University of Murcia

Deep recognition of regulatory DNA sequences

Applications to the identification of 3'UTR regions of the human genome

Author: Jaime Martínez Legaz
Tutor: Juan Antonio Botía Blaya

June 2020 – Degree in Computing Engineering

Acknowledgements

First and foremost, I would like to express my sincerest gratitude to my supervisor, Prof. Juan Antonio Botía Blaya, for his patience and his invaluable help, not limited to this project, as he was happy to help me whenever I was in need of some aid. He has been an excellent supervisor, and I could not ask for a better one.

I am also extremely grateful to be able to participate in the joint project between Astex Pharmaceuticals, University College London and the University of Murcia about identification of 3'UTR and their applications in neurological diseases. Specially, thanks to Siddhart Sethi and the other members of the team, Harpreet Saini and Mina Ryten, for this offer and all the assistance given during this project. Their support has been tremendous, and this project would not have been a shadow of what it is now without it.

Signed statement of authorship

D. Jaime Martínez Legaz con DNI 48854925G, estudiante de la titulación de Grado en Ingeniería Informática de la Universidad de Murcia y autor del TF titulado "Deep recognition of regulatory DNA sequences: Applications to the identification of 3'UTR regions of the human genome".

De acuerdo con el Reglamento por el que se regulan los Trabajos Fin de Grado y de Fin de Máster en la Universidad de Murcia (aprobado C. de Gob. 30-04-2015, modificado 22-04-2016 y 28-09-2018), así como la normativa interna para la oferta, asignación, elaboración y defensa delos Trabajos Fin de Grado y Fin de Máster de las titulaciones impartidas en la Facultad de Informática de la Universidad de Murcia (aprobada en Junta de Facultad 27-11-2015)

DECLARO:

Que el Trabajo Fin de Máster presentado para su evaluación es original y de elaboración personal. Todas las fuentes utilizadas han sido debidamente citadas. Así mismo, declara que no incumple ningún contrato de confidencialidad, ni viola ningún derecho de propiedad intelectual e industrial

Murcia, a 14 de Junio de 2020

Fdo.: Jaime Martínez Legaz Autor del proyecto

Abstract

This project is a follow up to our Bachelor's Thesis "Deep recognition of regulatory DNA sequences", in which we develop a methodology that allows us to investigate sequence level differences based on elements of the human genome [1].

In this study we try to develop a better understanding of the problem, exploring new possibilities and novel techniques, in order to improve our methods and results.

To achieve this, we established a new methodology for making predictions at a genomic level. For that, we have been working in a different experiment from the previous project, with different sequences, trying to polish our procedure working in a more viable enterprise.

Our first objective was to know how Convolutional Neural Networks can help in this task of distinguishing sequences only by their composition, relying solely on their characters and patterns to tell appart the different types of sequence. This goal has already been introduced and worked on in our previous work.

Next, we decided to improve our understanding of the problem, studying the results and analysing them in depth to have a better grasp of the context and decide what our next step would be.

After that, our next goal was to explain the results with the Layer-wise Relevance Propagation (LRP) algorithm, trying to learn what does the model do in order to classify a sequence as positive or negative. This way, we could obtain new knowledge about the problem, and this knowledge could help us revise and improve our methodology again.

Our fourth question was checking if a k-mer analysis could support or compliment the results obtained by LRP, helping us with a second perspective. And, with this, we came to the final question, asking ourselves if Neural Networks were really needed for this problem, or if we could obtain a good enough result using alternative methods.

Resumen

Este proyecto es continuación de nuestro TFG "Reconocimiento Profundo de secuencias reguladoras del ADN", en el cual desarrollamos una metodología que nos permite investigar diferencias a nivel de secuencia basadas en elementos del genoma humano [1].

En este estudio intentamos desarrollar un mejor entendimiento del problema, explorando nuevas posibilidades y técnicas novedosas, para así mejorar nuestro método y resultados.

Para logarlo, establecemos una nueva metodología para realizar predicciones a nivel genómico. Para ello, hemos estado trabajando en un experimento distinto al del trabajo anterior, con diferentes secuencias, tratando de pulir nuestro procedimiento trabajando en un proyecto más viable.

Nuestro primer objetivo fue saber cómo las Redes Neuronales de Convolución pueden ayudar en esta tarea de distinguir secuencias solo por su composición, dependiendo sólamente de sus carácteres y patrones para diferenciar entre distintos tipos de secuencia. Esta meta fue introducida y trabajada en nuestro anterior trabajo.

Después, decidimos mejorar nuestro entendimiento del problema, estudiando los resultados y analizándolos en profundidad para tener una mejor comprensión del contexto y decidir cuál sería nuestro siguiente paso.

Tras eso, nuestra siguiente meta fue la de explicar los resultados con el algoritmo "Layer-wise Relevance Propagation" (LRP), intentando aprender qué es lo que hace el modelo para clasificar secuencias como positivas o negativas. De esta manera, podríamos obtener nuevo conocimiento sobre el problema, y este conocimiento podría ayudarnos a revisar y mejorar de nuevo nuestra metodología.

La cuarta pregunta que nos hicimos fue la de saber si un análisis de k-mers podría apoyar o complementar los resultados de LRP, ayudándonos a tener una segunda perspectiva. Y, con esto, llegamos a la última cuestión, preguntándonos si de verdad las Redes Neuronales son necesarias para resolver este problema, o si habría métodos alternativos suficientemente buenos.

Contents

1 Introduction

Machine Learning (ML) is the study of computer algorithms that improve automatically through experience [2] and one of the branches of Artificial Intelligence (AI), which is in turn defined as "a system's ability to correctly interpret external data, to learn from such data, and to use what was learnt to achieve specific goals and tasks through exible adaptation" by Andreas Kaplan and Michael Haenlein [3]. Artificially intelligent systems are those capable interpret, analize and work with external data, with the purpose of solving a problem.

In our thesis, this problem we are trying to solve is the classification of DNA sequences through application of ML techniques to know if they belong or not to a certain group. This can have applications in many fields, such as biology, genetics or medicine, for example in the treatment of molecular based diseases, such as intellectual disability.

1.1 Genomic background

To understand what this is about, we need to have a basic understanding of some genetic concepts.

First, we have to mention the **Central Dogma of molecular biology**. This dogma explains the process the DNA goes through to produce protein: **DNA** makes **RNA**, and **RNA** makes **protein**. Those two steps are also known as **transcription** and **translation**. We will be entirely focusing on the **transcription** step.

There are two types of sequences inside the DNA: Coding sequences, and non-coding sequences. Coding sequences' work consist of encoding protein sequences. Non-coding sequences serve other purposes, like the transcriptional and translational regulation of coding sequences. Those are the elements we are working with, specifically the ones that regulate the transcription of coding sequences. Inside that category, we will be focusing on **5'UTR** and **3'UTR**.

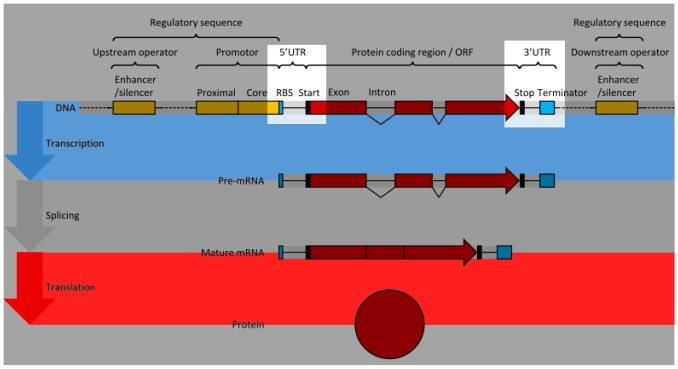

Figure 1: Placement of 5'UTR and 3'UTR. Red sections compose the gene. Orange sections compose the regulatory elements that control the transcription of the gene [1]

DNA is the template that produces mRNA via a process of copy. A particular segment of the DNA – the red segment in *Figure 1* – is copied into this mRNA by the RNA polymerase.

Untranslated Regions (UTR) are two sections that can be found on each side of a coding sequence. UTR sections are neither transcribed nor translated, but intervene in the **translation**

[1] Image extracted from http://www.ensembl.info/2018/08/17/ensembl-insights-how-are-utrs-annotated/

of the final protein. If it is found on the 5' side, it is called **5'UTR**. If it is found on the 3' side, it is called **3'UTR**. These untranslated regions are involved in many regulatory aspects of gene expressions in eukaryotic organisms.

1.2 Problem specifications

We are working from the perspective of a supervised classification problem (since we are labeling a series of examples whose real class is known to us), whose distinctive feature is that we are going to only use the **DNA sequences** for the classification, without considering any other aspect from the genes we study.

Our sequences are composed of a certain number of nucleotides: characters A, C, G or T (adenine, cytosine, guanine and thymine). In our project, the **individual meaning** of each character is completely ignored, being relevant to us only the ways in which they combine to form patterns and sequences. Some examples of sequences might be seen in *Table 1*:

Table 1: Examples of DNA sequences

Example	Sequence
#1	GACTGTAGTGTACGTAGCTTACGTACTTACGTACGTAC...
#2	TTTAGTACGTATTGATTTATTATTATATCGTCTTATAAA...
#3	AAAAATTATATATAAATATAGCCCTATTATTAGATTATTT...

Each one of those sequences belongs to a specific group, such as "3'UTR", "ICE", "Promoter"... Even though each group has its own traits and meaning, for us they are only aggregations of different sequences. Our task is to build an ML model capable of correctly predicting the group to which any sequence belongs.

We work with **binary** classification. That means we classify sequences with two possible outcomes: positive or negative. For example, classifying sequences as "3'UTR" (positive) or "non-3'UTR" (negative). We have worked with multiple different groups for different experiments, and all of these groups will be explained in *section 4 Methodology*.

This model built would later be used to uncover new sequences of the genome. Only a 3% of the genome is annotated. An annotation in the context of genomics is the process of identification of the locations and functionality of the genes. That means we only know what do the 3% of the genes. The rest is still unknown, and, even though probably most of it is rubbish, there are surely lots of unannotated genes that are functional. We want to contribute to rise that 3% with new annotated regions.

For that, we want to train a model that classifies the genes as *"genes that belong to a group"* (or **positive**) or *"genes that do not belong to a group"* (or **negative**). If we build a model capable of distinguishing genes from those two groups, and a gene that is supposedly negative gets stubbornly predicted as positive, we would have reasons to believe that gene might be positive instead.

This model will give us a numeric prediction from 0 to 1: A value higher than 0.5 means the model thinks it is a positive sequence, and a value lower than 0.5 means it thinks it is a negative sequence. The closer the output is to 0 or 1, the more convinced the model is of its prediction.

For example, in *Table 2*, we can see four sequences, each one labeled as either positive or negative, which represents our current knowledge for that sequence, with a numeric value that represents the conviction of the model as we explained before. Positive sequences are already annotated and, thus, we are sure that they are positive, regardless of whatever the model says about them, but that does not happen with negative sequences. In this example, for sequences 1 and 2 the model agrees with the labels with a high value con certainty, which is good for knowing it is working properly. For sequence 3, the model thinks it is a positive sequence, but its certainty is very low, so it could be a mistake. For sequence 4, the model predicts it as a positive sequence with a high value of certainty. This could mean that this is one of the unannotated positive sequences, so we would study the gene of that sequence more carefully.

Table 2: Examples of labeled DNA sequences

Example	Sequence	Label	Prediction	Certainty
#1	GACTGTAGTGTACGTAG....	Positive	Positive	0.93
#2	TTTAGTACGTATTGATT...	Negative	Negative	0.96
#3	AAAAATTATATATAAAA...	Negative	Positive	0.52
#4	GGGCGCGAGAGGTGTGA...	Negative	Positive	0.95

In case this division is possible to achieve with good results, we would want to know **what** makes our ML models discern the sequences and correctly predict their class, which would help us obtain new knowledge and have a better understanding of the problem, maybe giving us tools to improve the result even further.

1.3 Our examples

Each one of the examined genes are our "examples", the subjects of our study. Of all these examples, we only use their genetic sequence, as we stated before. Each one of the characters of the sequence will be **one-hot encoded**, which means they will be translated to a series of ones and zeroes: {A = 10000, C = 01000, G = 00100, T = 00010, X = 00001}.

Each digit of these encoded sequences will be considered a **feature of the example** for the network. The "X" is a special character called "character of padding", which we will use to extend short sequences to be able to work with them, since ML techniques require all inputs to be of same length. A more detailed explanation on padding can be seen in [1].

1.4 Explainability

Once trained our network, the next step is explaining the results, so that we can learn from them. Explainability provides us with the knowledge of what parts of the sequence contribute the most to produce a decision, and helps us having a deeper understanding of our problem. A more in depth description of explainaibility will be provided in *section 3 State of the Art*. Application of eXplainable Machine Learning techniques (XML) is one of the core aspects of our project, and is the **main improvemet** we made regarding our original project.

The XML algorithm we use in our project is **Layer-wise Relevance Propagation**, also known as **LRP**, which obtains a **saliency map**, an array of relevances, one for each input feature, that indicates how important was that input for determining the class of the sequence. This algorithm is comprehensively explained in *section 3.4 Layer-wise Relevance Propagation*.

2 Objectives

Our main objective is to continue developing a methodology that allows us to investigate sequence level differences based on elements of the human genome. We want to know if multiple types of genes can be differentiated by analyzing their sequences.

Sub objectives:

- Develop techniques to find positive sequences hidden among the negatives, analyzing the differences between negative sequences that are easily classified, and those that are usually mistaken as positives.
- Study in depth the effect of some choices we made in the original project, such as the complexity of the network or the effects of the padding.
- Implement the LRP algorithm in R, specifically designed to work with our network, to study the results of the model.
- Explain the results of our models using LRP, and obtain new knowledge from it.
- Establish multiple ways of verifying the correctness of the LRP results.

3 State of the Art

In this project, we will try to explain the results of our Convolutional Neural Network using XML techniques. But first, we need to know what is explainability, and how to apply it to our model.

3.1 Explainability

What is an explanation in this context? We will use the definition from [6]: *"An explanation is the* ***collection of features*** *of the interpretable domain, that* ***have contributed*** *for a given example to* ***produce a decision*** *(e.g., classification or regression)."*

Explainability is, therefore, the ability to produce these explanations of our experiments, the capacity of knowing which features have contributed to the solution. Do not confuse *explanation* with *interpretation*, since the later would be defined as *"the mapping of an abstract concept (e.g. a predicted class) into a domain that the human can make sense of"*, that is, a technique whose goal is to let a human understand a concept [6].

For each one of the features of our individuals we can also indicate the **level of contribution** in form of **saliency scores**, that will tell us which were the most important features. This will help us know what our CNN, essentially a "black box", is doing internally, and will shed some light on which features are useful and which are not.

We will work with genomic sequences, composed of multiple characters. For us, the explanation to obtain will be a **sequence of relevances** that will contain one relevance for each character of the original genomic sequence, so that we can know which areas of the sequences have the most importance.

3.2 Why is it important to explain our results?

Explaining the results is a very vital task if we want to be able to trust our model. This is more important in certain contexts, but if a human does not trust a model, then they will not use it [19].

Explaining the results also has another benefit: it allows us to understand the problem better, and we might even be able to detect flaws in our approach, or ways to improve it. Explaining our results is the way of knowing why the solution was reached, what has been decisive to it, and gives us the chance to upgrade our procedure.

Our reasons to explain our results are the later: we want to understand what made the model perform they way it did, and to know what is that make multiple types of genomic sequences different.

3.3 Techniques to explain the results

How do we explain the results? What techniques can we use for this? Explainability in Machine Learning is a relatively new concept. The first techniques focused more in extracting the knowledge embedded in the neural network as a set of symbolic rules [22]. That the first techniques utilized rule-based approaches is no surprise, since Rule Based Systems are interpretable by default, as we can see in [23]: *"The "unique selling point" of fuzzy systems is usually the interpretability of its rule base".*

However, this same paper [22] hints the possibility of explaining results by identifying the most important features: *"In particular it was noted that some merit exists in identifying what were termed as* ***causal factors*** *or alternatively as left-reduced functional dependencies (i.e. the attributes and in particular the combination of attributes which are* ***most significant in determining the decision attribute****)".*

Some new techniques have been developed since then. In the early 2010s there were some of them like "de-convolution" [9], technique introduced in [24] that is described as *"A ConvNet model that uses the same components (filtering, pooling) but in reverse, so instead of mapping pixels to features does the opposite".* It needs to use certain techinques to reverse the effects of some components, like "unpooling" *(Figure 2).* **Reversing** is a core concept of XML, as we will see it in multiple techniques.

Since we are working with Convolutional Neural Networks, most of the algorithms we will see are designed to work with images, but that does not mean that they are not useful to work with sequences, since convolution1d layers (the layers that work with sequences) work the same way as convolution2d layers.

Other early techniques can be seen in [16], were two techniques are proposed, but they are in essence generalisations of the de-convolution previously mentioned: *"...we demonstrated that gradient-based visualisation techniques generalise the DeconvNet reconstruction procedure".*

More recent methods are described in [8]. In this paper four **gradient-based** methods are described and compared. They are **attribution methods**, which means they assign an **attribution value** to each input feature, values we will call **"relevances"** from now on.

These four methods are compared in *Figure 3*, in the third column called "Example of attributions on MNIST". In that image, we can see the output for each of those methods when

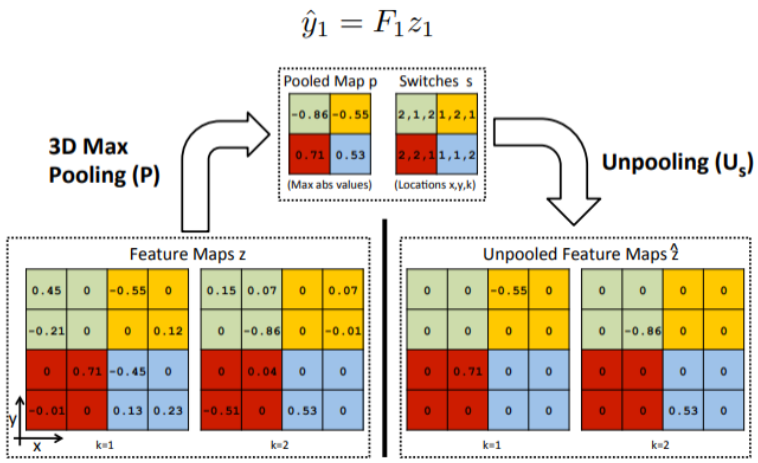

Figure 2: Example of the Unpooling technique.[2]

the input is one of the zeroes of the MNIST dataset, depending on the activation function used. We can see how, if the activation is a ReLU or a Tanh, **every one of these methods behaves the same**, obtaining the same result. This is important, since it says that, when working with ReLUs, it does not really matter which method you pick. From these four methods, we ended up considering **Layer-wise Relevance Propagation** (LRP) for our work. There are more

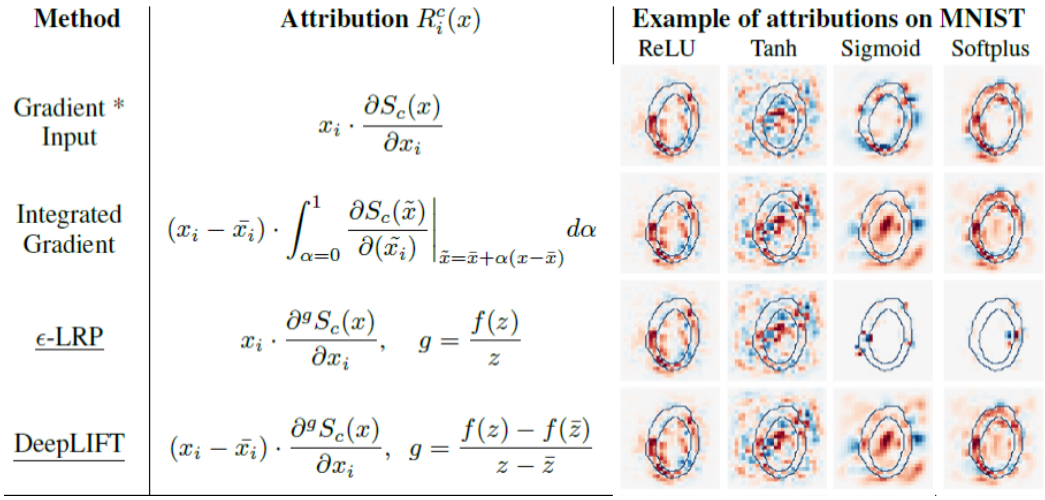

Figure 3: Comparing four gradient based methods.[3]

examples of different XML techniques, such as the one in [18], named Excitation Backdrop, which is capable of explaining with precise **relevance maps** all the important elements of the images *(Figure 4)*.

