftarrow \beta b_{t-1} + (1-\beta) g_t \\
&\quad\quad \boldsymbol{else} \\
&\quad\quad\quad b_t \leftarrow g_t \\
&\quad \theta_t \leftarrow \theta_{t-1} - \gamma b_t \\
&\boldsymbol{return}\ \theta_t
\end{aligned}
$$

Equation 7. Pseudocode that explains how the parameters are updated with SGD with momentum.

In the first iteration, b_t, which is the weighted average of the previous values of θ, is initialized to the value of the gradient. Then for each t we calculate the gradients of the parameters, the b_t and we update the parameters subtracting from the weight of the previous iteration the b_t by the learning rate. It is possible, as with all the different optimizers with PyTorch, to introduce a weight decay to the gradient as a method of regularizing the weights. But this is not introduced in our models optimized with SGD with momentum .

2.3.4. Adaptive Gradient (AdaGrad)

The Adaptive Gradient (AdaGrad) algorithm (PyTorch, s.f.) updates the parameters according to their importance through the use of the $state\_sum$ variable, which iteratively adds the square of the gradient of each model parameter. This term is added to the denominator of the final expression that updates the weights. The parameters of the previous iteration are subtracted by the value of the gradient divided by the root of the $state\_sum$ times the learning rate.

By taking the square of the gradient, weights that infrequently change, and consequently have a smaller gradient, will undergo larger updates, and weights with a larger gradient or that change more often will undergo smaller updates.

$\boldsymbol{input}$: γ(lr), θ(params), f(θ)(objective), λ(weight decay)
$\boldsymbol{initialize}$: $state\_sum_0 \leftarrow 0$

$\boldsymbol{for}\ t = 1\ \boldsymbol{to}\ num\ iterations\ \boldsymbol{do}$
  $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$
  $\boldsymbol{if}\ \lambda \neq 0$
    $g_t \leftarrow g_t + \lambda\theta_{t-1}$
  $state_{sum_t} \leftarrow state_{sum_{t-1}} + g_t^2$
  $\theta_t \leftarrow \theta_{t-1} - \gamma \frac{g_t}{\sqrt{state_{sum_t}} + \epsilon}$

$\boldsymbol{return}\ \theta_t$

Equation 8. Pseudocode that explains how the parameters are updated with AdaGrad algorithm.

A significant limitation of this method is that the sum of the squares of the gradient always increases during training, which leads to a point where the learning rate is so low that it cannot continue learning. The ϵ is a hyperparameter with a very small value that is added to the denominator to avoid divisions by 0. To solve these limitations, Adadelta and RMSprop algorithms were developed.

2.3.5. Adadelta

This algorithm, based on the AdaGrad concept, seeks to reduce the monotonic decay of the learning rate caused by the accumulation of the sum of squared gradients by performing a decaying average of only a certain number of previous gradients (Chablani, 2017) (PyTorch, s.f.).

$\boldsymbol{input}$: γ(lr), θ(params), f(θ)(objective), ρ (decay), λ(weight decay)
$\boldsymbol{initialize}$: $v_0 \leftarrow 0\ (square\ average), u_0 \leftarrow 0\ (accumulate\ variables)$

$\boldsymbol{for}\ t = 1\ \boldsymbol{to}\ num\ iterations\ \boldsymbol{do}$
  $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$
  $\boldsymbol{if}\ \lambda \neq 0$
    $g_t \leftarrow g_t + \lambda\theta_{t-1}$
  $v_t \leftarrow \rho v_{t-1} + (1-\rho) g_t^2$
  $\Delta_{x_t} \leftarrow \frac{\sqrt{u_{t-1} + \epsilon}}{\sqrt{v_t + \epsilon}} g_t$
  $u_t \leftarrow \rho u_{t-1} + (1-\rho)\Delta_{x_t}^2$
  $\theta_t \leftarrow \theta_{t-1} - \gamma\Delta_{x_t}$

$\boldsymbol{return}\ \theta_t$

Equation 9. Pseudocode that explains how the parameters are updated with Adadelta algorithm.

2.3.6. RMSprop

The RMSprop algorithm seeks to correct the limitations of AdaGrad using an exponential weighted average of the square of the gradients (Pytorch, s.f.).

$\boldsymbol{input}$: α (alpha), γ(lr), θ(params), f(θ)(objective), λ(weight decay)
$\boldsymbol{initialize}$: $v_0 \leftarrow 0$ $(square\ average)$

$\boldsymbol{for}\ t = 1\ \boldsymbol{to}\ num\ iterations\ \boldsymbol{do}$
  $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$
  $\boldsymbol{if}\ \lambda \neq 0$
    $g_t \leftarrow g_t + \lambda\theta_{t-1}$
  $v_t \leftarrow \alpha v_{t-1} + (1-\alpha) g_t^2$
  $\theta_t \leftarrow \theta_{t-1} - \gamma \frac{g_t}{\sqrt{v_t} + \epsilon}$

$\boldsymbol{return}\ \theta_t$

Equation 10. Pseudocode that explains how the parameters are updated with RMSprop algorithm.

Where α is a smoothing constant with a default value of 0.99 and has a similar function to momentum in SGD.

2.3.7. Adaptive Moment Estimation (Adam)

Adaptive moment estimation (Adam) is an optimization algorithm (Adam, s.f.) that combines the techniques of SGD with momentum and RMSprop. Where m_t refers to the SGD moment (β_1) and therefore is an exponential weighted average of the previous gradients and v_t refers to the moment of the RMSprop (β_2) and therefore is also an exponential weighted average, but of the square of the gradients. The v_t is used to scale the learning rate, computing individual learning rates for different parameters.

$\boldsymbol{input}$: γ(lr), β_1, β_2 (betas), θ(params), f(θ)(objective), λ(weight decay)
$\boldsymbol{initialize}$: $m_0 \leftarrow 0$ $(first\ moment - SGD\ with\ momentum)$, $v_0 \leftarrow 0$ $(second\ moment - RMSprop\ moment)$

$\boldsymbol{for}\ t = 1\ \boldsymbol{to}\ num\ iterations\ \boldsymbol{do}$
  $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$
  $\boldsymbol{if}\ \lambda \neq 0$
    $g_t \leftarrow g_t + \lambda\theta_{t-1}$
  $m_t \leftarrow \beta_1 m_{t-1} + (1-\beta_1) g_t$
  $v_t \leftarrow \beta_2 v_{t-1} + (1-\beta_2) g_t^2$
  $\widehat{m_t} \leftarrow m_t / (1-\beta_1^t)$
  $\widehat{v_t} \leftarrow v_t / (1-\beta_2^t)$
  $\theta_t \leftarrow \theta_{t-1} - \gamma \frac{\widehat{m_t}}{\sqrt{\widehat{v_t}} + \epsilon}$

$\boldsymbol{return}\ \theta_t$

Equation 11. Pseudocode that explains how the parameters are updated with Adam algorithm.

By default, the values of β_1 and β_2 are 0.9 and 0.999, respectively; and a value of $1 \cdot 10^{-9}$ of the hyperparameter ε gives numerical stability. The variables $\widehat{m_t}$ y $\widehat{v_t}$ are calculated to speed up the process in the first few iterations.

2.3.8. AdamW

The difference between this method (PyTorch, s.f.) (Hutter, 2019) and the traditional Adam method is that a weight decay function is added directly in the main function that updates the weights and not in the gradient of the weights. As it has been mentioned before, this is a regularization technique than prevents overfitting, reducing the values of the weights. The default value that we used in our model was $1 \cdot 10^{-2}$.

$\boldsymbol{input}: \gamma(\text{lr}), \beta_1, \beta_2\ (\text{betas}), \theta(\text{params}), f(\theta)(\text{objective}), \lambda(\text{weight decay})$
$\boldsymbol{initialize}: m_0 \leftarrow 0\ (first\ moment - SGD\ with\ momentum), v_0 \leftarrow 0\ (second\ moment - RMSprop\ moment)$

$\boldsymbol{for}\ t = 1\ \boldsymbol{to}\ num\ iterations\ \boldsymbol{do}$

$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$
$\theta_t \leftarrow \theta_{t-1} - \gamma\lambda\theta_{t-1}$
$m_t \leftarrow \beta_1 m_{t-1} + (1-\beta_1) g_t$
$v_t \leftarrow \beta_2 v_{t-1} + (1-\beta_2) g_t^2$
$\widehat{m_t} \leftarrow m_t / (1-\beta_1^t)$
$\widehat{v_t} \leftarrow v_t / (1-\beta_2^t)$
$\theta_t \leftarrow \theta_t - \gamma \dfrac{\widehat{m_t}}{\sqrt{\widehat{v_t}} + \epsilon}$

$\boldsymbol{return}\ \theta_t$

Equation 12. Pseudocode that explains how the parameters are updated with AdamW algorithm.

2.4. WEIGHT AND BIASES

Throughout the model development phase, the Weights & Biases (wandb) (Weights & Biases, s.f.) platform has been used to track our experiments developed in Pytorch, save versions of the models with their trained parameters, and visualize interactively the training results (value of the metrics, artifacts and plots) as well as the running settings. This platform also allows collaborative reporting and creating sweeps for the optimization of hyperparameters.

In the configuration class of the models, a boolean variable (if_wandb) allows, when the user wishes so, to upload all the training information to wandb.

Thus, during the training of a model with our methodology, if if_wandb is equal to true, the login is made in wandb and a run is started, assigning in which project it is in and which is the configuration of the model. Regarding the gradients and topology of the model, logs of all the gradients were made to wandb for every 10 batches. In addition to that, for each epoch the MSE error in the training and in the validation was logged. After training, the model parameters were saved in a dictionary using the wandb state_dict() function. Different plots for the evaluation

of the models and tables were also uploaded to the platform using wandb.log(dict).

2.5. Deep Learning of Features (DEEPLIFT)

Identifying which are the SFs capable of regulating the genome-transcriptome expression is one of the objectives of this project. To this end, our hypothesis was that we could identify which SFs regulate certain genes from the parameters of a DNN trained to predict the expression of the transcriptome from the gene expression of SFs.

To overcome the Black box barrier that DNNs impose, the Deep Learning of Features (DeepLIFT) method was applied. This algorithm is able to determine the importance of a specific input with respect to an output by comparing the difference between the input values and a reference baseline.

DeepLIFT calculates the contribution of each feature of an input of interest to the target node by backpropagation. And, unlike those methods based on a forward propagation of the perturbation, which are also computationally slower, it does not underestimate the importance of features that have saturated their contribution to the output (Avanti Shrikumar, Learning Important Features Through Propagating Activation Differences, 2019).

3. MATERIALS AND METHODS

3.1. WORKFLOW OF RAW DATA PROCESSING

3.1.1. DeepSF2hidden with gene expression

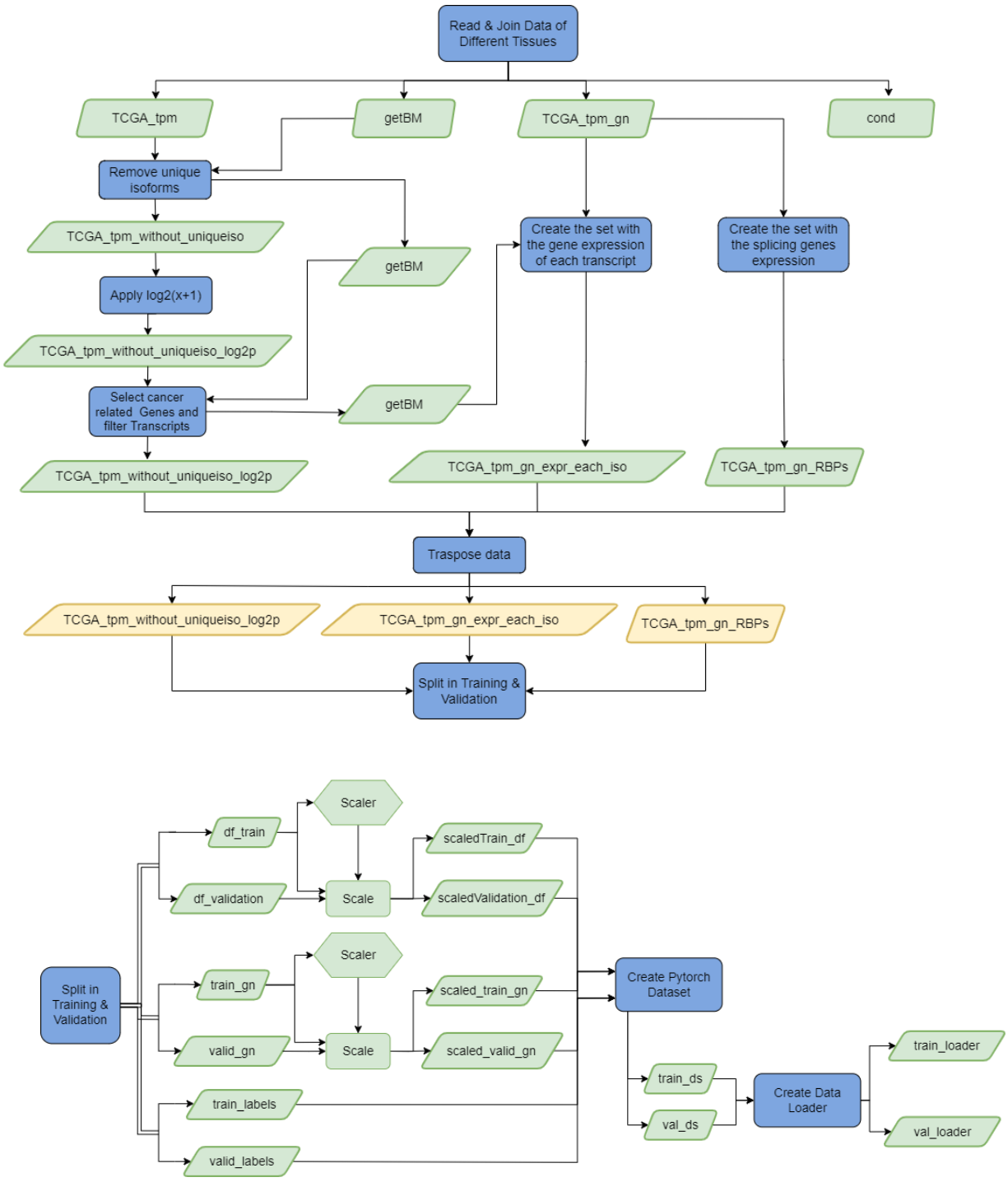


Figure 6. Pre-processing workflow to prepare the data to train the DeepSF2hidden model with Pytorch.

The workflow shown on **Figure** 6 includes the following steps:

- the original dataset preparations: read and join different cancer types,
    - TCGA_tpm dataset contains the isoform expression data
    - TCGA_tpm_gn dataset contains the gene expression data
    - *TCGA_tpm_gn_RBPs* contains the expression of the splicing factor genes (SFs).
    - getBM dataset contains the information that relates each one of the isoforms with its respective gene
    - cond dataset indicates the condition of the patients
- removing the isoforms of the genes that have just one transcript from the isoform expression dataset,
- applying a log2 transformation of the isoform expression dataset,
- filtering of the getBM dataset, to just keep the isoforms of interest,
- filtering of the isoform expression dataset with only the desired genes,
- creating a dataset with the gene expression of each of the transcripts,
- transposing different datasets,
- splitting data in training and validation,
- scaling with Standard Scaler of the dataset with expression of the SFs and gene expression,
- creating the Pytorch dataset and the data loader.

3.1.2. DeepAE

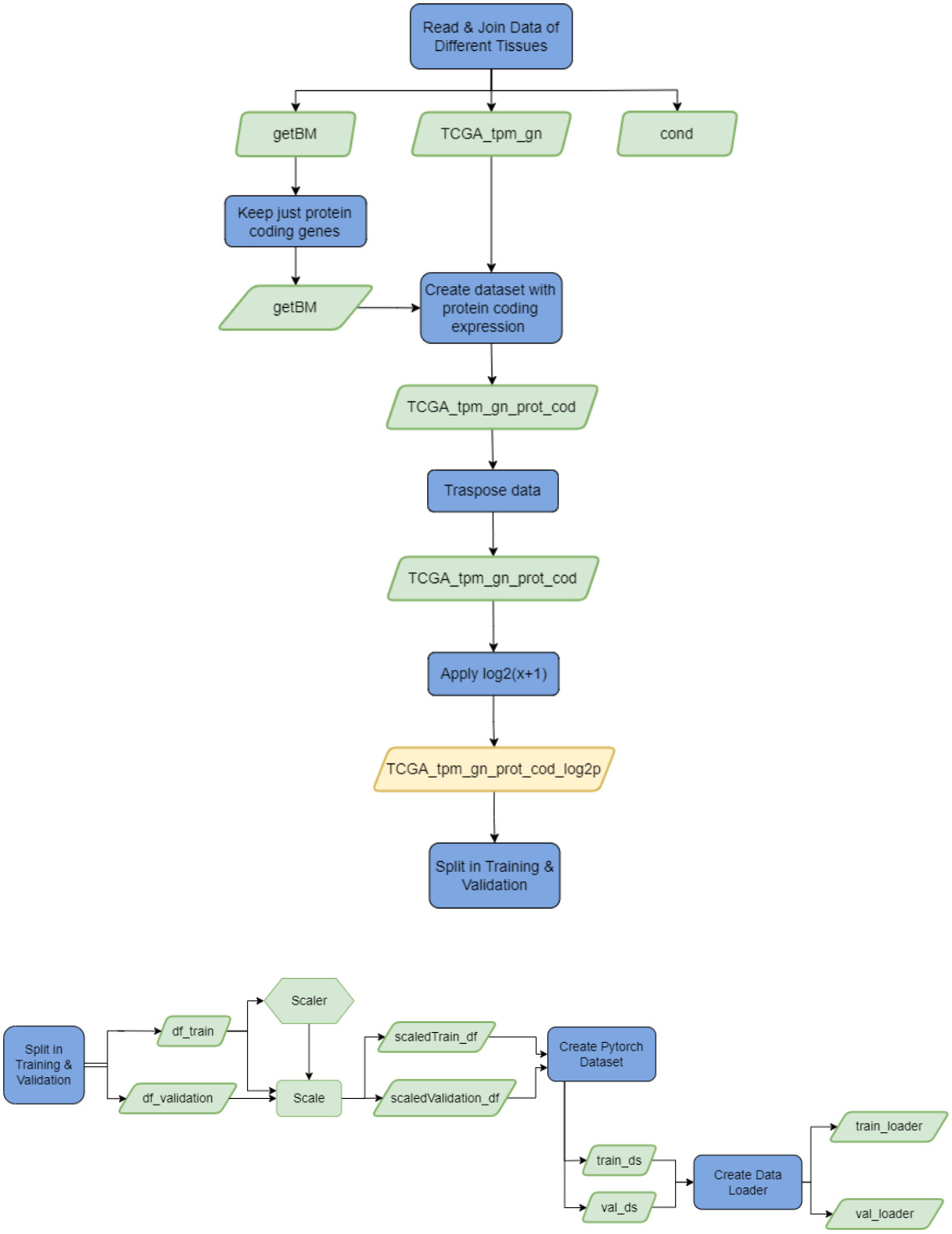


Figure 7. Pre-processing workflow to prepare the data to train the DeepAE model with PyTorch.

The workflow shown on **Figure 7** includes the following steps:

- the original dataset preparations: read and join different cancer types,
    - TCGA_tpm_gn dataset contains the gene expression data,

    - getBM dataset contains the information that relates each one of the isoforms with its respective gene,
    - cond dataset indicates the condition of the patients,
- filtering of the getBM dataset, to just keep the protein coding genes,
- creating a dataset with the expression of the protein coding genes,
- transposing the protein coding expression dataset,
- applying a log2 transformation of the dataset with the expression of the protein coding genes,
- splitting data in training and validation,
- scaling with MinMax Scaler of the the protein coding genes expression,
- creating the Pytorch dataset and the data loader.

3.2. RAW DATA PROCESSING

The data of patients was obtained from the The Cancer Genome Atlas Program (TCGA) (The Cancer Genome Atlas Program, s.f.). This program was carried out in a collaboration between NCI and the National Human Genome Research Institute and is publicly available. For each tumor type the samples were parsed by means of some tsv files and a code written in R to calculate the gene expression of all of them. As a result, the following four datasets were obtained:

- The TCGA_tpm dataset contains the isoform expression data of the patients in transcripts per million (TPM) of 199,169 transcripts.
- The TCGA_tpm_gn dataset contains the gene expression data of patients in TPM from 60,554 genes.
- The getBM dataset contains the information that relates each one of the isoforms with its respective gene. Both the ID and the name of the transcript and of the gene and the biotype are indicated. It has as many records as isoforms, i.e, 199,169.
- The cond dataset indicates the condition of the patients, that is, they tell us if the sample is tumor, normal or another. There is one sample for each patient.

3.2.1. Data processing for DeepSF2hidden with gene expression model

3.2.1.1. *Read and join data of different tumor types*

In the first steps of modeling only lung adenocarcinoma tissue (LUAD) samples were used. Subsequently, samples from 7 different adenocarcinomas were utilized to train the final models. For each tumor type, the samples were parsed and concatenated to form the same 4 datasets described in the previous section. In *cond*, it was added to each sample which tissue it came from.

The number of tumor, normal and other samples for the different tumor types is presented in the following table:

Tissue	Condition	Number of samples	Total
Lung Adenocarcinoma (LUAD)	Tumor	516	600
	Normal	59	
	Other	25	
Colon adenocarcinoma (COAD)	Tumor	458	524
	Normal	41	
	Other	25	
Pancreatic adenocarcinoma (PAAD)	Tumor	178	183
	Normal	4	
	Other	1	
Prostate adenocarcinoma (PRAD)	Tumor	497	554
	Normal	52	
	Other	5	
Rectum adenocarcinoma (READ)	Tumor	166	177
	Normal	10	
	Other	1	
Ovarian serous cystadenocarcinoma (OV)	Tumor	422	430
	Other	8	
Stomach adenocarcinoma (STAD)	Tumor	416	453
	Normal	37	
			2921

Table 1. Number of tumoral, normal and other samples per adenocarcinoma used to train the final model.

It is very interesting to emphasize that there was a large number of tumor samples.

In addition, the data expression of the transcripts come from 47 different biotypes, being the five main ones protein coding, processed transcript, retained intron, lincRNA and nonsense mediated decay.

Biotype	*Number of isoforms*
Protein coding	79,930
Processed transcript	26,977
Retained intron	26,704
LncRNA	13,481
Nonsense mediated decay	13,409

Table 2. Number of isoforms per biotype for the main five biotypes.

As can be seen in the last table, not all the transcriptome has a direct coding function. We can see, among others, a large number of processed transcripts that are genes or transcripts that do not have a sequence of bases through which the ribosome can travel and translate their information, read in codons (series of three bases), into amino acids to synthesize a protein.

Long non-coding RNA (lncRNA) are sequences of more than 200 bp that do not encode proteins. Within this group the retained introns can be described, which are transcripts that from an alternative splicing have introns (non-coding reads) of transcripts that are protein coding (Ensembl, s.f.).

In addition, the nonsense mediated decay are transcripts that are marked with a premature stop codon so that they can be eliminated by a RNA quality control mechanism so that these sequences do not end up being translated into proteins that can cause any pathology in the organism (Tatsuaki Kurosaki, 2016).

3.2.1.2. *Obtaining the dataset with the expression of the splicing genes*

From a list containing 1279 known RNA-binding proteins (RBPs), which were splicing factors (SFs), and the dataset with the gene expression, the dataset with the expression of SFs was generated. This set of data, which has a size of *1279 x number of samples*, will be the main input of the model.

3.2.1.3. *Removing single transcripts from the isoform expression dataset*

In this project we want to predict the expression of the different transcripts of the genes using the expression of the SF genes. For this reason, genes that only have one isoform are not of interest since alternative splicing will not occur in these cases. Therefore, these unique isoforms were removed from the transcript dataset, being 36,740 the number of isoforms eliminated. These transcripts were also removed from the getBM dataset. Then, the dataset with the gene expression of the transcripts without the unique isoforms and the getBM will have a size of *162,429 transcripts x number of patients*.

3.2.1.4. *Logarithmic transformation of the transcript expression*

Due to the great differences in the expression levels between the different transcripts and the presence of outliers in them, the implementation of a base 2 logarithmic transformation of the variables was studied:

$$\vec{z} = log2(\vec{x} + 1)$$

Equation 13. The base-2 logarithmic transformation applied to the isoform expression.

Where $\vec{x}$ is a vector with the original expression values of all patients for a transcript and $\vec{z}$ is the result of the logarithmic transformation for a given transcript.

For example, as it can be seen in the following figure for the ENS50000050290 transcript, for most LUAD patients it is poorly expressed but then there are certain patients who have an abnormally high expression. So there is presence of outliers.

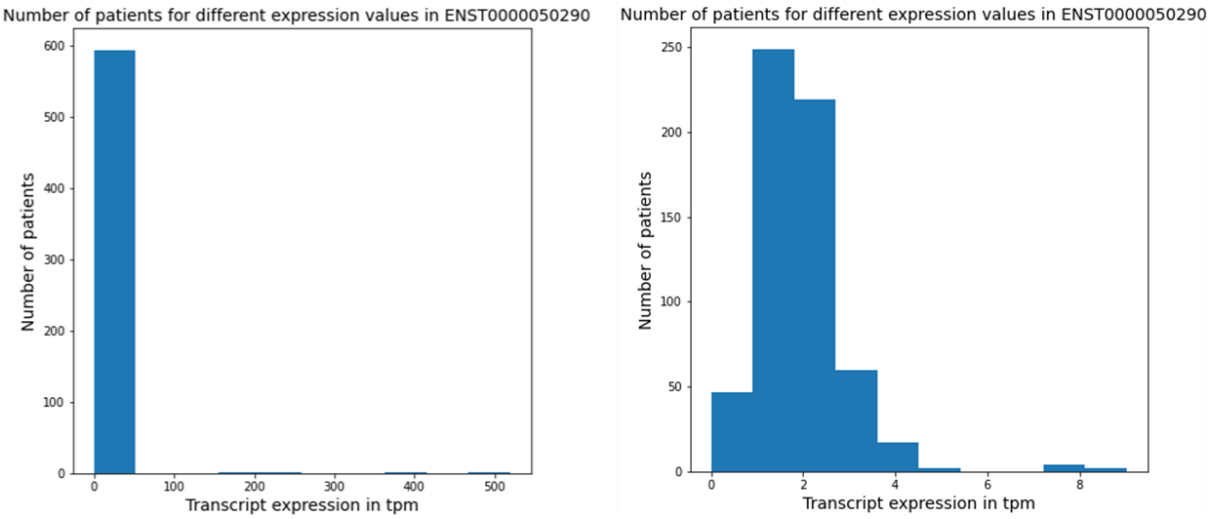


Figure 8. Number of patients for different expression values in tpm for the ENST000050290 isoform before and after applying a base-2 logarithmic transformation function.

A new dataset was created with the expression of the transcripts after performing the transformation function. Later on, in the model development process, the performance of the model was compared using the expression of the original transcripts with those that had been transformed.

3.2.1.5. *Select the cancer related genes and filter transcript data*

From a list of 898 cancer-related genes and getBM, the dataset that relates the isoforms to their genes, the cancer-related transcripts were filtered. This made a total of 10095 isoforms. This processing step was carried out at the same time as the gene expression for each isoform was introduced in the model.

The samples in getBM were filtered only with the information of the selected genes and their isoforms.

3.2.1.6. *Create the dataset with the gene expression of each transcript*

After this, the dataset with the expression of the genes for each transcript was generated from getBM dataset and the dataset with the expression of all genes. This new set, which will later be the second input of the model, has the same size as the label we want to predict (the transcript expression data) and contains the gene expression of the patients for each isoform, where the expression of a gene can belong to two or more isoforms.

3.2.1.7. *Transposition of the datasets*

The datasets with the expression of the SFs, the genes for each transcript and the isoforms were transposed, in such a way that the information of the patients was in the rows and the information of the genes in the columns.

3.2.1.8. *Dividing data in training and validation and scaling*

Random partitioning of the data was performed with the scikit-learn package using a test size of 0.2. That is to say, 80% of the set was used for training and the remaining 20% to validate the model. The splitting was done in the same proportion both when we worked with the LUAD samples alone and when we worked with the samples of the different adenocarcinomas.

Moreover, since the expression values of the genes vary considerably between their magnitude and ranges, which can negatively affect the optimization of the model, the data with the SFs expression and the gene expression were scaled by applying the standard scaler.

This is a preprocessing method from the scikit-learn library that standardizes the features by eliminating their mean (thus ensuring that each of the features presents a Gaussian distribution with mean 0) and obtaining a standard deviation of 1. The formula used for calculating the Standard Scaler score of a sample is as follows:

$$z = \frac{x - \mu}{\sigma}$$

Equation 14. Calculation of the Standard Scaler scores

Where x is a sample and μ and σ are the mean and the standard deviation of the training samples. Therefore, for each feature in the set, the mean and standard deviation were calculated fitting the training data into the scaler. These calculations are then used to, through a scalar transformation function, scale the training and validation data.

3.2.1.9. *Create the PyTorch Dataset and Data Loader*

As a preliminary step to modeling using Pytorch, the data was converted to Pytorch tensors and tensor datasets were created for training and validation data. Each tensor dataset contained the following three tensors: the expression of the SFs, the transcript expression (the label to be predicted in modeling) and the expression of the genes.

From these two tensor datasets two loaders were created so that batches could be generated automatically when training.

3.2.2. Data processing for DeepAE

The information from the different cancer types was merged in a similar way as it was done for the DeepSF2hidden model.

With the development of this autoencoder our intention was that our model would be able to identify the particular characteristics of each tumor type. To this end, it was decided to focus only on protein-coding genes. For this reason, the getBM dataset was filtered so that it only had data referring to protein-coding genes.

Subsequently, from this filtered dataset and the set with the expression of all genes, a dataset with the expression of only protein-coding genes was generated. This was the input data of the model that we wanted to replicate.

The data was then transposed so that the patient information was in the rows and the protein-coding gene expression in the columns. Also, a 2-base logarithmic transformation was applied following the preprocessing methodology applied in (Sanjiv K. Dwivedi, 2020). The division into training and validation was once again made in 80% for training and 20% for validation.

The training and validation data were scaled using a MinMaxScaler from the preprocessing package of scikit-learn library. This was the scaling method employed by the researchers in (Sanjiv K. Dwivedi, 2020). With this method we can scale the features in a desired range (in our case between 0 and 1) using the following mathematical expression:

$$\hat{x} = \frac{x - \min(x)}{\max(x) - \min(\mathrm{x})}$$

$$\hat{x}_{scaled} = \hat{x} \cdot (\max feature\ range - \min feature\ range) + \min feature\ range$$

Equation 15.Mathematical formulation of the MinMaxScaler

Where if the maximum value range of a desired feature is equal to one and the minimum equal to zero, $\hat{x}_{scaled} = \hat{x}$.

Once the data was scaled, the Pytorch dataset and data loaders were created.

3.3. MODELING

3.3.1. Development of DeepSF2hidden with gene expression

3.3.1.1. *Optimize the number of nodes using a 2-hidden-layer arquitecture*

For the first model tried a fully-connected sequence model with two hidden layer (HL) was designed to predict the isoform expression. To build the model the Keras deep learning API was used, which integrates the TensorFlow Machine Learning library. The activation function used for each of the layers was the rectified linear unit (ReLU).This function allows to activate a node if the input introduced is positive, having a linear dependency relationship between input and output.

The number of nodes per hidden layer was optimized using a 3-fold Randomized Search CV from the scikit learn library using just the training data. The following configuration settings were tried for this run:

Training configuration setup

Set of samples	LUAD training set (n=480)
Nº of nodes in hidden layer 1	range(100,1000)
Nº of nodes in hidden layer 2	range(10,100)
Nº of parameter settings tried	5
Nº of epochs	100
Batch size	20
Learning rate	0.1
Loss function	Negative Mean Squared Error
Optimizer	Adagrad
Epsilon	1e-08
Decay	0.0

Table 3. Run configuration setup used for the optimization of the number of nodes with Randomized Search CV.

Being the number of parameter settings tried the number of different combinations of HL1 and HL2 used from the specified distributions.

This run was repeated several times, always resulting in the best parameters for layer 1 around 183 nodes and for layer 2 around 82 nodes. So these values were used in the models with two hidden layers.

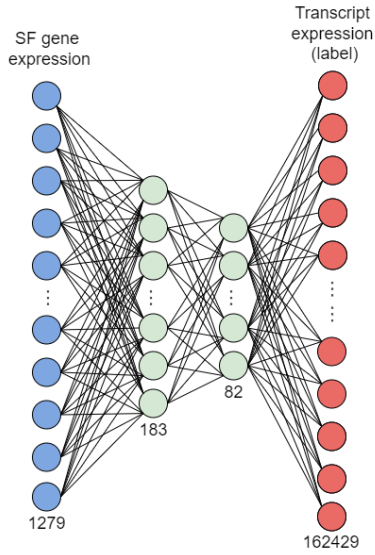


Figure 9. Architecture of the best model obtained in the optimization of the number of nodes with Randomized Search CV.

Where the blue nodes refer to the 1279 SF genes, the light blue nodes refer to the 183 and 82 nodes of HL1 and HL2, and the red nodes are the 162,429 isoforms whose expression is to be predicted.

3.3.1.2. *Select the number of hidden layers, the loss function and trying different pipelines to counteract variability in isoform expression*

In this step different pipelines were tested, using different loss functions, applying or not a transformation function to the labels and adding weighting values of the features to the loss function. In addition, two model architectures were studied using Keras: a model that had 2 hidden layers and another with 3 hidden layers.

The pipelines studied in this step were the following:

- RMSE as loss function with a 2 hidden layer (183-82) model without label transformation.
- RMSE as loss function with 2 hidden layer model (183-82) without label transformation but adding weight values to the loss function.
- MAE as loss function with 2 hidden layer model (183-82) without label transformation.
- RMSE as loss function with 2 hidden layer (183-82) and the application of a transformation function to the label.
- RMSE as loss function with 3 hidden layer model (256-128-64) without label transformation.

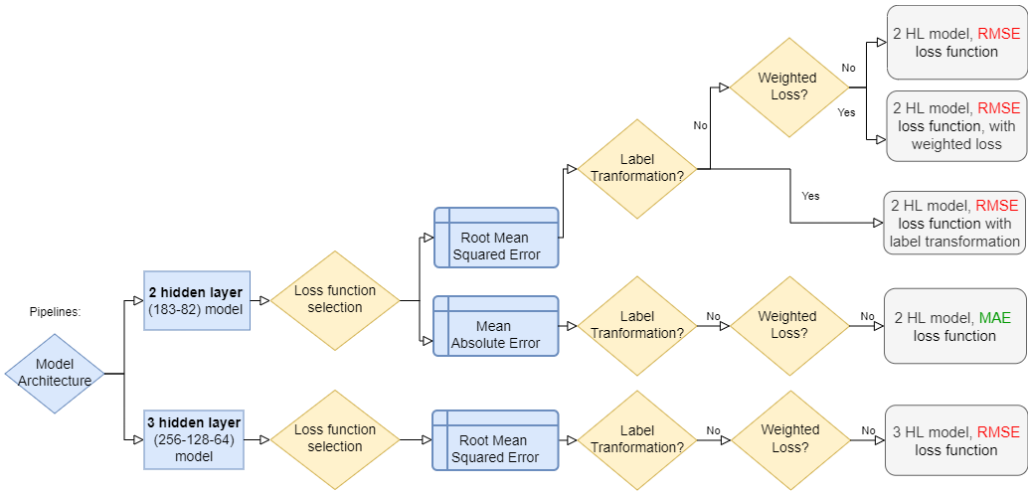


Figure 10. Different pipelines studied: selection of the loss function, number of hidden layers, label transformation and adding weight values to the loss function.

For all the models run, a learning rate of 0.1, 300 epochs, Adagrad as optimizer and a batch size of 32 were used. The performance of the models was evaluated using the validation dataset. The error of the model was compared with the one made by computing the mean and the median of all the patients. Also, the model error per isoform was calculated.

In addition, for each run model, the next plots were made:

- The five best and five worst predictions of transcripts. For the patients in the validation set, the theoretical and model-predicted expression in tpm were shown in descending order of expression values.
- The model error in log10(x+1) for each of the transcripts against the median of the expression of each of them in log10(x+1).

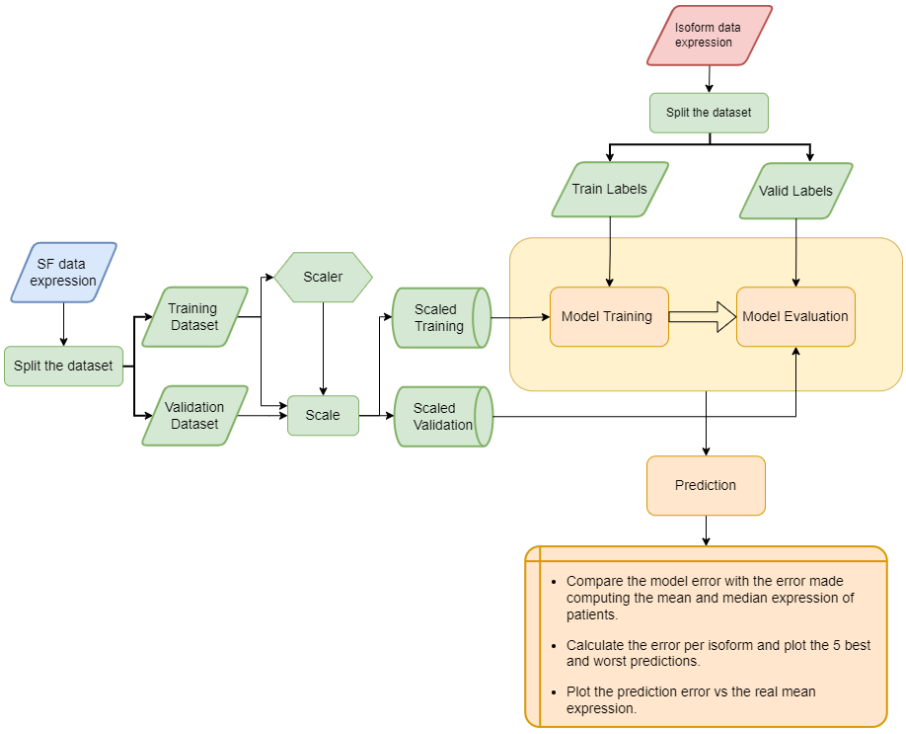


Figure 11. Data processing, model training and evaluation to select the best pipeline.

