Departmental Coversheet

MSc in Computer Science 2018-19

Project Dissertation

Project Dissertation title: Deep Learning models for the analysis of short tandem repeats in DNA sequences

Term and year of submission: Trinity 2021

Candidate Number:1048808

Words counted using Overleaf: 21620

Deep Learning models for the analysis of short tandem repeats in DNA sequences

Candidate Number: 1048808

University of Oxford

A dissertation submitted for the degree of

Master of Science in Computer Science

Trinity 2021

Abstract

The emergence of short tandem repeats (STRs) as functional genomic elements associated with various diseases, has highlighted the need for efficient methods aiming at their identification and analysis. Recently, deep learning has emerged as an important tool in analyzing and capturing characteristics of genomic data.

In this work, we explore for the first time the use of deep learning architectures for STR prediction, identification, and analysis. For that purpose, we train neural networks to predict if a sequence contains STRs as well as to predict their exact starting and ending position. We investigate the ability of the models to detect the repeats in cases of extensive mutations, i.e. when their repeated units are imperfect, a highly desirable property that is usually difficult to be achieved by other software and algorithms. Subsequently, we attempt to correlate the repeat presence with the genomic composition of their flanking sequence by training models to predict if a sequence comprises a flanking sequence of a repeat. We use both supervised and unsupervised deep learning architectures, like Convolutional Neural Networks and Variational Autoencoders. To achieve all these, we firstly design different ways of creating training data from the raw human genome and evaluate how the design of the datasets affects the model performance. We show that the implemented models can achieve high performance in most of these tasks.

This work, introduces a new approach of analysing STRs and lays the foundation for future capitalization of the power of deep learning for this complex task.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

In the last decade, huge progress has been made in the genomics field, which has led to a better understanding of the genetics of various diseases. A central topic of research is the correlation of DNA variants with phenotypic variations, namely the exploration of how different variants in the DNA sequence are associated with human diseases and traits. Eukaryotic genomes are composed of both repetitive and unique components. However, for many years, the repetitive part of the genome has been considered as non-important and, thus, the above studies have only focused on the exploration of genetic variation and mutations in the non-repetitive part.

Nonetheless, the repetitive elements of the DNA account for more than half of the human genome. Repetitive DNA can be divided in tandem and interspersed repeats. While in tandem repeats the repeated sequences are adjacent to one another, in interspersed repeats, they are dispersed throughout the genome [7]. Short tandem repeats (STRs), consisting of 1–9 bp motifs, are usually found in the non-coding regions and are one of the most polymorphic and abundant repeat classes [8]. Their repetitive structure causes frequent errors during DNA recombination and replication, leading to STR mutation rates that are 10 to 10^4-fold higher than those of non-repetitive loci [9].

The direct association of tandem repeats expansion with certain diseases has been proven. The term repeat expansion is referring to repeat mutations in which the number of repeated units, and thus the total length of the repeat, become unstable [10]. Certain tandem repeat expansions have been proven to cause Huntington disease, spinocerebellar ataxias, Friedreich ataxia (FRDA), fragile X syndrome (FXS), myotonic dystrophy, and other diseases [11] [12]. These are monogenic diseases, which means that they are caused by mutations in a single gene [13]. Although the role of

the tandem repeats in some of these monogenic diseases is known since the 1990s, the perception that most tandem repeats are "genomic junk" and are reflecting the fallibility of DNA replication but do not affect the differences in complex phenotypic features, such as the occurrence of polygenic diseases, was held for many years. This changed when the revolution in genomic sequencing and functional genomics allowed the identification of tandem repeats as functional sequences with complex roles in DNA, RNA, and protein levels, many of which are still unexplored.

Specifically, latest evidence suggests that STRs are contributing to gene regulation and expression [14] [12] [15] [16]. This is of high importance since research has suggested that the way non-coding variants influence traits and phenotypes is by changing gene expression, which determines the diversity of cell types and states in multi cellular organisms [17]. Therefore, STR expansions, the majority of which are found in non-coding regions, by influencing gene expression could potentially be the source of variants in phenotypes, e.g., different traits or existence of a disease.

The fact that emerging data indicate that STRs influence gene expression could potentially mean that they control gene products associated with polygenic disorders. Polygenic disorders are caused by mutations in multiple genes and, thus, the identification of their sources is more complex. Tandem repeats' involvement in disorders with complex genetics is mostly unknown but recently efforts have been made to identify tandem repeat expansions that are associated with complex diseases e.g. autism [18]. The evidence of the association of tandem repeats with polygenic disorders indicates that a potential source of the "missing heritability" problem is the non-inclusion of tandem repeats polymorphisms in the analysis, which focuses only on the polymorphisms of the non-repetitive part of the genome, mainly the single nucleotide polymorphisms (SNPs) [19].

The so called "missing heritability" problem observed in most complex traits or diseases is a problem that researchers have been trying to explore for years, and it condenses to the fact that all the SNP variations taken together confer less disease risk than expected and cannot account for most of the heritability of a given trait or disease [20]. Literature suggests that one of the potential causes for these heritability gaps in polygenic disorders could be the fact that the relevant studies do not capture polymorphisms of the repeatome, particularly tandem repeat polymorphisms (TRPs) or tandem repeat variants [15] [12]. As tandem repeats have quite higher mutation rates than single nucleotides, SNPs do not identify the vast majority of TRPs. Therefore, to discover TRs that are associated with complex polygenic diseases, an approach that views tandem repeats as potentially functional polymorphisms should

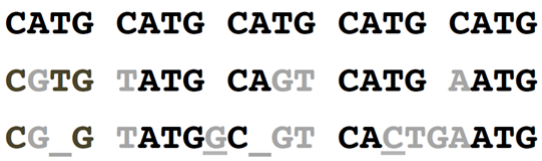


Figure 1.1: *Tandem repeat mutations.*

be taken. After identifying the tandem repeats that influence and contribute to a range of disorders, these can be targeted to prevent and treat the disease.[12].

Having underlined the importance of tandem repeats and their influence on gene regulation and expression, as well as their effect on phenotypes, the importance and need of a method for their identification and discovery is clear. The definition of tandem repeats as adjacent copies of a small motif is not strict and substitution mutations, while also insertions and deletions exist in motifs. In line 1 of Fig. 1.1, substitution mutations where a base pair is replaced by a different base pair are shown. Another type of mutation in TRs are insertion and deletions, which can concern a repeat unit or single nucleotides as shown at line 3 of Fig. 1.1. Thus, the repeated units are usually not identical leading to heterogeneity between the copies. This heterogeneity makes TR detection complicated and challenging.

In the last years, deep learning (DL) has emerged as an important tool in analyzing genomic data, learn motifs and capture characteristics of the input DNA sequences. One key application of DL in genomics is the prediction of phenotypes ab initio from DNA sequences. In line with this direction of research, in the last 5 years, researchers have attempted to predict gene expression levels, which can be viewed as an intermediate phenotype, ab initio from DNA sequences using DL models. These efforts were met with high success with models evolving over the years and achieving higher performance [21], [22], [23], [22], [24].

In [25], the authors used a DL model to confirm the hypothesis that STRs influence gene expression by showing that a significant fraction of Cap Analysis of Gene Expression (CAGE) peaks initiate at STRs. To that end, they trained a convolutional neural network (CNN) that predicts CAGE signal from STRs. That is, instead of using the whole DNA sequence as in the previous models [21], [22], [23], [22], [24], they used as input only the STRs. The importance of that is two-fold. Firstly, it highlights the contribution of STRs in gene expression, as discussed at the start of this Chapter, but also underpins the ability of DL models to analyse STRs and discern their effect on gene expression.

Inspired by these studies, while also the general emergence of DL as a bioinformatics tool that enables the better understanding of DNA functionality, in this work, we explore the use of deep learning architectures for STR prediction and identification. In more detail, we test both supervised and unsupervised architectures and compare our results with other software and algorithms proposed in literature. In addition, we use DL models to explore the correlation between the presence of repeats with the genomic composition and underlying characteristics of their flanking sequences. In more detail, we investigate whether DL models can predict the initiation of a repeat, namely that a repeat is coming, purely from its flanking sequence. This constitutes a fist step towards the association of certain genomic composition or other factors with the initiation and, potentially, the origin of tandem repeats. The origin of microsatellites is a quite complex issue and is still not completely clear among the research community [26], [27]. More details on that are given in Section 3.1.1.1.

1.2 Identification of repetitive elements – Related work

Many different algorithms and software have been proposed for the identification and detection of TRs. These can be divided in two broad categories based on the general strategy they follow. The first group consists of algorithms that exhaustively search for known repeat units throughout the input sequence. These algorithms use pre-existing knowledge on known TRs, provided by relevant databases. The second group consists of ab initio/de novo methods that do not require any pre-existing knowledge on known TRs. This enables them to identify instances of unknown TRs.

The amount of works is vast and some of the software use quite complicated algorithms and concepts. Here we will mention mainly the algorithms with which we will compare our models. As it will be described in Section 2.4, we will compare our models in terms of their performance on a manually curated dataset created in [6]. In that work the authors have evaluated the performance of a number of algorithms on their manually curated dataset of non-naive, biologically relevant TRs. The authors in their work have noted that despite the vast amount of works on TR detection tools, they were able to only use in practice the ones that they eventually compared evaluated and reported on their work. For that reason we will mainly focus on presenting these algorithms which are Dot2dot ([6]), RepeatMasker (*https://www.repeatmasker.org/*), Tandem Repeats Finder (TRF) [28], TRStalker [29], Mreps [30], Troll [31]. Along with them, we will also briefly mention some other works.

RepeatMasker is an instance of the first group of software that rely on already known repeat candidates and extensively searches for instances of them. RepeatMasker is widely used for locating both tandem and interspersed repeats. Tandem Repeats Finder (TRF [28]), models the alignment of tandem copies as a series of Bernoulli trials, whose probability of success depends on the average percent identity between the copies. TRF is fast and is considered accurate in finding repeat locations. Dot2dot is an ab-initio method inspired by graphical methods used to visually display local alignments between pairs of sequences. The algorithm exploits the fact that aligning a TR with itself forms a regular patterns, and uses visual search techniques to detect TRs. The method allows for a degree of divergence between repeat copies.

TRStalker [29] is a heuristic algorithm aiming to find long repeats that have been subject of extensive mutations. The authors deal with on a Steiner version of the problem of detecting fuzzy TR repeats for which the fuzziness is measured with respect to a motif string not necessarily present in the input string. This is equivalent to an NP problem and, thus, the algorithm cannot guarantee that all NP-Complete Tandem Repeats satisfying the target definition in the input sequence will be found. The aim is to detect highly mutated repeats that are hard to find my other algorithm Mreps [30] is an algorithm based on the concept of maximal repeats. These are repeats that can be extended as much as possible to the right/left without breaking the periodicity of the motif. Mreps finds all maximal repeats in an input sequence. It includes a resolution parameter allowing for a certain flexibility in order to find approximate repeats, i.e. mutated repeats. The detection of approximate repeats is done by computing the so-called runs of k-mismatch tandem repeats, i.e. repeats that allow at most k mismatches between two adjacent repeated units. Troll [31] is an algorithm that is extensively searching for pre-determined lists of motif patterns using Aho–Corasick [32] algorithm used to solve the equivalent dictionary problem, which it to find all occurrences from a list of patterns in a text string.

TANTAN [33] and Look4TRs [34] are two algorithms using hidden Markov models. TANTAN uses simple hidden Markov model (HMM) to quickly compute the probability that each residue is part of a tandem repeat. The generative model includes a state for non–repetitive sequence, with transitions into states for repeats of various periodicity. Look4TRs is a self-supervised Hidden Markov Model (HMM), which converts the sequence of nucleotides to a sequence of scores indicating whether each one of them is part of a repeat. The model is trained using self-generated scores of repeat regions and non-repeat regions. To generate the scores, look4TRs, generates

two random/synthetic chromosomes based on a real chromosome of the input genome and inserts artificially generated repeats in these data. Then, the HMM is trained on the scores of the generated MS and the scores of the regions in between.

1.3 Contributions

In this dissertation, we explore the use of deep learning for the prediction of regions that contain short tandem repeats, as well as for their exact localization. In addition, we use deep learning models to explore the correlation between the genomic composition of the flanking sequence of the repeats with their presence. The contributions presented in this work are the following:

1. We explore what type of dataset is more appropriate for the proposed problem. That is, we investigate two different ways of creating training data from the raw human genome sequence. In more detail, we design two different ways of creating training samples and explore how this different design affects the performance of the models trained on them. Given the fact that we are not aware of any previous work using deep learning to predict broader regions where tandem repeats are located or to annotate input sequences by finding the exact starting and ending position of tandem repeats, the most appropriate way to create the training data from the available raw human genome sequence is unexplored.

2. We experiment with, evaluate and compare three different, supervised and unsupervised, deep learning architectures to predict tandem repeat locations, namely fixed length windows that contain repeats. We investigate how the model performance is influenced by different factors, such as the length of the repeat observed by the model. We also test the ability of our models to predict the repeats at different mutation levels, which is a highly desired property, since mutated repeats are often associated with diaseases.

3. We test the performance of the models in annotating sequences. That is, to find the exact starting and ending position of repeats. For that purpose, we adapt and extend an existing algorithm [35] to be used as an extra processing step, in combination with the model's prediction of the exact starting and ending positions.

4. We experiment with the deep learning architectures for the task of predicting if a sequence comprises a flanking sequence of a repeat. This way, we attempt to correlate the repeat presence with the genomic composition of their flanking sequence. That is a first step towards exploring the flanking sequence composition and characteristics as potential contributor to the origin and emergence of repeats, which constitute still unclear and complicated issues.

5. We design different experiments and use different evaluation datasets to fulfill the aforementioned objectives. We analyse our results and propose different lines of future work.

1.4 Dissertation Structure

This dissertation is structured as follows:

- Chapter 2 (Problem Definition) formulates the problems targeted in this work. We discuss the format of the input and output of the implemented models and the details of the experiments designed and performed.

- Chapter 3 (Background) Gives the genomic background around tandem repeats and the theoretical background of deep learning models that are used in this work.

- Chapter 4 (Dataset Creation) discusses the nature of the data, the pre-processing steps, and the creation of the datasets used in the experiments.

- Chapter 5 (Methods) describes the specific architectures and optimization settings that were selected and discusses related challenges.

- Chapter 6 (Experiments) describes the different experiments performed, presents, compares and analyses the results.

- Chapter 7 (Conclusions) performs a structured assessment of the results. Then, it describes the limitations of our work and suggests potential future directions to address them and further improve the performance of our models.

Chapter 2

Problem Definition

In this Chapter, we will discuss the description of the problem addressed in the current work. The detailed problem definition, input and output format of the models used, while also the evaluation metrics used to assess the performance of the models will be given. In addition, details on the dataset nature and metrics will be discussed before a more detail description is given in Chapter 4.

2.1 Input format

The input of our models will be a human DNA sequence. DNA consists of two chains (strands) that form a double helix Fig. 2.1. Each polynucleotide chain consists of monomeric units called nucleotides, corresponding to one of the four chemical bases: adenine (A), guanine (G), cytosine (C), and thymine (T). Each nucleotide binds with its respective counterpart, forming a base pair (bp), according to the base pairing rules (A is bound with T and G is bound with C) [36]. The complementarity of DNA results in the two strands of DNA containing the same biological information. For that reason, only one strand will be used as input to our models. Thus, the input consists of a sequence of A,G,T,C letters.

Around 30% of the genome consists of repetitive sequences while 3% of it consists of short tandem repeats (STRs). STRs are sequences where a short motif (1-9 bp) is repeated a number of times, with the repeats being adjacent to each other. More details on STRs are discussed in Section 3.1.1. From the human genome, we created sequences containing STRs, which constitute the positive samples of our dataset. Negative samples were selected to be sequences that did not contain any repetitive element (tandem repeat or interspersed repeats). In Section 2.1.1, we will introduce the different types of datasets as well as the objectives targeted by each of them.

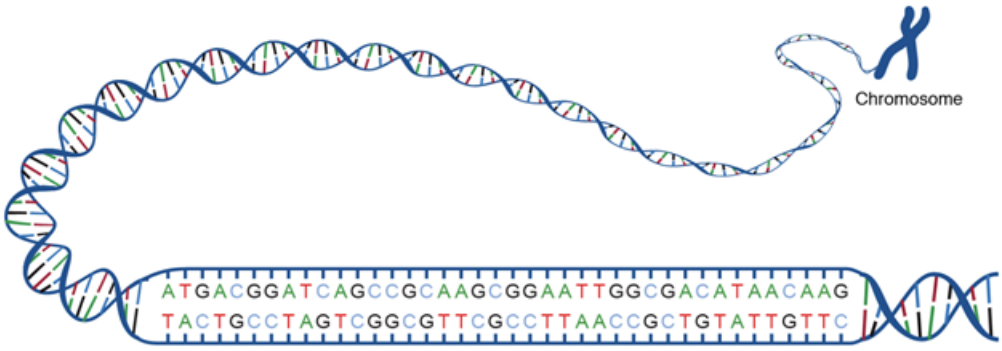


Figure 2.1: *DNA is packed in structures called chromosomes. It is comprised of 4 chemical bases A, C, T and G, called nucleotides. Nucleotides bound in a complementary way A-T and C-G.*

More details on the way these datasets were created will be given in Chapter 4. In Section 2.4, the different experiments related to each dataset type are presented.