From all of these, we will choose LRP for our project, for reasons explained later in detail.

3.4 Layer-wise Relevance Propagation

This is the technique we chose for our project to obtain the relevances of our sequences. Layer-wise Relevance Propagation (LRP for short) is an **attribution method**, which, as said before, has the purpose of obtaining a list of **relevances**, one for each input feature.

[2]Image extracted from [9]
[3]Image extracted from [8]

Figure 4: Localizing the words.[4]

Attribution methods may be confused with **feature selection techniques**, since both of them identify the most relevant features. The difference lies in that LRP is applied individually for each input sequence, and returns a result just for it without the need of comparing it with the rest of the sequences, and feature selection needs the whole sequence set to work, and returns a single output for the network.

To obtain this list of relevances, LRP needs the **output** of the network for the given sequence. This output has to be **reintroduced** in the network at the output layer, and will go though the network until reaching the input layer *(Figure 5)*.

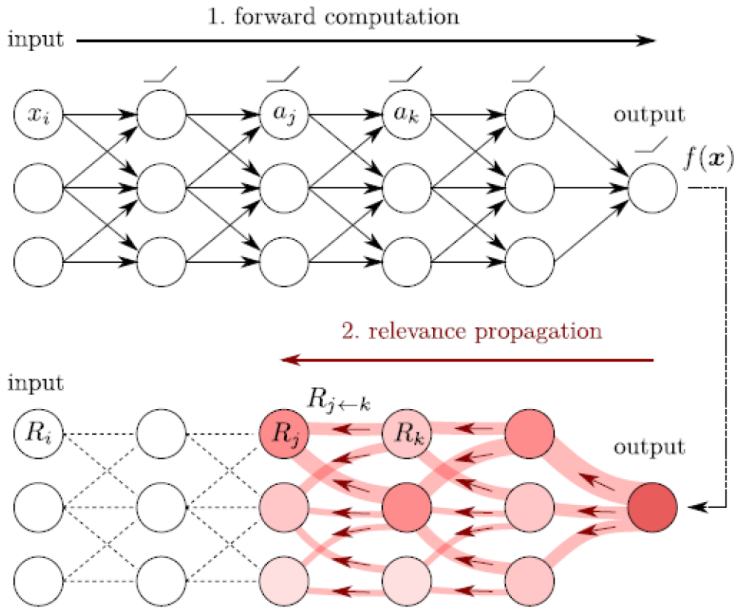

Figure 5: Top figure is the evaluation of the sequence, bottom figure is the reinsertion of the output in the network, from end to beginning, obtaining a list of relevances.[5]

When evaluating the sequence, we introduced a list of n input features at the input layer, and got one single output from the output layer. When explaining with LRP, we introduce one single input at the output layer, and get a list of n outputs from the input layer. These outputs will be the **relevances** of our original n input features.

The previous output of the evaluation (what is now the input of LRP) will be the **relevance of the output node.** Each node will have its own relevance, with no excepctions. The

[4]Image extracted from [18]
[5]Image extracted from [6]

relevances of the nodes of the layer L_{i+1} will be used to calculate the relevances of the nodes of layer L_i.

Rule 1: *The sum of the relevances of all the nodes in a layer has to be the same for every layer in the network.*

This means that if our output node has a relevance of 0.95, every layer of the network has to have a combined relevance of 0.95 if we add the relevances of all its nodes.

3.4.1 Propagation rules

The relevances are propagated backwards using propagation rules. A very detailed explanation of these rules can be found in [6]. First of all, let us describe a neuron as:

$$\alpha_k = \sigma(\sum_j \alpha_j w_{jk} + b_k)$$

Being α_k the activation of the neuron k, α_j the activations of the neurons of the previous layer, w_{jk} the weight between the neurons k and j, b_k the bias of the neuron k, and σ a positive and monotonically increasing activation function.

The first propagation rule that we will see was proposed by [17], and looks like this:

$$R_j = \sum_k (\alpha \frac{\alpha_j w_{jk}^+}{\sum_j \alpha_j w_{jk}^+} - \beta \frac{\alpha_j w_{jk}^-}{\sum_j \alpha_j w_{jk}^+}) R_k$$

The + and – superindexes that are over the weights refers to the positive and negative parts. That means that the first half of the subtraction will use the positive weights, and the second half the negative weights. R_j is the relevance of one node at layer L_i, and R_k is the relevance of one node at layer L_{i+1}.

This means that every node k that the node j has a connection with will contribute to R_j proportionally to their own relevance and the weight between them. This will be denoted as $R_{j \leftarrow k}$ *(Figure 6).*

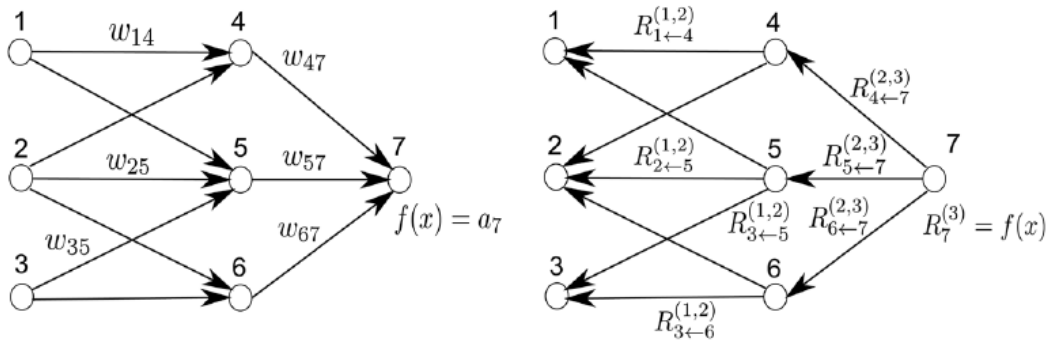

Figure 6: Left: How does the evaluation of a network work. Right: How does the LRP of the same network work.[6]

[6]Image extracted from [17]

Rule 2: *The parameters α and β of the propagation rule have to abide by these constraints:* $\alpha - \beta = 1$ *and* $\beta \geq 0$

Different combinations of α and β provide different results. Not only that, this is not the only propagation rule that exists. There are multiple propagation rules, and not every rule work well with every problem: these are not silver bullets [13]. [13] also provides us with a table that shows the properties of three different variations of LRP *(Figure 7)*.

	naive ($\epsilon = 0$)	ϵ-variant	β-variant
numerically stable	no	yes	yes
consistent with linear mapping	yes	yes	no
conserves relevance	yes	no	yes

Figure 7: This table shows that different rules entail different properties.[7]

The propagation rule we use is the one shown before, with $\alpha = 1$ and $\beta = 0$. This specific assignation lets us interpret LRP as a **deep Taylor decomposition** [26], as explained in [6]. The propagation rule would look like this:

$$R_j = \sum_k \frac{\alpha_j w_{jk}^+}{\sum_j \alpha_j w_{jk}^+} R_k$$

This means we only use positive weights to calculate the relevances of the neurons. This is the propagation rule implemented in this project. According to [6], a first set of propagation rules to be tried are the ones derived from deep Taylor decomposition, such as the one chosen: *"As a default choice, use the deep Taylor LRP rules".*

As per [6], should the $\alpha = 1$ and $\beta = 0$ version fail to provide acceptable results, we should try with $\alpha = 2$ and $\beta = 1$, and thus introducing the negative weights to the function. In compliance with [15]: *"This set of parameters works well for a wide range of neural network models, and allows to express contradicting evidence in the input image, through negative relevance scores".*

One thing we have to keep in mind is, according to [15]: *"Although the alpha-beta LRP rule was shown to work well for DNNs with ReLU activations, there is not one single LRP rule which serves all machine learning models and datasets equally well, as different models have specific layer-to-layer nonlinear mappings and input domains, that need to be considered".* This, together with [13] mentioned before, means that trying only one version of one propagation rule does not secure us an acceptable result.

This propagation rule works for all **dense layers**, but **convolutional layers** have extra complexity. The propagation rule is the same, but, unlike dense layers, not every node of Layer L_i is connected to every node of layer L_{i+1}. We have to consider **how filters work**, and which nodes are "connected" to which. Once that is solved, the process is the same as before.

As for special layers, such as dropout or max pooling, an extra effort is needed to make them work for LRP. For that reason, we got rid of those layers from our network (refer to *section 4.5 Improving the Convolution*).

[7] Table extracted from [13]

3.5 Real cases of LRP usage in similar projects

In this section we will ilustrate some real cases that worked with LRP to explain their models. Some of them are very similar to our work, while others work in very different domains.

3.5.1 Predicting mRNA Abundance Directly from Genomic Sequence

This first case, explained in [7], shows us a very similar project to ours. Even though its actual goal differs from ours (predicting gene expression levels), its methodology is mostly the same: Applying **Deep Convolutional Networks** to **genome sequences**, and then **explaining** the results using multiple techniques, being LRP one of them.

Architecture

The architecture of the network is the following *(Figure 8)*:

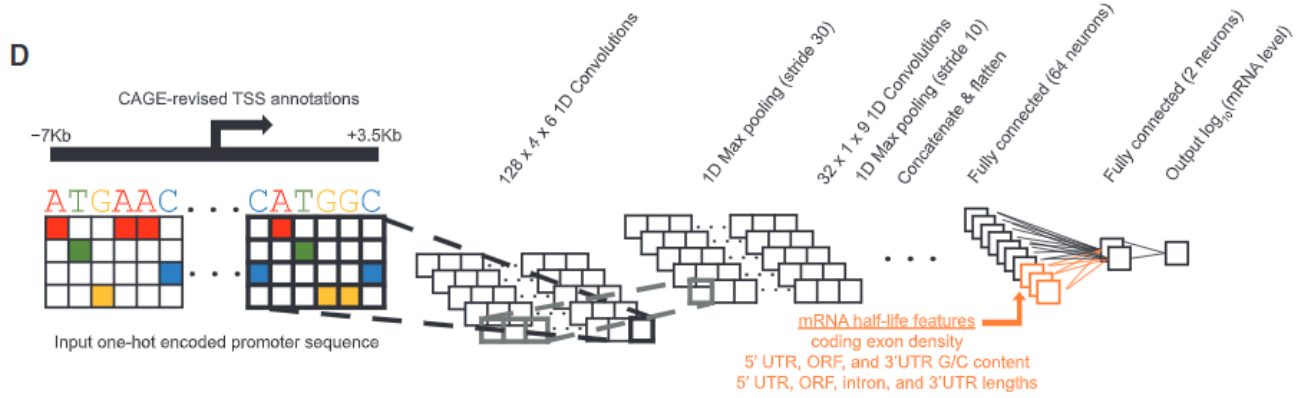

Figure 8: Architecture of the experiment.[8]

This network is composed of:

- **One-hot encoded input,** the same as we use in our experiment.
- **128 x 4 x 6 convolution layer.** Each sequence contains 128 blocks of 6 characters, and each character is composed of 4 digits because of the one-hot encoding.
- **Max Pooling layer,** which samples down the features
- **32 x 1 x 9 convolution layer**
- **Max pooling layer**
- **Concatenate and flatten,** which reshapes the input so that fully connected layers can work with it
- **Fully connected layer** of 64 nodes, which learns from the data.
- **Fully conected layer** of 2 nodes, which produces the output
- **Output log,** which calculates the log in base 10 of the output to obtain the **result**.

This network is a more complex version of ours [1]. It is designed to learn from the patterns inside the genome sequence.

Explanations

[8]Image extracted from [7]

This paper explained the predictions of the network with four different methods: *"(1) gradient * input, (2) integrated gradients, (3) DeepLIFT (with rescale rules), and (4) ϵ-LRP"*. Each studied gene was inserted into one of four different groups, depending on their level of expression: None, Low, Mid or High expression.

For each group, the mean saliency scores (relevances) was calculated, and then computed the difference in mean scores relative to predicted non-expressed genes for each nucleotide position in the entire input window.

The graph shown in *Figure 9* shows the mean relevance of each relevance level. The x-axis indicates the position within the sequence, relative to the TSS (Transcript Start Site), and the y-axis show the value of relevance: The higher the value, the higher the relevance. According to these results, the area with most relevance corresponds to the central region, around the position 0 relative to TSS.

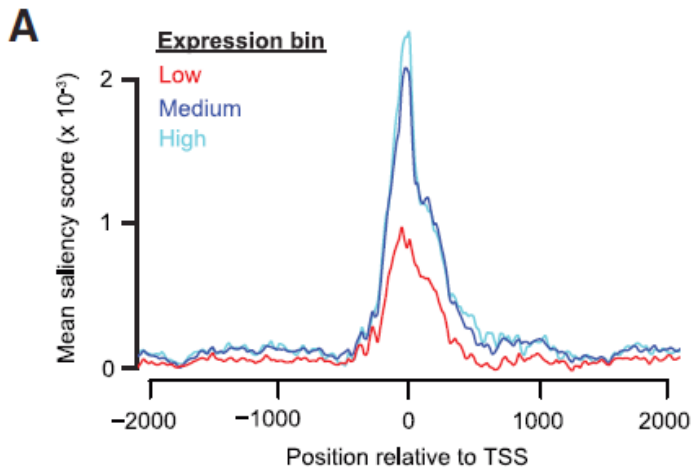

Figure 9: Relevances of the different levels of expression.[9]

This is very similar to what we did in our experiments. Refer to the section 4.8 for more details.

3.5.2 LRP for Explaining NN Decisions in Alzheimer's Disease Classification

Deep Neural Networks have achieved more than acceptable results in many medical tasks, including the one relevant to this paper: Alzheimer's disease detection [20]. However, with them being "black boxes", and having almost zero transparency, it is very difficult to apply their results in clinical routine.

As claimed by this study, *"the LRP method is able to directly highlight positive contributions to the network classification in the input space"*.

Architecture

Again, CNNs are used in combination to LRP to explain its results *(Figure 10)*.

[9]Image extracted from [7]
[10]Image extracted from [20]

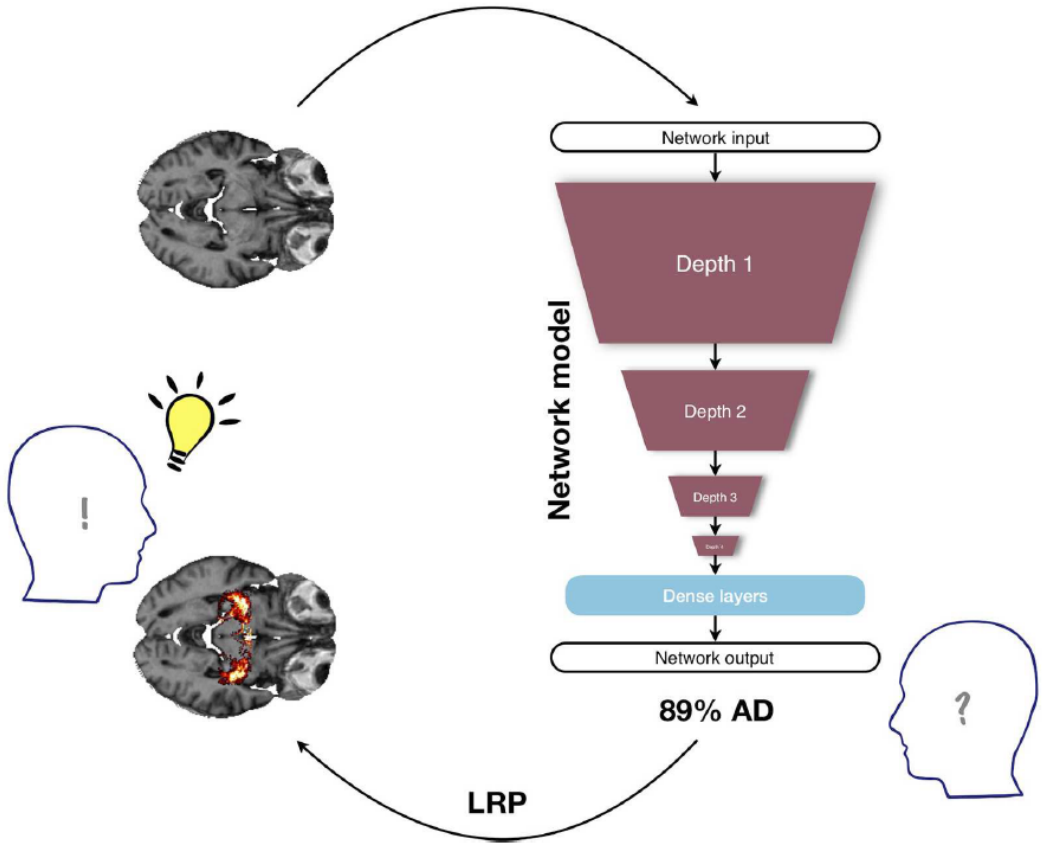

Figure 10: Simple schema of the architecture used.[10]

This time we are not provided with all the details of the architecture, such as the size of the layers or its parameters, but we can see the number of layers used, and the result obtained.

Explanations

Since the input in this experiment is an **image**, unlike the previous paper in which the input was a sequence, this time the explanation will be in form of a **heatmap**, which is a colored image that represents the relevances using a color scale *(Figure 11)*.

These heatmaps show the most important areas for the predictions. They also show the differences between different values of beta, one of the arguments of the LRP propagation rules.

3.5.3 Deep Motif Dashboard

Deep Motif Dashboard is a toolkit developed in [27] that introduces multiple ways of visualizing and explaining the results for three types of models (convolutional, recurrent, and convolutional-recurrent) that work with **DNA sequences**.

One of these ways of visualizing uses **saliency maps**, like the ones we could get with LRP, though this study uses an older technique for that, similar to [16]. This method visualizes the relevances of the three models andf then tries to find an area in which their relevances are higher than normal *(Figure 12)*.

[11] Image extracted from [20]
[12] Image extracted from [27]

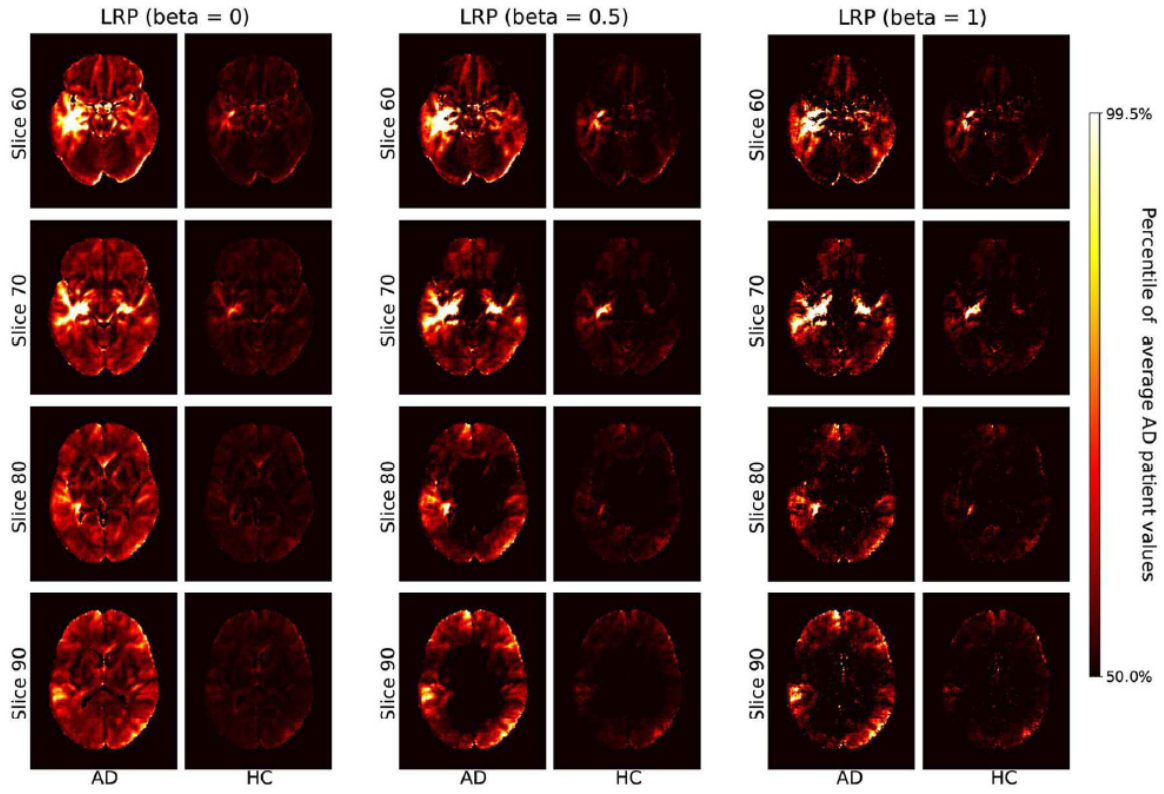

Figure 11: Heatmaps obtained with LRP.[11]

This paper covers more techniques than this, like **Temporal Output Scores** or **Class Optimizations**. However, these do not work with saliency maps, so they are not really relevant to our study.