Regarding the implementation of the parameter loss weights to the loss function to weight the model output losses, the patient mean for each transcript was calculated (from LUAD samples) and the following function was applied to obtain the vector of transcript weights:

$$\vec{z} = \frac{1}{\sqrt{\vec{x}}}$$

Equation 16. Function applied to calculate the loss weight vector of the transcripts.

Where $\bar{x}$ is a vector with the mean of the different transcripts and $\bar{z}$ is a vector with its associated loss weights. In such a way that the higher the mean value of the expression of the isoform, the lower its weight would be. Subsequently, the weight vector was normalized in such a way that the total sum of the weights of the transcripts would add up to 1.

$$\vec{z} = \frac{\vec{z} - \min(\vec{z})}{\sum(\vec{z} - \min(\vec{z}))}$$

Equation 17. Normalization of the loss weight vector.

On the other hand, in this step the label transformation strategy was studied. To all the transcripts in the LUAD training and validation set a logarithmic transformation of the expression was calculated to counteract the presence of expression outliers.

In light of the disappointing results obtained when adding a weighted loss of the transcripts, it was decided, instead, to add a new input to the model: adding the gene expression, that is, associating each transcript with the expression of its gene. In addition, a log2 transformation of the transcript data continued to be used.

Regarding the loss function, it has been observed that the RMSE (and MSE) allows us to make better predictions of the transcripts and we will continue using them metric in the next step of the model development.

Due to the need of implementing a new input to the model, continuing to use a Keras sequential model was no longer a viable option, and it was decided to develop the models using PyTorch.

3.3.1.3. *Add gene expression to the model and study just the cancer related genes*

Pytorch allows you to create models with more complex architectures, having a much greater control of all the variables when developing and training a Machine Learning model. The main difference with Tensorflow is that the data is stored directly in tensors, which allow computing the gradients of the parameters with respect to the defined loss function. This is very useful when optimizing the loss function, since we need to update the parameters in the direction that minimizes it.

In this step it was decided to use only certain genes in the modeling, i.e, only the expression of the isoforms of the selected genes were calculated. Even though the code allows us to select all the genes and therefore create a model that tries to predict all the isoforms, we chose to select only the genes that have been

shown to be related to the development of cancer (which means 898 genes and 10,095 isoforms to predict).

Each model developed with PyTorch was defined as a class. Within this, the following functions were programmed to:

- create the model object, indicating which are going to be the transformations to be applied, that is, initializing which are going to be the model layers with its number of input and output nodes.
- perform the forward step, that is, generate predictions when input data is introduced into the model. The different layers are joined, an activation function is applied to them and this output is the prediction made by the model.
- generate the training_step. The input of this function must be a batch containing the input(s) and the output. This function is capable of generating predictions, calculating the loss and the gradients of the parameters with respect to that loss function and updating the weights through the use of an optimization function.
- validation_step is a function that calculates the model loss for a batch of validation data.
- combine the losses of different batches to show the results per epoch.

A configuration class was created to set the characteristics of the Deep Neural Network. It indicates the number of epochs for training, the optimizer to use, the name of the model, the batch size, the learning rate, if the data is saved in Weight and Biases (wandb) or not, the size of the validation set and the number of genes and number of tumor types to be introduced into the model.

By means of the get_data function, which requires as an input argument the path of the data files and the model configuration, we are able to prepare all the data necessary to train the model.

Regarding the model architecture, it has two inputs: the expression of the SFs introduced in the input layer and the gen expression of each of the isoforms introduced into the last linear layer. In addition, there are 2 hidden layers with 183 and 82 nodes, respectively, and the output layer is formed by 10,095 nodes, which are the cancer related transcripts we want to predict.

The output of a layer results from applying a linear function of the signal coming from the nodes of the previous layer, which activation is regulated by the function ReLU. The mathematical formulation of the DNN model *Deep2hidden with gene expression* can be seen below:

$$out\ L_1 = relu(SF \cdot W^{(1)^T} + b^{(1)})$$

$$out\ L_2 = relu(\widehat{out\ L_1} \cdot W^{(2)^T} + b^{(2)})$$

$$y = relu(\widehat{out\ L_2} \cdot W^{(3)^T} + b^{(3)} + W_{krn} \odot Gn)$$

$where,$

$$SF \in \mathbb{R}^{batch\ size\ x\ 1279\ (nº\ features)}$$

$$W^{(1)} \in \mathbb{R}^{183\ x\ 1279}, W^{(2)} \in \mathbb{R}^{82\ x\ 183}, W^{(3)} \in \mathbb{R}^{10095\ x\ 82}$$

$$b^{(1)} \in \mathbb{R}^{1\ x\ 183}, b^{(2)} \in \mathbb{R}^{1\ x\ 82}, b^{(3)} \in \mathbb{R}^{1\ x\ 10095}$$

$$Gn \in \mathbb{R}^{batch\ size\ x\ 10095\ (nº\ transcripts)}$$

$$W_{krn} \in \mathbb{R}^{batch\ size\ x\ 10095}$$

$$y \in \mathbb{R}^{batch\ size\ x\ 10095}$$

Equation 18. Mathematical formulation to create the DeepSF2hidden with gene expression model.

Where SF and Gn are the two inputs of the model, the expression of the splicing genes and the gen expression of each transcript, y is the output of the model and $W^{(i)}$ and $b^{(i)}$ are the parameters of each linear layer.

Having introduced the expression of the genes in the model a new parameter Wgn was created, which is the vector of weights associated with these genes. This must be multiplied scalarly element by element with the expression values of the genes for each transcript. However, in order to perform this operation Wgn and Gn must have the same size. For this reason, the matrix W_{krn} was calculated, so that the weights and the gene expression data could have the same size. This matrix contains the vector of weights repeated for each sample in the batch.

This W_{krn} was obtained by performing the following Kronecker product between a matrix of ones and the Wgn:

$$W_{krn} = a \otimes Wgn = \begin{bmatrix} [a_1] \cdot [Wgn_1, Wgn_2, Wgn_3, \dots, Wgn_n] \\ [a_2] \cdot [Wgn_1, Wgn_2, Wgn_3, \dots, Wgn_n] \\ \dots \\ [a_m] \cdot [Wgn_1, Wgn_2, Wgn_3, \dots, Wgn_n] \end{bmatrix} = \begin{bmatrix} [Wgn_1\ \ Wgn_2\ Wgn_3 \dots\ \ Wgn_n] \\ [Wgn_1\ \ Wgn_2\ Wgn_3 \dots\ \ Wgn_n] \\ \dots \\ [Wgn_1\ \ Wgn_2\ Wgn_3 \dots\ \ Wgn_n] \end{bmatrix}$$

$$a \rightarrow matrix\ of\ ones \in \mathbb{R}^{batch\ x\ 1}$$

$$Wgn \in \mathbb{R}^{nº\ transcripts}$$

$$m = 1, \dots, batch\ size$$

$$n = 1, \dots, num\ transcripts$$

And therefore the dot product between gene expression and its weights is:

$$W_{krn} \odot Gn = \begin{bmatrix} [Wgn_1 \cdot Gn_1^{(1)}\ \ Wgn_2 \cdot Gn_2^{(1)}\ Wgn_3 \cdot Gn_3^{(1)} \dots\ \ Wgn_n \cdot Gn_n^{(1)}] \\ [Wgn_1 \cdot Gn_1^{(2)}\ \ Wgn_2 \cdot Gn_2^{(2)}\ Wgn_3 \cdot Gn_3^{(2)} \dots\ \ Wgn_n \cdot Gn_n^{(2)}] \\ \dots \\ [Wgn_1 \cdot Gn_1^{(m)}\ \ Wgn_2 \cdot Gn_2^{(m)}\ Wgn_3 \cdot Gn_3^{(m)} \dots\ \ Wgn_n \cdot Gn_n^{(m)}] \end{bmatrix}$$

Equation 19. Dot product between the weight values and the gen expression in the DeepSF2hidden with gene expression model.

The operation to obtain W_{krn} was coded in the forward step prior to the calculation of the predictions. During training the values of this matrix were updated as the vector of weights Wgn was updated. The gradients of this parameter were calculated with respect to the same loss function as the parameters of the model layers. So we are able to optimize all weights with the same loss function.

Moreover, just as the input data was scaled before introducing it into the model, two batch normalization layers were placed after the first hidden layer and the second hidden layer, so that a similar distribution of the data was maintained. As can be seen in Equation 18, the input argument for the L2 and output layer is the normalized output data from the previous layer.

The technique proposed (Sergey Ioffe, 2015) allows to reduce how much the values of the hidden units shift around, avoiding the node activation to be too high or to low and increasing the stability of the NN.

This allows to increase the learning rate and thus accelerate the learning process. It also helps to reduce the need to use regularization methods such as drop-out, as this can be an effective technique to reduce overfitting.

Thus the batch-normalization layer normalizes the signals coming from previous layer by removing for each node the mean of the batch and dividing it by the standard deviation, so that to have a mean zero and standard deviation of one.

A smoothing term for numerical stability ϵ (set to a value equal to $1 * 10^{-5}$), which permit the output not to go to infinity when the standard deviation tends to 0. The mathematical formulation of the batch normalization of a node is presented below:

$$E[x] = \frac{1}{n}\sum_{i=1}^{n} x_i \ (batch\ mean)$$

$$Var[x] = \frac{1}{n}\sum_{i=1}^{n} (x_i - \mu_B)^2 \ \ (batch\ variance)$$

$$\hat{x} = \frac{x - E[x]}{\sqrt{Var[x] + \epsilon}} \cdot \gamma + \beta \ (batch\ normalization)$$

Equation 20. Mathematical expression of a Batch normalization

Where x are the values of a feature for a batch and y are the normalized values.

Two learnable parameters γ and β can be introduced, which can scale and shift the normalized layer during training; so that the representation power of the NN is optimal.

However, in PyTorch by default this re-scaling and shifting processes are not applied, as they are set to a value equal to one and a value equal to zero, respectively. So we did not introduce them in our models either.

3.3.1.4. *Best optimizer selection using the Pytorch model*

Once the MSE was defined as the loss function for our model, the next step was to choose the appropriate optimizer. Several training runs were carried out to select the best of them.

To evaluate and compare the performance of the model using different optimizers, the following results were considered:

- The MSE loss with the validation set,
- the total correlation between the real values and the predicted ones and the correlation of each of the transcripts for training and validation data, using the Spearman's correlation coefficient.

Spearman's correlation measures how strong is the monotonic relationship between two sets of data. It performs this by ranking the values of each of the variables from lowest to highest and calculates the linear correlation of these rankings by using the Pearson's correlation. The values of the coefficients vary between -1 and 1, where a value of 1 means there is a positive exact monotonic relationship, a value of -1 a negative exact monotonic relationship and a value of 0 means that there is no relationship between the compared data.

In addition, the correlation for each biotype was also plotted for training and validation data.

Among the optimizers that were tested we find the Stochastic Gradient Descent (SGD) with momentum (0.9, 0.7 and 0.5); the Averaged Stochastic Gradient Descent (ASGD), the Adaptive Gradient (AdaGrad), the Adadelta, the Adaptive moment estimation (Adam) and the Adaptive moment estimation with weight decay (AdamW).

The configuration parameters for this run were the following:

Settings	*Value*
Nº of epochs	3000
Learning Rate	0.001
Cancer Type	LUAD
Test Size	0.2
Number of Genes	898 (Cancer related genes)
Batch size	32
Loss function	MSE

Table 4. Training configuration setup in the selection of the most suitable optimization algorithm with DeepSF2hidden with gene expression model.

The trained parameters of the models and the results obtained were saved in wandb.

Since the models optimized with Adam and AdamW obtained the highest correlation values and the lowest error values, it was decided to continue using AdamW when training the model with all the samples.

3.3.2. Development of DeepAE: training an autoencoder from samples of different adenocarcinomas

Based on the architecture designed in (Sanjiv K. Dwivedi, 2020), an autoencoder was developed to detect the particular characteristics of different tumor tissues from the expression data of protein-coding genes. The architecture of this model consists of 3 hidden layers of 512 nodes each, and an input and output layers that have as many nodes as protein-coding genes we have: 19594.

Nodes in all layers are densely connected. The signal is propagated through the network by applying linear functions to which an activation function is passed. In this case a sigmoid function was employed as realized in (Sanjiv K. Dwivedi, 2020). After L1 and L2, a batch normalization layer was introduced.

The mathematical formulation of DeepAE is presented below:

$$out\ L_1 = sigmoid(PCGn \cdot W^{(1)^T} + b^{(1)})$$

$$out\ L_2 = sigmoid(\widehat{out\ L_1} \cdot W^{(2)^T} + b^{(2)})$$

$$out\ L_3 = sigmoid(\widehat{out\ L_2} \cdot W^{(3)^T} + b^{(3)})$$

$$y_{PCGns} = sigmoid(out\ L_3 \cdot W^{(4)^T} + b^{(4)})$$

$$where,$$

$$PCGn \in \mathbb{R}^{batch\ size\ x\ 19594\ (nº\ features)}$$

$$W^{(1)} \in \mathbb{R}^{512\ x\ 19594\ (nº\ features)}$$

$$W^{(2)}, W^{(3)} \in \mathbb{R}^{512\ x\ 512}$$

$$b^{(1)}, b^{(2)}, b^{(3)} \in \mathbb{R}^{1\ x\ 512}$$

$$W^{(4)} \in \mathbb{R}^{19594\ (nº\ features)\ x\ 512}$$

$$b^{(4)} \in \mathbb{R}^{1\ x\ 19594\ (nº\ features)}$$

$$y_{PCGns} \in \mathbb{R}^{batch\ size\ x\ 19594\ (nº\ features)}$$

Equation 21. Mathematical formulation of the DeepAE autoencoder model.

Thus, the training configuration is presented below:

Settings	*Value*
Epochs	1000
Learning Rate	0.0001
Cancer Type	7 different adenocarcinomas
Test Size	0.2
Loss function	MSE
Number of Genes	19594
Batch size	256
Optimizer	Adam
Beta 1	0.9
Beta 2	0.999
Epsilon	$1 \cdot 10^{-8}$
Weight decay	$1 \cdot 10^{-6}$

Table 5. Training configuration setup for the DeepAE model.

Where we applied the learning rate, the batch size, the optimizer and its hyperparameters that were utilized in (Sanjiv K. Dwivedi, 2020).

For each training step, the MSE error between the theoretical protein gene expression values and those predicted by the model was calculated.

The trained model with its weights was saved for then be used in the final model by transfer learning.

3.3.3. Development of deepSF&AEensemble

3.3.3.1. *Concatenate deepAE with DeepSF2hidden with gene expression to create the final model*

To create the final model, the already trained autoencoder was introduced in the architecture, thus adding the particular characteristics of each tumor type. For this purpose, using Pytorch, the DeepAE model object was again initialized and the already trained weights were loaded. Then, the last layer of the autoencoder was removed and the remaining model was introduced in the DeepSF&AEensemble model object as an input argument.

In the forward function of the model class, the L3 of the autoencoder was connected to the last linear layer of the model, i.e., the nº batchx82 layer of the model2hidden and the nº batchx512 layer of the autoencoder were concatenated in the last linear layer. The previous layers of the autoencoder were frozen so that the already trained parameters were not updated during training. Only the L3 parameters were updated. Again, batch normalization was performed after L1 and L2.

The mathematical formulation of the final model is presented as follows:

$$\begin{cases} out\ L_1 = relu(SF \cdot W^{(1)^T} + b^{(1)}) \\ out\ L_2 = relu(\widehat{out\ L_1} \cdot W^{(2)^T} + b^{(2)}) \end{cases} \qquad \begin{cases} out\ AE_{L_1} = sigmoid(PCGn \cdot W^{(AE_1)^T} + b^{(AE_1)}) \\ out\ AE_{L_2} = sigmoid(\widehat{out\ L_1} \cdot W^{(AE_2)^T} + b^{(AE_2)}) \\ out\ AE_{L_3} = sigmoid(\widehat{out\ L_2} \cdot W^{(AE_3)^T} + b^{(AE_3)}) \end{cases}$$

$$y = relu((\widehat{out\ L_2} \oplus\ out\ AE_{L_3}) \cdot W^{(3)^T} + b^{(3)} + W_{krn} \odot Gn)$$

where,

$$SF \in \mathbb{R}^{batch\ size\ x\ 1279\ (nº\ features)}$$

$$W^{(1)} \in \mathbb{R}^{183\ x\ 1279}, W^{(2)} \in \mathbb{R}^{82\ x\ 183}$$

$$b^{(1)} \in \mathbb{R}^{1\ x\ 183}, b^{(2)} \in \mathbb{R}^{1\ x\ 82}, b^{(3)} \in \mathbb{R}^{1\ x\ 10095}$$

$$Gn \in \mathbb{R}^{batch\ size\ x\ 10095\ (nº\ transcripts)}$$

$$W_{krn} \in \mathbb{R}^{batch\ size\ x\ 10095}$$

$$PCGn \in \mathbb{R}^{batch\ size\ x\ 19594\ (nº\ features)}$$

$$W^{(AE_1)} \in \mathbb{R}^{512\ x\ 19594\ (nº\ features)}$$

$$W^{(AE_2)}, W^{(AE_3)} \in \mathbb{R}^{512\ x\ 512}$$

$$b^{(AE_1)}, b^{(AE_2)}, b^{(AE_3)} \in \mathbb{R}^{1\ x\ 512}$$

$$W^{(3)} \in \mathbb{R}^{10095\ x\ 82+512}$$

$$y \in \mathbb{R}^{batch\ size\ x\ 10095}$$

Equation 22. Mathematical formulation to create the DeepSF&AEensemble model.

Where $\oplus$ is the concatenation of both matrices and $\odot$ is the dot product.

In order to run this DNN and make the predictions, the training and validation data loader must have the following data sets:

- The gene expression of the SFs
- The expression of the transcripts
- The gene expression of each transcript
- The gene expression of the protein-coding genes.

This is training configuration of the DeepSF&AEensemble model:

Settings	*Value*
Nº of epochs	3000
Learning Rate	$1 \cdot 10^{-4}$
Cancer Type	7 adenocarcinoma types (n=2921)
Test Size	0.2
Number of Genes	898 (Cancer related genes)
Batch size	128
Loss function	MSE
Optimizer	AdamW
Betas	(0.9,0.999)
Epsilon	$1 \cdot 10^{-8}$
Weight decay	0.01

Table 6. Training configuration setup to train the DeepSF&AEensemble model.

To evaluate the performance of the model the MSE loss and the total Spearman's correlation coefficient between the real values and the predicted ones for training and validation data were used.

To plot the final results the methodology used in DeepSF2hidden model was applied.

The trained parameters of the models and the results obtained were saved in wandb. The required time to run the model was 2h 10m 34s.

3.4. INTEPRETATION OF MODEL PARAMETERS USING DEEP LEARNING IMPORTANT FEATURES (DEEPLIFT)

3.4.1. Weight-score calculation with DeepLIFT

After training the models, we tried to identify the SFs that regulate the gene expression by interpreting the parameters of the model. For this purpose, the Deep Learning of Features (DeepLIFT) algorithm was applied to the Deep2hidden with gene expression model, which was optimized with AdamW and trained with LUAD samples.

Since DeepLIFT requires reference values to compare with to calculate the weight-scores, the selection of a suitable baseline is the key to see the effect we want to discover. In this case, the median expression of the SFs was used as the baseline.

Therefore, in order to calculate the contribution of the SFs to each model output, i.e., to the expression of each isoform, the following information was introduced in the algorithm:

- the validation set with the expression of the SFs from which we want to calculate their contribution to the output.
- the median expression of the SFs, which has a baseline function.
- additional arguments needed to run the forward step of the model, but from which we do not want to obtain the contributions. Formed by the validation set of the expression of the genes.

Where the contribution can be positive or negative depending if the increase of the SF expression leads to a increase or a decrease of the isoform expression.

In our case, since we developed an output layer with 10095 nodes, the SF scores for each isoform were computed by changing the target index at each iteration. And the stored in a data frame of size $1279\ x\ 10095\ (\#SFs\ x\ \#isoforms)$

3.4.2. Comparison of DeepLIFT results with theoretical values

3.4.2.1. *Data Processing*

To contrast the results obtained in DeepLIFT, they were compared with theoretical results of some SFs known to regulate the expression of certain genes and transcripts. This information was stored in binary-format in a matrix of 171 rows (SFs) and 118,830 columns (events). Where values equal to one indicate an existing *SF-gene-transcript-event* regulatory relationship and values equal to zero indicates a non-existent regulatory relationship. Thus, the theoretical data do not reflect whether such regulation is positive or negative. In other words, it will only be possible to study whether there is a relationship between the SFs and the genes.

See by example how the event information was stored for each gene-transcript initially: "ENSG0000022393972.5_2". Where the first part to the left of the dot refers to the Gene ID, the number to its right indicates the transcript and the number after the symbol "_" the event.

In order to make the theoretical data and the calculated by DeepLIFT comparable, both sets were processed. First, the number of SFs was filtered to have the same ones in both matrices. From the 171 SFs in the theoretical set, 148 were also present in out data set.

Then from the columns of the theoretical matrix the information related to the transcript and event was removed (what it was to the right of the dot) so to just keep the gene names. The information of the genes was merged by calculating the maximum value (which for this case will be always 1). This leaves us with a total of 148 rows (SFs) and 20784 columns (genes).

On the other hand, in the data frame with the DeepLIFT scores, the names of the columns were changed from the Transcript ID to its respective Gene ID. In

addition, genes were also merged by the maximum value of the absolute gene expression (resulting in the 898 cancer related genes). It was performed this way knowing that DeepLIFT also calculates negative contributions, which are not the subject of study in this case.

After this, the number of genes was filtered by number of common genes in both matrices. Only four of the genes we worked with were not in the theoretical set.

3.4.2.2. *Select the threshold to create a binary matrix from the DeepLIFT score data*

Once both sets of data had the information regarding the same SFs and genes, the scores from the DeepLIFT data frame were processed to create a binary matrix, so that the results could be compared with the theoretical data.

For this purpose, it was proposed to perform the median number of genes that a SF regulates, using the theoretical data. Thus, the number of genes regulated by each theoretical SF was calculated, being the SFs with the highest number of cancer-related genes regulated: DDX3X and TARDBP with 758 genes each. On the other hand, NSUN2 just regulates one gene. And the median number of genes regulated by a SF is 191.5.

Threshold values were tested between in a range between $1 - 10^{-4}$ and 0.1, with a step of $5 - 10^{-5}$ between the previous and the next tried value. Giving a total of 1998 combinations.

For each iteration a threshold value within that range was chosen and a copy of the DeepLIFT matrix was created. Values that exceeded the threshold were changed by a value equal to one and those that did not were assigned a value equal to 0. Subsequently, from the binary matrix resulted the median number of genes regulated by a SF was calculated and the absolute difference between the theoretical median value and the predicted one was performed.

So, for one iteration, if the value of the difference was lower than the minimum difference calculated up to that point, the new difference became the minimum difference and the threshold used was considered the current optimal threshold.

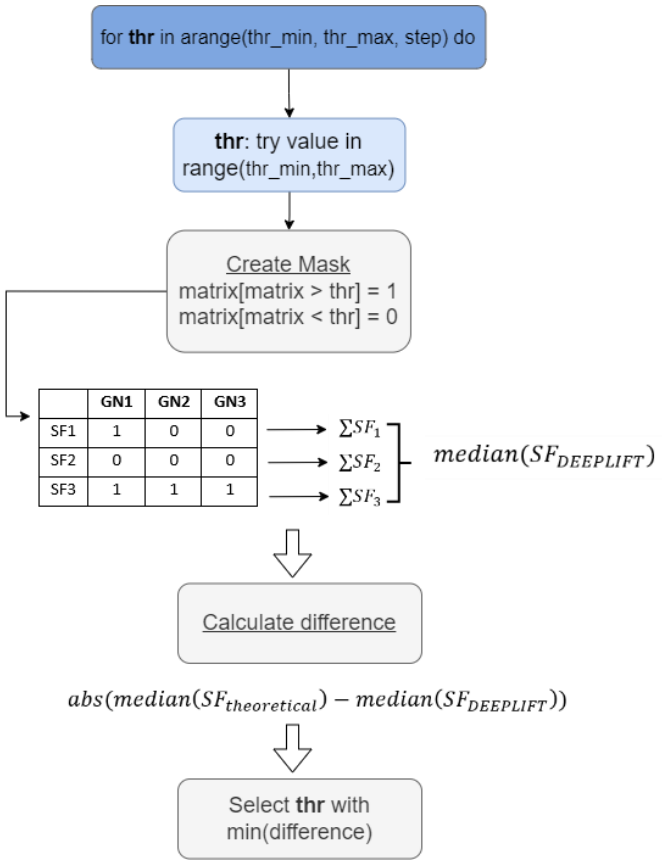


Figure 12. Methodology applied to select the optimal threshold to create a binary matrix from the scores calculated with DeepLIFT.

3.4.2.3. *Evaluation of the scores predicted by DeepLIFT by comparing the binary matrices with the confusion matrix*

To evaluate how good were the contributions calculated by DeepLIFT for our model, we worked with confusion matrices. Performing a confusion matrix is very useful to visualize many performance parameters as:

- accuracy, which studies the examples correctly classified over the total number of samples.
- sensitivity, that is the true positive rate.
- specificity, that is the true negative rate.

As well as other metrics that derived from these such as F1 score and recall.

Below a confusion matrix is presented. Where the rows refer to the theoretical values and the columns to the values predicted by our binary matrix. Also, $\boldsymbol{SF_{positive}}$ means that a corresponding SF regulates a gene and $\boldsymbol{SF_{negative}}$ that it does not.

Ground Truth /Pred Label/	$SF_{negative}$	$SF_{positive}$
$SF_{negative}$	TN	FP
$SF_{positive}$	FN	TP

Equation 23. Definition of confusion matrix.

Where TN refers to the negative examples and TP indicates the number of true regulations that our model has been able to identify. While FN are the samples that we have predicted as negative but the regulation actually exists and FP are the negative samples that we have predicted as positive. The above metrics can be calculated from these four values.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

$$Sensitivity = \frac{TP}{TP + FN}$$

$$Specificity = \frac{TN}{TN + FP}$$

$$recall = \frac{TP}{TP + FP}$$

$$F1 = \frac{2 \cdot precision \cdot recall}{precision + recall}$$

Equation 24. Definition of accuracy, sensitivity, specificity, recall and F1 score.

Thus, both a general confusion matrix, with the prediction results of all SFs, and a particular confusion matrix for each of the SFs were performed to evaluate their individual performance in predicting gene regulation.

4. RESULTS

4.1. DEEPSF2HIDDEN WITH GENE EXPRESSION

4.1.1. Selection of the number of hidden layers, the loss function and trying different pipelines to counteract variability in isoform expression

Regarding the use of the RMSE as loss function, different methodologies were tested such as the application of a transformation function to the isoforms to counteract the variability of the its expression and the presence of outliers and the addition of weight values of the transcripts to the loss function to penalize more the error made when predicting the low-expressed isoforms.

The following table presents the results obtained using these methodologies and comparing the model error with the error made when using the mean and median of the transcripts:

Model	*RMSE values*
2 HL with label transformation	0.42
Mean imputation of the features with label transformation	0.46
Median imputation of the features with label transformation	0.48
2 HL model without label transformation but with weighted loss	50.30
2 HL model without label transformation	50.86
3 HL model without label transformation	51.95
Mean imputation of features without label transformation	116.76
Median imputation of features without label transformation	120.72

Table 7. Comparison of the error committed using different methodologies with the error made by using the mean and median of the transcripts with RMSE as the loss function.

As can be seen in the table above, the use of a logarithmic transformation to the base 2 has been very successful, being much smaller the error made when performing the mean and median of the transcripts using this methodology that with all the other models tested. Since the RMSE penalizes larger errors to a greater extent, only with this methodology, which reduces the effect of outliers, was it possible to not have the transcript with the greatest expression variance in the entire data set, the IGKC-001 transcript, among the five isoforms with the highest error in the model. In addition, the learning curve during training was smoother and the model reached convergence earlier (See Appendix 8.1.1.1 and 8.1.1.4).

However, the use of weight values has not improved the predictive ability of the model for poorly expressed isoforms (see Appendix 8.1.1.2).

About the use of a greater number of hidden layers in the model, it has been seen that with 3 hidden layers the model converges faster (even before reaching epoch 150), as it has more parameters to train, than when using 2 layers (which didn't occurred until around epoch 200). However, then the model starts to overfit, as the training error continues to decrease, while it starts to increase in the validation set. Thus, the model losses its ability to generalize. The overfitting could have been controlled by introducing a regularization method such as dropout or an early stop when the loss in the validation set doesn´t not improve after a certain number of epochs. Otherwise, the results obtained in the model with 3 hidden layers do not justify introducing additional layers to the 2 hidden layer model (see Appendix 8.1.1.1 and Appendix 8.1.1.5).

On the other hand a model was trained defining the MAE as the loss function and its results are shown in the following table:

Model	***MAE value***
2 HL without label transformation	2.65
Mean imputation of the features	2.86
Median imputation of the features	2.54

Table 8. Comparison of the MAE error committed by the model with respect to the error made when performing the mean and median of the transcripts.

As can be seen in the table above, comparing the model error results with the error made when using the mean and median, the model obtained even worse

results than when using the median of the samples. The model trained with MAE as a loss function predicts expression levels close to zero when dealing with isoforms in which there is high inter-patient variability of expression (See Appendix 8.1.1.3).

Another interesting aspect is that, while using RMSE as a loss function, MAE is also reduced during training, since the loss function is being optimized; when MAE is defined as a loss function, the RMSE and MSE error are hardly reduced during training.

A trend that has been repeated using both RMSE and MAE as a loss function in the different methodologies is that a higher mean expression of transcripts is associated with a higher error. In addition, the model is not able to correctly predict the isoforms that are poorly expressed, which although they are the transcripts with the lowest relative error this is only because they have a smaller range of expression variability.

In the following modeling steps, the information of the gene expression was added to the model, knowing the dependency relationship between genes and transcripts.

Given the results obtained, it was decided to maintain the architecture with 2 hidden layers, the MSE as loss function and the application of a logarithmic transformation function to the transcripts before modeling in the next stage of model development.

The plots concerning to the evolution of the loss function during training, the information about the isoforms with the highest and lowest error in the model prediction and the relationship between model error and median expression for the different methodologies tested in this step can be found in Appendix 8.1.1.

4.1.2. Best optimizer selection with PyTorch

Regarding the results got using the model with two hidden layers with gene expression, below the Spearman's correlation between the predicted values and the theoretical values (total correlation), the final MSE loss and its evolution during training for the training and validation set, using different optimizers are shown:

Optimizer	*Total Correlation Training*	*Total Correlation Validation*	*Training Loss*	*Validation Loss*
AdamW	0.8872	0.8495	0.0733	0.1739
Adam	0.8830	0.8460	0.0748	0.1893
AdaGrad	0.8153	0.7883	0.2421	0.3849
SGD90	0.6726	0.6533	0.5373	0.6841
SGD70	0.4712	0.4382	0.9824	1.1760
SGD50	0.2773	0.2508	1.8420	2.0460
Adadelta	0.1152	0.1039	2.4560	2.5660
ASGD	0.1028	0.0920	2.4860	2.5910

Table 9. Results obtained in the Spearman's correlation between predicted and theoretical values and in the Mean Squared Error loss with training and validation data, using different optimizers.

Figure 13. Evolution of the training loss during training using different optimizers.

Figure 14. Evolution of the validation loss during training using different optimizers.

The best results were achieved using AdamW and Adam methods, with which we obtained the highest correlation values (see Figures below that show a linear dependency between the theoretical values and those predicted by the model) and the lowest error values in both training and validation data. Being the only models able to overcome a correlation of 0.8 in the validation set.

It is observed that, with the exception of these two algorithms and Adagrad (which has the smoothest loss function decay), with the rest of the optimizers convergence has not yet been achieved despite training the model for 3000 epochs. In Adam and AdamW the decrease of the loss function curve has been too pronounced, practically reaching convergence in the first epochs, which is not entirely desirable. Therefore, it would be interesting to use a lower learning rate in the following runs.

It is also interesting to mention that the higher the momentum applied in the SGD, i.e. the more we take into account past updates, the better results are obtained and the faster the loss function curve goes down.

Even though both have given us similar results, it was decided to continue using the AdamW for the next steps of the model development.

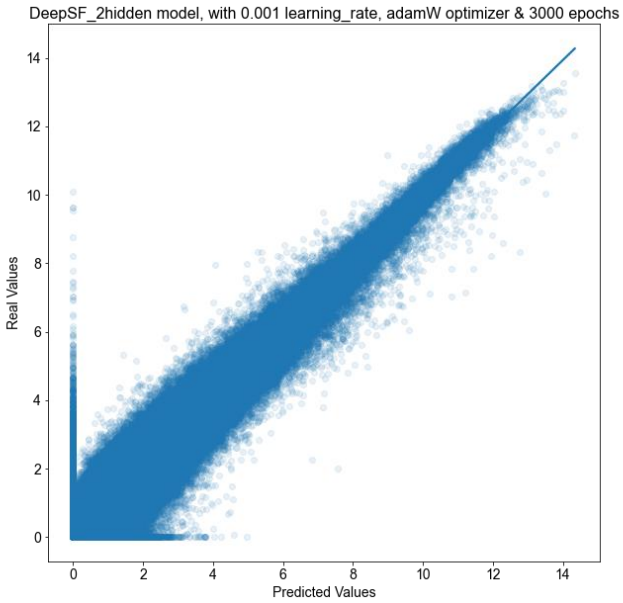


Figure 15. Predicted values against real values for the training data using the AdamW optimization algorithm.

Despite the fact that, as mentioned above, there is a high correlation between the values predicted by the model and the theoretical values, there is some presence of outliers, especially in the validation set.

Looking at the results obtained with the optimizers that have given a lower correlation value, the model has more problems in predicting especially the isoforms that have lower expression levels. See Appendix 8.1.2.4 to 8.1.2.8).

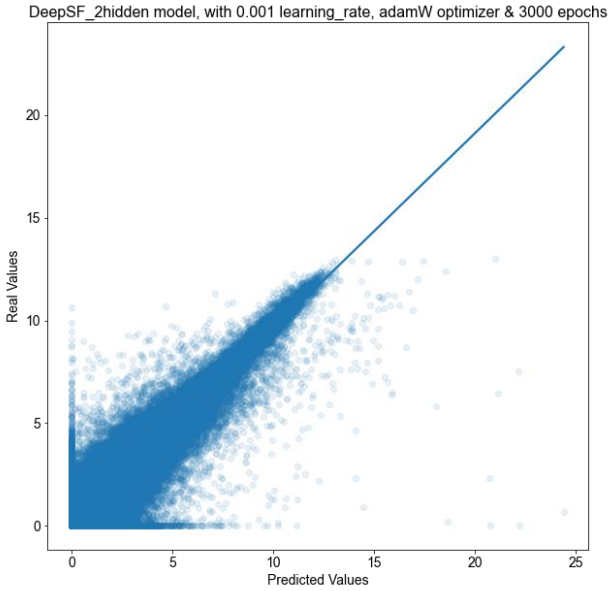


Figure 16. Predicted values against real values for the validation data using the AdamW optimization algorithm.

As for the correlation per biotype for the model optimized with AdamW the distributions of the correlation values have a greater variance, but there is a higher confluence in the range 0.6-1 for most of the biotypes, especially in training.

For training, correlation values have been calculated for 7829 isoforms (out of 10095 (4286 of them from protein coding genes). Where the mean of the protein-coding genes is 0.73 and the median is 0.8. The rest have turned out to be unassigned values because the Spearman's correlation is not defined when the standard deviation of one of the sets is 0, which could have occurred for isoforms with expression levels close to 0.

As for the validation set, the number of isoforms for which a correlation value has been obtained is 7610, with 4173 protein-coding genes and the correlation of protein-coding genes is lower, with a mean of 0.61 and a median of 0.67.