2.1.1 Dataset types

Three different dataset types were created.

1. The first dataset contains 80bp windows where the starting point of the repeat is found between positions 0-20. This ensures that the window contains a substantial part of the repeat that will allow the model to identify it. This dataset contains each repeat only one time.

2. The second dataset contains 80bp windows where the starting point of the repeat could be in any position. Each repeat is contained in more than one sample, in a different position in each of them.

3. The third dataset consisted of sequences flanking the repeat, meaning that they were met just before the start of the repeat.

The first two datasets were used in experiments that aim to identify repeats. We will refer to this aim as objective I. Firstly, we aim to predict if a window contains a repeat (objective I.I). Subsequently, apart from predicting windows containing repeats, the models were also tested on their performance on annotating sequences with repeats. That is, on their ability and precision on finding the exact starting and ending position of repeats (objective I.II).

The third dataset was used in experiments that aim to predict if a sequence constitutes a flanking sequence of a repeat, namely if a repeat is following immediately after it. We will refer to this aim as objective II. The idea behind these experiment is to test the ability of the models to identify patterns that are potentially signaling and initiating the repeats. More details on the importance of that are given in Section 3.1.1.1.

2.2 Output format

The experiments related to both the objectives discussed above constitute a two-class classification problem. For each window the output of the model can be interpreted as the probability of it belonging to the positive class. The positive class depends on the specific experiment and dataset used. For the first group of experiments, targeting objective I, the positive class constitutes of samples containing repeats, while for the second group, targeting objective II, he positive class constitutes of samples containing the flanking sequence. To make the class prediction we define a threshold of 0.5. Thus, for outputs of the model higher than 0.5 we consider the input window belonging to the positive class.

We face the annotation task (objective I.II), again as a two-class classification problem. To annotate a certain sequence, we slide a window of 80bp over it. For each such window, the model predicts if it contains a repeat or not. In each position, the nucleotide that is on the ending part of the window (last position) is annotated. That is, as the window slides over a sequence with step equal to one, one extra nucleotide is annotated as being contained in a repeat or not. The goal is for the model to be able to predict the start of the repeat as early as possible.

2.3 Evaluation metrics

This section discusses the evaluation metrics used to assess the performance of the models. For the experiments related to the prediction of an input sequence as containing a repeat (objective I.I) or as being a flanking sequence before the repeat (objective II), we use two main metrics: accuracy and F1-score. These are standard metrics used for the evaluation of the quality of classification performed by machine learning models.

For the experiments involving the exact annotation of sequences (objective I.II), a measure indicating the similarity/matching between the sequence annotated by the

model as a repeat and the ground-truth repeat should be used. For this purpose, Jaccard coefficient was used. It provides an efficient way to measure the similarity between the model's annotation, namely the part of the sequence the model has masked as the repeat and the ground truth. Jaccard coefficient is generally used to measure the similarity of two intervals and, in our case of two genomic intervals. It is defined as the ratio between the length of the intersection of the two intervals divided by the size of their union. It takes values in the range of $[0, 1]$ and is one only when the two sequences exactly match.

Apart from the different evaluation metrics used in the experiments for the exact annotation of sequences, different evaluation datasets were used. That is, although the models were trained using the datasets discussed above in Section 2.1.1, the performance of the models was evaluated in different datasets. In more detail, for the training data, RepeatMasker has been used to annotate the DNA sequence, namely, to identify the repeats so that the labelled samples can be created. More details on the creation of the labelled data using RepeatMasker are given in Chapter 4. Thus, when evaluating the model's performance on the left-out data using the accuracy, f1, precision and recall metrics, we consider RepeatMasker annotation of the DNA as the ground truth.

However, RepeatMasker is imperfect. The contribution of annotating the repeats, which is one of the main objectives of this work, would be important in two cases. The first aim is that our models can annotate the DNA with higher accuracy than RepeatMasker or other software. The second aim would be for our models to be able to achieve comparable performance with RepeatMasker or other software and produce similar annotation of tandem repeats, while being significantly faster.

In relation to the first aim, in order to compare the performance of our models with other software and RepeatMasker, we need a test dataset manually curated, where the location of the repeats has been independently found and verified. That is, we need to compare our models' performance on a dataset that can be considered accurate and that is not the result of another software's annotation. For that reason, we use a manually curated collection of five datasets containing biologically relevant, non-naïve repeats, created in [6]. In more detail, the authors collected:

- 45 verified disease-related TRs,
- 21 TRs relevant to DNA profiling,
- 86 TRs located in the Y-chromosome commonly used for paternity or genealogical tests,

- 600 TRs distributed across the autosomes each of which containing a short and highly polymorphic TR and,
- 15,326 TRs located in upstream regulatory regions.

2.4 Project Focus

Initially, we explored what type of dataset is more appropriate for the proposed problem. We have designed two different ways to create the training datasets from the raw genomic data and, also, designed experiments to evaluate the performance of our models in different tasks. We aim to use our models to make predictions about the location of repeats, and also analyse the genomic data to give as insights about their presence that will allow us to draw biological conclusions. Thus, we have designed the following experiments:

- **Experiment I**: We create the dataset so that the starting point of the repeat is placed on the initial part of the sample by randomly selecting a position in a prespecified range. The dataset is comprised of samples containing repeats (positive class) and random sequences that do not contain repeats (negative class). We train our models to perform 2-class classification predicting which samples contain the repeats.

  We investigate how the models' performance is influenced by different factors, such as the length of the repeats observed by the model and the repeats mutation levels.

- **Experiment II**: We create the dataset so that the starting position of the repeat can be in any position of the input sample. Again, we train our models to perform 2-class classification predicting which samples contain the repeats. We investigate how the performance of the second type of models is influenced the by the length of the repeats and the repeats mutation levels.

  Then we compare the performance of the models of this experiment with the ones from Experiment I, in order to conclude on the most appropriate dataset.

- **Experiment III**: We use our models to find the exact starting and ending position of the repeats in sequences containing biologically-relevant repeats. We design an extra processing step to make the prediction of our models more precise. We compare our models with other software/algorithms proposed in the literature.

- **Experiment IV**: We create the dataset so that it contains the upstream flanking sequence of repeats (positive class) and random sequences that do not comprise repeat flanking sequence (negative class). We train our models to perform 2-class classification predicting which samples comprise flanking sequence of repeats. That way we aim to correlate the genomic composition of the flanking sequence with the presence and potentially the genesis of repeats.

Chapter 3

Background

The current chapter provides an overview of the theoretical background related to this work. In Section 3.1, the genomics background related to tandem repeats is given. Information on the structure of TRs, their hypermutability, and potential sources accounting for it are introduced. In addition, their established association with monogenic diseases as well as their potential role in polygenic diseases according to latest evidence is discussed. Then, in Section 3.2, initially, the general theoretical background of deep neural networks is introduced. Then, we discuss the theoretical background of convolutional neural networks (CNNs) and variational autoencoders (VAEs), the two main architectures used in this work.

3.1 Genomics preliminaries

Chromosomes are long DNA molecules containing part of the genetic material of organisms. Human cells have 23 pairs of chromosomes. Each chromosome contains a different number of genes, which are made of DNA. DNA consists of two chains that form a double helix. Each of the chains consists of monomeric units called nucleotides, corresponding to one of the four chemical bases: adenine (A), guanine (G), cytosine (C), and thymine (T). The chains are bound together through their nucleotides and are complementary (A is bound with T and G is bound with C). DNA can be distinguished in repetitive and non-repetitive.

3.1.1 Tandem repeats

Repetitive DNA constitutes of patterns of one or more nucleotides that are repeated. Repetitive elements are constituted of two groups: tandem repeats and interspersed repeats. Interspersed repeats are specific patterns that are dispersed in multiple loci

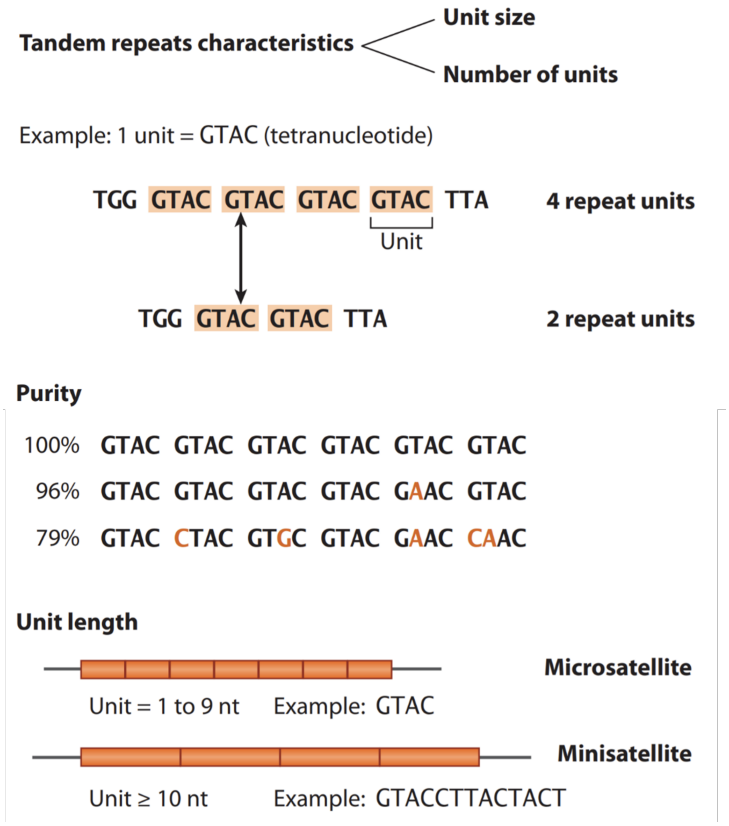


Figure 3.1: *Unit size and number of units are factors that characterize tandem repeats. Microsatellites (Short tandem repeats), are comprised of repeated motifs of 1-9 bp, while minisatellites of* $\geq$ *10 bp.* .

throughout the genome, while tandem repeats are characterized by the repeated patterns being adjacent to each other. The number of times the motif is repeated varies from a small number to more than a hundred [37]. Although definitions vary, short tandem repeats (STRs), also called microsatellites, are the tandem repeats where the repeated motif is 1-9 bp and comprise 3% of the human genome. Based on the length of the major repeated motif, STRs can be classified into mono-, di-, tri-, tetra-, penta-, and hexanucleotide repeats. Longer tandem repeats, also called minisatellites, have a motif from 10-100bp and long tandem repeats, also called satellite DNA have a repeated motif longer than 100 bp [38]. In Fig 3.1 taken from [39], the repeat GTACGTACGTACGTAC is constituted of the repeat motif GTAC repeated four times.

Tandem repeat mutations

DNA replication is the biological process by which a double-stranded DNA molecule makes a copy of itself and produces two identical DNA replicas. During DNA replication, the repetitive nature of tandem repeats can induce slippage replication events, namely the misalignment of two DNA strands during replication, which leads to genetic rearrangements such as microsatellite instability [40]. In more detail, replication slippage causes the generation of deletions or insertions within repeat regions leading to varying number of copies of the repeated motif. These variations are called "dynamic mutations" [41]. Research has shown that the slippage replication events are influenced by the repeat unit length, with lower repeat lengths corresponding to higher slippage rates. In addition, studies suggest a positive correlation between repeat number and mutation rates [8]. Mutation rates are also influenced by external factors, e.g., during transcription of a tandem repeat, higher transcription levels could result in mutations [39]. Recombination events [42], which is the biological process by which pieces of DNA are broken and recombined to produce new combinations of alleles, can also lead to contraction and expansions of TR sequences. Strand-slippage replication is considered as the main mechanism of STR mutation process, while recombination is considered the main mechanism causing the mutation of minisatellites [43].

Since the repetitive nature of tandem repeats favors the occurrence of such events, tandem repeats are hypermutable [44], with mutation rates $10–10^5$ times higher than the average DNA polymorphisms rates throughout the genome [39] and, thus, constitute a large part of the genetic variation across humans. Apart from the insertion and deletions of one or several repeat units, STRs are also affected by "non-specific" mutations as shown in Fig 1.1, such as insertions/deletions of single nucleotides, while also transition, where a single (two ring) purine (A or G) is changed for a (one ring) pyrimidine (T or C) or vice versa, and transversion, where a purine nucleotide is changed to another purine (A $\longleftrightarrow$ G), or a pyrimidine nucleotide to another pyrimidine (C $\longleftrightarrow$ T).

Position of STRs and how it affects their importance

Most STRs are located in non-coding regions, while around 8% of them are located in coding regions. While 3% of the human genome constitutes of STRs, 6% of the coding regions are considered to contain STR variations [9]. As mentioned in [9], in the coding regions, STRs tend to be located in genes whose products affect transcriptional

regulation, DNA binding, protein–protein binding, and developmental processes. It is also known that in non-coding regions, many STRs are located in promoters and enhancers and other regions having a role in gene regulation and gene expression [45]. The location of TRs suggests that they have a functional role in gene regulation, expression and in phenotypic variants. As described in the Section 1.1, for many years, repetitive DNA was considered as non-functional and "junk" [46], [47]. However, the last years, the importance of STRs has been highlighted through many studies. STRs variations have proven to have an important role in regulation, e.g., by modulating the binding of transcription factors [48], having the capacity to inhibit promoter activity [49]. In addition, latest studies have highlighted STRs play a key role in regulating gene expression [15] [16]. All this evidence has underlined the biological importance of STRs and their role in genetic and phenotypic variation and indicates the need to investigate their yet unexplored association with diseases.

TR expansions associated with diseases

As briefly mentioned in the Section 1.1, the determinant role of TR expansions in some monogenic diseases was already uncovered since the 1990's. Over 40 mendelian disorders, which result from a mutation at a single genetic locus have been attributed to STR expansions. Huntington disease, spinocerebellar ataxias, Friedreich ataxia (FRDA), fragile X syndrome (FXS), myotonic dystrophy are some prominent examples of diseases caused by STR expansions [Hannan, 2018]. For example, Huntington disease is caused by the expansion of a CAG repeat in exon 1 of the IT15 gene. While the repeats in healthy individuals vary from 6 to 35, 36-39 repeat units indicate an increased risk of developing the disease and individuals with more than 40 repeats have the disease.

Apar from the contribution in monogenic disorders, the emerging data discussed above that indicate the functional role of STRs and their determinant role in gene expression, indicate that STRs have the potential to contribute to the missing heritability of polygenic disorders by controlling gene products that are associated with certain polygenic disorders [12].

The recent findings that highlight the importance of STRs have increased the interest in them, with some studies identifying associations between STR expansion and polygenic disorders. In [18], the correlation between tandem repeats with a motif of 2-20 bp and autism spectrum disorder was investigated and it was observed that tandem repeat expansions at 2,588 gene-associated loci were significantly more prevalent among individuals with the disorder, while it was rare among the health population.

In various studies, short tandem repeat instability, also called microsatellite instability (MSI) has been highlighted as a molecular phenotype for different types of cancer, e.g. endometrial, and colorectal tumors [50], [51]. In [52], the authors identified MSI-positive tumors in 14 of the 18 cancer types explored, while also observed a correlation of patient survival rates with the number of unstable sites. Tandem repeat polymorphism has been associated with various other complex diseases, e.g. diabetes [53].

3.1.1.1 Origin of tandem repeats flanking sequence

The origin of microsatellites is a complex matter that has concerned the research community for years. [26]. Their genesis appears to be non-random [27]. They can arise de novo, spontaneously within unique sequences, while also brought to a genomic location by transposable elements.

Many theories have been proposed for the microsatellite's origin. Recombination and replication slippage, as it was described previously in the current section 3.1, are considered two of them. However, several studies have suggested that a minimum number of repeats is needed before DNA slippage can extend the repeat [54], [55]. A number of studies have been conducted to correlate the origination of microsatellites with the presence of specific genomic elements, e.g. mobile genetic elements [56], [57], [58]. However, other studies have shown that the high density of transposable elements does not always correlate with the presence of microsatellites [59], [60], and thus, their contribution in microsatellite genesis remains unconfirmed.

The amount of studies exploring the origin of TRs, and the fact that these occasionally come to conflicting conclusions, highlight the complexity of the issue of microsatellite origin. The investigation of this issue could lead to a better understanding of the function and organization of the genome, the evolution of species and the population genetic studies [26].

The flanking sequence of the repeat is defined as the sequence immediately upstream and downstream of the repeat locus and is of high importance. Its genomic composition has been proven to significantly influence the mutability of the repeat [61], an assumption that was first made in [62]. The flanking region's GC content and type of DNA (e.g. if it is gene-related) are a few factors that have been proposed as ones that potentially affect the mutation rates of the repeat [27].

However, conflicting results have been observed. For example, in [62], the richness in GC content in flanking sequence of microsatellites has been negatively correlated with their genetic diversity. However, one study in shrews [63] and another in

Drosophila melanogaster [64] did not find any such positive or negative correlation. Studies have suggested that the influence of flanking regions on repeats should be further explored [27].