There are many more projects working with LRP to explain their results, for example [14], which we are not going to review in depth since its domain is very different from ours (diagnosis of Gear Box faults), but we are mentioning it anyway since it shows that LRP is **not limited to an specific domain.**

3.6 Limitations of LRP

Even though LRP has proven to be a very viable tool for explaining CNNs, it has certain limitations one has to be aware of. Some of these limitations are specific to LRP, while others are general to all XML algorithms.

As mentioned in [20], *"it is largely acknowledged that heatmaps are quite sensitive to the specific algorithms (and its parameters, e.g., the value in case of LRP) used to produce them".* This is more of a thing to keep in mind than a limitation per se, but what we try to show with this is that trying only one algorithm with one assignation of values is probably not going to obtain the optimal result. Many tries with different algorithms should be made in order to obtain the most accurate explanation. This other paper [13] supports this claim.

As the same paper explains, *"heatmaps just highlight voxels that contributed to a certain classifier decision, but do not allow making a statement about the underlying reasons (e.g., atrophy or shape differences) or potential interactions between voxels or brain areas".* This means that, even though we can know which were the most relevant areas of a sequence or

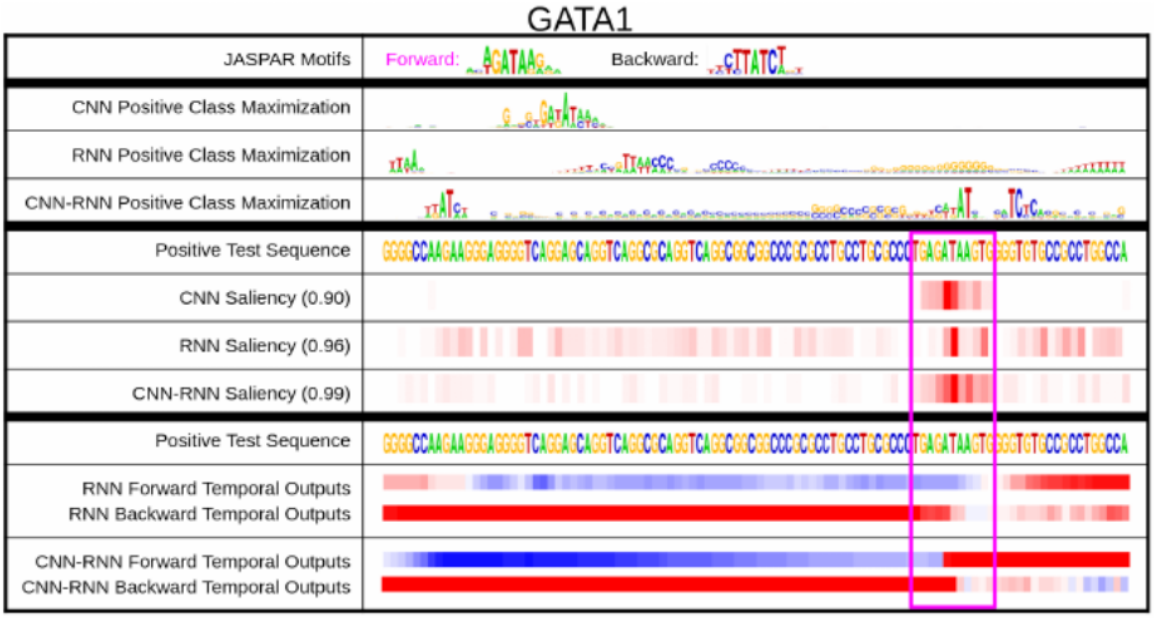

Figure 12: Heatmaps of the dashboard. The boxed area represents the area of higher relevance.[12]

image, we still do not know why, and a posterior analysis would be needed to obtain the complete explanation. Interactions between areas are also not shown, and in certain cases this could be vital to explain why something happens.

That is also stated in [12]: *"Saliency maps can be useful to determine what part of the image is being omitted by the classifier, but this leaves out all information about how relevant information is being used. Knowing where the network is looking within the image does not tell the user what it is doing with that part of the image".*

The third one also comes from [20]: *"Heatmaps strongly depend on the type and quality of the classifier, whose decisions are sought to be explained. Therefore, each heatmap should be read as an indication of where the specific network model sees evidence. For badly trained networks, this does not have to correlate at all with the presence of actual biomarkers."* This is very reasonable: if the classifier is not getting good results, then they should not be expected from the explanation.

That situation changes in certain scenarios: *"However, heatmaps are also useful in cases, where classification performance is low or sample size is rather small, e.g., for better understanding if the classifier picks up relevant or irrelevant features".* This means that the network not obtaining a good result is not a certainty that the explanation will not be useful, because it can help us understand why it is not performing well.

As stated by [12], *"Explanations are often not reliable, and can be misleading".* This article affirms that explaining ML models is a technique that should be avoided for high stake decisions, as we can never be sure that the explanation is 100% correct. This, though, does not really affect us, since we are not working with high stake decisions, as we are only looking to understand the solution, not trying to make a model decide something important for us.

Finally, as per [10], gradient algorithms have some limitations. First, a neuron can be signaling meaningful information even when its gradient is zero. Second, since gradients have a discontinuous nature, small changes in the input can cause sudden jumps in the salience score. These limitations are addressed by DeepLIFT, which we will talk more about later.

3.7 Why LRP

LRP was chosen for our project because of a combination of factors: First, it was relatively easy to find information about it. This includes sources that helped with the implementation, such as [6].

Second, it is a very **customizable** algorithm. This means that, if we are not satisfied with a certain explanation, we can easily modify the parameters, or even pick a different version of LRP, without having to implement a whole different method. Even though we ended up using only one version of LRP, this was one of the reasons for choosing it.

Third, it does not require additional arguments, unlike DeepLIFT, one of its recent alternatives [10]. DeepLIFT adresses some of the limitations of the gradients (last paragraph of previous section), but it has some drawbacks: DeepLIFT needs a **reference input.**

A reference input is an example for the real input to be compared to. It has to be "neutral", not having recognizable patterns with meaning. For example, a reference for the MNIST dataset could be an image composed solely of zeroes, as that is the background of the MNIST images. For DNA sequence inputs it can be a sequence that contains the expected frequencies of the characters A, C, G, T found in the "background" of the original input.

The choice of a reference input is **critical** for obtaining insightful results. In practice, choosing a good reference relies on domain-specific knowledge, and often requires computing DeepLIFT scores agains multiple different references. This is what makes the usage of DeepLIFT more complex than LRP, and the main reason why we did not choose it, even though it solves some issues from LRP.

Some rule-based approaches obtain very good results, like the one shown at [11], and are already interpretable by default. The problem is that they do not use Deep Neural Networks, which are what we are working with since the start, so these are not really suited for our work.

Last, there is the option of using older algorithms, for example deconvolution, but some papers claim that LRP tends to obtain better results than deconvolution, like [25].

In conclusion, LRP seems like a very good entry point for an explanation algorithm, as it is relatively easy to learn and use, and provides enough customization.

4 Methodology

In this section we will follow the whole process from start to finish, explaining what we did and why, reviewing each one of the experiments and explaining our thought process in detail.

We worked with two main problems: "5'UTR" and "3'UTR". Our first two sections describe these problems in depth, then we review all experiments of the first problem, and then those of the second. Each subsection will include the problem they refer to in their name, for clarity.

The results for every experiment described here will be found in its correspondant subsection inside *section 5 Results*.

4.1 Introduction to the 5'UTR problem

The first two groups we are going to classify our sequences into are "disease associated" and "non-disease associated". We worked with intellectual disability genes as our disease associated genes. As for the non disease associated, they are the rest of the genes that encode protein. We will clasiffy them using their 5'UTR sequences.

5'UTR sequences are those that belong to an area of the genome called *UnTranslated Region*. This is not exactly *a single* region, but multiple regions. 5' is one of those. This experiment was named "The UTR experiment" in the original project [1].

Thus, we have a series of labeled sequences, classified as either "disease associated' or "non-disease associated", and our first goal was to build a Convolutional Neural Network for their classification, to know how these structures might help in this task of labeling.

In this original dataset there are 3.657 positive sequences, and 17.391 negative sequences. Each sequence came from a different gene, and they could be either positive (associated with the intellectual disability) or negative (not associated).

Also note that we do not work with a single model, but with **five of them**. This is because we apply a custom version of cross-validation, in which we train five different models with five different subsets of sequences, and make the mean of all of them the final result, to address the unbalance in our positive and negative sequences, since we have way more negative sequences than positives.

This custom cross-validation works like this: We extract five subsets of sequences from the negative set, each of the same size of the positive subset, and each of the five models use the whole positive set plus one of the negative subsets, dividing them into training and evaluation. Even though each of the negative subsets is used only once, it lets us train and evaluate with more negative sequences than traditional cross-validation. We tried this method because we wanted to take advantage of the high number of negative sequences at our disposal, as an experimental alternative to cross-validation. Later we would change this procedure for a traditional cross-validation, but that will be explained in future sections. With this procedure, every model was trained with a perfect balance between positive and negative.

For the classification, we trained these five Convolutional Neural Netwowks, whose architectures will be reviewed in future sections, and then evaluated all genes using these trained models. That will be the results of our prediction.

For this problem, we applied a reduction of the parameters of our network, since we suspected we could obtain a similar result with a smaller network.

Then, we tried to improve our knowledge of the problem, trying to learn something that would help us obtain a better result. For that, we detailed the results of our evaluation, dividing them into positive and negative results for every one of our five models. Then, we analyzed or hardest and easiest sequences to learn, and then looked at the minimum length our sequences should have in order to discard sequences so short they would contain no relevant information.

4.2 Introduction to the 3'UTR problem

This time we are working with 3'UTR sequences instead of 5'UTRs. Although they seem similar because of their name, in truth they are very different. While the 5'UTRs are found

at the start of the mRNA, right after the Cap, the 3'UTR is located at the end of it, before the Poly-A tail *(Figure 13)*. They have different functions and different sequences. They only thing in which they resemble one another is the name.

These two

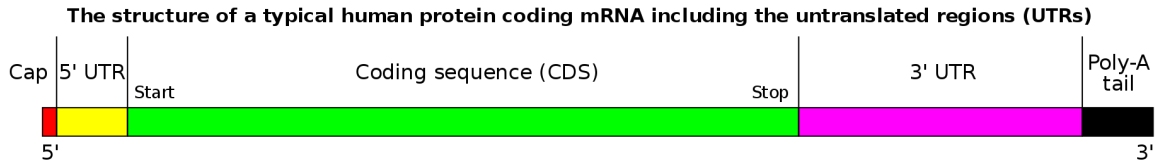

Figure 13: Location of the 5'UTR and 3'UTR in the genome.[13]

So, while the first experiment tries to differentiate association with a disease of a gene given its 5'UTR sequences, this second experiment tries to differentiate 3'UTR sequences from those that are not 3'UTR *(Figure 14)*. The type of sequences and the amount of them we possess is detailed in *Table 3*.

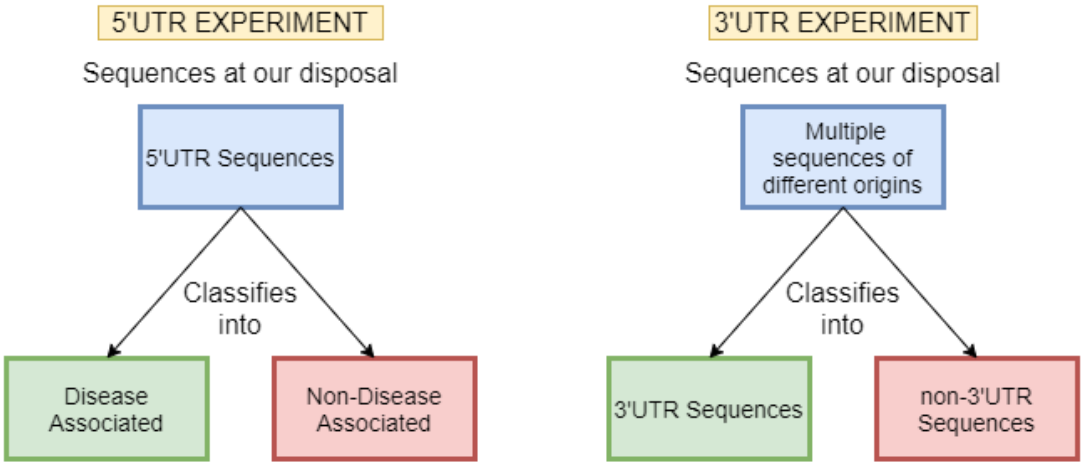

Figure 14: Visual guide for the two main problems we work with in this project.

These sequences have been extracted from multiple genes with only one requirement, and that is that these sequences had to have a **high confidence** at the transcript level.

Table 3: Amount of sequences of each type.

5'UTR	ICE	lncRNA	ncRNA	pseudoGene	3'UTR
21798	130768	3718	3819	2146	17719

These are in total 179.978 sequences, which we divided into two groups: 3'UTR (positive sequences) and non-3'UTR (negative sequences). That makes 17.719 positive sequences, and 162.259 negative sequences.

[13]Image extracted from `https://en.wikipedia.org/wiki/Polyadenylation`

What the subtypes of non-3'UTR sequences are is not important, since they were chosen only because of them not being 3'UTR. Them being "ICE" or "5'UTR" is not relevant to our study, only the fact that they are not 3'UTR sequences.

These positive and negative sequences are used to train a Convolutional Neural Network in the same way the first problem does. We used the same network and same preprocessing, we only changed the dataset. This means these two problems are different in concept, but very similar in execution.

For this problem, first we will review its results, then we analyze the perfomance for every different type of negative sequence, and last we will again simplify the network, since we realize it can be further simplified while still getting the same result.

4.3 Why change?

Why change our dataset in the middle of the project? We considered the first problem was mostly solved, and all that was left were small paths that we tried and did not found anything useful.

Our main goal is not to solve our original problem, but to **develop a methodology** that could help us solve many problems of similar specifications. We already got our result for a problem that tried to classify genes via their 5'UTR sequences. We wanted to see how this same network would behave with a different problem and with different sequences. In this case, predicting the belonging to the 3'UTR group for a set of unseen 3'UTR and non-3'UTR sequences.

4.4 Obtaining the data

In this thesis we worked with two different datasets: One of them was the original dataset we worked with in our first thesis [1], which is composed of 5'UTR sequences, some of them being associated with intellectual disability genes. The other one studies the 3'UTR region: It contains sequences from multiple genes that are either 3'UTR, or non-3'UTR, and tries to classify them. Both of them were extracted from the online database "Ensembl Genome Browser", which contained all sequences we needed, among many other things.

A detailed guide on how to extract the data for these experiments can be found at [1] for the first dataset, and in *Appendix 9.2.4 Recovery of sequences of new dataset* for the second.

4.5 5'UTR - Improving the Convolution

As we said before, our first objective was to know how Convolutional Neural Networks can help in this task of distinguishing sequences only by their character composition. This has already been worked on in our previous project [1], where we learned that we could indeed use them to solve this problem, with relatively acceptable results

This time we tried some improvements to the already solved problem, like the simplification of the network to do the same job with a simpler architecture. The main reason we looked for a simplification of the network is because we suspected we could get a very similar result

with way less parameters, which gets translated as lower time consumption when training the network. For that, we just had to reduce the number of parameters by removing layers from the network, reducing filters of the convolutional layer or rising its kernel size. The details about the previous and following architectures, and the results of this simplification can be found at *section 5.1.*

4.6 5'UTR - Improving our understanding of the problem

Once we arrived at an upgraded version of our network, we set the goal of obtaining more knowledge about our problem, either by detailing our results, or by studying the sequences more in depth. Our goal was to learn something that could lead to a better result. We tried many things; some worked, and some did not. We explored paths we left unexplored in the original work.

4.6.1 Detailed evaluations results

The first improvement to our original project was providing a detailed breakdown of the results for our experiment. Instead of just showing the result as "validation accuracy" and nothing more, we were interested in knowing the behaviour of the models for positive and negative sequences depending of whether they were used or not for the training of the models.

These are the values we analyzed:

- Rate of Trained Right, Rate of success of the prediction for a sequence used for the training of the networks
- Rate of Untrained Right, Rate of success of the prediction for a sequence used for the evaluation of the networks
- Disease Association, whether the sequence is positive or negative

We will obtain these values for every type of sequence (positive and negative), for every model, to compare how do positive and negative sequences perform.

4.6.2 Hard and easy sequences

We have two kind of disease associated sequences: The "easy" sequences, which tend to be correctly predicted, and the "hard" sequences: those that even though help train most of the models, are incorrectly predicted. We asked ourselves: What makes the hard and easy sequences different? And we studied this considering only the sequences.

Obtaining the hard sequences

To obtain these sequences, first we tried selecting all disease associated sequences that were not correctly predicted even once, but that selection returned an empty list of sequences. This means no sequence is so hard as to not be correctly predicted even once.

We looked at the distribution of usage in training for positive sequences: those used once, twice... up to five times in the training of the five models. This distribution, together with the proportion of those sequences that are incorrectly classified, can be seen in *Figure 15.*

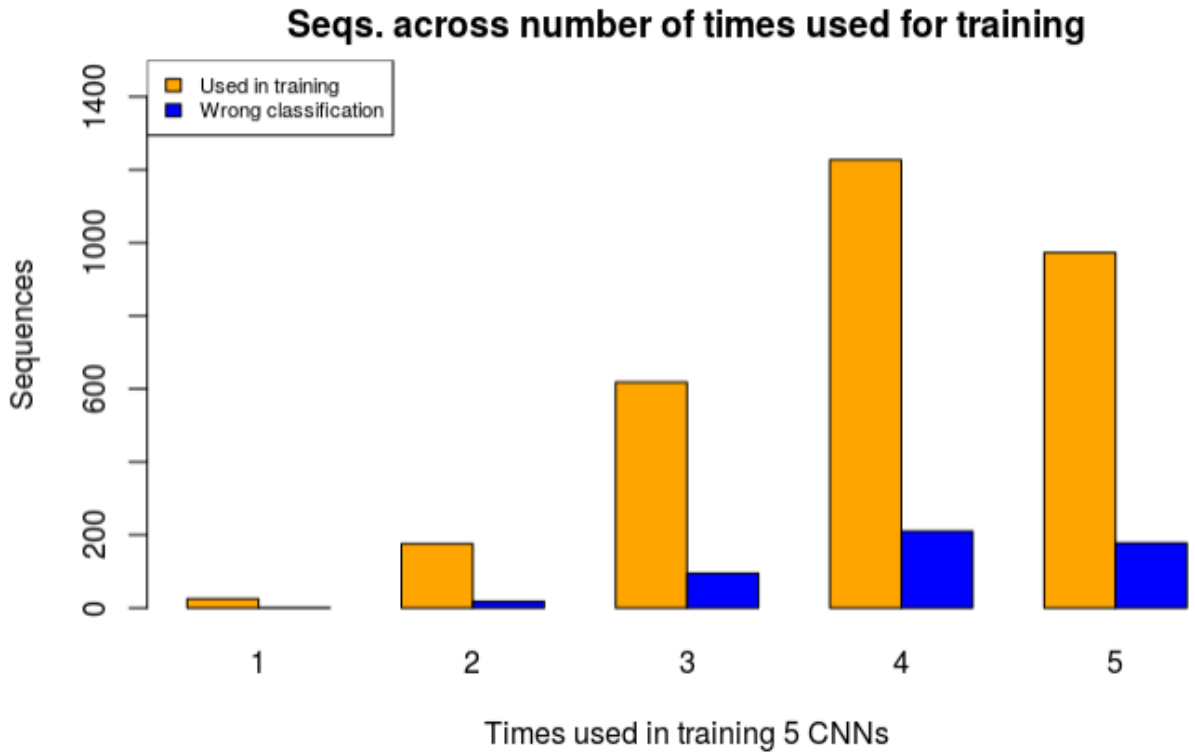

Figure 15: x-axis: Number of times the sequences were used for the training of the five networks. Orange: Number of sequences used that many times for the trainings, Blue: Number of those sequences that are wrongly classified.