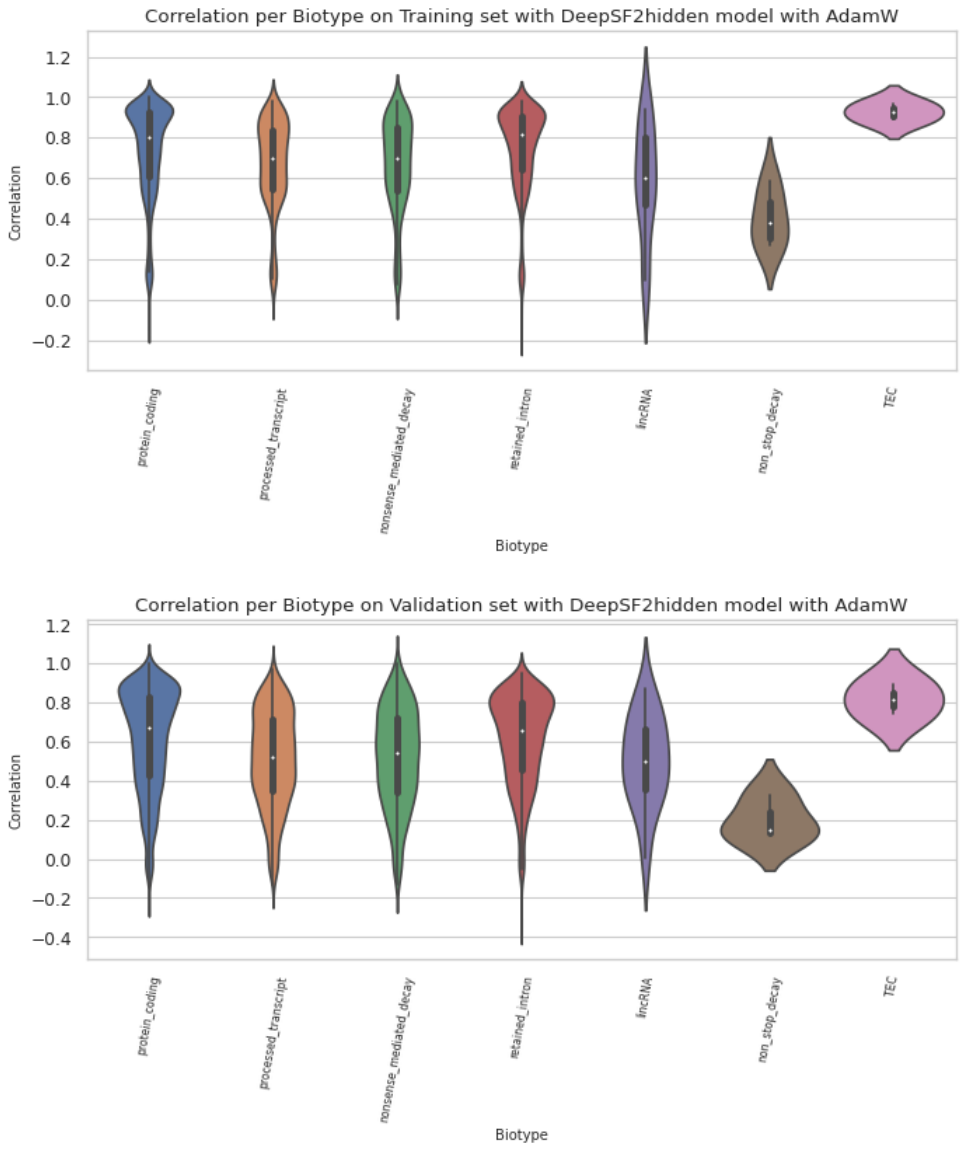


Figure 17. Spearman's correlation per biotype in training and validation data using DeepSF2hidden model with AdamW. Optimizer

For the TEC (To be Experimentally Confirmed) biotype, which has a correlation very close to 1 in training and around 0.8 in validation, we only have 2 samples in our dataset.

Transcripts of the MGA and NUTM2A genes appear several times among the best correlated using the Adam, AdamW and AdaGrad optimizers (see appendices 8.1.2.1, 8.1.2.2 and 8.1.2.3).

Using the AdamW optimizer, in particular, for the MGA-003 transcript, which is a protein-coding gene, correlations of 0.9 and 0.82 were obtained for the training data and validation data, respectively. For NUTM2A-001, on the other hand, which is another protein-coding gene, a correlation of 0.99 was achieved for training and 0.98 for validation. Other transcripts of these genes, in contrast, did not obtain such high correlation values.

4.1.3. Results in the training of the DeepAE model

From the gene expression of protein-coding genes a deep autoencoder (DeepAE) was trained, using samples from different adenocarcinomas. The final MSE, as well as the evolution of the loss function in the training and validation data, is presented below:

Model	*Training MSE Loss*	*Validation MSE Loss*
DeepAE	0.0066	0.0071

Table 10. The Mean Squared Error in training and validation data obtained in the training of the DeepAE model.

Table 11. Evolution of the training and validation Mean Squared Error loss during the training of the DeepAE model.

About the evolution of the loss function we see a very sharp decrease in the first epochs, but then the curve stabilizes and the learning becomes very slow for the remaining 800 epochs.

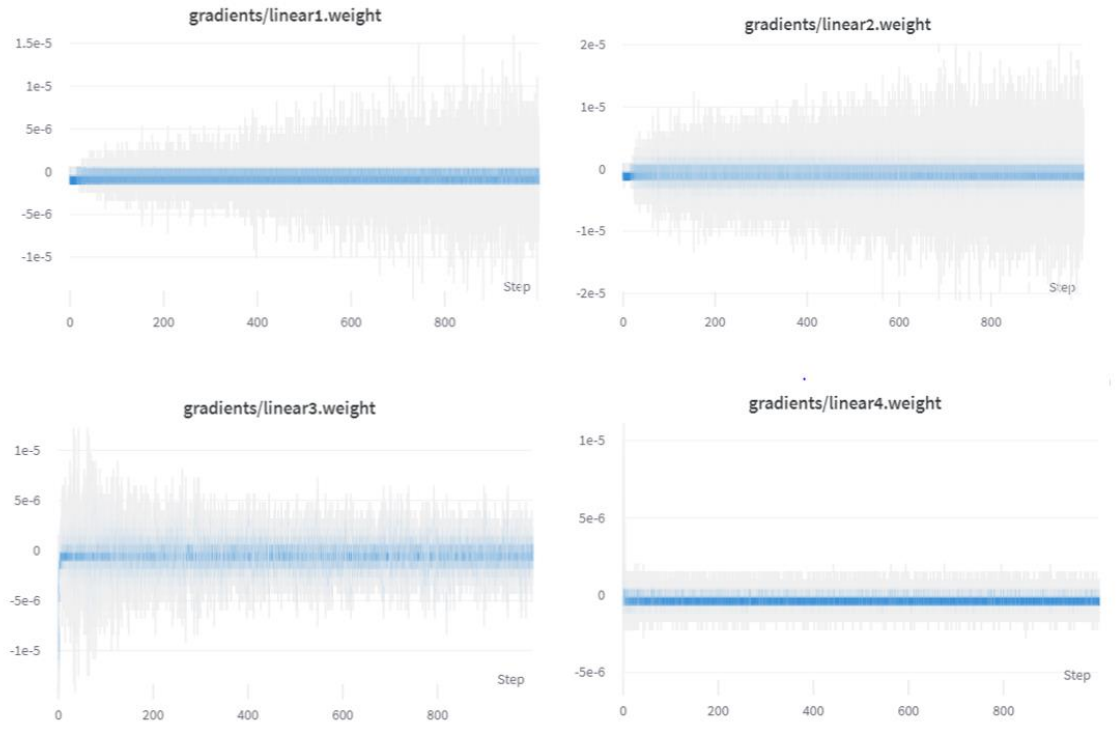


Figure 18. The evolution of the gradient values of the parameters during the training of the DeepAE model.

As for the evolution of the gradients during training, it can be observed that for linear layers 1 and 2 the variance increases during training (the gradient diverges), which is not convenient. This phenomenon causes the update step of the parameters to continue increasing, which can cause instability in the neural network.

However, in linear layers 3 and 4 the gradients do converge, coinciding especially with the phase in which the loss in training is reduced the most. This is important since the output of linear layer 3 will be the one connected to the final model and the appearance of divergent gradients also in these layers could cause poor prediction results in the ensemble model.

4.1.4. Results in the training of the DeepSF&AEensemble model

The following table shows the correlation results obtained in all the samples and the MSE error obtained in the training of the DeepSF&AEensemble model:

Optimizer	*Correlation Total Training*	*Correlation Total Validation*	*Training Loss*	*Validation Loss*
AdamW	0.8662	0.8470	0.09878	0.2293

Table 12. Results obtained in the Spearman's correlation between predicted and theoretical values and in the Mean Squared Error loss with training and validation data, using different adenocarcinoma samples.

Comparing the results obtained with this model with those achieved when training the DeepSF2hidden with gene expression model, it can be seen that the total correlation in the validation data has been slightly lower and the error higher. The latter could be explained, among other things, by the larger number of samples used from different tissues to train the model. Despite this, the correlation of the predicted values with respect to the theoretical values in the validation set exceeds 0.84, which is still positive (see figures for training and validation set). It is only surpassed in these statistics by the DeepSF2hidden model optimized with the AdamW algorithm. Although there is a presence of certain large outliers.

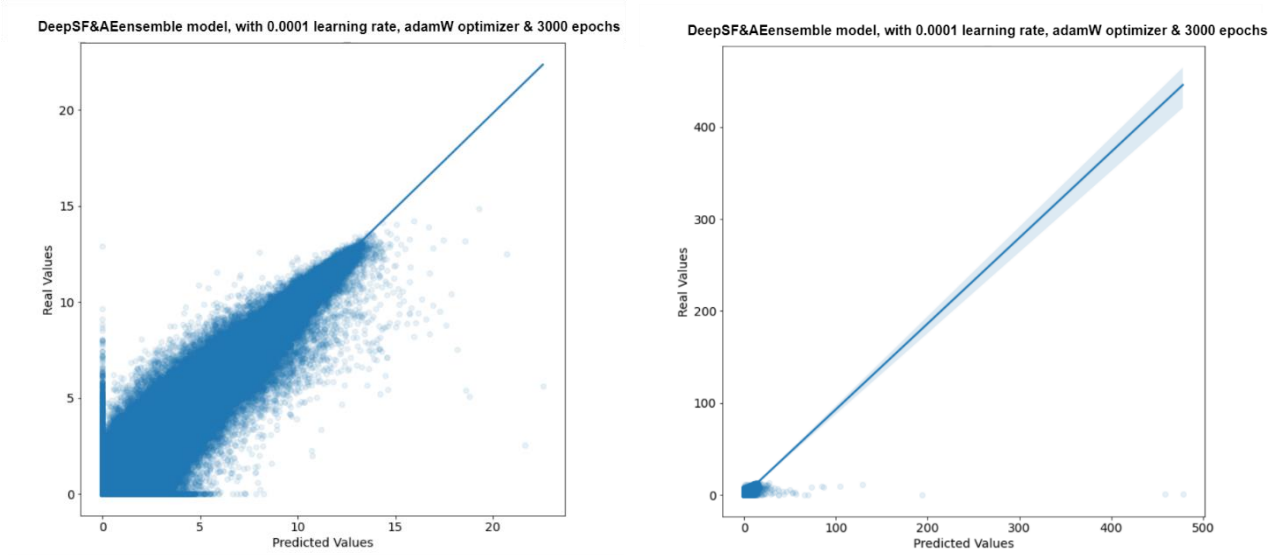


Figure 19. Predicted against theoretical values for the training and validation data using the DeepSF&AEensemble model.

It can be observed that the learning curve drops sharply in the first 200 epochs (despite having reduced the learning rate in this step to avoid it) and then remains mainly stable, reducing the error in the validation and training set slowly.

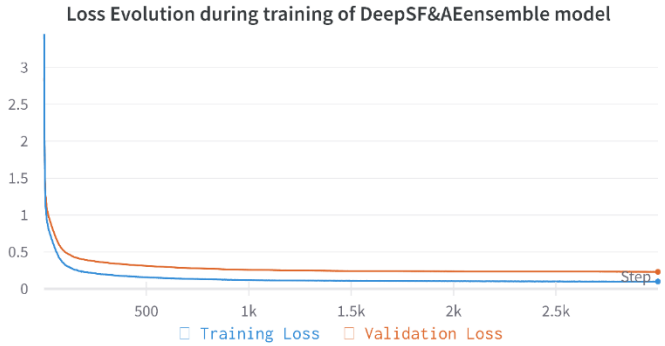


Figure 20. Evolution of the training and validation Mean Squared Error loss during training for the DeepSF&AEensemble model.

As for the evolution of the value of the gradients during the training of the model, in the first stages of training, in which there is a large drop in the error, the magnitude of the gradient in the linear layers and in the vector of weights

associated with the genes is greater, there is more variance in the values. In this stage of learning is normal to have larger parameter updates. Subsequently, the values converge as the evolution of the losses stabilises and the updates are smaller. The gradient evolution for layers 1, 2 and 3 and the associated genetic parameters are presented below. The rest of the parameter gradients and the value of the model parameters can be found in the appendix:

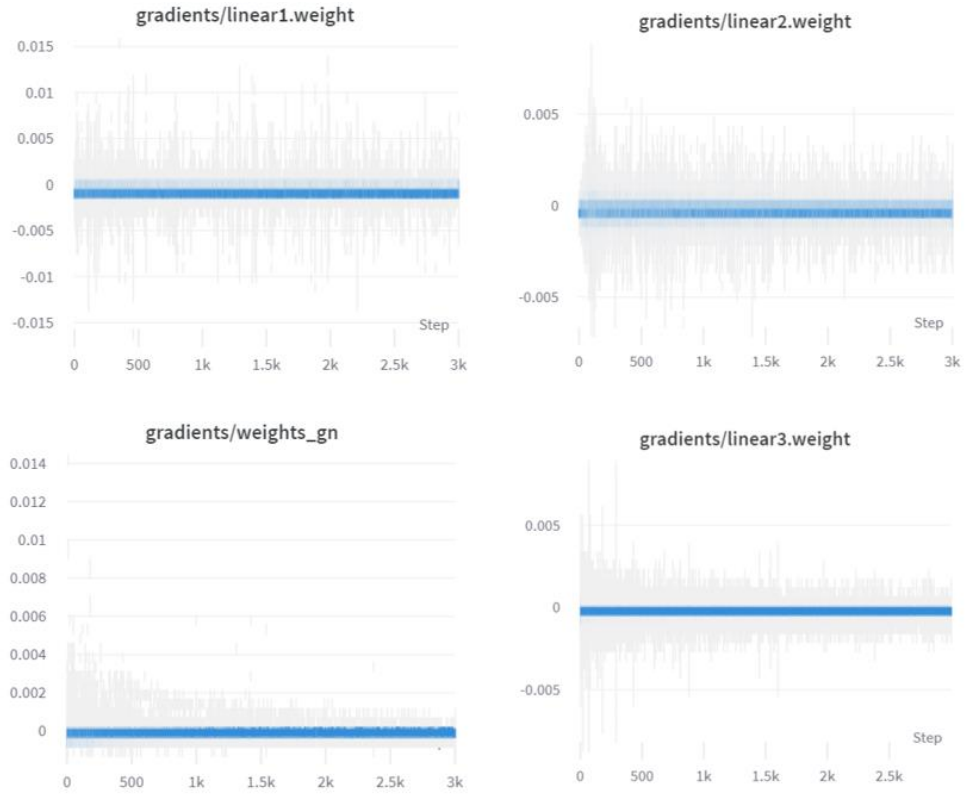


Figure 21. Evolution of the gradient values for the main model parameters with DeepSF&AEensemble model.

As for the correlation per individual transcript, this model has managed to increase the number of protein-coding genes for which correlation values have been calculated. It went from 7829 transcripts (4286 of them protein-coding genes) in the training data to 8313 (and 4657 protein-coding genes).In the validation set, on the other hand, we went from 7610 transcripts (and 4173 of them protein-coding genes) to 8433 (4720 protein-coding genes).

For this reason, in order to compare the correlation of the DeepSF2hidden with gene expression model and the DeepSF&AEensemble model using the same protein-coding genes, the intersection of the transcripts correlated in the training set and validation set of both models was plotted by biotype, giving a total of 7060 transcripts (and 3912 protein-coding genes).

Model	Sample size	Correlation	
DeepSF2hidden	$n = 3912$	$mean$	0.77
		$median$	0.83
DeepSF&AEensemble	$n = 3912$	$mean$	0.75
		$median$	0.81
DeepSF2hidden	$n = 4286$	$mean$	0.73
		$median$	0.80
DeepSF&AEensemble	$n = 4657$	$mean$	0.70
		$median$	0.76

Table 13. Spearman's correlation of the training data for protein-coding genes using DeepSF2hidden and DeepSF&AEensemble models.

Model	Sample size	Correlation	
DeepSF&AEensemble	$n = 3912$	$mean$	0.70
		$median$	0.75
DeepSF&AEensemble	$n = 4720$	$mean$	0.64
		$median$	0.70
DeepSF2hidden	$n = 3912$	$mean$	0.63
		$median$	0.69
DeepSF2hidden	$n = 4173$	$mean$	0.61
		$median$	0.67

Table 14. Spearman's correlation of the validation data for protein-coding genes using DeepSF2hidden and DeepSF&AEensemble models.

Comparing the two tables above, it can be seen that the DeepSF&AEensemble model has obtained worse results for the mean and median of protein-coding genes in the training data. However, with this set we have been able to: 1) increase the number of protein-coding genes for which we have obtained correlation data and 2) increase the correlation of protein-coding genes with a fix set of genes in the validation set. See the following two figures:

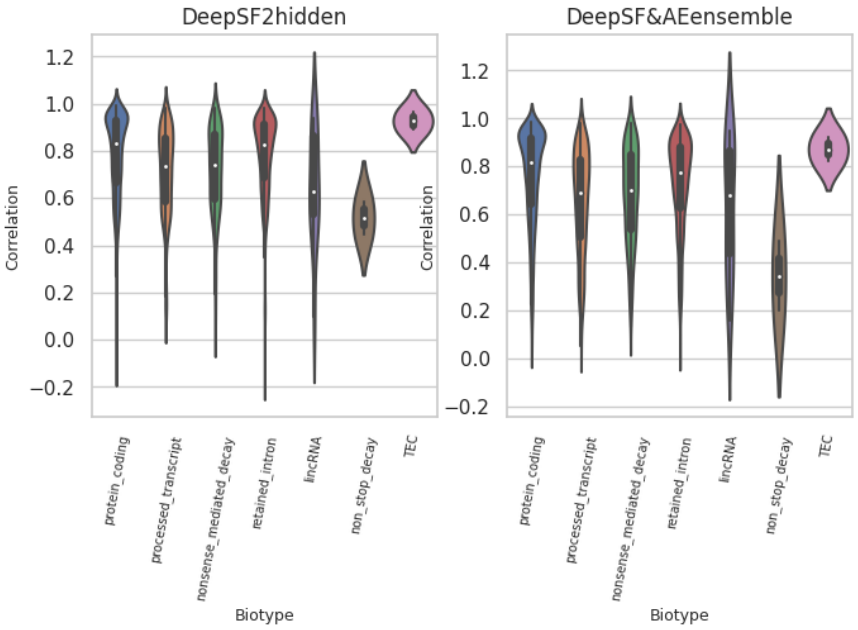


Figure 22. The Spearman's correlation per biotype in the training data for the same 7060 transcripts in DeepSF2hidden and DeepSF&AEensemble.

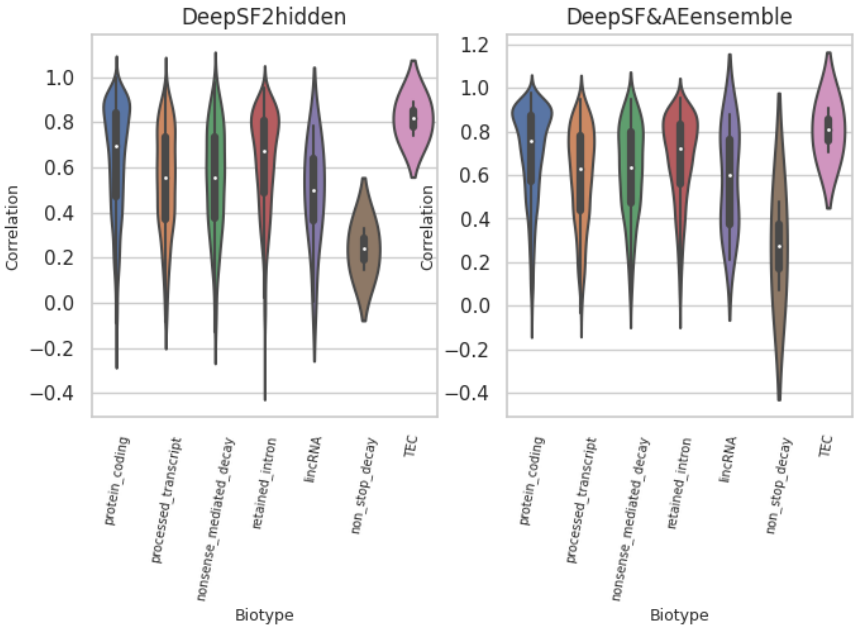


Figure 23. The Spearman's correlation per biotype in the validation data for the same 7060 transcripts in DeepSF2hidden and DeepSF&AEensemble.

4.1.5. Interpretation of the model parameters

The optimal threshold selected to create the binary matrix from the DeepLIFT scores was 0.0022, which resulted in a difference between the theoretical median and our binary matrix of 2 units.

To compare both binary matrices with the gene regulation data, a general confusion matrix and a confusion matrix for each SF gene were performed. The overall results show that only the 30% of SF-Gene regulation relationships (12349 out of 39861) were correctly predicted. While the model is able to predict the 74.2% of the non-SF-gene interactions, since most of the genes are not regulated by any SF. It is for this reason that the accuracy result is higher, but this does not mean that the performance of the model is better. The important thing is to be able to predict when regulation does occur. The value of the precision-recall metric makes us see this problem of the model to distinguish when a SF-gene regulation appears.

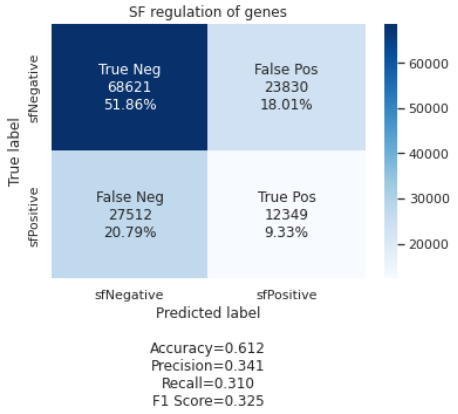


Figure 24. General confusion matrix between the theoretical and the predicted binary matrices.

To see this case, have a look at the following confusion matrix results of different SF genes:

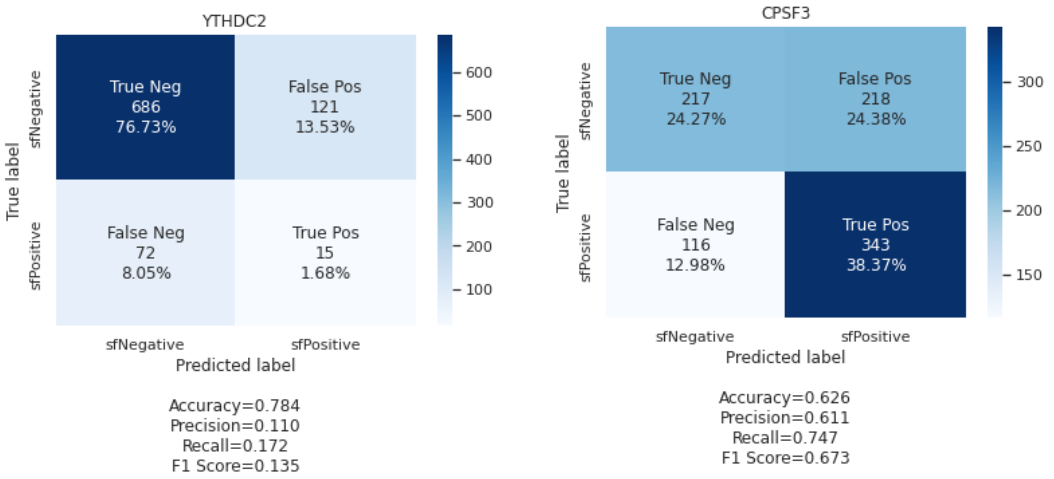


Figure 25. Confusion matrix for YTHDC2 and CPSF3 splicing factor genes.

In the first case, the incapacity of the model to predict the regulation SF-gene when there is a SF that regulates few genes is observed. On the other hand, with CPSF3, the 74.72% of the regulations have been identified, but suffering from a reduction in accuracy.

5. CONCLUSIONS

In this project, a DNN model capable of predicting transcriptome expression of cancer-related genes from gene expression data of splicing factors and genome expression has been trained, using samples from different adenocarcinomas.

Regarding the optimization process of the model architecture and data processing, the use of 2 hidden layers, a logarithmic transformation in base 2 of the transcript expression, the use of MSE as loss function, AdamW as an optimizer and a learning rate of between 10^{-3} have given the best results.

With this training configuration, a Spearman's correlation of 0.84 was achieved between the theoretical validation values and those predicted by the model. However, the correlation coefficient for each transcript and biotype did show a higher variance, with 0.61 being the mean and 0.67 the median correlation for the protein-coding genes in the validation set.

On the other hand, the cellular context in the prediction of transcript expression was studied by training a DeepAE with different types of adenocarcinomas. The evolution of the gradients for the first layers suggests that we should have introduced a stronger regularization to control the divergence of the gradient values during training. However, the last linear layers did manage to converge the value of the gradients as the network learned. In summary, we have not been able to optimize the autoencoder architecture, despite having used the architecture presented in (Sanjiv K. Dwivedi, 2020) to capture information from

gene expression. Given how fast the model converges, the number of parameters in the hidden layers could have been increased.

The output of the last hidden layer was connected to the last linear layer of the model with gene expression. Correlation values similar to the DeepSF2hidden model have been obtained but the correlation of the model between the predictions and the theoretical values in the validation set has not been improved. However, in terms of correlation by biotype using the same set of 7060 transcripts, the mean and median correlation has increased with respect to the previous model for the validation set. From a mean of 0.63 to a 0.7 and from a median of 0.69 to 0.75.

Regarding the interpretation of the model parameters, the results were not significant, with only 30% of the SF-Gene regulatory interactions identified. Among other factors limiting the interpretation just to gene regulation could have been no effective. Since a SF might not regulate all the transcripts of the same gene or positively regulate some and negatively regulate others.

In addition, the model may not have been fully optimized or may not have correctly interpreted the DeepLIFT scores through the choice of the threshold.

6. FUTURE LINES

As for future lines, it is proposed to optimize the architecture of the autoencoder using more parameters and more biotypes, and to study the positive and negative regulation of SFs with genes and transcripts by means of Graph Neural Networks (GNN).

Could be also interesting to calculate the isoform usage, i.e. the percentage of each isoform expressed in each gene, and being this is the variable to be predicted.

7. APPENDIX

7.1. DEEPSF2HIDDEN WITH GENE EXPRESSION

7.1.1. Selection of the number of hidden layers, the loss function and trying different pipelines to counteract variability in isoform expression

7.1.1.1. *RMSE as loss function with a 2 hidden layer (183-82) model without label transformation*

In the following figure, the evolution of the loss function (RMSE) during training as well as the different metrics calculated using training and validation data (colored in blue and orange respectively) can be seen:

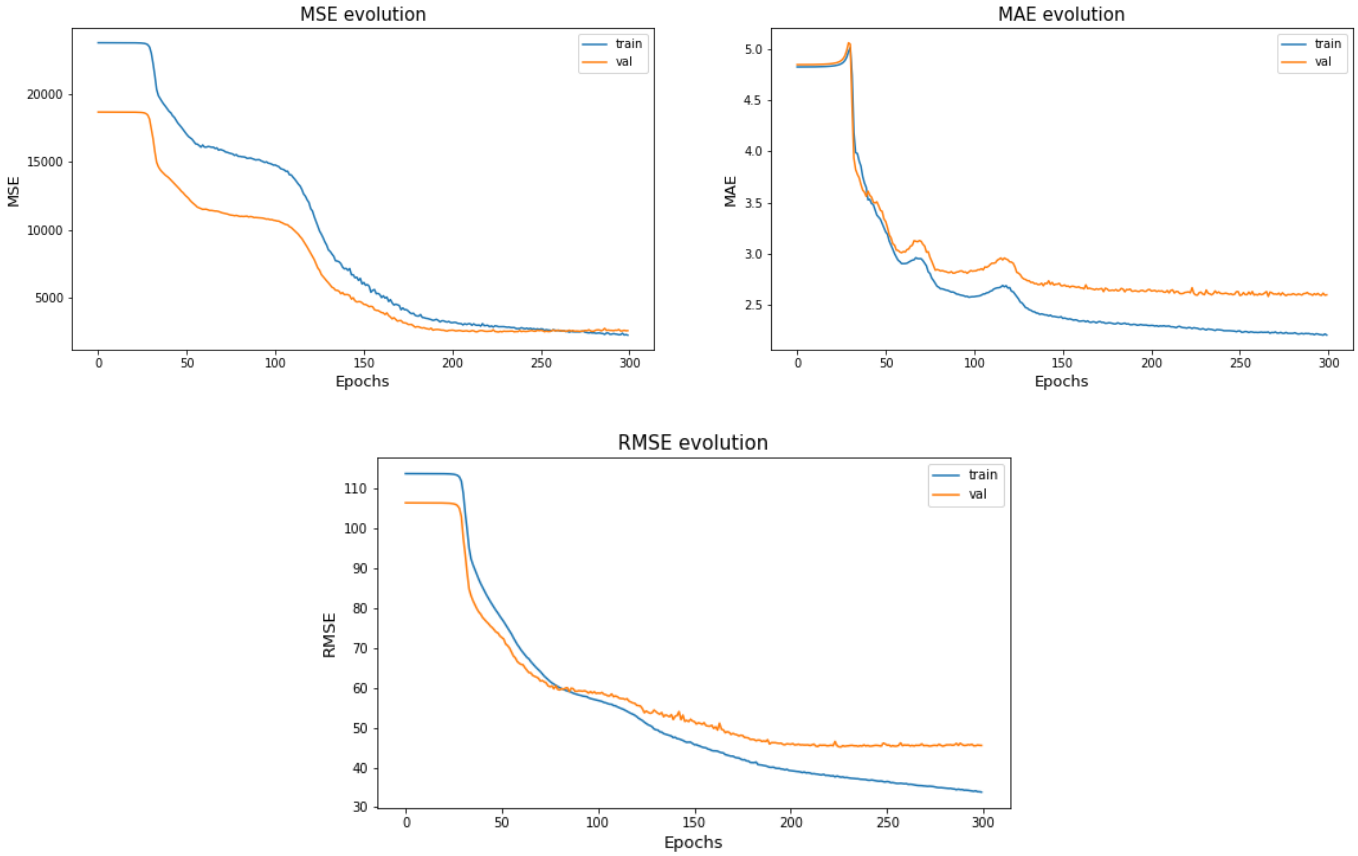


Figure 26. Evolution of RMSE, MAE and MSE metrics for the model with RMSE as loss function with 2 hidden layer (183-82) without label transformation.

Transcript ID	*Gene ID*	*Transcrip Name*	*Gene Name*	*Biotype*	*RMSE value*
ENST00000390237	ENSG00000211592	IGKC-001	IGKC	IG_C_gene	9429.85
ENST00000490232	ENSG00000274012	Metazoa_SRP.45-201	Metazoa_SRP	misc_RNA	7017.37
ENST00000390549	ENSG00000211896	IGHG1-001	IGHG1	IG_C_gene	6486.37
ENST00000618786	ENSG00000276168	RN7SL1-201	RN7SL1	misc_RNA	5213.59
ENST00000373025	ENSG00000096088	PGC-001	PGC	protein_coding	4911.89

Table 15. Isoforms with the highest RMSE value in the prediction for the model with RMSE as loss function with 2 hidden layer (183-82) without label transformation.

Transcript ID	*Gene ID*	*Transcrip Name*	*Gene Name*	*Biotype*	*RMSE value*
ENST00000628689	ENSG00000196628	TCF4-069	TCF4	nonsense_mediated_decay	0.43
ENST00000503737	ENSG00000164292	RHOBTB3-011	RHOBTB3	protein_coding	0.43
ENST00000458174	ENSG00000152936	LMNTD1-001	LMNTD1	protein_coding	0.45
ENST00000618928	ENSG00000277526	RP11-378A12.1-001	RP11-378A12.1	lincRNA	0.45
ENST00000521971	ENSG00000164934	DCAF13-003	DCAF13	protein_coding	0.45

Table 16. Isoforms with the lowest RMSE value in the prediction for the model with RMSE as loss function with 2 hidden layer (183-82) without label transformation.

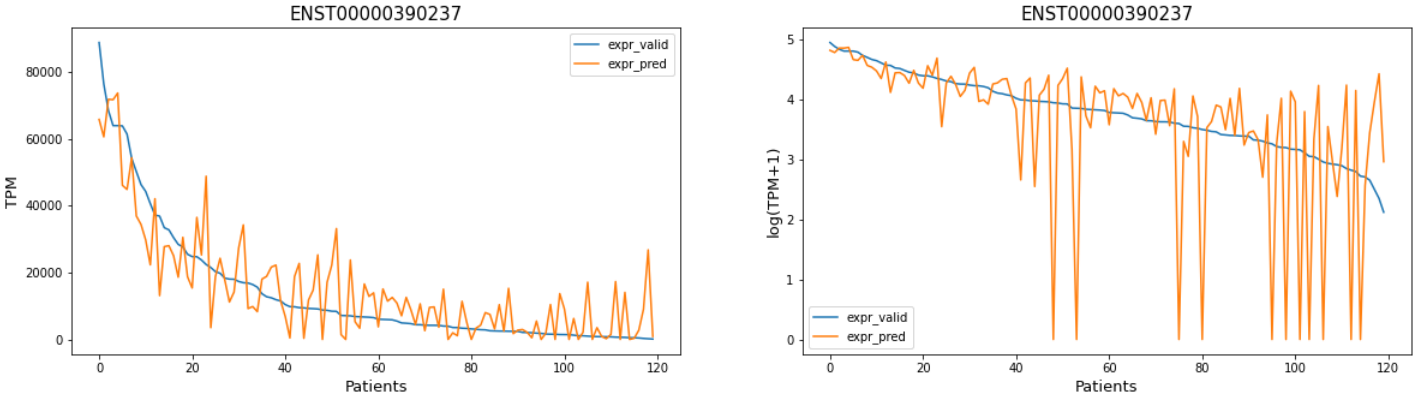


Figure 27. Theoretical vs model-predicted expression values (colored in blue and orange, respectively) in tpm and log(tpm+1) for the isoform with highest RMSE using the model with RMSE as loss function with 2 hidden layer (183-82) without label transformation.

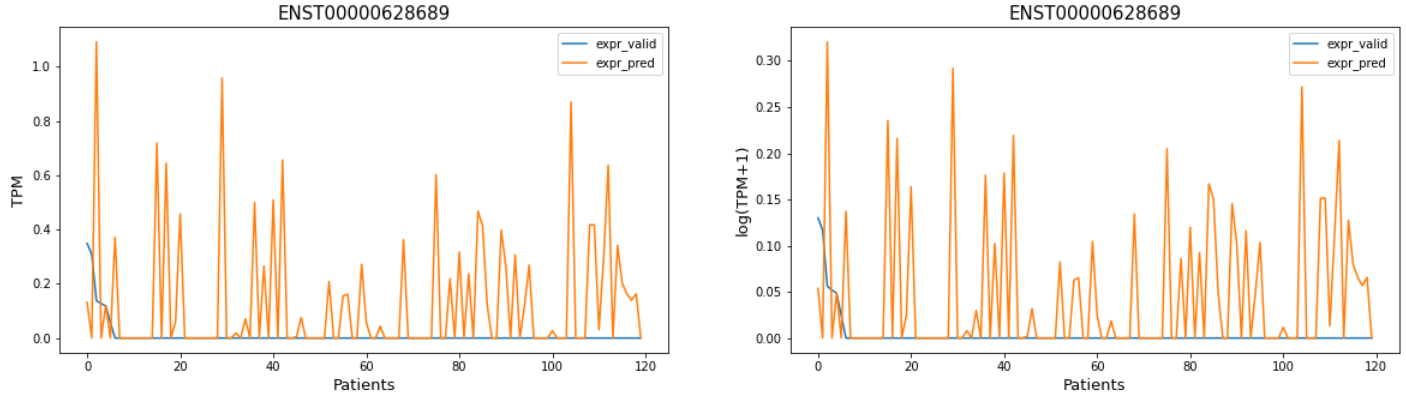


Figure 28. Theoretical and model-predicted expression values (colored in blue and orange, respectively) in tpm and log(tpm+1) for the isoform with lowest RMSE using the model withl RMSE as loss function with 2 hidden layer (183-82) without label transformation.

Next, the RMSE of the model in log10(x+1) is plotted in each of the transcripts against the median of the expression of each of them in log10(x+1):

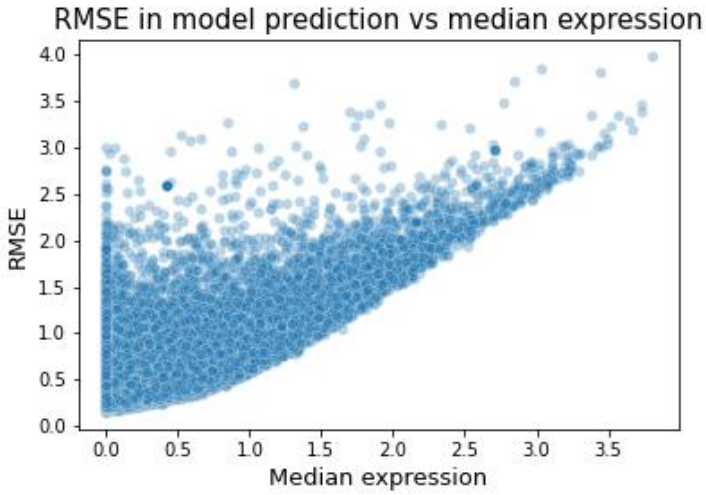


Figure 29. The RMSE in model prediction in log10(x+1) against the real median expression for the model with RMSE as loss function with 2 hidden layer (183-82) without label transformation.