One objective of the current work (objective II as introduced in Section 2.1.1), is to investigate the correlation between the genomic composition of the repeats' flanking sequence with their presence. We will use NNs to capture patterns and characteristics that are present in the flanking regions of repeats and could potentially be a factor contributing to their genesis/origin.

3.2 Deep Learning preliminaries

In this Section, we summarise some machine learning background that is relevant in the current work. Initially, we introduce the general ideal behind neural networks (NNs) and we continue by describing the specific architectures that we have implemented in this work's experiments, the convolutional neural networks (CNNs) and the variational autoencoders (VAEs).

3.2.1 Neural networks and Backpropagation

A feed-forward neural network (FNN) is an artificial neural network(ANN) in which the nodes do not come in circular forms. These networks can only process information in one direction. These can be single-layer networks (i.e. consisting only of input and output layers) or multi-layer networks with multiple hidden layers. In a feedforward neural network, every perceptron in one layer is connected with each node in the next layer. The general idea is that the FNN receives an input, propagates its information through the layers by computing an 'activation' for each neuron, and finally outputs the activations in the output layer. In between, it assigns each input a learnable weight, and adds a learnable bias term to produce a pre-activation value. Typically, all the nodes are fully connected. More formally, given a k-dimensional input vector $\mathbf{x} \in \mathbb{R}^k$ the output $\mathbf{y} \in \mathbb{R}$ of a perceptron can be calculated as:

$$y = f(\mathbf{W}\ \mathbf{x} + \mathbf{b})$$

where W $\in \mathbb{R}^k$ are weight parameters of the perceptron, $\mathbf{b} \in \mathbb{R}$ is a bias term which provides an independent constant added to the input and $f : \mathbb{R} \longrightarrow \mathbb{R}$ is an activation function. Typically, every unit has a linear function followed by a non-linear activation function. There are no back-loops in the feed-forward network. These networks are

mostly used for supervised machine learning tasks, where the model learns to predict outputs $\mathbf{y}$ from inputs $\mathbf{x}$.

Backpropagation algorithm is a widely used technique for training neural networks, in which the two more essential factors are the loss function and gradient descent optimization algorithm. On a training dataset, the loss function evaluates a model with kernels and weights through forward propagation, which next, are updated, in respect to the loss value, through backpropagation and gradient descent.

More specifically, through forward propagation, the model, takes a certain input $\mathbf{x}$, whose true label is $\mathbf{y}$, and predicts a label $\mathbf{z}$. Comparing those two labels, true and predicted, the loss value is calculated. Concerning the backpropagation phase, gradient descent optimization is used, in order to determine in what direction the learnable weights should be adjusted to decrease the loss. The procedure is called backpropagation, since the loss value is fed backwards in order to, fine-tuned the weights of the network. In order to finish the training procedure through backpropagation, a stopping criterion is applied, which typically refers to a number of epochs, a loss value threshold etc.

3.2.2 Convolutional Neural Networks

In this Section, we will introduce convolutional neural networks (CNNs). The following introduction is based on the the notes of the Machine Learning course at the University of Oxford [1] and [65]. In Section 3.2.2.1, we will discuss CNNs in the context of genomics and DNA sequences.

Convolutional neural networks, firstly introduced in [66], is a class of neural networks that has been introduced for the analysis of imaging data and its successful application has expanded in multiple other domains. As the feedforward NNs introduced in the previous Section, CNNs are also comprised of neurons that have learnable weight ans biases. However, their layers have neurons arranged in 3 dimensions: width, height, depth. Their distinct feature are two main operations, convolution and pooling.

Convolution Operation

The parts of the network where the convolution operation takes place are called convolutional layers. These layers are comprised of learnable filters, also known as kernels, which scan the input, i.e. are moved over the image. At each position, the dot product between the filter and the same size patch of the input at that position is computed, as shown in Fig. 3.2. The aim is to discover spatial patterns in patches.

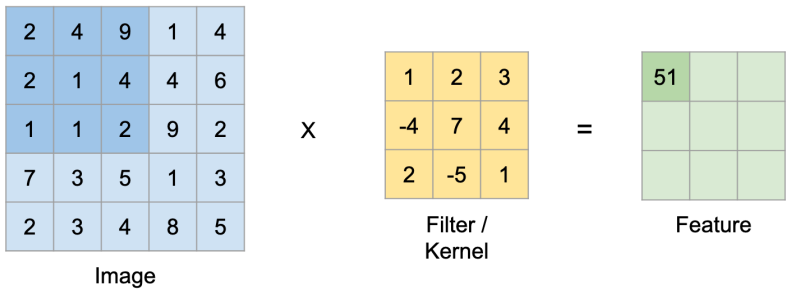


Figure 3.2: Convolution operation. The dot product between the patch of the input image (shaded blue part) and the filter is calculated (left corner of the green box).

In more detail, let the input be a 2-dimensional tensor of shape $m \times n$, and the convolutional filter a real-valued 2-dimensional tensor of shape $W \times H$. While the filter is scanning the input over all possible dimensions (width and height), the element-wise product of it with each input patch of shape $W \times H$ is calculated. The tensor occurring from the element-wise multiplication is summed and the result corresponds to an element of a 2-dimensional feature map. That feature map (green box in Fig. 3.2) corresponds to the responses of that filter at every spatial position. Each filter scanning the input image produces a feature map. The filters are learnable parameters, that are learned through backpropagation. Filters of initial convolutional layers typically learn to detect less complex patters, such as edges, while filters in later layers learn more complex patterns.

Different ways of how the filter is scanning the input can be selected, i.e. which patches of the input are considered. The parameters that determine the way the input is scanned are stride and padding. For a 2D convolution, the stride parameter controls the step towards each direction. For example, one could select to not scan every possible input patch, but every other one, i.e using a stride of 2. Pixels in the corners are not treated in the same way as pixels at the inside parts of the input, since the filter cannot move to some positions at the corners of the input. Although, this is usually not a problem, especially for large inputs, padding is a way to eliminate this behavior by adding a frame of zero-valued pixels at the boundary of the input data.

Typically multiple filters are applied at each convolutional layer, each one producing a different feature map. The feature maps generated when the filters slide over the input, correspond to identified simple or complex features present in the image.

Pooling Operation

Another operation that is often used in CNNs is pooling. After the convolutional layer, the dimensionality of feature maps is typically getting reduced. This is done by applying a certain operation on all patches. That operation is typically taking the maximum, average or minimum among all elements for each input patch, i.e. pooling the pixels together. The aim is to decrease the size of the feature maps but at the same time preserve important features detected before the pooling layer. Since we typically want to decrease the dimension of the feature maps, pooling is usually applied with a stride of 2 or higher.

By using only the maximum, average or minimum activation through the pooling layer, the convolutional layers become more invariant to distortions of the target in extracted feature maps. In Fig. 3.3 the case of max pooling is shown, where the maximum value in each patch of the feature map is calculated. Different patches of the feature map are shown with different colours. This way, the feature map, showed in the left part of Fig. 3.3 is down-sampled and the outcome is comprised of the most highlighted feature of each patch volume.

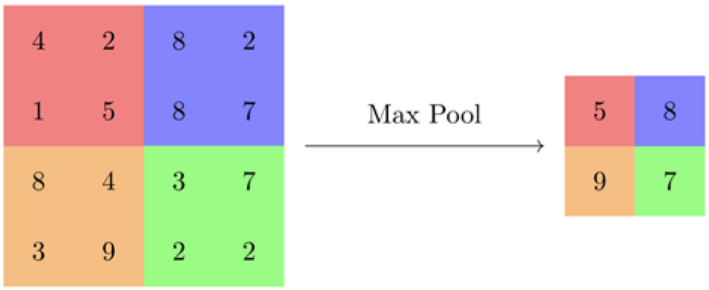


Figure 3.3: Pooling operation. Maximum pooling is applied with pool size (2,2) and stride (2,2). Figure taken from [1]

.

3.2.2.1 CNNs in genomics

CNNs is one of the main architectures that is applied to genomic data. Fig. 3.4 shows the procedure of applying a CNN model to one-hot encoded DNA sequences for the different layers of the model. In the first convolutional layer, the sequence is scanned by several filters, with each filter application outputting a scalar value, as shown in Fig. 3.4b. This value corresponds to a measure of similarity between the filter and the part of the sequence it was applied to. In the example of Fig. 3.4b, the learned filters correspond to transcription factors GATA1 and TALI1. After each

convolutional layer, a non-linear activation function, typically a ReLU, usually follows (Fig. 3.4c). In part d of Fig. 3.4, a pooling layer is applied, reducing the dimension of its input.

Different numbers of convolutional layers are appropriate for different tasks. After the total number of convolutional layers, the output is flattened (Fig. 3.4h) and used as an input to a series of fully-connected layers. The output of the last fully-connected layer is the final prediction of the CNN and its size depends on the nature of the problem.

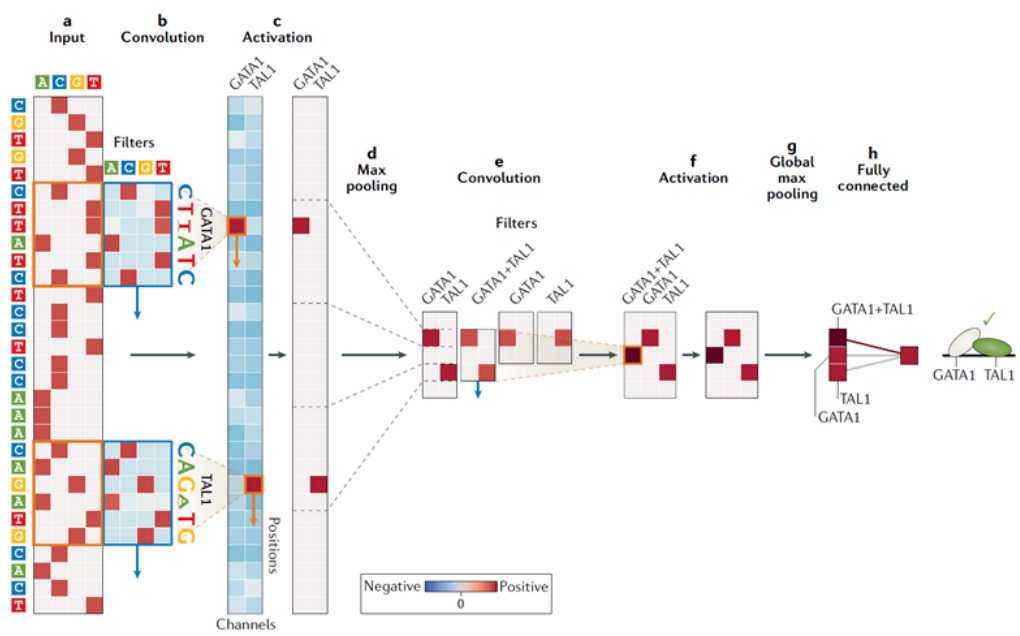


Figure 3.4: Convolutional neural networks applied to DNA sequences. Figure taken from [2].

CNNs have been widely used to detect motifs in DNA sequences. In initial layers, the CNN captures simpler motifs, while in deeper layers the network captures combinations of them and is able to learn more abstract representations and underlying characteristics of the sequence. An advantage of the end-to-end neural networks is that there is no need to prespecify any feature, since they adaptively extract features to map the raw input data to meaningful internal representations through backpropagation.

3.2.3 Variational Autoencoders

Machine Learning models can be grouped into generative versus discriminative models. While discriminative models aim to predict some output based on the observed

data, the generative models aim to learn the underlying joint distribution over the variables, which include the inputs and outputs, both of which are modelled as probability distributions. [4].

Variational autoencoders [3] [67] are an instance of generative models that aim to jointly learn a deep-latent variable model and corresponding inference models using stochastic gradient descent. In the following sections, firstly, the ideas of a probabilistic models, latent variable models and variational inference will be introduced, as they are core concepts in the VAE framework. Secondly, the basics of VAE will be introduced. The following introduction on VAEs is based on the original paper introducing the Variational autoencoders [3] and the authors' follow up notes [4].

Probabilistic models, Latent Variable Models and Variational Inference

Probabilistic models are mathematical descriptions that model events and phenomena through random variables and probability distributions. They aim to capture the correlations between the variables of interest in the form of the joint probability over them.

Let $\mathbf{x}$ be the vector denoting all the observed variables of a model. We assume that these are generated through an underlying process $p^*(\mathbf{x})$, which we aim to model with an approximate probabilistic model $p_{\boldsymbol{\theta}}(\mathbf{x})$. The learning consists in searching the value of $\boldsymbol{\theta}$ that leads to the best approximation of the true underlying distribution of the data $p^*(\mathbf{x})$, such that for the observed $\mathbf{x}$: $p_{\boldsymbol{\theta}}(\mathbf{x}) \approx p^*(\mathbf{x})$.

Latent variable models (LVMs) are probabilistic models that incorporate latent variables $\mathbf{z}$ corresponding to the observed variables $\mathbf{x}$. Latent variables $\mathbf{z}$ are not observed but are part of the model and, in conjunction with the global parameter $\boldsymbol{\theta}$, can help explain the observed variables $\mathbf{x}$. They are often lower-dimensional and more interpretable than $\mathbf{x}$ and are usually supposed to capture some underlying, complex, or conceptual characteristics of the generative process, which cannot be measured and observed directly. The joint distribution over both the observed and latent variables is given by $p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z})$, while the marginal distribution over the observed variables $\mathbf{x}$ is given by:

$$p_{\boldsymbol{\theta}}(\mathbf{x}) = \int p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}) d\mathbf{z}$$

LVMs are connected to two main purposes: model learning and inference. Model learning is focused on finding the best configuration for $\boldsymbol{\theta}$, while inference aims to calculate the posterior $p(\mathbf{z}|\mathbf{x})$ for a given fixed $\boldsymbol{\theta}$. In the case of model learning, latent variables are useful to capture the high-ordered dependencies between the observed

variables, whereas in the inference case, latent variables are the important unobserved features that we want to retrieve. The inference problem focuses on finding the posterior distribution, namely the conditional distribution of the latent variables given the observations, which is given by:

$$p(\mathbf{z}|\mathbf{x}) = \frac{p(\mathbf{z}, \mathbf{x})}{p_{\boldsymbol{\theta}}(\mathbf{x})}$$

For most models, the denominator, also called normalizing constant, is intractable. Even in cases where the denominator is tractable, the posterior distribution can be complex, which makes it impossible to characterize it, sample from it etc.

Variational inference (VI) tackles this issue by attempting to approximate the posterior in a functional, simpler form. Thus, the inference problem is reformulated into an optimization problem. A variational family $q_{\boldsymbol{\phi}}(\mathbf{z})$ is introduced and the set of parameters $\boldsymbol{\phi}$, which characterize it, are optimized in terms of the quality of approximation of $p(\mathbf{z}|\mathbf{x})$ with $q_{\boldsymbol{\phi}}(\mathbf{z})$. Kullback-Leiber divergence between $q_{\boldsymbol{\phi}}(\mathbf{z})$ and the target $p(\mathbf{z}|\mathbf{x})$, which is typically used to measure the quality of the approximation, is given by:

$$\mathrm{KL}\left(q_{\boldsymbol{\phi}}(\mathbf{z}) \| p_{\boldsymbol{\theta}}(\mathbf{z} \mid \mathbf{x})\right) = \mathrm{E}_q\left[\log \frac{q(\mathbf{z})}{p(\mathbf{z}|\mathbf{x})}\right]. \tag{3.1}$$

This proves to be equivalent to maximizing an evidence lower bound (ELBO) with respect to the parameters $\boldsymbol{\phi}$. More details on this will be discussed in Section 3.2.3.1. By maximizing the ELBO instead of minimizing the KL divergence, the optimization does not involve the posterior in its normalized form, which is usually intractable. Instead, only the unnormalized version of the posterior is needed.

Deep latent variable models (DLVMs) are LVMs where the probability distributions are parameterized by flexible deep neural networks. The most common DLVM used is specified as the factorization of the prior distribution, $p_{\boldsymbol{\theta}}(\mathbf{z})$, and the likelihood $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$, which is parameterized by a neural network. This is mathematically formulated as follows: $p_{\boldsymbol{\theta}}(\mathbf{x}, \mathbf{z}) = p_{\boldsymbol{\theta}}(\mathbf{z})p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$ Typically, $p_{\boldsymbol{\theta}}(\mathbf{z})$ is considered fixed, e.g. $p_{\boldsymbol{\theta}}(\mathbf{z}) = N(0, 1)$. DLVMs are LVMs, as shown in Fig. 3.5, with the arrows corresponding to neural networks. Thus, $\boldsymbol{\theta}$, which, from a probabilistic perspective, corresponds to the parameters of the distribution $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$, from the NN perspective, corresponds to the NN parameters, e.g. weight, biases etc.

