University of Trento

Department of Cellular, Computational and Integrative Biology (CIBIO)

Bachelor's Degree in Biomolecular Sciences and Technologies

~ · ~

Academic Year 2022–2023

Development of a machine learning approach for the prediction of RG4-binding proteins

Supervisor
Prof. Erik Dassi

Graduate Student
Roberto Togni
223530

Final examination date: July 3, 2023

Abstract

The discovery of RNA G-quadruplexes (RG4s) and RG4-binding proteins (RG4BPs) has opened up exciting possibilities in the field of RNA biology. RG4s are unique four-stranded secondary structures formed by guanine-rich regions found in various functional regions of RNA, including telomeres, untranslated regions (UTRs) of mRNA, and regulatory elements within long non-coding RNAs (lncRNAs).

RG4s play a crucial role in a wide range of biological processes and regulatory mechanisms, such as translation, alternative splicing, and mRNA stability. Their functional repertoire is further expanded by their interaction with RG4BPs, a diverse group of proteins that recognize and bind to RG4 structures, thereby influencing their stability, accessibility, and biological functions.

In this work, we used a computational approach to facilitate the identification of novel RG4BPs. After constructing a dataset by merging information from different databases, we explored state-of-the-art machine learning techniques to create predictive models. Autokeras, an automated machine learning framework, was utilised to build a baseline model capable of detecting potential RG4BPs from the dataset. We then delved deeper into model optimisation using Keras, fine-tuning hyperparameters to enhance performance. We finally combined the predictions of three different models, identifying a small number of putative RG4BPs.

Contents

Chapter 1

Introduction

1.1 Post-Transcriptional Regulation of Gene Expression

The regulation of gene expression plays a fundamental role in the organization of the synthesis and localisation of a number of macromolecular structures. Given its importance, it is not surprising that it is a complex, multi-layered process, taking place at the transcriptional, post-transcriptional, and translational level [15] (Figure 1.1).

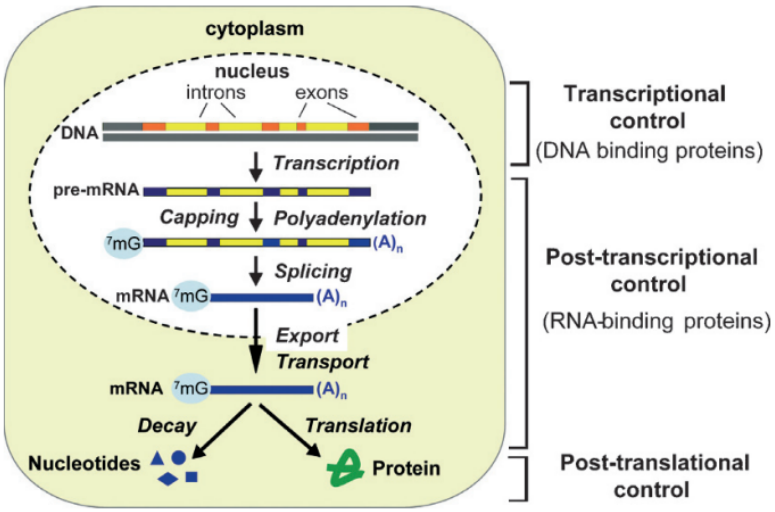

Figure 1.1: Gene expression is controlled at multiple steps. Image taken from [15].

Transcriptional regulation occurs in the nucleus and is responsible for key events such as the recruitment of the replication machinery and the regulation of its activity. The smooth running of these steps is directly related to the accessibility of chromatin, as well as depending on epigenetic modifications, and results in the production of an RNA molecule [15]. After RNA synthesis, post-transcriptional regulation determines its processing, localisation and propensity to be translated into protein, effectively decoupling the amount of mRNA and corresponding protein.

mRNA processing begins during transcription. First, the 5' end is capped by the addition

of a 7-methylguanosine molecule via an unusual 5,5' triphosphate bond. Next, a poly-A tail is added to the 3' end. While the two modifications share the purpose of protecting mRNA from degradation by exonucleases and facilitating its interaction with proteins (including ribosomes), polyadenylation also plays a role in the context of RNA diversity. Indeed, the presence of multiple polyadenylation sites allows the formation of different transcripts which are subject to different regulatory mechanisms [44].

Another mechanism underlying RNA diversity is alternative splicing: the alternative removal of introns from pre-mRNA allows the production of multiple isoforms, resulting in the addition of a further layer of complexity [29].

mRNA stability is directly related to its concentration in the cell, which is mainly determined by cis-regulatory elements (CREs) encoded within the mRNA sequence itself. These include secondary structure, binding sites for RNA-binding proteins (RBPs) and microRNAs [5]. Properly processed mRNAs are exported into the cytoplasm in the form of large export-competent complexes. Here, the translation process takes place, which consists of initiation, elongation, termination, and ribosome recycling. Within the mRNA, the AUG at the start of an open reading frame is identified by initiation factors (IFs), the small ribosomal subunit, and a special initiator methionine tRNA. The translation process then proceeds in the 5'-3' direction until it encounters a stop codon. Recognition of the latter by release factors (RFs) causes the release of the polypeptide, followed by dissociation of the ribosome from the mRNA [19].

1.1.1 Non-Coding RNAs

A significant fraction of the transcribed human genome does not code for proteins, and produces instead the so-called non-coding RNAs. At the functional level, it is possible to distinguish a plethora of non-coding RNAs, including microRNAs, small interfering RNAs, and piwi-interacting RNAs. Most of these are involved in the regulation of gene expression at several levels, from epigenetic silencing to post-transcriptional regulation of RNA stability [32].

MicroRNAs (miRNAs) are small, single stranded RNAs (19 to 25 nucleotides) derived from endogenous primary single stranded transcripts made by RNA polymerase II. They down-regulate cytoplasmic RNAs through mRNA degradation and translational repression, thus silencing gene expression. Individual miRNAs share sequence complementarity to the 3' UTR of multiple target mRNAs, thus fine-tuning entire transcriptional networks, and have been shown to be involved in the pathogenesis of many allergic diseases including asthma, eosinophilic esophagitis, allergic rhinitis and eczema [28, 27].

Small interfering RNAs (siRNAs) are small, double stranded RNA molecules derived from long endogenous or exogenous RNA molecules (such as viral RNAs). Although the mechanism of action is very similar to that of miRNAs, complementarity with the target is total, making them ideal candidates for RNA interference, the mechanism by which dsRNA induces gene silencing by targeting complementary mRNA for degradation [9].

Piwi-interacting RNAs (piRNAs) are generally 25-33 nucleotide long and are derived from separate piRNA clusters linked to transposon sequences. The synthesis process, which is not well known, is different from that observed for miRNAs and siRNAs, and the mechanism of action is based on the interaction with Piwi proteins. PiRNAs are most abundant in germ cells, although they can also influence gene expression in somatic cells, and are critical in silencing retrotransposons and other repeating elements [34].

1.1.2 Cis Elements

The fate of mRNA molecules is often dictated by cis-regulatory elements (CREs), distinctive secondary structures at the heart of the interaction with other RNA molecules, DNA, or RNA-binding proteins (RBPs) [11].

Among the most relevant structures are Adenylate/Uridylate-rich elements (ARE), internal ribosome entry site (IRES) and RNA G-quadruplexes. The former, located in the 3' untranslated region (UTR), represent one of the main determinants of mRNA stability. They are binding sites for proteins that can determine the mRNA rapid decay, thus regulating the amount of protein synthesised upstream of the translation step [40]. A further control layer is represented by Internal Ribosomal Entry Sites (IRESs), which allow ribosome recruitment and translation initiation in a cap-independent manner [1]. Finally, RNA G-quadruplexes are dynamic secondary structures underlying the interaction of RNA with a series of RBPs, which regulate aspects such as mRNA half-life and propensity to be translated [31].

1.1.3 RNA-Binding Proteins

RNA-binding proteins (RBPs) are a diverse family of proteins that bind to RNA molecules through specific RNA-binding domains. These are the functional units responsible for binding RNA, and often occur in modular arrangements that can coordinate and enhance binding to RNA [13]. Additionally, RBPs frequently contain intrinsically disordered regions, which operate themselves as RNA-binding domains [8]. Despite the growing interest in the latter, however, due to their inherently complex nature their mechanisms of binding, regulation, and physiological consequences remain poorly understood [22].

RBPs play a variety of roles in post-transcriptional gene regulation, including the control of mRNA splicing, stability, localization, and translation. They can interact with different regions of RNA, including 5' and 3' UTRs, coding regions, and non-coding RNAs, and can influence the fates of RNA molecules by affecting their interactions with other proteins or RNA molecules [13]. Proteins can interact with RNA using the main-chain of any residue and the side-chains of most residues, and the main interactions are based on Hydrogen bonds and Van der Waals forces. Not surprisingly, polar amino acids Ser and Asn and positively charged amino acids Lys and Arg predominate these interaction thanks to the formation of strong salt bridges [8].

1.2 RNA G-Quadruplexes

RNA G-quadruplexes (RG4s) are non-canonical secondary structures in mRNA, widely acknowledged as essential post-transcriptional regulators of gene expression [10]. They are composed by guanine tetrads held together by Hogsteen hydrogen bonds, and their folding can be controlled by several factors, including proteins, cations and small molecules [10].

RG4s have been validated as regulators of several biological processes, such as transcription termination, pre-RNA processing, mRNA translation and maintenance of telomere homeostasis [31, 12, 10]. Given the fundamental role played by RG4s, it is not surprising that their activity is heavily regulated. Of particular interest for this work are RG4-binding proteins (RG4BPs), which interact with RG4s in all their cellular functions and can alternatively stabilize, unwind

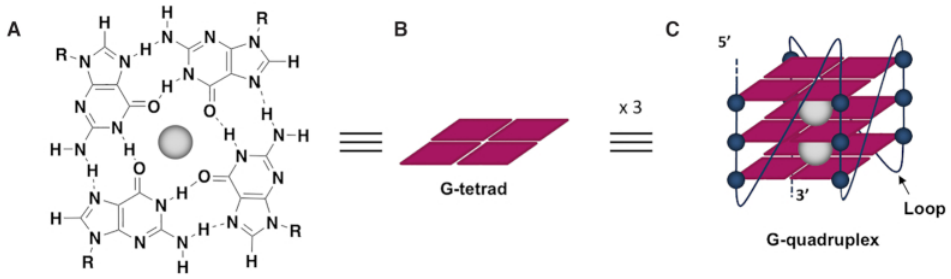

Figure 1.2: The RG4 structure. (A) Chemical structure and (B) schematic illustration of a G-tetrad (in purple), composed of four Gs linked together through Hoogsteen H-bonds; (C) example of intramolecular parallel RG4. Cations coordinated at the center of the tetrad are represented in gray, while Gs in blue. Image taken from [42].

or prevent their formation [42].

1.3 RG4-Binding Proteins

RG4-binding proteins (RG4BPs) are a class of RNA-binding proteins (RBPs) known for interacting with RG4s and for regulating - directly or indirectly - their dynamics. The analysis of known RG4BPs has led to the identification of a series of domains, motifs, or unstructured regions in their binding regions [25]. Among them, the most commonly reported are RRM (RNA-recognition motif) and RGG (Arginine-Glycine-Glycyne, also known as GAR domain) motifs (see Figure 1.3).

