University of Murcia

Deep recognition of regulatory DNA sequences

Author: Jaime Martínez Legaz
Tutor: Juan Antonio Botía Blaya

September 2018 – Degree in Computing Engineering

Contents

1 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".

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 Grado 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 05 de Septimebre de 2019

[illegible]

Fdo.: Jaime Martínez Legaz Autor del proyecto

2 Abstract

Machine Learning (ML) is one of the branches of the Artificial Intelligence (AI) that, even though it has existed for decades, is raising in popularity now thanks to, among many other things, to the bigger computational power available to us. Among the many applications it has, we will highlight the medicine as the one we will be focusing on, more specifically on the field of genetics.

In this project we will study genes associated with intellectual disability, working with ML techniques with the purpose of studying if there is something in the regulatory sequences of said genes that makes them different from the rest. The regulatory elements we will work with are the promoters of the genes and the 5'UTR regions.

That implies we will work with DNA sequences, so we will need an ML model that is designed to work with sequences. One option would be Recurrent neural networks (RNNs), which, while they promise good results, they are also computationally costly.

The second option, and the one we finally chose, are Convolutional neural networks (CNNs). Even though they were originally designed to work with images, they are also totally valid for sequence analysis. While they are not as potent as RNNs, they promise more than acceptable results with a relatively low computational cost. A series of previous studies working in the field of genome sequences that also opt to work with CNNs supports our decision.

We performed three main experiments: The first tried to tell apart sequences corresponding to promoters to those corresponding to 5'UTR, with the only purpose of find out if our networks were suited to learn from our sequences. This experiment had a satisfactory result of approximately 0.7 accuracy.

The second and third experiments were the important of the three for our goal. The second experiment tried to figure out if there is something in the promoters that make genes associated with intellectual disability different from the rest. The third one tried the same thing but working with 5'UTR sequences instead of the promoters.

The second experiment was not successful, but maybe that was caused because of a mistake in the collection of the sequences we worked with. The third experiment, with an accuracy of approximately 0.8 validation accuracy, obtained the satisfactory result we hoped for, proving that said region of the human genome can be used to distinguish associations with intellectual disability.

3 Resumen

El aprendizaje máquina o machine learning (ML) es una de las ramas de la Inteligencia Artificial (IA) que, si bien lleva existiendo de hace décadas, está cogiendo más fuerza en los últimos años debido, entre otras cosas, a la mayor potencia de computación de la que disponemos hoy día. Entre las muchas aplicaciones que tiene, destacaremos la medicina como aquella en la que nos enfocaremos, más concretamente en el campo de la genética.

En este proyecto estudiaremos genes asociados con la discapacidad intelectual, trabajando con técnicas de ML con el objetivo de estudiar si hay algo en las secuencias reguladores de dichos genes que los haga diferentes del resto. Los elementos reguladores en los que nos centraremos son los promotores de los genes y las regiones 5'UTR.

Esto implica que trabajaríamos con secuencias de ADN, por lo que necesitaríamos un modelo de ML que esté diseñado para trabajar con secuencias. Una opción son las Redes neuronales recurrentes (RNNs), las cuales, si bien prometen buenos resultados, también son bastante costosas computacionalmente.

La segunda opción, y por la que nos decantamos, son las Redes de Convolución (CNNs). Pese a estar originalmente diseñadas para trabajar con imágenes, también son totalmente válidas para el análisis de secuencias. Si bien no son tan potentes como las RNNs, prometen resultados más que aceptables con un coste computacional relativamente bajo. Una serie de estudios anteriores en el campo de las secuencias genómicas que también optan por trabajar con CNNs nos apoyan a la hora de tomar esta decisión.

Realizamos tres experimentos principalmente: Uno que intenta distinguir secuencias correspondientes a promotores de las correspondientes a 5'UTR, con el único propósito de averiguar si nuestras redes poseen la capacidad de aprender con nuestras secuencias, el cual obtuvo un resultado satisfactorio de aproximadamente 0.7 de accuracy.

El segundo y tercer experimento eran los importantes para nuestro objetivo. El segundo intentaba averiguar si hay algo en los promotores que distinga a los genes asociados con discapacidad intelectual del resto. El tercero intentaba lo mismo pero con las secuencias 5'UTR en lugar de con las de promotores.

El segundo experimento no tuvo éxito, pero quizá estuvo provocado por una mala obtención de las secuencias con las que trabajamos. El tercer experimento, con un accuracy de aproximadamente 0.8 en la validación, sí obtuvo el resultado satisfactorio que esperábamos, probando que dicha región del genoma puede ser usada para distinguir asociaciones con la discapacidad intelectual.

4 Introduction

The Oxford Dictionary defines Artificial Intelligence (AI) as "an area of study concerned with making computers copy intelligent human behaviour". Reading this, a person unfamiliarized with the reality of AI tends to think of computers that are able to "feel" and "think" as we humans do, capable of reasoning or expressing feelings. The media, specially with movies like "Ex Machina" or "I, robot", make people believe that the reality of the AI is little less than an Isaac Asimov book.

Andreas Kaplan and Michael Haenlein propose another definition for AI: "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 flexible adaptation" [1]. Here we see clearer that the main ability a machine has to have to be considered "intelligent" simply is to be able to interpret, analyze and work with external data.

Machine learning (ML) is a branch of the AI, the one in which we will work, and is a relatively new field of investigation. In less than a century, after rising and falling in popularity many times, today we live in a time when it is more important than ever in areas like robotics, finance and, the most relevant for us right now, medicine.

Today we find lots of projects in which ML intervene in this field of medicine. Specifically, we will be focusing in genetics and monogenetic disorders, like intellectual disabilities.

Intellectual disability is, in fact, the disease we will be working with. It is a disability characterized by significant limitations both in intellectual functioning (reasoning, learning, problem solving) and in adaptive behavior. This disease is sometimes caused by lack of oxygen at birth or other physical causes, but in most cases its roots are genetics, either because of alterations in the number of chromosomes, or abnormal genetic heritage, to name a few. The causes of a 30% to 50% of the cases are still unknown.

Therefore, there are two main fields we will be working in: Machine learning, using more specifically Convolutional Neural Networks, and genetics. We will briefly introduce those two areas:

4.1 CNNs

The main element we will work with will be the Convolutional Neural Networks, also called CNNs or ConvNets.

CNNs are a type of Neural Network, which are networks composed of neurons capable of learning from external data, and help us classify and predict from that data. More about them will be explained later.

CNNs are designed to work with images. capturing features that helps the network identify the image. Although that is their main purpose, they are also suited to work with sequences, and that is what we are going to do.

They usually are connected to a Multi-Layered Perceptron – the most basic of the Neural Network architectures, composed of one or more layers of neurons, fully connected – because CNNs are not suited to learn by themselves, only to capture features, so they need help of another structure for that task.

An alternative to CNNs would be Recurrent Neural Networks (RNNs), and Long Short-Term Memory (LSTMs, a type of RNN). They are designed to work with temporal sequences, like our DNA sequences. LSTMs and CNNs have been compared multiple times in various lines of work. Some of these comparisons can be seen in [2].

This is not, however, a radical approach. Many experiments in the field of genetics choose CNNs over RNNs/LSTMs. We will describe CNNs and some experiments that work with them in more detail later.

4.2 Genetics

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 **Promoters** and **5'UTR**.

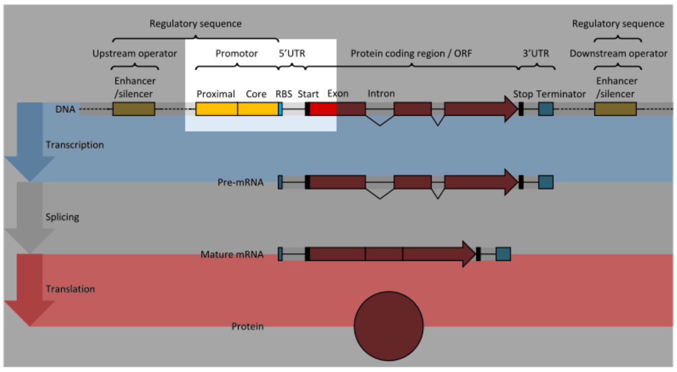


Figure 1: Placement of Promoter and 5'UTR. Red sections compose the gene. Orange sections compose the regulatory elements that control the transcription of the gene [1]

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.

A promoter is a region of the DNA that marks the starting point of the transcription, that is, the start of the gene, since it is what will be transcribed. Promoters contain a variety of regulatory elements required for accurate and efficient initiation of the transcription.

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 intervenes in the **translation** 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**. We will be working with the first kind. This untranslated region is involved in many regulatory aspects of gene expressions in eukaryotic organisms.

Although lots of interesting features can be found in these regulatory elements, we will **only** focus in their sequences and the hidden patterns contained in them. Even though promoters and 5'UTR sequences have different features – or at least we will try to prove that in one of our experiments –, both look the same when viewed by a human:

ATTCCTGCCTGGGAGGTTGTGGAAGAAGGAAG

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

This, for example, is one of the 5'UTR sequences of the ITGA2B gene. Our promoter sequences will be of the same structure – strings composed of As, Cs, Ts and Gs. The main difference lies in the length of the sequence: Promoters will have 250 characters of length and 5'UTR sequences will have various lengths. We will talk about this in the Methodology section.

Now that all the concepts we will work with have been introduced, we can finally present the main question we will be trying to answer in this thesis: Do regulatory elements present specific properties that make them responsible of certain phenotypes, like intellectual disability?

5 Objectives

Main objective: Develop a methodology that allows us to investigate sequence level differences based on elements of the human genome. Are there relevant differences between promoters or 5'UTR regions of different groups of genes?

Sub objectives:

- Mechanisms for recovering sequences of interest. We will develop some tools that help us obtain the promoter and 5'UTR sequences for all genes we will work with.
- Mechanisms for coding sequences before their insertion in the networks. Since our deep learning models cannot work with untreated sequences – since they need sequences of numbers of certain fixed length, and the sequences we will obtain are strings composed of characters – we need to encode them in a valid format.
- Build CNN architectures for sequence treatment. They will do the learning and the prediction and classification.
- Develop CNN evaluation mechanisms and determination of typical levels of accuracy, so that we have a general idea of how good are our results.

6 State of the Art

In this project, we will work primarily with Convolutional Neural Networks (CNNs). But before introducing those, we will have a look at the concept of "Neural Network".

6.1 Neural Networks

A Neural Network, as the name says, can be viewed as a network composed of neurons. But, what are those neurons, and how are they connected?

A neuron, also known as "perceptron", is basically a "Node that holds a number". They are arranged in multiple layers: one input layer, one output layer, and one or more "hidden layers". These layers are known as "Fully connected layers", since each node of each layer is connected to every node in the next layer. An example of a simple network can be found in Figure 2.

Our input goes into the input layer. Each input node has to receive a numerical value between 0 and 1, so the structure of the data used will oftentimes have to be modified and reshaped so it meets that condition.

[2]Image extracted from https://medium.com/pankajmathur/a-simple-multilayer-perceptron-with-tensorflow-3effe7bf3466

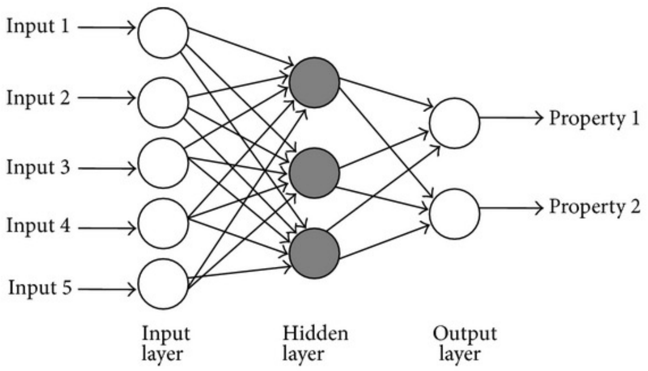


Figure 2: An example of a Multi-Layered Perceptron, a Neural Network composed of Fully connected layers. They are the most basic of the Neural Networks. [2]

Our result can be extracted from the Output layer. Each one of the nodes composing that layer contains one value also from 0 to 1. Depending on the purpose of the network, it can mean a raw value or it can represent a classification. In this project, we only work with **classification networks**. That means that the value itself will not be used, but it will tell us which class fits that input the best.

And what happens inside the Hidden Layers? We will not go into much detail here, but basically neurons do activate depending on the previous layer's nodes and the connections between those layers. These layers are where the "learning" happens. More information about the learning (and more) in [3].

6.2 CNNs

CNNs are a more specialized version of those Neural Networks. Usually they are used in image recognition problems, but they are not limited to that. In fact, as we said before, we will use them to recognize patterns in **sequences**. We will talk about convolution in image recognition and in sequence classification later.

CNNs retain most of the things we have already mentioned: Layers, Nodes and Connections. But they introduce a few new elements: Convolutional Layers, Filters and Pooling.