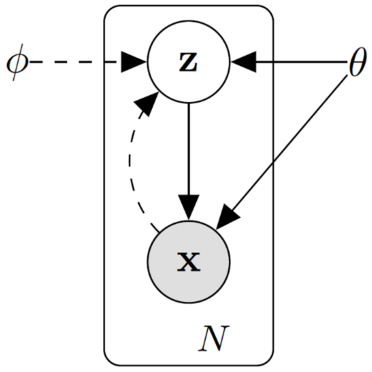


Figure 3.5: *Variational autoencoder model: Solid lines correspond to the generative model* $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})p_{\boldsymbol{\theta}}(\mathbf{z})$ *and dashed to the inference model* $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$*. Figure taken from [3].*

3.2.3.1 Variational Autoencoder framework

After having introduced the above relevant ideas of DLVMs and VI, variational autoencoders combine these ideas by providing a principled method for optimizing a DLVM with its corresponding inference model using stochastic gradient descent (SGD) [4]. In other words, VAEs jointly address the inference problem of approximating the posterior of the latents $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$ and the model learning problem of approximating the parameters $\boldsymbol{\theta}$.

For this purpose, VAEs introduce an inference model $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$, which approximates the true intractable posterior $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$. The VAE framework provides a computationally efficient way to jointly learn the inference model's parameters $\boldsymbol{\phi}$ and the generative model parameters $\boldsymbol{\theta}$. $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ is also referred to as encoder and $p_{\boldsymbol{\theta}}(\mathbf{x}|\mathbf{z})$ as decoder.

The goal is to maximize the log-probability of the data under the model $\log p(\mathbf{x})$. However, as mentioned above. VAEs use as optimization objective the ELBO, a lower bound for the log marginal likelihood $\log p(\mathbf{x})$ that can be derived as follows:

$$
\begin{aligned}
\log p_{\boldsymbol{\theta}}(\mathbf{x}) &= \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log p_{\boldsymbol{\theta}}(\mathbf{x})\right] \\
&= \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log\left[\frac{p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z})}{p_{\boldsymbol{\theta}}(\mathbf{z}\mid\mathbf{x})}\right]\right] \\
&= \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log\left[\frac{p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z})}{q_{\boldsymbol{\phi}}(\mathbf{z}\mid\mathbf{x})}\frac{q_{\boldsymbol{\phi}}(\mathbf{z}\mid\mathbf{x})}{p_{\boldsymbol{\theta}}(\mathbf{z}\mid\mathbf{x})}\right]\right] \\
&= \underbrace{\mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log\left[\frac{p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z})}{q_{\boldsymbol{\phi}}(\mathbf{z}\mid\mathbf{x})}\right]\right]}_{=\mathcal{L}_{\theta,\phi}(\mathbf{x})} + \underbrace{\mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})}\left[\log\left[\frac{q_{\boldsymbol{\phi}}(\mathbf{z}\mid\mathbf{x})}{p_{\boldsymbol{\theta}}(\mathbf{z}\mid\mathbf{x})}\right]\right]}_{\mathrm{KL}\left(\mathrm{q}_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})\|\mathrm{p}_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})\right)}
\end{aligned}
$$

Indeed, we have $\mathrm{KL}\left(q_{\boldsymbol{\phi}}(\mathbf{z}\mid\mathbf{x})\|p_{\boldsymbol{\theta}}(\mathbf{z}\mid\mathbf{x})\right) \geq 0$ which is equal to zero only if $q = p$, and, thus, the ELBO is a lower bound for $\log p(\mathbf{x})$ that is given by:

$$
\mathcal{L}_{\theta,\phi}(\mathbf{x}) = \log p_{\boldsymbol{\theta}}(\mathbf{x}) - \mathrm{KL}\left(q_{\boldsymbol{\phi}}(\mathbf{z}\mid\mathbf{x})\|p_{\boldsymbol{\theta}}(\mathbf{z}\mid\mathbf{x})\right) \leq \log p_{\boldsymbol{\theta}}(\mathbf{x}) \tag{3.2}
$$

This convenient mathematical formulation describes the VAE framework. By maximizing the ELBO w.r.t the parameters $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$, we jointly:

1. Approximately maximize $\log p_{\boldsymbol{\theta}}(x)$, which comprises the model learning procedure. Thus, we learn a generative model, namely the joint distribution $p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z})$, which can be factorized as $p_{\boldsymbol{\theta}}(\mathbf{x},\mathbf{z}) = p_{\boldsymbol{\theta}}(\mathbf{z})p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$. $p(\mathbf{z})$ corresponds to the prior distribution and $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$ to the decoder.

2. We minimize the KL divergence, leading to a better approximation of $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$ by $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$. Thus, we learn an approximation $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ of the true intractable posterior of the decoder $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$ through the encoder.

Conveniently, the ELBO allows for the joint stochastic gradient descent optimization with respect to both the parameters $\boldsymbol{\phi}$ and $\boldsymbol{\theta}$. The gradients of the ELBO are typically intractable but we can derive unbiased estimators of them which allows us to perform SGD. For gradient of the Eq. (3.2) w.r.t the generative parameters $\boldsymbol{\theta}$, an unbiased estimator of the gradient is straightforward [Appendix]. The problematic part is the gradient w.r.t. the variational parameters $\boldsymbol{\phi}$, because the ELBO's expectation is w.r.t. $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$.

In the case of continuous latent variables, the reparameterization trick, which will be discussed in the next section, is typically used to obtain unbiased estimators of the ELBO's gradient w.r.t $\boldsymbol{\phi}$. In the case of discrete variables, different approaches have been proposed. In the following section, we will discuss one of them, the Gumbel-Softmax trick introduced in [5] and [68]. This is relevant for this work, since one of

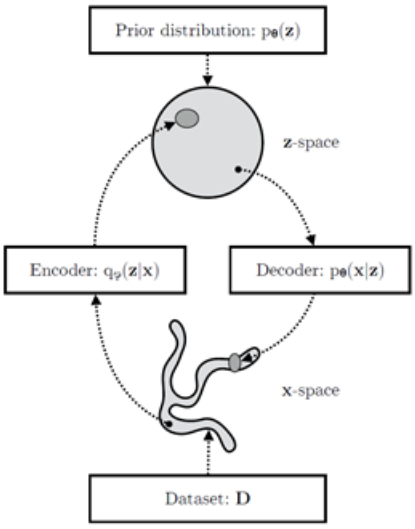


Figure 3.6: *The forward pass of a VAE is comprised of encoding the datapoints through the inference network to a latent distribution, then, sampling from this distribution and decoding them through the generative network. Figure taken from [4].*

the models implemented was a VAE with discrete latent variables using the Gumbel-Softmax trick.

3.2.3.2 Reparameterization trick and Gumbel-Softmax

In the case of continuous latent variables and differentiable encoder/decoder models, the ELBO's gradient is computed through the change of variables. The initial problem is that we cannot backpropagate gradients through the random/stochastic variable $\mathbf{z}$. To solve that, we re-express the random variable $\mathbf{z}$ as a deterministic and differentiable function of $\boldsymbol{\phi}$, namely, $z = g(\phi, x, \epsilon)$, and make the randomness derive from another auxiliary variable ϵ as in Fig. 3.5. The distribution $p(\epsilon)$ is independent of $\boldsymbol{\phi}$ and $\boldsymbol{\theta}$. This allows the backpropagation through z and the computation of the gradients w.r.t $\boldsymbol{\phi}$ with a simple Monte Carlo estimator.

In the case of discrete latent variables (the stochastic node that we want to back-propagate through is discrete), an efficient way to backpropagate has been proposed in [5]. Jang et al. introduced an efficient way to replace a non-differentiable sample from a categorical distribution with a differentiable sample from a novel Gumbel-Softmax distribution. This is feasible due to the fact that the Gumbel-Softmax distribution can be smoothly annealed into a categorical distribution.

Let z be a categorical variable with class probabilities $\pi_1, \ldots \pi_k$, whose samples are $(k-1)$-dimensional one-hot encoded vectors. A way to sample from the categorical

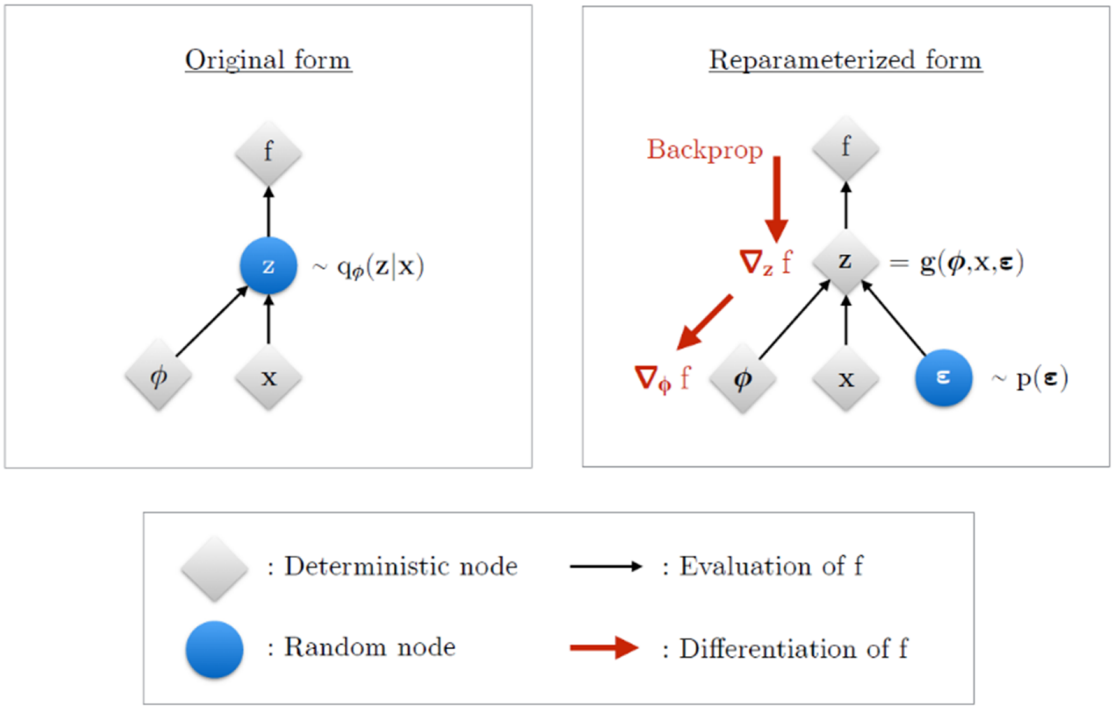


Figure 3.7: *Reparameterization trick Left: for continuous latent varaibles Right: for categorical latent variables Figure taken from [5].*

distribution is the Gumbel-max trick which is described in the following formulation:

$$z = \text{ one\_hot } \left(\arg\max_i \left[g_i + \log \pi_i\right]\right) \tag{3.3}$$

Where $g_1, ..., g_k$ are i.i.d samples drawn from Gumbel(0, 1) [1]. By taking the argmax of these we have sampled from the categorical distribution. However, since we cannot backpropagate through the not differentiable argmax, Jang et al. proposed approximating it with the continuous differentiable softmax. The output of this sampling mechanism is y_i, a continuous soft variable instead of an one-hot encoded vector.

$$y_i = \frac{\exp\left(\left(\log\left(\pi_i\right) + g_i\right)/\tau\right)}{\sum_{j=1}^{k} \exp\left(\left(\log\left(\pi_j\right) + g_j\right)/\tau\right)} \quad \text{for} \quad i = 1, \ldots, k \tag{3.4}$$

Temperature τ determines how close the sample y_i, is to one-hot encoded, for $\tau \to 0$, the Gumbel-Softmax distribution becomes identical to the categorical distribution that we are approximating, and the samples become one-hot encoded. Thus, in the backward pass of the VAE with discrete latent variables, the gradients are taken w.r.t. the soft samples yi, allowing for backpropagation. For completeness, we give the Gumbel Softmax distribution density:

$$p_{\pi,\tau}\left(y_1, \ldots, y_k\right) = \Gamma(k)\tau^{k-1}\left(\sum_{i=1}^{k} \pi_i / y_i^{\tau}\right)^{-k} \prod_{i=1}^{k} \left(\pi_i / y_i^{\tau+1}\right) \tag{3.5}$$

[1] Gumbel(0,1) distribution can be sampled using inverse transform sampling. Firstly, we draw $u \sim$ Uniform(0,1) and then compute $g = -\log(-\log(u))$

Chapter 4

Dataset Creation

This chapter discusses the nature of the data, the pre-processing steps, and the creation of the datasets used in the experiments.

The human DNA sequence data for each chromosome was obtained from the USCS Genome Browser in the FASTA format, a text-based format commonly used in bioinformatics to represent nucleotide or amino acid sequences. The version of the USCS data used was hg38 (GRCh38 Genome Reference Consortium Human Reference 38), which was released on December 2013.

To create the labelled dataset, RepeatMasker was used to mask the parts of the sequence corresponding to the repeats of interest. A version of all the chromosomes with the repetitive regions masked can be obtained from the USCS server, where the masking contains both the tandem and the interspersed repeats. However, in this work, we have focused on the analysis and identification of short tandem repeats (STRs). Thus, to create the appropriate dataset containing only the tandem repeats as positive examples, the RepeatMasker was run with the -noint (no interspersed) option to include only the tandem repeats. Tandem repeats were masked with lowercase letters. Using the masked data we created the datasets with the format explained in the following section.

4.1 Dataset Creation

This section will describe how the different datasets that were used to train our models were created. As described in Section 2.1.1, we aim to explore two different objectives with our models. The first one is to detect and identify the STRs. The second one is to explore the capability of our models to find patterns in the flanking sequence, namely the sequence that is adjacent to the repeat that would enable them to accurately

predict the emergence of a repeat. For the training towards the fulfillment of the first objective, the identification of the STRs, we created two different datasets.

1. The first dataset contains 80bp windows where the starting point of the repeat is found between positions 0-20. This ensures that the window contains a substantial part of the repeat that will allow the model to identify it. This dataset was created by firstly separating each chromosome into 80bp windows. Then in each positive window the starting point of the repeat was identified. A number between 0-20 was randomly sampled and the starting point of the repeat was placed there. This dataset contains each repeat one time. We will refer to this dataset as **Non-overlapping windows Dataset**.

2. The second dataset contains 80bp windows where the starting point of the repeat could be in any position. This dataset was created by sliding a 80bp window over the chromosome. Each window was labelled as positive (containing a repeat) if at least one bp of the repeat was contained in it. Any window not containing at least one bp of a repeat was labelled as negative. The sliding window scanned the chromosome with a step of 10. The step was required to reduce the amount of resulting data. From the above procedure, it can be inferred that, in this dataset, each repeat was contained more than once. In other words, different samples of the dataset could contain the same repeat at different positions. We will refer to this dataset as **Sliding windows Dataset**.

For the models targeting the second objective, we have created a dataset where positive samples are comprised of the 60bp windows met exactly before a repeat starts. For negative samples, we ensured that no repeat existed for at least 60bp of the sequence following the sample. We also created a dataset in the same way but leaving a 10 bp gap between the repeat and the windows labelled as positive samples. This is to account for potential mistakes in annotation of the RepeatMasker, e.g. RepeatMasker misannotating the initial part of a repeat. This way, we ensure that there is no misannotated part of the repeat included in the positive samples.

4.2 Creation of Train/Test splits

Naturally, the amount of negative data is much larger than the amount of positive since positive data are restricted to the locations of TRs, which comprise around 3% of the whole genome. Since accuracy and f1 score are the main metrics that we used

for the evaluation of our models, we balanced the classes and used the same amount of negative samples as positives. For that purpose, we applied random under-sampling on the negative class. Under the assumption that the critical part of information that we want the model to capture lies on the positive samples that contain the repeated patterns, under-sampling the negative class does not involve any risk of loosing potentially useful or critical information.

For all other experiments, apart from the ones on the datasets coming from sliding windows, we used two different train/test splits schemes. We trained our models using data from chromosome 2-22 and 10% of the data were left-out of the training procedure and were used as test data. The second testing dataset was comprised of repeats coming from chromosome 1. Chromosomes 1 was left out of the training procedure to evaluate how the models performed on identifying repeats from never-before-seen chromosomes. In both cases, testing data did not overlap training or validation data.

For the experiments on the datasets coming from sliding windows, the vast amount of data resulting from each chromosome does not allow for the use of the data from all chromosomes. The amount of positive samples resulting from scanning chromosome 19 with a 80bp sliding window and a step equal to 10 is larger than 500,000 samples. For that reason, we used only data from chromosome 19 for the models trained on sliding window data and the models' performance was measured on data coming from chromosomes 1. We selected chromosome 19 since in humans, it is the one with the highest density of STRs [8].

Chapter 5

Methods

This Chapter discusses the specific architectures and optimization settings that were explored for targeting the different objectives of this work and describes the ones that were eventually selected. The theoretical background of the proposed architectures has been extensively covered in Section 3.2. In Chapter 2, the input and output format of the models, which is determined by the problem definition for each objective, has been introduced. The current chapter focuses on the specific architectures explored, the optimization settings, e.g. loss functions used, and the hyperparameter tuning procedure.

5.1 Implementation

The DNA data as explained in Chapter 2 consists of a sequence of A,G,T,C letters. Before training, the input raw data are one-hot encoded. That is, A is encoded to (1 0 0 0), C is encoded as (0 1 0 0), G is encoded as (0 0 1 0), T is encoded as (0 0 0 1). Recall that every letter actually denotes a base pair (bp), as A always binds with T, G always binds with C, and vice versa. The letter N, which is contained in the input data in positions where the nucleic acid is unknown, was encoded as (0 0 0 0). As a result, the input to our models is a $4 \times l$ matrix, where l is the chosen length of the input sequences. In our case, l is equal to 80.