1.4 Making Predictions With Deep Learning Models

Deep learning is a branch of machine learning based on models that are composed of multiple layers of interconnected artificial neurons. Each neuron, also called perceptron, is an algorithm used for supervised learning of binary classifiers. In short, a perceptron (see Figure 1.4) takes input values, multiplies them by weights, adds them together and applies an activation function to the weighted sum. The output is then compared to the actual value, and the weights are updated in order to reduce the loss. Due to their nature, deep learning models are particularly good for pattern recognition tasks, being able to learn nonlinear and complex relationships between input and output data.

1.4.1 Dropout Multi-Layer Perceptron

A multi-layer perceptron (MLP) is a type of feed-forward neural network that consists of multiple layers of nodes, with each layer fully connected to the previous one (see Figure 1.5). While being structure-agnostic prevents MLPs from making spurious assumptions about the input, this very same feature also exposes them to issues such as overfitting. In order to prevent this problem, dropout regularization is often used. During training, some layer outputs are randomly ignored or "dropped out", making the training process noisy. As a consequence, the network is encouraged to learn a sparse representation [41].

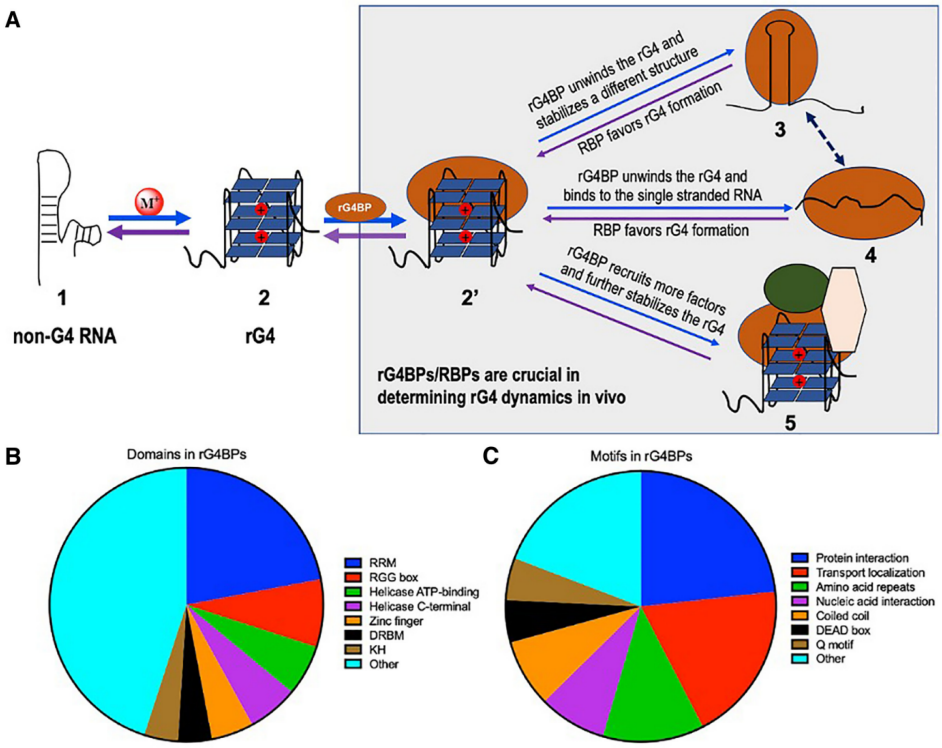

Figure 1.3: (A) Different possibilities of RG4–RG4BP interactions. Unfolded RNA folds into an RG4 with the help of RG4BP, or the latter can interact with pre-folded RG4, potentially resulting in one of the following consequences: stabilization of the RG4 or melting of RG4 to transit into an alternate structure [20] or a non-structured form [38], or recruitment of other binding factors to further stabilize the RG4 [2]. (B, C) Classification of reported proteins composition based on their loosely defined domain and motif compositions. Image adapted from [25].

1.4.2 Convolutional Neural Networks

At the base of convolutional neural networks (CNNs) is the assumption that the inputs are images. CNNs use a process known as convolution to extract features from input images, applying filters in order to highlight important characteristics such as edges, corners, and textures. CNNs typically consist of three layers: a *convolutional layer*, a *pooling layer*, and a *dense layer*.

The *convolutional layer* is an essential block in a CNN. Initially, a collection of kernels - small matrices of numerical values - is created. During the forward pass, the filter slides over the image, pixel by pixel. At each position, the dot product between the kernel and the corresponding pixels in the input image is performed, which results in a single value. The output values are placed in the corresponding position of a new matrix, called a feature map. These steps are summarised in Figure 1.6. The advantage of a convolution layer over trivial dense layers is that the former is provided with sparse interaction, which both reduces the memory requirement of the model and improves its statistical efficiency. Due to parameters sharing, the layers of a CNN also have the attribute of equivariance to translation - that is, a change in the input will be reflected in the same way in the output [14]. Non-linear layers are frequently included right after the convolutional layer in order to add non-linearity to the activation map.

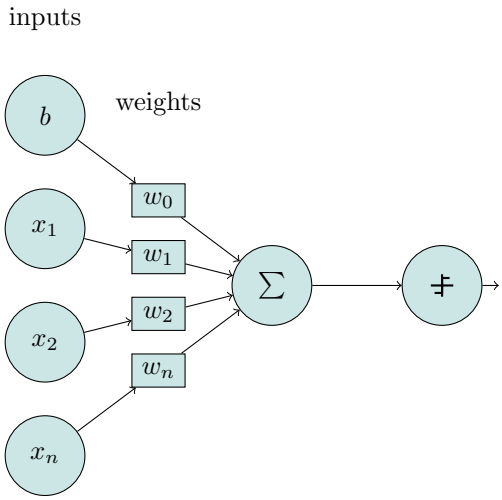

Figure 1.4: Graphic representation of a perceptron.

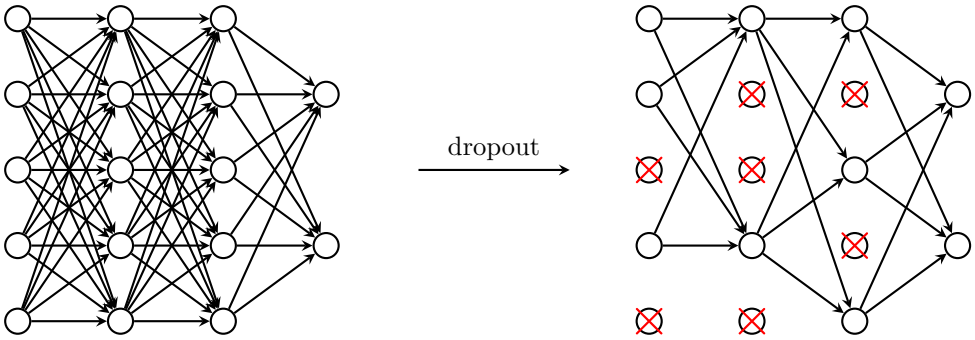

Figure 1.5: Dropout Neural Network Model. Left: A fully connected neural network with 2 hidden layers. Right: An example of a thinned network produced by applying dropout to the network on the left. Crossed units have been dropped. Image adapted from [36].

The aim of a *pooling layer* is to reduce the spatial dimensions (width and height) of the feature maps produced by the previous convolutional layer. This reduction can help control overfitting, reduce the number of parameters in the model, and improve overall computational efficiency. Pooling works by dividing the feature map into non-overlapping regions, or pools, and computing a summary statistic for each pool, such as the maximum (max pooling) or average (average pooling) value. The size of the pooling region, as well as the stride (the number of pixels by which the kernel is shifted), determine how much the spatial dimensions of the feature map are reduced. The pooling layer operates independently on each feature map produced by the previous convolutional layer. By reducing the spatial dimensions, pooling helps capture the presence of features in different regions of the image, rather than their precise location. This makes the model invariant to small translations of the input. In other words, it becomes more robust to changes in the position of features within the image.

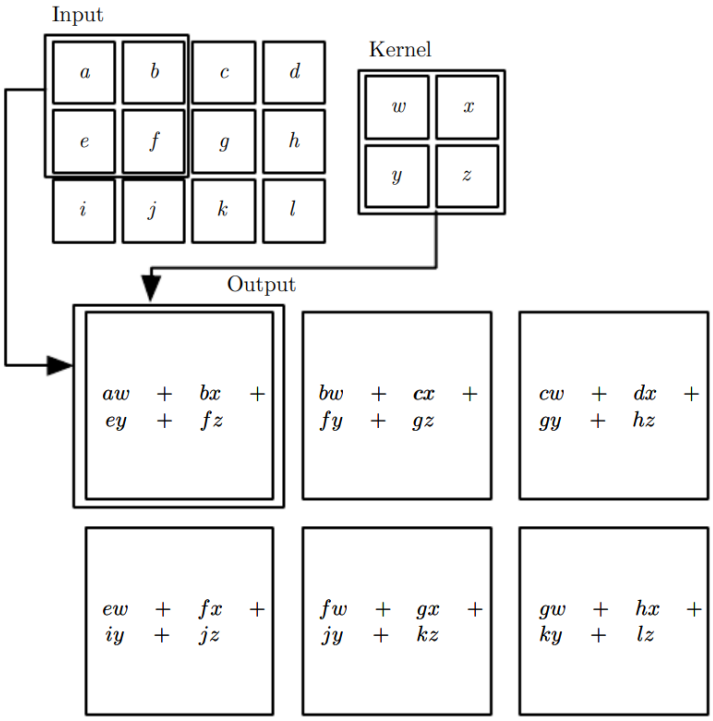

Figure 1.6: An example of 2-D convolution. Image taken from [14].

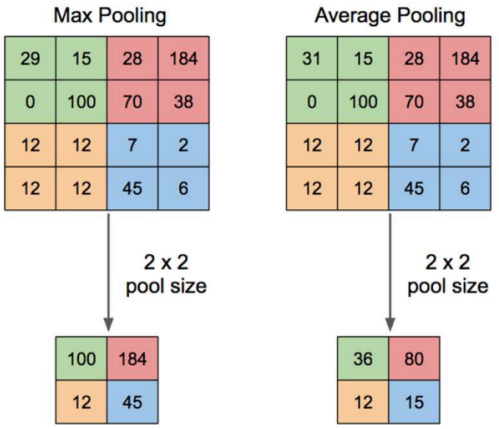

Figure 1.7: Illustration of Max Pooling and Average Pooling. Image taken from [48].

After the pooling layer, the input is flattened (turned into a column vector) and fed to a regular neural network consisting of one or more *dense layers* to perform classification. This last step allows to combine all of the learned features together in order to make a prediction about the input image.

1.4.3 Structured Data Classification

Structured data classification refers to the process of automatically assigning pre-defined labels to a dataset containing structured data. The latter is typically organized in a tabular format (e.g., CSV, Excel, etc.), with rows representing individual samples and columns representing different features. Structured data classification is typically performed using machine learning algorithms, such as decision trees, random forests, or deep neural networks, which learn patterns in the data and use them to make predictions about the label of new observations. The process of structured data classification usually involves several steps, including data cleaning, feature engineering, model selection, training and evaluation, and hyperparameter tuning.

Data cleaning is the most important step in machine learning. It is the process of preparing data during which incorrect, incomplete, duplicated, or incorrectly formatted data are removed or modified. Intuitively, this step becomes even more important when working with data from multiple sources.