Convolutional layers work with filters instead of nodes. A filter is, in essence, a matrix composed of numerical values. It is smaller than the sequence (or image) we are working with, and its purpose is to, by a lineal combination of the filter's values and the area of the sequence affected by the filter, obtain a new, modified sequence or image *(Figure 3)*. The filter, as mentioned before, affects a particular area of the source, and runs through the whole source.

The output of the convolution will then be passed through an **activation function**, which modifies the value according to a curve. Two of the most popular functions are **ReLU** and **Sigmoid**. We can see their curve in Figure 4.

The reason why we don't work with a Multi-Layered Perceptron for this is because of the amount of work that would require. For example, sequence of 1000 characters means 1000 input values. Using only one layer of 100 nodes would equal to 100.000 parameters. And we have a lot of those sequences.

But filters alone cannot solve this. Each filter generates a "feature map", which is in essence a matrix of values. It also generates a set of weights, and all these values are inputs the network has to learn and train with. For that, we have the Pooling layers.

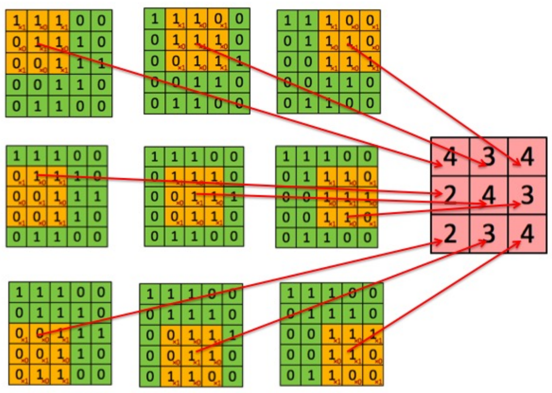


Figure 3: Example of a filter (orange) applied to a source image (green), generating an output (red) [3]

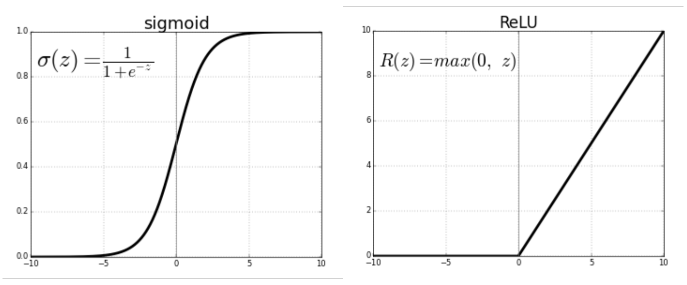


Figure 4: Sigmoid and ReLU functions. Both of them have a minimum value of 0 and a maximum value of 1, but they approximate the result in very different ways [4]

Pooling layers act as a way of down-sampling the output images, creating a lower resolution version of an input signal that still contains the large or important structural elements, without the fine detail that may not be as useful to the task. That achieves two goals:

– Reduces the volume of data that goes from a layer to the next.
– A limitation of the feature map output of convolutional layers is that they record the precise position of features in the input. This means that small movements in the position of the feature in the input image will result in a different feature map. Down-sampling the features map means we can work with a more generic map that serves the task good enough.

There are two types of pooling: Max and Average. Both of them divide the source into multiple patches, and then obtain a single value from that patch. Max pooling calculates the maximum value of that patch, while Average pooling calculates the average value. There is an example of Max Pooling in *Figure 5*.

After these transformations, the resulting outputs become the inputs of the Fully Connected sections, thanks to a layer called **Flatten layer**, which is in charge of transforming those outputs into the required structure.

[3]Image extracted from the notes of our Machine Learning subject
[4]Image extracted from https://blog.paperspace.com/vanishing-gradients-activation-function/

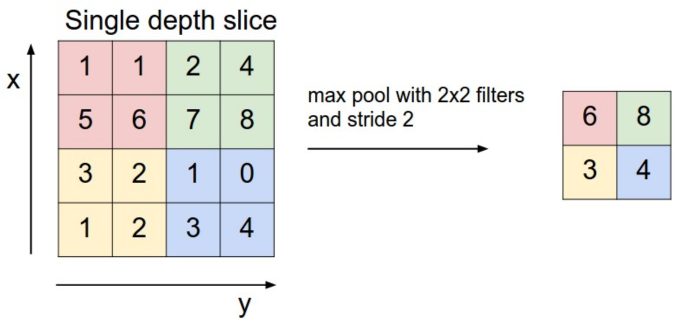


Figure 5: Example of Max Pooling, with 2x2 patches. Each color represents a different patch, and each patch is then reduced to its maximum value [5]

6.3 Typical architecture of a CNN

Lots and lots of different networks have been built in the history of Machine Learning. Differences between them range from a simple change in the number of nodes in a layer, to a whole different looping structure.

We will classify all these networks in two main groups: The “Classic” and the “Modern” architectures. These do not necessarily mean they are “old” and “new”, temporarily speaking, but we can say that most of the classic architectures are similar to the most basic networks – simply stacking a couple of layers –, and the modern usually introduce some kind of intricate design – usually involving the use of a repeatable unit, used multiple times.

We will take a look at some known examples of architectures, both classic and modern:

6.3.1 Classic

LeNet-5

LeNet-5 was developed in 1998 with the purpose of identificating 32x32 grey scale images, depicting handwritten digits. This is one of the most obvious examples of a classic CNN, composed of convolution layers followed by sub-sampling layers, repeating until desired. The output of the convolution block is then fed to a Multi-Layered Perceptron, which in turn produces the final output. The sub-sampling layers work with average pooling. LeNet-5 is nowadays pretty much outdated in every way. More on the subject can be found in the original paper [4].

This structure is similar to the one we will be working with, but because of different reasons. Sequence networks tend to be simpler than those that work with image recognition. More on that later.

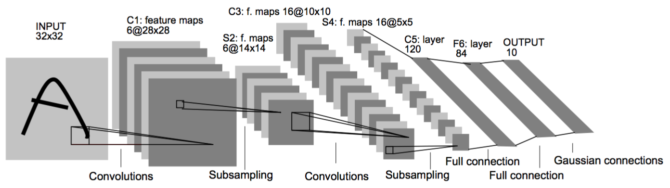


Figure 6: Structure of the LeNet-5 architecture. [6]

[5]Image extracted from http://cs231n.github.io/convolutional-networks/#pool
[6]Image extracted from the original paper

VGG-16

The VGG-16 dates from 2014, and it is basically a deeper version of the concept of the LeNet-5. It works with bigger images, and a very big difference can be seen between LeNet-5 and this example in terms of parameters: The former works with 60.000 parameters, and the latter needs around 138 million parameters.

It differs from previous works in the field, reducing the receptive fields' typical size of 11x11 with stride 4, or 7x7 with stride 2, to smaller fields of 3x3 with stride 1 [5].

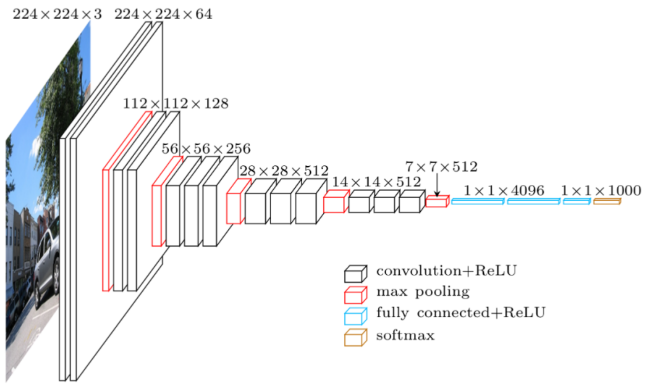


Figure 7: Structure of the VGG-16 architecture. [7]

6.3.2 Modern

GoogLeNet

Also known as **Inception**, GoogLeNet was a network which won Google the ILSVRC 2014 competition for visual classification and detection.

Its design is much more complex than the networks mentioned before, but performance-wise that complexity is really worth it. Compared to VGG-16 138 million parameters, GoogLeNet only uses 5 million parameters, and the results it obtains are much better

The idea behind this network have been mentioned before: it is based on a repeatable unit which is used throughout the network. In this case, the unit is called "Inception cell", and it is composed of multiple convolutions with different scales, whose results are aggregated [6].

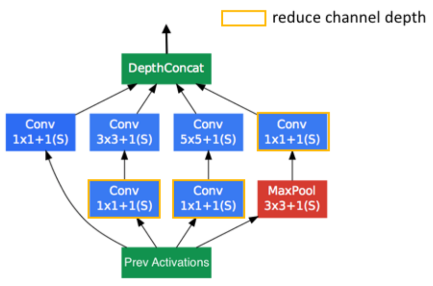


Figure 8: Design of an Inception cell. [8]

The whole network would look like this:

[7]Image extracted from https://www.jeremyjordan.me/convnet-architectures/
[8]Image extracted from https://www.jeremyjordan.me/convnet-architectures/

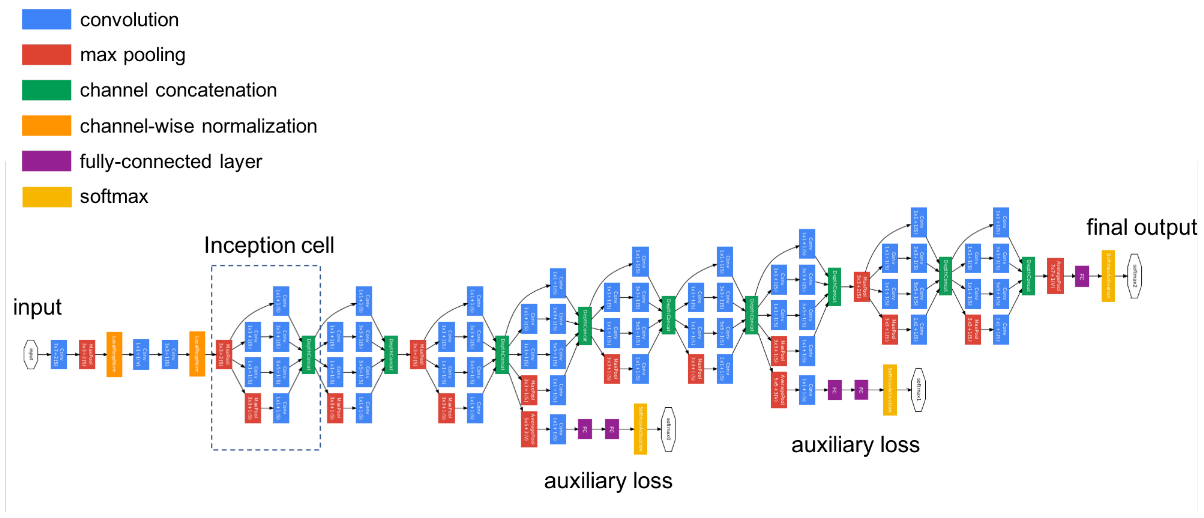


Figure 9: The GoogLeNet architecture. [9]

By adding auxiliary classifiers connected to these intermediate layers, they expected to encourage discrimination in the lower stages in the classifier, increasing the gradient signal that gets propagated back, and provide additional regularization.

In a follow-up paper [7] they introduced more efficient versions of the Inception cell, taking advantage of the fact that you can emulate big filters through combining multiple, smaller filters. This setup reduces the parameter count by sharing the weights between adjacent tiles.

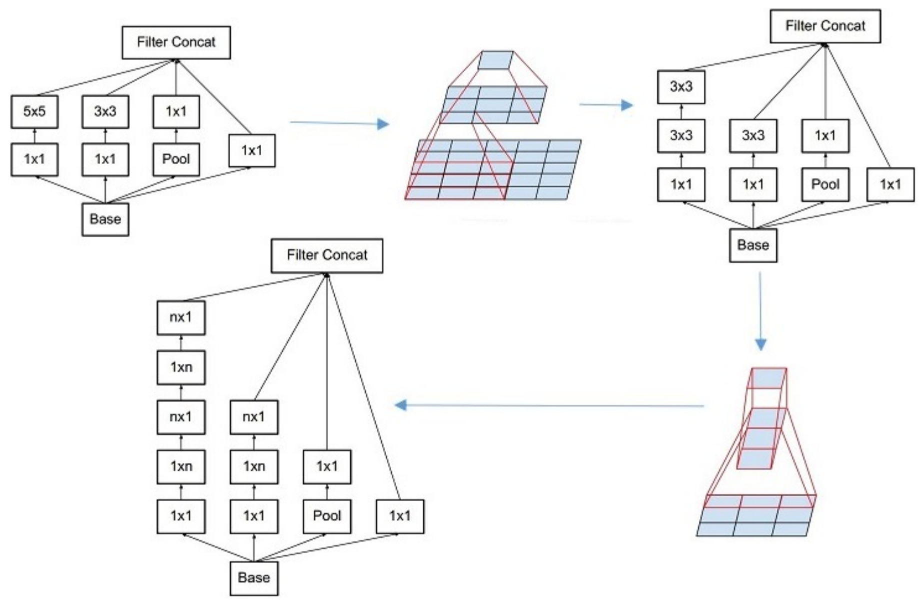


Figure 10: Evolution of the Inception cell, thanks to the combination of multiple small filters. [10]

DenseNet

This network works with a unit called a **dense block**, which is a group of layers in which each layer's feature map is connected to **every successive layer**. Concatenating feature-maps learned by different layers increases variation in the input of subsequent layers and improves efficiency [8].