We saw that most sequences were used four or five times for the training of the models, so we focused on those sequences. We plotted the number of times those sequences were incorrectly classified in *Figure 16*.

Those were the sequences we selected as hard: Sequences used for training four or more times and that even so sometimes they were incorrectly classified.

Obtaining the easy sequences

The extraction of easy sequences was way easier than the hard sequences: We just selected those positive sequences that were correctly predicted by all five models.

PCA Study

For this, we did a PCA study of the frequencies of the alphabet in both hard and easy sequences. The code for this, together with the code to find the easy and hard sequences, can be found in the Appendix 2.

PCA, or Principal Components Analysys, represents in a plot relatedness between populations. It shows in a graph points that corresponds to a single individual each. Points that are close have similar characteristics, while those that are far are different.

That means that, by looking at a PCA plot, we can try to differentiate two different groups (in this case, easy and hard) of genes. If the points corresponding to hard genes are close together, and far from the points that correspond to the easy genes, that would mean that hard and easy genes present different characteristics. In this case, since we are studying frequencies of their alphabets, it would mean that the frequencies of the alphabet that compose the hard

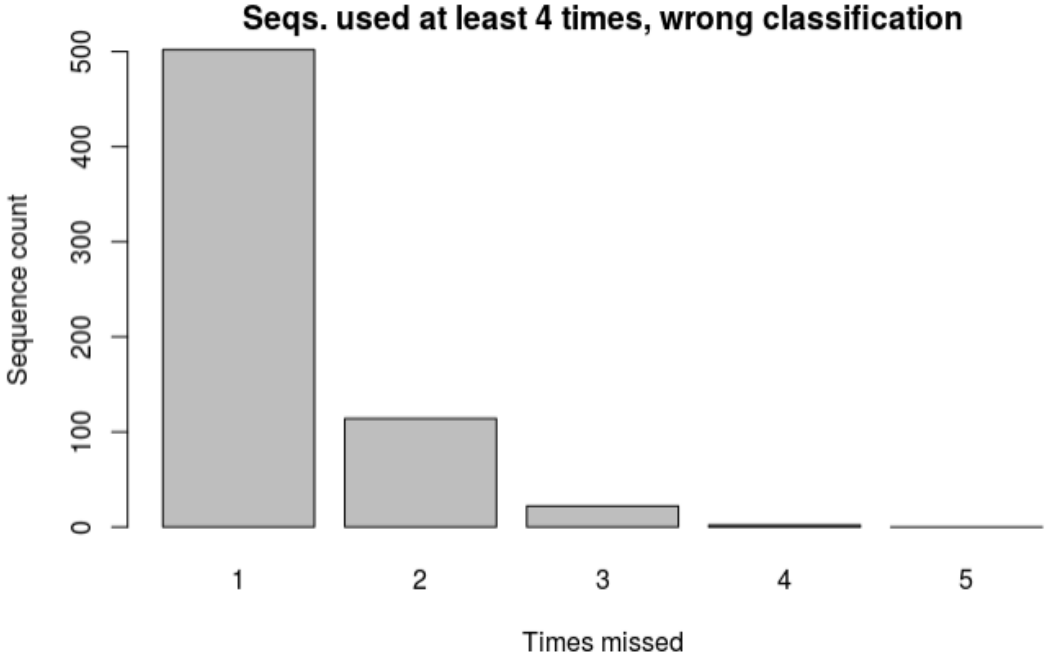

Figure 16: Number of times the sequences used for the training of four or five sequences were incorrectly classified.

sequences are different than those of the easy sequences. We chose this feature because it is the most relevant feature we could find for determining if two sequences were similar or not.

The results of the PCA study and its conclusions are found in *section 5.2.2 Results for hard and easy sequences*.

4.6.3 Study of the Sequence Treatment

In the original project, we did the following treatment to our strings: First we mirrored our sequences, then, depending of the length of the sequence, we padded them with the padding character X if they where shorter than the desired length, or cut them to the desired length if they were longer; and finally, we one-hot encoded them (*Table 4*, suppose the desired length for our strings is four characters).

Table 4: Treatment of our sequences.

Original		Reversed		Padded/Cut		One-hot encoded
AACCG	→	GCCAA	→	GCCA	→	00100010000100010000
GGC	→	CGG	→	CGGX	→	01000001000010000001
T	→	T	→	TXXX	→	00010000010000100001
TC	→	CT	→	CTXX	→	01000000100000100001

We wanted to test the optimal position of the padding (start or end of the sequence) and the need of reversing the sequence. For that reason, we performed four experiments. If the original sequence were "ABC", and the desired length were 5 characters, the four sequences of these experiments would be:

– Original sequence, padding at the end → ABCXX

– Reversed sequence, padding at the end → CBAXX
– Original sequence, padding at the start → XXABC
– Reversed sequence, padding at the start → XXCBA

Once obtained these four sets of sequences, we would train the five models once with each different set of sequences, and then would compare their result to see which one was the best way to treat our sequences.

4.6.4 Optimal length of the Sequences

In the original project [1], we set the length of the sequence in 250 characters, since that was the mean of the length of all 5'UTR sequences. This number was chosen just because of that, and could be changed in the future should a good reason for it appear (we will see it in future sections). We introduced padding to every sequence shorter than that, and cut the end of the reversed sequence of those longer than 250 characters. This way, every sequence had the same length.

In this small analysis, we chose to take a closer look at the minimum length of the sequences, and decided whether we should make any changes to it. The code for this can be found at the Appendix 2.

4.7 3'UTR - A change of dataset

At this point we though the first problem was finished, and decided to change to the second one, explained in *sectiom 4.2 Introduction to the 3'UTR problem*, so all experiments from now on will be using this second dataset.

From now on, every test and experiment will be performed using this new dataset.

4.7.1 3'UTR - Comparison of the two experiments

As we said, the experiment remained essentially the same, with the only difference being that we used other sequences. Even so, the results were much better than before.

We did the same training of the network and evaluation of the sequences as we did with our original 5'UTR problem, and did the same breakdown of the results as we did with the other dataset.

In *section 5.3.1*, we will compare the results of both 3'UTR and 5'UTR experiments, and see which of the two performs the best with the same network.

4.7.2 3'UTR - Addressing the unbalance

We followed a similar procedure for dealing with the unbalance to the older project with this new dataset. With our custom cross-validation, every model was trained with a perfect balance between positive and negative sequences. But negative sequences have an internal unbalance that has not been taken into account.

As we said before, negative sequences (non-3'UTR sequences) are divided into a series of subtypes: 5'UTR, ICE, lncRNA, ncRNA and pseudoGene. But these subtypes are not balanced, being ICE the holder of the absolute majority of the negative sequences. So our next step was to analyze the effects of this unbalance, to see if the network was behaving similarly with every subtype of negative sequence. The code for this can be found, as usual, in the Appendix 2.

For each subtype of negative sequence, we would be reviewing its rate of success and the percentage of the training set it constitutes. This way we can see if a type is under or over-represented in the training set, and its results, to put it in context in comparison with the rest of them.

4.7.3 3'UTR - Reducing the size of the network

Since LRP is a very resource-heavy algorithm, the smaller the network, the more sequences we could calculate their relevances with our resources, so we decided to reduce our network a second time, keeping special convolution layers (dropout, max pooling) out of the network, so that the LRP implementation becomes manageable.

For that, we made 10 experiments, trying to reduce filters of the convolutional layer, kernel size for the filters, or nodes from the dense layer (refer to 4.5 to see the current state of the CNN), and taking notes of the number of parameters and performance (accuracy and loss).

After that point we had no more improvements for our experiment, so the next step was to **explain our results** with LRP.

4.8 Explaining the 3'UTR Results with LRP

We had two possible ways of continuing with this project. The first possibility was to further investigate some questions to which we still had no answer, for example why do certain subtypes of negative sequences perform worse, or going back to the old experiment and try to see what is the difference between the hard and the easy sequences (Refer to their correspondent section of results for more information about these questions).

The second option was to find a way to explain the results of the network, and knowing what does the model do in order to predict a sequence as positive or negative. That way of explaining was going to be the **Layer-wise Relevance Propagation** algorithm we talked about before. We also studied possible side effects of the padding in our results, and we wanted to see if thanks to LRP results we could infer some new knowledge about the problem.

We are only going to analyze the 3'UTR sequences with LRP, since they behave way better than the 5'UTR sequences, and we wanted to focus on developing a solid methodology instead of studying two different problems with an incomplete process.

From that moment, we also started working with only the **ICE** sequences as our negative set, discarding all the other subtypes, such as 5'UTR or pseudoGenes. This was because of two reasons: First, there were so many ICE sequences we could work with only them and still having too many. Second, we wanted to simplify the problem even more for these experiments, since LRP was new for us and we wanted to work in a simple environment, and we thought that maybe the relevances would vary between the different subtypes of negative sequence and that could make our results harder to interpret.

Our process started with the implementation of our algorithm, and then the application to our sequences to get their relevances. Our goal was to be able to generate new knowledge from the results of LRP. Along the way, we studied the negative effects the padding may have for calculating the relevances.

4.8.1 Implementation of the algorithm

LRP is a relatively new algorithm, being the first paper that introduces it only five years old [17]. For that reasons, not many implementations of it do yet exist, and certainly not for R, the language we used for this project. For that reason, LRP had to be implemented from scratch for this project, following the formulas and indications of [6].

Our implementation of LRP can be found with more detail in the Appendix 3.

4.8.2 Application of the algorithm

Each sequence has to be evaluated by the algorithm. LRP takes the sequence and returns a listof relevances: one for each character of the input sequence. These values are then combined,and this way we obtain the final relevances. There are two sets of relevances: **positives** and **negatives**.

LRP has to be executed once for each different model, since it looks at the output of the sequence forthat specific model. For this, we only used 1000 sequences (500 positives and 500 negatives),since LRP is a resource-heavy algorithm, and each sequence had to be evaluated five times.

Once we obtain the relevances for these five models, we will proceed to investigate the possible negative effects of the padding in our relevances.

4.8.3 Effects of padding

After calculating the relevances for the five models, we performed some additional experiments with LRP. We were targeting padding and its effects, since we still did not know if padding may have some negative effect on the relevances of the areas where it is present. For that reason, we divided our sequences into "Sequences with padding" and "Sequences without padding".

If the sequences with padding behaved exactly the same as the ones without it, we would consider keeping it, but we wanted to avoid any kind of alteration of the results because of the usage of padding.

If the padding were found to be harmful for the results, we would calculate the relevances of all sequences without padding, so we could obtain a faithful result.

Once obtained all the results, the next steps would be verifying them, and trying to gain more knowledge from them.

4.9 Study of k-mers

In this section, we will try to explain the reason why that last fragment of the sequence holds the highest relevance. For that, we will study the **k-mers** that compose the sequence. The code for this is available at the Appendix 4.

First of all, what is a *k-mer*? A k-mer is basically a subsequence of length k contained within a DNA sequence. For example, there are *monomers*, like A, G, C or T; *2-mers*, like AA, AC, AG... and so on.

In this part of the project we tried to detect relevant patterns inside the sequence, and we will be focusing on the last 46 characters of it, since that area is the most important part of the sequence.

To analyze them, we built a 10 characters long sliding window, which we advanced five characters each time we moved it, and looked for all k-mers in each window with k = {7,8,9}. Then, we would obtain all k-mers for those values of k inside the window, and select those that are **significant**, that is, k-mers that tend to appear more consistently on positive sequences than on negatives. We check if the difference is significative or not with a t-test, checking if the value of p obtained is lesser than 0.05.

We chose those values for k for two reasons: first, smaller values of k correspond to common subsequences. The smaller the k, the more frequent the k-mers are, and the less information we obtain from them: they would be more noise than information. Second, higher values of k (k = 10) correspond to subsequences that did not appear consistently in any type of sequence.

5 Results

In this section, we review the results of all the experiments previously described in the Methodoly part. There are also some experiments explained here that were not mentioned in previous sections, since to understand them we would need some context in form of solutions of other experiments, so it made no sense to mention them before.

5.1 5'UTR - Results of improving the Convolution

We thought we could get more or less the same results simplifying the network. For that, we tried to reduce the parameters by removing layers, reducing filters or rising kernel size. Our original architecture was the following:

- Convolutional 1d layer: 192 filters, Kernel Size 24, ReLU activation
- Max pooling layer
- Dropout layer: 0.25 rate
- Flatten layer
- Dense layer: 50 nodes, ReLU activation
- Dense layer: 1 node, sigmoid activation
- Additional information:
  - Weights initializer: Glorot uniform
  - Optimizer: Nadam

  - Batch size: 125
  - Epochs: 125
  - Learn rate: 0.0001
  - Parameters: 5.889.701

Refer to [1] for a detailed explanation on how we arrived to that version of the network.

The numbers selected for the number of filters or the kernel size were arbitrary, with the only condition of being smaller (or bigger in case of kernel size) than the original value. We can see the evolution of these values in *Table 5*. Each row introduces a new change to the network, while keeping the changes of the rows above it.

Table 5: Evolution of accuracy and parameters when changing the network.

Experiment	Accuracy obtained	Number of Parameters
Original	0.7003127	5.889.701
Removing dropout	0.7040258	5.889.701
Removing Max Pooling Layer	0.6962087	11.784.101
Reducing filters to 128	0.7065663	7.856.101
Reducing filters to 48	0.7078366	2.946.101
Rising kernel size to 75	0.7233731	2.826.149

We can see that every experiment in the table obtains almost the same result in accuracy, while the last one does it with less than 50% of the parameters of the original sequence.

The number of parameters rose with the removal of the pooling layer, but we got rid of it anyway because the presence of this type of layer difficults the implementation of LRP. For this reason, and having the best reduction of the parameters, we ended up choosing the last version, which looks like this:

- Convolutional 1d layer: 48 filters, Kernel Size 75, ReLU activation
- Flatten layer
- Dense layer: 50 nodes, ReLU activation
- Dense layer: 1 node, sigmoid activation
- Additional information:
  - Weights initializer: Glorot uniform
  - Optimizer: Nadam
  - Batch size: 125
  - Epochs: 125
  - Learn rate: 0.0001
  - Parameters: 2.826.149

This network could be simplified even further, but at this point we had other priorities than slightly reducing the parameters, such as expanding our knowledge about this problem. We could focus later on things like performance, once we understood the problem completely.

5.2 5'UTR - Improving our understanding of the problem

In this subsection of the Methodology we talked about the detaling of the evaluation results, the hard and easy sequences, and we studied the length of the sequences. Here we will show the results for those parts.

5.2.1 Detailed evaluations results

This were the values needed for analyzing the evaluation of our network:

- Trained: Rate of Trained Right, Rate of success of the prediction for a sequence used for the training of the networks
- Untrained: Rate of Untrained Right, Rate of success of the prediction for a sequence used for the evaluation of the networks
- DA: Disease Association, whether the sequence is positive or negative

The results obtained can be seen in *Table 6*. There are ten rows total: One for each one of the types of sequence (positive or negative), once per each of the five models trained. The code for obtaining these results can be found in the Appendix 2.

Table 6: Detailed breakdown for the original experiment

Model	Trained	Untrained	DA
1	0.949	0.930	1
2	0.943	0.953	1
3	0.962	0.953	1
4	0.949	0.945	1
5	0.943	0.939	1
1	0.783	0.556	0
2	0.846	0.551	0
3	0.758	0.433	0
4	0.762	0.559	0
5	0.656	0.516	0

Conclusions

We can clearly see in *Table 6* that, even though positive genes are predicted with a high accuracy, unseen negative genes do not get the same result, having an accuracy of 0.55 or less. This means that our models does not correctly learn what makes a sequence negative, and tends to classify sequences as positive. This could mean the data used was not suited for learning from them.

Our first objective was to see whether we could or not identify disease associated genes using only certain elements in the sequence, such as the 5'UTR, like in this experiment. This result makes us think that 5'UTRs do not seem enough to make such a distinction, which would be our first contribution in this project.

5.2.2 Results for hard and easy sequences

Before, we talked about how to obtain these hard and easy sequences, and introduced the concept of a PCA study. Now, we will show the results for the PCA study, and our conclusions on the matter.

Remember that PCA represents in a graph points that corresponds to a single individual each. Points that are close have similar characteristics, while those that are far are different. In *Figure 17* we can see this representation:

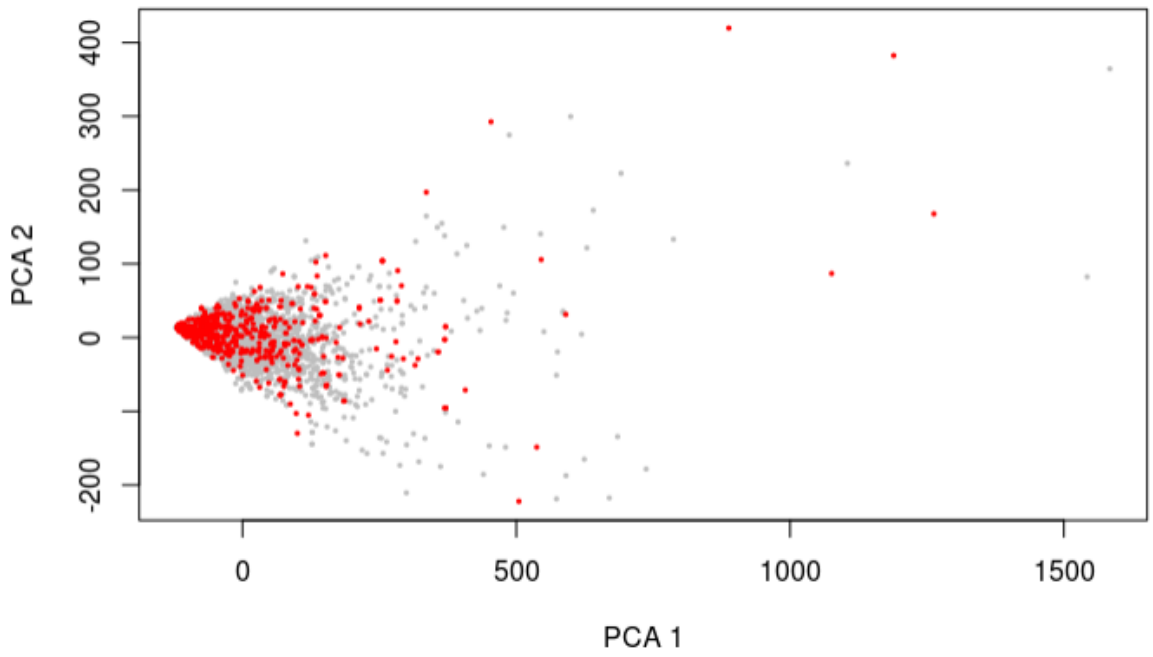

Figure 17: Results of the PCA study of alphabet frequency on hard and easy sequences. Red dots are the hard genes, and grey dots are the easy genes.

As we can see in the plot, the red (hard) and grey (easy) points are very close, with no difference between them, as if they were a single group. This means there is almost no distance between these two groups for this feature, which is a clear example of them not being distinguishable by this characteristic alone.

Conclusions

We selected the easiest and hardest sequences to predict for our models, and tried to see if they differed in their alphabet frequency, with a negative answer to this question.

5.2.3 Study of the Sequence Treatment

In the Methodology section, we introduced an experiment consisting of treating the same set of sequences in four different ways:

- Original sequence, padding at the end → ABCXX
- Reversed sequence, padding at the end → CBAXX
- Original sequence, padding at the start → XXABC
- Reversed sequence, padding at the start → XXCBA

We used these sets to train our network again, writing down the results for their comparison. The results for these experiments were the following (we are looking at validation accuracy of the model):

- Original sequence, padding at the end → 0.520313
- **Reversed sequence, padding at the end → 0.729919**
- Original sequence, padding at the start → 0.643932

– Reversed sequence, padding at the start → 0.550224

We can see that the sequences that have the padding next to the start of the sequenced, be it original or reversed, have better results than those that do not. In fact, those sequences that do not abide by this only behave slightly better than the random predictor.