The hyperparameters for each architecture were chosen with random search among the hyperparameter space selected. The hyperparameter space for the CNN architecture is shown in Table 5.1 and 5.2, for the Gumbel-Softmax VAE in Table 5.3 and in Table 5.4, and for the vanilla VAE in Table 5.5 and Table 5.6. 150 models were trained for each architecture for each of the Experiments I, II and IV. For Experiment III, the models trained in Experiment II were used, since Experiment III aims

at evaluating the performance of the already trained models on the task of exactly annotating the sequence.

The hyperparameters were selected based on the accuracy and F1 score among the models trained on the testing set, which comprised 10% of the overall data. For the CNN model this is straightforward. For the VAE models, the accuracy and F1 score concern the classification performed in the latent space after the training of the VAE. As will be further discussed in Section 5.3.2, the way we use the VAE architecture to target our objective of predicting the repeats is the following. First, the VAE is trained in the standard unsupervised setting. Our aim is for the model to learn meaningful representations of the data in the latent space. That is, we aim at creating a latent space where positive samples, i.e. the ones containing repeats, will be encoded in a different way than negative samples. Thus, after having trained the VAE, we use its encoder part to encode the (positive and negative) input data into the latent space before performing classification using a feedforward NN.

We note that when we refer to hyperparameter tuning for the VAE case, we refer to the hyperparameters of the VAE model and not of the feedforward NN that was used to perform classification in the VAE's latent space. The classification in the latent space is a relatively easy task for any powerful enough NN and, thus, we did not perform any hyperparameter tuning for it. For the classification in the latent space, we selected a feedforward NN with 2 hidden layers, each comprising 128 hidden units. When we refer to the best model selected during the hyperparameter tuning procedure, we refer to the one that resulted in the best combination of accuracy and F1 score in the classification performed in the latent space. More details on that will be given in Section 5.3.2, after some discussion of theoretical background.

All the experiments involved a 2-class classification problem. For that purpose, binary-cross entropy was used as the loss function. We note that for the VAE case, we distinguish the 2-class classification performed in the latent space with the training of the VAE. More on the loss function selected for the training of the VAE will be discussed in Section 5.3.1. The optimizer used was Adam [69] and the learning rate was one of the tuned hyperparameters (between the values 1e-5 and 1e-4). The batch size was also tuned, with the values explored being 32, 64, and 128 for CNN and 64, 128, and 256 for VAE models. For the CNN case, the number of epochs was selected to be 100. The number of epochs for training the VAE was selected to be 200, while for the classification in its latent space, the feedforward NN was trained for a maximum of 50 epochs. Early stopping was performed in case the validation accuracy was non-increasing for 10 continuous epochs to prevent overfitting to the

training data. All models were implemented in Pytorch [70] and run on NVIDIA GeForce GTX 1080 Ti GPUs machines with 12GB memory.

5.2 Model Architecture: CNN

For the CNN, both the case of 1D and 2D convolutions were explored. In the first case, we view the input as a 1D matrix, of size $1 \times l$ with four input channels. In the second case, we view the input as a 2D matrix of size $4 \times l$, with one input channel. Also, in the case of 2D convolutions, for the first convolutional layer, we explored sizes of filters $(1 \times n)$ and $(n \times n)$. This was because filters of size $(1 \times n)$ were considered a more natural choice for the task of identifying repeats in sequences. In more detail, $(n \times n)$ filters on the first convolutional layer typically correspond to letter combinations and motifs, as explained in Section 3.2.2.1. We consider that, in the first convolutional layer, filters of size $(1 \times n)$ scanning both the vertical and horizontal direction (for the case of 2D convolution) are more likely to correspond to repeat patters. In Table 5.2, the search space for the hyperparameters related to this 2D versus 1D convolutions exploration can be seen. In Table 5.1, the hyperparameter search space for the rest of the parameters is shown. Padding was selected to be 2. In both tables the best identified hyperparameters for each experiment are shown.

For example, for Experiment II, the selected architecture consists of 2 convolutional layers. The first layer is comprised of 25 filters of size (1×6) and the second layer of 32 filters of size (6×6). The stride was selected to be 1. Each convolutional layer is followed by a Rectified Linear Unit activation function (ReLU). Then after each ReLU, a max pooling layer is following with a kernel size of (2×2) and a pool stride of (2×2). After the second convolutional layer, two fully connected layers follow. The number of units in the first fully connected layer is determined by the size of the output of the last convolutional layer. The number of units in the second fully connected layer was selected to be 64. After each of the convolutional layers, dropout [71] with a rate of 0.2 is applied to avoid overfitting. Finally, a sigmoid function is applied to the output to obtain a probability for the 2-class classification problem. The network's output then corresponds to the probability that the input belongs to the positive class.

Hyperparameter	Search Space	Selected Exp I	Selected Exp II	Selected Exp IV
Batch size	[32, 64, 128]	64	32	32
Learning rate	[1e-3, 1e-4, 5e-5]	5e-5	5e-5	5e-5
No. conv. layers	[1,2,3]	2	2	2
No. conv. filters	[8, 16, 25, 32]	32	25	25
No. Fully connected layers (fcl)	[1,2]	2	2	2
No. of neurons in fcl	[32, 64, 128]	128	128	64
Dropout	[0.0, 0.2, 0.5]	0.2	0.2	0.2

Table 5.1: *Part I of the hyperparameter Search Space for CNN. The first column lists the hyperparameters tuned and the second column the corresponding values explored. Columns four to six denote the best choice identified for each experiment.*

	2D	1D	Selected Exp I	Selected Exp II	Selected Exp IV
Size of conv. filters layer 1	[(5, 5), (7, 7), (1, 5), (1, 6), (1, 7)]	[5, 6, 7]	6	(1, 6)	5
Size of conv. filters layer 2	[(5, 5),(6, 6),(7x7)]	[5, 6, 7]	6	(6, 6)	5
Conv. layer stride	[(1, 1), (1, 2)]	[1, 2]	1	(1, 1)	1
Max pooling kernel size	[(2, 2), (3, 3)]	[2, 3]	2	(2, 2)	2
Max pooling stride	[(2, 2), (3, 3)]	[1,2]	2	(2, 2)	2

Table 5.2: *Part II of the hyperparameter Search Space for CNN. The first column lists the hyperparameters tuned and the second column the corresponding values explored. Columns four to six denote the best choice identified for each experiment.*

5.3 Model Architecture: VAE

In the following section, we will first highlight some of the different options we have for the VAE training configuration. Then, we will discuss some main challenges related to the training of VAEs and connect them to our choice of exploring the Gumbel-Softmax VAE, in addition to the standard vanilla VAE. Finally, we will give the specific details on the hyperparameter search space and on the architecture we used for each experiment, taking into account all the aforementioned issues and challenges.

5.3.1 Optimization configuration

For the VAEs, two different optimization configurations have been explored in terms of the loss function. As discussed in Section 3.2, variational autoencoders are trained by maximizing a lower bound for $\log p(\mathbf{x})$, the ELBO introduced in Eq. 3.2, which we restate here in Eq. 5.1 for convenience.

$$\mathcal{L}_{\theta,\phi}(\mathbf{x}) = \log p_{\boldsymbol{\theta}}(\mathbf{x}) - \mathrm{KL}\left(q_{\boldsymbol{\phi}}(\mathbf{z} \mid \mathbf{x}) \| p_{\boldsymbol{\theta}}(\mathbf{z} \mid \mathbf{x})\right) \leq \log p_{\boldsymbol{\theta}}(\mathbf{x}) \tag{5.1}$$

The maximization of the ELBO both approximately maximizes $\log p_{\boldsymbol{\theta}}(\mathbf{x})$ and minimizes the KL divergence leading to a better approximation of $p_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$ by $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$. This equation can be reformulated as follows:

$$\mathcal{L}_{\theta,\phi}(\mathbf{x}) = -\underbrace{\mathbb{E}_{\mathbf{z}\sim q(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x} \mid \mathbf{z})]}_{\text{reconstruction error}} + \underbrace{\mathrm{KL}\left(q_{\phi}(\mathbf{z} \mid \mathbf{x}) \| p(\mathbf{z})\right)}_{\text{regularization}} \tag{5.2}$$

The first term corresponds to a reconstruction loss, measuring how effectively the decoder has learned to reconstruct the data. The second term acts as a regularizer to the way the encoder is encoding the input data in the latent space, namely encouraging the approximate posterior $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ to be close to a prior $p(\mathbf{z})$, which is typically the (multivariate) standard Gaussian distribution $N(0, 1)$ in the case of VAE with continuous latent variables.

For the reconstruction error term, we have explored two different options. The standard one, used in Variational Autoencoders trained on binarized data, is the binary cross entropy (BCE). When BCE is used, a sigmoid function is applied on the last layer of the decoder, which transforms each element of the output to a probability, i.e. to a value between 0 and 1. However, in our case the inputs are not just binarized, but correspond to one-hot encoded data. Because of that, we have explored the use of the categorical-cross entropy. In that case, instead of the sigmoid function, a softmax function is applied over the vertical direction of the final layer of the decoder. We

will make the reason behind this more clear. In the VAE setting, where the NN is aiming at reconstructing the $4 \times l$ one-hot encoded input, every four output elements on the first (vertical) dimension correspond to an one-hot encoded character. Thus, applying softmax over these 4 elements of the output of the decoder is equivalent to creating a prediction for this character. Then, the reconstruction error of Eq. 5.2 is taken as the sum of all the categorical cross entropies between the softmax-activated prediction for each character and the true class label, i.e the true character.

This modification of the reconstruction error term of the objective function has been used in various works in the literature when the input is a sequence of one-hot encoded elements [72], [73]. During our architecture exploration, we explored both aforementioned options. The loss function that produced better results was BCE.

Challenges on training VAEs: Posterior Collapse

The aim of our VAE model is to learn meaningful latent representations. A model that achieves that will have a nonzero KL divergence term and a relatively small reconstruction term [74] in Eq. 5.2. However, the training of VAE is complex and issues may occur. Such an issue is the "posterior collapse" [74] [75] where the model's variational distribution $q_{\boldsymbol{\phi}}(\mathbf{z}|\mathbf{x})$ closely matches the uninformative prior $p(\mathbf{z})$ [76]. This is an issue that emerges because we are trying to minimize the KL term in Eq. 5.2. In the case that posterior collapse occurs, the KL term of the cost function goes to zero. Despite that, the model is still able to achieve likelihoods that are close to the optimal [74], meaning high values of ELBO. However, that by itself does not guarantee the usefulness of the model, but should be accompanied by a nonzero KL term. As a result, both of these two terms should be investigated to evaluate the usefulness of the model. Various workarounds have been proposed to alleviate the issue of posterior collapse. Some works have proposed less powerful encoder-decoder architectures to direct the model towards making greater use of the latent space [3] while others to artificially weaken the decoder e.g. through the use of word-dropout [75].

Another solution that has been proposed in [74] is the annealing of the KL term in Eq. 5.2. A variable weight is added to the KL term in the cost function. This weight is annealed from a zero value, at the start of the training to a maximum value of one. This way the model is encouraged to encode as much information as possible in the latent space. Then, as the KL annealing weight increases, the encodings in the latent space are smoothed according to the prior, since the KL term between the

approximate posterior and the prior is starting be taken fully into account in the loss function.

During the selection of the architecture, of the hyperparameters explored and of the optimization configuration we have taken all the above into account. In more detail, we have avoided particularly deep and powerful architectures in the encoder/decoder that could potentially contribute to the posterior collapse. Also, we apply KL annealing for the first 50 epochs, with the weight of the KL term in Eq. 5.2 equal to 0 at the start of the training and then progressively increasing, reaching reaching a value of 1 in epoch 50.

5.3.2 Classification on the latent space

The goal of unsupervised training of the VAE is to extract a meaningful representation in the latent space. As a next step, we perform classification over it. In more detail, the trained encoder is used to encode the negative and positive samples into the latent space. These lower-dimension encodings in the latent space that correspond to the higher-dimensional initial positive and negative data, are fed as an input to a simple feedforward neural network. This NN is trained to perform classification and the performance is evaluated in terms of the accuracy and F1 score. Note that we selected the hyperparameters that led to the best classification in the latent space in terms of the accuracy and F1 score rather than the hyperparameters leading to the maximum ELBO. That is because, as discussed before, maximum ELBO does not always coincide with meaningful latent representations; e.g. in case posterior collapse occurs, the ELBO can have a high value without meaningful latent representations and, thus, without good classification performance on the latent space. For example, we observed that the choice of extremely powerful encoder and decoder can easily lead to posterior collapse but will also lead to a high value of the ELBO.

5.3.3 Discrete latent variables: Reasoning behind Gumbel-Softmax VAE

In this Section, we will provide the reasoning behind the usefulness of discrete latent variable models and, as a result, the reasoning behind our choice to explore the Gumbel-Softmax VAE for our problem.

In language models, we have sequences of words or character tokens, which are inherently discrete. In the same way, genomic data are represented as discrete sequences

of symbols. This inherent nature of the genomic data suggests the exploration of models with discrete latent variables as a natural choice. As a result, many works using generative models on genomic data have used discrete latent representations [77], [78], [77], [79].

Discrete variables are more interpretable and have a natural correspondence to classes/categories of the data [80]. While using discrete latent variables has been proven to be challenging, mainly because of issues with backpropagation, many approaches have been proposed to work around these issues. An example is the Gumbel-Softmax reparameterization trick introduced in [5], that we described in Section 3.2.3.2. Many works have proposed different ways and architectures to train VAEs to allow for discrete latent variables [81], [80], [82]. Modifications of the variational autoencoder framework where the encoder outputs discrete, rather than continuous latent representations have been shown to alleviate the problem of "posterior collapse" [80], [83], [84]. This is of high importance since, as described above, posterior collapse is a serious and complex issue where many latent variables are ignored and, thus, latent representations are rendered useless.

In the current work, taking into account the inherently discrete nature of genomic data and inspired by these studies, we have implemented a VAE with discrete latent variables using the Gumbel-Softmax reparameterization trick, which will be referred to as Gumbel-Softmax VAE. We note that for the Gumbel-Softmax VAE, the KL term in Eq. 5.2 corresponds to the KL divergence between categorical distributions, since the latent variable $\mathbf{z}$ is discrete. Thus, the prior selected is the discrete uniform distribution instead of the standard Gaussian used in the vanilla VAE case. Also, in Gumbel-Softmax VAE, we have two different parameters for the latent dimension.

5.3.4 Selected architecture

Tables 5.3 and 5.4 describe the hyperparameter space explored and the best hyperparameter choice for each experiment for the model of Gumbel-Softmax VAE. Tables 5.5 and 5.5 describe the hyperparameter space explored and the best hyperparameter choice for each experiment for the model of vanilla VAE. Both models of VAE contain one or two convolutional layers. Again, like in the case of CNNs, both the case of 1D and 2D convolutions were explored for the convolutional layers of the VAE.

Hyperparameter	Search Space	Selected Exp I	Selected Exp II	Selected Exp IV
Batch size	[64, 128, 256]	128	128	128
Learning rate	[1e-4, 1e-5]	1e-5	1e-5	1e-5
No. conv. layers	[1, 2]	2	1	2
No. conv. filters	[8, 16, 25, 32]	8	16	25
No. Fully connected layers (fcl)	[1, 2]	2	2	2
No. of neurons fcl	[32, 64, 128]	128	128	128
Latent dimension	[4, 6, 8, 10]	6	6	8
Dropout	[0.0, 0.2, 0.5]	0.2	0.2	0.2

Table 5.3: *Part I of the hyperparameter Search Space for Gumbel-Softmax VAE. The first column lists the hyperparameters tuned and the second column the corresponding values explored. Columns four to six denote the best choice identified for each experiment.*

Hyperpar.	2D	1D	Selected Exp I	Selected Exp II	Selected Exp IV
Size of conv. filters layer 1	[(5, 5), (7, 7), (1, 5), (1, 6), (1, 7)]	[5, 6, 7]	6	(1,7)	5
Size of conv. filters layer 2	[(5,5),(6x6),(7x7)]	[5, 6, 7]	6	-	5
Conv. layer stride	[(1,2), (1,3)]	[1, 2]	1	(1,1)	1
Max pooling kernel size	[(2x2), (3x3)]	[2, 3]	2	(4,3)	2
Max pooling stride	[(1,2), (1, 3)]	[1,2]	2	(1,3)	2

Table 5.4: *Part II of the hyperparameter Search Space for Gumbel-Softmax VAE. The first column lists the hyperparameters tuned and the second column the corresponding values explored. Columns four to six denote the best choice identified for each experiment.*

Hyperparameter	Search Space	Selected Exp I	Selected Exp II	Selected Exp IV
Batch size	[64, 128, 256]	128	256	128
Learning rate	[1e-4, 1e-5]	1e-5	1e-5	1e-5
No. conv. layers	[1, 2]	2	2	2
No. conv. filters	[8, 12, 16, 25]	12	16	12
No. Fully connected layers	[1, 2]	2	2	2
No. of neurons fcl	[32, 64, 128]	128	128	128
Latent dimension	[4, 6, 8, 10]	8	10	8
Dropout	[0.0, 0.2, 0.5]	0.2	0.2	0.2

Table 5.5: *Part I of the hyperparameter Search Space for vanilla VAE. The first column lists the hyperparameters tuned and the second column the corresponding values explored. Columns four to six denote the best choice identified for each experiment.*

Hyperpar.	2D	1D	Selected Exp I	Selected Exp II	Selected Exp IV
Size of conv. filters layer 1	[(5, 5), (7, 7), (1 × 5), (1 × 6), (1 × 7)]	[5, 6, 7]	7	5	5
Size of conv. filters layer 2	[(5,5),(6x6),(7x7)]	[5, 6, 7]	7	5	5
Conv. layer stride	[(1,2), (1,3)]	[1, 2]	1	1	1
Max pooling kernel size	[(2x2), (3x3)]	[2, 3]	2	2	2
Max pooling stride	[(1, 2), (1, 3)]	[1, 2]	2	2	2

Table 5.6: *Part II of the hyperparameter Search Space for vanilla VAE. The first column lists the hyperparameters tuned and the second column the corresponding values explored. Columns four to six denote the best choice identified for each experiment.*

Chapter 6

Results

In Section 3.2, the theoretical background of the deep learning models, which are implemented in this work, has been introduced. In Chapter 5, the specific architectures and implementation details have been discussed. In the current chapter, we present the results using the aforementioned models in different experiments. The different objectives, the experiments associated with each, while also the different training and evaluation datasets and metrics are summarized in Table 6.1.