[9]Image extracted from https://www.jeremyjordan.me/convnet-architectures/
[10]Multiple images extracted from the follow-up paper
[11]Image extracted from the original paper

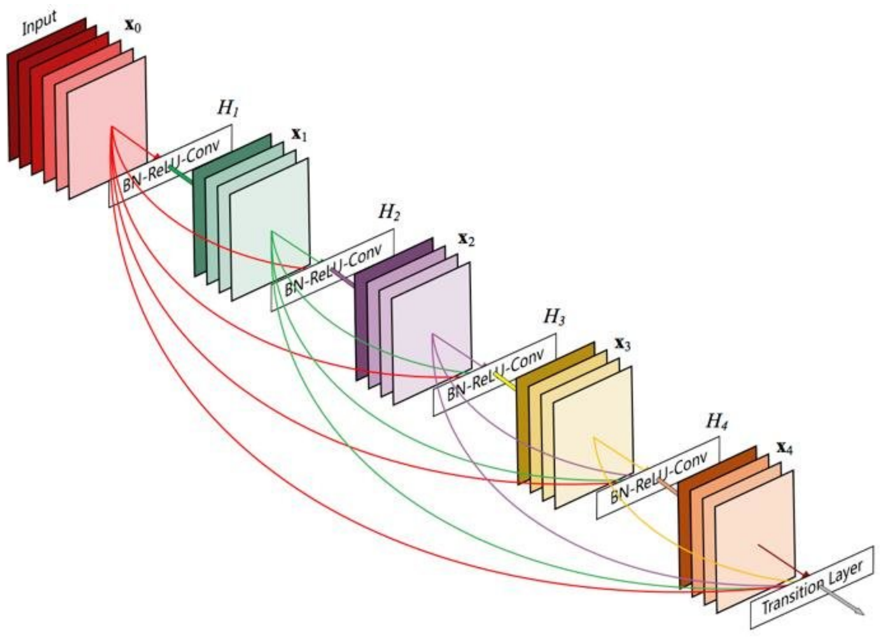


Figure 11: Example of a dense block. Every layer receives every past feature map as input. [11]

DenseNet layers are very narrow (e.g., 12 filters per layer), adding only a small set of feature-maps to the "collective knowledge" of the network and keep the remaining feature maps unchanged. The final classifier makes a decision based on all feature-maps in the network.

DenseNet's structure is based on the ResNet (which is covered in [9]), simply changing the repeated unit, in this case being the dense blocks.

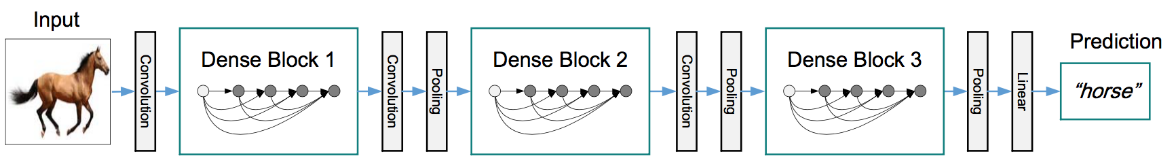


Figure 12: Structure of the DenseNet architecture. We can see how it is composed of a repetition of this Dense Block. [12]

As we can see, even though there are plenty of different networks, they can clearly be classified in these two groups. Image classification is a much heavier task than sequence classification, as we will see in the following section.

For that reason, there is not much from these models that we can use. Any try of replicating such complex structures for our task would result in heavily overfitted models and enormous training times.

6.4 Architecture of a CNN in the context of sequences

All the examples we have already seen have been developed with images in mind, not sequences. We will work with sequences, not images, so we will have a look at some real examples to see how differently they are built in the context of sequences:

Basenji

[12]Image extracted from the original paper

Successor to Basset, this is a network that tries to identify promoters and distal regulatory elements and synthesize their content to make effective gene expression predictions [10]. As we can see, their study and ours work with similar sequences. Since they also use CNNs for their models, it supports our decision of working with CNNs instead of LSTMs.

Basenji works with a one-hot encoded version of the DNA sequences, since we can't feed our networks with characters or strings. One-hot encoding is one typical way of representing discrete input variables in machine learning. It works by transforming each possible value of the variable in a set of "0" with only one "1", being the position of that "1" what represents the value of the variable. A=0001, C=0010, G=0100, T=1000 could be an example of this one-hot encoding. Since DNA sequences can be made of four different letters (A, C, G and T), the resulting sequence will be four times larger than the original.

After being encoded, those sequences are introduced in the Basenji network, composed at first of blocks of convolution layers and max poolings, then seven dilated convolution layers.

Dilated convolutions are composed of convolution filters with gaps whose size increases by a factor of two in each layer, enabling the receptive field width to increase exponentially. They are densely connected, which means that each layer takes all previous layers as input, as opposed to taking only the preceding layer, kind of similar to DenseNet. After that, the final prediction is obtained.

We can see that it does not differ much from the classic architectures of the image recognition networks. Since they work with similar sequences to ours, part of their design should not be much different from ours.

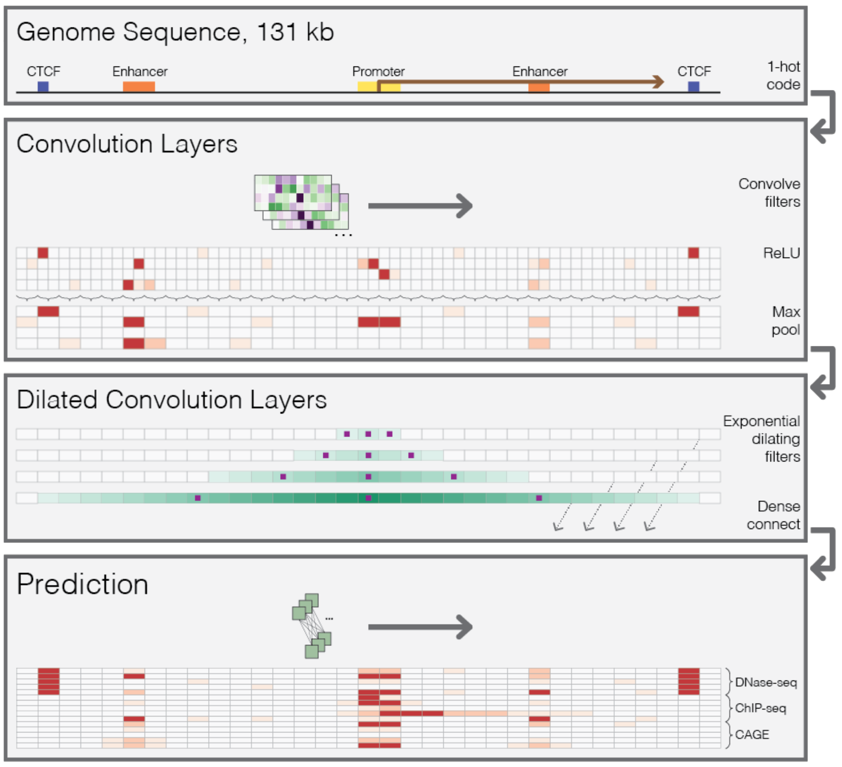


Figure 13: Structure of Basenji. [13]

As we can see in *Figure 13*, first of all they obtain part of the promoter and the enhancer, and one-hot encode that data. Then they feed that data to the model, composed of these convolution layers and dilated convolution layers, and lastly it passes trough the dense layers to make the prediction.

[13]Image extracted from the original paper

Our work and theirs does not focus on the same objective, so we can compare the preprocessing and postprocessing but not the structure of the model, since they will try to learn different things.

As we will see later in the corresponding section, we will also one-hot encode our sequences and then pass them to some convolution layers. After that, our data will also go to a dense layer to make a prediction. Thus, their and our work only differ in the structure of the model itself.

Optimus 5-Prime

In this recent article [11] the authors look for a way of engineer new 5'UTR sequences that accurately direct specified levels of ribosome loading. In other words, they want to be capable of tuning sequences for optimal protein expression.

To achieve that, they first created a library with 280.000 gene sequences consisting of a random 5'UTR and a constant region. The model itself, the Optimus 5-Prime, had the goal of quantitatively capture the relationship between 5'UTR sequences and their associated MRLs (Mean Ribosome Load). And it achieved that with an accuracy of 0.93 in the test set, compared to the 0.66 of other linear models.

The Optimus 5-Prime works feeding a one-hot encoded 5'UTR sequence into a CNN composed of three convolution layers and a fully connected layer to produce a linear output predicting MRL *(Figure 16)*. It is a **very simple network**, and to achieve a 0.93 accuracy value on a sequence based problem says good things about the capacities of the CNNs in this field.

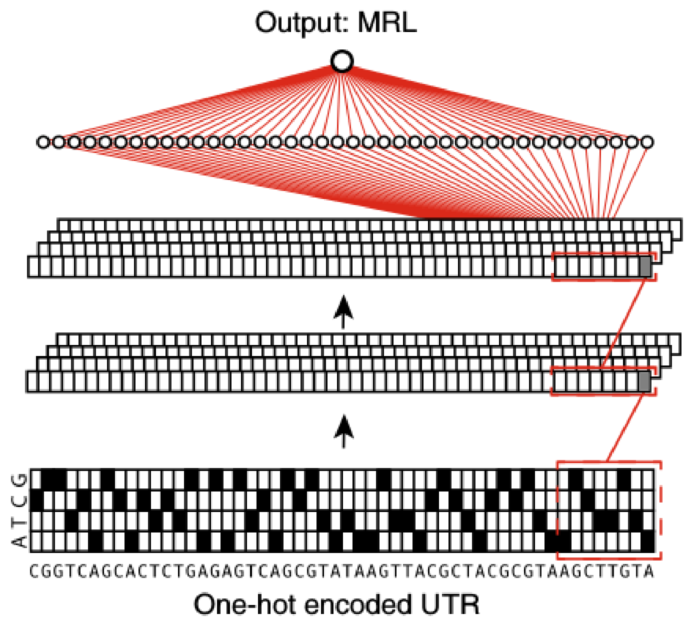


Figure 14: Structure of the Optimus 5-Prime. First the data is one-hot encoded, then it goes trough the convolution layers, and last it goes trough the dense layers to output the prediction. [14]

In the article they also mention in more detail the composition of each layer:

- First convolution layer: 120 filters (8 x 4), rectified linear unit (ReLU) activation and 0% dropout.
- Second convolution layer: 120 filters (8 x 1), ReLU activation and 0% dropout.

[14]Image extracted from the original paper

- Third convolution layer: 120 filters (8 x 1), ReLU activation and 0% dropout.
- Dense layer: 40 nodes and 20% dropout.
- Output layer: one linear output.

What we can see thanks to these two examples is that CNNs that work with sequences tend to be simpler than the ones that work with images. This can be because images are more complex and have more features than sequences, thus needing more intricate designs.

Also the general structure of both is very similar: They one-hot encode the data, train a simple CNN with it (this is where both of them differ, and also where our structure will differ with them) and then pass that data to a fully-connected layer to obtain the prediction. This is a good sign that this is a good way to work with DNA sequences.

7 Methodology

In this section we will explain what we did, and how we did it. We will follow the whole process from start to finish.

First of all we will talk about how we obtained our data, then how we transformed those sequences so that our model can work with them. Last, we will cover the experiments we made and their results.

7.1 Obtaining the data

For this thesis, some data had to be obtained. We needed the genes to study and their promoter and 5'UTR sequences.

The positive genes were reference by this paper [12]. Control genes are the rest of the genes that encode protein. We had a list of positive genes (genes appearing in cases of intellectual disability) and control genes (genes that we had no proof of them appearing in those cases).

It is still not known if there are more positive genes hidden between the control genes. Opening the door to find more of them is one of the goals of this study. But we will cover this later.

7.1.1 Promoter sequences

The promoter sequences were obtained in two different ways. We did it with two different methods so we could be sure the result was correct. Both of them required the use of ensembl **BioMart**, a tool that allowed us to retrieve information from diverse datasets. In this case, it let us obtain the DNA sequences corresponding to the promoter of each gene. The promoter is the section left of the **Transcription Start Site** (TSS). Since in the context of DNA sequences "left" is what we mean by "upstream", we asked for the first 250 base pairs (characters) upstream from the TSS. However, even though the BioMart tutorials also used this as a way of obtaining the promoter of a gene, we had no way of knowing 100% if we were obtaining the correct data. That encouraged us to look for a second method, just in case.

The second way also uses BioMart, but not in the same way. Since there is no option that returns the promoter corresponding to a gene, we had to take a little detour.

There is a certain project, FANTOM5 [13] that detected promoters from the neurons. We obtained the list of promoters from this source. Since they were not associated with any gene, we had to look for, for each of these genes, its corresponding promoter from this list.