7.1.1.2. *RMSE as loss function with a 2 hidden layer (183-82) model without label transformation but adding weight values to the loss function.*

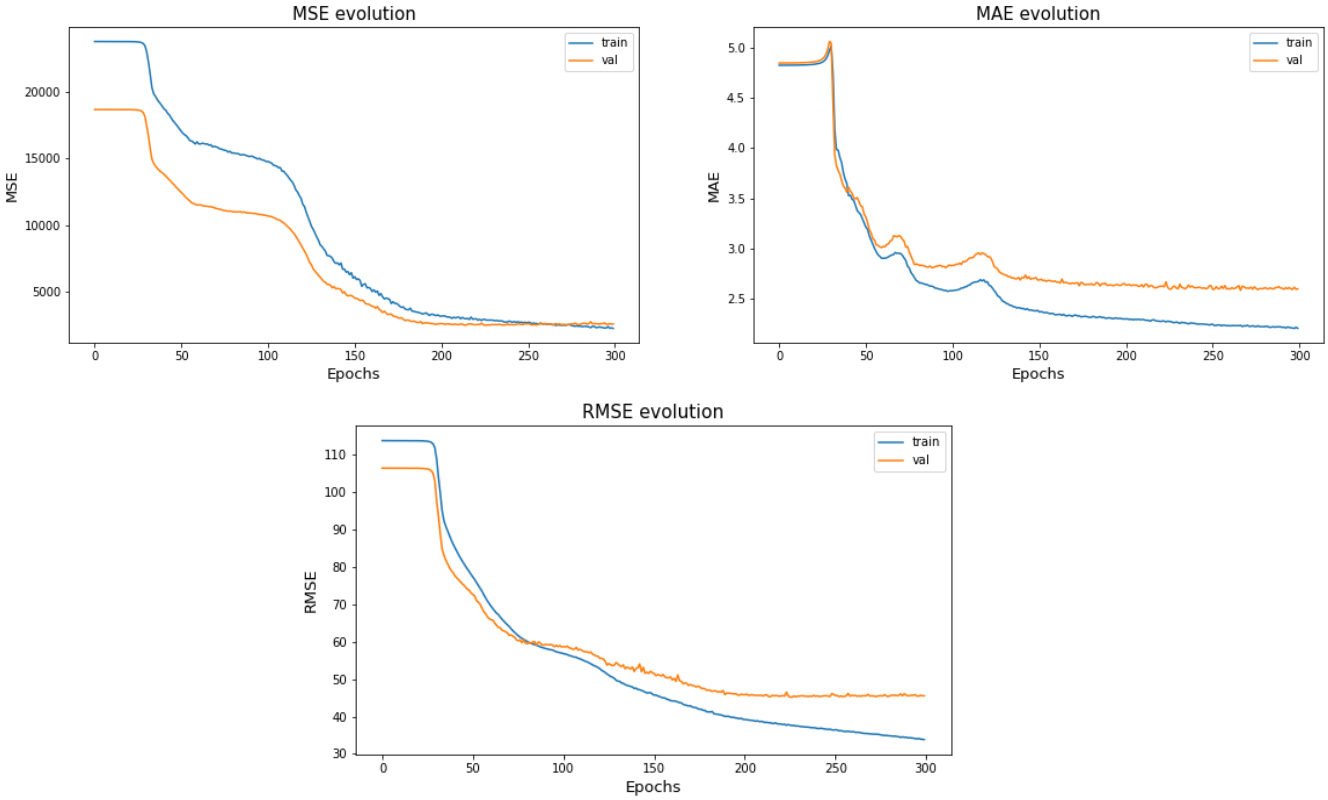


Figure 30. The evolution of RMSE, MAE and MSE for the model with RMSE as loss function with 2 hidden layer (183-82) without label transformation but adding weight values to the loss.

Transcript ID	Gene ID	Transcrip Name	Gene Name	Biotype	RMSE value
ENST00000390237	ENSG00000211592	IGKC-001	IGKC	IG_C_gene	9235.06
ENST00000490232	ENSG00000274012	Metazoa_SRP.45-201	Metazoa_SRP	misc_RNA	7007.01
ENST00000390549	ENSG00000211896	IGHG1-001	IGHG1	IG_C_gene	6276.89
ENST00000618786	ENSG00000276168	RN7SL1-201	RN7SL1	misc_RNA	5351.31
ENST00000373025	ENSG00000096088	PGC-001	PGC	protein_coding	4910.92

Table 17. Isoforms with the highest RMSE value in the model prediction with RMSE as loss function with 2 hidden layer (183-82) without label transformation but adding weight values to the loss.

Transcript ID	Gene ID	Transcrip Name	Gene Name	Biotype	RMSE value
ENST00000502332	ENSG00000206120	EGFEM1P-004	EGFEM1P	processed_transcript	0.46
ENST00000610628	ENSG00000249948	GBA3-201	GBA3	protein_coding	0.47
ENST00000490169	ENSG00000100221	JOSD1-005	JOSD1	processed_transcript	0.48
ENST00000519578	ENSG00000104299	INTS9-016	INTS9	processed_transcript	0.48
ENST00000588259	ENSG00000099203	TMED1-005	TMED1	nonsense_mediated_decay	0.48

Table 18. Isoforms with the lowest RMSE value in the model prediction with RMSE as loss function with 2 hidden layer (183-82) without label transformation but adding weight values to the loss.

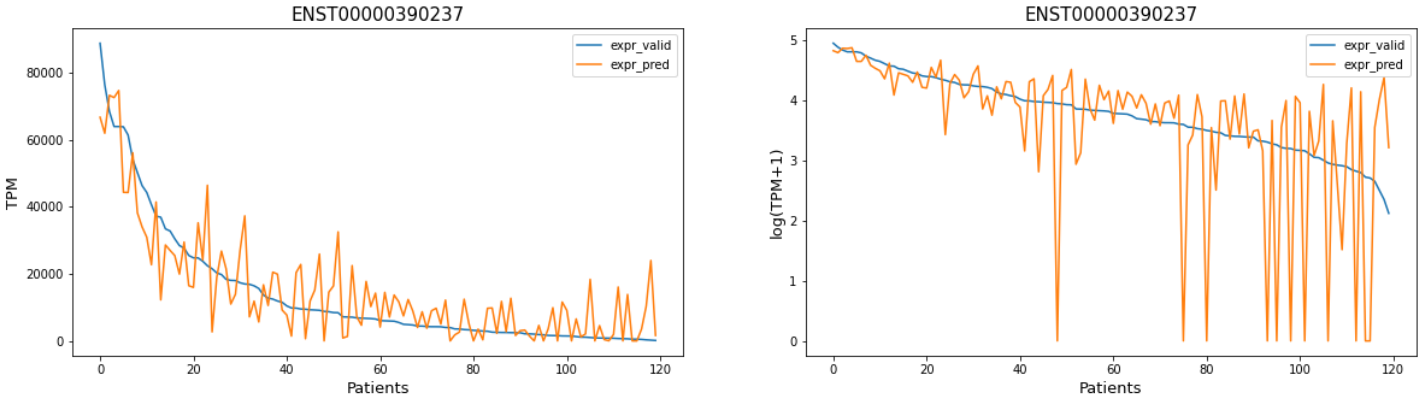


Figure 31. Theoretical and model-predicted expression values (colored in blue and orange, respectively) in tpm and log(tpm+1) for the isoform with highest RMSE using the model with RMSE as loss function with 2 hidden layer (183-82) without label transformation but adding weight values to the loss.

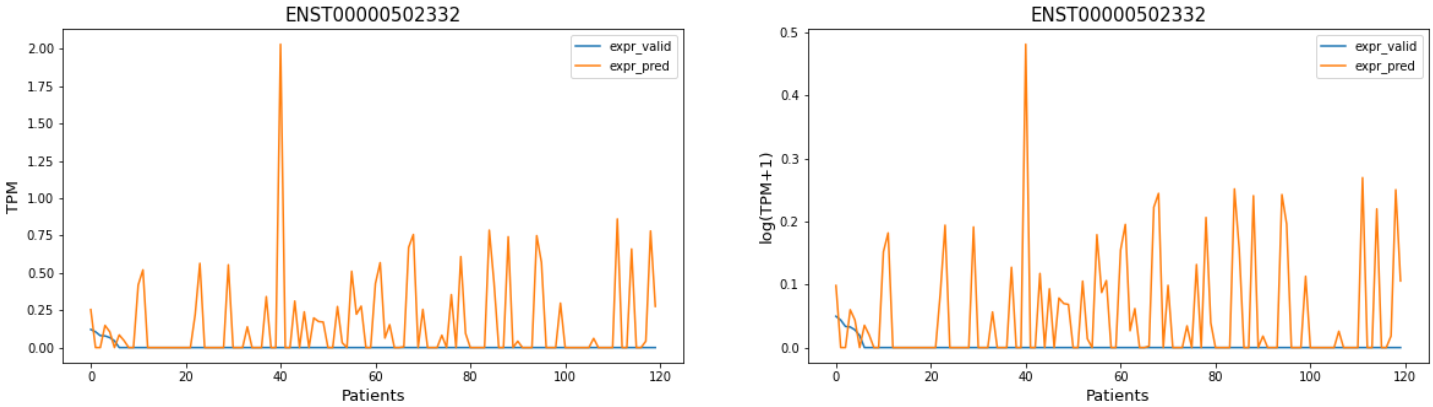


Figure 32. Theoretical and model-predicted expression values (colored in blue and orange, respectively) in tpm and log(tpm+1) for the isoform with lowest RMSE using the model with RMSE as loss function with 2 hidden layer (183-82), without label transformation but adding weight values to the loss.

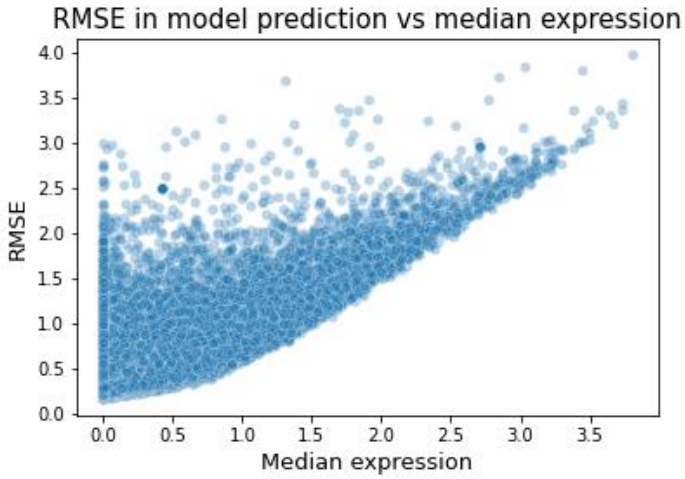


Figure 33. The RMSE in model prediction in log10(x+1) against the real median expression for the model with RMSE as loss function with 2 hidden layer (183-82), without label transformation but adding weight values to the loss.

7.1.1.3. *MAE loss function with 2 hidden layer model (183-82) without label transformation*

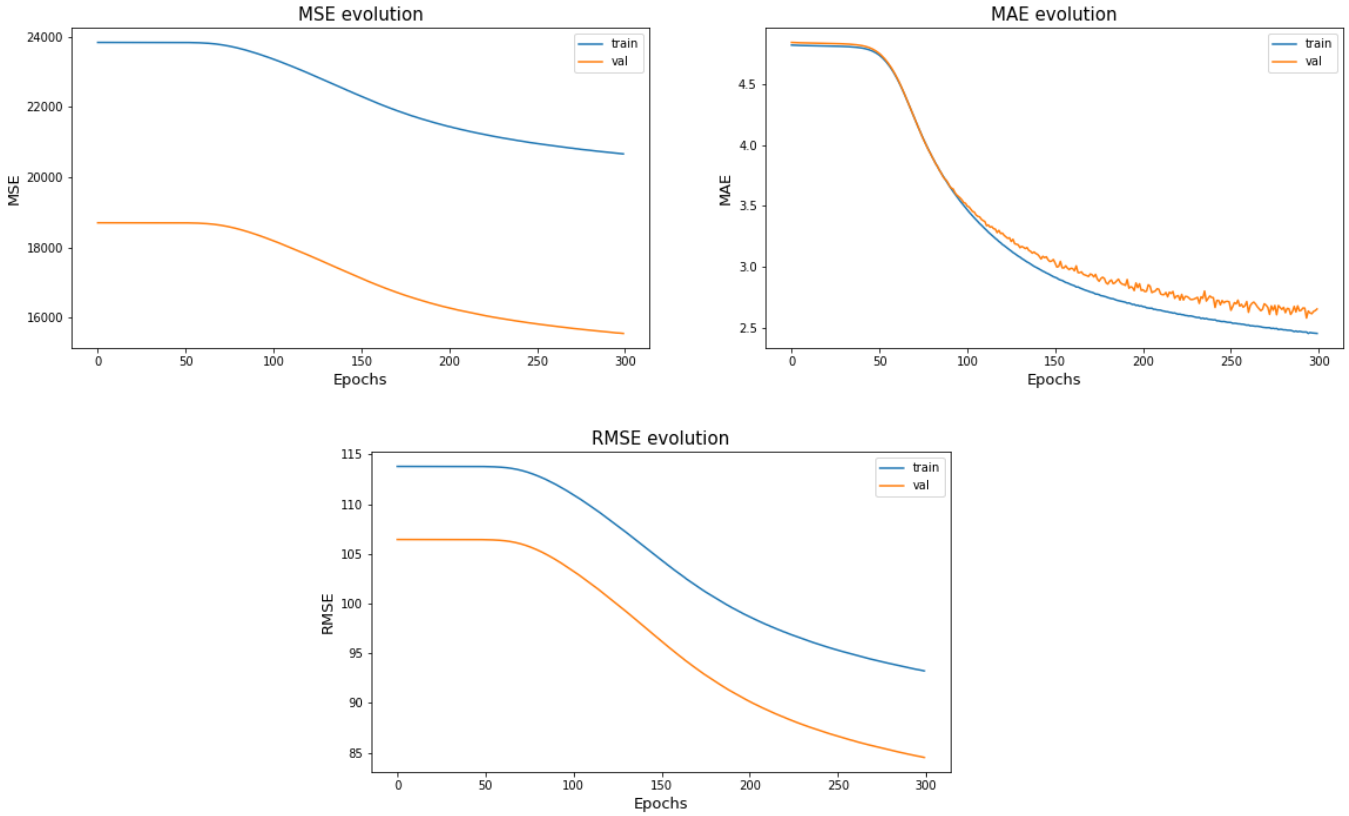


Figure 34. The evolution of RMSE, MAE and MSE for the model with MAE as loss function with 2 hidden layers (183-82) without label transformation.

Transcript ID	Gene ID	Transcrip Name	Gene Name	Biotype	MAE value
ENST00000390237	ENSG00000211592	IGKC-001	IGKC	IG_C_gene	12246.14
ENST00000390549	ENSG00000211896	IGHG1-001	IGHG1	IG_C_gene	6127.29
ENST00000490232	ENSG00000274012	Metazoa_SRP.45-201	Metazoa_SRP	misc_RNA	5358.89
ENST00000618786	ENSG00000276168	RN7SL1-201	RN7SL1	misc_RNA	4945.17
ENST00000331825	ENSG00000087086	FTL-001	FTL	protein_coding	4082.01

Table 19. Isoforms with the highest RMSE value in the model prediction with MAE as loss function with 2 hidden layers (183-82) without label transformation.

It is observed that even changing the loss function, the worst predictions of the model continue to be practically the same transcripts, which are those with a higher expression.

Transcript ID	Gene ID	Transcrip Name	Gene Name	Biotype	MAE value
ENST00000395854	ENSG00000073146	MOV10L1-004	MOV10L1	nonsense_mediated_decay	0.06
ENST00000611394	ENSG00000186654	PRR5-201	PRR5	protein_coding	0.07
ENST00000531352	ENSG00000078124	ACER3-008	ACER3	nonsense_mediated_decay	0.09
ENST00000517218	ENSG00000253027	SNORA70.26-201	SNORA70	snoRNA	0.09
ENST00000489317	ENSG00000132693	CRP-005	CRP	processed_transcript	0.09

Table 20. Isoforms with lowest MAE value in the model prediction with MAE as loss function with 2 hidden layer model (183-82) without label transformation.

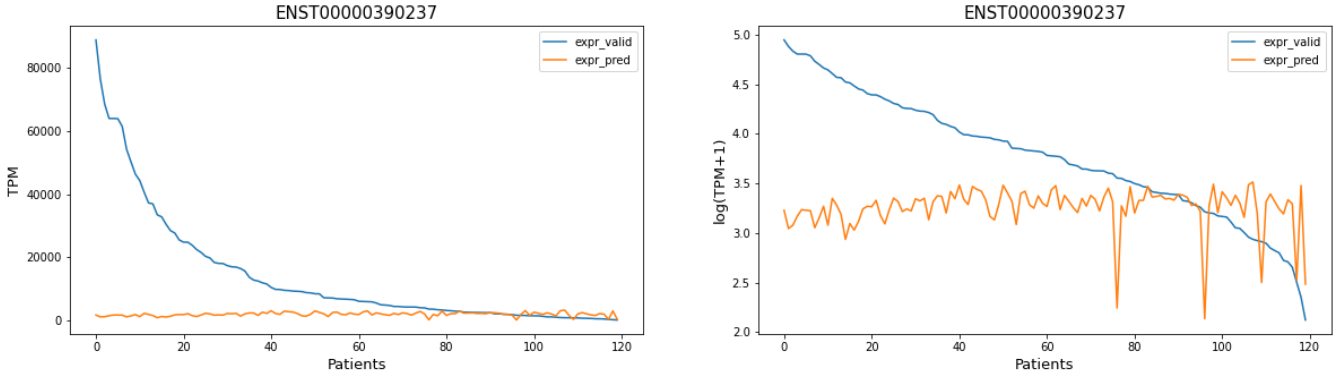


Figure 35. Theoretical and model-predicted expression values (colored in blue and orange, respectively) in tpm and log(tpm+1) for the isoform with highest RMSE using the model with MAE as loss function with 2 hidden layer (183-82), without label transformation.

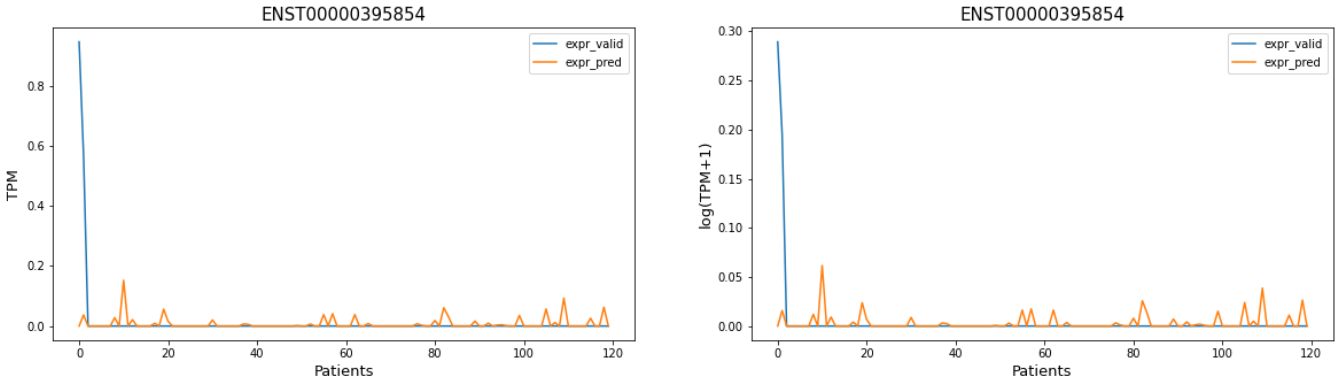


Figure 36. Theoretical and model-predicted expression values (colored in blue and orange, respectively) in tpm and log(tpm+1) for the isoform with lowest RMSE using the model with MAE as loss function with 2 hidden layers, without label transformation.

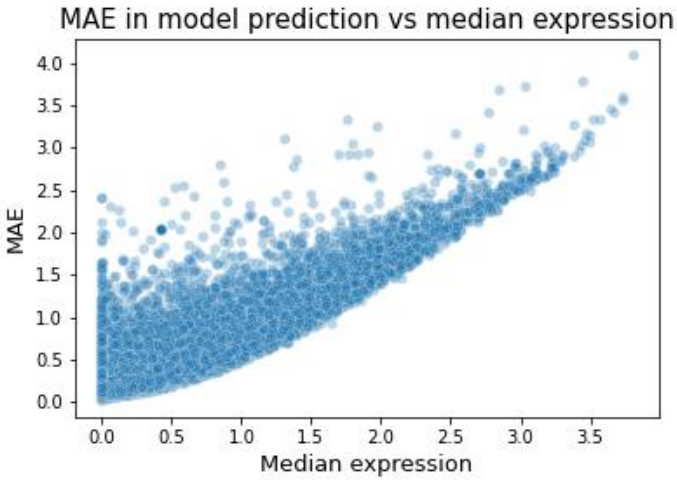


Figure 37. The MAE in model prediction in log10(x+1) against the real median expression for the model with MAE as loss function with 2 hidden layers, without label transformation.

7.1.1.4. *RMSE loss function with 2 hidden layer (183-82) and the application of a transformation function to the label*

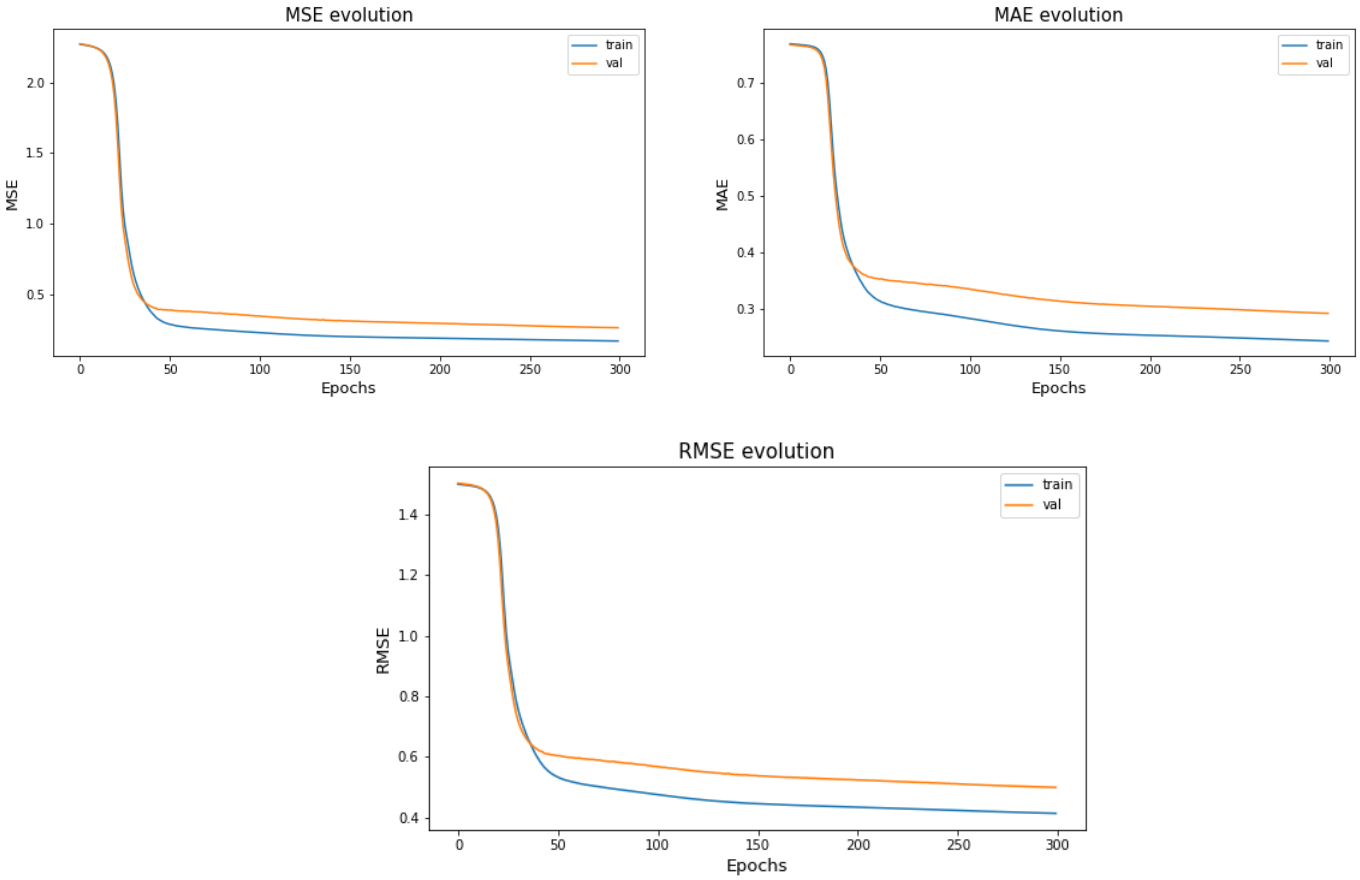


Figure 38. Evolution of RMSE, MAE and MSE for the model with RMSE as loss function with 2 hidden layers (183-82) and applying a transformation function to the labels.

Transcript ID	Gene ID	Transcrip Name	Gene Name	Biotype	RMSE value
ENST00000518615	ENSG00000168484	SFTPC-009	SFTPC	retained_intron	3.55
ENST00000278282	ENSG00000149021	SCGB1A1-001	SCGB1A1	protein_coding	3.53
ENST00000296695	ENSG00000164266	SPINK1-001	SPINK1	protein_coding	3.48
ENST00000372325	ENSG00000185303	SFTPA2-001	SFTPA2	protein_coding	3.47
ENST00000430575	ENSG00000129824	RPS4Y1-002	RPS4Y1	protein_coding	3.47

Table 21. Isoforms with the highest RMSE value in the model prediction with RMSE as loss function with 2 hidden layers and with label transformation.

Transcript ID	Gene ID	Transcrip Name	Gene Name	Biotype	RMSE value
ENST00000599080	ENSG00000231933	CTA-125H2.2-012	CTA-125H2.2	antisense	0.025
ENST00000490799	ENSG00000112053	SLC26A8-001	SLC26A8	protein_coding	0.026
ENST00000471303	ENSG00000007372	PAX6-014	PAX6	processed_transcript	0.027
ENST00000582631	ENSG00000265845	RP11-20B24.5-002	RP11-20B24.5	antisense	0.027
ENST00000615833	ENSG00000236922	LINC01378-005	LINC01378	lincRNA	0.027

Table 22. Isoforms with the lowest RMSE value in the model prediction with RMSE as loss function with 2 hidden layers and with label transformation.

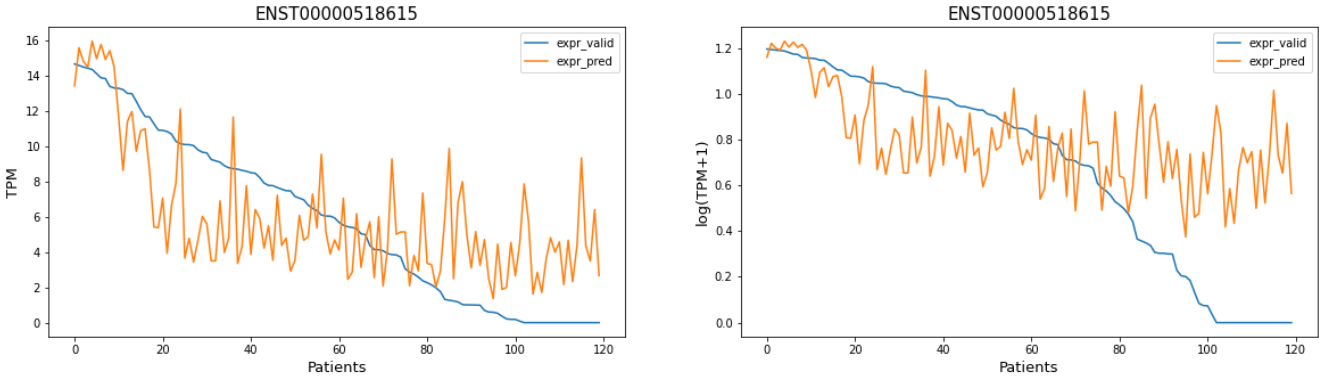


Figure 39. Theoretical and model-predicted expression values (colored in blue and orange, respectively) in tpm and log(tpm+1) for the isoform with the highest RMSE using the model with RMSE as loss function with 2 hidden layers and with label transformation.

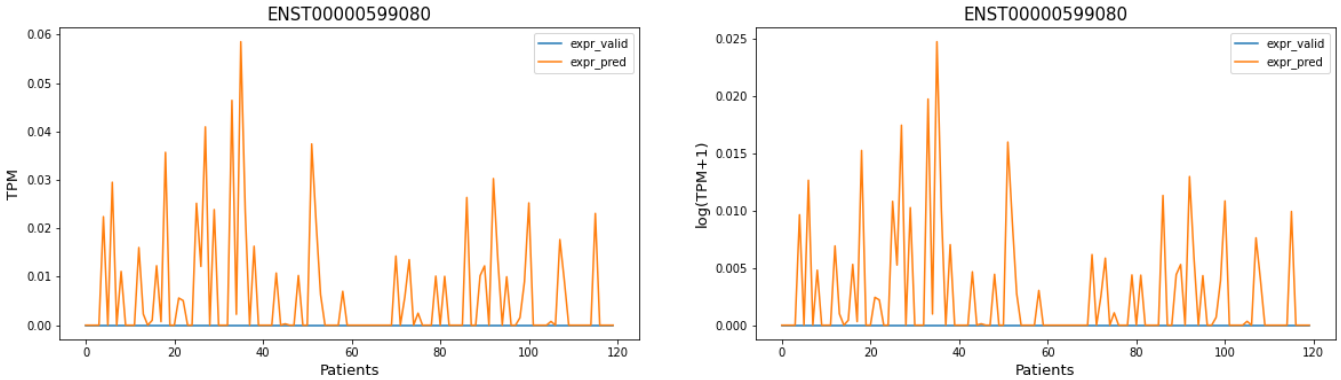


Figure 40. Theoretical and model-predicted expression values (colored in blue and orange, respectively) in tpm and log(tpm+1) for the isoform with the lowest RMSE using the model with RMSE as loss function with 2 hidden layers and with label transformation.

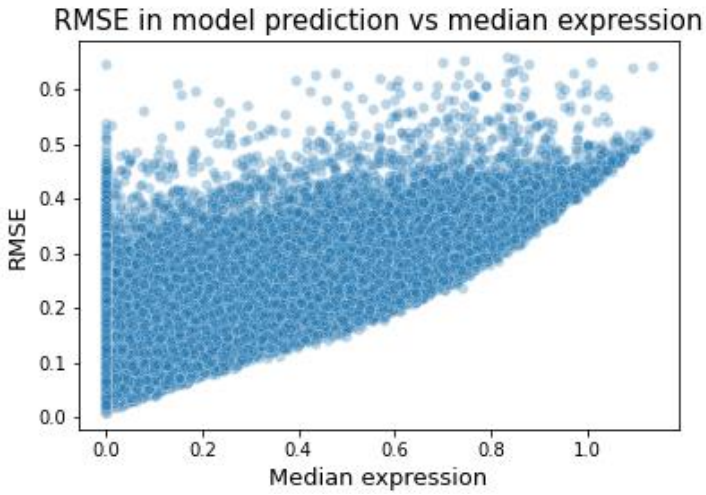


Figure 41. The RMSE in model prediction in log10(x+1) against the real median expression for the model with RMSE as loss function with 2 hidden layers and with label transformation.

7.1.1.5. *RMSE loss function with 3 hidden layer model (256-128-64) without label transformation.*

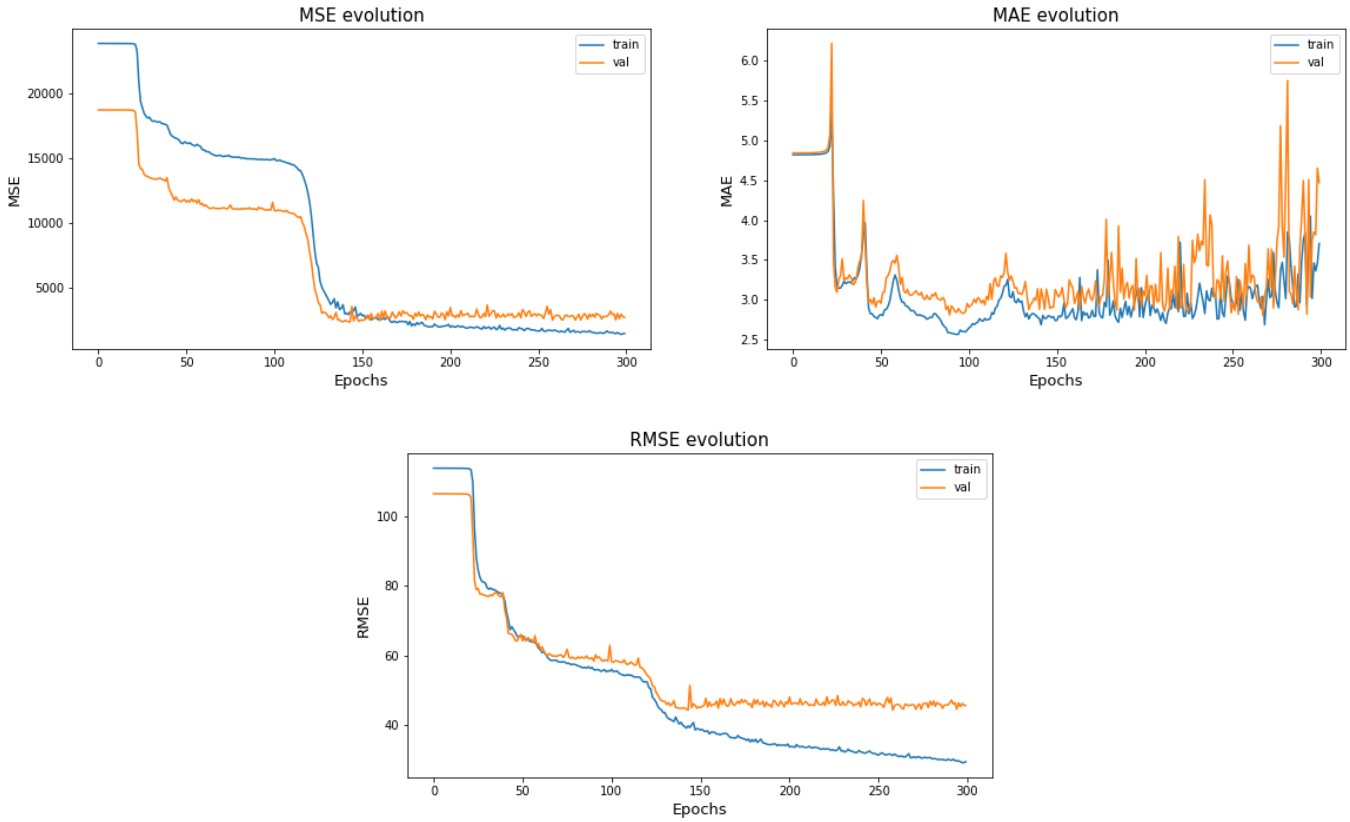


Figure 42. Evolution of RMSE, MAE and MSE for the model with RMSE as loss function with 3 hidden layers and without label transformation.

Transcript ID	Gene ID	Transcrip Name	Gene Name	Biotype	RMSE value
ENST00000390237	ENSG00000211592	IGKC-001	IGKC	IG_C_gene	9264.48
ENST00000490232	ENSG00000274012	Metazoa_SRP.45-201	Metazoa_SRP	misc_RNA	8355.62
ENST00000390549	ENSG00000211896	IGHG1-001	IGHG1	IG_C_gene	5940.95
ENST00000618786	ENSG00000276168	RN7SL1-201	RN7SL1	misc_RNA	5710.32
ENST00000373025	ENSG00000096088	PGC-001	PGC	protein_coding	5433.95

Table 23. Isoforms with the highest RMSE value in the model prediction with RMSE as loss function with 3 hidden layers and without label transformation.

Transcript ID	Gene ID	Transcrip Name	Gene Name	Biotype	RMSE value
ENST00000585882	ENSG00000232295	RP11-154D6.1-015	RP11-154D6.1	antisense	0.50
ENST00000469506	ENSG00000182606	TRAK1-008	TRAK1	retained_intron	0.51
ENST00000381747	ENSG00000139496	NUP58-009	NUP58	protein_coding	0.54
ENST00000397029	ENSG00000132170	PPARG-015	PPARG	protein_coding	0.55
ENST00000549315	ENSG00000185046	ANKS1B-029	ANKS1B	processed_transcript	0.56

Table 24. Isoforms with the lowest RMSE value in the model prediction with RMSE as loss function with 3 hidden layers and without label transformation.