As discussed in Section 2.1.1, the first objective of this work is to create models that are able to identity short tandem repeats (STR). We note that throughout this section when we use the word repeats, we refer to short tandem repeats, which is the type of data we aim to detect and analyse in the current work. The objective of identifying STRs is further split in two parts. The first part, referred to as objective I.I, is for the models to be able to predict if a certain window contains a repeat. The second part, referred to as objective I.II, is for the models to be able to exactly annotate a sequence in terms of the presence of an STR. That is, the models should find the exact starting and ending position of the repeat contained in an input sequence. The second objective, which will be referred to as objective II, is to explore if the flanking sequence before the repeat, signals the repeat coming, that is, to investigate if the NN can predict the emergence of a repeat only from its upstream flanking sequence.

Experiment I and Experiment II both deal with the 2-class classification problem of predicting if a window contains a repeat (objective I.I). The difference between them lies in the dataset that is investigated. In Experiment III, we use the models to attempt the exact annotation of input sequences (objective I.II). Lastly, Experiment IV again constitutes a 2-class classification problem, where the models are used to predict if the input sequence is a flanking sequence of a repeat or not, that is, if after this sequence a repeat is immediately following.

Objective	Description	Experiments	Training Dataset	Evaluation Dataset	Evaluation Metrics
Obj. I.I	Predict if an input window contains an STR	Exp. I	"Non-Overlapping windows" / type I	Held-out part of the training Dataset	Acc., F1
		Exp. II	"Sliding windows"/ type II		
Obj. I.II	Annotate STRs (find the start/end position)	Exp. III	"Sliding windows"/ type II	Curated dataset	Jaccard
Obj. II	Predict if a window comprises upstream flanking sequence	Exp. IV	"Flanking seq."	Held-out part of the training Dataset	Acc., F1

Table 6.1: *Table summarizing the different experiments, training/evaluation datasets and metrics used for each objective.*

6.1 Experiment I & II

Experiment I and Experiment II both deal with objective I.I. The difference between the two experiments is that they involve different types of datasets during training and testing. The dataset used in Experiment I is the **Non-overlapping windows Dataset** (where the repeats are placed on the initial positions of the samples), which will also be referred to as type I dataset. The dataset used in Experiment II is the **Sliding windows Dataset** (where the starting position of the repeats is placed randomly), which will also be referred to as type II dataset. The nature of these datasets and the way they were created was discussed in detail in 2.1.1 and Chapter 4, respectively. In both Experiment I and II, the model predicts if a window contains a repeat or not. The evaluation metrics used to assess the model's performance are Accuracy, F1 score, Precision and Recall. The models of Experiment I will be referred to as **type I models**, since they were trained on the type I dataset, and the models of Experiment II as **type II models**.

We present the results for Experiment I in Section 6.1.1, and the results of Experiment II in Section 6.1.2. Since both experiments target the same objective but make use of different training datasets, we will compare their results and identify the type of training dataset that is more appropriate to use. This is of high importance, since the nature of the training dataset greatly affects the model performance.

6.1.1 Experiment I

The **Non-overlapping windows Dataset**, as explained in Chapter 4, contains 80 bp samples, where the starting position of the repeats is placed randomly in position 0-20 of the samples. That means that for a certain sample, if the start of the repeat is randomly placed at the 10th position, then the sample will contain 9 nucleotides (nt) of the flanking sequence before the repeat. Then, on the 10th position, the repeat's first nt will be placed. Depending on the length of the repeat the sample could contain part of the downstream flanking sequence that follows the repeat. For

Figure 6.1: *Examples of 80 bp samples.*

	CNN		Vanilla VAE		GS VAE	
Seq Length	Acc	F1	Acc	F1	Acc	F1
80bp	0.957	0.956	0.838	0.823	0. 906	0.903

Table 6.2: Performance on Experiment I on 10% of the data from chr 2-22 that was held-out of training

example, in the first row of Fig. 6.1, an 80 bp sample where the starting point of the repeat has been placed on the 10th position is shown; since the length of the repeat is smaller than the remaining 70 bp, a part of the sequence that follows the repeat is also contained in the sample. The second row of Fig. 6.1 depicts a sample that would not be contained in the Non-overlapping windows Dataset, since the starting point of the repeat is not placed on positions 0-20 of the sample. This type of dataset ensures that a substantial part of the repeat is included in the sample.

We ran the three model architectures to classify the dataset sequences in terms of if they contain a repeat or not. Table 6.2 shows the results in terms of the accuracy and F1 score for our three different architectures: CNN, vanilla VAE and Gumbel-Softmax VAE. For the variational autoencoder models, the evaluation metrics are referred to the 2-class classification performed in the latent space, as was explained in Chapter 5.

The metrics in Table 6.2 concern the test dataset, which is 10% of the total dataset, that was left out of training and was used only for testing. The training and testing dataset included samples from chromosomes 2-22. We did not include samples from chromosome 1 in the training procedure, so that we also test the performance of our models on data from never-before seen chromosomes. The performance on the never-before-seen chromosome 1 is shown in Table 6.3. We observe that the models can achieve the same level of performance on the never-before-seen chromosomes.

We observe that our models, especially the supervised CNN, achieve high similarity to RepeatMasker in terms of predicting whether a repeat is contained in the

	CNN		Vanilla VAE		GS VAE	
Chr	Acc	F1	Acc	F1	Acc	F1
1	0.957	0.956	0.83	0.818	0.905	0.902

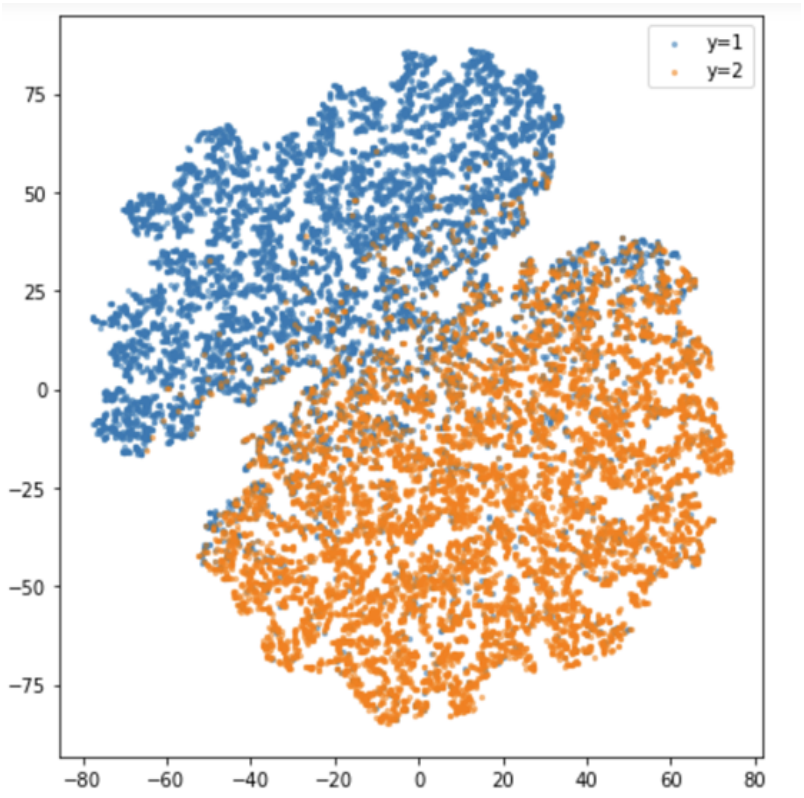
Table 6.3: Performance on Experiment I on never-before-seen chromosome 1

input window. CNN achieves better performance in comparison to the unsupervised VAE models. However, we must consider the fact that in evaluating the performance, we accept as ground truth the RepeatMasker annotation. Thus, we expect the supervised model, which has been trained on labelled data annotated in the same way as the testing dataset, to achieve better performance. The possible advantage of the VAE is that, since it does not rely on any labelled data during training, it could potentially be more flexible. In more detail, the idea behind explorating the applicabilty of VAEs is to create a structured latent space where repeats are separated (encoded in a different way) than non-repeats. The way the data are encoded in the latent space is independent of any labels, since VAEs are unsupervised models, and thus, any fallibility of the RepeatMasker annotation does not influence the model. One way in which this flexibility could be exploited would be the ability to discover repeats not found by RepeatMasker.

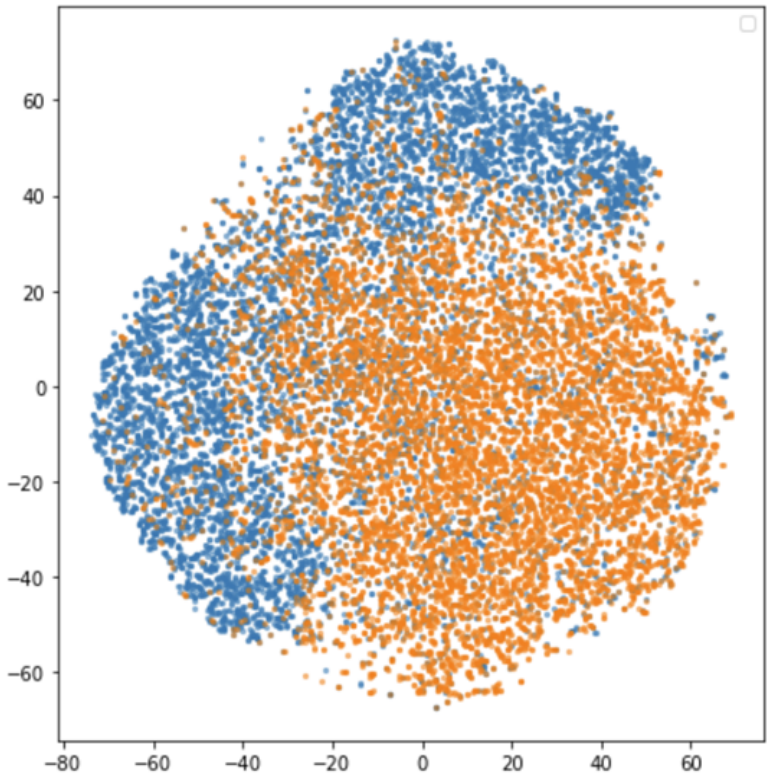
Despite the lower performance in comparison with the CNN, both VAE architectures are able to create a latent space where the negative samples are separated from the positives. In Fig. 6.2a, the t-distributed stochastic neighbor embedding (t-SNE) of the latent space encodings of the GS VAE is shown. t-SNE is a technique commonly used for the visualization of high-dimensional data in lower-dimensional spaces. To construct this figure, after the VAE model has been trained, we use the trained encoder to map the input samples to latent space encodings. In the case of the specific VAE model, the latent space is 6-dimensional. Then, the t-SNE of these 6-dimensional representations is taken. In Fig. 6.2b, the t-SNE [85] for the vanilla VAE is shown. The GS-VAE distinguishes more clearly between repeats and non-repeats in the latent space, which explains its better performance in Table 6.2.

Length of repeat in samples vs Accuracy

To explore how the length of the repeat included in the sample affects the accuracy of each model, we separated 100,000 positive, randomly selected, samples from chromosome 1 in terms of the amount of nt of the repeat that are included in the samples. For example, the sample shown in the top of Fig. 6.1, includes 36 nt that are contained in the repeat, while the sample on the bottom row contains 11 such nt. In Fig.



(a) Gumbel-Softmax VAE



(b) vanilla VAE

Figure 6.2: t-SNE of the high dimensional latent space. Negative samples (non-repeats) are shown with orange and positive samples (repeats) are shown with blue.

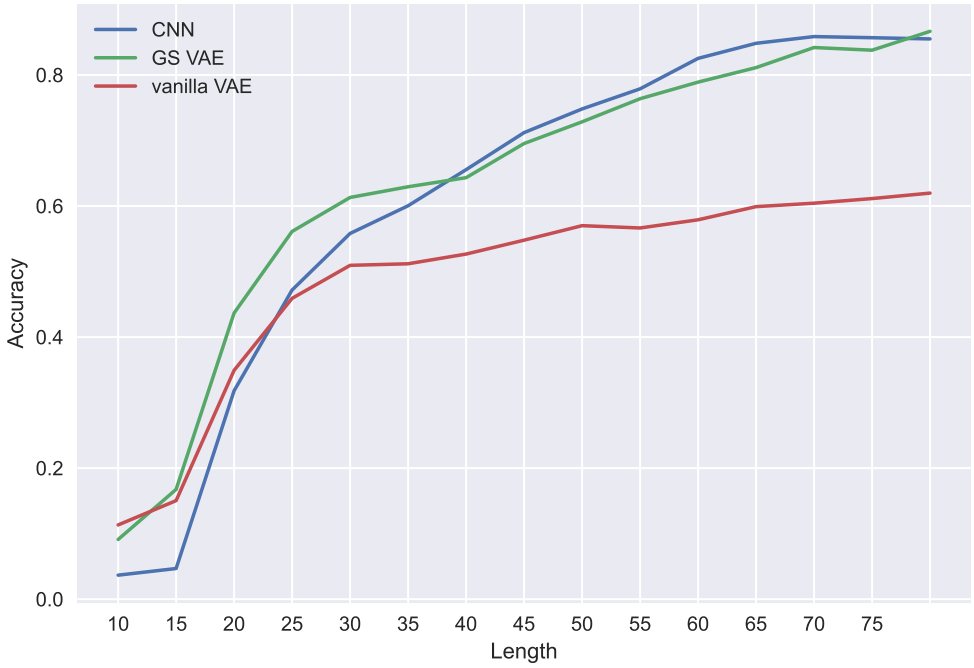


Figure 6.3: *Accuracy over the repeat length contained in the samples (type I models). We separated 100,000 sequences containing repeats from chr 1 based on the length of the repeat contained in the samples and measured the models' accuracy for each. The groups considered repeat lengths 6-10, 11-15 and so on. At the x-axis the upper limit of the range of each group is denoted. All models are more accurate when longer part of the repeat is contained in the input sample*

6.3, the accuracy of the three different architectures over the length of the repeat contained in the sample is shown. While for a low number of nt contained in the window, GS VAE performs better than the CNN, namely it identifies more repeats, for larger length of the repeat contained in the window, CNN slightly outperforms it. We note that only positive samples have been used for this figure, so that we can monitor how the repeat length contained in positive samples affects the prediction.

Effects of mutation in type I model performance: Identifying mutated repeats

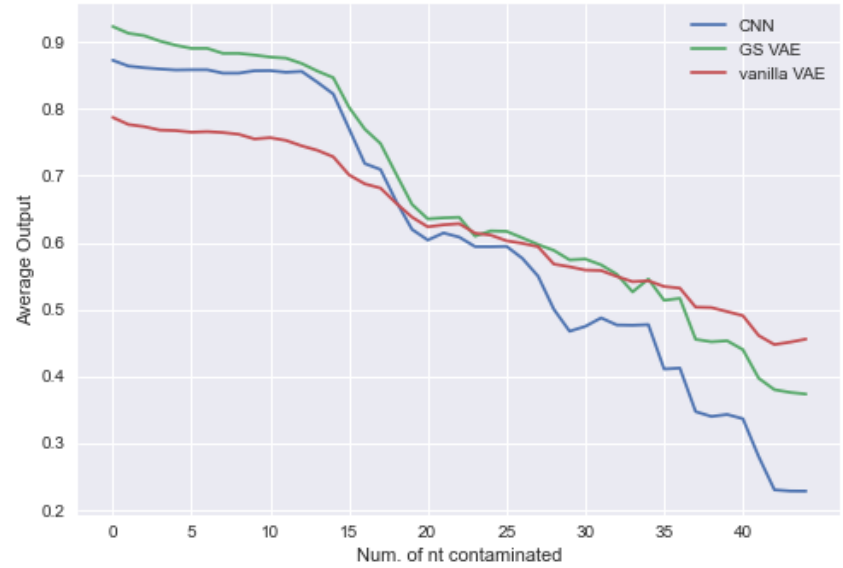
As described in Section 3.1, repeats are hypermutable. Thus, the repeat unit is, typically, impure, containing different insertions, deletions, or changes of nucleotides (nt). These repeat polymorphisms are quite important to detect, since in many cases they are associated with diseases, as described in Section 3.1.