The way we did this was the following: First, we had to ask BioMart for some information:

- For each gene, its **starting** and **ending location**. For example, a certain gene might start in the position 1950000000 and end in the position 1950002000.
- For each promoter, not only we needed its **starting** and **ending location**, but also its **Chromosome**.

After that, we had to connect every gene to its corresponding promoter, and we did that by joining each gene with the promoter that ends the closest to its start, without going over it.

For example, if we had a gene starting at the position 100, and two promoters, one ending at the position 90 and the other one at the position 102, the only valid promoter would be the first one, since it does not end after the 100th position.

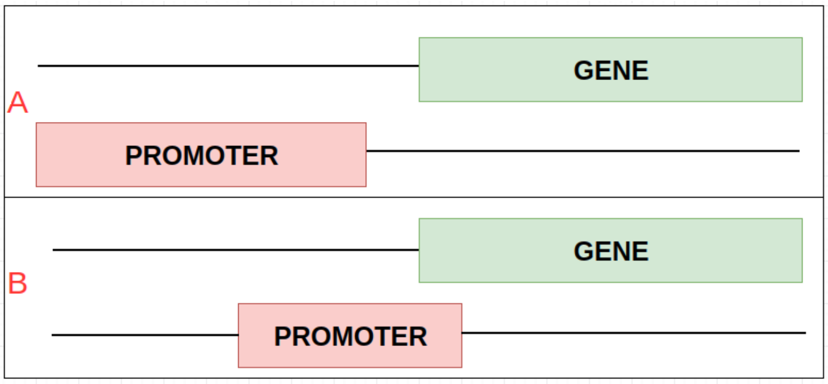


Figure 15: A. Since the promoter ends before the gene starts, it could be a valid assignment. B. The promoter ends after the gene start, so it would be an impossible scenario in a real sequence.

This way, we obtained a list that contained all the genes and their respective promoter. This second way, however, got more complicated than we thought. Multiple genes had the exact same promoter assigned and, even though that is possible and can happen in specific situations, it is an occurrence that has to be study.

One explanation could be that some genes can present different versions and variations, located in different – but sometimes very close – locations. This could be an explanation of some of these occurrences, that the genes that share promoters are different versions of the same gene, or are related in some way. But we cannot be sure without studying it properly, and that could mean a lot of work. This issue is closely related to this study, but even with that, it is out of the scope of this particular project.

Since we were not sure this data was usable for our project, we did not use it in the end. We only worked with the data we obtained from the first way, entirely through BioMart.

7.1.2 5'UTR sequences

The 5'UTR sequences were much easier to obtain, since BioMart let us ask directly for the 5'UTR sequences of a gene. We just had to make a query asking specifically for them to obtain all the 5'UTR sequences of every gene we have.

Additional information about the retrieval of sequences can be found at the Appendix 1.

7.2 Treating the sequences

We could not feed directly the data to our network, since some preprocessing and transformations had to be done.

First we had to reverse the sequences. The reason why we do this is because BioMart returns sequences that are in the 5' – 3' direction, in other words, sequences whose start is closer to the 5' end, and whose end is closer to the 3' end.

The polymerase II transcripts the sequences in the reverse order, in the 3' – 5' direction. We want our sequences to match the polymerase II direction, so that is why we have to reverse them.

Second, the sequences had to be padded and trimmed so we could work with them, since all had to have the same length. Considering every promoter sequence we got had a length of 250 characters (since we asked BioMart for the first 250 base pairs) and the mean length of the 5'UTR sequences was also around 250 characters, we set the max length of our sequences to 250 characters, trimming the longer sequences and padding the shorter with an special character, "X".

Finally we had to encode the sequences to one-hot encoding, since the network has to work with numeric values between 0 and 1, not characters. For that, we assigned a code to each character (A = 10000, C = 01000, G = 00100, T = 00010, X = 00001), and this way obtained the final sequence we would feed our network with.

Here is an example of this whole process:

$$\begin{array}{ccccccc}
\text{ACCG} & \rightarrow & \text{GCCA} & \rightarrow & \text{GCCA} & \rightarrow & 00100010000100010000 \\
\text{GGC} & \rightarrow & \text{CGG} & \rightarrow & \text{CGGX} & \rightarrow & 01000001000010000001 \\
\text{T} & \rightarrow & \text{T} & \rightarrow & \text{TXXX} & \rightarrow & 00010000010000100001 \\
\text{TC} & \rightarrow & \text{CT} & \rightarrow & \text{CTXX} & \rightarrow & 01000000100000100001
\end{array}$$

This procedure is explained more in depth in its corresponding Markdown script, found in the GitHub repository (link in the Appendix).

7.3 The three experiments

Here we will explain in depth the three main experiments we did. We will also explain here the way we worked with these experiments. First of all, We will briefly introduce them:

- **Promoter vs UTR**: In this experiment we will **compare** promoter sequences and 5'UTR sequences, and see if the network is capable of **telling apart those two** kind of sequences. We, at least, wanted to know if our networks are capable of recognizing patterns in promoters and 5'UTR. If not, then we would have to move to another architecture.
- **Promoter experiment**: Here we tried to see if we can predict if a gene is positive or negative using only the **promoter sequences**. We wanted to achieve that by introducing promoter sequences to our model and checking whether or not it would learn from them how to tell apart positive genes from negative ones.
- **UTR experiment**: We tired to see if we can predict if a gene is positive or negative using only the **5'UTR sequences**. It is the same experiment as the second one, but with 5'UTR sequences.

Why these three experiments? We really only need one of them to work, either the Promoter or the UTR, to validate our hypothesis that there are differences between different groups of genes (genes that present intellectual disability versus genes that do not) in their regulatory elements. Of course if both of them work, then even better. The first experiment is there only to tell us whether we are using networks suited for this kind of problem, capable of learning features from sequences.

These three experiments were developed in Keras R, also using TensorFlow. Each one of them had its own way to be executed. Some required cross-validation while others did not, and each of them had their own values for their hyperparameters. All these aspects will be mentioned in their respective sections.

What was common to every experiment was the loss function and the metrics. We worked with "loss_binary_crossentropy" from Keras as our loss function. This was because we work with only two classes, and our output goes through a Sigmoid, opposed to a Softmax, that would need a different loss function. This loss function is designed for Binary Classification problems.

For the metrics, we focused on the accuracy over the loss, because our priority was maximizing the correct number of predictions, we do not care as much about "how wrong are our wrong predictions" as much.

We developed some scripts and R Markdowns for the experiments. They will be explained in the Appendixes, and accessible to everyone so they can be replicated.

7.3.1 Promoter vs UTR

The purpose of this experiment is to know if the networks we are using are capable of recognizing patterns in promoters and 5'UTRs and telling them apart.

Should this experiment fail, it could mean that there might not be enough relevant features in our sequences, but it could also mean the relevant features they possess are shared between promoter and 5'UTR sequences – even thought this would be highly unlikely. It would not mean that the other two experiments **would** also fail, but in case they did also fail, it probably means the networks we tried were not suited for these kind of experiments.

The model was training with 17391 promoter sequences and 17391 5'UTR sequences, so the data we were working with was balanced. A difference between this experiment and the other two is that both the promoter and the UTR sequences come from the **same genes**. That does not happen in the other two experiments, since in those cases the positive sequences come from positive genes, and the negative sequences from negative genes (and no gene is both positive and negative).

Networks used

We worked with a very simple network for this experiment:

- Convolutional1d layer – 48 filters, Kernel Size 24, ReLU activation
- Flatten layer
- Dense layer – 50 nodes, ReLU activation
- Dense layer – 1 node, sigmoid activation

Our convolutional layer tries to capture the features of the sequences, while the dense layer tries to learn from them. Finally, the last dense layer converts the output in a 0 or a 1, thus predicting negative or positive (promoter or 5'UTR). Any network more complex than that achieved a worse performance.

Since this experiment only tries to show that our networks work fine and that our sequences contain relevant features, we will not try to get the best possible result. We just need a result that shows some success. For this reason, we see no need to apply techniques such as cross-validation to this model.

7.3.2 Promoter experiment

In this experiment, we wanted to check whether the promoter sequences were useful at distinguishing intellectual disability genes. To achieve that, we obtained, as explained before, the promoter sequences for both the positive and control cases that we will feed to our model.

We work with the promoter sequences of the positive genes provided, approximately 750 genes. Each one of these genes have only one promoter sequence, so our model will be trained with that number of positive sequences.

For the negative sequences, we had 17.000 negative genes. That equals a maximum of 17.000 negative sequences we could have used. However, we wanted our data to be balanced, so we could only work with 750 of them.

In case the results were good enough, we would then apply cross-validation to get a more accurate result, and work with more genes. But for that, we needed first some small results.

<u>Networks used</u>

We worked with a wide range of different networks, trying to look for the best possible result. This was the structure that achieved the best performance:

- Convolutional 1d layer – 30 filters, Kernel Size 24, ReLU activation
- Convolutional 1d layer – 30 filters, Kernel Size 24, ReLU activation
- Max Pooling
- Flatten layer
- Dense layer – 50 nodes, ReLU activation
- Dense layer – 1 node, sigmoid activation

It is a simple network – no regularization, few filters in only two layers – but any more than that heavily overfitted the model and/or achieved worse results.

7.3.3 UTR experiment

In this experiment, we also wanted to check if the 5'UTR sequences had any impact in the existence of intellectual disability. We follow a similar procedure to the previous experiment.

Every network experiment was performed with k-fold cross-validation, k=5. In each fold, we worked with all the positive cases (3567 positive sequences) and the same number of negative cases (sub-sampled from a larger list). The value of the final result is the mean of the result of the 5 folds.

<u>Networks used</u>

We started our experiments with a very basic network. It was composed of:

- Convolutional 1d layer – 48 filters, Kernel Size 24, ReLU activation
- Flatten layer
- Dense layer – 50 nodes, ReLU activation
- Dense layer – 1 node, sigmoid activation

As we can see, is the same network used in the Promoter vs 5'UTR experiment. Our idea consisted in starting with a very basic network, and run many blocks of experiments. Every block freezes all parameters except for one, and we will try to get the best value for that parameter. Once every parameter is optimized, we will be able to say that we got the best possible result. This process is more detailed at Annex 3.

The chosen final network is composed of:

- 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

7.4 Results

These were the results we achieved from the three experiments.

7.4.1 Promoter vs UTR experiment results

The accuracy is the main metric we will be focusing on with these experiments. In this experiment, since it is only a test to know if we are working with acceptable sequences and networks, we will not be analyzing neither the loss nor the sensitivity/specificity.

Accuracy

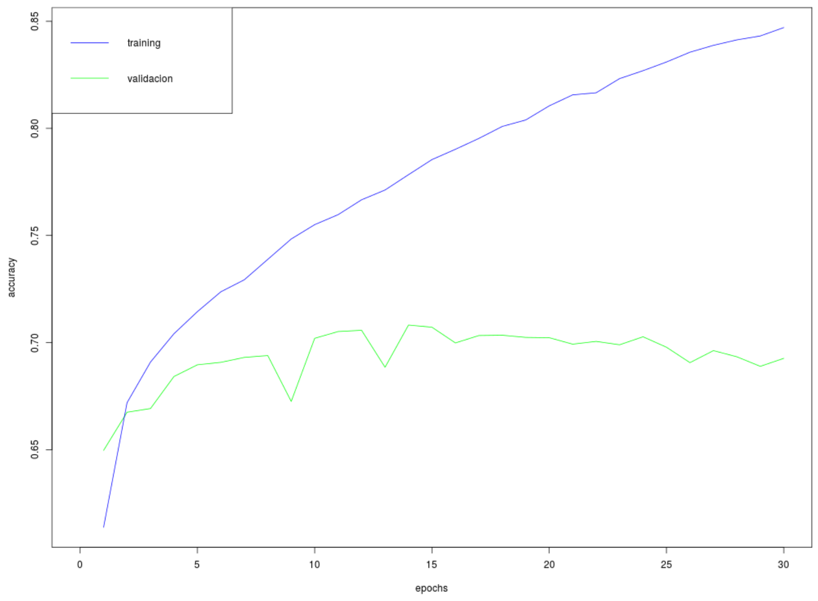


Figure 16: Accuracy and validation accuracy of the model of this experiment.

We can see the validation accuracy varies a lot, around 0.6 to 0.7 accuracy. The best accuracy we get is 0.708166 in the 14th epoch, so the model is capable of telling promoters and 5'UTR apart with some success in very few epochs. Since this is only a test, we do not need to improve the result.

7.4.2 Promoter experiment results

This is one of the two experiments that have to achieve a good enough result for our hypothesis to be true. Like before, the main metric we want to improve is the accuracy, but we will also have a look at the loss, sensitivity and specificity.