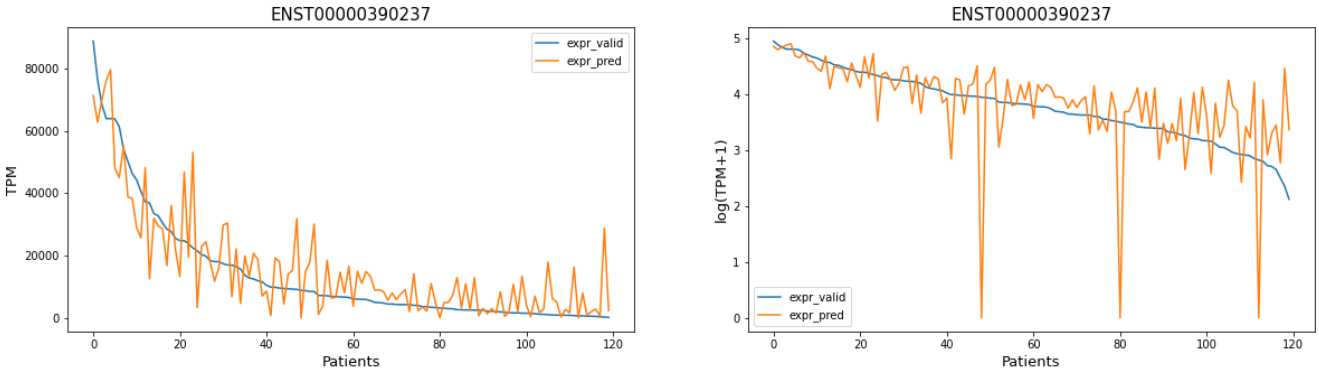


Figure 43. Theoretical and model-predicted expression values (colored in blue and orange, respectively) in tpm and log(tpm+1) for the isoform with the highest RMSE using the model with RMSE as loss function with 3 hidden layers and without label transformation.

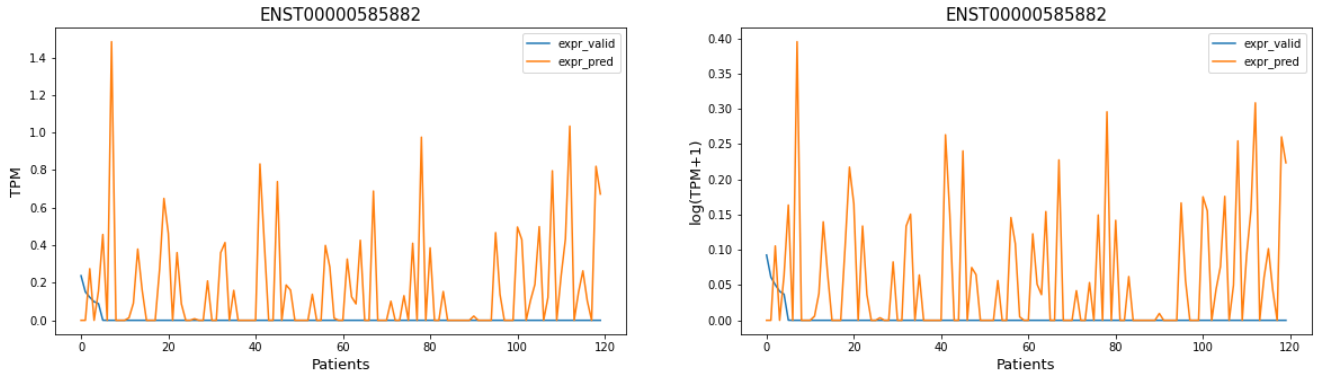


Figure 44. Theoretical and model-predicted expression values (colored in blue and orange, respectively) in tpm and log(tpm+1) for the isoform with the lowest RMSE using the model with RMSE as loss function with 3 hidden layers and without label transformation

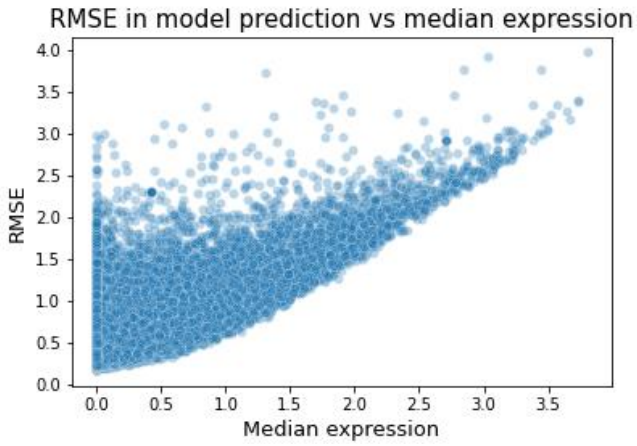


Figure 45. The RMSE in model prediction in log10(x+1) against the real median expression for the model with RMSE as loss function with 3 hidden layers and without label transformation.

7.1.2. Best optimizer selection with PyTorch

7.1.2.1. *Results using AdamW optimizer with DeepSF2hidden model*

The following table shows the transcripts that have obtained a higher correlation of the prediction with the real values:

Gene Name	*Transcript Name*	*Correlation Values*
SDHAF2	ENST00000536250	0.9999
MGA	ENST00000566718	0.9998
NUTM2A	ENST00000381707	0.9828
ADAM10	ENST00000396136	0.9733
GNA13	ENST00000439174	0.9727

Table 25. The five isoforms with the highest correlation value in the validation data using AdamW as optimization algorithm.

7.1.2.2. *Results using Adam optimizer using DeepSF2hidden model*

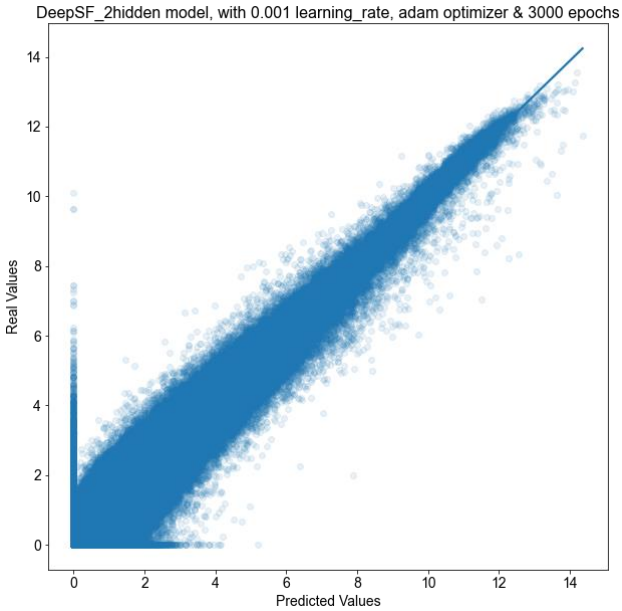


Figure 46. Predicted values against real values for the training data using the Adam optimization algorithm.

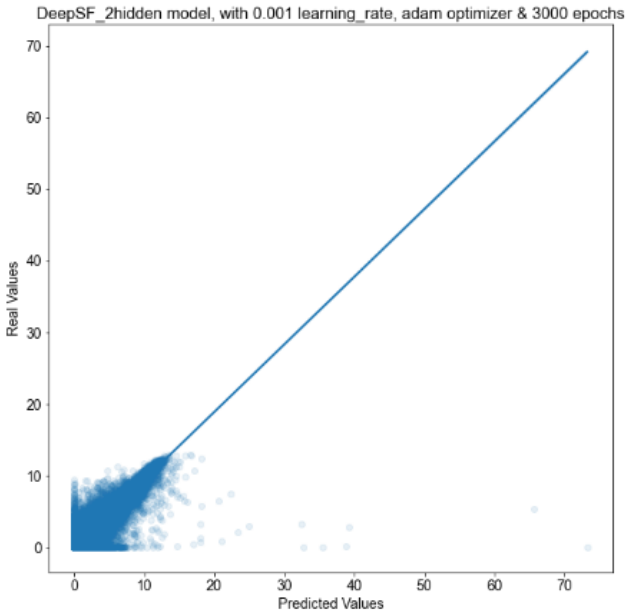


Figure 47. Predicted values against real values for the validation data using the Adam optimization algorithm.

Gene Name	*Transcript Name*	*Correlation Values*
MGA	ENST00000566718	0.9999
NUTM2A	ENST00000381707	0.9849
ABL2	ENST00000502732	0.9781
TRIM23	ENST00000231524	0.9779
AMER1	ENST00000330258	0.9776

Table 26. The five isoforms with the highest correlation values in the validation data using Adam as optimization algorithm

7.1.2.3. *Results with AdaGrad optimizer using DeepSF2hidden model*

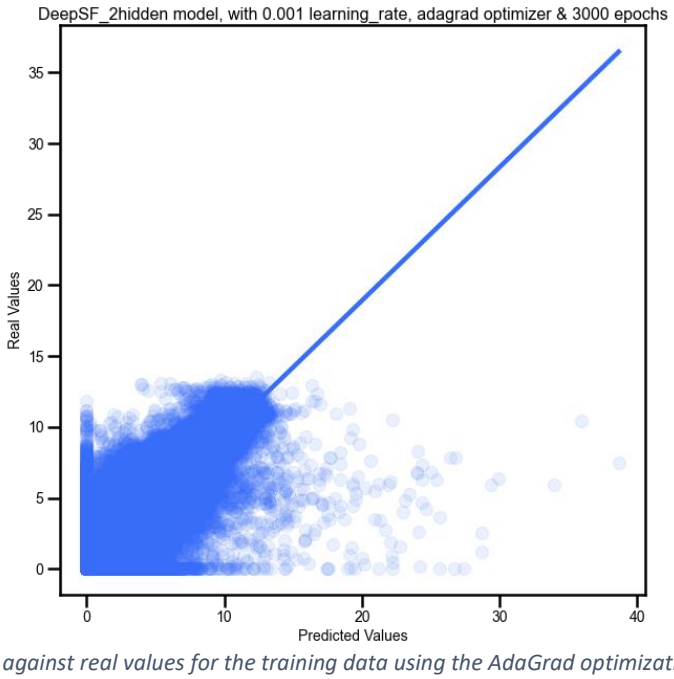


Figure 48. Predicted values against real values for the training data using the AdaGrad optimization algorithm

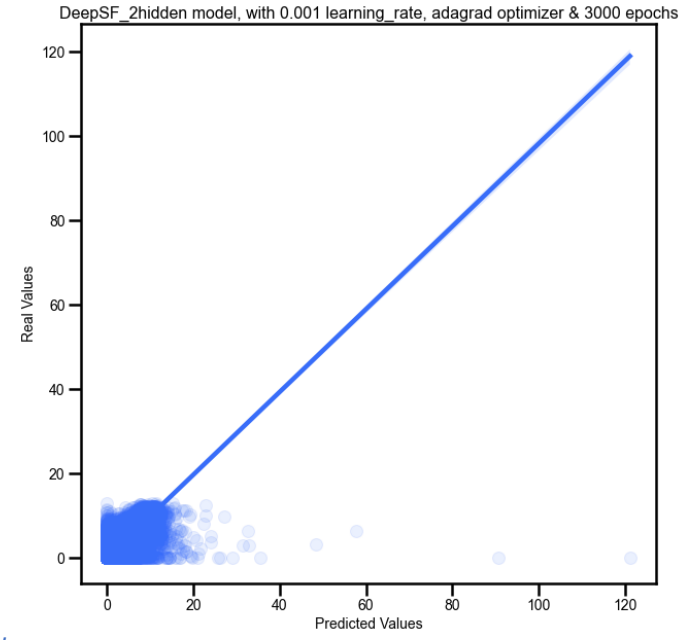


Figure 49. Predicted values against real values for the validation data using the AdaGrad optimization algorithm.

Gene Name	*Transcript Name*	*Correlation Values*
MDM2	ENST00000517852	1
NUTM2A	ENST00000381707	0.9616
BIRC5	ENST00000350051	0.9401
STAT3	ENST00000585517	0.9383
ARHGEF12	ENST00000356641	0.9359

Table 27. The five isoforms with the highest correlation value in the validation data using AdaGrad as optimization algorithm.

7.1.2.4. *Results with SGD90 optimizer using DeepSF2hidden model*

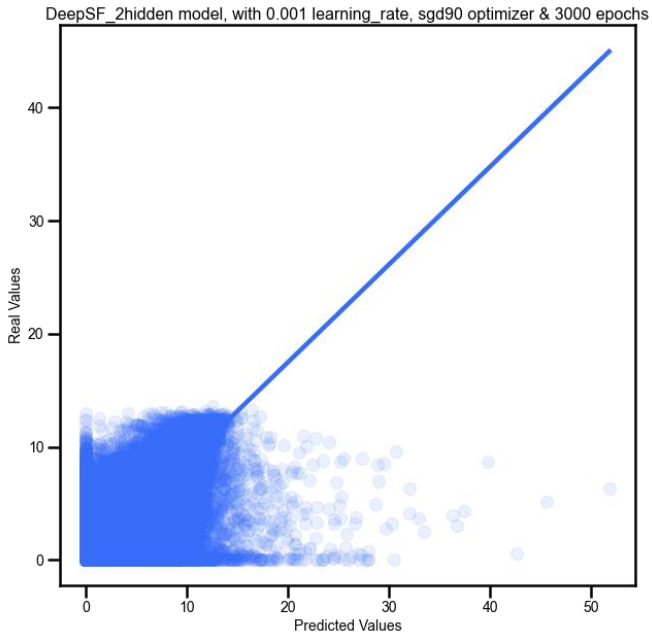


Figure 50. Predicted values against real values for the training data using the SGD90 optimization algorithm.

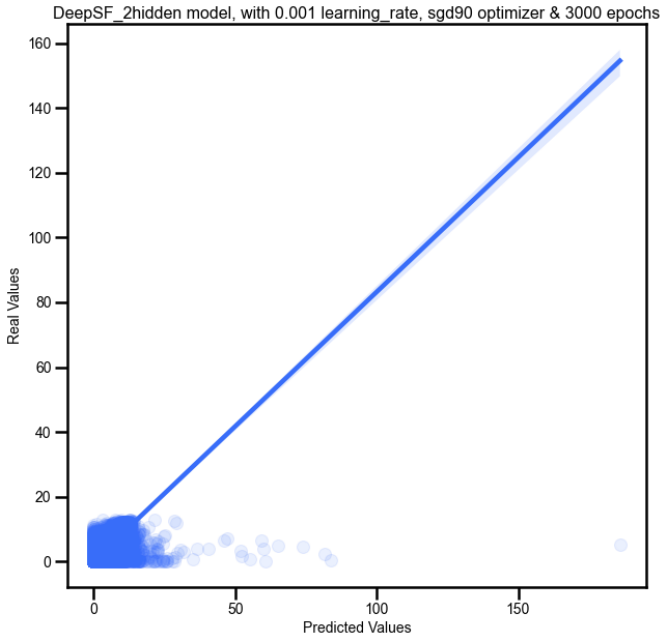


Figure 51. Predicted values against real values for the validation data using the SGD90 optimization algorithm.

Gene Name	*Transcript Name*	*Correlation Values*
BIRC5	ENST00000350051	0.9796
MAML2	ENST00000524717	0.9468
CBLC	ENST00000270279	0.9449
RIPK2	ENST00000522965	0.9449
EPHA2	ENST00000358432	0.9442

Table 28. The five isoforms with the highest correlation value in the validation data using SGD90 as optimization algorithm.

7.1.2.5. *Results with SGD70 optimizer using DeepSF2hidden model*

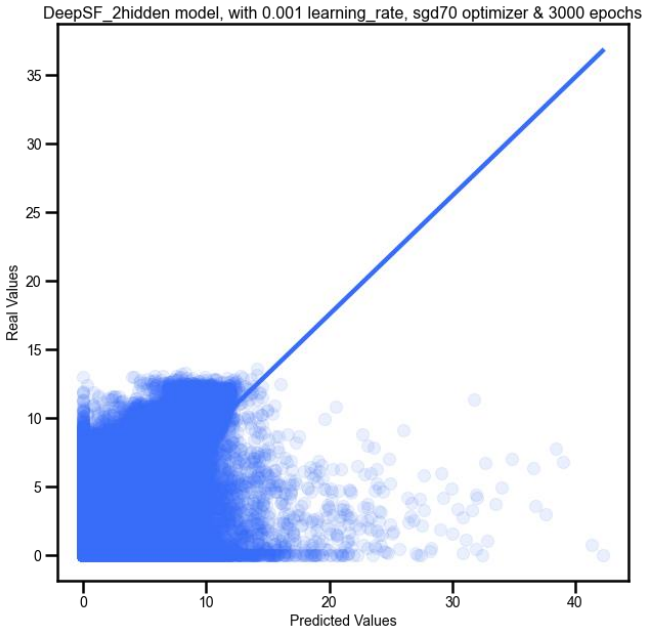


Figure 52. Predicted values against real values for the training data using the SGD70 optimization algorithm.

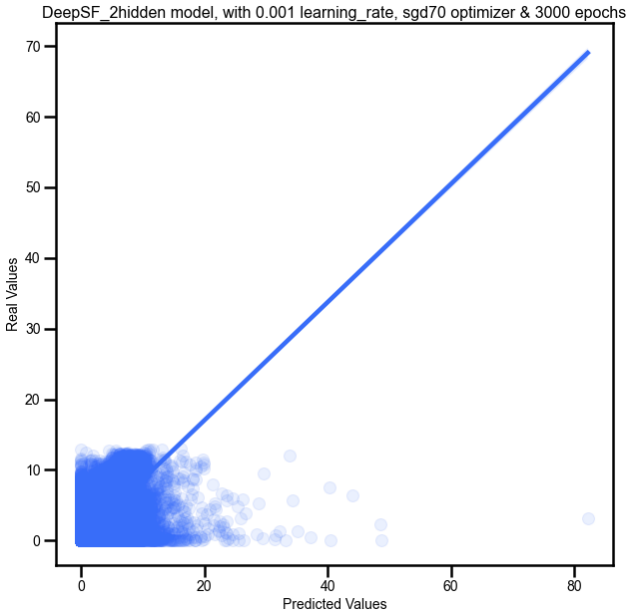


Figure 53. Predicted values against real values for the validation data using the SGD70 optimization algorithm.

Gene Name	*Transcript Name*	*Correlation Values*
STX2	ENST00000261653	0.9535
PDAP1	ENST00000426447	0.9355
HSPA8	ENST00000227378	0.9310
RANBP2	ENST00000283195	0.9173
IPO7	ENST00000379719	0.9139

Table 29. The five isoforms with the highest correlation value in the validation data using SGD70 as optimization algorithm

7.1.2.6. *Results with optimizer using DeepSF2hidden model*

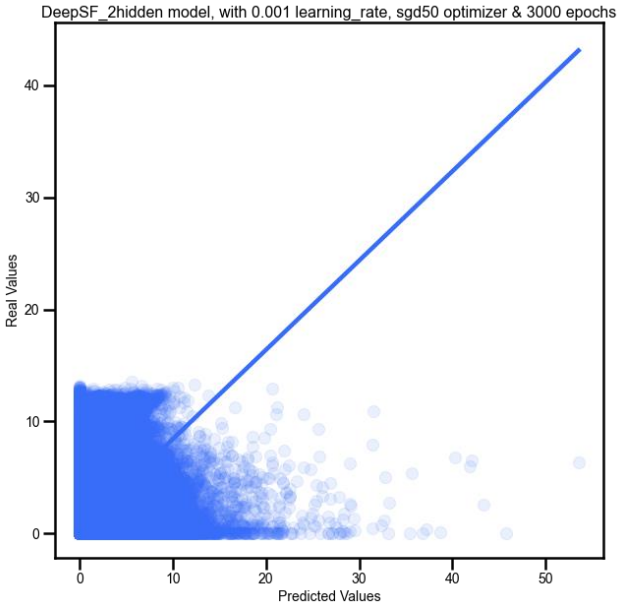


Figure 54. Predicted values against real values for the training data using the SGD50 optimization algorithm.

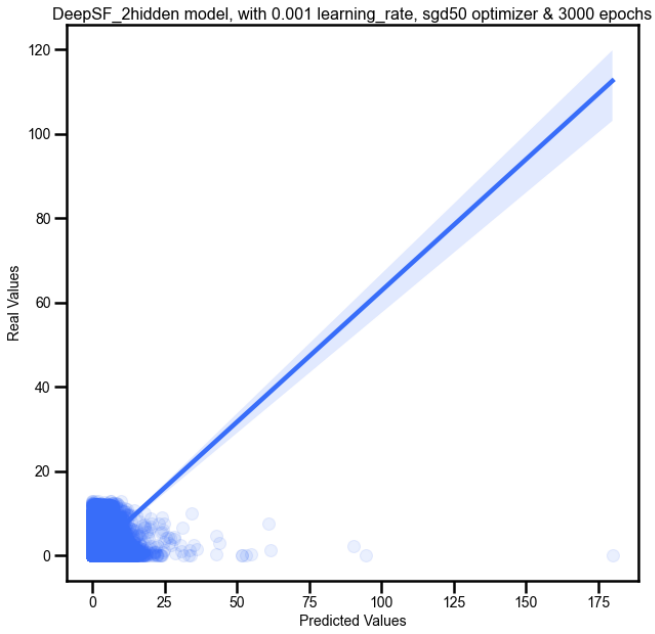


Figure 55. Predicted values against real values for the validation data using the SGD50 optimization algorithm.

Gene Name	*Transcript Name*	*Correlation Values*
LCP1	ENST00000	0.9455
ARID5B	ENST00000	0.9168
C2orf44	ENST00000	0.9140
MSN	ENST00000	0.9118
EXT1	ENST00000	0.9105

Table 30. The five isoforms with the highest correlation value in the validation data using SGD50 as optimization algorithm.

7.1.2.7. *Results with ASGD optimizer using DeepSF2hidden model*

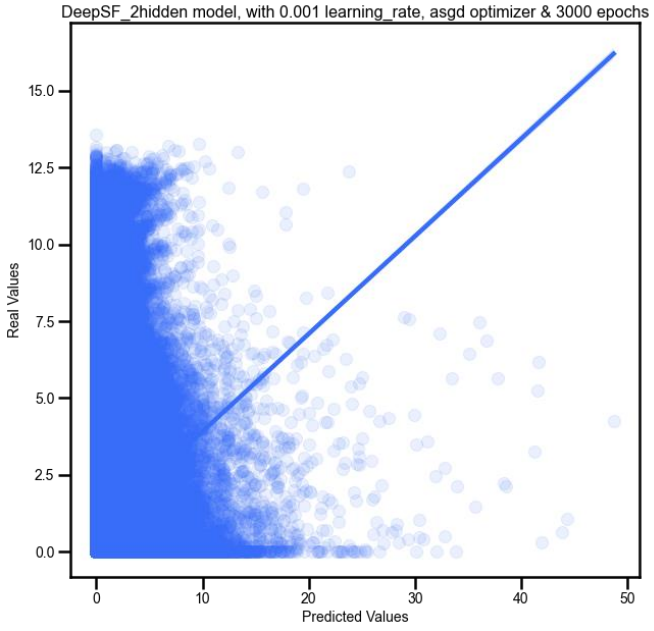


Figure 56. Predicted values against real values for the training data using the ASGD optimization algorithm.

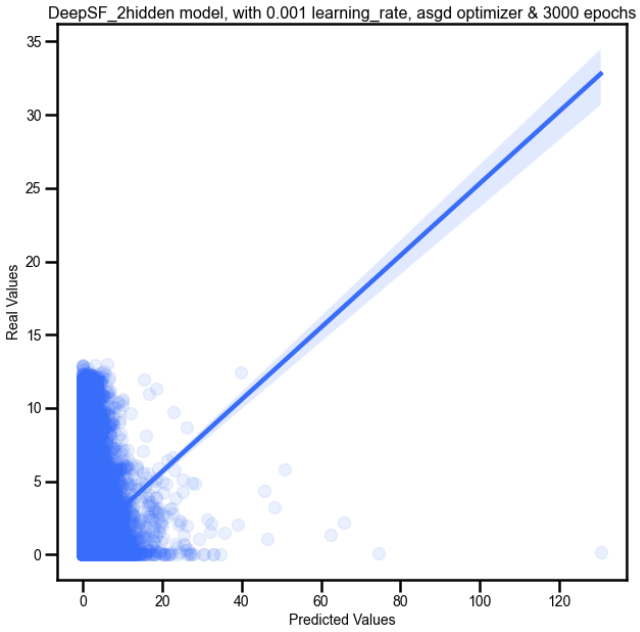


Figure 57. Predicted values against real values for the validation data using the ASGD optimization algorithm.

Gene Name	*Transcript Name*	*Correlation Values*
MYH9	ENST00000216181	0.9581
HSP90AA1	ENST00000216281	0.9361
GNB1	ENST00000378609	0.9338
CD74	ENST00000353334	0.9096
BCL10	ENST00000370580	0.9029

Table 31. The five isoforms with the highest correlation value in the validation data using ASGD as optimization algorithm.

7.1.2.8. *Results with Adadelta optimizer using DeepSF2hidden model*

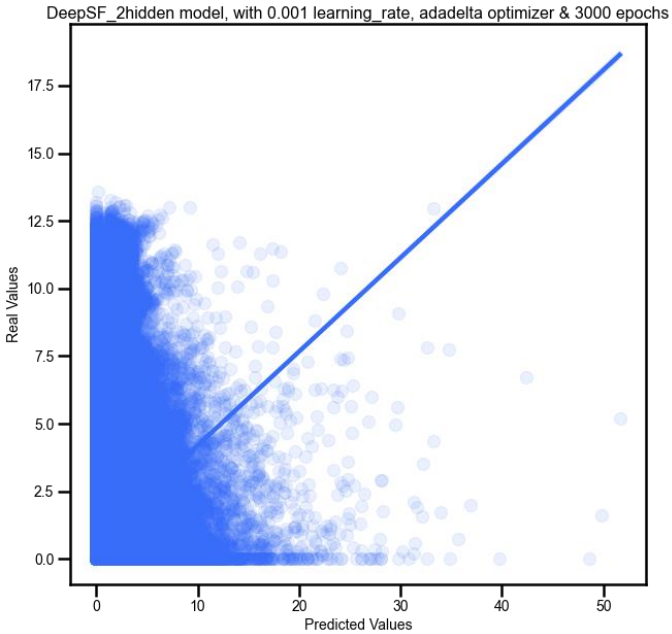


Figure 58. Predicted values against real values for the training data using the AdaDelta optimization algorithm.

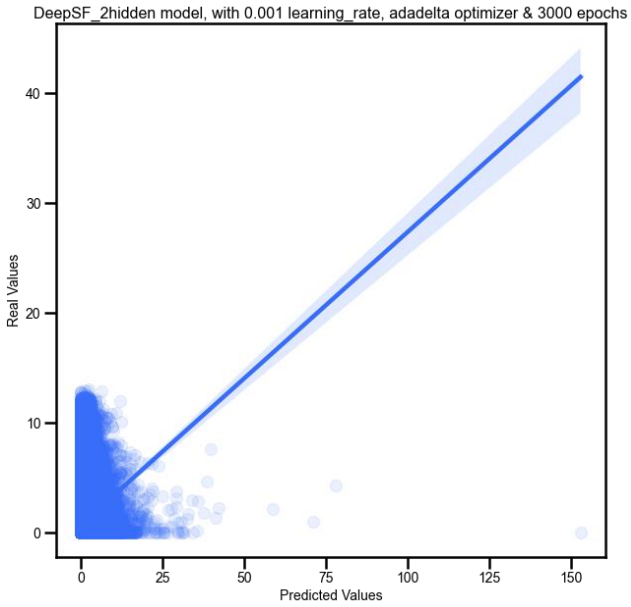


Figure 59. Predicted values against real values for the validation data using the AdaDelta optimization algorithm.

Gene Name	*Transcript Name*	*Correlation Values*
GNA13	ENST00000439174	0.9258
HMGA1	ENST00000311487	0.9212
IPO7	ENST00000379719	0.9148
SLC34A2	ENST00000382051	0.9130
YES1	ENST00000314574	0.9062

Figure 60. The five isoforms with the highest correlation value in the validation data using AdaDelta as optimization algorithm.

7.1.3. Results in the training of the DeepAE model

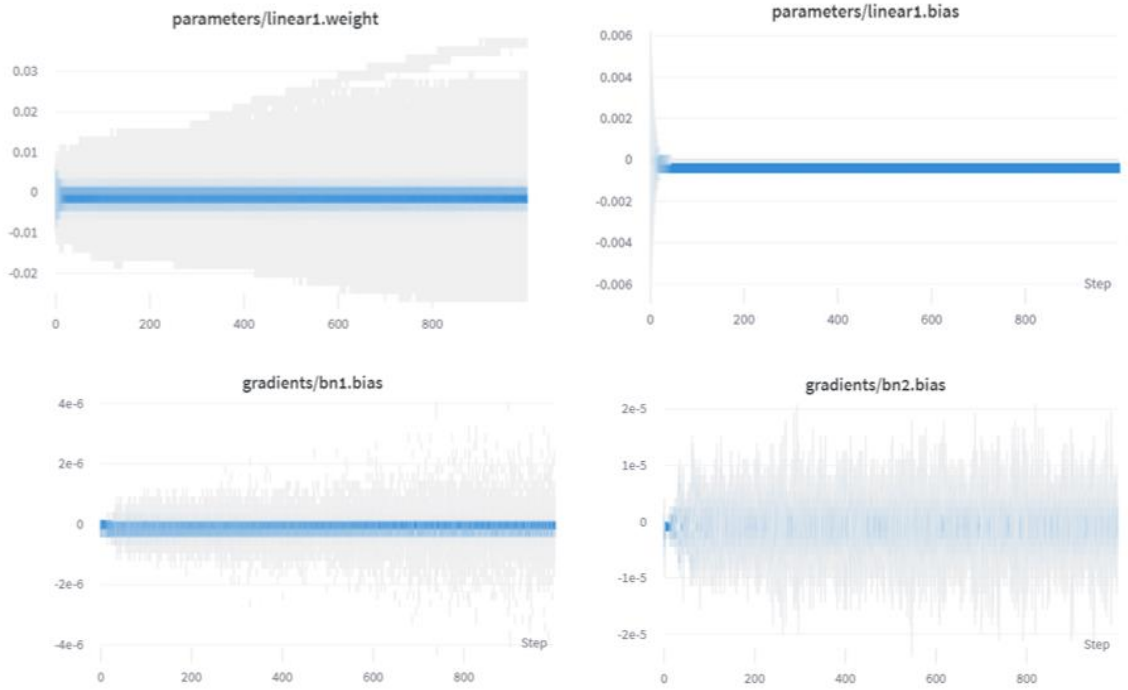


Figure 61. Evolution of gradient values of the bias parameters in the linear layers during training of DeepAE.

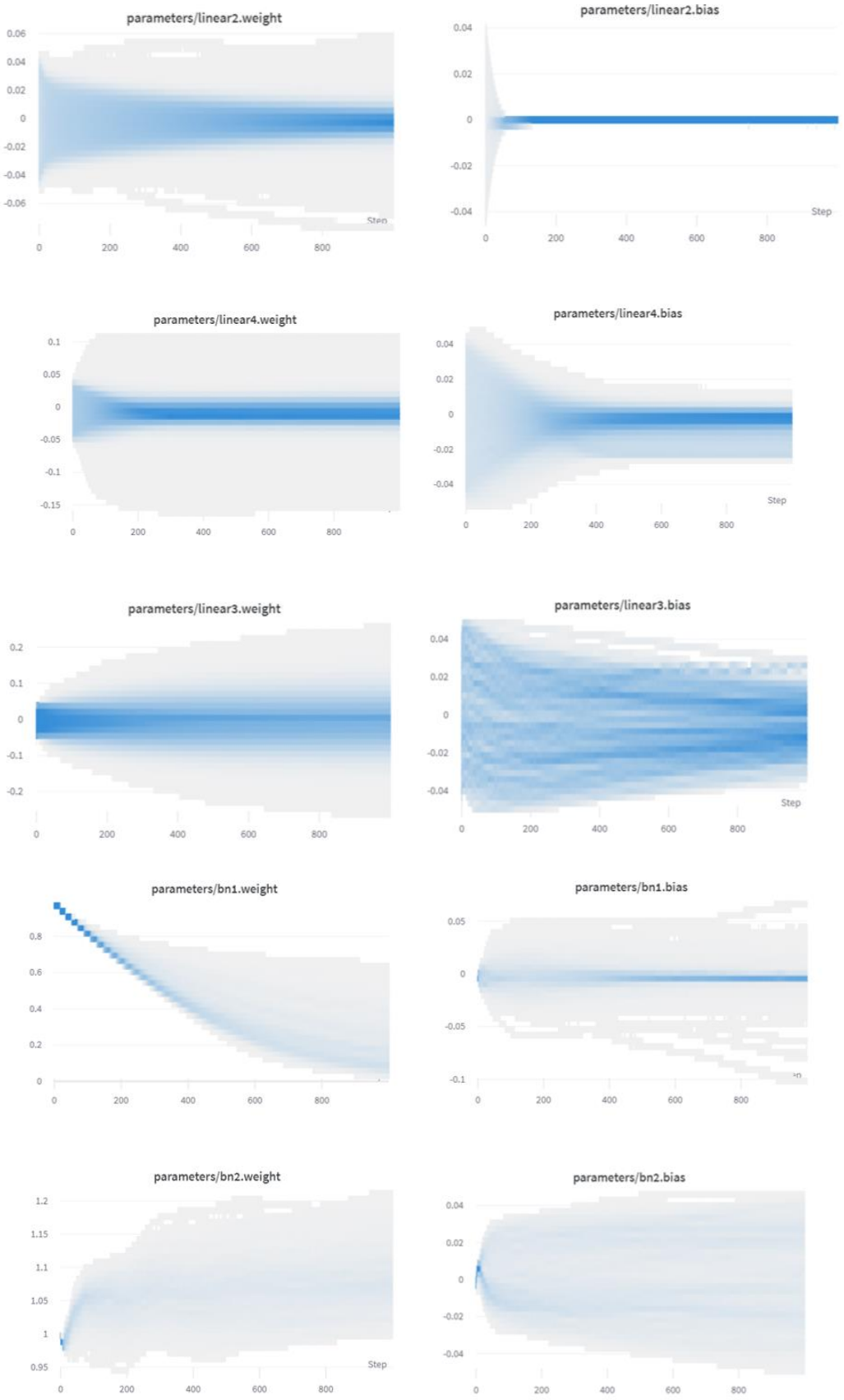


Figure 62. Evolution of parameter values during training in DeepAE model.

7.1.4. Results in the training of the DeepSF&AEensemble model

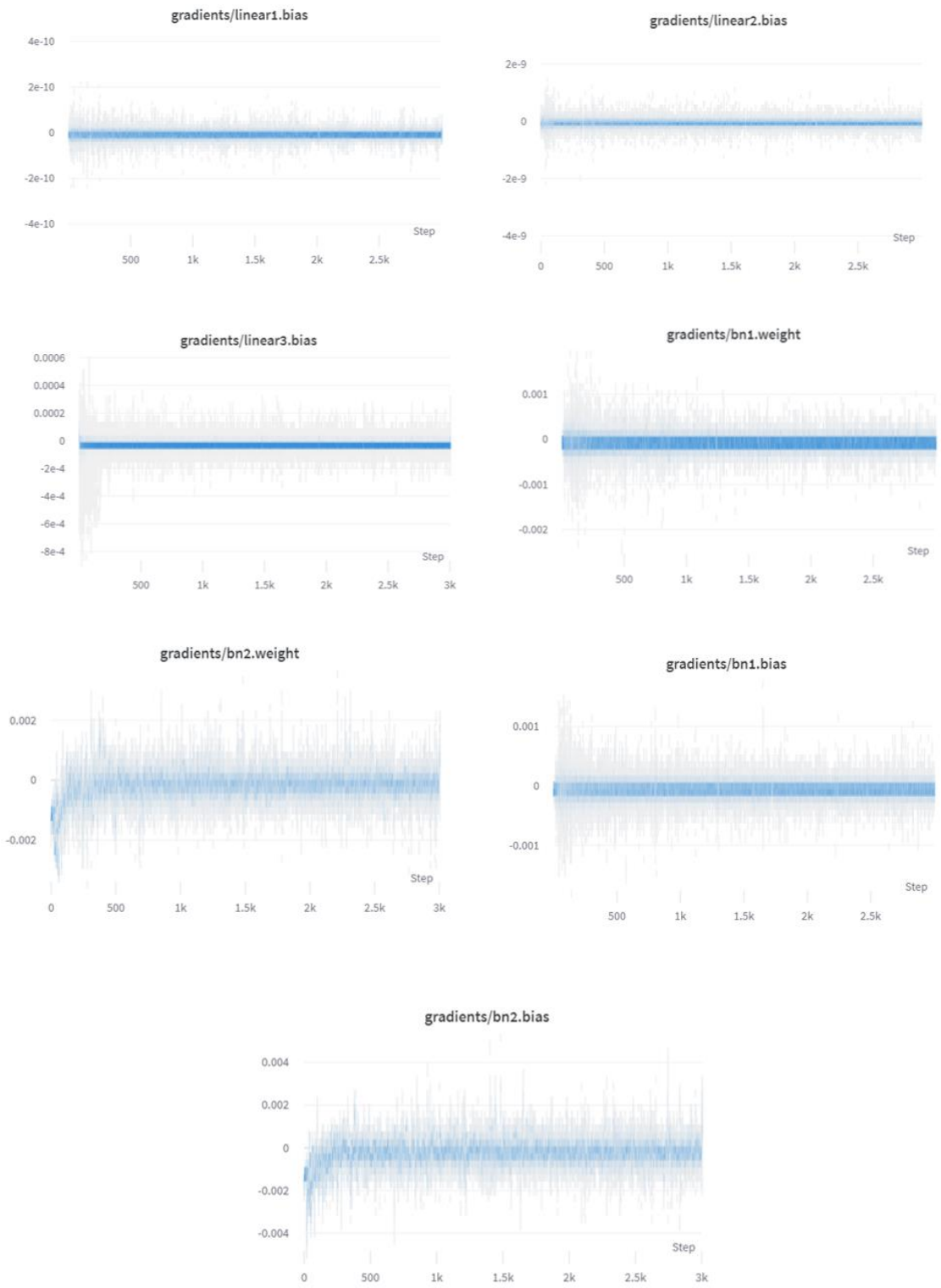
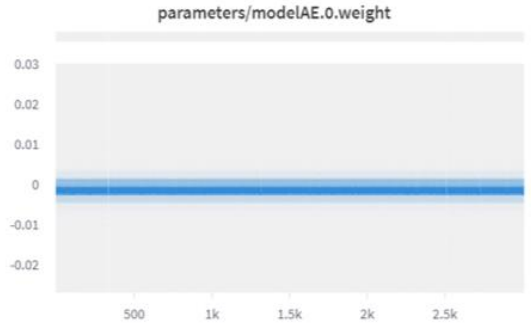
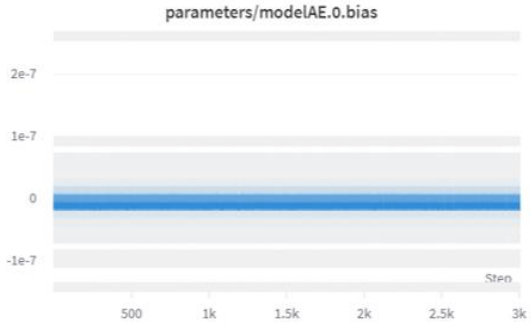
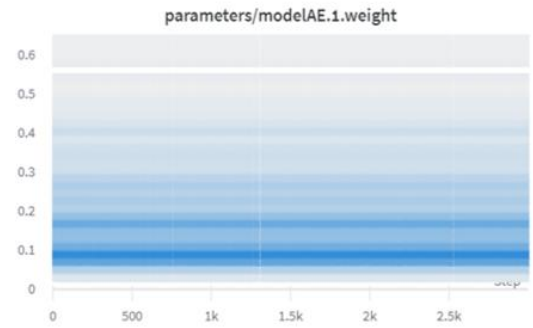
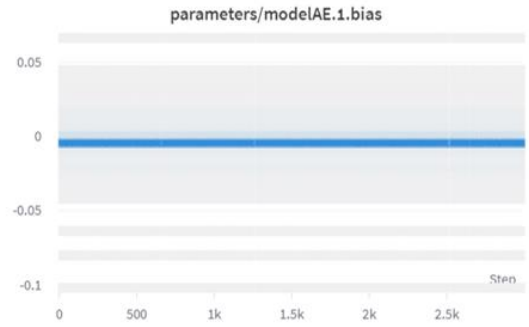


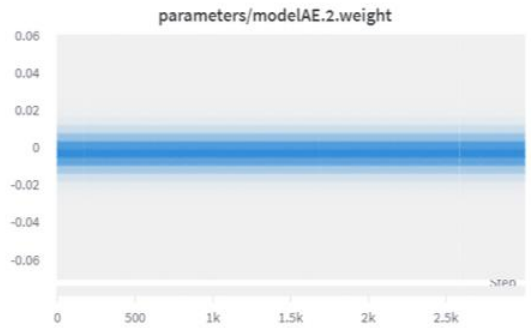
Figure 63. Evolution of the values of the gradients of the parameters in DeepSF&AEensemble model.