Feature engineering is the process by which the most relevant features for the task are selected or created. In other words, it is the pre-processing step of machine learning which extracts features from raw data. Feature engineering encompasses one or more of the following steps: (1) *feature extraction*, whose objective is to find the most useful variables to be used within a predictive model; (2) *feature transformation*, which includes practices such as standardisation and data encoding and is aimed at transforming the input data into a more suitable form for the model; (3) *feature selection*, by which the subset of the most relevant features is selected by discarding redundant, irrelevant or noisy features.

Model selection involves choosing the most appropriate algorithm (or combination of algorithms) for the task, based on the properties of the data and the desired performance metrics. Other factors that should be taken into account when choosing a model are maintainability and model complexity. Intuitively, for equal performance a relatively simple model tends to be preferable to a complex one.

The next step consists of *model training and evaluation*, and results in the creation and assessment of a predictive model. Firstly, the data collected must be split into three datasets, as shown in Figure 1.8: (1) the *training set*, which is used by the model to learn the weights; (2) the *validation set*, whose purpose is to provide an unbiased set in order to assess the model's performance during training and prevent it from overfitting; (3) the *test set*, which consists of data never seen by the model that are used for its final evaluation.

During training, the network is presented with a set of inputs and the corresponding desired outputs. The network processes the inputs and produces an output, which is compared to the desired output to calculate the loss. The weights and biases of the network are then adjusted based on the loss using an optimization algorithm, such as gradient descent. This process is repeated over multiple iterations, known as epochs, until the network achieves the desired level of accuracy on the training data. After every epoch the network's performance is also evaluated on the validation set in order to avoid overfitting.

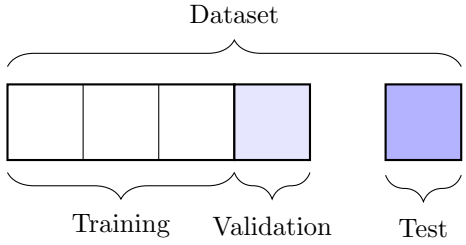

Figure 1.8: Example of dataset splitting.

The model's performance can be assessed using various metrics. Among the most used for classification tasks are accuracy, precision, recall and F1-score. *Accuracy* is defined as follows:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{1.1}$$

where TP = True Positives, TN = True Negatives, FP = False Positives and FN = False Negatives. Despite being an intuitive and easily interpretable evaluation metric, it is not sufficient to describe the full picture when working with an imbalanced dataset - that is, a dataset with skewed class proportions.
Two more robust metrics are *precision* and *recall*, defined in Equation 1.2 and 1.3, respectively. Although they might appear similar at first sight, they describe different things. In particular, while precision is aimed at identifying what proportion of positive identifiers is actually correct, the purpose of recall is to quantify what proportion of actual positives has been correctly identified.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{1.2}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{1.3}$$

Precision and recall are combined in the *F1-score*, which measures the accuracy of a model and is defined as the harmonic mean between precision and recall (1.4). Unlike accuracy, the F1-score gives more relevance to False Negatives and False Positives, thus being preferable when working with imbalanced datasets.

$$\text{F1-score} = \frac{2}{\frac{1}{\text{precision}} \cdot \frac{1}{\text{recall}}} \tag{1.4}$$

The last step in structured data classification and, more generally, in any classification task, is *hyperparameter tuning*. Hyperparameters are tunable parameters set prior to the learning process that directly affect how well a model performs. In the context of deep learning, some important hyperparameters are *train-test split ratio*, *learning rate*, *number of hidden layers*, *number of units in each layer*, *number of epochs*, *drop-out rate* and *activation function*.

Chapter 2

Aims

Although the interest in RNA G-quadruplexes is relatively recent, it is becoming increasingly clear that these elements play a fundamental role in a number of biological processes, including the control of mRNA processing, stability, and translation. While various approaches for the prediction and identification of these elements have been developed in the last few years [26], a less investigated aspect concerns the identification of proteins whose activity influences RG4 dynamics.

Given their extensive involvement in various processes, a better understanding of the factors regulating RG4 activity would improve our understanding of the dynamics underlying these processes. With this in mind, the project presented here aims to provide a tool for the ***identification*** of unknown RG4BPs, using a limited number of known features to ***predict*** whether a given protein is capable of binding RG4 motifs.

A further aim of the work - somehow related to the intrinsic nature of the method used - is to propose an unbiased approach to the study of RG4BPs, which involves investigating the biological characteristics of the proteins only after they have been computationally identified. In other words, this approach aims to streamline the study procedure in the laboratory by providing predictions on possible targets to be later analysed in detail.

Chapter 3

Methods

3.1 Data Sources

3.1.1 QUADRatlas

QUADRatlas [3] is a database of RNA G-quadruplexes (RG4s) and RG4-binding proteins (RG4BPs) resulting from the joint effort of two laboratories: the Laboratory of RNA Regulatory Networks at the Department CIBIO (University of Trento), led by Dr. Erik Dassi, and the RNA-binding proteins and genotoxic stress team at the CRCT (Toulouse), led by Dr. Stefania Millevoi. It includes experimentally-derived and computationally predicted RG4s in the human transcriptome, enriched with annotations describing their biological functions and disease associations. All the proteins representing the set of positives were taken from this database.

3.1.2 UniProt

The Universal Protein Resource (UniProt) [7] is a comprehensive resource for protein sequence and annotation data. For this work, we relied on the UniProt Knowledgebase (UniProtKB), containing functional information on proteins with accurate, consistent and rich annotation. In particular, we used UniProtKB/Swiss-Prot, which contains manually-annotated records with information extracted from the literature and curator-evaluated computational analyses. The code used to programmatically access the database can be found in Supplementary Materials 5.1.

3.1.3 AlphaFold

AlphaFold [24, 47] is a deep learning-based protein 3D structure prediction system developed by researchers at the University of Washington and the European Molecular Biology Laboratory (EMBL). It has demonstrated state-of-the-art performance, and has been used to predict the structures of thousands of proteins in a wide range of organisms, including Human. From the database it is possible to download the PDB (Protein Data Bank) files of the entire proteome of the organism of interest, which provide high-quality, experimentally-validated protein structure predictions.

3.1.4 Stride

STRIDE [18] is a program for protein secondary structure elements assignment from atomic resolution protein structures. It works by analyzing the backbone dihedral angles of the protein and identifying patterns that correspond to different secondary structure elements, such as alpha helices, beta strands, and turns. The program uses a sliding window approach to assign secondary structure elements, where a window of a fixed length is moved along the protein chain and the secondary structure element is assigned based on the dihedral angles within the window. The program can be run from command line as follows:

```
stride input.pdb -foutput.txt
```

Given the large number of files considered in this work, the process was automated as reported in Supplementary Materials 5.2.

3.2 Libraries for Dataset Creation

3.2.1 Pandas

Pandas [43] is a popular data manipulation and analysis library for Python. It provides a variety of data structures for working with tabular and time-series data, as well as tools for data cleaning, exploration and visualization. Among the key functionalities provided by pandas are reading and writing data from various sources (CSV, Excel, etc.), filtering, selecting and manipulating data based on conditions, joining, merging and reshaping datasets, and grouping and aggregating data based on different criteria.

3.2.2 NumPy

NumPy [16] is a Python library for numerical computing. It provides support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. NumPy is a powerful way to work with arrays and matrices, providing efficient memory management and optimized routines for numerical computations.

3.3 Neural Network Development

3.3.1 TabPFN

TabPFN [21] is an offline-trained Transformer algorithm that can perform supervised classification for small tabular datasets extremely rapidly. The model does not require hyperparameter tuning, and shows competitive performance compared to state-of-the-art classification methods. Although it shows inefficiencies related to dataset size, data type and missing values, it is an excellent tool for quick and relatively inexpensive preliminary predictions.

3.3.2 AutoKeras

AutoKeras [23] is an open-source Python library for automated machine learning (AutoML). It is built on top of Keras, and provides a high-level API for training deep learning models with minimal input from the user. AutoKeras uses a technique called neural architecture search (NAS) to automatically search for the optimal architecture of a deep learning model for a given task. It includes various pre-defined search spaces for different types of models, such as image classification, regression, and text classification, and can also generate custom architectures.

3.3.3 Keras

Keras [6] is a high-level deep learning API for Python, designed to simplify the process of building neural networks. It provides a user-friendly interface to create and train deep learning models, hiding the complexity of the underlying mathematical computations. Built on top of TensorFlow [30], Keras can run on both CPUs and GPUs. It supports a wide range of neural network architectures, including feed-forward networks, convolutional networks, and recurrent networks.

3.4 Data Preprocessing

3.4.1 Sampling Methods

Due to the nature of the dataset, an initial problem that was encountered consisted in the presence of underrepresented data and a severe class distribution skew. In order to solve this issue, we applied sampling methods to modify the imbalanced dataset and provide a balanced distribution [17]. As a result, we were able to use accuracy as the evaluation metric for our model.

Initially, we randomly downsampled the negatives, reducing the disparity between the two classes. We then oversampled the positives by randomly duplicating samples until we obtained a balanced dataset (see Supplementary Materials 5.3). In order to evaluate the performance of the model and reduce the risk of overfitting, cross-validation was used.

3.4.2 One-Hot Encoding

While some machine learning algorithms, such as decision trees, can work directly with categorical data, neural networks require data to be numeric. As a consequence, techniques are needed to map categorical data to integers. With one-hot encoding, each categorical value is converted into a new categorical column, and at each new column is assigned a binary value of 0 or 1. Each categorical value is therefore represented as a binary vector where all elements are 0 except for the index corresponding to the actual value, which is set to 1. An example of the process is provided in Figure 3.1.

Ids	Countries
1	Spain
2	Austria
3	China
4	Germany
5	France
6	Italy

	Austria	China	France	Germany	Italy	Spain
0	0	0	0	0	0	1
1	1	0	0	0	0	0
2	0	1	0	0	0	0
3	0	0	0	1	0	0
4	0	0	1	0	0	0
5	0	0	0	0	1	0

Figure 3.1: Example of one-hot encoding.

Among the advantages of one-hot encoding are the fact that the result is binary and that all the elements are in an orthogonal vector space. However, the feature space can blow up quickly for high cardinality, which leads to facing the curse of dimensionality.

3.4.3 Embedding Layers

Embedding layers can be used to learn a dense representation of categorical variables. While with one-hot encoding we obtain a dummy feature for each variable, embedding layer enables us to convert each entry into a fixed length vector of defined size. The dimensionality of the vector is a hyperparameter that can be tuned based on the size of the vocabulary and the complexity of the task. Since the resultant vector has real values instead of just 0's and 1's, it is said to be dense.

During training, the embedded layer takes as input a sequence of categorical variables (e.g. a sequence of words in a sentence) and maps each category to its learned vector representation. The resulting output is a sequence of dense vectors that can be used as input to the subsequent layers of the neural network.

3.4.4 Principal Component Analysis

Principal Component Analysis (PCA) [39] is an orthogonal linear transformation technique aimed at transforming a high-dimensional dataset into a lower-dimensional representation while preserving the most important information. The main idea behind PCA is to find a new set of uncorrelated variables, called principal components, that capture the maximum amount of variation in the original data.