Accuracy

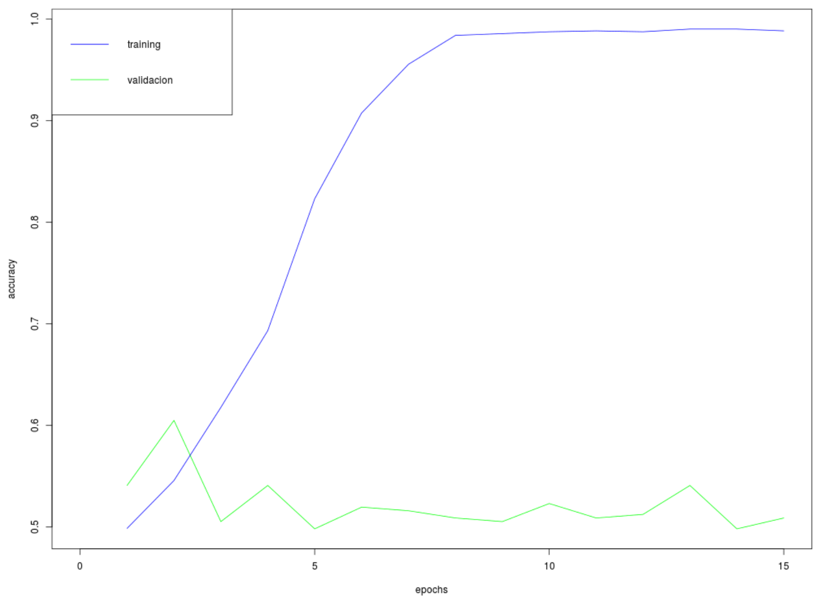


Figure 17: Accuracy of the promoter experiment.

The best accuracy we got was in the 2nd epoch with a value of 0.60498. It's a very low value, and the model does not seem to correctly learn from the sequences. Either that, or the model learns very fast all that can be learned, being that not enough.

Loss

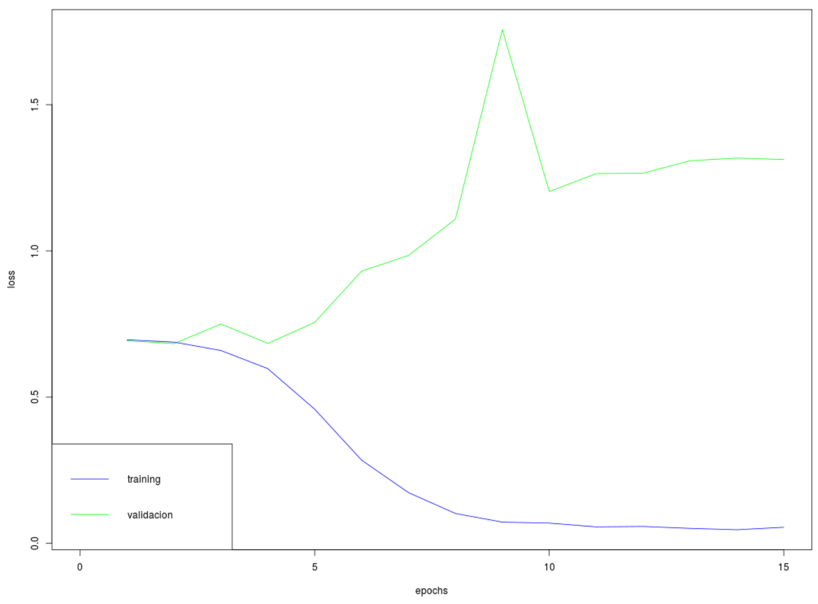


Figure 18: Loss of the promoter experiment.

We can see here a clear case of overfitting. Not only the starting result does not seem good enough, it keeps getting worse the more it trains.

Sensitivity and specificity

Sensitivity and specificity are statistical measures for the performance of a binary classification test.

– Sensitivity, also called **true positive rate**, measures the proportion of real positives that are correctly identified. Specificity, also called **true negative rate**, is the negative counterpart to sensitivity: measures the proportion of real negatives that are correctly identified.

The closer to 1 these values are, the better the classification test. For example, a value of sensitivity of 1 would mean that every positive case is correctly identified as such. A value of 0 would mean that no positive case is correctly identified.

Some tests value one more than the other. For example, if a test were to determine if a patient has cancer, we would value sensitivity over specificity: It is better to falsely diagnose a patient with cancer, than to miss a cancer in an infected patient. In our case both of them are equally important, since we are not going to give a diagnosis to a patient with our model.

There is also a third value, called **Youden's J statistic**, or simply Youden's index, that determines how good a diagnostic test is. The value of J is the result of the following operation:

$$J = \textit{sensitivity} + \textit{specificity} - 1$$

Since sensitivity and specificity are constantly changing, this value lets us know whether our model is improving or not.

The graphic depicting the results of the experiment is this one:

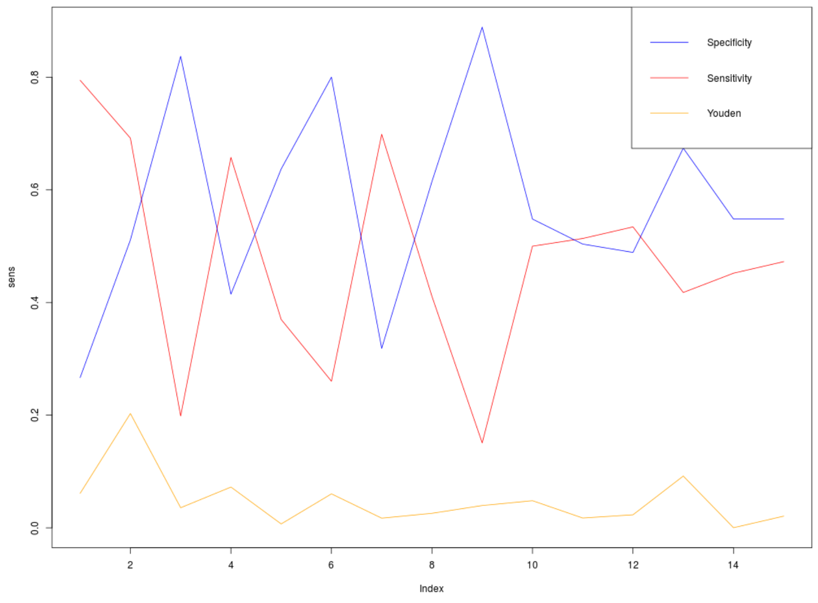


Figure 19: Sensitivity, specificity and Youden's index of the promoter experiment.

What we have to look at in this graphic is the value of the Youden's index, since the other two rise and fall in every epoch. The value of J lets us know whether the combined value of sensitivity and specificity combined is better or worse than before.

Having its peak at the 2nd epoch, we can see that the model is not doing a good job at learning. There might be some small features in the sequences that determine whether it is a positive or a negative case, but not enough to obtain an acceptable result.

7.4.3 UTR experiment results

Since the promoter experiment did not achieve an acceptable result, we depend on this experiment to obtain a better result for our hypothesis to have a chance to be proved. We will start, like before, with the accuracy obtained.

Accuracy

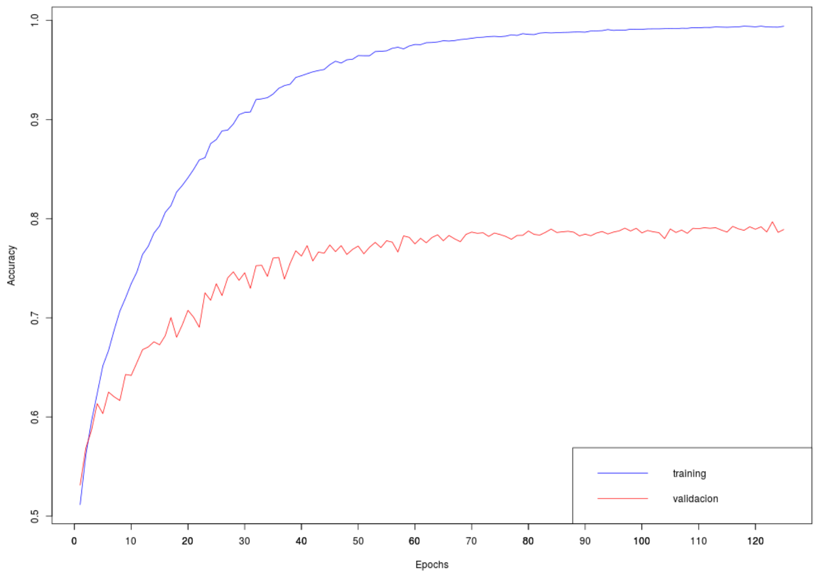


Figure 20: Accuracy of the UTR experiment.

The model reaches easily a value of almost 0.8 validation accuracy. It is a very acceptable result, which confirms us that the 5'UTR region of the genes have some relevance in the appearance of intellectual disability.

The network was trained for 125 epochs, however as we can see in the graphic, we can stop the training at the 60$^{\text{th}}$ epoch and still get a good result. Though, if we really want the very best accuracy, we would need to train the network the full 125 epochs, or even more, even thought that would result on an improvement of less than 0.05 validation accuracy.

Loss

We can see in *Figure 21* that it appears to be some overfitting in the model. That means that if we train the full 125 epochs, or more, we would be improving our accuracy at the cost of our loss. That is something to keep in mind wen choosing whether to make an early stop or not.

Sensitivity and Specificity

We can see in *Figure 22* that our specificity is almost always higher than our sensitivity. That means our model is better at predicting negative values than positive values in every epoch.

The Youden's index shows us how, even though both these values oscillate constantly, if we combine them they have a tendency to improve. That is a good way to know if our model is improving or not.

Comparing the results with a Random Forest

Now that we know the results, we have the following question. Does deep learning improve this problem, in comparison with traditional machine learning? For this reason, we have to compare our results with the results of a classical Machine Learning model.

For that, we will use Caret's train method, which lets us train a Random Forest model. We chose Random Forest because it usually gets good results and it is easy to set up, and it is one of the most used Machine Learning techniques used outside of Neural Networks.

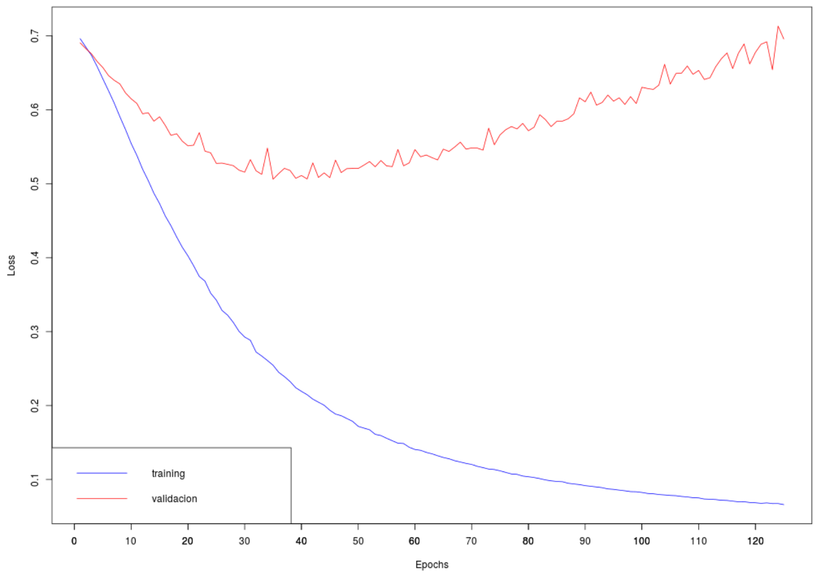


Figure 21: Sensitivity, specificity and Youden's index of the UTR experiment.

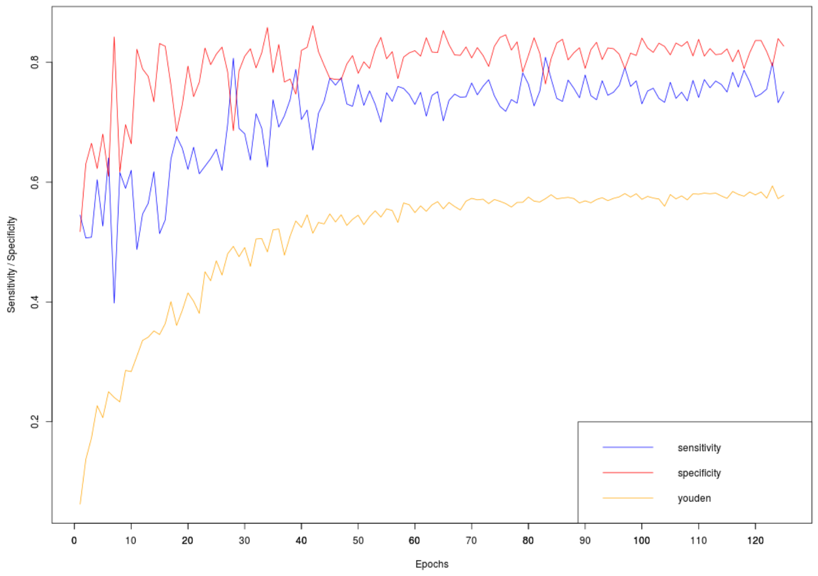


Figure 22: Sensitivity, specificity and Youden's index of the UTR experiment.