As for the two experiments that comply with this restriction, the one that behaves the best is the one that works with the reversed sequences. This made us think that the sequences had to be reversed to obtain the best result, and we had many theories to explain why (such as the need to follow the *strand direction*), but, as we will see in future sections, the answer was much more simple than that (refer to *section 5.4.6 New knowledge from LRP).*

The other question was why the sequences with the padding next to the start of the sequence behaved better than those that had it next to the end. We could only theorize about it, since we did not yet had a way to know what parts of the sequence were more relevant. We thought that maybe the most relevant parts of the sequences were at the end of them, and having the padding near them could distort the filters and making their work harder.

To look for an answer of that, we decided to study the **length of the best sequences**. This was our theory: Being the original sequence "ABC", the sequences of the "CBAXX" experiment that tend to classify correctly are longer, while those that are incorrectly classified are usually shorter, meaning this that they have **more padding**, and that this padding is **closer to the theoretically most relevant area, the end of the sequence**.

If that were true, we would have a reason to try to minimize, or even completely remove the padding, since it would be sabotaging our results.

We checked the distribution of the lengths to see if it was true that easy negative sequences tend to be longer while hard negative sequences are shorter. The code developed to check this is referenced in the Appendix 2. First, we plotted the distribution of the length of all sequences *(Figure 18)*:

Next, we do the same, but this time with only easy or hard sequences *(Figure 19)*:

These plots do not seem to confirm our theory that good (easy) sequences tend to be longer than bad (hard) sequences, as they seem to have a very similar distribution. We tried grouping the lengths in small intervals to see if we could detect a difference that way in *Figure 20*:

This time, we can see that easy sequences usually are slightly longer than negative sequences, needing less padding than them, but even so their differences are not significative enough, so we ended up discarding this hypothesis of easy sequences being longer. Why the XXCBA and ABCXX sequences perform worse than the others is a issue we have not still addressed.

Getting back to the four experiments, we decided to investigate what could be causing the difference in accuracy between the "CBAXX" and "XXABC" cases, since, even though one is the mirror of the other, they do obtain different results.

For that, we looked for **overlappings** in the easy and hard sequences of both experiments. We thought that, if the easy or hard sequences were different in both cases, that would mean that sequences that get good predictions in a direction, get bad predictions in the other, and maybe we could invert only certain sequences depending on the treatment that works best for them.

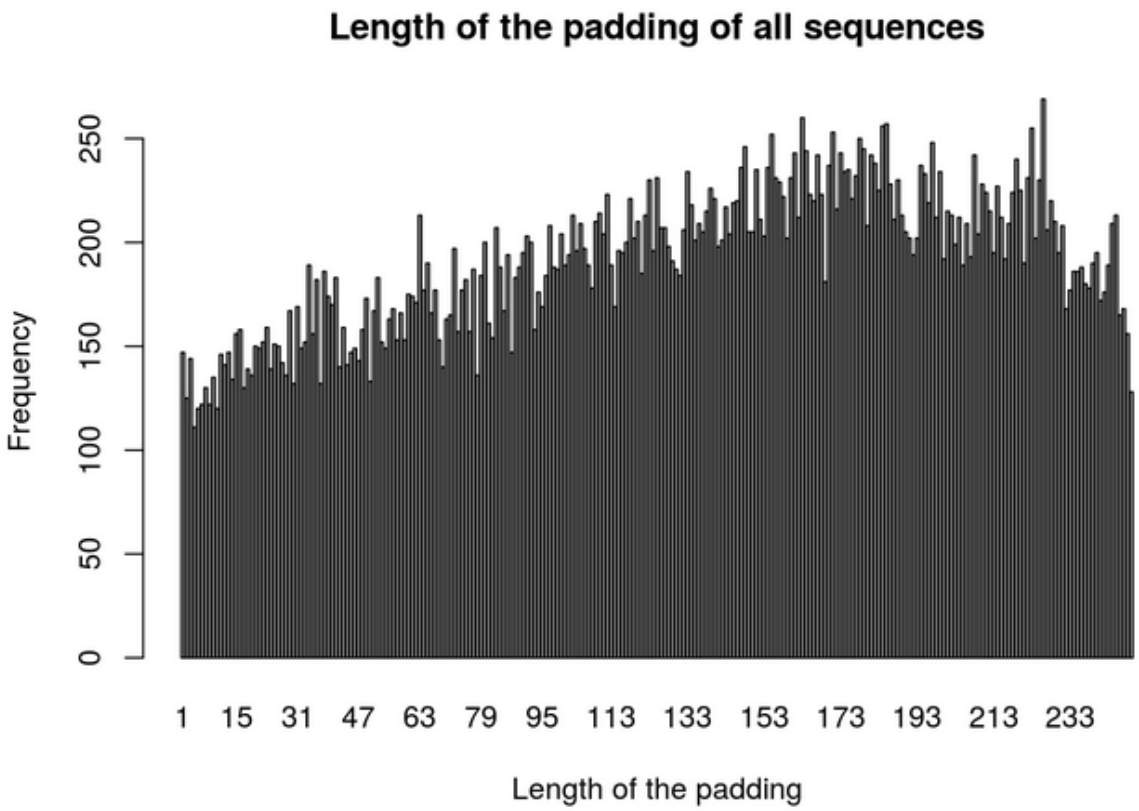

Figure 18: Distribution of the length of the paddings of all sequences.

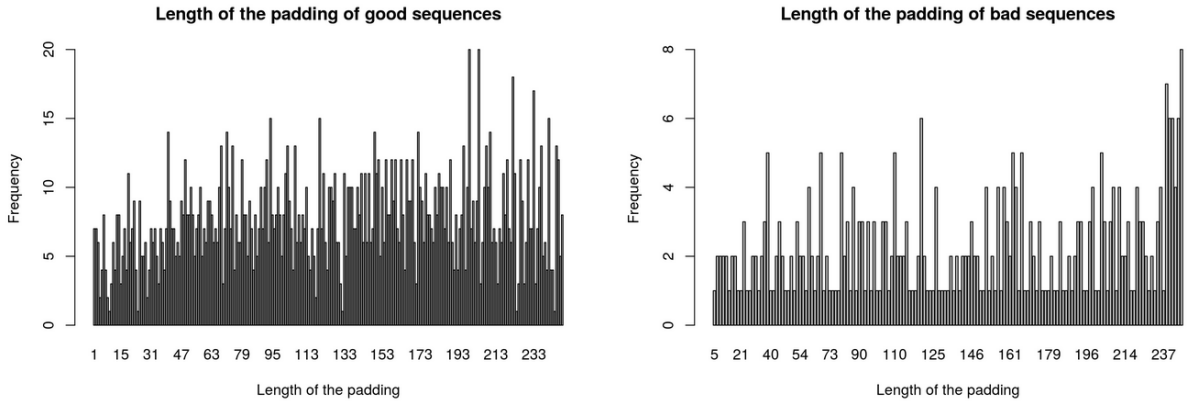

Figure 19: Distribution of the length of the paddings of easy (left) and hard (right) sequences.

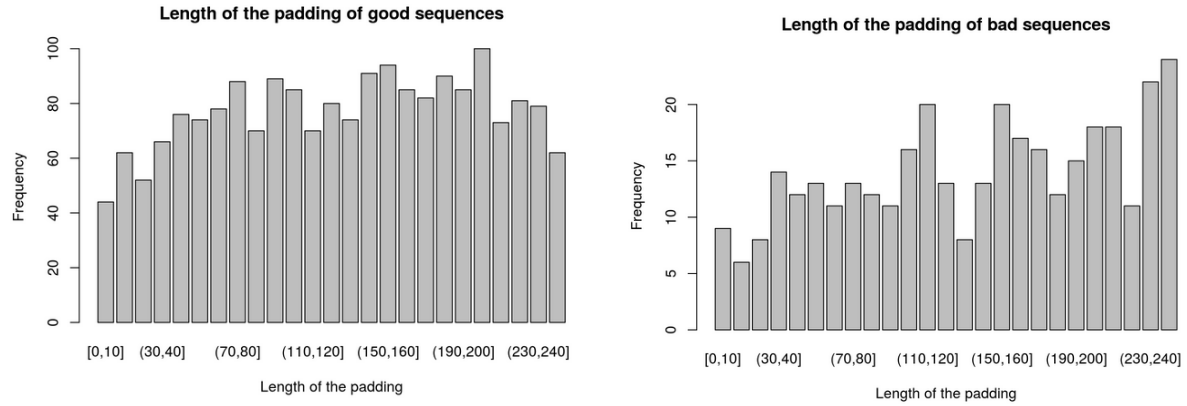

Figure 20: Distribution of the length of the paddings of easy (left) and hard (right) sequences.

The results for this experiment can be seen in *Table 7*. The first column show the number of easy and hard sequences that are exclusive to CBAXX (meaning that in the XXACB experiment they are no longer considered either easy or hard), the middle column shows the number of sequences that are easy or hard for both experiments, and the third column is like the first, but this time for the XXABC experiment.

Table 7: Number of sequences exclusive and shared.

Easy CBAXX	Easy shared	Easy XXABC
623	2021	428
Hard CBAXX	Hard shared	Hard XXABC
355	131	477

There is high overlapping in the easy sequences compared to the total number of their sequences, and that is bad news, since that means we cannot reverse some sequences depending on the treatment that would suit them best, since they are usually the same sequences always. That means an easy sequence tends to be easy regardless of its direction.

As for the hard sequences, they do not have the high overlap of the easy sequences, but this is not a deal as big as easy sequences not overlapping, since even though we could make some sequences not as hard to predict, nothing tells us that they will not be only slightly easier than before.

Conclusions

We know that the CBAXX structure works best for this experiment, and that having the padding next to the end of the sequence gets us a very bad result, but our experiments for knowing exactly why have not been succesful.

5.2.4 Optimal length of the Sequences

As we introduced in the Methodology section, we reviewed the minimum length of the sequences, and decided whether we should make any changes to it. The code for this can be found at the Appendix 2.

First, we plotted the distribution of lengths of the sequences, so that we could have some context *(Figure 21)*. Since most of the sequences have lengths between 1 and 300 characters, we will zoom that area so that we can see clearly what can be found there.

We see that most sequences are 50 to 150 characters long, and from there the density starts decreasing the more the length increases. We also see that a relevant amount of sequences are shorter than 30 characters, and wonder if they should be removed, because maybe sequences that short do not contain enough significative patterns, and could be difficulting the task of prediction. We took this into consideration for future experiments but we did not change the sequences we selected, because, as we explain in the next section, that was the last experiment we made with this dataset.

Conclusions

We learnt that there are lots of very short sequences that might have to be removed at some point, because they would be almost entirely be composed of padding and too short to contain any relevant pattern, and thus might became bad examples for training.

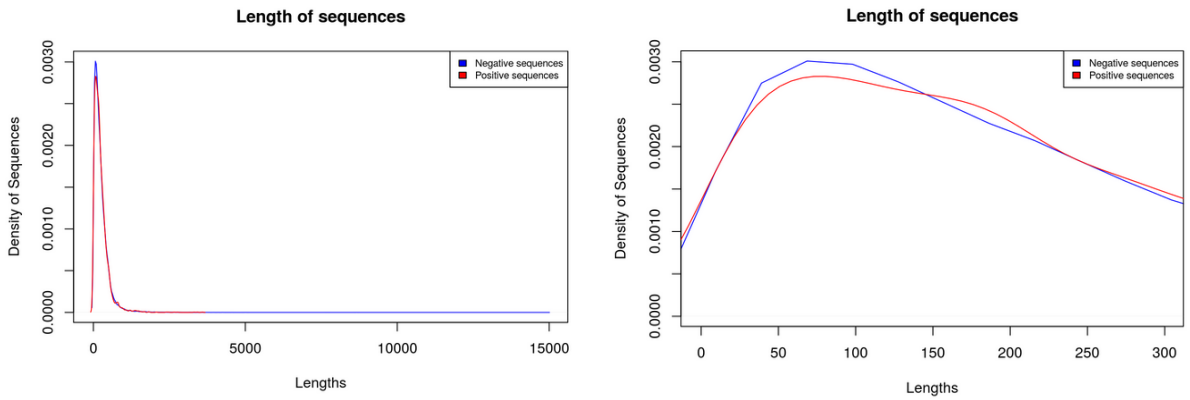

Figure 21: Distribution of the lengths of the sequences. Right image is the same but zoomed

5.3 3'UTR - A change of dataset

Now, we will present the results of the experiments we made with the 3'UTR dataset. We compared it to the 5'UTR, addressed the topic of the unbalance, and reduced the size of the network even more.

5.3.1 3'UTR - Results of the comparison of the two experiments

The old experiment achieved a validation accuracy of approximately 0.7. This new dataset, on the other hand, got a validation accuracy of approximately **0.95**. This is a much better and promising result than before *(Figure 22)*. Note that we also need way less epochs to obtain the best result, as seen in *Figure 22*.

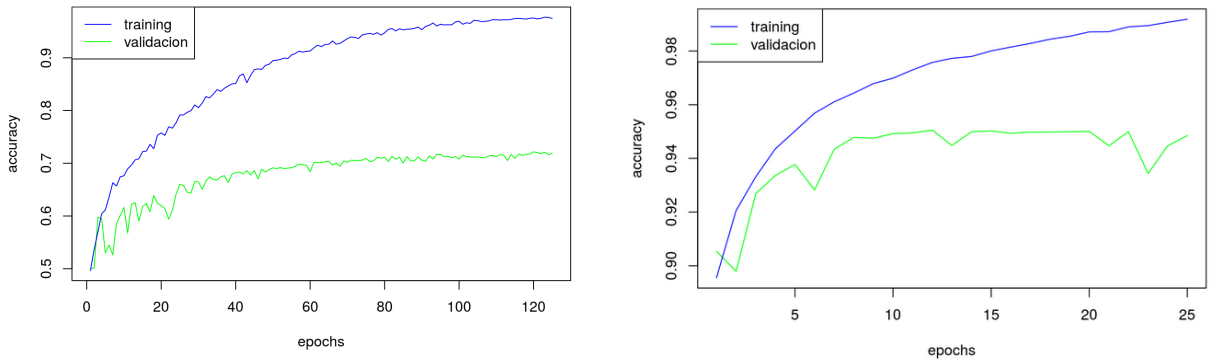

Figure 22: Left: Accuracy of the older experiment, Right: Accuracy of the new experiment. Keep an eye on the y-axis scale, since it is not the same in both of them.

This does not mean our first problem was a failure, it only means that the hidden information found in 5'UTR sequences only lets us classify genes into intellectual disability and non-intellectual disability with a 70% accuracy. This is interesting because it is suggesting that inside the sequence there are elements that allow us to classify intellectual disability associated genes, and that those characters inside the sequence are definitely forming different patterns in different types of genes.

We will also show the detailed results for this experiments, as we did for the old one in *section 5.2.1 Detailed evaluation results*. We can see this breakdown of the results in *Table 8*.

Table 8: Detailed breakdown for the experiment with the new dataset

Model	TR	TW	UtR	UtW	RTR	RUtR	DA
1	13873	232	3202	324	0.983	0.908	1
2	13768	337	3164	362	0.976	0.897	1
3	13833	272	3174	352	0.980	0.900	1
4	13905	200	3262	264	0.985	0.925	1
5	13803	302	3236	290	0.978	0.917	1
1	14100	5	139000	7127	0.999	0.951	0
2	14103	2	139920	6207	0.999	0.957	0
3	14098	7	140018	6109	0.999	0.958	0
4	14093	12	136958	9169	0.999	0.937	0
5	14103	2	138241	7886	0.999	0.946	0

This confirms us that the problem with the results for the negative sequences of our original dataset was caused by the data, and not by the network.

5.3.2 3'UTR - Addressing the unbalance

We analyzed the results for every subtype of negative sequence, to see this way if a type if performing better or worse, or if it is under or over-represented in the training of the models. The results of the analysis can be found in *Table 9*. This is what each of the columns for said table means:

- NumberSuccess: The number of sequences of this subtype correctly predicted across the five models
- NumberFailures: The number of sequences of this subtype incorrectly predicted across the five models
- RateSuccess: The rate of sequences of this subtype correctly predicted across the five models
- RateTraining: The rate this subtype of sequence constitutes from the training sets across the five models

Table 9: Statistics for the multiple subtypes of negative sequence

Type	NumberSuccess	NumberFailures	RateSuccess	RateTraining
ICE	626888	21682	0.9665	0.8100
lncRNA	12482	5958	0.6768	0.0228
ncRNA	16233	2242	0.8786	0.0228
pseudoGene	7586	1929	0.7972	0.0120
5'UTR	94844	11316	0.8934	0.1321

Note that every sequence is evaluated five times, one for each model, so the number of successes plus the number of failures equals to five times the original amount of sequences.

With these results, it is clear that the ICE sequences are the ones that are obtaining the best result, while the other types range from slightly worse than them (such as 5'UTR) to an

even worse result than the experiment of the original project (like lncRNA). Still, we do not know if that was caused because of the low representation of these sequences (most of them are not even a 3% of the total sequences used for training), or because these types of sequences perform worse for other reasons.

After this, our next step would be trying to reduce the network even more, in anticipation of LRP.

5.3.3 3'UTR - Reducing the size of the network

In *section 4.5 Improving the Convolution* we presented the original version of our Convolution Neural Network, together with a revision we made to reduce its parameters. Since LRP is a very resource-heavy algorithm, the smaller the network, the more sequences we could calculate their relevances with our resources. For that, we decided to reduce again the parameters of our network.

In *Table 10* we will see the 10 experiments we made for this. First row indicates the original model (#1), and the following are the different versions we tried. Our goal, as specified before, was to reduce the parameters as much as we could while trying not to reduce the accuracy or rise the loss.

Values of accuracy and loss are the mean of the results for each one of the five models. Values in red for the filters, kernel size or nodes mean a change respecting the original value (#1). The rate of reduction of parameters is also with regard to the original value.

Table 10: Experiments for further reducing the network.

#	Filters	Kernel size	Nodes	Parameters	Rate reduction params	Accuracy	Loss
1	48	75	50	1765349[14]	0	0.957	0.126
2	48	101	50	1704197	0.0358	0.956	0.128
3	48	8	50	1922933	-0.081	0.942	0.159
4	24	75	50	882725	0.999	0.955	0.119
5	48	75	101	884499	0.995	0.957	0.119
6	48	101	25	854547	1.065	0.961	0.112
7	24	75	25	442275	2.991	0.954	0.121
8	24	101	50	852149	1.071	0.954	0.121
9	24	101	25	427299	3.131	0.958	0.114
10	12	101	18	154189	10.449	0.950	0.131

Of all these experiments, we ended up choosing the 10^{th} version, since it reduces the parameters to the minimum value we obtained, while not having any effect to the accuracy nor the loss. The resulting network architecture can be seen in *Figure 23*.

Conclusions

[14]This number differs from the number of parameters of *section 4.5 Improving the Convolution* because we were working with smaller sequences (202 characters instead of 250). This is because this simplification of the network was performed chronologically after applying LRP for the first time and deciding that we needed to reduce the length of the sequences. Refer to *section 5.4.3 Experiment without padding* for more information about this reduction of the sequence length.

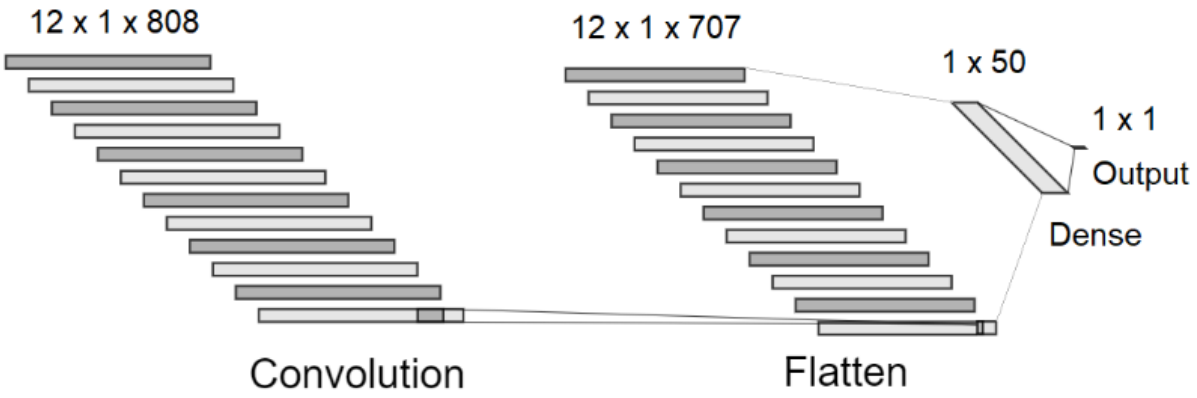

Figure 23: Final architecture of our CNN.