The process of PCA involves several steps. First, the data is standardized to have zero mean and unit variance. Then, the covariance matrix is computed to capture the relationships between different variables. The eigenvalues and eigenvectors of the covariance matrix are calculated, with the latter representing the directions in which the data varies the most, and the former the magnitude of the related variation. The eigenvectors corresponding to the largest eigenvalues are selected as the principal components, and are used to form a new coordinate system in which the data is projected. By selecting a subset of the principal components, it is thus possible to reduce the dimensionality of the data while retaining a significant amount of their variance.

Chapter 4

Results

The following sections present the steps followed and the results obtained in this work. Given the extremely important biological role played by RG4s, a key aspect to better understand their functioning revolves around their regulation. To this end, we adopted a computational approach aimed at identifying new potential RG4BPs, using a deep learning approach to extrapolate features useful for their identification.

4.1 Dataset Preparation

To begin with, we created a dataset containing information on the entire human proteome so that we could then proceed with the training of various models. To do so, we first performed a bulk download from UniProt (see Supplementary Materials 5.1), obtaining a dataset with protein name, UniProt ID, protein length and amino acid sequence. We then used the information from QUADRatlas [3] to add the target feature of being an RG4-binding protein to each of the UniProt proteins. Next, we processed the information regarding domains and regions. Since the latter, although extremely numerous, had few representatives per category (median of 2 proteins per regions type), we decided to only consider the presence of disordered regions. Domains were instead processed in the following way: first, we grouped domains belonging to the same family under a single ID; then, each ID was associated with a number to convert the positional information from UniProt into strings containing sequences of numbers corresponding to the various domains. The process is depicted in Figure 4.1.

After adding the information on domains and regions, we proceeded with that concerning the protein secondary structure. To do this, we first downloaded the PDB-format structure of the entire human proteome as predicted by AlphaFold [24, 47]. We then processed these files (see Supplementary Materials 5.2) to obtain strings representing the secondary structure, which we added to the previously constructed dataset.

At this point, we removed proteins already known to be RBPs but labelled as non-RG4BPs from the dataset (the RBPs lists were obtained merging results from [4, 33, 37, 13, 35, 46, 45]); since many RG4BPs are also RBPs, removing them reduced the false negative rate within the starting dataset, thus improving the chances of building accurate models. With this last step, we obtained the starting dataset; subsequent modifications were different depending on the model to be trained.

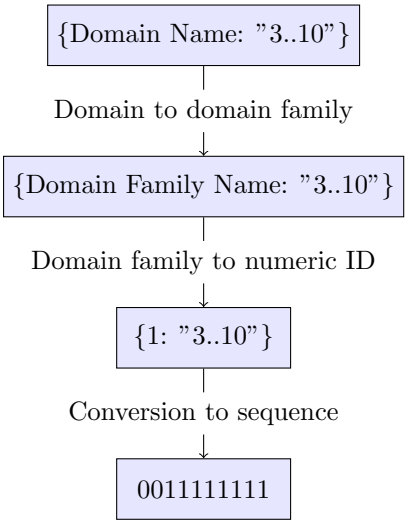

Figure 4.1: Domain processing.

4.2 Structured Data Classification

For this approach, we processed the dataset in order to obtain a tabular format file containing information on a number of features for each sample (protein). In particular, through a feature engineering process we obtained the following attributes:

- Protein length;
- Percentage of disordered structure;
- The IDs of the first two domains. The threshold of domains considered was chosen according to the statistic in the table below:

Number of domains	Number of proteins
1	4160
2	770
3	176
4	50
5	3
6	1
7	1

- The percentage of the protein characterised by a certain secondary structure element. More precisely, we added a column for each of the following one-letter secondary structure codes obtained from stride [18]:

Symbol	Secondary structure element
H	α-helix
G	3_{10} helix
I	π-helix
E	Extended conformation
B or b	Isolated bridge
T	Turn
C	Coil (none of the above

- Label, i.e. whether the protein is known to be an RG4BP or not.

As a first approach, we built a predictive model using TabPFN [21]. Since this can handle datasets containing at most 1000 samples, we built a subdataset by randomly extracting 500 positives and 500 negatives from the whole dataset. We then trained the model on the training dataset (75% of the whole dataset) and tested the performance on the test dataset (25% of the whole dataset), obtaining an accuracy of 0.696 and the confusion matrix shown in Figure 4.2.

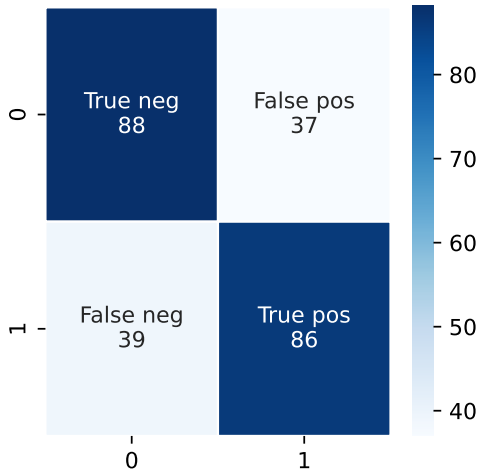

Figure 4.2: Confusion matrix representing the best prediction obtained with the TabPFN model. Accuracy: 0.696.

Given the promising results, we decided to proceed using AutoKeras [23] to automate the process of building and training a model. AutoKeras allows to automatically test an array of architectures and hyper-parameters values, thereby giving us indications regarding the best choices of these values for our problem (see Supplementary Materials, Figure 5.1). We then reproduced the obtained model in Keras [6] in order to perform a further fine-tuning of the parameters. To assess the impact of different feature processing methods on the performance of the model, we decided to study two separate cases. In the first case (model A), we processed the categorical variables by one-hot encoding, while in the second one (model B) we used an embedding layer.

Since embedding requires the size of the output vocabulary, we performed a PCA on the protein domains (see Figure 4.3), ending up choosing a vector size of 20.

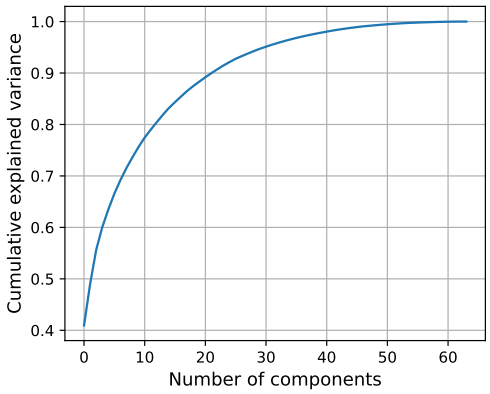

Figure 4.3: Results of a Principal Component Analysis (PCA) applied to protein domains. 20 components are enough to explain 88 % of the variance.

We then built the two models, obtaining the architectures represented in Figure 5.2 and Listing 5.4 of the Supplementary Materials. We first proceeded with the training and evaluation phases of the two neural networks, obtaining the results shown in Table 4.1 and Figure 4.4.

Model A		Model B	
Accuracy	Loss	Accuracy	Loss
0.86	0.537	0.826	0.528

Table 4.1: Performance of models A and B.

Then, once the training of the two models was completed, we used them to make a prediction on the complete dataset, i.e. the one also containing the proteins already known to be RBPs. The results are shown in the confusion matrices depicted in Figure 4.5 and in the distribution histograms reported in Figure 4.6. Both models are characterised by a certain difficulty in minimising the validation loss, which is not too surprising given the nature of the dataset. An interesting difference between the two models concerns the accuracy, which is used as the main evaluation metric during the training phase: while in model A we observe a certain discrepancy between training and validation accuracy (consistent with the high validation loss), in model B this discrepancy is small, indicating a lower degree of overfitting. However, overall, Model A performs slightly better than model B, with an accuracy of 0.86 (compared to 0.826 for model B).

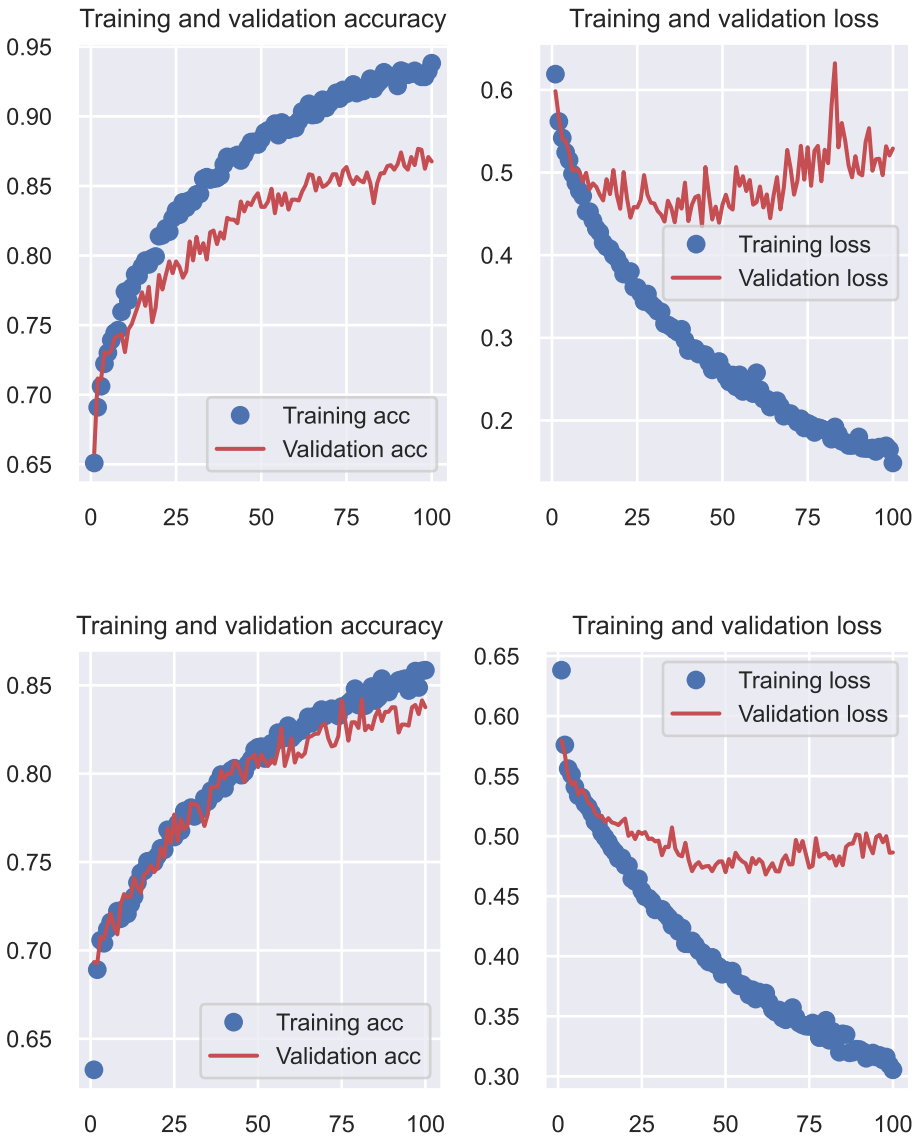

Figure 4.4: Top: training process of model A. Bottom: training process of model B.

Since the aim of the work is to find potential RG4BPs that are not yet known, we only kept the false positives (FP) and merged those predicted by the two models in a single array. In other words, we built a dictionary whose keys were the IDs of the false positives, and whose values were a list containing a combination of the numbers 1 and 2 depending on whether the false positive in question was predicted by model A, model B or both. Since false positives were present in a large number, we explored different thresholds to adjust the prediction outcomes. Specifically, we applied the thresholds shown in Table 4.2 in order to identify a suitable threshold that would reduce the number of positive predictions while maintaining the confidence level of the model's outputs.