We used k-fold cross-validation, like before, with k = 5, each fold working with the same set of positive genes and a different set of negative genes. The result (the mean of the five models) had a validation accuracy of 0.7650452, versus the 0.798495 we got with our CNN.

Not by much, but it is a worse result. This way, we can say that convolution get us better results than ML models such as Random Forest.

8 Additional experiments

Apart from the main experiments, we performed some additional experiments to test some ideas. They were conducted to try potential discoveries, check if everything was in order or earn additional information. This is the list of these experiments:

8.1 Shared prefixes in UTR sequences

At the time of retrieval of the UTR sequences, we noticed that, in most of the genes, some, if not all its sequences, started with the same characters. We decided to investigate these prefixes.

These were the rules we set for the prefixes: We considered prefixes to be strings that were whole UTR sequences in their own, and that were contained at the start of other sequences. For example, in this case:

- ACCTG
- ACCTGAAG
- ACCTGGGAA
- ACTCCC

"ACCTG" would be a prefix, while "AC" would not. We also only considered sequences of the same gene, we never checked sequences of multiple genes.

These were the statistics we obtained:

Percentage of sequences that share a prefix:	48.62057%
Length of the longest prefix:	977
Length of the shortest prefix:	1
Mean length of the prefixes:	92.05437
Times the most shared prefix is shared:	29
Times the least shared prefix is shared:	2
Mean of times a prefix is shared:	3.317532
Mean number of sequences a gene has:	4.01912

A 48% of sequences sharing a prefix seemed like a relevant percentage, but we were not sure of how to work with this in this thesis. We did not push this idea further, but maybe it could be explored in the future in another study.

8.2 Padding to the left instead of to the right in the UTR experiment

We wanted to check whether the direction of the padding had any relevance in the result of the UTR experiment. For example, if the sequence "ACCG" had to be padded to be of length 6, we would have gotten the sequence “ACCGXX” as a result. What if we worked instead with the sequence “XXACCG”? We decided to try this change.

The only thing that changed from the UTR experiment was the padding of the sequences. The model we used was the one that got the best result in that experiment.

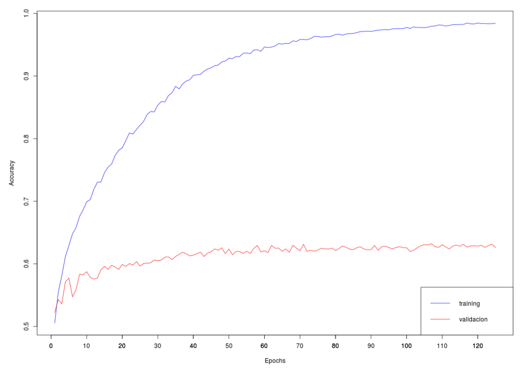


Figure 23: Accuracy of the experiment

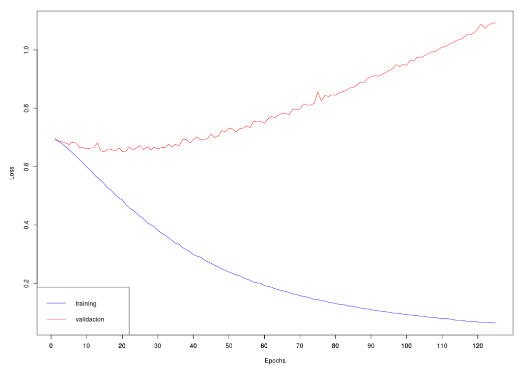


Figure 24: Loss of the experiment

We can see the model does not reach the same accuracy than the one trained with sequences padded to the right. We don’t know why padding before or after the sequence has this effect, but it what it is clear is that we have to work with sequences padded to the right, not to the left.

8.3 Not reversing the sequences in the UTR experiment

We tried to train the model without reversing the sequences, so that they do not match the direction of the polymerase II, to see if it was really the correct thing to do. So, if BioMart returned us the sequence “ACCTG”, we used to reverse it to the sequence “GTCCA”, but here we tried to train the model with the original “ACCTG”.

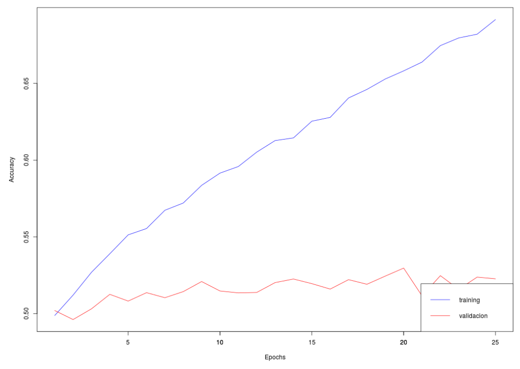


Figure 25: Accuracy of the experiment

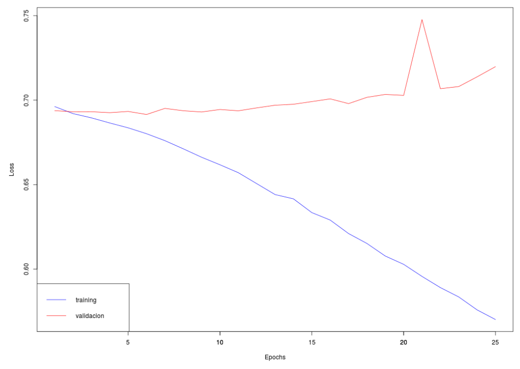
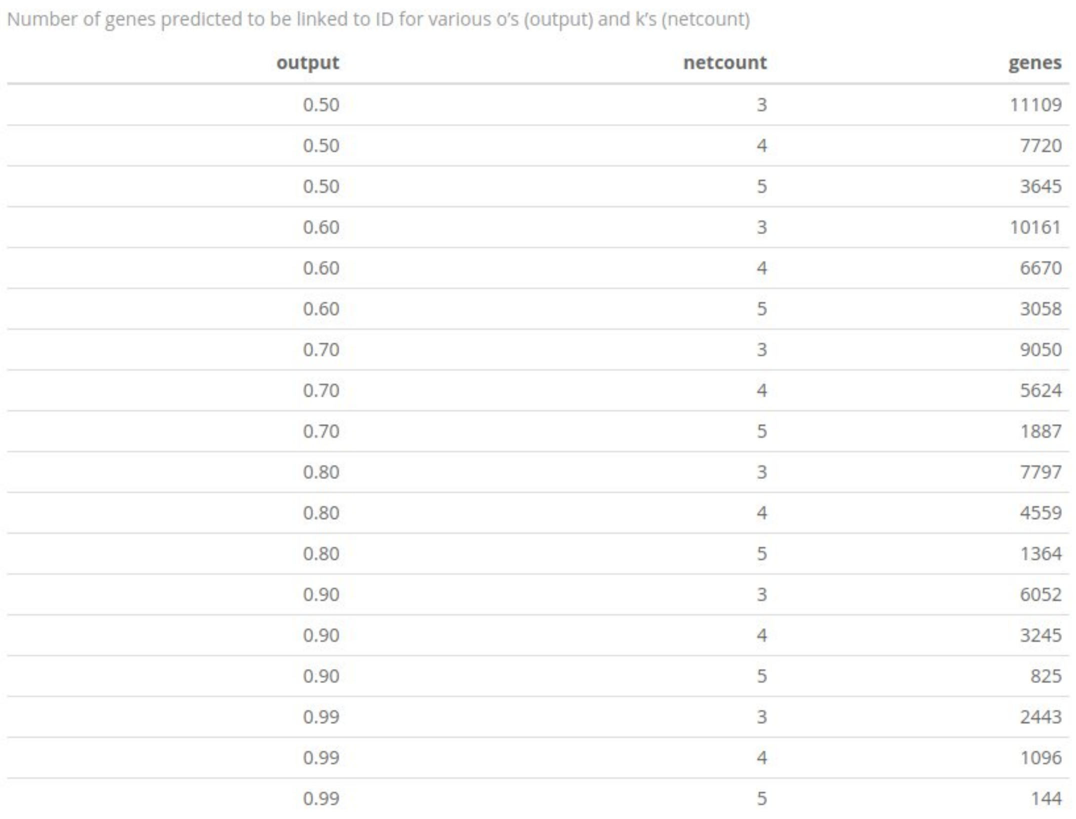


Figure 26: Loss of the experiment

We can see this model does not perform nearly as well as the other one. Thanks to these two last experiments, we now know for sure our sequences have to be **padded to the right and mirrored**.

8.4 Evaluating all negative genes

After all these tests, we decided to evaluate all the negative genes with our models. Why did we do this? Because not every gene listed in the negative list is really a negative gene, it could


Number of genes predicted to be linked to ID for various o's (output) and k's (netcount)

output	netcount	genes
0.50	3	11109
0.50	4	7720
0.50	5	3645
0.60	3	10161
0.60	4	6670
0.60	5	3058
0.70	3	9050
0.70	4	5624
0.70	5	1887
0.80	3	7797
0.80	4	4559
0.80	5	1364
0.90	3	6052
0.90	4	3245
0.90	5	825
0.99	3	2443
0.99	4	1096
0.99	5	144

Figure 27: From left to right: Threshold value of the output node, number of networks required to surpass that threshold, and number of genes that fulfils these conditions.

perfectly be a positive gene that has not yet been discovered.

We will, for that reason, evaluate all these genes from our negative list with the five models we trained for the UTR experiment(remember that we trained five models for our cross-validation).

To analyze this evaluation, an R markdown was made by the tutor of this project. In it we can find some interesting data related to that.

First of all, we have to talk about the outputs of our models. Until now, we only considered two possible values: "Predicted as positive" and "Predicted as negative". That, however, is not exactly what is happening. What the network really does is outputting a value from 0 to 1, and approximating that number to 0 or 1 thanks to an activation function, in our case, a sigmoid.

In the following *Figure 27*, we can see a table containing the number of genes that return an output bigger than the threshold indicated, in the number of networks specified. For example, the first row says that there are 11109 genes that output a value superior than 0.50 in at least 3 networks. Remember that we are working with approximately 17.000 genes.

After that, as a sanity check of how the training went, we plotted the raw output of the output node of each of our five models, for all the genes we are working with.

First we will show the plot of the positive genes. Since we know that these genes are positive thanks to other studies, we should expect density curves with most of the values tending to 1, and a long left tail ending at zero for each one of the five models. We can see the result in *Figure 28*

There are three main things to discuss in this graphic. From left to right: First we see a

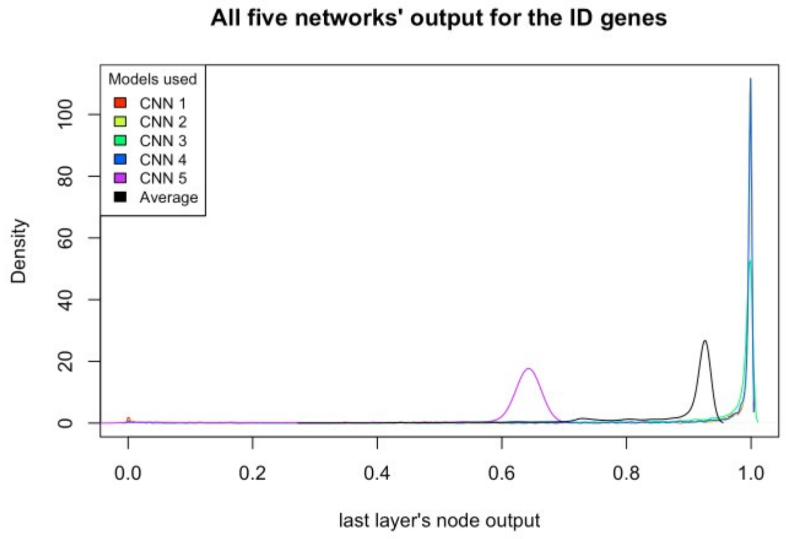


Figure 28: Output of the five networks for the positive genes

small bump at the 0.0 value, that corresponds to the genes that the network does not properly manage to learn. They are a very small percentage of the total so we will not worry about them.

Second, there is a bigger peak near the 0.65 mark for the 5th network, referring to some genes that did not get to saturate their output to 1. Lastly, we can see that the lines for the outputs of the five networks are very similar. Also, most of the average values are above 0.7, which is a good thing since we expected them to be relatively close to 1.

For the control genes, the expected result is quite different. Some of the genes labeled as negative should instead be positive, associated with the intellectual disability. They are not labeled as such because we do not know which are these genes.

Because of that, we hope to see a graph with two big spikes: One near the 0, for the genes that are mostly predicted as negative (our "true" negative genes), and one near the 1, for the genes that are predicts as associated with the intellectual disability, even though they belong to this control list of genes. Those would probably be the hidden positive genes, still not discovered. We can see in *Figure 29* that we were indeed on the right track.