To explore the tolerance of our models in polymorphisms, namely their ability to detect the repeats when these have been subject to mutations, we contaminate some of the samples with noise and monitor the models' prediction. Noise contamination is done by iteratively changing one nt at a random position and is a way to simulate mutations in the repeats. Fig. 6.4a shows the average model output among 10,000 sequences that contain a repeat, for different contamination/mutation levels. In more detail, the input sequences are 80bp repeats and we successively contaminate one extra bp of the repeat, namely we change it to another base. Thus, the y-axis indicates the (average) model output among the 10,000 repeat sequences, while the x-axis indicates the number of nucleotides that have been contaminated in each repeat sample. In 6.4a, we observe that all three models can identify the repeats even when they have been subject of extensive mutations. For example, we observe that even for 25 nt mutated, all models average output is above 0.6, indicating that the models still detect the repeats in a substantial part of the samples.

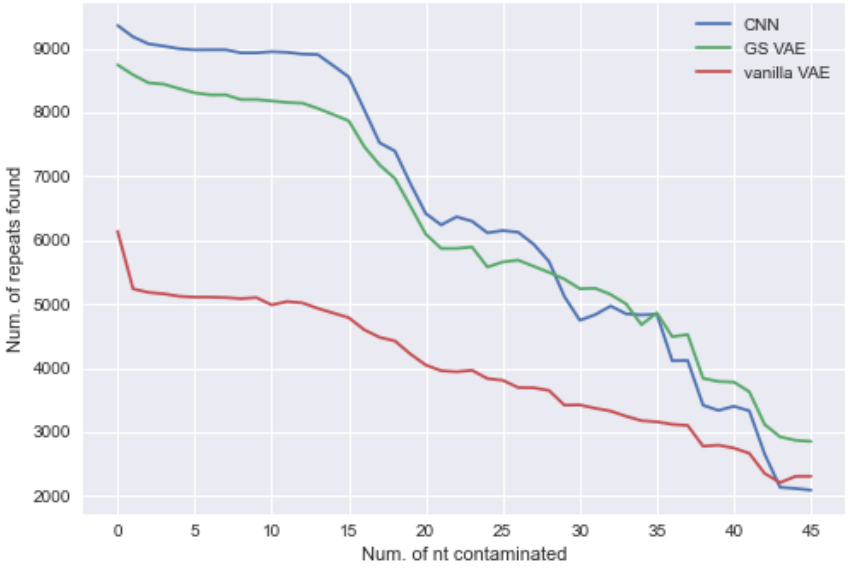
We note that in Fig. 6.4a, for each model, the average output among only the sequences that were identified as repeats for zero nt contaminated has been calculated. That is, for each type of model only the sequences that were initially predicted as being repeats have been considered, since we want to compare how the prediction changes when we mutate the sequences. We report that for zero contamination level, namely the initial samples, the CNN model identified 9,360 out of the 10,000 sequences as repeats, GS VAE identified 8,744, while vanilla VAE, 6,140. In Fig. 6.4b, we monitor how this initial number changes. That is, the number of sequences identified as repeats (out of the total 10,000) for different contamination levels, is reported for each model. We observe that CNN and Gumbel-Softmax VAE can detect a substantial part of the repeats despite the high contamination level. For example, for 30nt mutated, CNN and GS-VAE can detect around half of the repeats.

6.1.2 Experiment II

Experiment II uses the **Sliding windows Dataset**, which constitutes a more challenging dataset since the starting position of the repeat could be placed in any position of the window. Thus, there is no guarantee that the model will be able to observe a substantial part of the repeat in certain samples. This type of dataset implicitly connects with the objective III. In more detail, by training the model with this type of dataset, our aims are twofold. On the one hand, we aim to allow the model to explicitly learn to distinguish the repeat patterns from samples that contain a substantial part of the repeat, in the same way that this was done in Experiment I. On



(a) *Model average output (y-axis) among repeats from chr1 for different contamination levels (x-axis). The average for each model is calculated among the number of sequences each one identified as repeats for zero contamination level (x=0). Total number of sequences considered was 10,000.*



(b) *Number of repeats identified among 10,000 repeats from chr1, for different contamination levels (x-axis).*

Figure 6.4: *Modelling performance on mutated repeats (type I models): We contaminate the repeat sequences by iteratively changing one bp of the repeat to a random one to create artificial mutations. The no. of bp changed is denoted on the x-axis. (a) Average model output (b) No. of repeats identified, for different contamination levels.*

	CNN		Vanilla VAE		GS VAE	
Seq Length	Acc	F1	Acc	F1	Acc	F1
80bp	0.893	0.890	0.76	0.736	0.792	0.814

Table 6.4: Performance on Experiment II on 10% of the data from chr 19 that was held-out of training.

	CNN		Vanilla VAE		GS VAE	
Chr. num	Acc.	F1	Acc.	F1	Acc.	F1
1	0.84	0.832	0.708	0.691	0.799	0.770

Table 6.5: Performance on Experiment II on never-before-seen chromosome 1

the other hand, by including samples which contain a small part of the repeat and are comprised mostly of the flanking sequence before the repeat, we aim to allow the model to capture patterns that are potentially signaling the repeat and use them to make an accurate and early prediction of the repeat region.

Table 6.4 shows the results in terms of the accuracy and F1 score for our three different architectures on the test dataset, which is 10% of the total dataset that was left out of training and was used only for testing. The training and testing dataset for this experiment included samples only from chromosome 19. The reason for this is that the nature of the dataset, and the way it was created using sliding windows, which was described in Section 4.1, results in a vast amount of data coming just from one chromosome. As in Experiment I, we also test the performance of our models on data from never-before seen chromosomes. Table 6.5 shows the performance of the models on the never-before-seen chromosome 1.

As expected, the performance is lower in comparison with the results of type I models shown in Table 6.2. However, this lower performance is due to the more difficult nature of the evaluation dataset and does mean that type II models perform worse than type I models. To compare the two types of models we should evaluate them on a common testing dataset. The comparison between type I and type II models will be discussed in Section 6.1.3.

Effects of mutation in type II model performance: Identifying mutated repeats

Again, as in Section 6.1.1, we evaluate the performance of our models when we contaminate the repeats. The results are shown in in Fig 6.5a. All models are able to detect the repeat even when they are subject of many mutations. We report that for zero contamination level, the CNN model identified 9,744 out of the 10,000 sequences

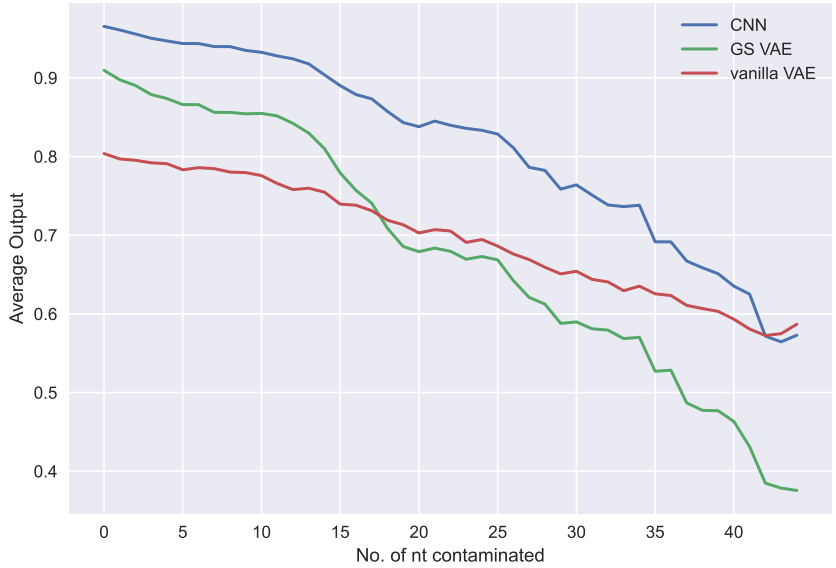
as repeats, GS VAE identified 8,926, while vanilla VAE 7,800. In Fig. 6.5b, we monitor how this initial number changes. That is, the number of sequences identified as repeats (out of the total 10,000) for different contamination levels, is reported for each model. We observe that all models can detect a substantial part of the repeats despite the high contamination level. For example, for 30nt mutated, vanilla VAE and GS-VAE can detect around half of the repeats, while the CNN model can detect more than 7,000 repeats.

6.1.3 Comparison of type I and type II models

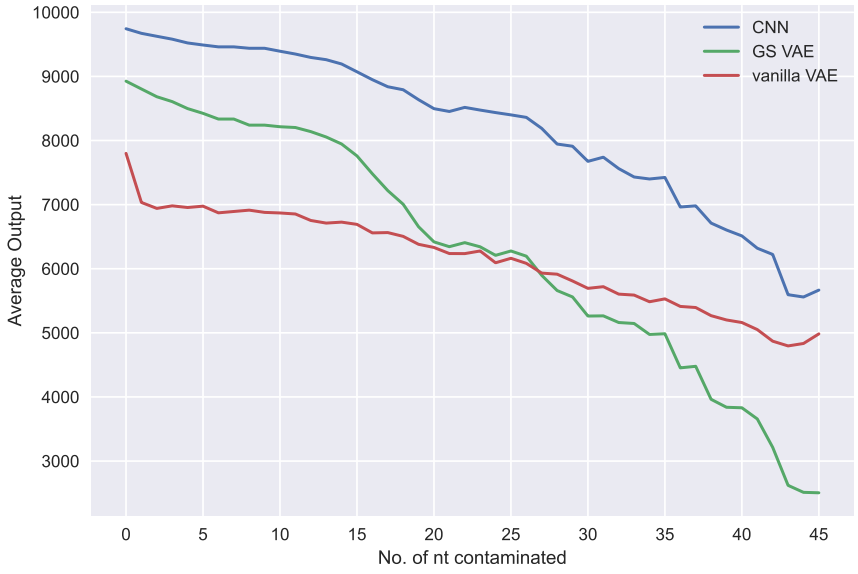
The comparison between Table 6.2 and Table 6.4 is not a fair indicator of the differences in performances between type I and type II models. The lower performance of type II models, shown in Table 6.4, in comparison with the performance of type I models, shown in Table 6.2, is expected due to the more difficult nature of the evaluation dataset used for the results in Table 6.4. In other words, it is not due to the models performing worse in identifying the repeats, but because of the nature of the type II dataset, which includes a significant number of samples containing only a few bp of the repeat. These samples are harder to identify as positive samples and, thus, their inclusion in evaluating type II models lowers their performance.

To quantitatively assess and confirm these assumptions, in Fig. 6.5a, the accuracy as a function of the length of the repeat included in the window (for 80 bp window) is given. With dotted lines are the curves from Fig. 6.3, which correspond to the type I models trained on the Non-overlapping windows Dataset, while the full lines correspond to type II models trained on the Sliding windows Dataset. The data used for this figure are the same as in Fig. 6.3, 100,000 positive, randomly selected, samples from chromosome 1 separated in terms of the amount of nt of the repeat that are included in the samples. This evaluation dataset, contains the repeats in random positions in the samples.

It is shown that for all lengths, type II models outperform type I models. The higher performance is particularly striking in cases where a small part of the repeat is included. A potential explanation would be that the model captures motifs and characteristics of the flanking sequence, namely the sequence just before the repeat and, thus, by exploiting this information is able to predict the repeat even when seeing a few bp. More on the capability of the model to capture characteristics of the upstream flanking sequence of the repeat and predict its emergence will be discussed in Experiment IV, in Section 6.3.



(a) *Model average output (y-axis) among repeats from chr1 for different contamination levels (x-axis). The average for each model is calculated among the number of sequences each one identified as repeats for zero contamination level (x=0). Total number of sequences considered was 10,000.*



(b) *Number of repeats identified among 10,000 repeats from chr1, for different contamination levels (x-axis).*

Figure 6.5: *Modelling performance on mutated repeats (type II models): We contaminate the repeat sequences by iteratively changing one bp of the repeat to a random one to create artificial mutations. The no. of bp changed is denoted on the x-axis. (a) Average model output (b) No. of repeats identified, for different contamination levels.*

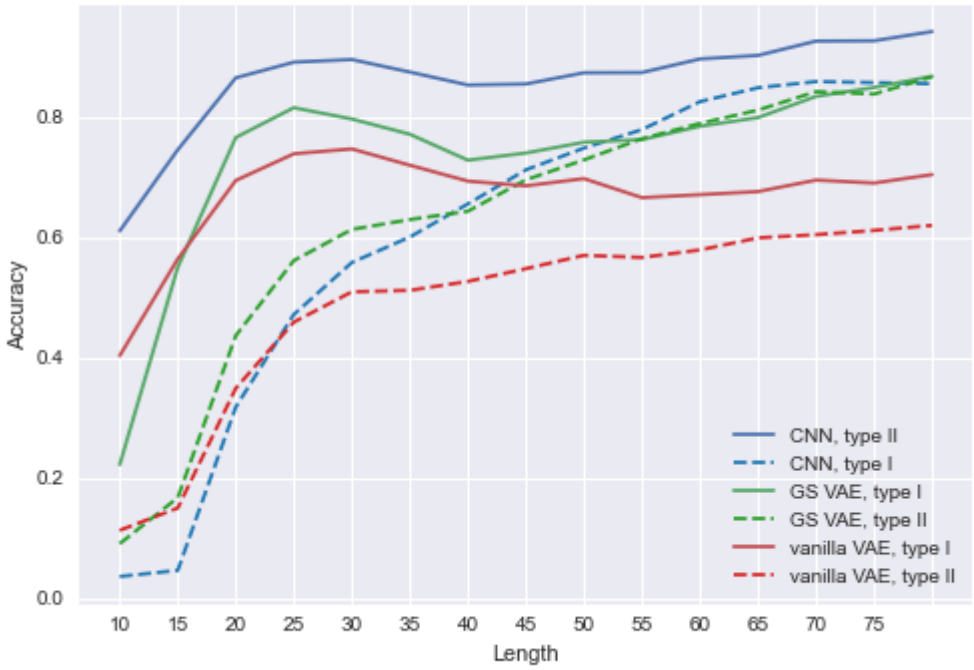


Figure 6.6: *Percentage of positive repeats found for different repeat lengths contained in the samples. Comparison between type I and type II models.*

Heatmaps

The comparison between type I and type II models can be presented qualitatively through the heatmaps shown in Fig. 6.8 – Fig 6.11. The heatmaps depict the models' prediction for 50 randomly selected samples. Fig. 6.8 and Fig. 6.10 show the heatmap for CNN and GS VAE type I models, while Fig. 6.9 and Fig. 6.11 for CNN and GS VAE type II models.

Each row corresponds to a sequence sample, and each column corresponds to the output of the model for an 80 bp window. The output of the model can be viewed as the probability of the input window containing a repeat. The window slides, namely is moved base-wise over the sequence and, at each position, we monitor how the output of the model changes. In the sequence samples used to create these heatmaps the starting position of the repeat has been placed in the same position for visualization purposes. However, the repeats are of different length and, thus, their ending position varies between samples. At position 1 of the x-axis, the window contains the first bp of the repeat. As the window slides over the sequence, at each step it contains one more bp belonging to the repeat.

To make the heatmap generation procedure clearer, we include an example of a window sliding over an input sequence in Fig. 6.7. In the example, the repeat region of the input sequence is shown with lower case letters, while the non-repeat region is indicated with uppercase letters. For the heatmap generation we slide an 80 bp window across that input sequence and, at each position, indicated by the position where the rightmost part of the window is we monitor the model output. In the first row the position of the window corresponds to the position 1 of the x-axis, while the sliding window in the second row of the image corresponds to the position 10 of the x-axis of the heatmap. Thus, ideally, we would want the model to start predicting the repeat at position 1 of the heatmap. However, realistically, the model will need to take as input at least a few bp of the initial part of the repeat before it can predict the presence of the repeat.

From the heatmaps in Fig. 6.8 – Fig 6.11, it is evident that the models from Experiment II can identify the presence of the repeat much earlier, namely when the window contains only a few bp of the repeat on its rightmost part as it slides the sequence. In more detail, models from Experiment I start identifying the repeat when the first bp of the repeat is placed around position 50-60 of the input window. On the other hand, models from Experiment II, in most cases, identify the start of the repeat when the first bp of the repeat is placed around position 0-10 of the input window, namely after seeing only 0-10 bp of the repeat.