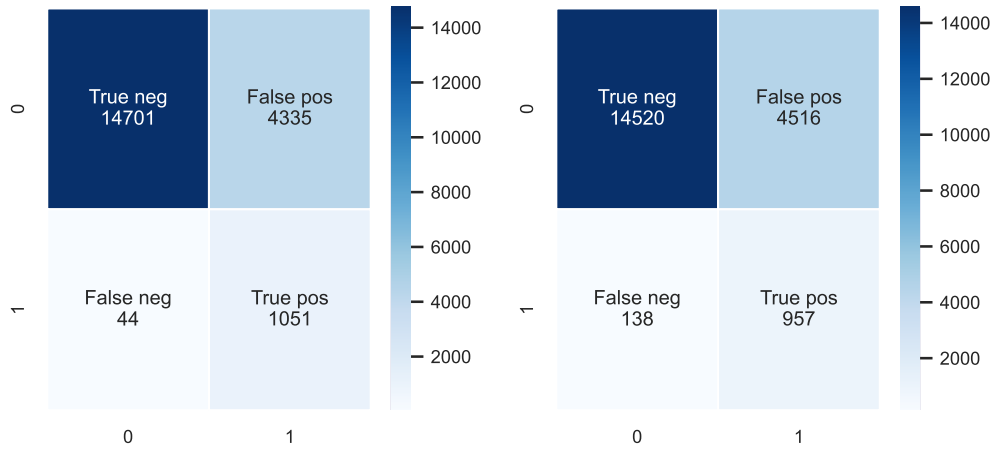

Figure 4.5: Left: confusion matrix of the prediction performed by model A on the full dataset. Accuracy: 0.86. Right: confusion matrix of the prediction performed by model B on the same dataset. Accuracy: 0.826.

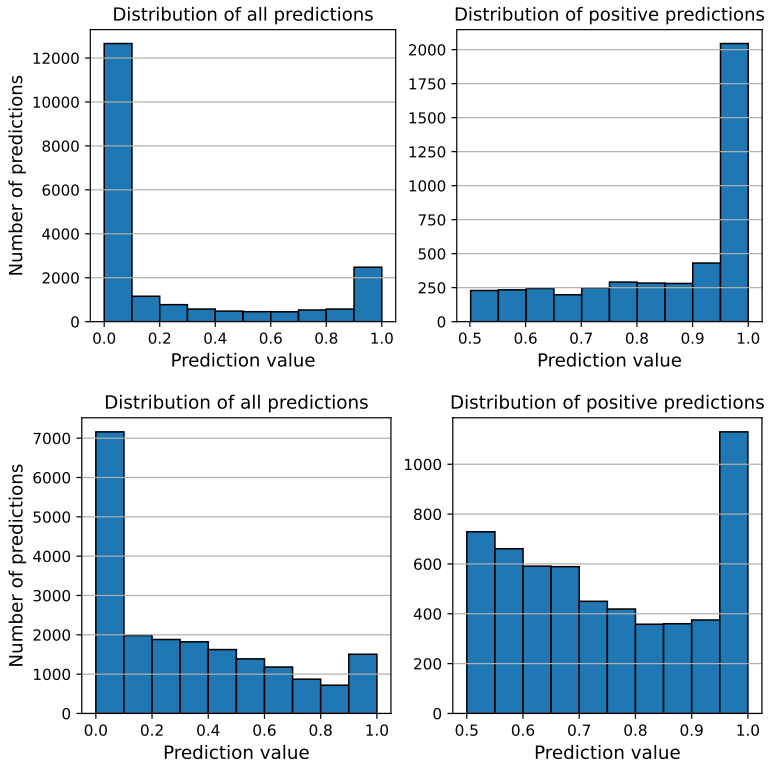

Figure 4.6: Top: histogram showing the distributions of all predictions (left) and the positive ones (right) for model A. Bottom: histogram showing the distributions of all predictions (left) and the positive ones (right) for model B.

Treshold	# FP
1.00	218
0.95	1586
0.90	2066
0.75	3251

Table 4.2: Common false positives predicted by models A and B at different thresholds.

4.3 Image Classification

For this second approach, we started from the idea that an image is de facto a two-dimensional array of integers, and looked for a way to process our dataset in order to convert it into a data structure formally analogous to an image. To do this, we first created two directories in which to store data for positives and negatives examples separately. We then processed the dataset in such a way as to save the information relative to each sample in the form of a numpy array. Specifically, each array consists of four rows, representing sequence, secondary structure, domains and regions respectively, and as many columns as are amino acids composing the protein in question. The following criterion was followed for the representation of the four features:

- The presence of a disordered region was indicated with a number of '1's equal to its extension in amino acids, while its absence was similarly indicated with '0's;
- Domains were represented similarly to regions, but using the ID of the corresponding domain family (numbers 1 to 799) as identifier;
- The single-letter codes representing amino acids were replaced with the numbers ranging from 800 to 819;
- The letters representing secondary structure elements were replaced with the numbers ranging from 819 to 825.

Although the use of large numbers places a computational burden on the training of the models, we opted for this option in order to avoid the introduction of artifacts due to the presence of spurious patterns. Since most image classification models require images of the same size, we padded each array with zeroes in order to obtain homogeneity along both dimensions. Once the data processing phase was completed, we used AutoKeras to build an image classification model (model C), thereby obtaining the architecture represented in Figure 5.3 of Supplementary Materials.

We then performed a prediction on the entire dataset, obtaining the result represented in Figure 4.7. The distribution of the predictions is represented in Figure 4.8. It should be taken into account that for time reasons it was not possible to use Keras to optimise the image classification model as was done for the structured data classification models. This absence of the fine-tuning process explains, at least in part, the relatively poor performance of the model.

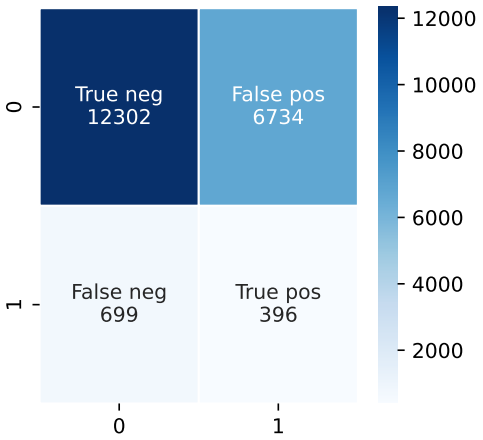

Figure 4.7: Confusion matrix representing the best prediction obtained with the AutoKeras model. Accuracy: 0.63.

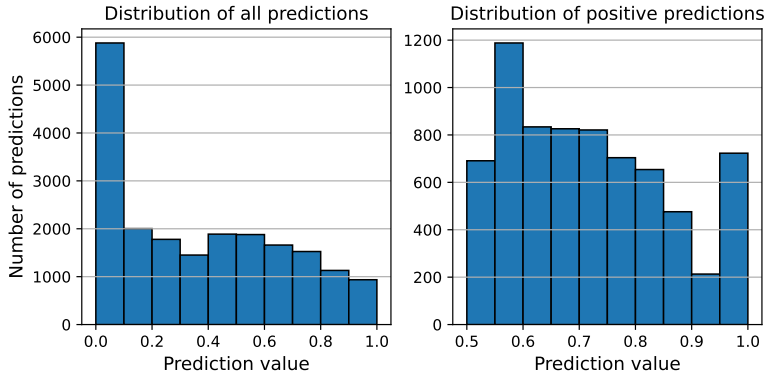

Figure 4.8: Histogram showing the distributions of all predictions (left) and the positive ones (right) obtained with the image classification model.

Similar to what we did with models A and B, we then calculated the number of false positives at various thresholds, obtaining the results shown in Table 4.3.

Treshold	# FP
1.00	0
0.95	679
0.90	879
0.75	2612

Table 4.3: False positives predicted by model C at different thresholds.

4.4 Model Comparison

Once we obtained the predictions with the three models (A, B and C), we analysed the fraction of common genes at different thresholds, as represented in Figure 4.9. Clearly, as the threshold increases, the contribution of the image classification model becomes progressively less relevant, which is consistent with the distribution of predictions represented in Figure 4.8. On the contrary, the predictions of models A and B appear to be much more overlapping, with 31 common predictions at the 100% threshold.

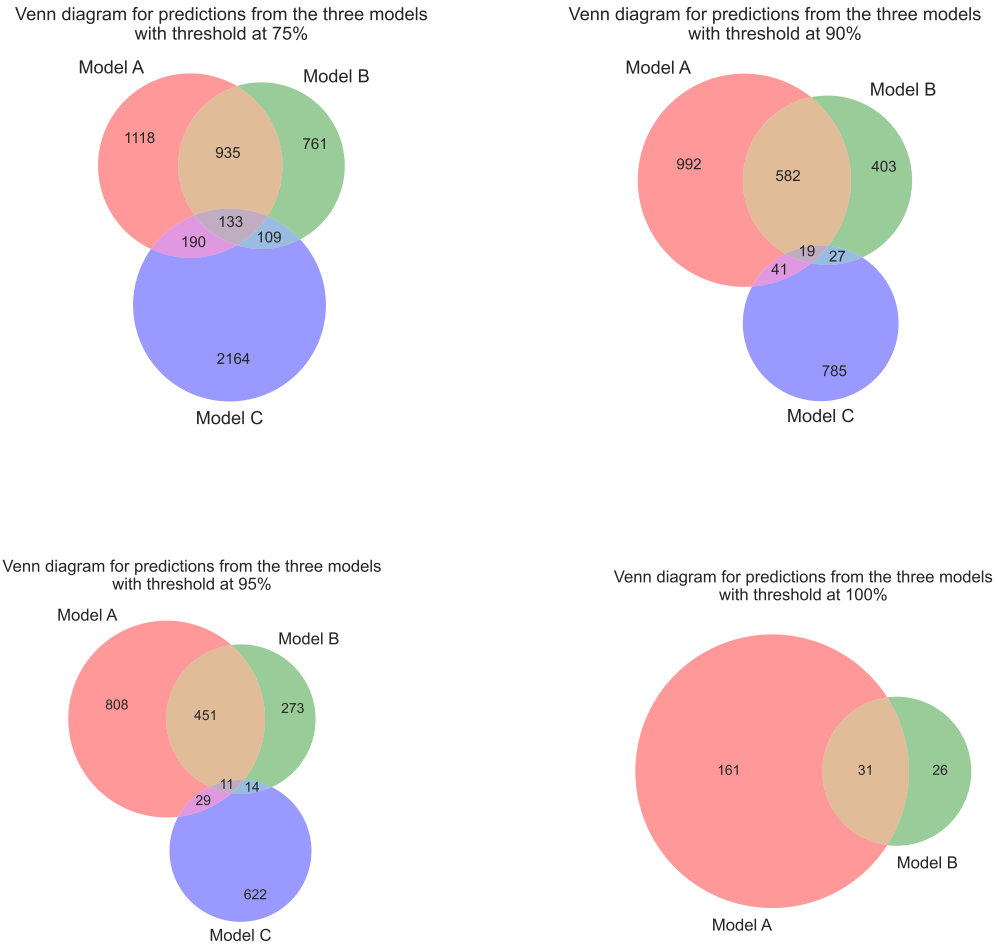

Figure 4.9: Venn diagrams representing the gene fractions common to the three models at different thresholds. Models A and B are the structured data classifier using one-hot encoding and embedding, respectively; model C is the image classification model.