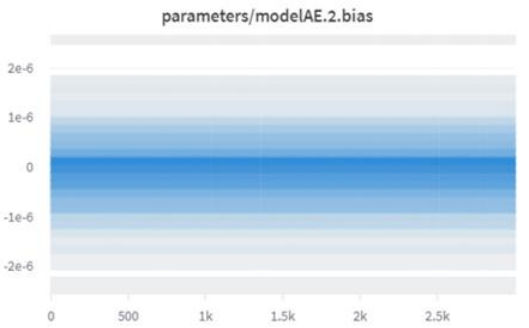

parameters/modelAE.0.weight
0.03
0.02
0.01
0
-0.01
-0.02
500
1k
1.5k
2k
2.5k

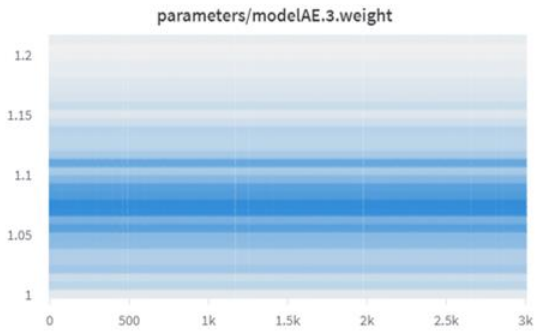

parameters/modelAE.0.bias
2e-7
1e-7
0
-1e-7
Step
500
1k
1.5k
2k
2.5k
3k

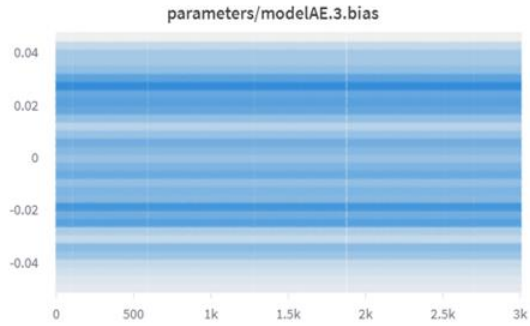

parameters/modelAE.1.weight
0.6
0.5
0.4
0.3
0.2
0.1
0
0
500
1k
1.5k
2k
2.5k

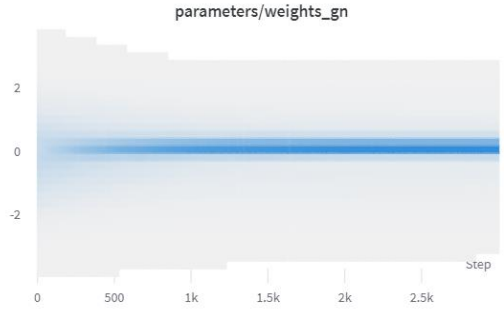

parameters/modelAE.1.bias
0.05
0
-0.05
-0.1
Step
0
500
1k
1.5k
2k
2.5k

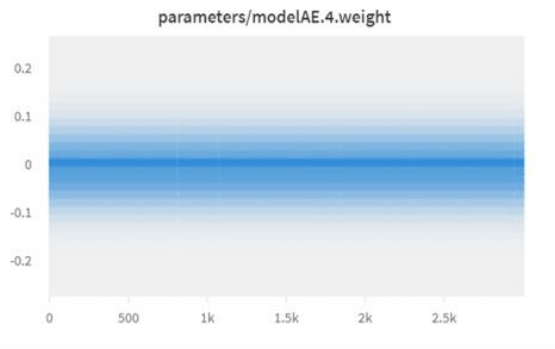

parameters/modelAE.2.weight
0.06
0.04
0.02
0
-0.02
-0.04
-0.06
Step
0
500
1k
1.5k
2k
2.5k

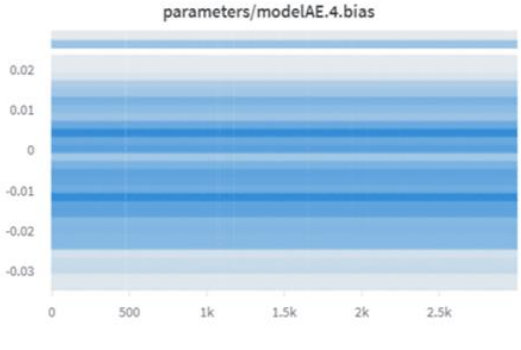

parameters/modelAE.2.bias
2e-6
1e-6
0
-1e-6
-2e-6
0
500
1k
1.5k
2k
2.5k

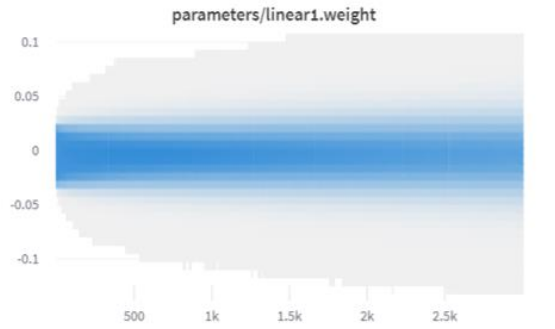

parameters/modelAE.3.weight
1.2
1.15
1.1
1.05
1
0
500
1k
1.5k
2k
2.5k
3k

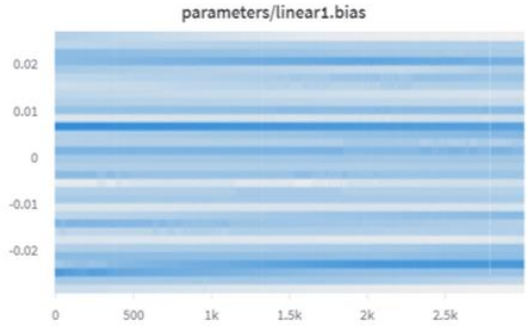

parameters/modelAE.3.bias
0.04
0.02
0
-0.02
-0.04
0
500
1k
1.5k
2k
2.5k
3k

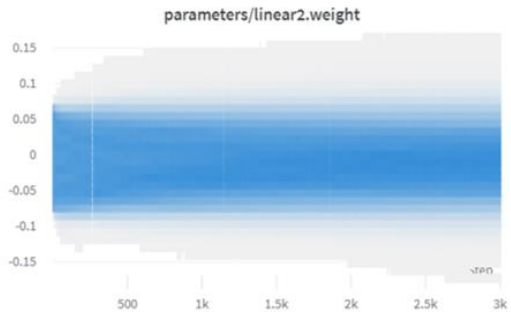

parameters/weights_gn
2
0
-2
Step
0
500
1k
1.5k
2k
2.5k

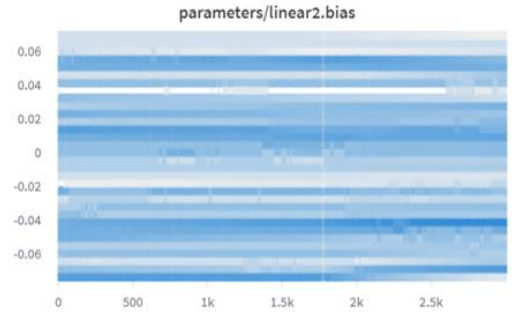

parameters/modelAE.4.weight
0.2
0.1
0
-0.1
-0.2
0
500
1k
1.5k
2k
2.5k

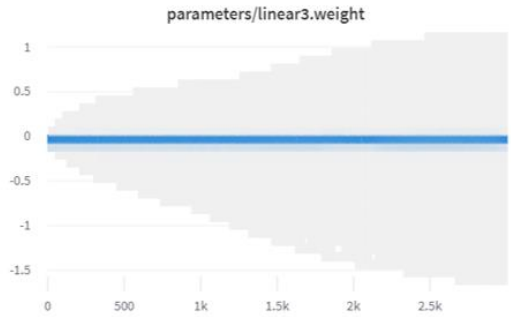

parameters/modelAE.4.bias
0.02
0.01
0
-0.01
-0.02
-0.03
0
500
1k
1.5k
2k
2.5k

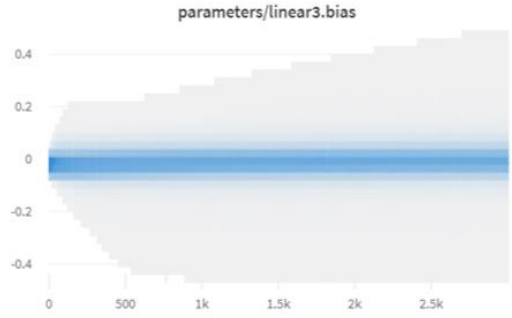

parameters/linear1.weight
0.1
0.05
0
-0.05
-0.1
500
1k
1.5k
2k
2.5k

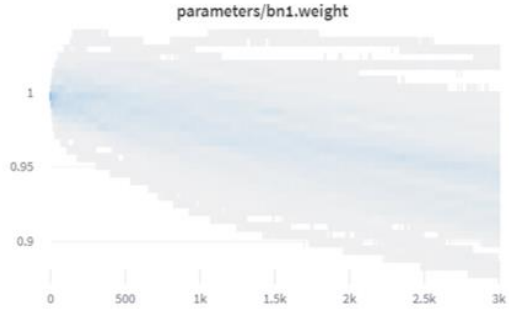

parameters/linear1.bias
0.02
0.01
0
-0.01
-0.02
0
500
1k
1.5k
2k
2.5k

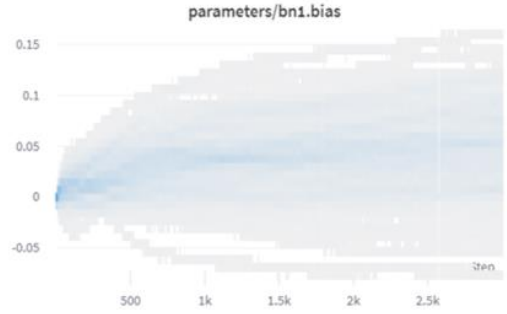

parameters/linear2.weight
0.15
0.1
0.05
0
-0.05
-0.1
-0.15
500
1k
1.5k
2k
2.5k
3k


parameters/linear2.bias
0.06
0.04
0.02
0
-0.02
-0.04
-0.06
0
500
1k
1.5k
2k
2.5k


parameters/linear3.weight
1
0.5
0
-0.5
-1
-1.5
0
500
1k
1.5k
2k
2.5k


parameters/linear3.bias
0.4
0.2
0
-0.2
-0.4
0
500
1k
1.5k
2k
2.5k


parameters/bn1.weight
1
0.95
0.9
0
500
1k
1.5k
2k
2.5k
3k


parameters/bn1.bias
0.15
0.1
0.05
0
-0.05
500
1k
1.5k
2k
2.5k

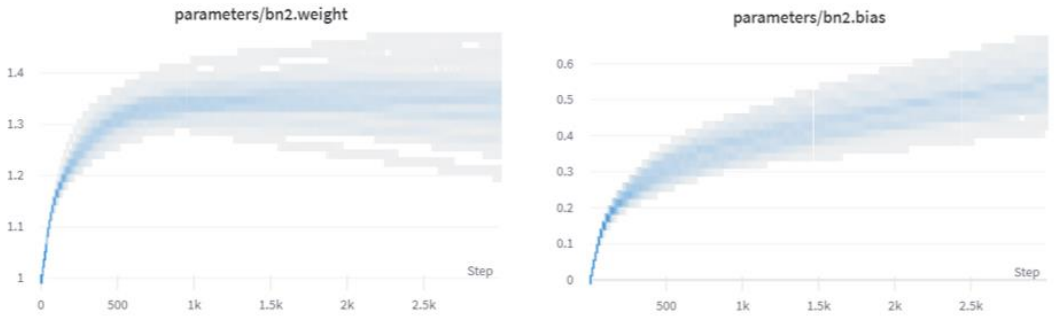
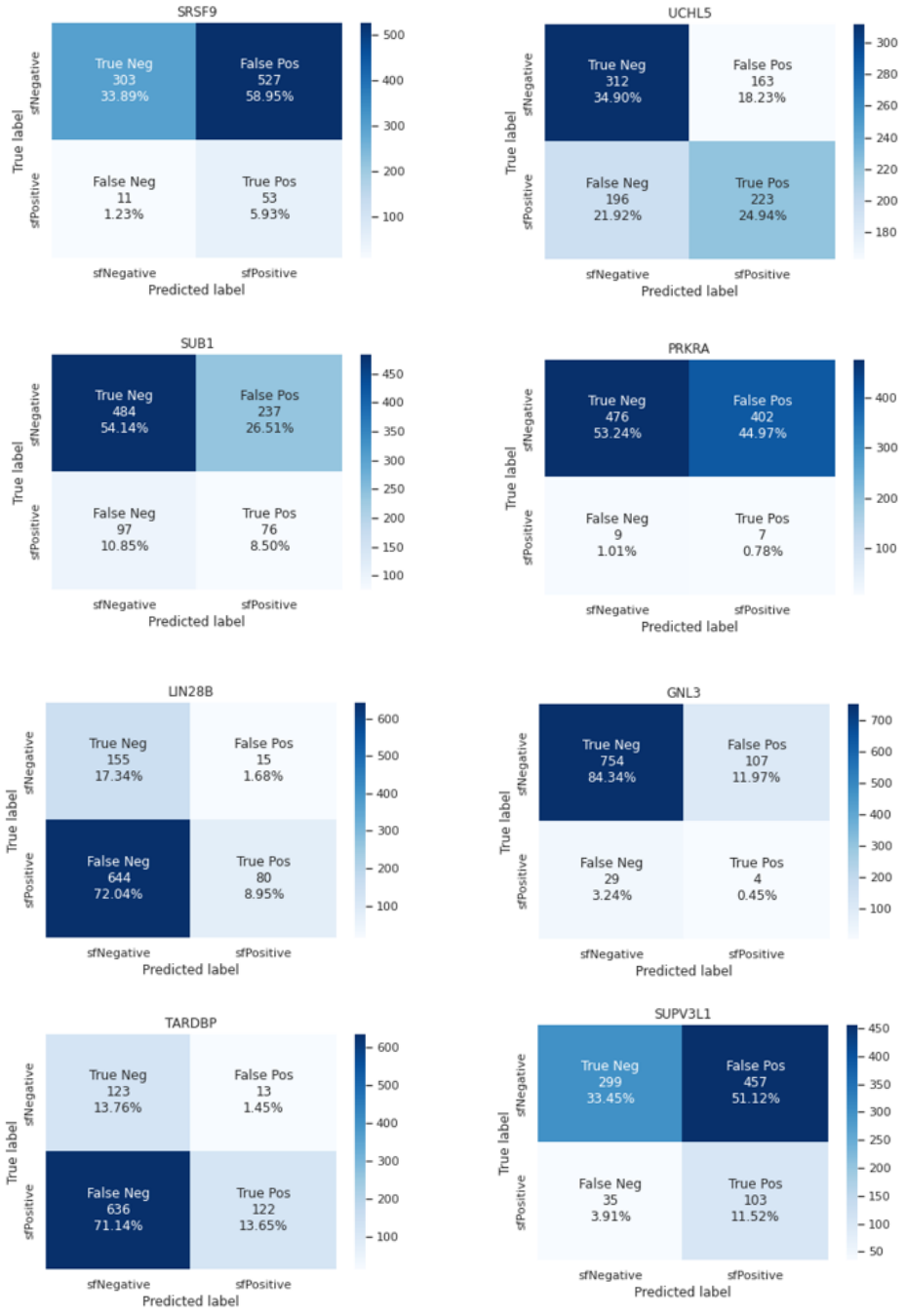
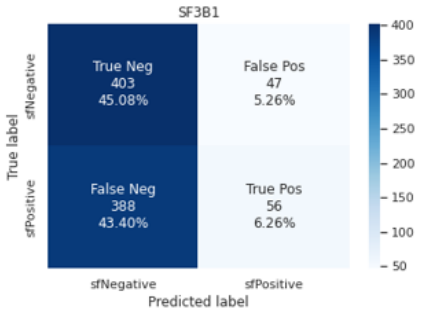
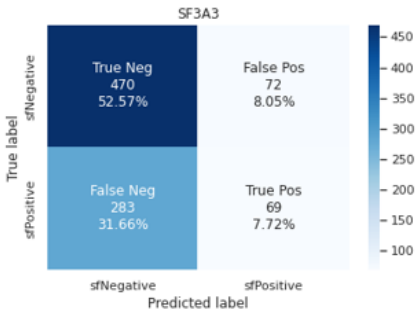


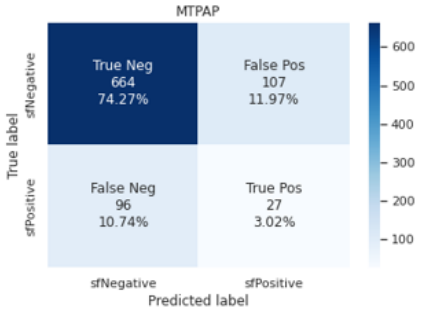
Figure 64. Evolution of the values of the parameters during the training of DeepSF&AEensemble model.

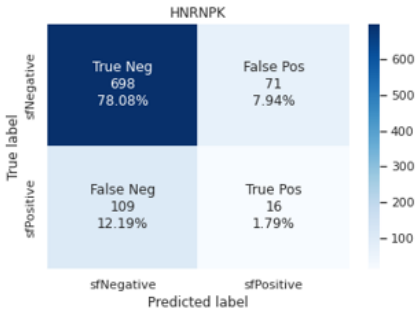
7.1.5. Interpretation of the model parameters

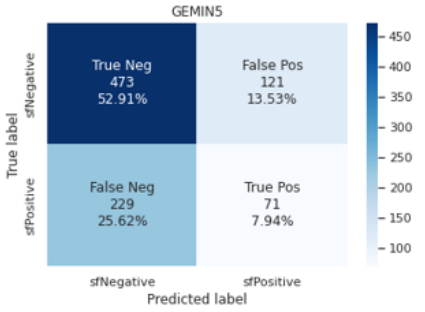


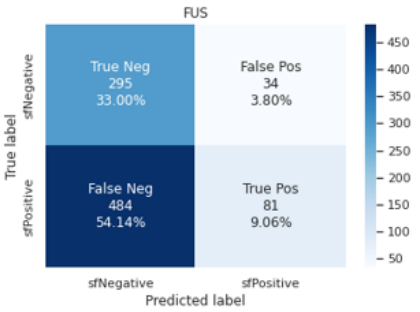

SF3B1
True Neg
403
45.08%
False Pos
47
5.26%
False Neg
388
43.40%
True Pos
56
6.26%
sfNegative
sfPositive
True label
Predicted label
400
350
300
250
200
150
100
50

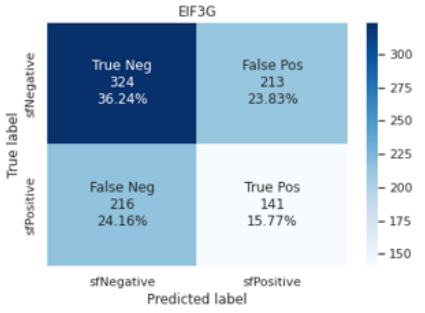

SF3A3
True Neg
470
52.57%
False Pos
72
8.05%
False Neg
283
31.66%
True Pos
69
7.72%
sfNegative
sfPositive
True label
Predicted label
450
400
350
300
250
200
150
100

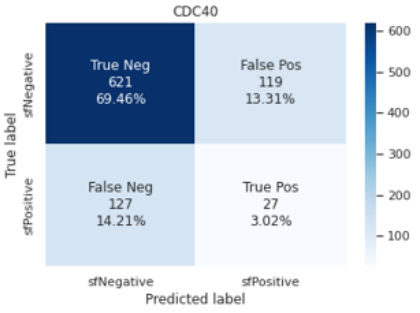

MTPAP
True Neg
664
74.27%
False Pos
107
11.97%
False Neg
96
10.74%
True Pos
27
3.02%
sfNegative
sfPositive
True label
Predicted label
600
500
400
300
200
100

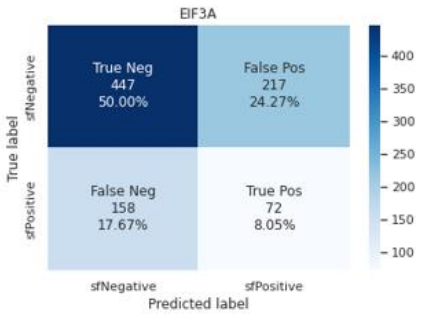

HNRNPK
True Neg
698
78.08%
False Pos
71
7.94%
False Neg
109
12.19%
True Pos
16
1.79%
sfNegative
sfPositive
True label
Predicted label
600
500
400
300
200
100

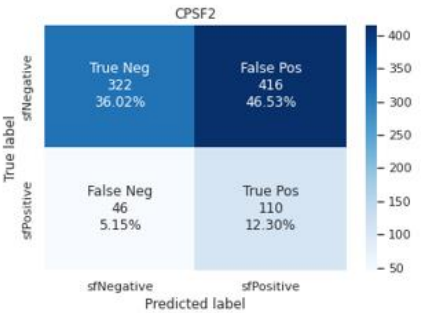

GEMIN5
True Neg
473
52.91%
False Pos
121
13.53%
False Neg
229
25.62%
True Pos
71
7.94%
sfNegative
sfPositive
True label
Predicted label
450
400
350
300
250
200
150
100

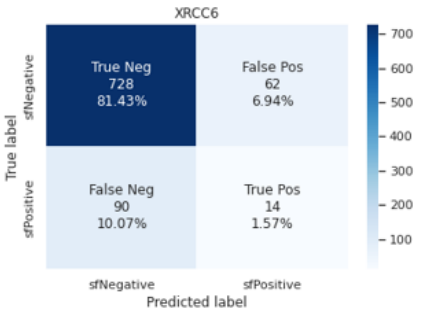

FUS
True Neg
295
33.00%
False Pos
34
3.80%
False Neg
484
54.14%
True Pos
81
9.06%
sfNegative
sfPositive
True label
Predicted label
450
400
350
300
250
200
150
100
50

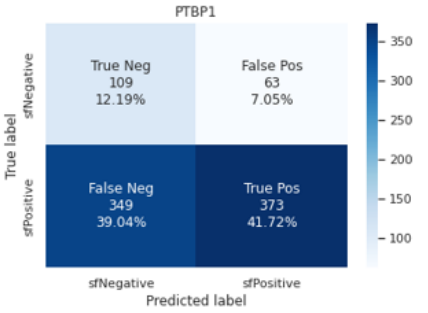

EIF3G
True Neg
324
36.24%
False Pos
213
23.83%
False Neg
216
24.16%
True Pos
141
15.77%
sfNegative
sfPositive
True label
Predicted label
300
275
250
225
200
175
150

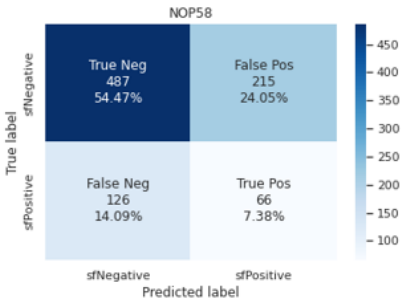

CDC40
True Neg
621
69.46%
False Pos
119
13.31%
False Neg
127
14.21%
True Pos
27
3.02%
sfNegative
sfPositive
True label
Predicted label
600
500
400
300
200
100

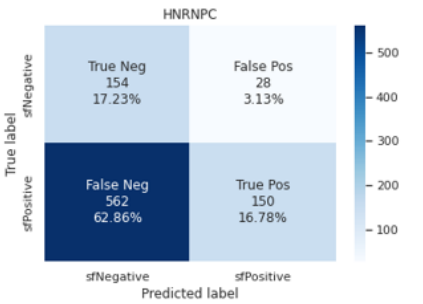

EIF3A
True Neg
447
50.00%
False Pos
217
24.27%
False Neg
158
17.67%
True Pos
72
8.05%
sfNegative
sfPositive
True label
Predicted label
400
350
300
250
200
150
100

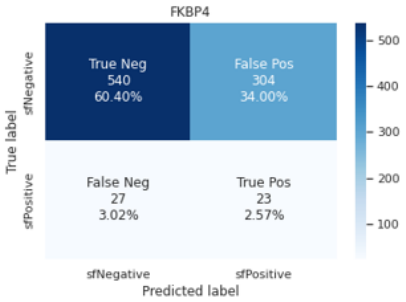

CPSF2
True Neg
322
36.02%
False Pos
416
46.53%
False Neg
46
5.15%
True Pos
110
12.30%
sfNegative
sfPositive
True label
Predicted label
400
350
300
250
200
150
100
50

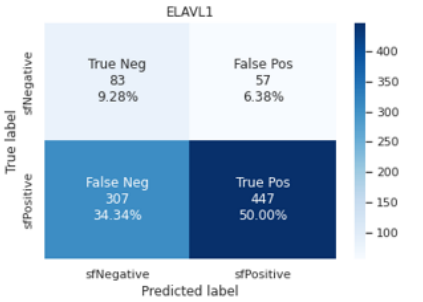

XRCC6
True Neg
728
81.43%
False Pos
62
6.94%
False Neg
90
10.07%
True Pos
14
1.57%
sfNegative
sfPositive
True label
Predicted label
700
600
500
400
300
200
100

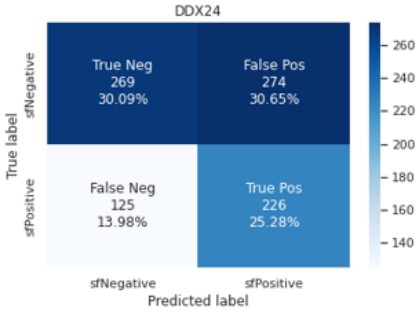

PTBP1
True Neg
109
12.19%
False Pos
63
7.05%
False Neg
349
39.04%
True Pos
373
41.72%
sfNegative
sfPositive
True label
Predicted label
350
300
250
200
150
100

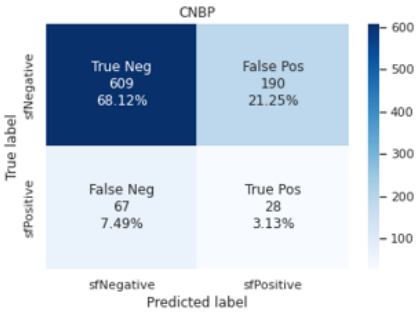

NOP58
True Neg
487
54.47%
False Pos
215
24.05%
False Neg
126
14.09%
True Pos
66
7.38%
sfNegative
sfPositive
True label
Predicted label
450
400
350
300
250
200
150
100

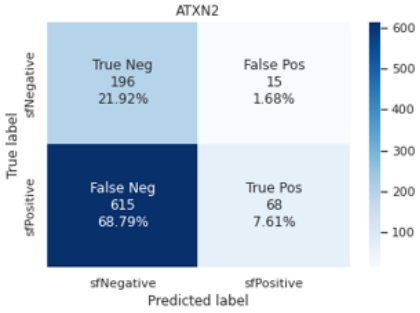

HNRNPC
True Neg
154
17.23%
False Pos
28
3.13%
False Neg
562
62.86%
True Pos
150
16.78%
sfNegative
sfPositive
True label
Predicted label
500
400
300
200
100

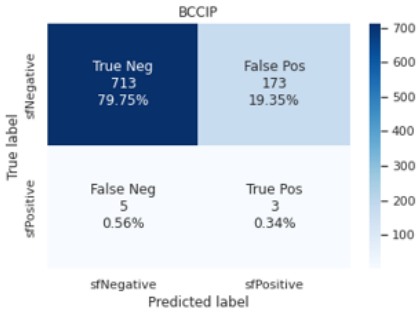

FKBP4
True Neg
540
60.40%
False Pos
304
34.00%
False Neg
27
3.02%
True Pos
23
2.57%
sfNegative
sfPositive
True label
Predicted label
500
400
300
200
100

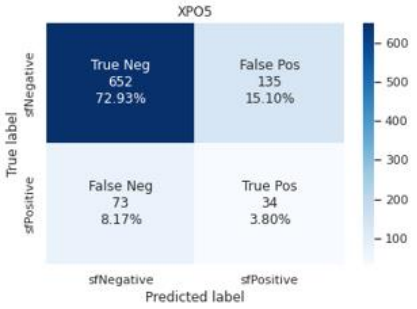

ELAVL1
True Neg
83
9.28%
False Pos
57
6.38%
False Neg
307
34.34%
True Pos
447
50.00%
sfNegative
sfPositive
True label
Predicted label
400
350
300
250
200
150
100

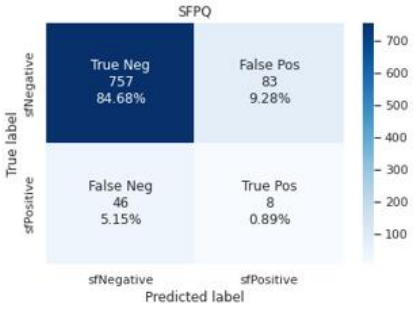

DDX24
True Neg
269
30.09%
False Pos
274
30.65%
False Neg
125
13.98%
True Pos
226
25.28%
sfNegative
sfPositive
True label
Predicted label
260
240
220
200
180
160
140

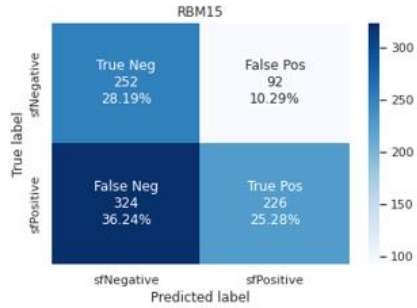

CNBP
True Neg
609
68.12%
False Pos
190
21.25%
False Neg
67
7.49%
True Pos
28
3.13%
sfNegative
sfPositive
True label
Predicted label
600
500
400
300
200
100

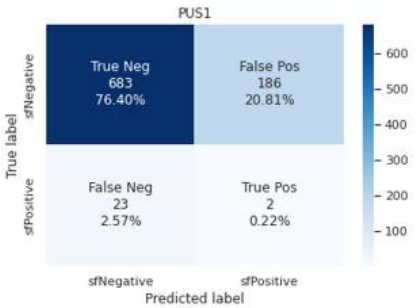

ATXN2
True Neg
196
21.92%
False Pos
15
1.68%
False Neg
615
68.79%
True Pos
68
7.61%
sfNegative
sfPositive
True label
Predicted label
600
500
400
300
200
100

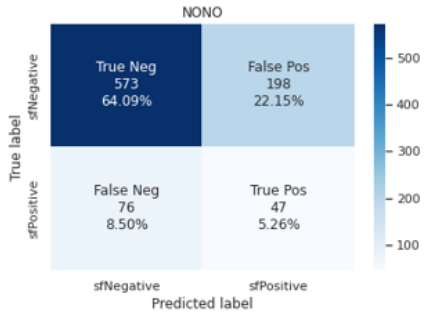

BCCIP
True Neg
713
79.75%
False Pos
173
19.35%
False Neg
5
0.56%
True Pos
3
0.34%
sfNegative
sfPositive
True label
Predicted label
700
600
500
400
300
200
100

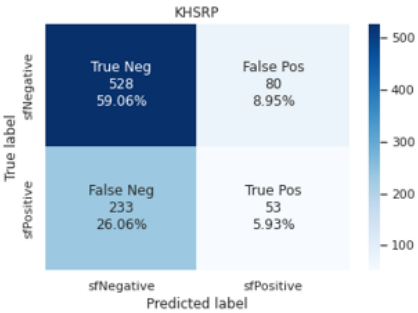

XPO5
True Neg
652
72.93%
False Pos
135
15.10%
False Neg
73
8.17%
True Pos
34
3.80%
sfNegative
sfPositive
True label
Predicted label
600
500
400
300
200
100

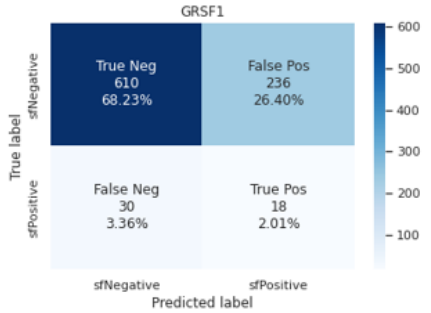

SFPQ
True Neg
757
84.68%
False Pos
83
9.28%
False Neg
46
5.15%
True Pos
8
0.89%
sfNegative
sfPositive
True label
Predicted label
700
600
500
400
300
200
100

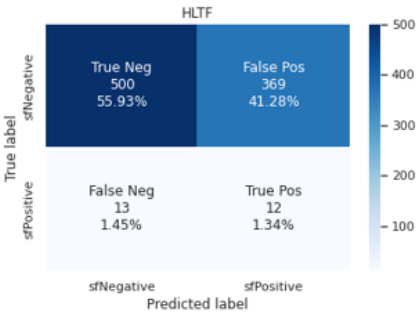

RBM15
True Neg
252
28.19%
False Pos
92
10.29%
False Neg
324
36.24%
True Pos
226
25.28%
sfNegative
sfPositive
True label
Predicted label
300
250
200
150
100

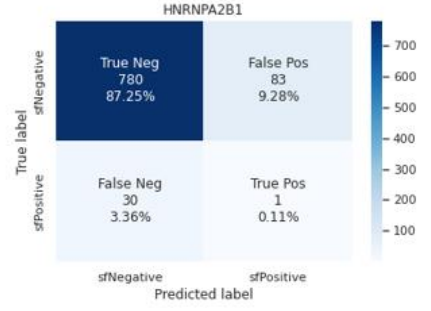

PUS1
True Neg
683
76.40%
False Pos
186
20.81%
False Neg
23
2.57%
True Pos
2
0.22%
sfNegative
sfPositive
True label
Predicted label
600
500
400
300
200
100

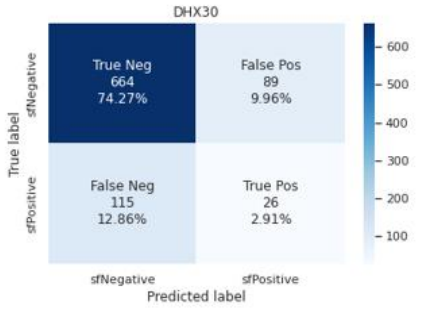

NONO
True Neg
573
64.09%
False Pos
198
22.15%
False Neg
76
8.50%
True Pos
47
5.26%
sfNegative
sfPositive
True label
Predicted label
500
400
300
200
100

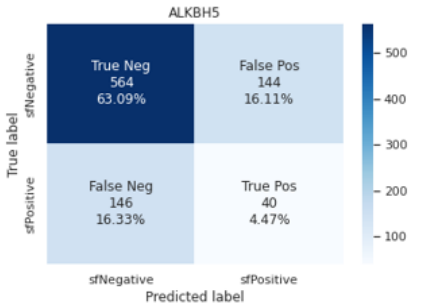

KHSRP
True Neg
528
59.06%
False Pos
80
8.95%
False Neg
233
26.06%
True Pos
53
5.93%
sfNegative
sfPositive
True label
Predicted label
500
400
300
200
100