The new dataset we tried performed much better than the original, with zero changes to the procedure. Since our objective is to establish a good general methodology, we do not really care about the domain in which we work to develop it. There seems to be some kind of internal discrepancies in the results among the different subtypes of the negative sequences, but we have not investigated why.

At this point we had no more improvements for our experiment, so the next step was to **explain our results** with LRP.

5.4 Explaining the 3'UTR Results with LRP

In the Methodology section, we explained the work we were going to do with LRP, explained the implementation, and then we introduced the topic of the possible negative effects of padding. This time, we are going to talk about the application of the algorithm, showing the results of LRP, and then we are going to investigate said effects of padding.

5.4.1 Application of the algorithm

First, we obtained the relevances for the five models for a set of 1000 sequences. The relevances obtained for each one of the five models can be seen in *Figure 24*. Remember that LRP has to be executed once for each different model, since it looks at the output of the sequence for that specific model.

There are three things we can point in these plots. First, we see that the distributions of positive and negative relevances are very similar. Second, the distributions of the five models are also very similar. And third, the most important part of the sequence seems to be the start.

Remember that the preprocessing of the sequence starts by **reversing the sequence**, and then **adding padding or cutting it**. What we mean by that is that the *start* of this sequence is in reality the *end* of the original sequence, reversed. In conclusion, the ending of the original sequence seems to be the most relevant section of it.

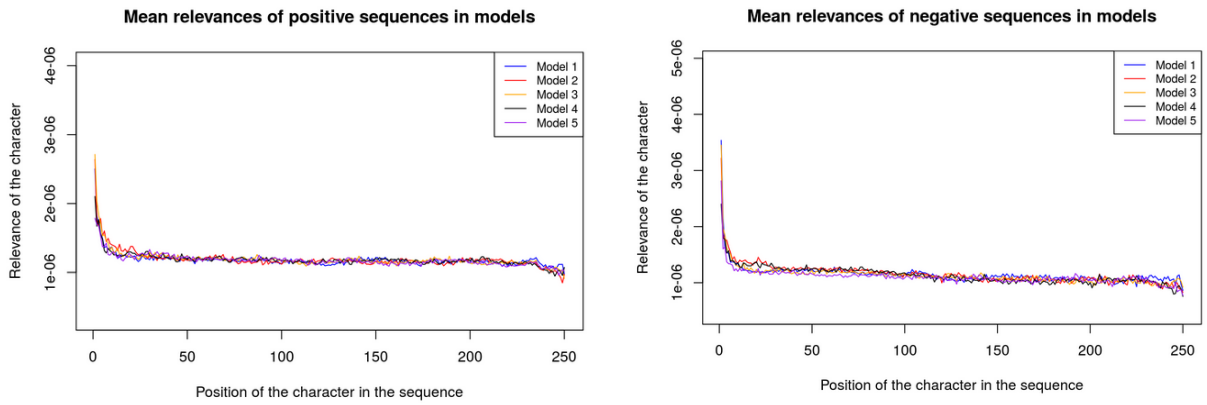

Figure 24: Left: Relevances of the positive sequences of the five models, Right: Relevances of the negative sequences of the five models.

5.4.2 Effects of padding

As we said before, we feared the padding may be disturbing the results of LRP, so we wanted to study the effects of it by removing the padding an recalculating the relevances. We divided our sequences into "Sequences with padding" and "Sequences without padding", and plotted them *(Figure 25)*. Since, as we have already seen in *Figure 24*, all models have the same distribution of relevances, these relevances we are showing in *Figure 25* come from only one of the five models, since calculating the relevances for all the sequences with all the models is a very resource heavy task.

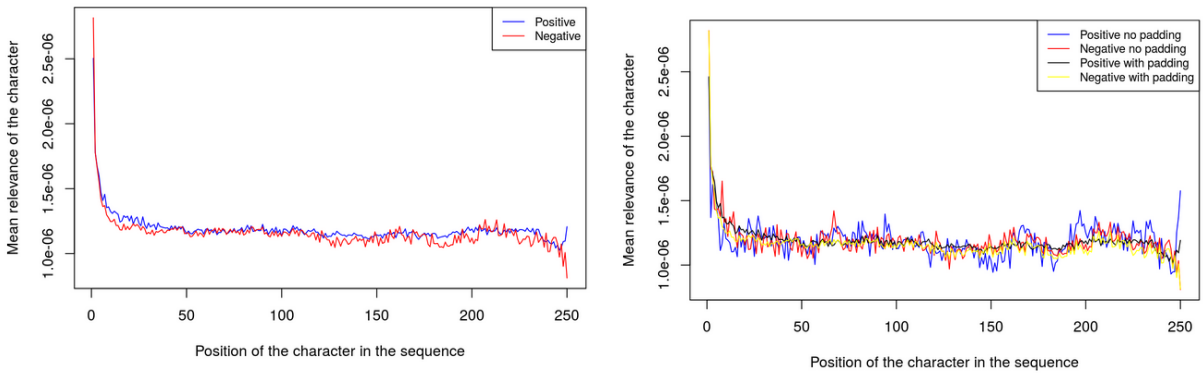

Figure 25: Left: Mean relevances of positive and negative sequences, Right: Mean relevances of positive and negative sequences, with and without padding.

Even though sequences without padding seem to have higher variance than padding sequences, their differences seem scarce. Even so, this smaller variance in padded sequences could maybe lead to some of the relevant areas being masked by the padding. For example, we can see in the right plot of *Figure 20* that from character 200 to character 250 there is an area of higher relevance for the red and blue lines, that does not seem to appear in the yellow or black lines, or in the left plot(the one that includes all the sequences). This is what led us to think that maybe padding was hiding some areas of higher relevance.

For that reason, we decided to start working with only the sequences that did not need padding, that is, sequences naturally equal or longer than 250 characters, and remove padding completely from our experiments. Also, since the relevances obtained for the five models are extremely similar, and calculating the relevances for all of them is very costly, from now on we

will only show the relevances of the first model.

5.4.3 Experiment without padding

For the reasons exposed before, we decided to repeat the whole experiment removing the sequences with padding. For that, we needed sequences that were at least 250 characters long. The problem with that is that the number of sequences for that length was completely unbalanced: We had 14.429 positive sequences longer or equal than 250 characters, but only 7243 negative sequences of that length.

Because of that, we had two choices: One was to perform the experiment like that and work around the unbalance as we did before (either by dividing the positive sequences into two subsets and switching them around, or by selecting only 7k positive sequences and perform a traditional cross-validation), and the other was to choose an alternative length for our sequences, since the length of 250 characters we were using was an arbitrary choice.

We analyzed the lengths of the two types of sequence, and we saw that around the length of 202 characters, the amount of sequences was more or less the same for both types, with a difference of only 371 sequences. For that reason, from now on, we started working with sequences of 202 characters or more.

For the split into training and test, we had to stop using our old method of using all the positive sequences in all models and a subset of the negative sequences, since now we had the same amount of sequences from both types. Instead, we started using traditional cross-validation split, dividing the sets into five different subsets (or *folds*), and rotating them for each model. We can see an example of traditional cross-validation in *Figure 26*.

This way we developed an alternative methodology, although similar to the previous one, to perform the experiment with shorter, unpadded sequences.

5.4.4 Relevances of the unpadded sequences

In figures *27* and *28* we will show the same relevances as before: first the mean relevances of each model for the first 1000 sequences (500 of each type of positive and negative), and then the relevances of all the sequences for the first model (since, like before, the distributions of all the models are very similar).

We see that, like before, in the padding experiment, the highest relevance seems to be at the start of the sequences (remember, the start of these sequences is equivalent to the end of the original sequences), while the rest of the sequence has a very low relevance in comparison. This confirms the previous result that the most important area of the original sequence is at the very end of it, regardless of the padding.

5.4.5 Verifying the result of LRP

After applying LRP to obtain all the relevances for our sequences, we still were not sure if those results were true, since, as we saw in *section 3.6 Limitations of LRP*, we were not guaranteed

[15]Image extracted from `https://scikit-learn.org/stable/modules/cross_validation.html`

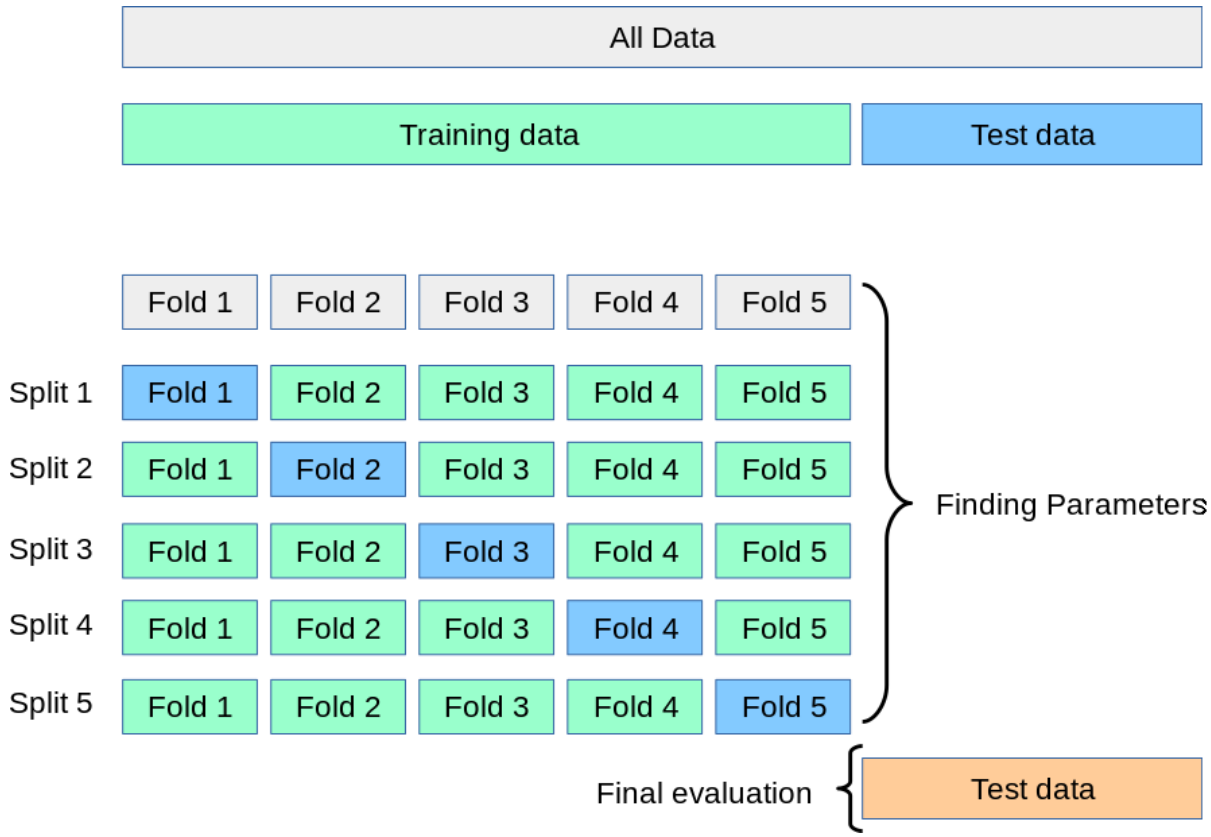

Figure 26: Visual guide for explaining how does cross-validation work. Each split goes to a different model.[15]

a correct result for LRP, since we only tried one version with one set of parameters. For that reason, we tried to verify the correctness of the result doing some additional experiments.

We had to do two things to ensure that LRP was working as intended: First, develop a very simple artificial example to confirm that the output of LRP made sense. Afterwards, we had to perform some experiments to ensure that the result LRP returned for this particular experiment was correct.

Simple example

First, we developed an artificial example to know if our LRP was working as intented. We created a series of random sequences of 800 characters, with different combinations of the characters "T", "G" and "A", and then we added 200 extra characters. Our positive sequences would have only "C" as those 200 characters, and our negative sequences would have "T". In *Table 11* we can see some examples of positive and negative sequences (although shorter). As we can see, classifying them into positive or negative sequences is very easy, and our network could do this with a 100% of accuracy in just an epoch.

Table 11: Examples of basic sequences

Example	Sequence	Class
#1	GTAGTAGTAGTGATTATATTATTATAATATGATATCCCCCCCC	Positive
#2	TATATTTATTATATGAGGAGTAGTTATAATTGTATTTTTTTTT	Negative
#3	GATTATTTATATATGTTAAGATGATTGAGATTATCCCCCCCCC	Positive

If LRP was working as intended, we figured that the result to obtain would be a plot showing

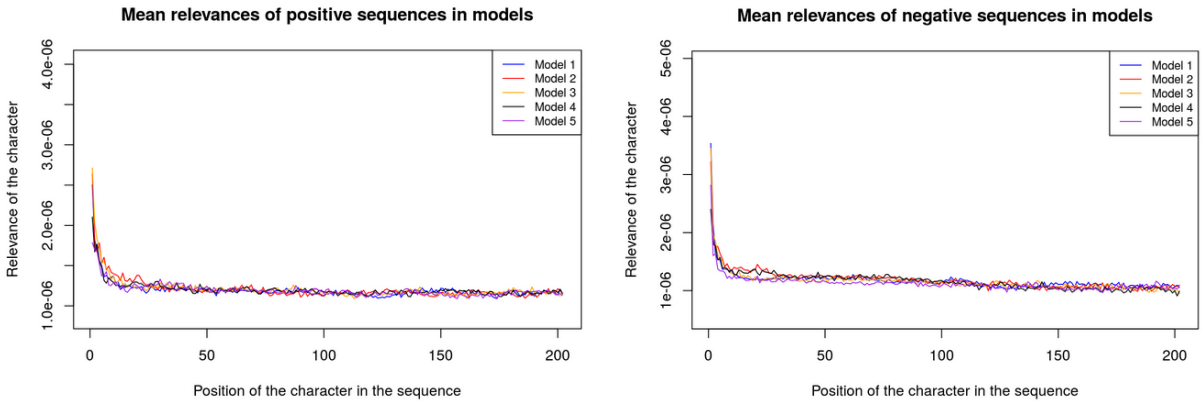

Figure 27: Left: Relevances of the positive sequences, Right: Relevances of the negative sequences.

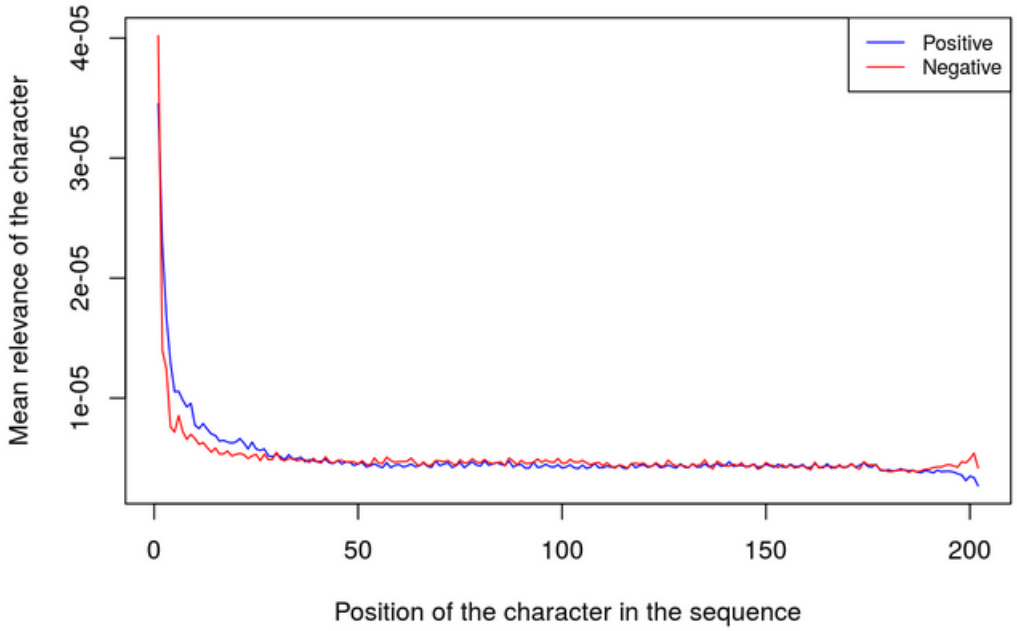

Figure 28: Mean positive and negative relevances with no padding

a spike of relevance for the last 50 characters. If we obtained that, it would be a great sign that our algorithm works as intended. In *Figure* we can see the relevances obtained for this easy experiment.

The plots show clearly how there is a spike in the relevance at the x = 800 mark, just were we add the Cs or Ts. Before, there is way less relevance, since that is the part of the sequence formed by random characters. This is a clear example of LRP detecting the most relevant part of the sequence, which let us know that our LRP algorithm worked as intended for at least a given set of sequences.

Next, we tried to see if our particular results for the 3'UTR problem made sense.

Verification of our results

According to LRP, the most important part of the sequence seems to be at the very end of it. To evaluate the accuracy of this, we performed some extra experiments, in which we do not work with the whole 202 long sequence, but with even smaller sections of it.

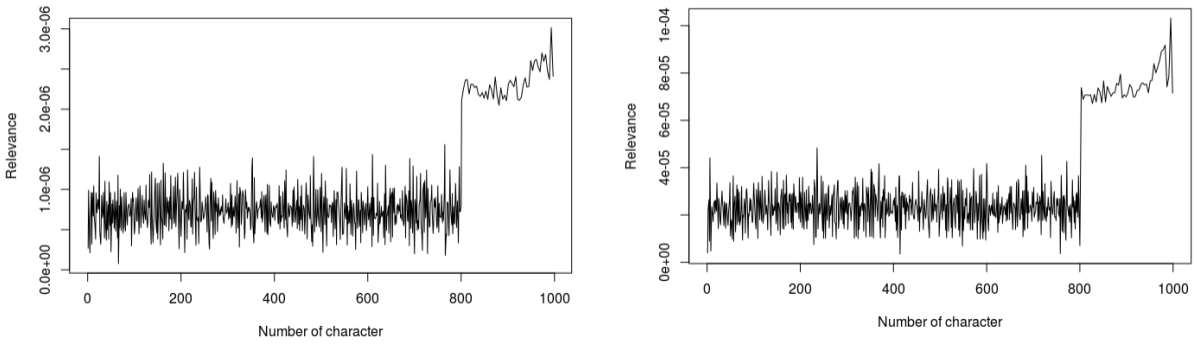

Figure 29: Left: Negative relevances, Right: Positive relevances, both for the easy artificial problem.

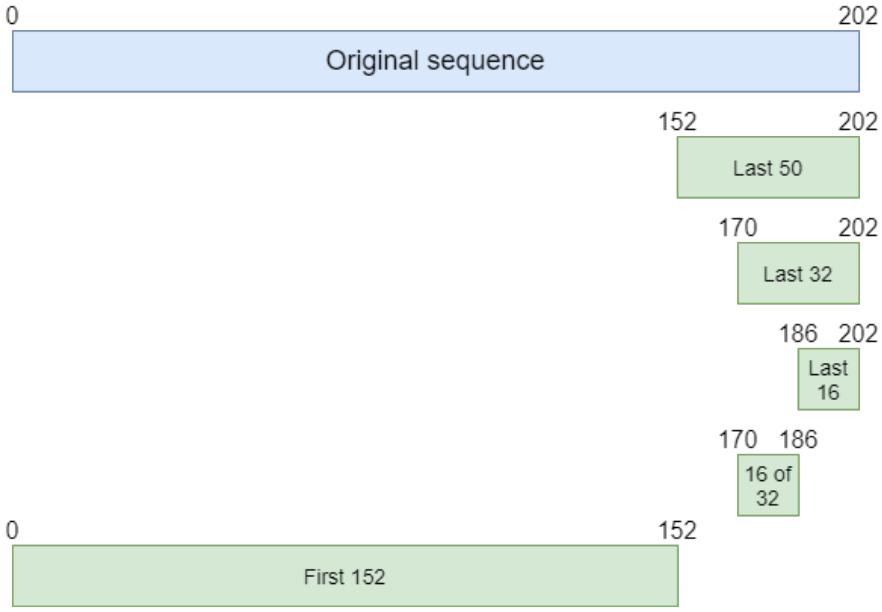

Figure 30: Visual guide for the six experiments we performed

We performed five additional experiments in total *(Figure 30)*. The first is the original experiment we performed, with the whole sequence of 202 characters. The following three train a network, each with less characters of the sequence than the last, to try to prove if it is true that the end of the sequence is the most important part by obtaining a similar result. The fifth experiment trains a network using 16 of the last 32 characters, and the last experiment trains the network with everything but that end of the sequence. If our hypothesis is true, the accuracy should be lowering as we do more experiments, with the last experiment obtaining the worst accuracy.