Chapter 5

Discussion and Conclusions

In this work we presented a computational approach to the identification of RNA G-quadruplex-binding proteins (RG4BPs), with the aim of supporting our understanding of the regulation of these cis elements by guiding experimental analyses. We first constructed a starting dataset by combining information from UniProt [7], then we integrated it with those derived from QUADRatlas [3] in order to have a pool of true positive proteins (i.e. proteins already known to be true RG4 binders). After a data processing phase, we started the construction of the classification models, following two main paths: structured data and image classification.

In the first approach we identified a number of features, including protein sequence, structural composition and presence of domains and regions, and saved the data in tabular form using pandas DataFrames [43]. We then used over- and undersampling techniques to reduce imbalances in the dataset and thus facilitate the training phase. Indeed, imbalance was one of the biggest limits of our dataset, as known positives were only around 1100 out of 20000 samples. We then removed proteins already known to be RBPs from the negatives; as RG4BPs are a subclass of RBPs, removing them reduced the number of false negatives. Then, we started the model building phase: after obtaining parameter estimates by means of AutoKeras [23], we moved on to their fine-tuning using Keras [6], finally obtaining models A and B with the performances shown in Table 4.1. Although the accuracy of the two models is rather high (see Figure 4.5), in both cases considerable difficulties in minimising the cost function were encountered, which resulted in non-optimised models. This problem, although partly related to the nature of the dataset, could certainly be mitigated through a more precise fine-tuning process. It is also possible that the features selected for training do not contain sufficient information to accurately discriminate between positives and negatives, hence the use of new features processed in a different way could be useful. Nevertheless, the fact that the two models resulted in 31 common predictions using a 100% threshold suggests that they were able to recognise a set of key features common to RG4BPs, which may make it worth investing some time in an optimisation process.

In parallel, we followed an image classification approach. Starting from the same dataset, we processed the data to obtain a formally image-like representation, i.e. a two-dimensional array of integers. In this case, we considered amino acid sequence, secondary structure, domains and regions, thus representing each protein with an array of size $4 \times l$, where l is the number of amino acids. Since proteins differ in length, we obtained a set of arrays heterogeneous for the second dimension. We therefore introduced a further processing step and padded all the matrices with zeros, thus obtaining a dataset ready to be fed into a convolutional neural network

model. Although there are other techniques for handling inconsistencies of this type, we opted for padding with zeros because of its simplicity, even though we are aware that such an operation could introduce artefacts. Having done so, we used AutoKeras to construct a neural network, resulting in the C model with the performance shown in Figure 4.7. Clearly, the model obtained is rather poorly performing, with an accuracy slightly above 60%. The underlying reason for this result is to be found within several factors. Firstly, for reasons of time we were not able to carry out the fine-tuning process via Keras followed for the other two models, de facto using the non-optimised hyperparameters obtained via AutoKeras. Secondly, the processing of the dataset was rather heavy, and may have resulted in the introduction of spurious patterns. This last aspect could be improved through the use of alternative processing techniques, so that the preprocessing phase would impact the original data as little as possible.

We finally compared the predictions of the three models, obtaining the results shown in Figure 4.9. Despite the sub-optimal performance of model C, the intersection of the results of models A and B allowed the identification of a small number of proteins potentially capable of binding RG4 motifs. Two of these, DDX55 and CPEB4, are known to be RBPs and their binding sites are frequently overlapping with RG4 elements ($> 50\%$). This suggests that the models are actually able to find patterns useful for identifying novel RG4BPs.

5.1 Future Perspectives

Despite the rather promising results, there certainly remains room for improvement. First of all, the image classification model obtained with AutoKeras was not subjected to the same fine-tuning procedure via Keras as the other two, which explains at least in part the poor performance. An improvement of the hyper-parameters tuning would probably result in an increase in performance, with a consequently higher contribution to the predictions obtained by means of models A and B. Secondly, in order to overcome the problems arising from the high unbalance of the dataset, we used over- and undersampling techniques which, although optimised, may have introduced a bias within the models. A potential alternative route consists of using synthetic data, so as to increase the number of true positives and thus be able to reduce the problem to its root. Finally, it must be taken into account that the results are purely the product of predictions and need to be confirmed experimentally. In this regard, laboratory validation of the two most promising predictions, i.e. DDX55 and CPEB4, is currently underway by means of affinity chromatography assays with synthetic RG4 sequences.

Bibliography

[1] Chiara Ambrosini, Francesca Garilli, and Alessandro Quattrone. "Chapter Thirteen - Reprogramming translation for gene therapy". In: *Curing Genetic Diseases Through Genome Reprogramming*. Ed. by Gianluca Petris. Vol. 182. Progress in Molecular Biology and Translational Science. Academic Press, 2021, pp. 439–476. DOI: https://doi.org/10.1016/bs.pmbts.2021.01.028. URL: https://www.sciencedirect.com/science/article/pii/S1877117321000399.

[2] Debmalya Bhattacharyya, Gayan Mirihana Arachchilage, and Soumitra Basu. "Metal Cations in G-Quadruplex Folding and Stability". In: *Frontiers in Chemistry* 4 (Sept. 2016). DOI: 10.3389/fchem.2016.00038. URL: https://doi.org/10.3389/fchem.2016.00038.

[3] Sébastien Bourdon et al. "QUADRatlas: the RNA G-quadruplex and RG4-binding proteins database". In: *Nucleic Acids Research* 51.D1 (Sept. 2022), pp. D240–D247. DOI: 10.1093/nar/gkac782. URL: https://doi.org/10.1093/nar/gkac782.

[4] Alfredo Castello et al. "Identification of RNA-binding domains of RNA-binding proteins in cultured cells on a system-wide scale with RBDmap". In: *Nature Protocols* 12.12 (Nov. 2017), pp. 2447–2464. DOI: 10.1038/nprot.2017.106. URL: https://doi.org/10.1038/nprot.2017.106.

[5] Jun Cheng et al. "iCis/i-regulatory elements explain most of the mRNA stability variation across genes in yeast". In: *RNA* 23.11 (Aug. 2017), pp. 1648–1659. DOI: 10.1261/rna.062224.117. URL: https://doi.org/10.1261/rna.062224.117.

[6] François Chollet et al. *Keras*. https://keras.io. 2015.

[7] The UniProt Consortium. "UniProt: the Universal Protein Knowledgebase in 2023". In: *Nucleic Acids Research* 51.D1 (Nov. 2022), pp. D523–D531. ISSN: 0305-1048. DOI: 10.1093/nar/gkac1052. eprint: https://academic.oup.com/nar/article-pdf/51/D1/D523/48441158/gkac1052.pdf. URL: https://doi.org/10.1093/nar/gkac1052.

[8] Meredith Corley, Margaret C. Burns, and Gene W. Yeo. "How RNA-Binding Proteins Interact with RNA: Molecules and Mechanisms". In: *Molecular Cell* 78.1 (Apr. 2020), pp. 9–29. DOI: 10.1016/j.molcel.2020.03.011. URL: https://doi.org/10.1016/j.molcel.2020.03.011.

[9] Hassan Dana et al. "Molecular mechanisms and biological functions of siRNA". en. In: *Int. J. Biomed. Sci.* 13.2 (June 2017), pp. 48–57.

[10] Leïla Dumas et al. "G-Quadruplexes in RNA Biology: Recent Advances and Future Directions". In: *Trends in Biochemical Sciences* 46.4 (Apr. 2021), pp. 270–283. DOI: 10.1016/j.tibs.2020.11.001. URL: https://doi.org/10.1016/j.tibs.2020.11.001.

[11] Irina A. Elcheva and Vladimir S. Spiegelman. "The Role of cis- and trans-Acting RNA Regulatory Elements in Leukemia". In: *Cancers* 12.12 (Dec. 2020), p. 3854. DOI: 10.3390/cancers12123854. URL: https://doi.org/10.3390/cancers12123854.

[12] Marta M. Fay, Shawn M. Lyons, and Pavel Ivanov. "RNA G-Quadruplexes in Biology: Principles and Molecular Mechanisms". In: *Journal of Molecular Biology* 429.14 (July 2017), pp. 2127–2147. DOI: 10.1016/j.jmb.2017.05.017. URL: https://doi.org/10.1016/j.jmb.2017.05.017.

[13] Stefanie Gerstberger, Markus Hafner, and Thomas Tuschl. "A census of human RNA-binding proteins". In: *Nature Reviews Genetics* 15.12 (Nov. 2014), pp. 829–845. DOI: 10.1038/nrg3813. URL: https://doi.org/10.1038/nrg3813.

[14] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. URL: %5Curl%7Bhttp://www.deeplearningbook.org%7D.

[15] R. E. Halbeisen et al. "Post-transcriptional gene regulation: From genome-wide studies to principles". In: *Cellular and Molecular Life Sciences* 65.5 (Nov. 2007). DOI: 10.1007/s00018-007-7447-6. URL: https://doi.org/10.1007/s00018-007-7447-6.

[16] Charles R. Harris et al. "Array programming with NumPy". In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[17] Haibo He and E.A. Garcia. "Learning from Imbalanced Data". In: *Knowledge and Data Engineering, IEEE Transactions on* 21 (Oct. 2009), pp. 1263–1284. DOI: 10.1109/TKDE.2008.239.

[18] M. Heinig and D. Frishman. "STRIDE: a web server for secondary structure assignment from known atomic coordinates of proteins". In: *Nucleic Acids Research* 32.Web Server (July 2004), W500–W502. DOI: 10.1093/nar/gkh429. URL: https://doi.org/10.1093/nar/gkh429.

[19] Christopher U.T. Hellen. "Translation Termination and Ribosome Recycling in Eukaryotes". In: *Cold Spring Harbor Perspectives in Biology* 10.10 (May 2018), a032656. DOI: 10.1101/cshperspect.a032656. URL: https://doi.org/10.1101/cshperspect.a032656.

[20] Eric Henderson et al. "Telomeric DNA oligonucleotides form novel intramolecular structures containing guanine·guanine base pairs". In: *Cell* 51.6 (Dec. 1987), pp. 899–908. DOI: 10.1016/0092-8674(87)90577-0. URL: https://doi.org/10.1016/0092-8674(87)90577-0.

[21] Noah Hollmann et al. *TabPFN: A Transformer That Solves Small Tabular Classification Problems in a Second*. 2022. DOI: 10.48550/ARXIV.2207.01848. URL: https://arxiv.org/abs/2207.01848.

[22] Aino I. Järvelin et al. "The new (dis)order in RNA regulation". In: *Cell Communication and Signaling* 14.1 (Apr. 2016). DOI: 10.1186/s12964-016-0132-3. URL: https://doi.org/10.1186/s12964-016-0132-3.

[23] Haifeng Jin et al. "AutoKeras: An AutoML Library for Deep Learning". In: *Journal of Machine Learning Research* 24.6 (2023), pp. 1–6. URL: http://jmlr.org/papers/v24/20-1355.html.

[24] John Jumper et al. "Highly accurate protein structure prediction with AlphaFold". In: *Nature* 596.7873 (July 2021), pp. 583–589. DOI: 10.1038/s41586-021-03819-2. URL: https://doi.org/10.1038/s41586-021-03819-2.