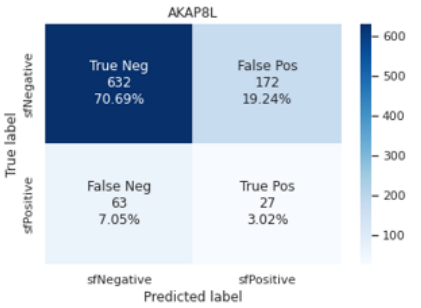

GRSF1
True Neg
610
68.23%
False Pos
236
26.40%
False Neg
30
3.36%
True Pos
18
2.01%
sfNegative
sfPositive
True label
Predicted label
600
500
400
300
200
100

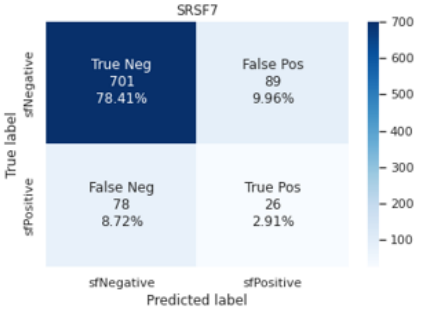

HLTF
True Neg
500
55.93%
False Pos
369
41.28%
False Neg
13
1.45%
True Pos
12
1.34%
sfNegative
sfPositive
True label
Predicted label
500
400
300
200
100

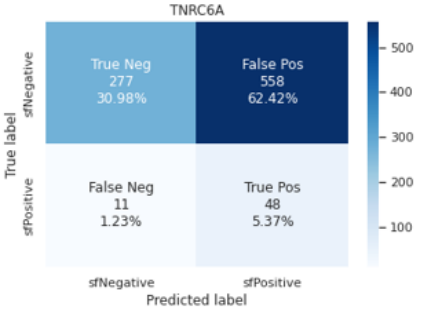

HNRNPA2B1
True Neg
780
87.25%
False Pos
83
9.28%
False Neg
30
3.36%
True Pos
1
0.11%
sfNegative
sfPositive
True label
Predicted label
700
600
500
400
300
200
100

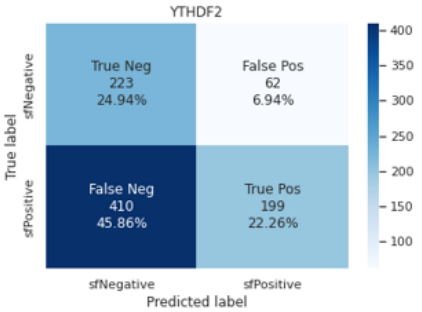

DHX30
True Neg
664
74.27%
False Pos
89
9.96%
False Neg
115
12.86%
True Pos
26
2.91%
sfNegative
sfPositive
True label
Predicted label
600
500
400
300
200
100

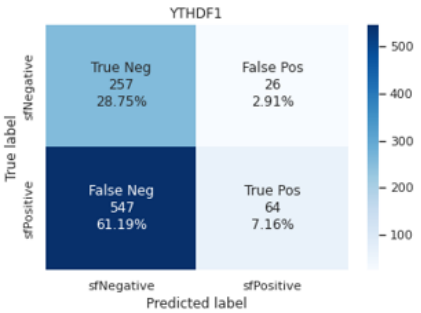

ALKBH5
True Neg
564
63.09%
False Pos
144
16.11%
False Neg
146
16.33%
True Pos
40
4.47%
sfNegative
sfPositive
True label
Predicted label
500
400
300
200
100

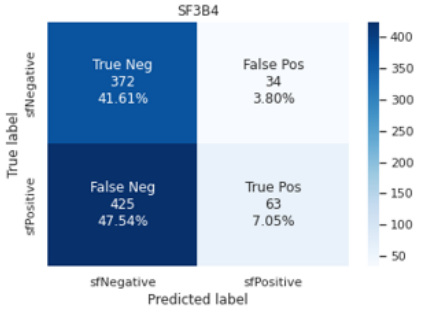

AKAP8L
True Neg
632
70.69%
False Pos
172
19.24%
False Neg
63
7.05%
True Pos
27
3.02%
sfNegative
sfPositive
True label
Predicted label
600
500
400
300
200
100

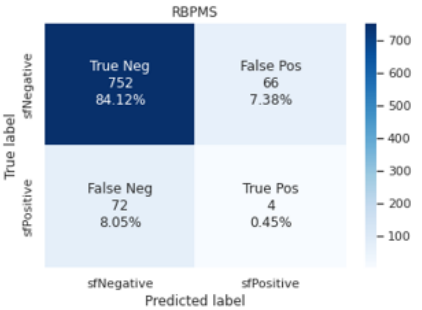

SRSF7
True Neg
701
78.41%
False Pos
89
9.96%
False Neg
78
8.72%
True Pos
26
2.91%
sfNegative
sfPositive
True label
Predicted label
700
600
500
400
300
200
100

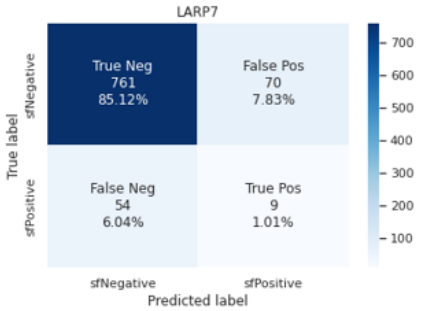

TNRC6A
True Neg
277
30.98%
False Pos
558
62.42%
False Neg
11
1.23%
True Pos
48
5.37%
sfNegative
sfPositive
True label
Predicted label
500
400
300
200
100

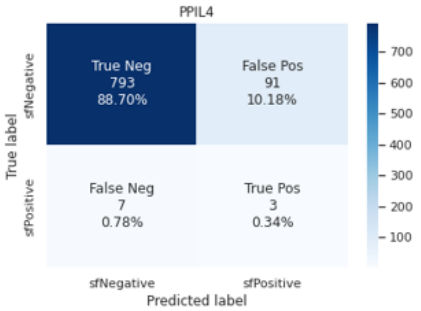

YTHDF2
True Neg
223
24.94%
False Pos
62
6.94%
False Neg
410
45.86%
True Pos
199
22.26%
sfNegative
sfPositive
True label
Predicted label
400
350
300
250
200
150
100

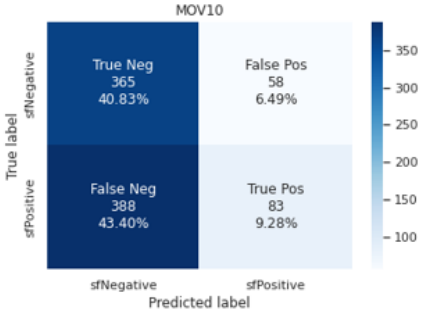

YTHDF1
True Neg
257
28.75%
False Pos
26
2.91%
False Neg
547
61.19%
True Pos
64
7.16%
sfNegative
sfPositive
True label
Predicted label
500
400
300
200
100

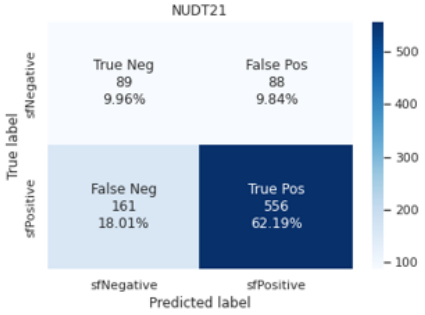

SF3B4
True Neg
372
41.61%
False Pos
34
3.80%
False Neg
425
47.54%
True Pos
63
7.05%
sfNegative
sfPositive
True label
Predicted label
400
350
300
250
200
150
100
50

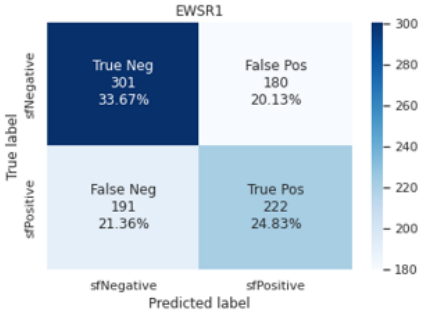

RBPMS
True Neg
752
84.12%
False Pos
66
7.38%
False Neg
72
8.05%
True Pos
4
0.45%
sfNegative
sfPositive
True label
Predicted label
700
600
500
400
300
200
100

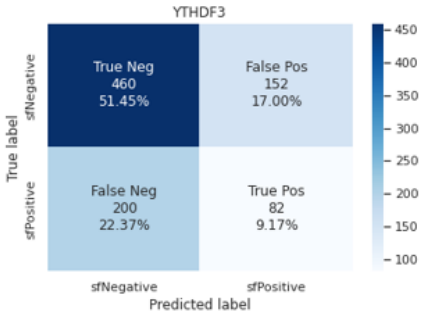

LARP7
True Neg
761
85.12%
False Pos
70
7.83%
False Neg
54
6.04%
True Pos
9
1.01%
sfNegative
sfPositive
True label
Predicted label
700
600
500
400
300
200
100

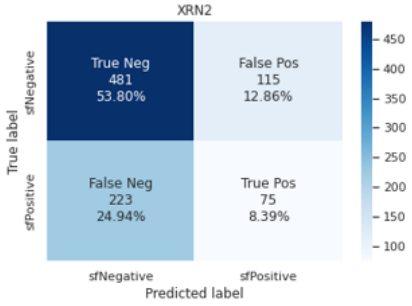

PPIL4
True Neg
793
88.70%
False Pos
91
10.18%
False Neg
7
0.78%
True Pos
3
0.34%
sfNegative
sfPositive
True label
Predicted label
700
600
500
400
300
200
100

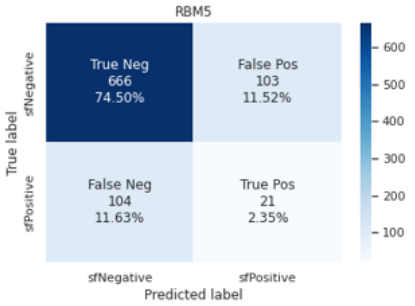

MOV10
True Neg
365
40.83%
False Pos
58
6.49%
False Neg
388
43.40%
True Pos
83
9.28%
sfNegative
sfPositive
True label
sfNegative
sfPositive
Predicted label
350
300
250
200
150
100

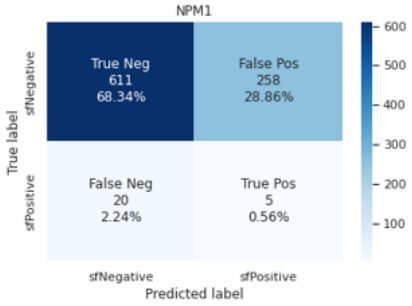

NUDT21
True Neg
89
9.96%
False Pos
88
9.84%
False Neg
161
18.01%
True Pos
556
62.19%
sfNegative
sfPositive
True label
sfNegative
sfPositive
Predicted label
500
400
300
200
100

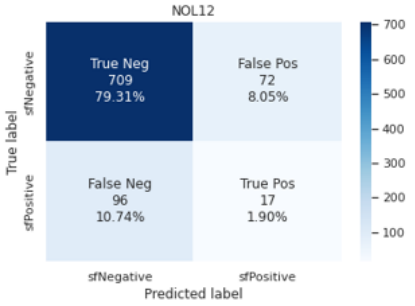

EWSR1
True Neg
301
33.67%
False Pos
180
20.13%
False Neg
191
21.36%
True Pos
222
24.83%
sfNegative
sfPositive
True label
sfNegative
sfPositive
Predicted label
300
280
260
240
220
200
180

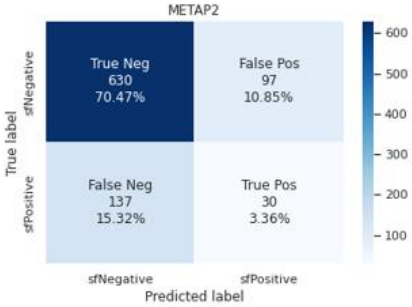

YTHDF3
True Neg
460
51.45%
False Pos
152
17.00%
False Neg
200
22.37%
True Pos
82
9.17%
sfNegative
sfPositive
True label
sfNegative
sfPositive
Predicted label
450
400
350
300
250
200
150
100

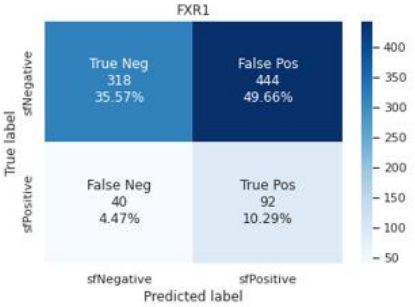

XRN2
True Neg
481
53.80%
False Pos
115
12.86%
False Neg
223
24.94%
True Pos
75
8.39%
sfNegative
sfPositive
True label
sfNegative
sfPositive
Predicted label
450
400
350
300
250
200
150
100

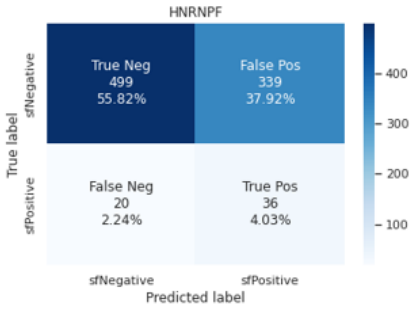

RBM5
True Neg
666
74.50%
False Pos
103
11.52%
False Neg
104
11.63%
True Pos
21
2.35%
sfNegative
sfPositive
True label
sfNegative
sfPositive
Predicted label
600
500
400
300
200
100

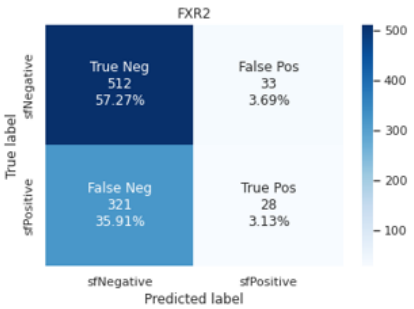

NPM1
True Neg
611
68.34%
False Pos
258
28.86%
False Neg
20
2.24%
True Pos
5
0.56%
sfNegative
sfPositive
True label
sfNegative
sfPositive
Predicted label
600
500
400
300
200
100

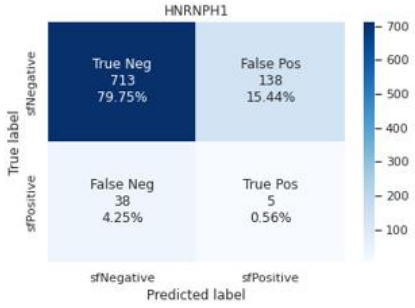

NOL12
True Neg
709
79.31%
False Pos
72
8.05%
False Neg
96
10.74%
True Pos
17
1.90%
sfNegative
sfPositive
True label
sfNegative
sfPositive
Predicted label
700
600
500
400
300
200
100

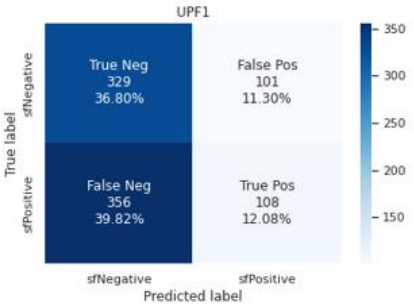

METAP2
True Neg
630
70.47%
False Pos
97
10.85%
False Neg
137
15.32%
True Pos
30
3.36%
sfNegative
sfPositive
True label
sfNegative
sfPositive
Predicted label
600
500
400
300
200
100

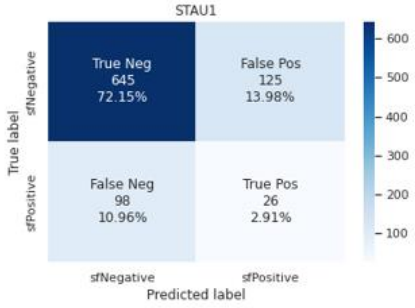

FXR1
True Neg
318
35.57%
False Pos
444
49.66%
False Neg
40
4.47%
True Pos
92
10.29%
sfNegative
sfPositive
True label
sfNegative
sfPositive
Predicted label
400
350
300
250
200
150
100
50

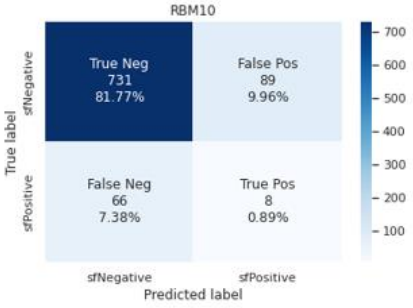

HNRNPF
True Neg
499
55.82%
False Pos
339
37.92%
False Neg
20
2.24%
True Pos
36
4.03%
sfNegative
sfPositive
True label
Predicted label
400
300
200
100

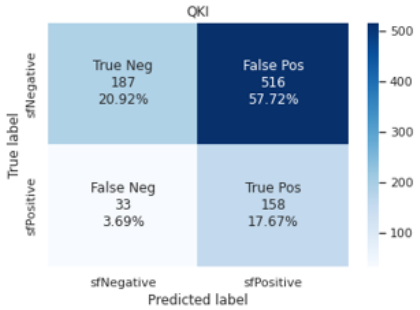

FXR2
True Neg
512
57.27%
False Pos
33
3.69%
False Neg
321
35.91%
True Pos
28
3.13%
sfNegative
sfPositive
True label
Predicted label
500
400
300
200
100

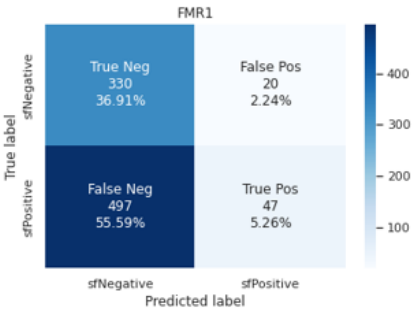

HNRNPH1
True Neg
713
79.75%
False Pos
138
15.44%
False Neg
38
4.25%
True Pos
5
0.56%
sfNegative
sfPositive
True label
Predicted label
700
600
500
400
300
200
100

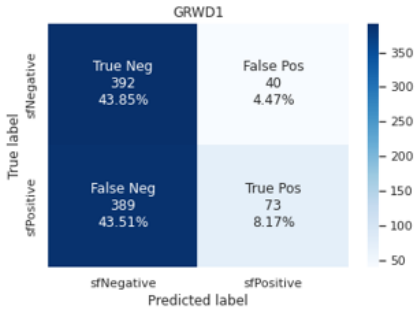

UPF1
True Neg
329
36.80%
False Pos
101
11.30%
False Neg
356
39.82%
True Pos
108
12.08%
sfNegative
sfPositive
True label
Predicted label
350
300
250
200
150

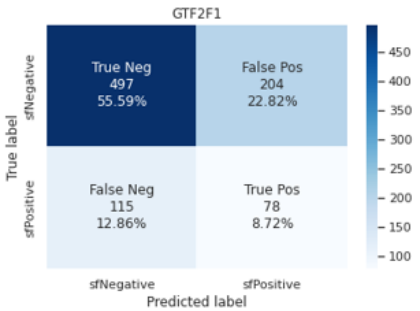

STAU1
True Neg
645
72.15%
False Pos
125
13.98%
False Neg
98
10.96%
True Pos
26
2.91%
sfNegative
sfPositive
True label
Predicted label
600
500
400
300
200
100

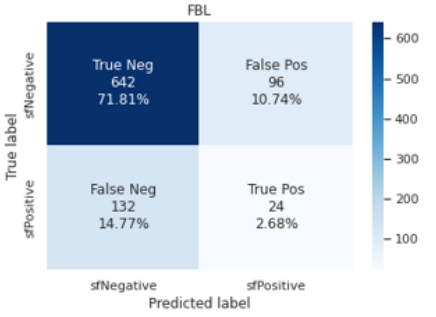

RBM10
True Neg
731
81.77%
False Pos
89
9.96%
False Neg
66
7.38%
True Pos
8
0.89%
sfNegative
sfPositive
True label
Predicted label
700
600
500
400
300
200
100

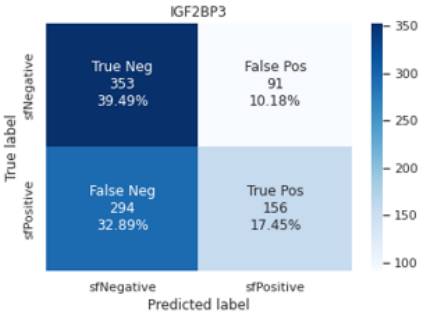

QKI
True Neg
187
20.92%
False Pos
516
57.72%
False Neg
33
3.69%
True Pos
158
17.67%
sfNegative
sfPositive
True label
Predicted label
500
400
300
200
100

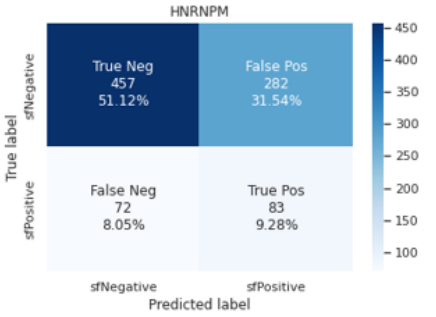

FMR1
True Neg
330
36.91%
False Pos
20
2.24%
False Neg
497
55.59%
True Pos
47
5.26%
sfNegative
sfPositive
True label
Predicted label
400
300
200
100

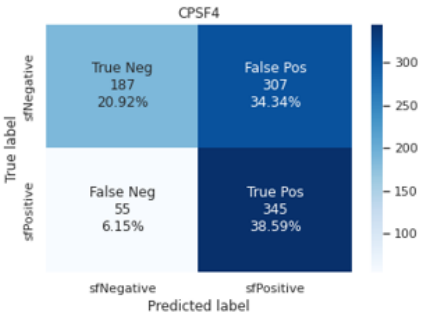

GRWD1
True Neg
392
43.85%
False Pos
40
4.47%
False Neg
389
43.51%
True Pos
73
8.17%
sfNegative
sfPositive
True label
Predicted label
350
300
250
200
150
100
50

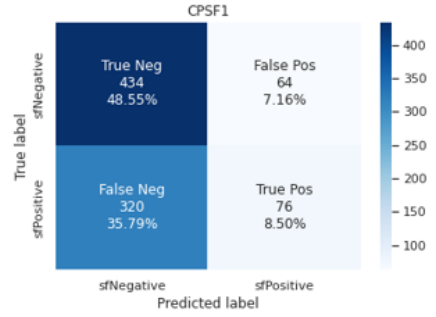

GTF2F1
True Neg
497
55.59%
False Pos
204
22.82%
False Neg
115
12.86%
True Pos
78
8.72%
sfNegative
sfPositive
True label
Predicted label
450
400
350
300
250
200
150
100

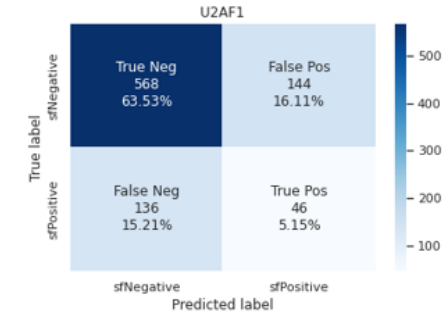

FBL
True Neg
642
71.81%
False Pos
96
10.74%
False Neg
132
14.77%
True Pos
24
2.68%
True label
sfNegative
sfPositive
sfNegative
sfPositive
Predicted label
600
500
400
300
200
100

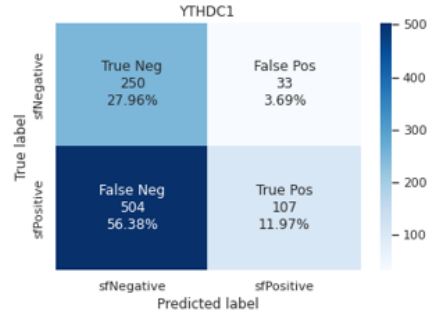

IGF2BP3
True Neg
353
39.49%
False Pos
91
10.18%
False Neg
294
32.89%
True Pos
156
17.45%
True label
sfNegative
sfPositive
sfNegative
sfPositive
Predicted label
350
300
250
200
150
100

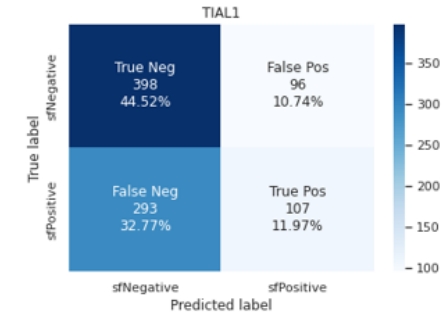

HNRNPM
True Neg
457
51.12%
False Pos
282
31.54%
False Neg
72
8.05%
True Pos
83
9.28%
True label
sfNegative
sfPositive
sfNegative
sfPositive
Predicted label
450
400
350
300
250
200
150
100

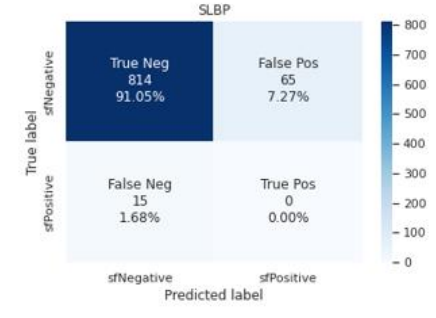

CPSF4
True Neg
187
20.92%
False Pos
307
34.34%
False Neg
55
6.15%
True Pos
345
38.59%
True label
sfNegative
sfPositive
sfNegative
sfPositive
Predicted label
300
250
200
150
100

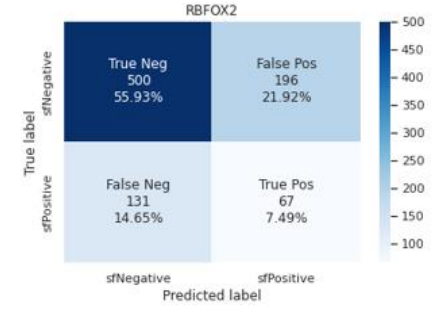

CPSF1
True Neg
434
48.55%
False Pos
64
7.16%
False Neg
320
35.79%
True Pos
76
8.50%
True label
sfNegative
sfPositive
sfNegative
sfPositive
Predicted label
400
350
300
250
200
150
100

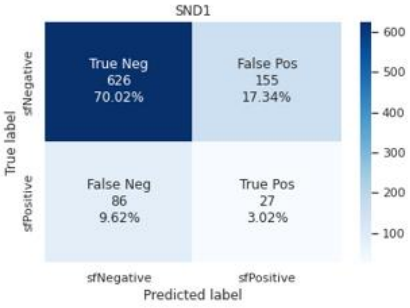

U2AF1
True Neg
568
63.53%
False Pos
144
16.11%
False Neg
136
15.21%
True Pos
46
5.15%
True label
sfNegative
sfPositive
sfNegative
sfPositive
Predicted label
500
400
300
200
100

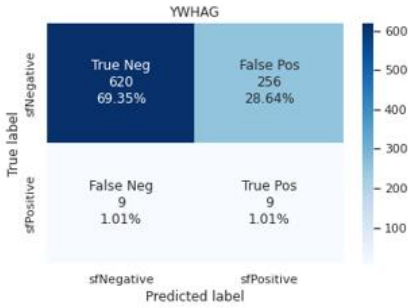

YTHDC1
True Neg
250
27.96%
False Pos
33
3.69%
False Neg
504
56.38%
True Pos
107
11.97%
True label
sfNegative
sfPositive
sfNegative
sfPositive
Predicted label
500
400
300
200
100

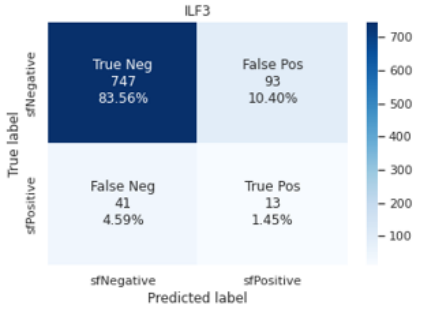

TIAL1
True Neg
398
44.52%
False Pos
96
10.74%
False Neg
293
32.77%
True Pos
107
11.97%
True label
sfNegative
sfPositive
sfNegative
sfPositive
Predicted label
350
300
250
200
150
100

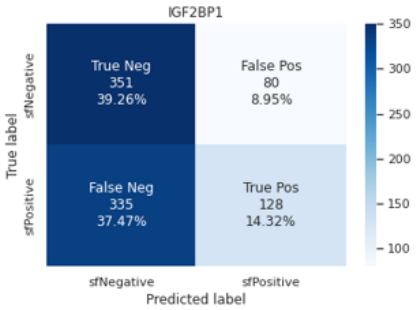

SLBP
True Neg
814
91.05%
False Pos
65
7.27%
False Neg
15
1.68%
True Pos
0
0.00%
True label
sfNegative
sfPositive
sfNegative
sfPositive
Predicted label
800
700
600
500
400
300
200
100
0

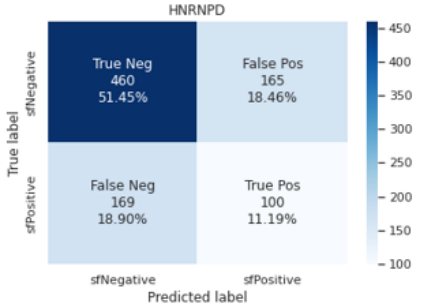

RBFOX2
True Neg
500
55.93%
False Pos
196
21.92%
False Neg
131
14.65%
True Pos
67
7.49%
True label
sfNegative
sfPositive
sfNegative
sfPositive
Predicted label
500
450
400
350
300
250
200
150
100

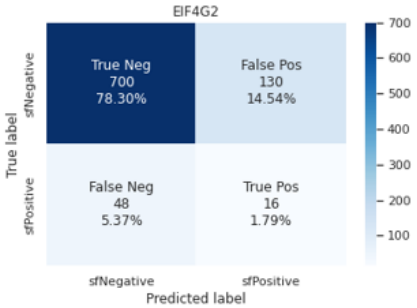

SND1
True Neg
626
70.02%
False Pos
155
17.34%
False Neg
86
9.62%
True Pos
27
3.02%
sfNegative
sfPositive
True label
Predicted label
600
500
400
300
200
100

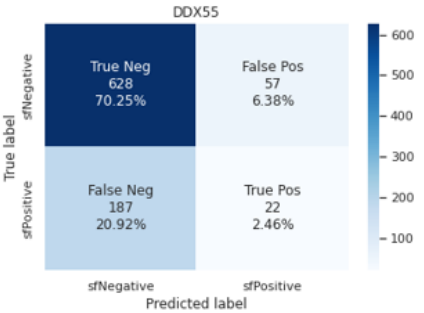

YWHAG
True Neg
620
69.35%
False Pos
256
28.64%
False Neg
9
1.01%
True Pos
9
1.01%
sfNegative
sfPositive
True label
Predicted label
600
500
400
300
200
100

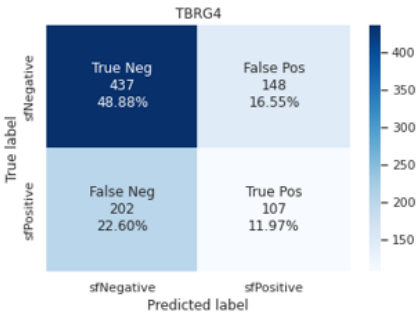

ILF3
True Neg
747
83.56%
False Pos
93
10.40%
False Neg
41
4.59%
True Pos
13
1.45%
sfNegative
sfPositive
True label
Predicted label
700
600
500
400
300
200
100

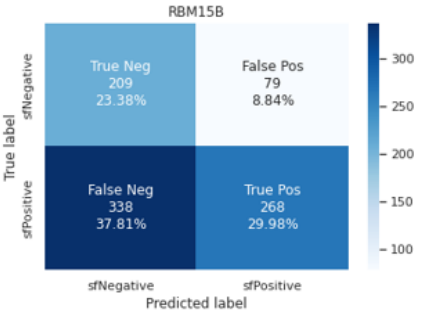

IGF2BP1
True Neg
351
39.26%
False Pos
80
8.95%
False Neg
335
37.47%
True Pos
128
14.32%
sfNegative
sfPositive
True label
Predicted label
350
300
250
200
150
100

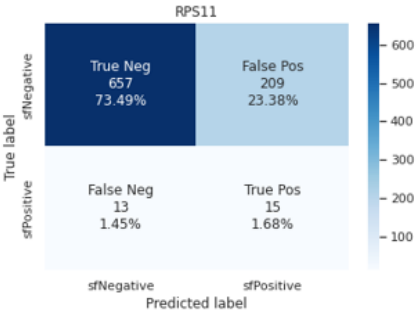

HNRNPD
True Neg
460
51.45%
False Pos
165
18.46%
False Neg
169
18.90%
True Pos
100
11.19%
sfNegative
sfPositive
True label
Predicted label
450
400
350
300
250
200
150
100

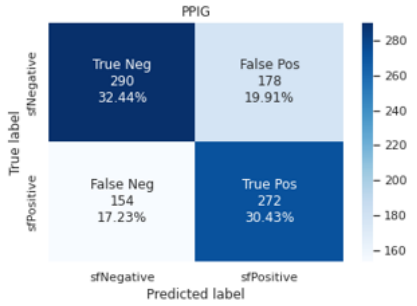

EIF4G2
True Neg
700
78.30%
False Pos
130
14.54%
False Neg
48
5.37%
True Pos
16
1.79%
sfNegative
sfPositive
True label
Predicted label
700
600
500
400
300
200
100

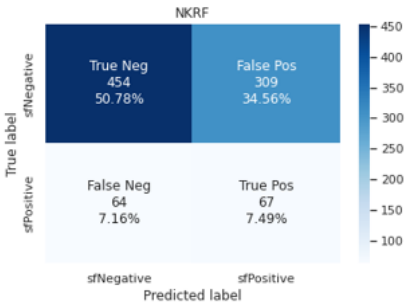

DDX55
True Neg
628
70.25%
False Pos
57
6.38%
False Neg
187
20.92%
True Pos
22
2.46%
sfNegative
sfPositive
True label
Predicted label
600
500
400
300
200
100

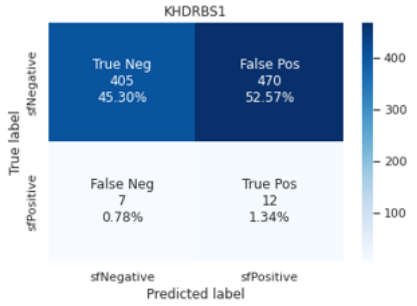

TBRG4
True Neg
437
48.88%
False Pos
148
16.55%
False Neg
202
22.60%
True Pos
107
11.97%
sfNegative
sfPositive
True label
Predicted label
400
350
300
250
200
150

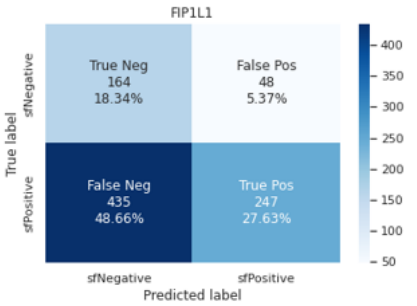

RBM15B
True Neg
209
23.38%
False Pos
79
8.84%
False Neg
338
37.81%
True Pos
268
29.98%
sfNegative
sfPositive
True label
Predicted label
300
250
200
150
100

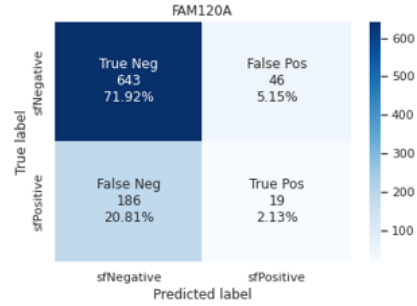

RPS11
True Neg
657
73.49%
False Pos
209
23.38%
False Neg
13
1.45%
True Pos
15
1.68%
sfNegative
sfPositive
True label
Predicted label
600
500
400
300
200
100

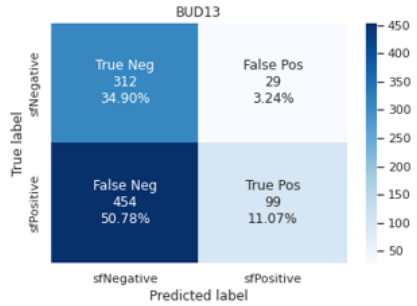

PPIG
True Neg
290
32.44%
False Pos
178
19.91%
False Neg
154
17.23%
True Pos
272
30.43%
sfNegative
sfPositive
True label
Predicted label
280
260
240
220
200
180
160

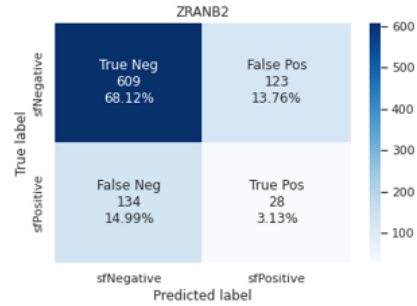

NKRF
True Neg
454
50.78%
False Pos
309
34.56%
False Neg
64
7.16%
True Pos
67
7.49%
sfNegative
sfPositive
True label
Predicted label
450
400
350
300
250
200
150
100

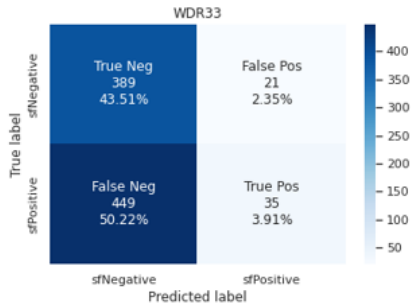

KHDRBS1
True Neg
405
45.30%
False Pos
470
52.57%
False Neg
7
0.78%
True Pos
12
1.34%
sfNegative
sfPositive
True label
Predicted label
400
300
200
100