The accuracy results for these experiments were the following:

- Original sequence: 0.95 validation accuracy
- Last 50: 0.93 validation accuracy
- Last 32: 0.92 validation accuracy
- Last 16: 0.89 validation accuracy
- 16 of 32: 0.87 validation accuracy
- First 152: 0.84 validation accuracy

As we thought, the experiments that obtain the best results are those that contain the ending of the sequence, and only 32 characters are needed to obtain a result almost as good

as with the whole sequence. Obviously, the higher the number of characters, the better the result, but this experiment's only purpose was to prove that, indeed, the ending of the sequence seems to be the most important part, because whenever we train a model with a sequence that contains it, no matter the size of it, we obtain a better result than when we train without it.

Even so, the rest of the sequence seems to behave relatively well even without this important section, since we could obtain a result of 0.84 accuracy without this fragment of the sequence. This might mean that, even though the ending of the sequence is the most important part of it, the rest of the sequence also contains information that could differentiate sequences of different types.

The last thing we learn from this is that, even though LRP says that the most important part is the **very end** of the sequence, that is, the last two or three characters, The results for the "Last 16" and the "16 of 32" experiments were pretty similar. This could mean one of two things: maybe the results of LRP were not accurate enough, and we would need to use another version of it to obtain a more precise result, or maybe the representation is not totally accurate, and the difference between the end of the sequence and the rest of it is not that big.

To check if it is the second hypothesis, we could remake the plot, this time modifying the lower limit of the y-axis to be zero. We can see in *Figure 31* that the proportion is more or less the same, so we can discard that second hypothesis.

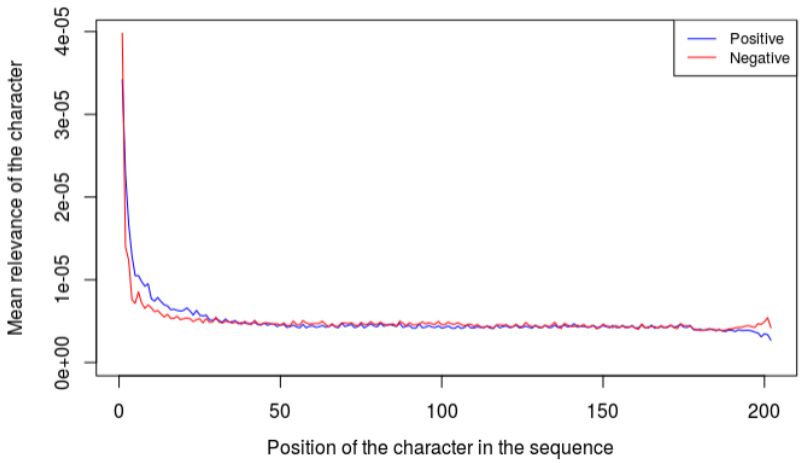

Figure 31: Mean relevances of positive and negative sequences, with no padding, and the lower limit of the y-axis as zero

Now that we verified that LRP results are correct, the next step was to infer new knowledge from them.

5.4.6 New knowledge from LRP

Thanks to this knowledge provided by LRP, we now know why reversing the sequences provided better results, and it was not related to strand direction, as we thought, but instead it was related to the **inclusion of the most relevant part of the sequence**.

As we mentioned in *section 4.6.3 Study of the Sequence Treatment*, the treatment of our sequences is reversing, following by padding/cutting, and then one-hot encoding.

For longer sequences, if we reverse the sequence and then select only the first 202 characters, we are selecting the last characters of the original sequence. If, instead, we do not reverse our

sequence, then by selecting the first 202 characters we would be discarding the end of the sequence *(Figure 32)*. This explains why we obtained better results reversing the sequence: it had nothing to be with the direction of the sequence, but with the characters we selected or discarded. This is only one example of new knowledge we obtain thanks to LRP.

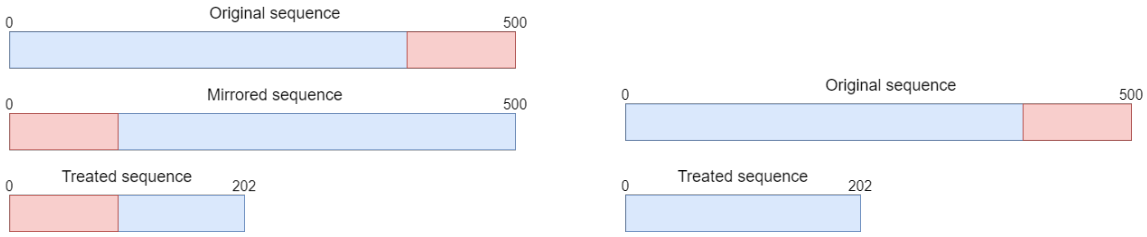

Figure 32: In red: The most important part of the sequence. In blue: The rest of the sequence. Left: The sequence we obtain by reversing the original sequence. Right: The sequence we obtain by not reversing the original sequence

To prove this, we will perform four aditional experiments, which will train our network using:

- The first 202 characters
- The first 202 characters in reverse order
- The last 202 characters
- The last 202 characters in reverse order

If our hypothesis is true, experiments 1 and 2, and experiments 3 and 4 should get similar results, with experiments 1 and 2 obtaining less accuracy than 3 and 4. This would mean that, indeed, the better result we obtained by mirroring the sequence had nothing to do with the fact that our characters were in an specific order, but due to the most important part of the sequence was being discarded. These were the results:

- First: 0.8491 accuracy
- First reverse: 0.8683 accuracy
- Last: 0.9384 accuracy
- Last reverse: 0.9400 accuracy

Our experiments seem to support this idea of the order of the characters having almost nothing to do with the result, since the sequences that obtain the best result are those that have not discarded the end of the original sequence, but since our reversed sequences always obtain a slightly better result than the original sequences, maybe they marginally improve the performance of our network, for reasons that are still not clear. Our theory is that, maybe, the strand direction does make a small difference, even though it is not comparable with the inclusion of the ending of the sequence.

Regardless of that, this experiment let us confirm that the most important thing we have to keep in mind when treating the sequence is not to discard the ending of it.

5.4.7 Conclusions of LRP

After applying the LRP algorithm to all the sequences, both with and without padding, it seems that the most important section of the sequence is at its very end. With this knowledge, we were able to make additional deductions.

Also, we verified the result of LRP, and even though it does not seem accurate at 100%, it did its job by telling us that we have to pay attention to the ending of the sequences.

Although we know which is the most important area, we still do not know exactly why. What happens at that point of the sequence that makes our model be so confident a sequence is 3'UTR or not? That is what we are trying to find in the next section.

5.5 Results of the study of k-mers

In this section of the Methodology, we introduced the k-mer analysis, in which we obtain the k-mers of the last 46 characters of the sequences via a sliding window, and then selecting those that are significant.

Once obtained all the significant k-mers, we plotted them. We can see the results in *Figure 33*. According to these results, the most important area is from character 17 to 31, in which are not only the majority of 7-mers, but also almost all the 8-mers, and the only 9-mer found. This means that the results do not completely agree with the LRP results, since LRP believes that the very end of the original sequence (which corresponds in this plot with characters 0 to 11, since this study works with the mirrored sequence.) is the most important part of it.

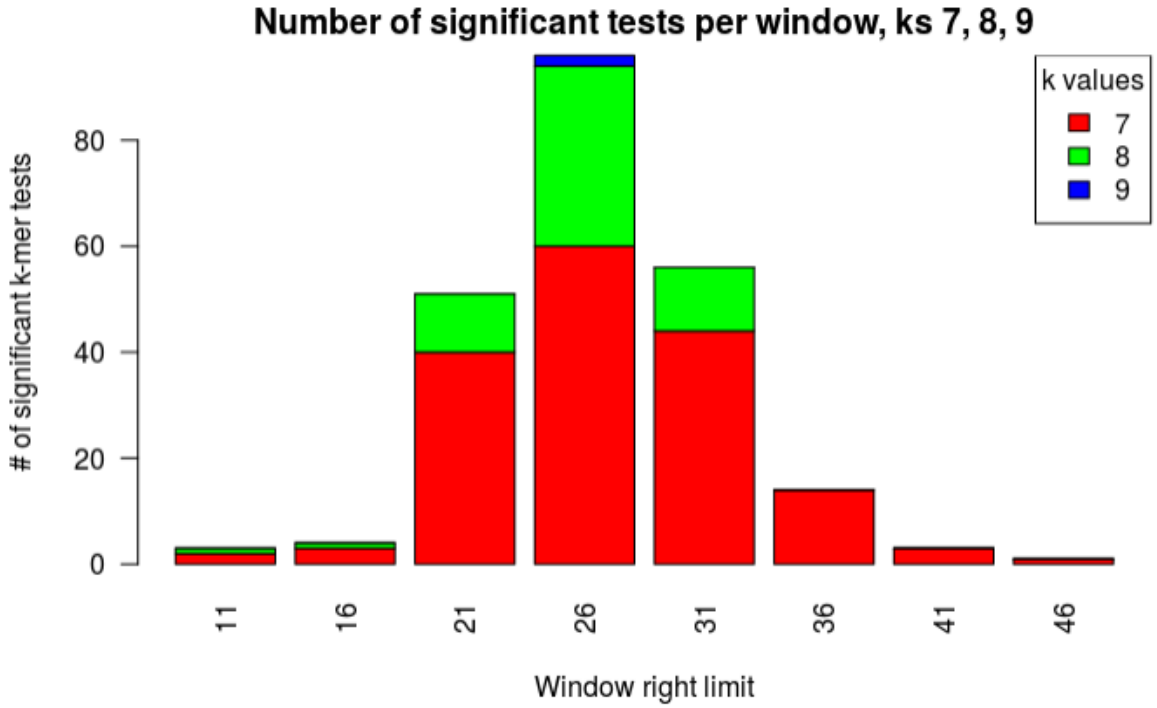

Figure 33: Results of the k-mer study. Character 1 corresponds to the last character of the sequence, and character 46 would be the character number 156, since this study works with the mirrored sequence.

The study from *section 5.4.5 Verifying the result of LRP* seems to also disagree with this, since this claims that characters 17 to 31 are the most important by far, while previously we tried training the network with only those characters and got a slightly worse result than training with the first 16, although not enough to be considered a real, important difference, but enough to know that these 16 characters are not as relevant as this k-mer study claims them to be in comparison with the rest of the characters of this fragment.

The list of k-mers found, together with the number of different windows in which they appear more in 3'UTR sequences with a significative difference over non-3'UTR sequences can be found in *Figure 34*. We can see that the adenine base ("A") is overrepresented in these fragments. We will talk more about this detail in the following subsection.

AAATAAA	AATAAAA	AAAATAA	AAATAAC	AAATAAG	AAATAAT	AATAAAT	ATAAAAA	CAAATAA	GAAATAA	TAAAAAT
6	5	4	4	4	4	4	4	4	4	4
TAAAATA	TAAATAA	AAAAAAA	AAAAATA	AAAAATAA	AAAATAAA	AAAATAAT	AAAATTA	AAATAAAA	AAATTTT	AATAACT
4	4	3	3	3	3	3	3	3	3	3
ACAAATA	AGAAATA	ATAAAAT	ATAAATA	ATAAATAA	GAAAATA	GTAAATA	TAAATAAA	TTAAATA	AAAAAAAA	AAAAAAT
3	3	3	3	3	3	3	3	3	2	2
AAAAATT	AAATAAAG	AAATAAAT	AAATAACT	AAATAATA	AAATTAA	AAATTAC	AAATTAT	AATAAAC	AATAAAG	AATAACA
2	2	2	2	2	2	2	2	2	2	2
AATAAGT	AATAATA	AATAATG	AATAATT	AATTATT	ATAAATT	ATAATTT	CAAAATA	CGAAATA	GAAATAAA	GTAAATAA
2	2	2	2	2	2	2	2	2	2	2
TAAAAATA	TAAAATAA	TAAATAAC	TAAATTA	TATAAAT	TGAAATA	TGAAATAA	TGTAAAT	TTAAAAT	TTAAATAA	TTGAAAT
2	2	2	2	2	2	2	2	2	2	2
TTTAAAT	AAAATAAAA	AAAATAAC	AAAATTT	AAATAAAAA	AAATAAAC	AAATAACA	AAATAAGT	AAATAATT	AATAAAAA	AATAAATA
2	1	1	1	1	1	1	1	1	1	1
AATAACC	AATAACG	AATAAGA	AATAAGG	AATAATC	AATTATA	ACAAATAA	AGAAATAA	ATAAAAG	ATAAAGT	ATAACTT
1	1	1	1	1	1	1	1	1	1	1
ATAATAA	ATATAAA	CAAAATAA	CAAATAAA	CAAATTA	CGAAATAA	GAAAATAA	GAAATAAT	GAAATTA	GGAAATA	GTAAAAT
1	1	1	1	1	1	1	1	1	1	1
TAAAAAA	TAAAATT	TAAATAAG	TAAATAAT	TAAATGT	TAGAAAT	TATAAAA	TCAAAAT	TCAAATA	TCAAATAA	TTAAAAA
1	1	1	1	1	1	1	1	1	1	1
TTATAAA	TTGTAAA	TTTAAAA	TTTTAAA							
1	1	1	1							

Figure 34: Motifs found together with the number of times they are detected as over-represented in the 3'UTRs with regards to non-3'UTRs.

There are probably some adjustments that could be made to this study, such as trying more values of k, or using different windows. Or maybe the reason for the CNN to believe that is the most important part of the sequence has nothing to do with k-mers, and it is looking for other structures that are not necessarily unsplit, which would mean that the k-mer analysis is of no use for us. But, whatever it is, we still do not know for sure.

5.5.1 Consensus motifs study

As we saw in *Figure 34*, adenine is predominant in the k-mers found in these 46 characters, and we thought they could be related to the *poly(A) signal*. The poly(A) signal is composed of hexamers (6-mers) identified, on average, 25 nucleotides upstream of the poly(A) site, at the end of 3'UTRS, as we previously saw in *Figure 13*. These hexamers are mostly composed of adenine bases, which is what made us think there was a relation between these two events.

Further studies were made in order to study this relationship. For that, we had to make use of the *Consensus motifs*. A *sequence motif* is a nucleotide sequence pattern that is widespread and has, or it is assume it has, a biological significance. A *consensus sequence motif*, or simply *consensus motif*, is, in turn, defined as the order of most frequent sequence motifs found at each position in a sequence.

By comparing the sequence motifs in multiple types of sequences, we can assess their resemblance to one another. With that, we can compare the similarity between what we found and the poly(A) signal. If they were too similar, it *could* mean that the most relevant thing in the 3'UTR is really the poly(A) signal, and that would mean that our network could not be properly learning patterns inside the 3'UTR sequences, but instead just classifying as positive or negative based on the manifestation of this poly(A) signal[16].

[16]Multiple experiments we made before were against this, such as the one in *section 5.4.5*, since that tells

For this study, we extracted the 7,8 and 9-mers from characters 21-36 of the reversed sequence, since that seemed like the section of the sequence with the most relevance according to the k-mer analysis. For each set of k-mers, a consensus motif were made. We did the same for the poly(A) signal: A consensus motif from its known sequences. Then, we compared them with this web tool: `http://meme-suite.org/tools/tomtom`.

From the results of this comparison, we learned that the only 9-mer was found to be of no use, since it was found on less than five sequences. As of the 7 and 8-mers, they were visually similar to the poly(A) motif, since they are all enriched with adenine, but their comparison did not have a significant p-value, which means we cannot prove they are equivalent.

From this, it seems that the poly-A elements are not as prevalent as we thought they would be, and it favours the argument of the network actually learning something complex in the structure of the sequence, and not depending on the poly(A) signal.

Conclusions

In this section, we found a discrepancy between LRP and the k-mers study. While the LRP verification seems to favour the former, that does not mean the later is of no use to us, as we will see in the following section.

We also confirmed that the network has to learn complex patterns inside the sequence, and does not only rely on the poly(A) signal hexamers.

6 Are CNNs really the only answer?

As a closure to this study, we wonder about the alternative to CNNs that could have been used instead. Note that this does not mean that choosing CNNs was a mistake, since the point of this study was to learn about the potential of CNNs for this particular task.

6.1 Evaluation with k-mers

Even though k-mers differed from the result of LRP, it does not mean that we can not evaluate genes using only them. We tried the following evaluation for our genes: Every sequence that contained one or more significant k-mers in its correspondent window would be predicted as positive, while every sequence that did not contain one or more of these subsequences would be predicted as negative. We only considered the k-mers contained in the last 50 characters of the sequence, since the k-mer analysis is a very resource-heavy technique, to the point that we could only analyze 5000 of the 30.000 sequences because of memory issues.

With this, we obtained an accuracy for the prediction of 0.888, which, considering we only analyzed the last 50 characters, is a relatively good result, although still inferior to the CNN, which obtains a result of 0.93 accuracy when only analyzing the same 50 characters, and a result of 0.95 accuracy when analyzing the whole sequence of 202 characters.

That, together with the fact that the k-mer analysis is way more costly than a CNN training, coupled with the fact that we needed to previously know what was the most relevant part of

us that even without this last part of the sequence we obtain an acceptable result, but we decided to try this anyway.

the sequence before doing the k-mer analysis (since we could not afford to analyse the whole sequence), we find ourselves before a technique that, not only is more costly than our CNN, obtains a worse result.

The only advantage it has versus the CNN is that this k-mer analysis is an **interpretable** technique, which means that we do not need to do anything else to know why we obtain our results, since we can look at the plot that shows the k-mer distribution (refer to *Figure 27* at *section 5.5 Results of the study*) and know exactly which are the most important windows, and the k-mer study also tells us which were these k-mers, in what windows they appeared, and in what proportion of positive and negative sequences they appear.

6.2 Evaluation with LSTM

An LSTM, or Long Short-Term Memory, is an artificial Recurrent Neural Network architecture designed to work specially well with **classification and processing** of **time series data**, such as **sequences**, for example. They seem to obtain good results with sequences of all types, including DNA sequences, as we can see on [28] and [29].

So, why did we not choose LSTMs for our project? As we said before, we had an interest on CNNs specifically, plus they are easier to use, specially with LRP, since LRP has to be adapted to LSTM layers, and that is not a trivial matter. More reasons for this choice are explained in the original project [1].

We received some help from Rafael Jordá Muñoz, another student who developed a LSTM for a project of his own, and tried to train it with our data. The jupyter notebook in which it is implemented can be found in Appendix 5. The training reached a validation accuracy of 0.96, which was even higher (although not by much) than the result of our CNN. The architecture of said LSTM network was the following:

- Bidirectional layer: LSTM, 128 memory units
- Dense layer: 1 node, sigmoid activation
- Additional information:
  - Learning rate: 0.001
  - Optimizer: Adam

It is a very simple model, but even so it achieved a very high accuracy value. In a future, it could be interesting to experiment with this kind of network.

In conclusion, there are many other options, apart from CNNs, but they all have its advantages and disadvantages.

6.3 Experiment without convolution

We also tried to train a Neural Network withut the convolutional layer. It would be something like this:

- Dense layer: 50 nodes, ReLU activation
- Dense layer: 1 node, sigmoid activation

This way we could learn how necessary is Convolution for this problem, and if a simple network would be able to obtain a good result.

Once trained the network without the convolution layer using our 3'UTR sequences without padding, we obtain a validation accuracy of approximately 0.93. This is only slightly worse than what we obtain with the convolution layer, so it probably means that this convolutional layer is not as important as we thought, at least for this problem in particular. More experiments would need to be made in order to confirm this.