[25] Prakash Kharel et al. "Properties and biological impact of RNA G-quadruplexes: from order to turmoil and back". In: *Nucleic Acids Research* 48.22 (Dec. 2020), pp. 12534–12555. DOI: 10.1093/nar/gkaa1126. URL: https://doi.org/10.1093/nar/gkaa1126.

[26] Chun Kit Kwok, Giovanni Marsico, and Shankar Balasubramanian. "Detecting RNA G-Quadruplexes (rG4s) in the Transcriptome". In: *Cold Spring Harbor Perspectives in Biology* 10.7 (July 2018), a032284. DOI: 10.1101/cshperspect.a032284. URL: https://doi.org/10.1101/cshperspect.a032284.

[27] Thomas X. Lu and Marc E. Rothenberg. "Diagnostic, functional, and therapeutic roles of microRNA in allergic diseases". In: *Journal of Allergy and Clinical Immunology* 132.1 (July 2013), pp. 3–13. DOI: 10.1016/j.jaci.2013.04.039. URL: https://doi.org/10.1016/j.jaci.2013.04.039.

[28] Thomas X. Lu and Marc E. Rothenberg. "MicroRNA". In: *Journal of Allergy and Clinical Immunology* 141.4 (Apr. 2018), pp. 1202–1207. DOI: 10.1016/j.jaci.2017.08.034. URL: https://doi.org/10.1016/j.jaci.2017.08.034.

[29] Luciano E. Marasco and Alberto R. Kornblihtt. "The physiology of alternative splicing". In: *Nature Reviews Molecular Cell Biology* 24.4 (Oct. 2022), pp. 242–254. DOI: 10.1038/s41580-022-00545-z. URL: https://doi.org/10.1038/s41580-022-00545-z.

[30] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[31] Stefania Millevoi, Hervé Moine, and Stéphan Vagner. "G-quadruplexes in RNA biology". In: *Wiley Interdisciplinary Reviews: RNA* 3.4 (Apr. 2012), pp. 495–507. DOI: 10.1002/wrna.1113. URL: https://doi.org/10.1002/wrna.1113.

[32] Simona Panni et al. "Non-coding RNA regulatory networks". In: *Biochimica et Biophysica Acta (BBA) - Gene Regulatory Mechanisms* 1863.6 (June 2020), p. 194417. DOI: 10.1016/j.bbagrm.2019.194417. URL: https://doi.org/10.1016/j.bbagrm.2019.194417.

[33] Joel I. Perez-Perri et al. "Discovery of RNA-binding proteins and characterization of their dynamic responses by enhanced RNA interactome capture". In: *Nature Communications* 9.1 (Oct. 2018). DOI: 10.1038/s41467-018-06557-8. URL: https://doi.org/10.1038/s41467-018-06557-8.

[34] Monica J. Piatek and Andreas Werner. "Endogenous siRNAs: regulators of internal affairs". In: *Biochemical Society Transactions* 42.4 (Aug. 2014), pp. 1174–1179. DOI: 10.1042/bst20140068. URL: https://doi.org/10.1042/bst20140068.

[35] Rayner M. L. Queiroz et al. "Comprehensive identification of RNA–protein interactions in any organism using orthogonal organic phase separation (OOPS)". In: *Nature Biotechnology* 37.2 (Jan. 2019), pp. 169–178. DOI: 10.1038/s41587-018-0001-2. URL: https://doi.org/10.1038/s41587-018-0001-2.

[36] Janosh Riebesell. *Random TikZ Collection*. Computer program. Version 0.1.0. Dec. 27, 2022. DOI: 10.5281/zenodo.7486911. URL: https://github.com/janosh/tikz.

[37] Endre Sebestyén et al. "Large-scale analysis of genome and transcriptome alterations in multiple tumors unveils novel cancer-relevant splicing networks". In: *Genome Research* 26.6 (Apr. 2016), pp. 732–744. DOI: 10.1101/gr.199935.115. URL: https://doi.org/10.1101/gr.199935.115.

[38] Dipankar Sen and Walter Gilbert. "Formation of parallel four-stranded complexes by guanine-rich motifs in DNA and its implications for meiosis". In: *Nature* 334.6180 (July 1988), pp. 364–366. DOI: 10.1038/334364a0. URL: https://doi.org/10.1038/334364a0.

[39] Jonathon Shlens. *A Tutorial on Principal Component Analysis*. 2014. arXiv: 1404.1100 [cs.LG].

[40] Ahmed Sidali et al. "AU-Rich Element RNA Binding Proteins: At the Crossroads of Post-Transcriptional Regulation and Genome Integrity". In: *International Journal of Molecular Sciences* 23.1 (Dec. 2021), p. 96. DOI: 10.3390/ijms23010096. URL: https://doi.org/10.3390/ijms23010096.

[41] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[42] Martina Tassinari, Sara N Richter, and Paolo Gandellini. "Biological relevance and therapeutic potential of G-quadruplex structures in the human noncoding transcriptome". In: *Nucleic Acids Research* 49.7 (Mar. 2021), pp. 3617–3633. DOI: 10.1093/nar/gkab127. URL: https://doi.org/10.1093/nar/gkab127.

[43] The pandas development team. *pandas-dev/pandas: Pandas*. Version 1.5.1. Feb. 2020. DOI: 10.5281/zenodo.3509134. URL: https://doi.org/10.5281/zenodo.3509134.

[44] Bin Tian and James L. Manley. "Alternative polyadenylation of mRNA precursors". In: *Nature Reviews Molecular Cell Biology* 18.1 (Sept. 2016), pp. 18–30. DOI: 10.1038/nrm.2016.116. URL: https://doi.org/10.1038/nrm.2016.116.

[45] Jakob Trendel et al. "The Human RNA-Binding Proteome and Its Dynamics during Translational Arrest". In: *Cell* 176.1-2 (Jan. 2019), 391–403.e19. DOI: 10.1016/j.cell.2018.11.004. URL: https://doi.org/10.1016/j.cell.2018.11.004.

[46] Erika C. Urdaneta et al. "Purification of cross-linked RNA-protein complexes by phenol-toluol extraction". In: *Nature Communications* 10.1 (Mar. 2019). DOI: 10.1038/s41467-019-08942-3. URL: https://doi.org/10.1038/s41467-019-08942-3.

[47] Mihaly Varadi et al. "AlphaFold Protein Structure Database: massively expanding the structural coverage of protein-sequence space with high-accuracy models". In: *Nucleic Acids Research* 50.D1 (Nov. 2021), pp. D439–D444. DOI: 10.1093/nar/gkab1061. URL: https://doi.org/10.1093/nar/gkab1061.

[48] Muhamad Yani, M.T. Budhi Irawan S Si., and M.T. Casi Setiningsih S.T. "Application of Transfer Learning Using Convolutional Neural Network Method for Early Detection of Terry's Nail". In: *Journal of Physics: Conference Series* 1201.1 (May 2019), p. 012052. DOI: 10.1088/1742-6596/1201/1/012052. URL: https://doi.org/10.1088/1742-6596/1201/1/012052.

Supplementary Materials

```
    import requests, sys, json
    import pandas as pd
    import matplotlib.pyplot as plt
    import seaborn as sns
    if sys.version_info[0] < 3:
        from StringIO import StringIO
    else:
        from io import StringIO

    # Documentation: https://www.uniprot.org/help/api
    WEBSITE_API = "https://rest.uniprot.org/"

    # Documentation: https://www.ebi.ac.uk/proteins/api/doc/
    PROTEINS_API = "https://www.ebi.ac.uk/proteins/api"

    # Helper function to download data
    def get_url(url, **kwargs):
      response = requests.get(url, **kwargs);

      if not response.ok:
        print(response.text)
        response.raise_for_status()
        sys.exit()

      return response

    r = get_url(f"""{WEBSITE_API}/uniprotkb/stream?query=
    (taxonomy_id:10090) AND (reviewed:true)&fields=id,accession,
    length,sequence&format=tsv""")

    p = StringIO(r.text)
    df = pd.read_csv(p, sep="\t")
```

Listing 5.1: Programmatic access to Uniprot.

```
import os
import pandas as pd
from tqdm.notebook import tqdm_notebook
import tqdm

my_dir = '/home/rob/Desktop/try with stride/PDB_files'
def parse_txt_file(my_file):
    with open(my_file) as myfile:
        content = myfile.readlines()
        my_str = ''
        for el in content:
            if "Detailed secondary structure" in el:
                my_str += el
        content = content[content.index(my_str)+3:]
        new_list = []
        for el in content:
            new_list.append(el.split())
        struct_string = "".join([el[5] for el in new_list])

    return struct_string

def get_structures(path):
    file_count = sum(len(files) for _, _, files in os.walk(path))-3
    structures = []
    # creating progress bar
    with tqdm.notebook.tqdm(total=file_count) as pbar:
        for filename in os.listdir(path):
            ext = os.path.splitext(filename)[-1].lower()
            if ext == ".pdb":
                cmd = "./stride {fname} -f{output}".format(
                fname = filename, output = "prova.txt")
                so = os.popen(cmd).read()

                # open txt file
                struct_string = parse_txt_file("prova.txt")
                structures.append((filename.split("-")[1], struct_string))
                pbar.update(1)
    return structures

df = pd.DataFrame(get_structures(my_dir))
```

Listing 5.2: Protein secondary structure from PDB files.

```
    df = pd.read_excel(my_file, index_col=None, na_values=['NA'])

    # Define dependent variable that needs to be predicted
    y = df["isRG4BP"].values

    # Define independent variable
    x = df.drop("isRG4BP", axis = 1)

    # Undersampling
    from collections import Counter
    from imblearn.under_sampling import RandomUnderSampler

    rus = RandomUnderSampler(sampling_strategy=0.2,random_state=42)
    """
    "sampling_strategy" is the desired ratio of the number of samples in the minority
    class over the number of samples in the majority class after resampling
    """

    x_under_sampled, y_under_sampled = rus.fit_resample(x, y)

    print('Original dataset shape %s' % Counter(y))
    print('Resampled dataset shape %s' % Counter(y_under_sampled))

    # Oversampling
    from imblearn.over_sampling import RandomOverSampler
    ros = RandomOverSampler(random_state=0)

    x_over_sampled, y_over_sampled = ros.fit_resample(x_under_sampled, y_under_sampled)
    df = pd.concat([x_over_sampled, y_over_sampled], axis=1)
```

Listing 5.3: Over- and undersampling technique used to reduce the imbalances between positives and negatives in the dataset.