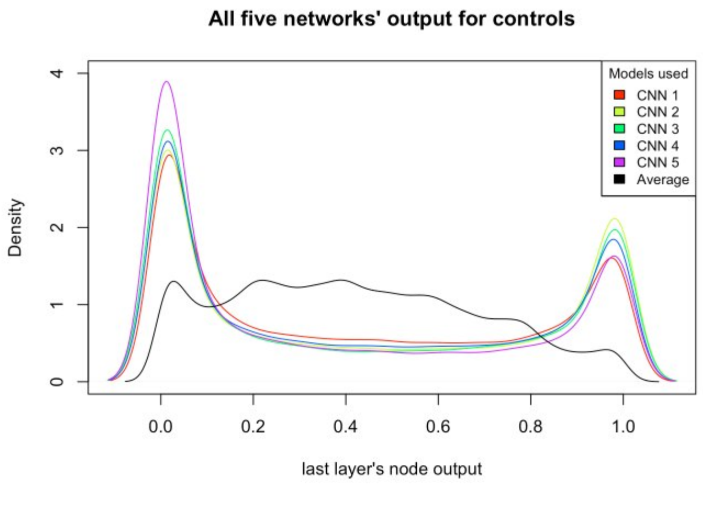


Figure 29: Output of the five networks for the negative genes

But this is not enough. By now, we do know three things:

- The models tend to correctly classify the positive genes.
- They show a tendency to identify control genes.
- They detect a group of genes that, even though they were labeled as negative genes, they could be intellectual disability genes when it comes to 5'UTR sequence features.

But even with that, we still do not have biological proof of these genes being really linked to the intellectual disability. For that, we used Amelie, a "To Good To Be True" tool, that finds associations for each gene with a list of papers about Mendelian diseases, and then rates those associations with a score from 0 to 100. The higher the value, the higher the confidence of the Mendelian association of the gene with the phenotype within the paper.

Amelie returns two plots: one representing the number of associations <gene,paper> given a query, and the other represents scores for these queries. Amelie works with two sets of data: Random queries and our predictions.

Our predictions will be represented with a red line. So, if the red line is over the 1750 mark in the first plot, it means that the query made with our predictions got 1750 associations <gene,paper>. How do we know if that is a good value? For that, Amelie makes 1000 random queries, and plots them with a black line. This way we can know if our query is better or worse than the mean.

For the second plot, it works exactly the same way: The associations of these 1000 random queries are rated, and their scores plotted. The red line marks how good our predictions were.

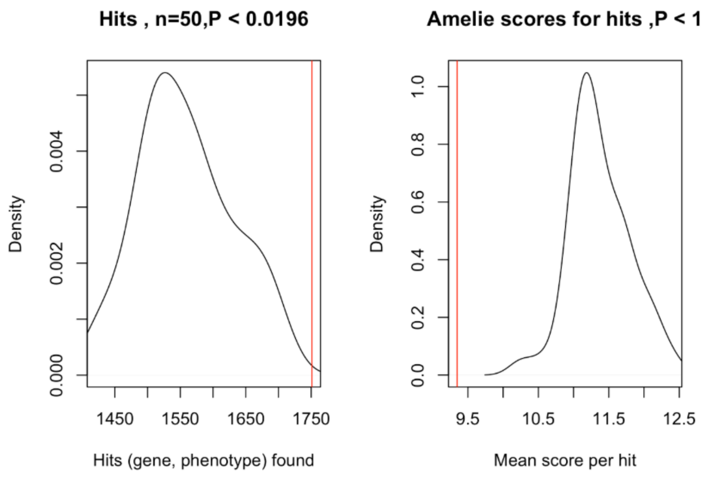


Figure 30: Plots created by Amelie

Looking at the first plot, we see a very good result: We got many more <gene,paper> associations than almost every random query, being that much bigger than the mean number of associations. With a P value of less than 0.0196 this value is pretty much certain.

On the other hand, the second plot shows a very poor result. The mean score per hit is much weaker than the scores obtained by the random queries. This can have many interpretations, but none of them definitive. And, of course, out of the scope of this project.

We can, however, suggest some explanations for this. It could be that these predictions are so new it is very hard to find good scores of them for Amelie. Another interpretation could be that the way Amelie produces scores could be biased by the number of phenotypes, or the number of genes, but that we do not know for sure.

In any case, it is a good sign that this tool finds more papers related to these genes and phenotype than the same number of genes chosen randomly.

9 Conclusions and future ways

Neural Networks, specially CNNs, have proven to be effective in solving this problem of distinguishing DNA sequences associated with intellectual disability with an acceptable accuracy of almost 80

With this, we have answered the questions we proposed through the study. We investigated sequence level differences based on elements of the human genome, such as promoters, and developed a methodology for such.

We also developed mechanisms for recovering and encoding sequences for their insertion in the networks, and treated those sequences with CNNs that we also evaluated.

Of the three experiments, two were satisfactory but one of them did not get much success. The experiment that distinguished promoter sequences from 5'UTR sequences, although the result was not exceptionally good, served us to prove that we were in the right track. And the UTR experiment obtained an acceptable result, that could in a future be improved even more with more resources and time.

The promoters experiment was not a success, tough. However, we can not say that promoters cannot help us distinguish intellectual disability genes from other genes. As we said before, both ways of obtaining the promoter sequences had some flaws, and we had no way of knowing if we got the correct sequences. It could be possible that we were working with sequences that do not correspond to the promoter of the genes. That could also mean that the Promoter vs UTR experiment was instead a Random vs UTR experiment. However, whatever the sequences we got were, it is clear they were structurally different from the 5'UTR sequences.

This work has potential to be expanded in the future in many different ways. We do not have proof of our network being the best one, so that is a possible improvement for the project. We also cannot be sure the Promoters experiment was performed with the correct data, so a third way of retrieve promoters sequences could be explored — and we know there are more ways to retrieve promoters.

Also, the evaluation of the control genes with these networks could be studied more in depth with more knowledge of genetics, and would be one of the most obvious follow-ups to this thesis. With the results we got for the evaluation, it is more than likely that our investigations could be used to discover the association with the intellectual disability of some genes, previously considered unrelated to that disease.

10 Bibliography

References

[1] 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

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

[3] Lapedes, A. S., & Farber, R. M. (1988). How Neural Nets Work. In D. Z. Anderson (Ed.), *Neural Information Processing Systems* pp. 442–456). http://papers.nips.cc/paper/59-how-neural-nets-work.pdf

[4] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 2278-2324. http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf

[5] Simonyan, K., & Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. *ArXiv:1409.1556 [Cs].* http://arxiv.org/abs/1409.1556

[6] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... Rabinovich, A. (2014). Going Deeper with Convolutions. *ArXiv:1409.4842 [Cs].* http://arxiv.org/abs/1409.4842

[7] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2015). Rethinking the Inception Architecture for Computer Vision. *ArXiv:1512.00567 [Cs].* http://arxiv.org/abs/1512.00567

[8] Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The LaTeX Companion.* Addison-Wesley, Reading, Massachusetts, 1993.

[9] He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep Residual Learning for Image Recognition. *ArXiv:1512.03385 [Cs].* Retrieved from http://arxiv.org/abs/1608.06993

[10] Kelley, D. R., Reshef, Y. A., Bileschi, M., Belanger, D., McLean, C. Y., & Snoek, J. (2018). Sequential regulatory activity prediction across chromosomes with convolutional neural networks. *Genome Research*, 28(5), 739–750. https://doi.org/10.1101/gr.227819.117

[11] Sample, P. J., Wang, B., Reid, D. W., Presnyak, V., McFadyen, I., Morris, D. R., & Seelig, G. (2018). *Human 5'UTR design and variant effect prediction from a massively parallel translation assay* [Preprint]. https://doi.org/10.1101/310375

[12] Vissers, L. E. L. M., Gilissen, C., & Veltman, J. A. (2016). Genetic studies in intellectual disability and related disorders. *Nature Reviews Genetics, 17*(1), 9–18. https://doi.org/10.1038/nrg3999

[13] A promoter-level mammalian expression atlas. (2014). *Nature*, 507(7493), 462–470. https://doi.org/10.1038/nature13182

11 Appendix

11.1 Appendix 1 – Retrieval of sequences with BioMart

To retrieve sequences – or any other kind of gene information – we can do it either by a query on its webpage https://www.ensembl.org/biomart/martview/, or by script. Since the scripts are included as Markdowns, we will only explain the queries.

11.1.1 Retrieval of promoter sequences

In the first two drop-down menus, we have to choose the options "Ensembl Genes 97", and then "Human Genes (GRCh38.p12)

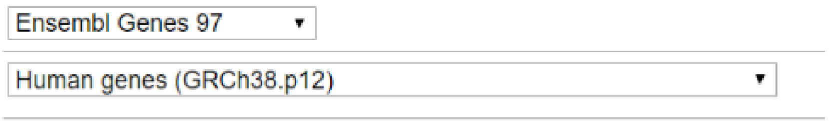


Figure 31

After that, we select the filters and the attributes. For the filters, we open the "Gene" section, and then check the "Input external references ID list" option. We can select a file containing all the genes we want to check, or type them in the box. Since this is only an example, we will type one example gene in the box. And we will specify that it is a HGNC symbol, since the genes we worked with were in that format.

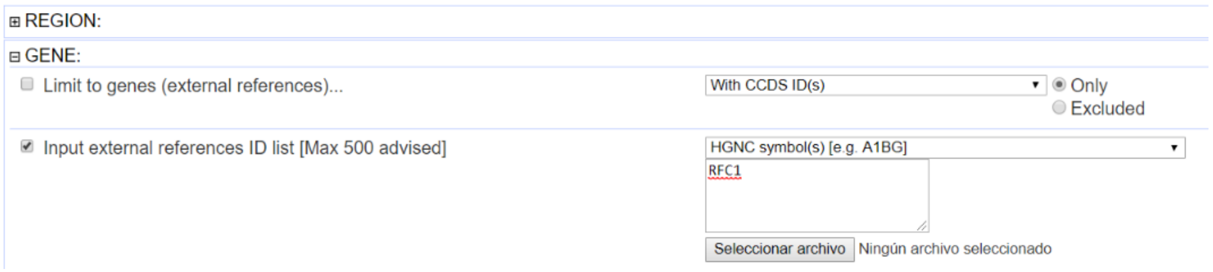

⊞ REGION:
⊟ GENE:
Limit to genes (external references)...
With CCDS ID(s)
Only
Excluded
Input external references ID list [Max 500 advised]
HGNC symbol(s) [e.g. A1BG]
RFC1
Seleccionar archivo
Ningún archivo seleccionado

Figure 32

For the attributes, we are interested in knowing the Flank coding region of the gene, 250 base pairs upstream, the area that corresponds to the promoter of the gene. For that, we have to check "Sequences" in the attributes window:

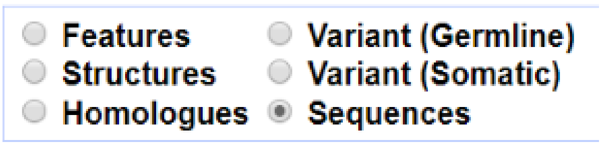
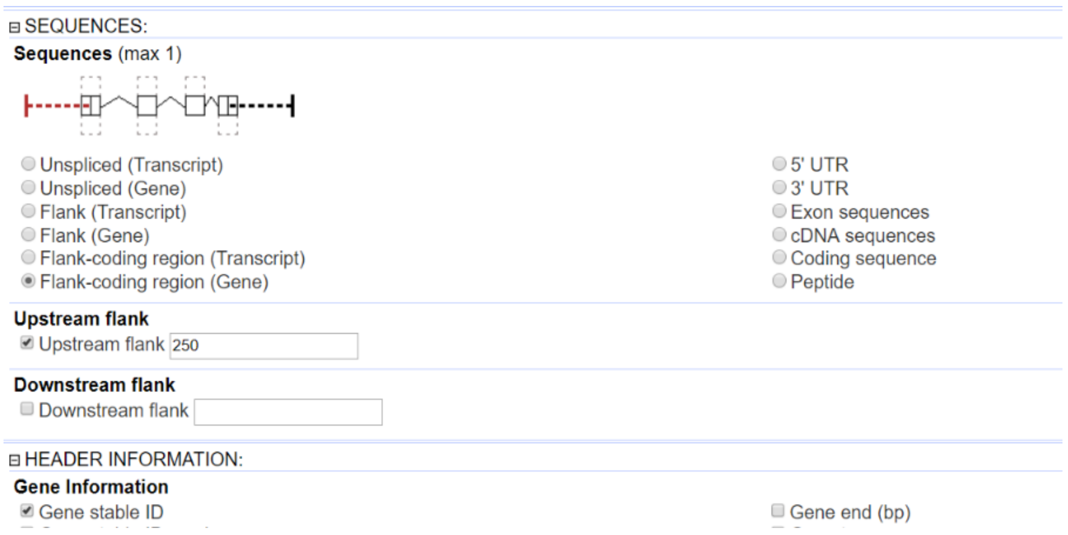


Figure 33

After that, we mark, inside "Sequences", Flank-coding region (Gene), and Upstream flank, with a value of 250.