We also calculated the mean relevances of 5000 for this model, and obtained the results of *Figure 35*:

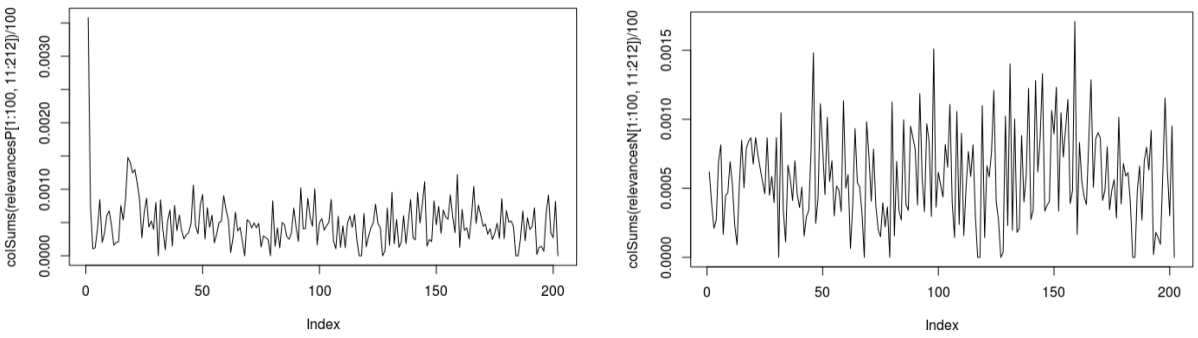

Figure 35: Left: Mean relevances for the positive sequences without convolution, Right: Mean relevances for the negative sequences without convolution.

The positive relevances are almost the same as those we obtained with the convolution layer, although we can see more variation since we are using less sequences. As for the negative relevances, its distribution is totally different from what we have seen until now, and we are not completely sure of what that means. Further studies would need to be made in order to reach a conclusion.

7 Conclusions and future ways

In this thesis we followed our previous investigation on differentiating various types of genes using CNNs and their DNA sequences. We started improving our understanding of the problem, analyzing the strong and weak points of our dataset, and investigating the "hard" and "easy" sequences, which we thought that had something to do with the possibility of some positive intellectual disability genes being disguised as negative genes, but did not arrive at a decisive conclusion.

We also theorized about the treatment of the sequence and whether we should reverse it or not, because of reasons we later discovered, thanks to Layer-wise Relevance Propagation, were not exactly the cause of it.

We finished with our first problem of 5'UTR sequences and changed to a second dataset of 3'UTRs to try our networks with a different problem, and explained the results of the model with LRP, verifying them with some additional experiments, arriving at the conclusion that the ending of the 3'UTR is the most relevant part of it, and the section the Neural Network looks the most when evaluating a sequence. We also extracted some new knowledge thanks to that, rectifying our misconception about the necessity of reversing the sequence, since it seems

our technique of reversing the sequence also lead to a selection of the last characters of the sequence, so they would not be discarded in long sequences.

Finally, we studied the k-mer composition of the sequences, and compared its results to LRP, resulting in a slight discrepancy from them both. Nonetheless, we could evaluate our genes using solely this k-mer analysis and, even though the results are worse than the CNN, they are still relatively acceptable. An LSTM network was also trained with this data, arriving at a result as good as the CNN, demonstrating it is a very viable alternative to our network.

A final consensus motif study confirmed to us that our CNN was learning complex patterns inside the sequences, and it was not over-relying on the poly(A) signal.

In the future, we could apply LRP to our original project, so we could see where the network is focusing when predicting. According to [20], LRP in a situation where the network does not perform as well as expected could shed some light on why that happens. We also could further investigate those questions to which we still have no answer, like why do certain subtypes of negative sequence of the new data perform worse, or the difference between hard and easy sequences for the original dataset.

We think our results were good, and, biologically speaking, very interesting. While the first problem could only reach a 70% accuracy for classifying intellectual disability genes — which, even though is not great, proves that our hypothesis of being able to classify genes associated with diseases is possible with just the sequence is true —, 3'UTR sequences let us predict new sequences with a 95% accuracy, also proving that these 3'UTR are not being classified only because of the poly(A) signal, because the motifs detected are not significantly similar to those of the poly(A), and the network is capable of classifying even without the ending of the 3'UTR sequence. In our opinion, this is worthy of further investigation.

8 References

[1] Martínez Legaz, J. (2019). Deep recognition of regulatory DNA sequences. *TFG of the University of Murcia*.

[2] Mitchell, T. M. (1997). Machine Learning, McGraw-Hill Higher Education. *New York.* https://dl.acm.org/doi/book/10.5555/541177

[3] Kaplan, A., & Haenlein, M. (2019). Siri, Siri, in my hand: Who's the fairest in the land? On the interpretations, illustrations, and implications of artificial intelligence. *Business Horizons, 62*(1) 15-25. https://doi.org/10.1016/j.bushor.2018.08.004

[4] ScienceDirect (2020). *Artificial Intelligence in Medicine* https://www.sciencedirect.com/journal/artificial-intelligence-in-medicine

[5] Yin, W., Kann, K., Yu, M., and Schütze, H. (2017). Comparative Study of CNN and RNN for Natural Language Processing. *ArXiv:1702.01923 [Cs].* http://arxiv.org/abs/1702.01923

[6] G. Montavon, W. Samek, and K.-R. Müller, (2018). Methods for interpreting and understanding deep neural networks». *Digit. Signal Process.,* vol. 73, pp. 1-15, feb. 2018. doi: 10.1016/j.dsp.2017.10.011.

[7] V. Agarwal and J. Shendure, (2018). Predicting mRNA abundance directly from genomic sequence using deep convolutional neural networks. *bioRxiv*, p. 416685, oct. 2018. doi: 10.1101/416685.

[8] M. Ancona, E. Ceolini, C. Öztireli, and M. Gross. (2018). Towards better understanding of gradient-based attribution methods for Deep Neural Networks *ArXiv171106104 Cs Stat*, mar. 2018. http://arxiv.org/abs/1711.06104.

[9] M. D. Zeiler and R. Fergus. (2013). Visualizing and Understanding Convolutional Networks. *ArXiv13112901 Cs*, nov. 2013. http://arxiv.org/abs/1311.2901.

[10] A. Shrikumar, P. Greenside, and A. Kundaje. (2019). Learning Important Features Through Propagating Activation Differences. *ArXiv170402685 Cs,* oct. 2019. http://arxiv.org/abs/1704.02685.

[11] A. Anguita-Ruiz, A. Segura-Delgado, R. Alcalá, C. M. Aguilera, and J. Alcalá-Fdez. (2020). eXplainable Artificial Intelligence (XAI) for the identification of biologically relevant gene expression patterns in longitudinal human studies, insights from obesity research. *PLOS Comput. Biol.*, vol. 16, n.o 4, p. e1007792, abr. 2020. doi: 10.1371/journal.pcbi.1007792.

[12] C. Rudin. (2019). Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nat. Mach. Intell.*, vol. 1, n.o 5, pp. 206-215, may 2019. doi: 10.1038/s42256-019-0048-x.

[13] Binder, A., Bach, S., Montavon, G., Müller, K.-R., and Samek, W. (2016). Layer-Wise Relevance Propagation for Deep Neural Network Architecture. *Information Science and Applications (ICISA) 2016*, Lecture Notes in Electrical EngineeringVolume: 376. http://iphome.hhi.de/samek/pdf/BinICISA16.pdf

[14] J. Grezmak, P. Wang, C. Sun, and R. X. Gao. (2019). Explainable Convolutional Neural Network for Gearbox Fault Diagnosis. *Procedia CIRP*, vol. 80, pp. 476-481, ene. 2019. doi: 10.1016/j.procir.2018.12.008.

[15] W. Samek, G. Montavon, A. Binder, S. Lapuschkin, and K.-R. Müller. (2016). Interpreting the Predictions of Complex ML Models by Layer-wise Relevance Propagation. *ArXiv161108191 Cs Stat,* nov. 2016. http://arxiv.org/abs/1611.08191.

[16] K. Simonyan, A. Vedaldi, and A. Zisserman. (2014). Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps. *ArXiv13126034 Cs,* abr. 2014. http://arxiv.org/abs/1312.6034.

[17] S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, and W. Samek. (2015). On Pixel-Wise Explanations for Non-Linear Classifier Decisions by Layer-Wise Relevance Propagation. *PLOS ONE,* vol. 10, n.o 7, p. e0130140, jul. 2015. doi: 10.1371/journal.pone.0130140.

[18] J. Zhang, Z. Lin, J. Brandt, X. Shen, and S. Sclaroff. (2016). Top-down Neural Attention by Excitation Backprop. *ArXiv160800507 Cs,* ago. 2016. http://arxiv.org/abs/1608.00507.

[19] M. T. Ribeiro, S. Singh, and C. Guestrin. (2016). "Why Should I Trust You?": Explaining the Predictions of Any Classifier. *ArXiv160204938 Cs Stat,* ago. 2016. http://arxiv.org/abs/1602.04938.

[20] M. Böhle, F. Eitel, M. Weygandt, and K. Ritter. (2019). Layer-Wise Relevance Propagation for Explaining Deep Neural Network Decisions in MRI-Based Alzheimer's Disease Classification. *Front. Aging Neurosci.,* vol. 11, 2019. doi: 10.3389/fnagi.2019.00194.

[21] R. Roscher, B. Bohn, M. F. Duarte, and J. Garcke. (2020). Explainable Machine Learning for Scientific Insights and Discoveries. *IEEE Access,* vol. 8, pp. 42200-42216, 2020. doi: 10.1109/ACCESS.2020.2976199.

[22] A. B. Tickle, R. Andrews, M. Golea, and J. Diederich. (1998). The truth will come to light: directions and challenges in extracting the knowledge embedded within trained artificial neural networks. *IEEE Trans. Neural Netw.,* vol. 9, n.o 6, pp. 1057-1068, nov. 1998. doi: 10.1109/72.728352.

[23] D. D. Nauck. (2003). Measuring interpretability in rule-based classification systems. *12th IEEE Int. Conf. Fuzzy Syst.* 2003 FUZZ 03, 2003 doi: 10.1109/FUZZ.2003.1209361.

[24] M. D. Zeiler, G. W. Taylor, and R. Fergus. (2011). Adaptive deconvolutional networks for mid and high level feature learning. *2011 International Conference on Computer Vision,* nov. 2011, pp. 2018-2025. doi: 10.1109/ICCV.2011.6126474.

[25] W. Samek, A. Binder, G. Montavon, S. Lapuschkin, and K.-R. Muller. (2017). Evaluating the Visualization of What a Deep Neural Network Has Learned. *IEEE Trans. Neural Netw. Learn. Syst.,* vol. 28, n.o 11, pp. 2660-2673, 2017. doi: 10.1109/TNNLS.2016.2599820.

[26] G. Montavon, S. Lapuschkin, A. Binder, W. Samek, and K.-R. Müller. (2017). Explaining nonlinear classification decisions with deep Taylor decomposition. *Pattern Recognit.,* vol. 65, pp. 211-222, may 2017. doi: 10.1016/j.patcog.2016.11.008.

[27] J. Lanchantin, R. Singh, B. Wang, and Y. Qi. (2017). Deep Motif Dashboard: Visualizing and understaing genomic sequences using Deep Neural Networks. *Biocomputing 2017, Kohala Coast, Hawaii, USA,* ene. 2017, pp. 254-265. doi: 10.1142/9789813207813_0025.

[28] J. M. Zhang and G. M. Kamath. (2016). Learning the Language of the Genome using RNNs. https://cs224d.stanford.edu/reports/jessesz.pdf

[29] Q. Liu, L. Fang, G. Yu, D. Wang, C.-L. Xiao, and K. Wang. (2019). Detection of DNA base modifications by deep recurrent neural network on Oxford Nanopore sequencing data. *Nature Communications,* vol. 10, n.o 1, Art. n.o 1, jun. 2019. doi: 10.1038/s41467-019-10168-2.

9 Appendix

9.1 Appendix 1 – GitHub projects

Two GitHub repositories were created to host this project. The first repository contains all the Markdown scripts and the instructions to replicate everything discussed here. It can be found at `https://github.com/JaimeMLegaz/recognition-regulatory-sequences-thesis`. This is the one that contains all the code necessary to do all the work, from obtaining the DNA sequences to train the network and obtain the same results we obtained, and the scripts to obtain the relevances. Every script referenced in this project can be found here.

This Github project will contain the updated scripts and hierarchy. The possibility of some discrepancy between these appendixes and the GitHub repository might happen at some point in the future. Since the GitHub project will always be updated, it is recommended to examine it instead of relying on these appendixes.

In the following sections, we will review all the scripts referenced in this document, and their location in the directories. However, keep in mind these things:

- There are more scripts that were used than those referenced here. Those that we did not mention were either incomplete experiments or extremely similar experiments which did not provide any relevant information.
- In every directory of the repository will be a file called "FilesExplained". This file will explain the purpose of each and every script of that directory.
- Most of the reviews of the scripts contained in this document have been simplified, since some scripts have many more tests done that what we show here. They were not shown in this document since most of the time they only provide context or redundant information. Nevertheless, it is highly recommended to check the markdowns of the scripts mentioned, since they could help shed some light on some topics. Specially, the markdowns for the LRP experiments contain lots of plots that could be helpful to understand them.

The second repository contains only the R functions developed for the treatment of sequences. It can be used by typing into the R console the following commands:

```
devtools::install_github("JaimeMLegaz/setup-sequences")
library(SetupSequences)
```

The repository can be also found in `https://github.com/JaimeMLegaz/setup-sequences`, but it does not contain any kind of useful information for this project.

9.2 Appendix 2 - Scripts for the training and evaluation of the Network

In this Appendix, we will list all scripts used for the training and evaluation of both problems, 3'UTR and 5'UTR. These scripts have multiple purposes, from the collection and treatment of sequences, to the analysis made to improve the original experiment.

The scripts we will review are in Markdown format. Markdown is a language which lets us create scripts that generate an html output of it. This way, we can work comfortably in R while providing a document that the reader will find clearer than a simple script.

We will divide this appendix into subsections, each dedicated to a specific section of the methodology, and will review all the scripts needed for that subsection, detailing what does it do and the files it creates.

9.2.1 Original experiment

The original experiment, the 5'UTR one, covered in [1], uses four markdowns mainly:

- UTR_ObtainSequences: Obtain the sequences used for the UTR experiment
- UTR_TreatingSequences: Treats the sequences previously obtained, reversing, padding/cutting and encoding them
- UTR_TrainNetwork: Trains the network with these sequences
- UTR_FinalEvaluation: Combines the three scripts, and evaluates all the sequences

These four files can be found at the root of the hierarchy. More information about the first three files and how they work can be found in [1]. The UTR_FinalEvaluation markdown generates two files:

- finalEvaluationAllGenes, which contains, for each sequence:
  - Its output when evaluated for each model
  - Which models it was used to train
  - Which models it was used for evaluation
  - Its disease association
- stats, which contains the information of *Table 6*, shown in *section 4.6.1 Detailed evaluation results*.

9.2.2 Hard and easy sequences

For *section 4.6.2 Hard and easy sequences* we developed two new markdowns:

- firstresults, which extracts the hard and easy sequences from the full set, and saves them in two files: goodseqs.tsv and badseqs.tsv.
- SequencesBasics, which uses the sequences from firstresults to do the PCA analysis of the alphabet frequency of said sequences.

These two files are found inside the "Phase2_PCA_Study" directory.

9.2.3 Study of the Sequence treatment and length

In this section we list the markdowns used for *section 4.6.3 Study of the Sequence Treatment* and *section 4.6.4 Optimal length of the Sequences*:

- UTR_FinalEvaluation_XXXABC: This markdown is the same as the UTR_FinalEvaluation previously described in this Appendix, only changing the treatment of the sequences. This covers the XXXABC experiment, in which the sequence is not reversed and the padding is at the start instead of at the end.
  - The previously described UTR_FinalEvaluation file covers the CBAXXX experiment.

  - As for the XXXCBA and ABCXXX experiments, we used the same two markdowns, and just changed the padding function to add the padding to the right or to the left, so these two experiments do not have a dedicated markdown.
- goodAndBadSeqs_XXXABC: this is similar to the firstresults scripts from last section. It gets the easy and hard sequences of the XXXABC experiment, and saves them in two files: goodseqs_XXXABC.tsv and badseqs_XXXABC.tsv.
- UTR_StatsVisualization: This studies the length of the sequences and their padding. Lets us obtain the information for *Figures 13 to 15*. It also compares the hard and easy sequences from both experiments XXXABC and CBAXXX, obtaining the information for *Table 7*.

These three files are located inside the "Phase4" directory.

9.2.4 Recovery of sequences in new dataset

We used Ensembl annotation (v94 GTF) to extract the genomic coordinates of different functional classes. For all functional classes in our training dataset, firstly, we selected high confidence annotations at the transcript level with transcript support level (TSL) = 1. Secondly, we collapsed and combined multiple transcripts associated with a single gene to make a consensus "meta-transcript" per gene. Thirdly, we extracted exons with width ¿= 40 nt from these meta-transcripts. Finally, for each exon region, we extracted the underlying DNA sequence from the human genome (hg38) to serve as learning examples.

To capture regions of 3'UTR exons, 5'UTR exons and internal coding exons (ICE), transcripts from protein-coding genes were selected. For ICE examples, transcripts with at least three coding exons were further selected (as transcripts with less than three exons would not contain an internal exon) and their first and last coding exons were removed to capture ICEs. To capture lncRNA exons, we selected annotations from the GTF with the following gene biotypes: "non_coding", "3prime_overlapping_ncRNA", "antisense", "lincRNA", "sense_intronic", "sense_overlapping" and "macro_lncRNA". For ncRNA exons, annotations with the following gene biotypes were selected: "miRNA", "misc_RNA", "rRNA", "snRNA", "snoRNA" and "vaultRNA". For pseudogene exons, annotations with the following gene biotypes were selected: "pseudogene", "processed_pseudogene", "unprocessed_pseudogene", "transcribed_processed_pseudogene", "transcribed_unitary_pseudogene", "transcribed_unprocessed-_pseudogene", "translated_processed_pseudogene", "unitary_pseudogene", "unprocessed_pseudogene", "TR_V_pseudogene", "TR_J_pseudogene", "rRNA_pseudogene", "polymorphic_pseudogene", "IG_V_pseudogene", "IG_pseudogene", "IG_J_pseudogene" and "IG_C_pseudogene"

Overall, for the positive examples, we captured 17,719 3'UTRs; while for the negative dataset, we captured a total of 162,249 Non-3'UTRs, which comprised of 21,798 5'UTRs, 130,768 ICEs, 3,718 lncRNAs, 3,819 ncRNAs and 2,146 pseudogenes.

9.2.5 Results of the new experiment

The markdowns used for the training and evaluation of the network with this new dataset of 3'UTR sequences are almost the same as the markdowns for the old experiment, and they are called the same, with the only change that the files' names start with "3UTR" instead of "UTR". These files are found in "3UTR Experiment Relevances/Scripts/".

As a note, there is no "3UTR_Treating_Sequences" since we use exactly the same treatment as the 5'UTR experiment, so we can use the same file.

As for the file that obtains the data for *Table 8*, it is called "3UTR_stats".

The data needed for the analysis of every subtype of negative sequence, needed for *Table 9*, can also be found in file "3UTR_FinalEvaluation", saved as a file called "typeStats".

9.3 Appendix 3 - Scripts for Layer-wise Relevance Propagation

In this third Appendix, we will review the scripts developed for the section related to the application of the Layer-wise Relevance Propagation algorithm to our problem.

There are mainly two scripts used for this:

- LRP_detailed: Where we repeat the whole process of obtaining and treating the sequences, and then training a network, and then we apply LRP to the sequences and visualize the results.
- LRP_detailed_nopadding: The same as before, but for the second LRP experiment, with sequences of 202 characters of length and no padding.

These are, like the files before, located in "3UTR Experiment Relevances/Scripts/".

But these Markdowns do not really calculate the relevances of all our sequences. They only implement the algorithm, add a pseudocode, and then visualize the relevances calculated by other scripts. The scripts (and this time they are R scripts, not markdowns, so they are not as easily interpretable) that really calculate the relevances are:

- LRPscript_multiple_models_nopadding: It calculates the relevances of the five models.
- LRPscript_nopadding_positive: It calculates the relevances of all positive sequences for just the first model.
- LRPscript_nopadding_negative: It calculates the relevances of all negative sequences for just the first model.

These are the scripts that generate all the relevances, and should be executed with caution, as LRP is a very resource-heavy algorithm, and this process of obtaining the relevances lasted **days** in the *Prodia* server of the University of Murcia. These files are found inside the "LRP_Scripts" directory, which is in the same directory as the previous files.

9.4 Appendix 4 - K-mers study

We only needed one script for the realization of this study. The file is called "kmer.Rmd", and it loads the positive and negative sequences and relevances to analyze them. The sequences are used for the k-mer study, and the relevances for the clustering. It is found in the "3UTR Experiment Relevances/Scripts/LastExperiments/" folder.

9.5 Appendix 5 - LSTM

The jupyter notebook developed by Rafael Jordá Muñoz is a file called "LSTM.ipynb", also inside the "3UTR Experiment Relevances/Scripts/LastExperiments/" directory.