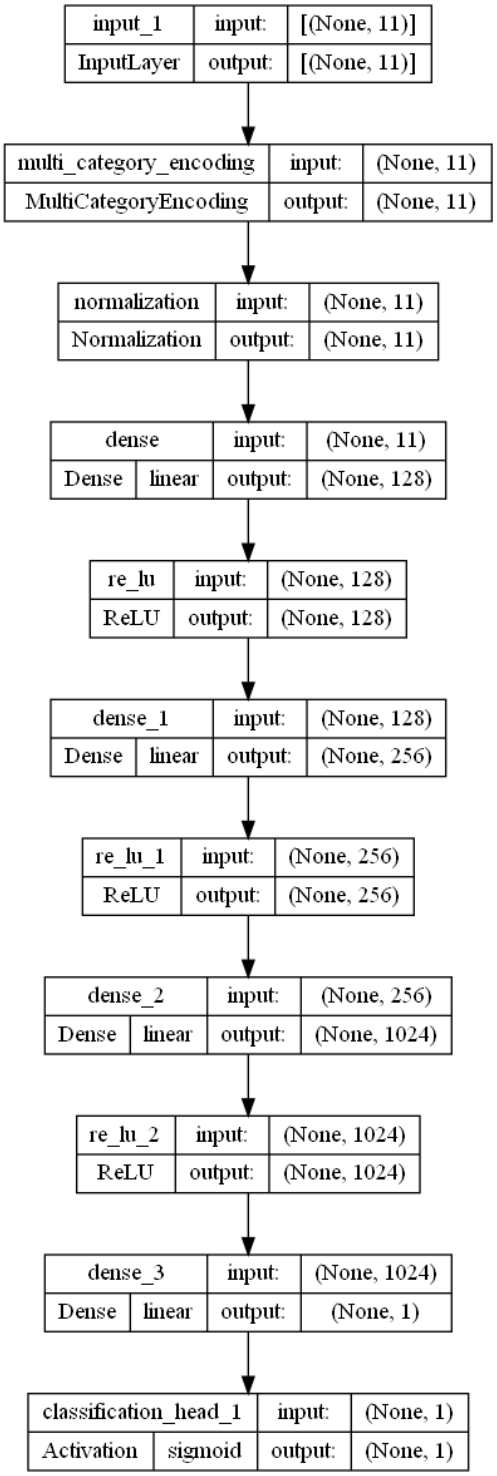

Figure 5.1: Architecture of the model obtained using AutoKeras.

```
import keras
import tensorflow as tf
import pandas as pd
import numpy as np
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split
from tensorflow.keras.layers import IntegerLookup
from tensorflow.keras.layers import Normalization
from tensorflow.keras.layers import StringLookup
from keras.layers import Embedding, Input, Flatten, Dense, BatchNormalization

my_file = "Dataset_no_RBPs_25_12.xlsx"
df = pd.read_excel(my_file, index_col=None, na_values=['NA'])
df["isRG4BP"] = df["isRG4BP"].astype(int) #convert true/false to 1/0
df = shuffle(df) #shuffle rows

# The part where the dataset was over- and undersampled to reduce the
# imabalances is not shown here

# Splitting into training, validation and test set
test_dataframe = df.sample(frac=0.3, random_state=1337)
train_dataframe = df.drop(test_dataframe.index)
val_dataframe = train_dataframe.sample(frac=0.2, random_state=1337)
train_dataframe = train_dataframe.drop(val_dataframe.index)

# Generating tf.data.Dataset objects for each dataframe
def dataframe_to_dataset(dataframe, batch_size=64):
    dataframe = dataframe.copy()
    labels = dataframe.pop("isRG4BP")
    ds = tf.data.Dataset.from_tensor_slices((dict(dataframe), labels))
    ds = ds.shuffle(buffer_size=len(dataframe))
    ds = ds.batch(batch_size)
    ds = ds.prefetch(batch_size)
    return ds

batch_size = 32;
test_ds = dataframe_to_dataset(test_dataframe, batch_size=batch_size)
train_ds = dataframe_to_dataset(train_dataframe, batch_size=batch_size)
val_ds = dataframe_to_dataset(val_dataframe, batch_size=batch_size)

def encode_numerical_feature(feature, name, dataset):
    # Create a Normalization layer for our feature
    normalizer = Normalization()

    # Prepare a Dataset that only yields our feature
    feature_ds = dataset.map(lambda x, y: x[name])
    feature_ds = feature_ds.map(lambda x: tf.expand_dims(x, -1))

    # Learn the statistics of the data
    normalizer.adapt(feature_ds)

    # Normalize the input feature
    encoded_feature = normalizer(feature)
    return encoded_feature

def encode_categorical_feature(feature, name, dataset, is_string):
    lookup_class = StringLookup if is_string else IntegerLookup
    # in this case we'll only use integerlookup
    # Create a lookup layer which will turn strings into integer indices
    lookup = lookup_class(output_mode="binary")

    # Prepare a Dataset that only yields our feature
```

```
    feature_ds = dataset.map(lambda x, y: x[name])
    feature_ds = feature_ds.map(lambda x: tf.expand_dims(x, -1))

    # Learn the set of possible string values and assign them a fixed integer index
    lookup.adapt(feature_ds)

    # Turn the string input into integer indices
    encoded_feature = lookup(feature)
    return encoded_feature

# Categorical features encoded as integers
dom1 = keras.Input(shape=(1,), name="dom1", dtype="int64")
dom2 = keras.Input(shape=(1,), name="dom2", dtype="int64")

# Numerical features
length = keras.Input(shape=(1,), name="Length")
disordered_perc = keras.Input(shape=(1,), name="% disordered")
C_perc = keras.Input(shape=(1,), name="% C")
H_perc = keras.Input(shape=(1,), name="% H")
G_perc = keras.Input(shape=(1,), name="% G")
I_perc = keras.Input(shape=(1,), name="% I")
E_perc = keras.Input(shape=(1,), name="% E")
B_perc = keras.Input(shape=(1,), name="% B")
T_perc = keras.Input(shape=(1,), name="% T")

all_inputs = [dom1, dom2, length, disordered_perc, C_perc, H_perc, G_perc, I_perc,
    E_perc, B_perc, T_perc]

# Integer categorical features
dom1_encoded = encode_categorical_feature(dom1, "dom1", train_ds, False)
dom2_encoded = encode_categorical_feature(dom2, "dom2", train_ds, False)

# Numerical features
length_encoded = encode_numerical_feature(length, "Length", train_ds)
disordered_perc_encoded = encode_numerical_feature(disordered_perc, "% disordered",
    train_ds)
C_perc_encoded = encode_numerical_feature(C_perc, "% C", train_ds)
H_perc_encoded = encode_numerical_feature(H_perc, "% H", train_ds)
G_perc_encoded = encode_numerical_feature(G_perc, "% G", train_ds)
I_perc_encoded = encode_numerical_feature(I_perc, "% I", train_ds)
E_perc_encoded = encode_numerical_feature(E_perc, "% E", train_ds)
B_perc_encoded = encode_numerical_feature(B_perc, "% B", train_ds)
T_perc_encoded = encode_numerical_feature(T_perc, "% T", train_ds)


all_features = layers.concatenate([dom1_encoded, dom2_encoded, length_encoded,
    disordered_perc_encoded, C_perc_encoded, H_perc_encoded, G_perc_encoded,
    I_perc_encoded, E_perc_encoded, B_perc_encoded, T_perc_encoded])

# Creating the layers

dense1 = layers.Dense(128, activation="relu")(all_features)
drop = layers.Dropout(0.2)(dense1)
relu1 = layers.ReLU(128)(drop)

dense2 = layers.Dense(256, activation='relu')(relu1)
relu2 = layers.ReLU(256)(dense2)

dense3 = layers.Dense(256, activation='relu')(relu2)
drop1 =  layers.Dropout(0.3)(dense3)

output = layers.Dense(1, activation='sigmoid')(drop1)
```

```
121
122 model = keras.Model(all_inputs, output)
123
124 opt = keras.optimizers.Adam(learning_rate=0.001)
125 model.compile(opt, "binary_crossentropy", metrics=["accuracy"])
126
127 # Evaluating the model on the test dataset
128 loss, accuracy = model.evaluate(test_ds)
129 print("Accuracy:", accuracy)
130 print("Loss:", loss)
```

Listing 5.4: Code used for the construction of model A.

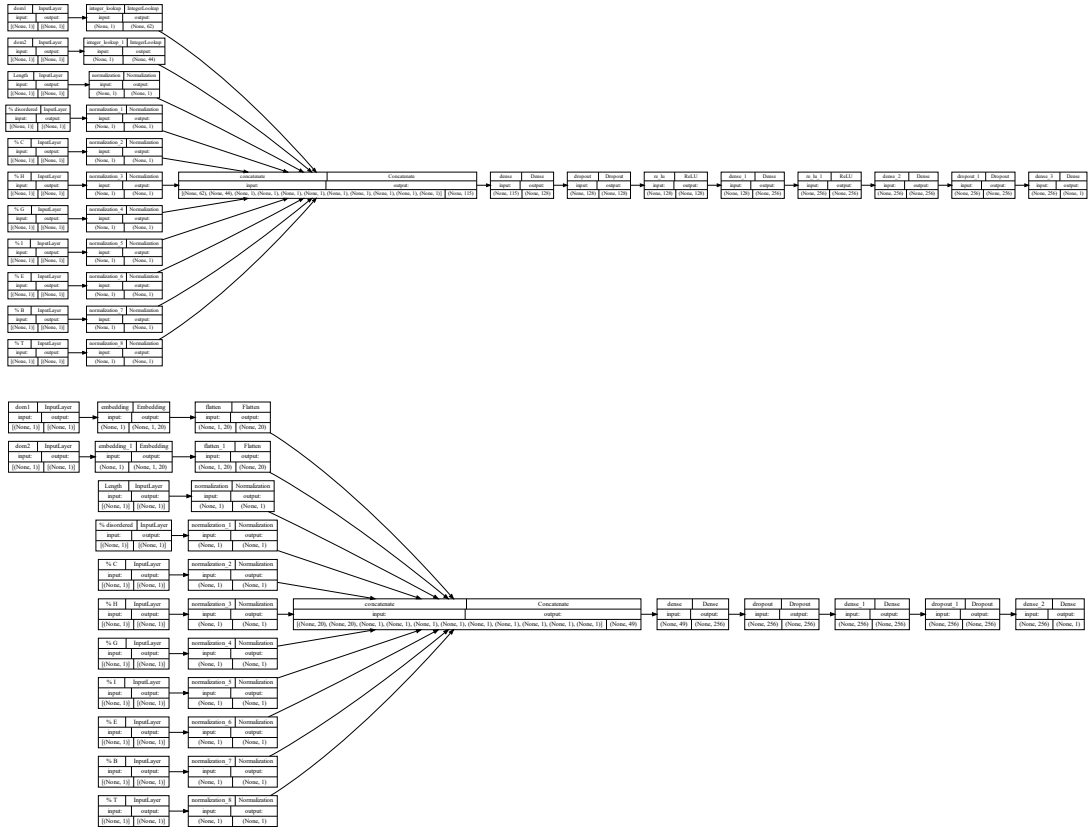

Figure 5.2: Connectivity graphs of the two models built using keras. Top: model using one-hot encoding. Bottom: model using embedding layer.

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 4, 14507)]        0

cast_to_float32 (CastToFloa  (None, 4, 14507)          0
t32)

expand_last_dim (ExpandLast  (None, 4, 14507, 1)       0
Dim)

normalization (Normalizatio  (None, 4, 14507, 1)       3
n)

conv2d (Conv2D)              (None, 4, 14507, 32)      320

conv2d_1 (Conv2D)            (None, 4, 14507, 64)      18496

max_pooling2d (MaxPooling2D  (None, 2, 7253, 64)       0
)

dropout (Dropout)            (None, 2, 7253, 64)       0

flatten (Flatten)            (None, 928384)            0

dropout_1 (Dropout)          (None, 928384)            0

dense (Dense)                (None, 1)                 928385

classification_head_1 (Acti  (None, 1)                 0
vation)

=================================================================
Total params: 947,204
Trainable params: 947,201
Non-trainable params: 3
_________________________________________________________________
```

Figure 5.3: Classification model obtained through autokeras.