Inside "Header information", we can also check Gene Stable ID, so we know which promoter sequence corresponds with each gene.



⊟ SEQUENCES:

Sequences (max 1)

○ Unspliced (Transcript)	○ 5' UTR
○ Unspliced (Gene)	○ 3' UTR
○ Flank (Transcript)	○ Exon sequences
○ Flank (Gene)	○ cDNA sequences
○ Flank-coding region (Transcript)	○ Coding sequence
● Flank-coding region (Gene)	○ Peptide

Upstream flank

☑ Upstream flank 250

Downstream flank

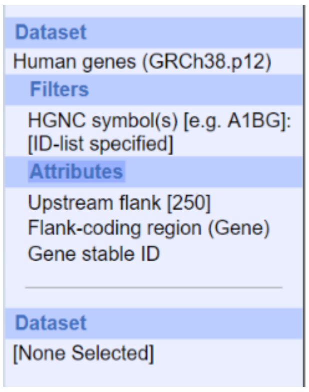
☐ Downstream flank

⊟ HEADER INFORMATION:

Gene Information

☑ Gene stable ID ☐ Gene end (bp)

Figure 34

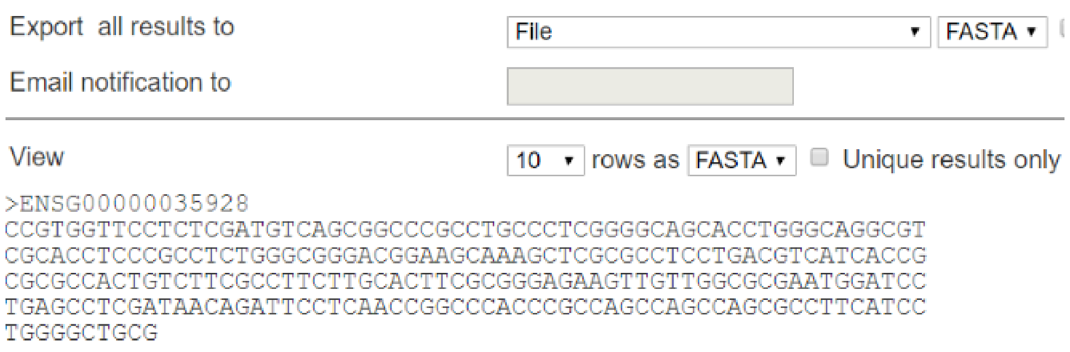


Dataset
Human genes (GRCh38.p12)
Filters
HGNC symbol(s) [e.g. A1BG]: [ID-list specified]
Attributes
Upstream flank [250]
Flank-coding region (Gene)
Gene stable ID

Dataset
[None Selected]

Figure 35

Our final query should look like the one in *Figure 31*. We then click “Results” (above the query), and obtain the sequences.



Export all results to File FASTA

Email notification to

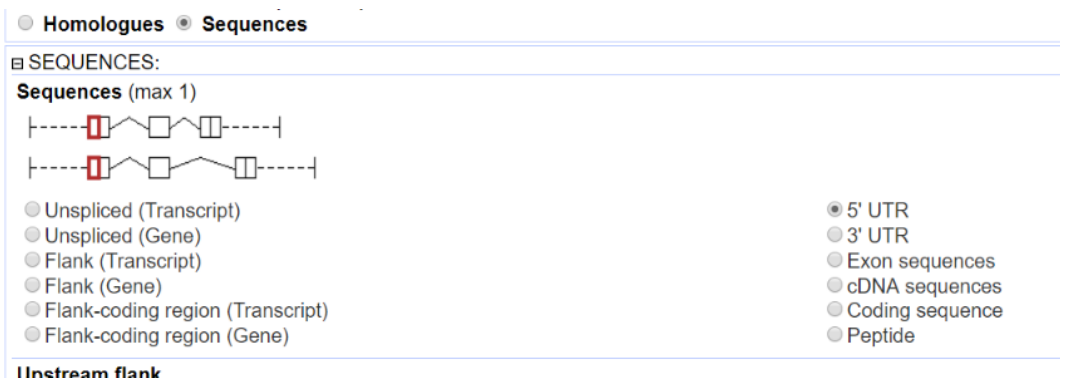
View 10 rows as FASTA ☐ Unique results only

```
>ENSG00000035928
CCGTGGTTCCTCTCGATGTCAGCGGCCCGCCTGCCCTCGGGGCAGCACCTGGGCAGGCGT
CGCACCTCCCGCCTCTGGGCGGGACGGAAGCAAAGCTCGCGCCTCCTGACGTCATCACCG
CGCGCCACTGTCTTCGCCTTCTTGCACTTCGCGGGAGAAGTTGTTGGCGCGAATGGATCC
TGAGCCTCGATAACAGATTCCTCAACCGGCCCACCCGCCAGCCAGCCAGCGCCTTCATCC
TGGGGCTGCG
```

Figure 36

11.1.2 Retrieval of 5’UTR sequences

The setup for this query starts the same way. The only difference is that, instead of asking for the Flank-coding region in the attributes, we will directly ask for the 5’UTR region:



Homologues [✓ Sequences]

⊟ SEQUENCES:

Sequences (max 1)

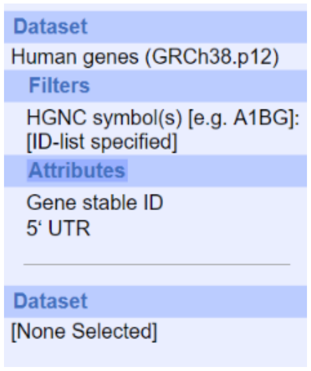
Unspliced (Transcript)
Unspliced (Gene)
Flank (Transcript)
Flank (Gene)
Flank-coding region (Transcript)
Flank-coding region (Gene)

[✓ 5' UTR]
3' UTR
Exon sequences
cDNA sequences
Coding sequence
Peptide

Upstream flank

Figure 37

Our final query should look like this:

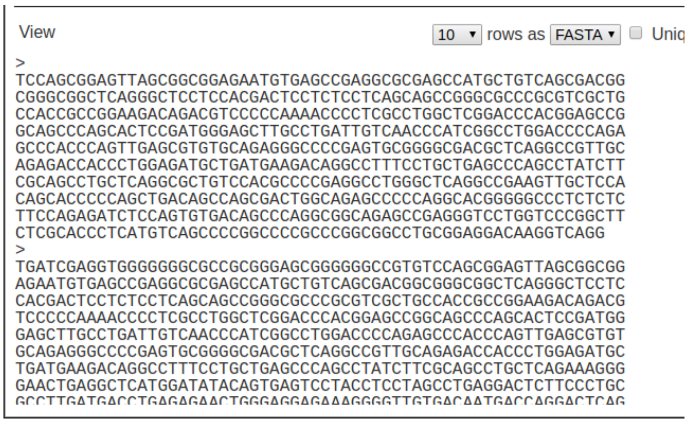


Dataset
Human genes (GRCh38.p12)
Filters
HGNC symbol(s) [e.g. A1BG]:
[ID-list specified]
Attributes
Gene stable ID
5' UTR

Dataset
[None Selected]

Figure 38

And the result would be something like the following image:



View 10 rows as FASTA Uniq

```
>
TCCAGCGGAGTTAGCGGCGGAGAATGTGAGCCGAGGCGCGAGCCATGCTGTCAGCGACGG
CGGGCGGCTCAGGGCTCCTCCACGACTCCTCTCCTCAGCAGCCGGGCGCCCGCGTCGCTG
CCACCGCCGGAAGACAGACGTCCCCCAAAACCCCTCGCCTGGCTCGGACCCACGGAGCCG
GCAGCCCAGCACTCCGATGGGAGCTTGCCTGATTGTCAACCCATCGGCCTGGACCCCAGA
GCCCACCCAGTTGAGCGTGTGCAGAGGGCCCCGAGTGCGGGGCGACGCTCAGGCCGTTGC
AGAGACCACCCTGGAGATGCTGATGAAGACAGGCCTTTCCTGCTGAGCCCAGCCTATCTT
CGCAGCCTGCTCAGGCGCTGTCCACGCCCCGAGGCCTGGGCTCAGGCCGAAGTTGCTCCA
CAGCACCCCCAGCTGACAGCCAGCGACTGGCAGAGCCCCCAGGCACGGGGGCCCTCTCTC
TTCCAGAGATCTCCAGTGTGACAGCCCAGGCGGCAGAGCCGAGGGTCCTGGTCCCGGCTT
CTCGCACCCTCATGTCAGCCCCGGCCCCGCCCGGCGGCCTGCGGAGGACAAGGTCAGG
>
TGATCGAGGTGGGGGGGCGCCGCGGGAGCGGGGGCCGTGTCCAGCGGAGTTAGCGGCGG
AGAATGTGAGCCGAGGCGCGAGCCATGCTGTCAGCGACGGCGGGCGGCTCAGGGCTCCTC
CACGACTCCTCTCCTCAGCAGCCGGGCGCCCGCGTCGCTGCCACCGCCGGAAGACAGACG
TCCCCCAAAACCCCTCGCCTGGCTCGGACCCACGGAGCCGGCAGCCCAGCACTCCGATGG
GAGCTTGCCTGATTGTCAACCCATCGGCCTGGACCCCAGAGCCCACCCAGTTGAGCGTGT
GCAGAGGGCCCCGAGTGCGGGGCGACGCTCAGGCCGTTGCAGAGACCACCCTGGAGATGC
TGATGAAGACAGGCCTTTCCTGCTGAGCCCAGCCTATCTTCGCAGCCTGCTCAGAAAGGG
GAACTGAGGCTCATGGATATACAGTGAGTCCTACCTCCTAGCCTGAGGACTCTTCCCTGC
GCCTTGATGACCTGAGAGAACTGGGAGGAGAAAGGGGTTGTGACAATGACCAGGACTCAG
```

Figure 39

As we see, not only it retrieves more than one sequence, but the sequences tend to differ a lot in length (from only one character to thousands of them). We have to take that into account when working with this data.

11.2 Appendix 2 – Evolution of the UTR experiments

We started the UTR experiment with a very simple network, with the purpose of improving it through multiple tests. The original architecture, as we said before, was:

- Convolutional1d layer – 48 filters, Kernel Size 24, ReLU activation
- Flatten layer
- Dense layer – 50 nodes, ReLU activation
- Dense layer – 1 node, sigmoid activation

For each block of tests, we froze every parameter of our model except for one, and tried to get the better value for it.

11.2.1 Optimizer

First of all, we wanted to set some parameters like the optimizer. These were the ones we tried, with their result:

Optimizer	Best validation accuracy	Epoch of best accuracy
Nadam	0.773187	85
Rmsprop	0.762106	88
Adam	0.771682	88
Adagrad	0.662927	88

Nadam gave us the best result, so it became the chosen optimizer.

11.2.2 Learning rate

The second thing we wanted to set, once decided the optimizer, was the learning rate.

Learning rate	Best validation accuracy	Epoch of best accuracy
0.001	0.500000	1
0.0001	0.773187	85
0.00001	0.708481	89

In the case of 0.001 learning rate, the model got stuck on 0.5 accuracy (predicting everything as negative) and stopped learning. Between the other two, 0.0001 returned the best result.

11.2.3 Kernel size

Third, we set the size of our filters. Remember that the size of our sequences is 1250 characters, since they are one-hot encoded.

Kernel size	Best validation accuracy	Epoch of best accuracy
10	0.744049	85
50	0.778796	87
75	0.783447	84
100	0.780027	83
150	0.778932	88
200	0.778522	89

11.2.4 First convolutional layer

We then decided the number of filters in our first convolutional layer. Since our best accuracies were appearing in the last epochs, we also decided to increase the number of epochs, to see if that way we could achieve better accuracies.

Filters	Best validation accuracy	Epoch of best accuracy
48	0.785551	120
64	0.787277	117
96	0.790013	124
128	0.792476	119
192	0.794528	123

Little by little our model was starting to perform even better. Also, we confirmed that increasing the number of epochs results in an improvement of the accuracy (at the cost of the loss, as we explained in the *Methodology* section)

11.2.5 Addition of regularization and max pooling

We want to try adding some more layers and see if the model improves with that. First we try adding some dropout, then we try adding a max pooling layer, and then we try adding both at the same time:

Layers	Best validation accuracy	Epoch of best accuracy
Dropout (0.25)	0.795622	111
Max Pooling	0.796853	123
Dropout (0.25) + Max Pooling	0.798495	118

As expected, adding dropout and max pooling improves the performance of the model. Lastly, we tried adding a second convolutional layer, right after the first one.

11.2.6 Second convolutional layer

Here we did only one experiment since training this model with more than one layer became a heavy time consuming task.

Filters	Best validation accuracy	Epoch of best accuracy
48	0.7923393	120

We can see that the result does not improve the best accuracy. Since we did not try more complex models, we are not sure this is the limit of the model. Even so, the result is more than acceptable, and it opens up the possibility of exploring this line of work in a future.

11.3 Appendix – 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.

The second one 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`.