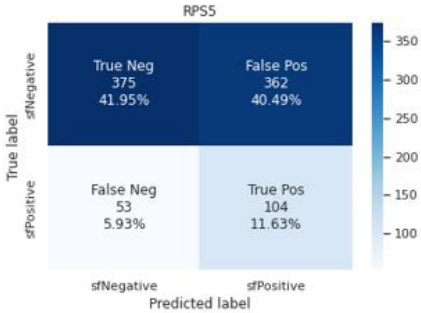

FIP1L1
True Neg
164
18.34%
False Pos
48
5.37%
False Neg
435
48.66%
True Pos
247
27.63%
sfNegative
sfPositive
True label
Predicted label
400
350
300
250
200
150
100
50

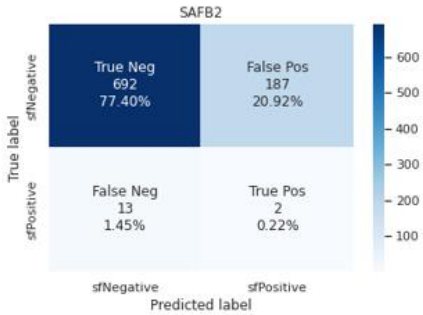

FAM120A
True Neg
643
71.92%
False Pos
46
5.15%
False Neg
186
20.81%
True Pos
19
2.13%
sfNegative
sfPositive
True label
Predicted label
600
500
400
300
200
100

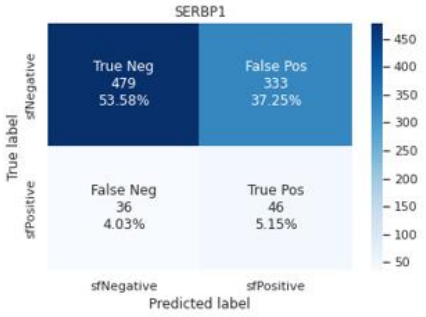

BUD13
True Neg
312
34.90%
False Pos
29
3.24%
False Neg
454
50.78%
True Pos
99
11.07%
sfNegative
sfPositive
True label
Predicted label
450
400
350
300
250
200
150
100
50

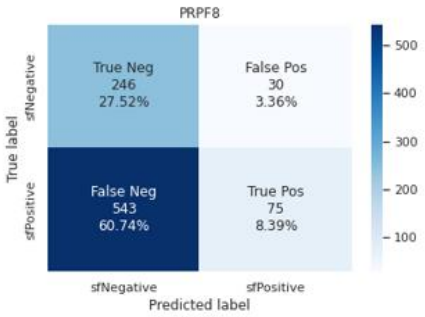

ZRANB2
True Neg
609
68.12%
False Pos
123
13.76%
False Neg
134
14.99%
True Pos
28
3.13%
sfNegative
sfPositive
True label
Predicted label
600
500
400
300
200
100

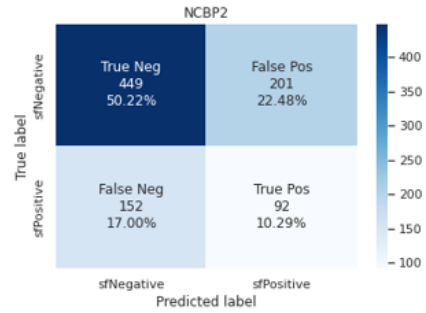

WDR33
True Neg
389
43.51%
False Pos
21
2.35%
False Neg
449
50.22%
True Pos
35
3.91%
sfNegative
sfPositive
True label
Predicted label
400
350
300
250
200
150
100
50

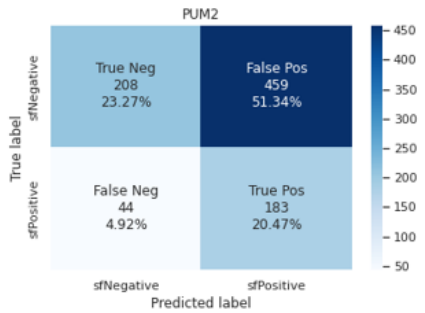

RPS5
True Neg
375
41.95%
False Pos
362
40.49%
False Neg
53
5.93%
True Pos
104
11.63%
sfNegative
sfPositive
True label
Predicted label
350
300
250
200
150
100

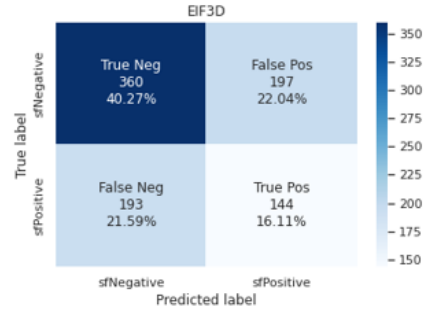

SAFB2
True Neg
692
77.40%
False Pos
187
20.92%
False Neg
13
1.45%
True Pos
2
0.22%
sfNegative
sfPositive
True label
Predicted label
600
500
400
300
200
100

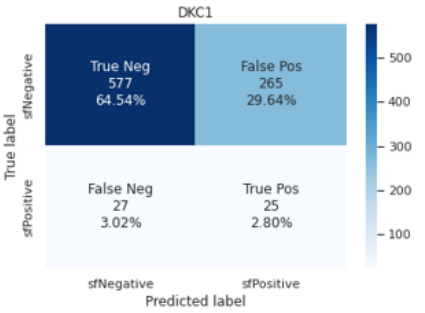

SERBP1
True Neg
479
53.58%
False Pos
333
37.25%
False Neg
36
4.03%
True Pos
46
5.15%
True label
sfNegative
sfPositive
sfNegative
sfPositive
Predicted label
450
400
350
300
250
200
150
100
50

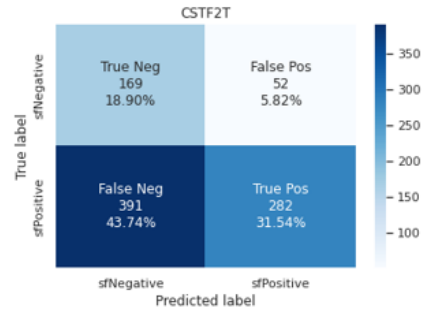

PRPF8
True Neg
246
27.52%
False Pos
30
3.36%
False Neg
543
60.74%
True Pos
75
8.39%
True label
sfNegative
sfPositive
sfNegative
sfPositive
Predicted label
500
400
300
200
100

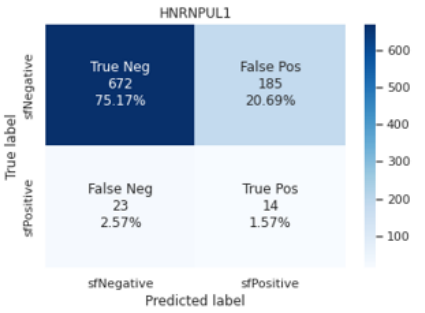

NCBP2
True Neg
449
50.22%
False Pos
201
22.48%
False Neg
152
17.00%
True Pos
92
10.29%
True label
sfNegative
sfPositive
sfNegative
sfPositive
Predicted label
400
350
300
250
200
150
100

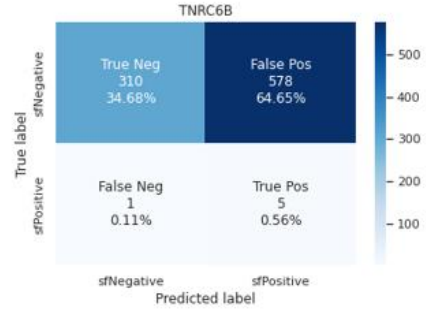

PUM2
True Neg
208
23.27%
False Pos
459
51.34%
False Neg
44
4.92%
True Pos
183
20.47%
True label
sfNegative
sfPositive
sfNegative
sfPositive
Predicted label
450
400
350
300
250
200
150
100
50

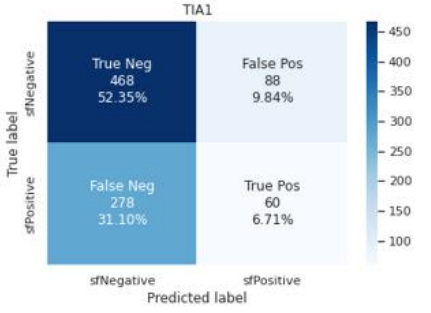

EIF3D
True Neg
360
40.27%
False Pos
197
22.04%
False Neg
193
21.59%
True Pos
144
16.11%
True label
sfNegative
sfPositive
sfNegative
sfPositive
Predicted label
350
325
300
275
250
225
200
175
150

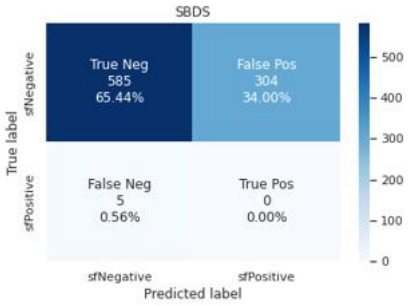

DKC1
True Neg
577
64.54%
False Pos
265
29.64%
False Neg
27
3.02%
True Pos
25
2.80%
True label
sfNegative
sfPositive
sfNegative
sfPositive
Predicted label
500
400
300
200
100

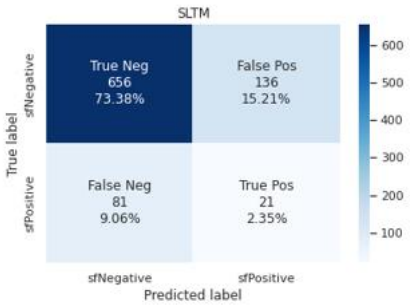

CSTF2T
True Neg
169
18.90%
False Pos
52
5.82%
False Neg
391
43.74%
True Pos
282
31.54%
True label
sfNegative
sfPositive
sfNegative
sfPositive
Predicted label
350
300
250
200
150
100

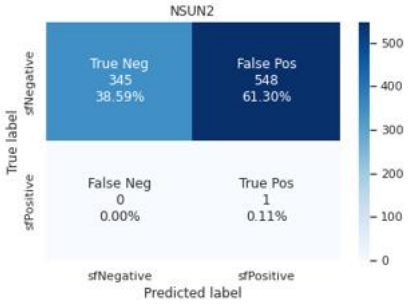

HNRNPUL1
True Neg
672
75.17%
False Pos
185
20.69%
False Neg
23
2.57%
True Pos
14
1.57%
True label
sfNegative
sfPositive
sfNegative
sfPositive
Predicted label
600
500
400
300
200
100

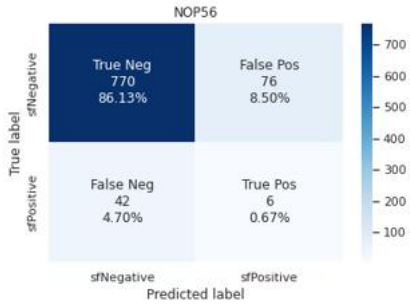

TNRC6B
True Neg
310
34.68%
False Pos
578
64.65%
False Neg
1
0.11%
True Pos
5
0.56%
True label
sfNegative
sfPositive
sfNegative
sfPositive
Predicted label
500
400
300
200
100

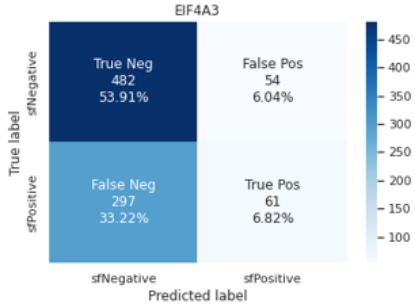

TIA1
True Neg
468
52.35%
False Pos
88
9.84%
False Neg
278
31.10%
True Pos
60
6.71%
True label
sfNegative
sfPositive
sfNegative
sfPositive
Predicted label
450
400
350
300
250
200
150
100

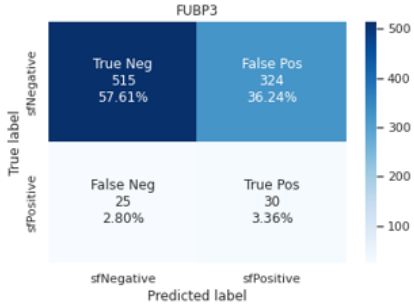

SBDS
True Neg
585
65.44%
False Pos
304
34.00%
False Neg
5
0.56%
True Pos
0
0.00%
True label
sfNegative
sfPositive
sfNegative
sfPositive
Predicted label
500
400
300
200
100
0

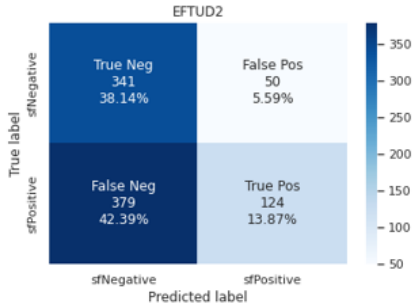

SLTM
True Neg
656
73.38%
False Pos
136
15.21%
False Neg
81
9.06%
True Pos
21
2.35%
True label
sfNegative
sfPositive
sfNegative
sfPositive
Predicted label
600
500
400
300
200
100

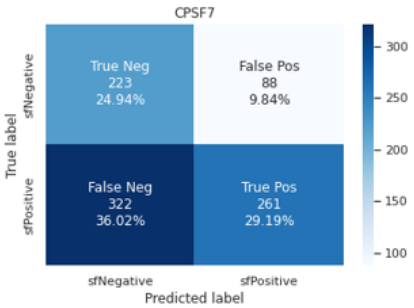

NSUN2
True Neg
345
38.59%
False Pos
548
61.30%
False Neg
0
0.00%
True Pos
1
0.11%
True label
sfNegative
sfPositive
sfNegative
sfPositive
Predicted label
500
400
300
200
100
0

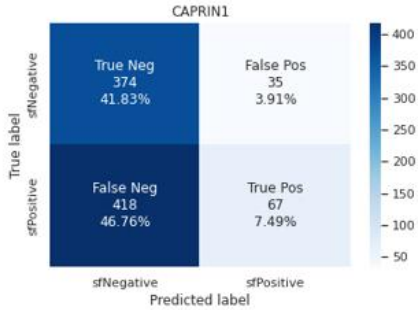

NOP56
True Neg
770
86.13%
False Pos
76
8.50%
False Neg
42
4.70%
True Pos
6
0.67%
True label
sfNegative
sfPositive
sfNegative
sfPositive
Predicted label
700
600
500
400
300
200
100

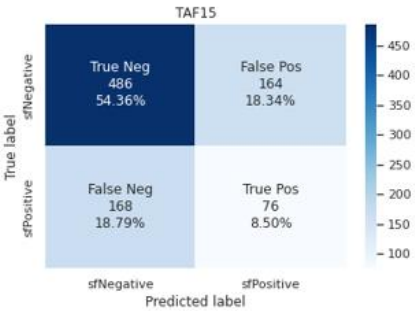

EIF4A3
True Neg
482
53.91%
False Pos
54
6.04%
False Neg
297
33.22%
True Pos
61
6.82%
True label
sfNegative
sfPositive
sfNegative
sfPositive
Predicted label
450
400
350
300
250
200
150
100

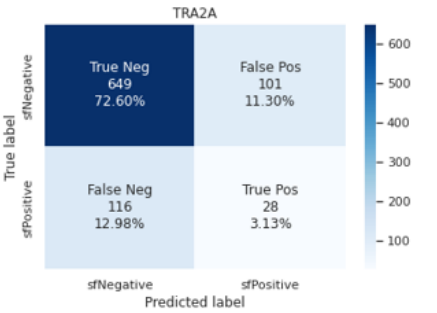

FUBP3
True Neg
515
57.61%
False Pos
324
36.24%
False Neg
25
2.80%
True Pos
30
3.36%
True label
sfNegative
sfPositive
sfNegative
sfPositive
Predicted label
500
400
300
200
100

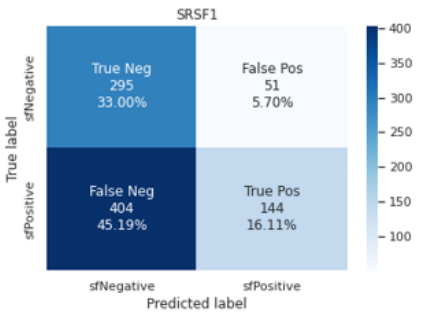

EFTUD2
True Neg
341
38.14%
False Pos
50
5.59%
False Neg
379
42.39%
True Pos
124
13.87%
True label
sfNegative
sfPositive
sfNegative
sfPositive
Predicted label
350
300
250
200
150
100
50

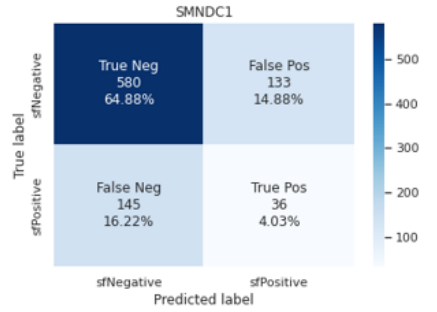

CPSF7
True Neg
223
24.94%
False Pos
88
9.84%
False Neg
322
36.02%
True Pos
261
29.19%
True label
sfNegative
sfPositive
sfNegative
sfPositive
Predicted label
300
250
200
150
100

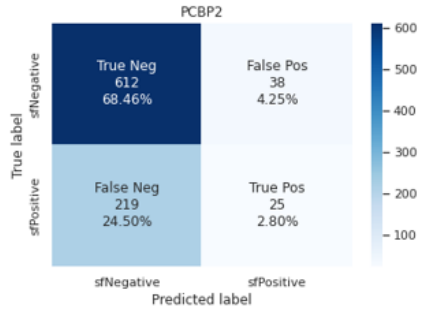

CAPRIN1
True Neg
374
41.83%
False Pos
35
3.91%
False Neg
418
46.76%
True Pos
67
7.49%
True label
sfNegative
sfPositive
sfNegative
sfPositive
Predicted label
400
350
300
250
200
150
100
50

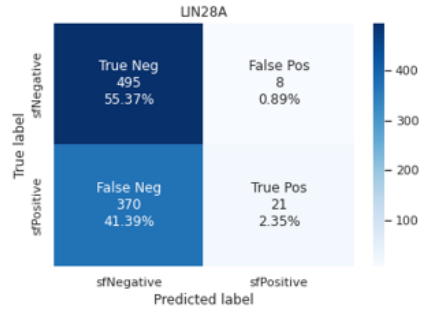

TAF15
True Neg
486
54.36%
False Pos
164
18.34%
False Neg
168
18.79%
True Pos
76
8.50%
True label
sfNegative
sfPositive
sfNegative
sfPositive
Predicted label
450
400
350
300
250
200
150
100

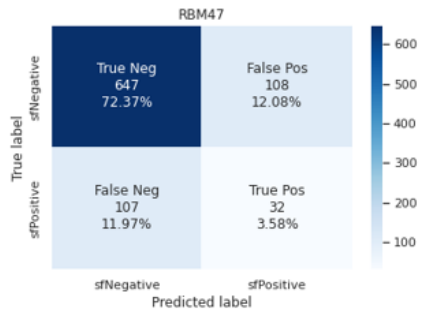

TRA2A
True Neg
649
72.60%
False Pos
101
11.30%
False Neg
116
12.98%
True Pos
28
3.13%
sfNegative
sfPositive
True label
sfNegative
sfPositive
Predicted label
600
500
400
300
200
100

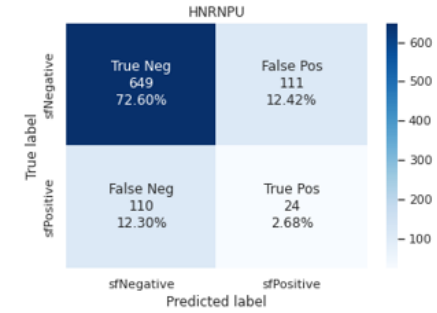

SRSF1
True Neg
295
33.00%
False Pos
51
5.70%
False Neg
404
45.19%
True Pos
144
16.11%
sfNegative
sfPositive
True label
sfNegative
sfPositive
Predicted label
400
350
300
250
200
150
100

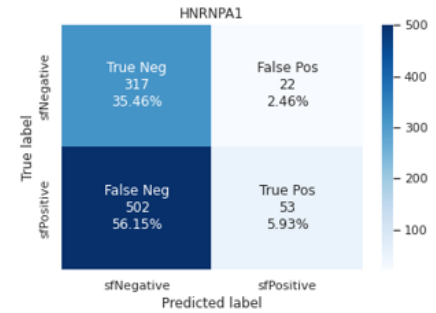

SMNDC1
True Neg
580
64.88%
False Pos
133
14.88%
False Neg
145
16.22%
True Pos
36
4.03%
sfNegative
sfPositive
True label
sfNegative
sfPositive
Predicted label
500
400
300
200
100

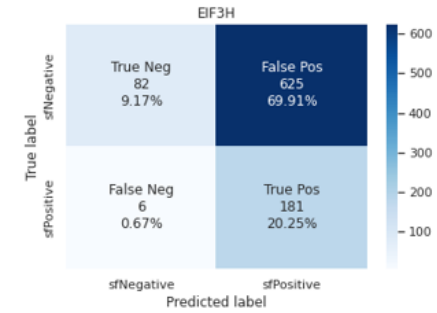

PCBP2
True Neg
612
68.46%
False Pos
38
4.25%
False Neg
219
24.50%
True Pos
25
2.80%
sfNegative
sfPositive
True label
sfNegative
sfPositive
Predicted label
600
500
400
300
200
100

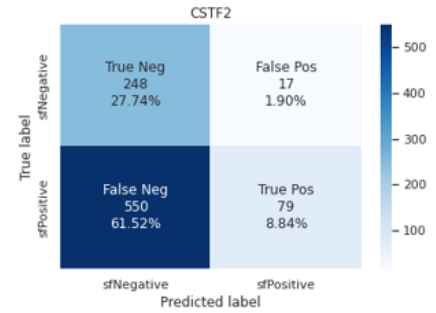

LIN28A
True Neg
495
55.37%
False Pos
8
0.89%
False Neg
370
41.39%
True Pos
21
2.35%
sfNegative
sfPositive
True label
sfNegative
sfPositive
Predicted label
400
300
200
100

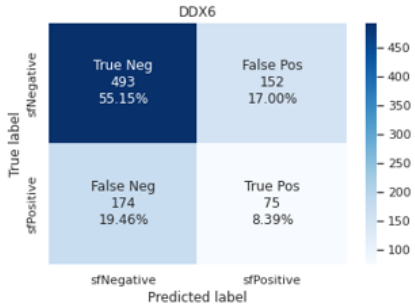

RBM47
True Neg
647
72.37%
False Pos
108
12.08%
False Neg
107
11.97%
True Pos
32
3.58%
sfNegative
sfPositive
True label
sfNegative
sfPositive
Predicted label
600
500
400
300
200
100

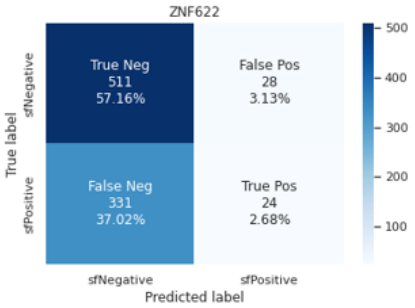

HNRNPU
True Neg
649
72.60%
False Pos
111
12.42%
False Neg
110
12.30%
True Pos
24
2.68%
sfNegative
sfPositive
True label
sfNegative
sfPositive
Predicted label
600
500
400
300
200
100

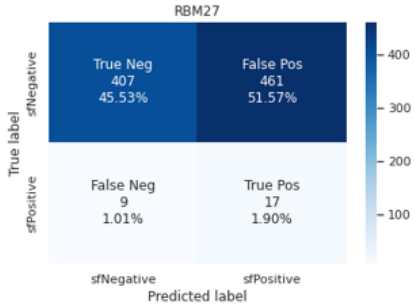

HNRNPA1
True Neg
317
35.46%
False Pos
22
2.46%
False Neg
502
56.15%
True Pos
53
5.93%
sfNegative
sfPositive
True label
sfNegative
sfPositive
Predicted label
500
400
300
200
100

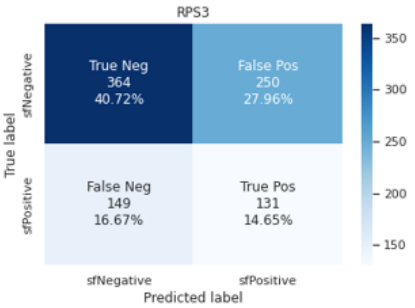

EIF3H
True Neg
82
9.17%
False Pos
625
69.91%
False Neg
6
0.67%
True Pos
181
20.25%
sfNegative
sfPositive
True label
sfNegative
sfPositive
Predicted label
600
500
400
300
200
100

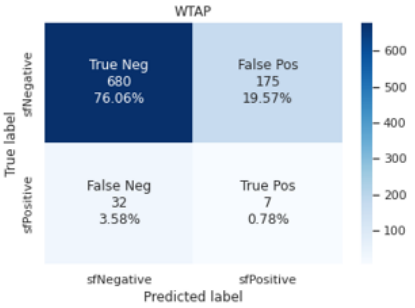

CSTF2
True Neg
248
27.74%
False Pos
17
1.90%
False Neg
550
61.52%
True Pos
79
8.84%
sfNegative
sfPositive
True label
sfNegative
sfPositive
Predicted label
500
400
300
200
100

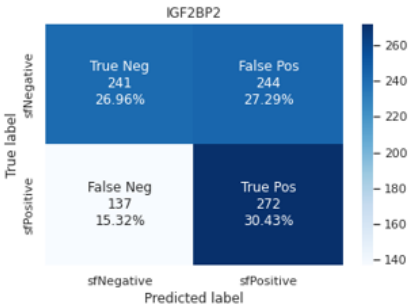

DDX6
True Neg
493
55.15%
False Pos
152
17.00%
False Neg
174
19.46%
True Pos
75
8.39%
sfNegative
sfPositive
True label
sfNegative
sfPositive
Predicted label
450
400
350
300
250
200
150
100

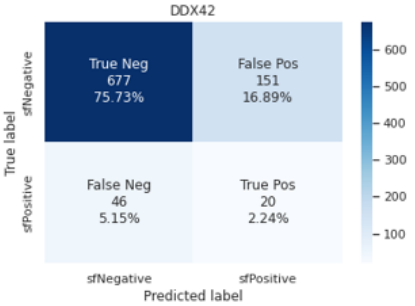

ZNF622
True Neg
511
57.16%
False Pos
28
3.13%
False Neg
331
37.02%
True Pos
24
2.68%
sfNegative
sfPositive
True label
sfNegative
sfPositive
Predicted label
500
400
300
200
100

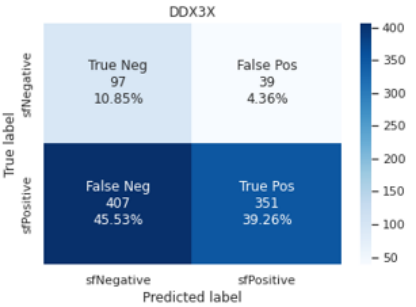

RBM27
True Neg
407
45.53%
False Pos
461
51.57%
False Neg
9
1.01%
True Pos
17
1.90%
sfNegative
sfPositive
True label
sfNegative
sfPositive
Predicted label
400
300
200
100

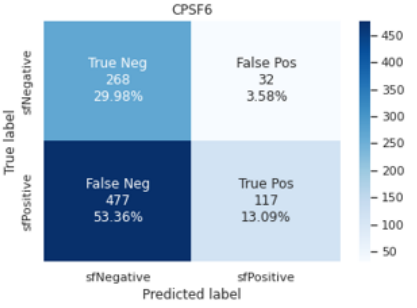

RPS3
True Neg
364
40.72%
False Pos
250
27.96%
False Neg
149
16.67%
True Pos
131
14.65%
sfNegative
sfPositive
True label
sfNegative
sfPositive
Predicted label
350
300
250
200
150

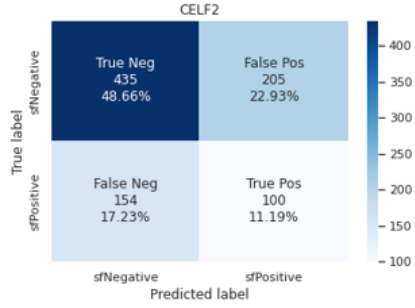

WTAP
True Neg
680
76.06%
False Pos
175
19.57%
False Neg
32
3.58%
True Pos
7
0.78%
sfNegative
sfPositive
True label
sfNegative
sfPositive
Predicted label
600
500
400
300
200
100


IGF2BP2
True Neg
241
26.96%
False Pos
244
27.29%
False Neg
137
15.32%
True Pos
272
30.43%
sfNegative
sfPositive
True label
sfNegative
sfPositive
Predicted label
260
240
220
200
180
160
140


DDX42
True Neg
677
75.73%
False Pos
151
16.89%
False Neg
46
5.15%
True Pos
20
2.24%
sfNegative
sfPositive
True label
sfNegative
sfPositive
Predicted label
600
500
400
300
200
100


DDX3X
True Neg
97
10.85%
False Pos
39
4.36%
False Neg
407
45.53%
True Pos
351
39.26%
sfNegative
sfPositive
True label
sfNegative
sfPositive
Predicted label
400
350
300
250
200
150
100
50


CPSF6
True Neg
268
29.98%
False Pos
32
3.58%
False Neg
477
53.36%
True Pos
117
13.09%
sfNegative
sfPositive
True label
sfNegative
sfPositive
Predicted label
450
400
350
300
250
200
150
100
50


CELF2
True Neg
435
48.66%
False Pos
205
22.93%
False Neg
154
17.23%
True Pos
100
11.19%
sfNegative
sfPositive
True label
sfNegative
sfPositive
Predicted label
400
350
300
250
200
150
100

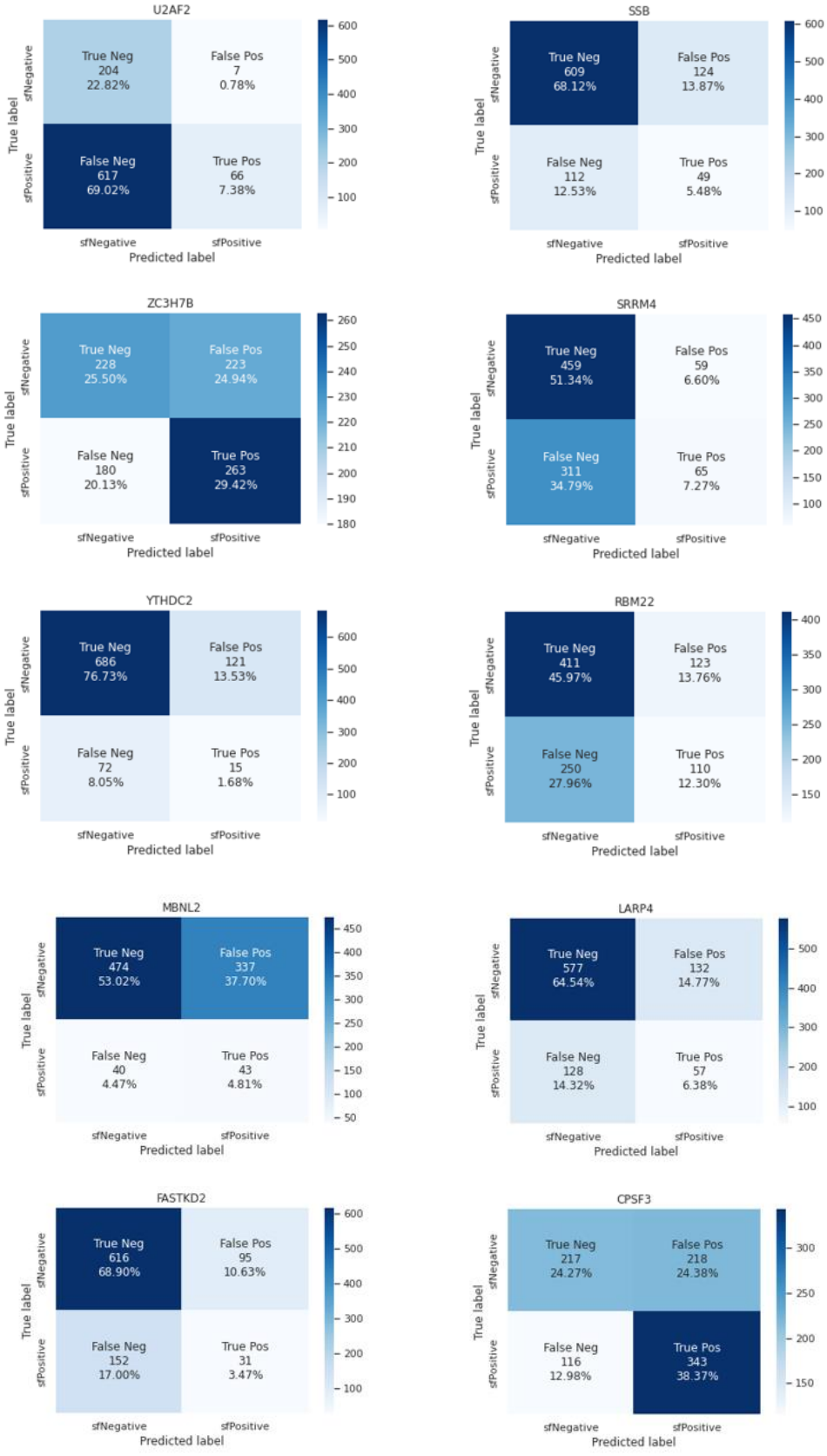


Figure 65. Confusion matrix of al the splicing factor genes

Bibliography

Adam. (n.d.). Retrieved from https://pytorch.org/docs/stable/generated/torch.optim.Adam.html

Bandyopadhyay, H. (2022, July 19). *Autoencoders in Deep Learning: Tutorial & Use Cases [2022]*. Retrieved from https://www.v7labs.com/blog/autoencoders-guide#:~:text=An%20autoencoder%20is%20an%20unsupervised,even%20generation%20of%20image%20data

Cantoral, P. (2021). *Stochastic Gradient Descent con MOMENTUM. Algoritmos de optimización para redes neuronales.* Retrieved from https://www.youtube.com/watch?v=ryZv18FVwCI&ab_channel=PepeCantoral%2CPh.D.

Chablani, M. (2017, July 14). *Gradient descent algorithms and adaptive learning rate adjustment methods*. Retrieved from Towards Data Science: https://towardsdatascience.com/gradient-descent-algorithms-and-adaptive-learning-rate-adjustment-methods-79c701b086be

Fernando Carazo, M. G.-B. (2019). Integration of CLIP experiments of RNA-binding proteins: a novel approach to predict context-dependent splicing factors from transcriptomic data. *BMC Genomics*.

Gi Bae Kim, Y. G. (2020). DeepTFactor: A deep learning-based tool for the prediction of transcription factors. *PNAS*.

Hutter, I. L. (2019). DECOUPLED WEIGHT DECAY REGULARIZATION.

Jesús. (n.d.). *Optimizadores en Deep Learning*. Retrieved from DataSmarts: https://datasmarts.net/es/que-es-un-optimizador-y-para-que-se-usa-en-deep-learning/

JJ. (2016, March 23). *MAE and RMSE — Which Metric is Better?* Retrieved from Medium: https://medium.com/human-in-a-machine-world/mae-and-rmse-which-metric-is-better-e60ac3bde13d

K.Koo1MattPloenzke2, P. (2020). Deep learning for inferring transcription factor binding sites. *ScienceDirect*.

Lirong Zhang, Y. Y. (2022). A deep learning model to identify gene expression level using cobinding transcription factor signals. *Briefings in Bioinformatics*.

Peter K.Koo, M. P. (2020). Deep learning for inferring transcription factor binding sites. *ScienceDirect*.

PyTorch. (n.d.). Retrieved from https://pytorch.org/docs/stable/generated/torch.optim.SGD.html

PyTorch. (n.d.). Retrieved from https://pytorch.org/docs/stable/generated/torch.optim.Adagrad.html

PyTorch. (n.d.). Retrieved from https://pytorch.org/docs/stable/generated/torch.optim.Adadelta.html

PyTorch. (n.d.). Retrieved from https://pytorch.org/docs/stable/generated/torch.optim.AdamW.html

PyTorch. (n.d.). *Adadelta*. Retrieved from https://pytorch.org/docs/stable/generated/torch.optim.Adadelta.html

PyTorch. (n.d.). *Adagrad*. Retrieved from https://pytorch.org/docs/stable/generated/torch.optim.Adagrad.html

Pytorch. (n.d.). *RMSPROP*. Retrieved from https://pytorch.org/docs/stable/generated/torch.optim.RMSprop.html

Rasmus Magnusson, J. N. (2022). Deep neural network prediction of genome-wide transcriptome signatures – beyond the Black-box. *Systems Biology and Applications*.

Sanjiv K. Dwivedi, A. T. (2020). Deriving disease modules from the compressed transcriptional space embedded in a deep autoencoder. *Nature communications*.

Sergey Ioffe, C. S. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *Proceedings of the 32nd International Conference on Machine Learning*, (pp. 37:448-456).

Szegedy, S. I. (n.d.). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *Proceedings of the 32nd International Conference on Machine Learning, PMLR 37:448-456, 2015.*

The Cancer Genome Atlas Program. (n.d.). Retrieved from https://www.cancer.gov/about-nci/organization/ccg/research/structural-genomics/tcga

Weights & Biases. (n.d.). Retrieved from https://wandb.ai/site