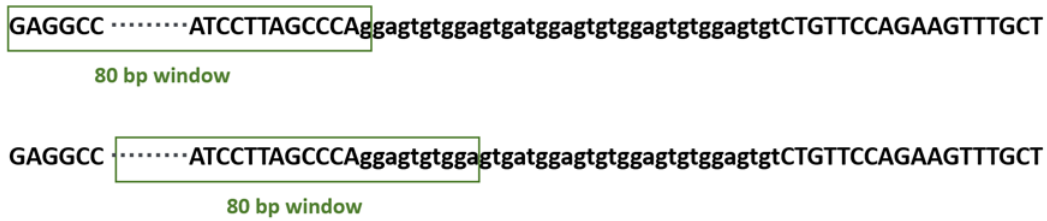


Figure 6.7: *Example of an 80bp window sliding over an input sequence to generate a heatmap. The 80bp window is given to the model as input and the output of the model corresponds to an element of the heatmap. Top: The output of the model corresponds to position 1 (x-axis) of the heatmap. Down: The output of the model corresponds to position 10 (x-axis) of the heatmap.*

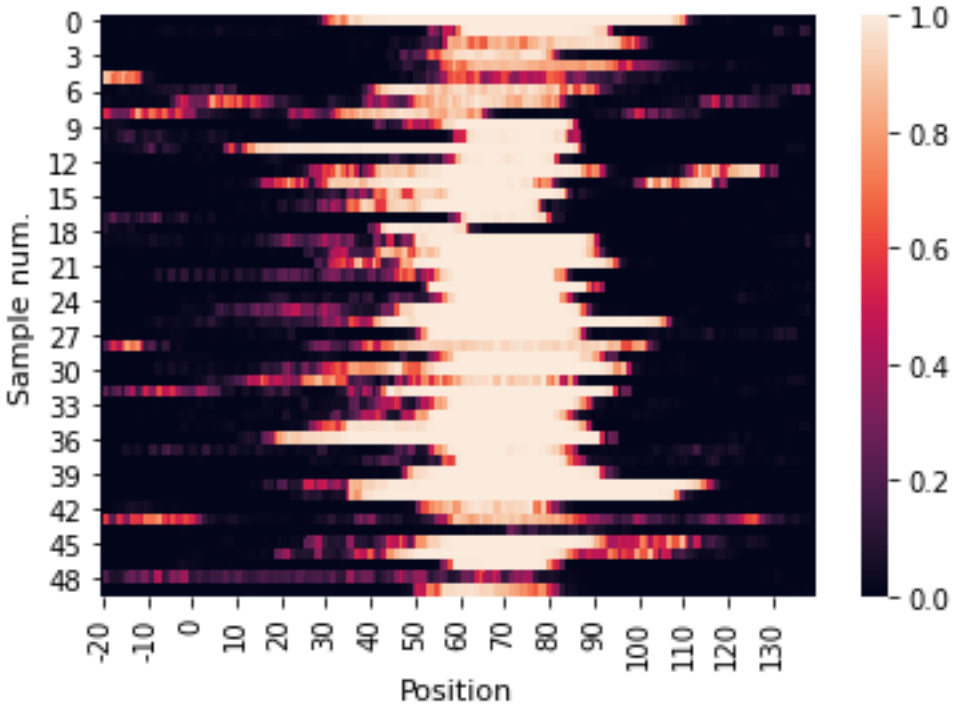


Figure 6.8: *Heatmap for CNN, type I model. Each row corresponds to a sample sequence and each column corresponds to the output of the model when it takes as input an 80bp part of the sequence. That is, an 80 bp window slides over the input sequence and at each position the output of the model is given in the heatmap. x-axis corresponds to the position of the rightmost part of the sliding window. Here, we have denoted as 0 in the x-axis the part where the first bp of the repeat is placed in the 50 samples used to generate the heatmap.*

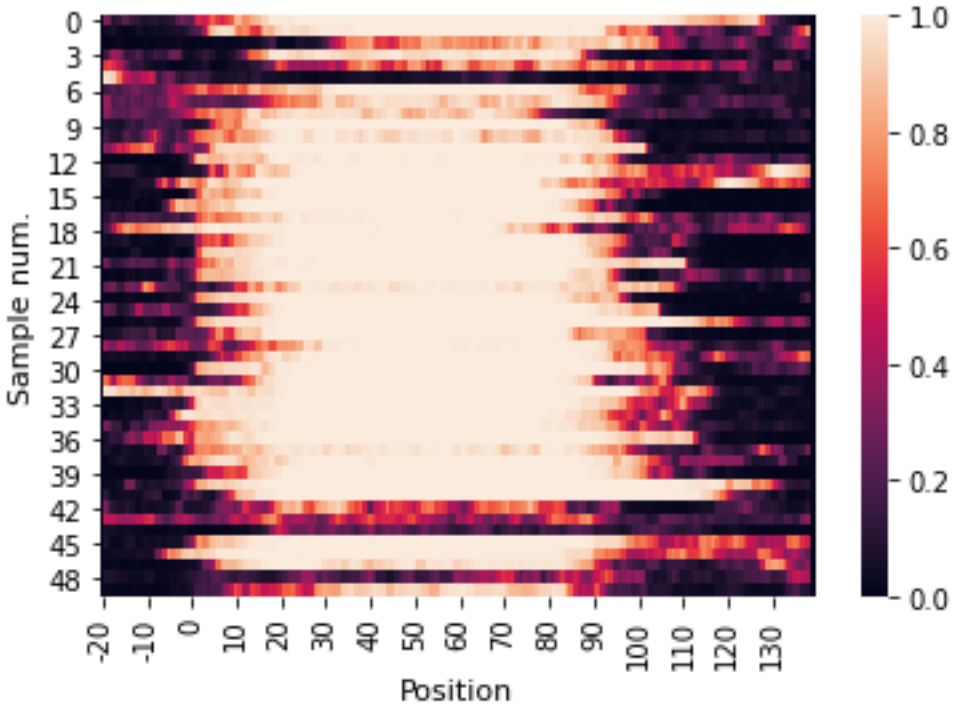


Figure 6.9: *Heatmap for CNN, type II model.*

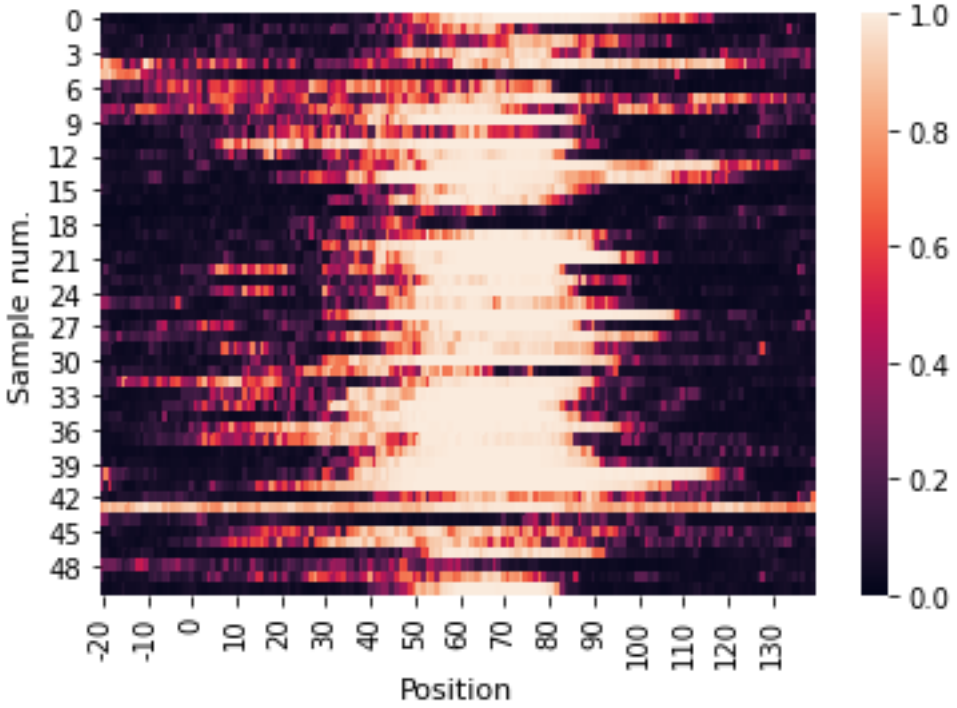


Figure 6.10: *Heatmap for GS VAE, type I model.*

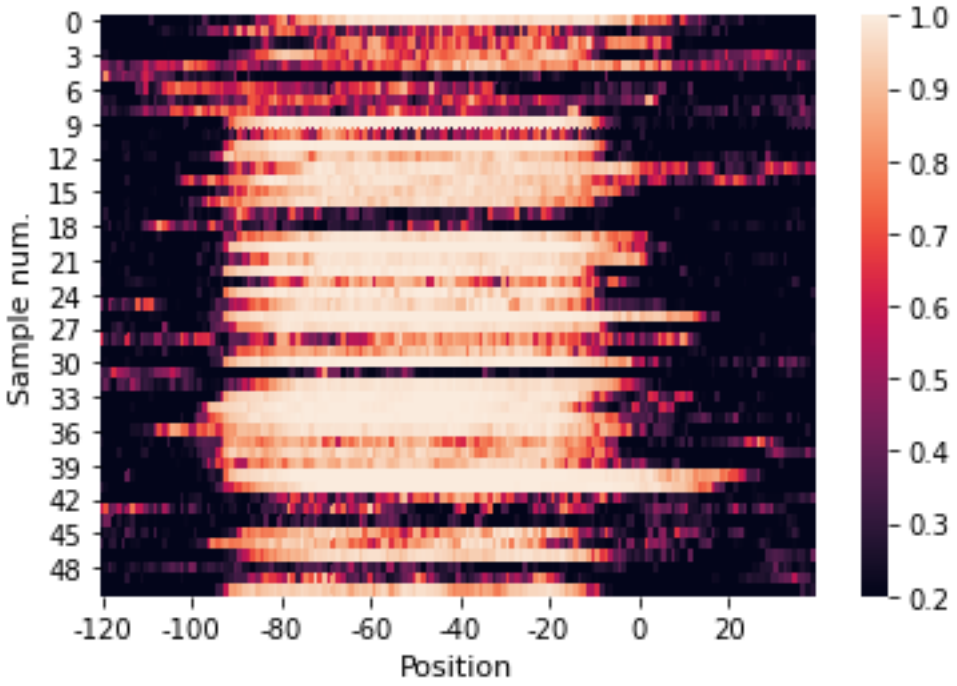


Figure 6.11: *Heatmap for GS VAE, type II model.*

From the above experiments (I&II), it is clear that the models trained on the **Sliding windows or type II Dataset** can predict the start of the repeat much earlier than the model trained on the **Non-overlapping windows or Type I Dataset**. This means that the model of Experiment II performs better in classifying windows where a small part of the repeat is included and, thus, is more appropriate to be used for the annotation of a sequence, since during annotation the model needs to be accurate in terms of the starting position of the repeat.

6.2 Experiment III

Experiment III deals with the objective I.II, which is to annotate the sequence in terms of the presence of repeats. Since the performance of the model trained on the Sliding windows Dataset was proven to be much more appropriate for the exact annotation of the starting point of a repeat, we used this model for the annotating purposes. We omit results for the models trained on Non-overlapping windows Dataset since they are considered inappropriate for the specific task.

The selection of the dataset used for the evaluation of the exact annotation is of high importance. The evaluation of ab initio methods on a dataset which is the

output of another algorithm is complex, since we need to accept the other algorithm's output as the gold standard. However, even in that case, although true positives and false negatives are easily found by comparing with the output of the gold standard algorithm, the case of false positives is more complex. It is difficult to determine if a model's positive output that is not included in the gold standard positive outputs is a false positive or if it is a novel TR (either not previously known or one that the gold standard algorithm is incapable of finding). For that reason, for the evaluation of our model in the task of sequence exact annotation, we used a manually curated dataset created in [6]. In addition, the use of this dataset has the advantage of comparing our models with other algorithms and software used, since the authors in [6] have evaluated the performance of a great selection of other software, along with their own. The dataset consists of 5 different smaller datasets containing biologically relevant, non-naïve repeats. More details on the nature of these datasets were discussed in Section 2.3. The evaluation metric used is Jaccard coefficient, introduced in Section 2.3, which provides a way to measure the similarity between the sequence output of our model and the gold ground truth repeats contained in the dataset.

Most of the repeats contained in the dataset are short in length (<100 bp). This poses a problem to our model. As shown in Section 6.1, type II models are much more accurate than type I models in predicting the start of the repeat. However, type II models need, indeed a smaller, but, still substantial part of the sequence to be seen by the 80bp window, before it is able to predict the starting point of the repeat. The extent to which this initial part is substantial depends on the total repeat length. In other words, since, the evaluation dataset is comprised of small length repeats, the model missing some initial parts of the repeat will greatly affect the evaluation metric, Jaccard coefficient. For example, whereas in the case of a 200 bp repeat, if the model misses the initial 10 bp and the last 5 bp, the Jaccard coefficient would be 0.925, in the case of a 40 bp repeat, the Jaccard coefficient will be $25/40 = 0.625$. Thus, the nature of the dataset requires a precise annotation of the exact starting and ending position.

For that reason, we designed an algorithm containing an extra processing step, which aims in finding the exact starting and ending positions more accurately. In more detail, the DL model is initially used to identify the region that contains the repeat identifying the potential starting and ending positions. Then, we accept the model's prediction with a flexibility range of a few bp (in our case $+-$ 15 bp). To identify the starting/ending positions more accurately within that range, the similarity of subsequent substrings (k-mers) around this position is analyzed. The addition of

this extra step has improved the performance of our model, since, as it was explained above, the precision of a few bp is quite important for annotating TRs of small length. It was observed that this extra step helps increasing the Jaccard coefficient in cases where the initial region is identified correctly (in some range) by the model. In some samples, where the model failed to identify the starting ending position with relative accuracy, the inclusion of the processing step (kmers) did not improve the Jaccard coefficient. Below, we will give a brief explanation of the k-mers-related processing step.

k-mers

In bioinformatics, the term k-mers refers to all substrings of length k of a sequence. For example, the sequence AGTACA, has six monomers (A, G, T, A, C, A), five 2-mers (AG, GT, TA, AC, CA) four 3-mers (AGT, GTA, TAC, ACA), three 4-mers (AGTA, GTAC, TACA), two 5-mers (AGTAC, GTACA) and one 6-mer (AGTACA). The total number of possible k-mers for a DNA sequence is 4^k. To determine if a window contains the start of a repeat we check its similarity to the previous and subsequent window. In more detail, in our extra processing step, we form windows of length w and, for a fixed k, we estimate the frequency vector f of that window, which is comprised of the number of occurrences of each k-mer. Thus, f is given by the following vector:

$$\vec{f}_i^w = (f_0, \ldots, f_j, \ldots, f_{4^k-1}) \tag{6.1}$$

where f_j denotes the number of occurrences of the j^{th} possible k-mer. By comparing the frequency vectors of subsequent windows, we aim to adjust the boundary of the repeat. The metric that is used to measure the similarity between the frequency vectors of two windows, f and h, is taken as the normalized Pearson correlation coefficient between them, given by:

$$\rho(\vec{f}, \vec{h}) = \frac{\sum_{i=1}^{n} \left(\vec{f}_i - \bar{f}\right)\left(\vec{h}_i - \bar{b}\right)}{\sqrt{\sum_{i=1}^{n} \left(\vec{f}_i - \bar{f}\right)^2} \sqrt{\sum_{i=1}^{n} \left(\vec{h}_i - \bar{h}\right)^2}} \tag{6.2}$$

where $\bar{f} = \sum_{i=1}^{n} \vec{f}_i/n$ and $\bar{h} = \sum_{i=1}^{n} \vec{h}_i/n$, for $n = 4^k$. Firstly, the similarity S_1 of two subsequent windows is calculated and then, this measure S_1 is compared with the similarity S_2 of the two subsequent windows that follow. In more detail, let $S_1 = S^w_{(i-w,i)}$ be the similarity measure between two windows W(i-w) and W(i),

then $S_2 = S^w_{(i,i+w)}$ is calculated as the similarity measure between the two windows that follow which are W(i) and W(i+w). In other words, the frequency vector of the window W(i) is compared with the frequency vectors of windows W(i-w) and W(i+w), resulting in similarity measures S_1 and S_2, respectively. The assumption is that if the window W(i) is the one containing the first repeated unit, then the similarity measure S_2, denoting the similarity with the window W(i+w), will be close to 1, since window W(i+w) is also part of the repeat and thus, would be quite similar with window W(i). In addition, since W(i) is the first window containing part of the repeat, the similarity measure S_1 with its previous window, W(i-w), will be lower. As a result, we expect measure:

$$B(i, w) = S^w_{(i,i+w)} - S^w_{(i-w,i)} \tag{6.3}$$

which depends on the triplet of windows windows W(i-w) - W(i) - W(i+w), to be high on the position that the repeat starts. Following the same procedure, with the difference of scanning the sequence from right to left, will denote the window containing the ending position of the repeat. This idea was adapted from [35], where it was used to find the boundaries of long repeats, namely cases where the repeat unit is repeated a hundred or thousand times. The authors used quite wide windows, using the fact that long repeats have hundred or thousands of repeated units. In [35], the authors noticed that using small length windows leads to a noisy B signal in broad regions, making it impossible to detect peaks of the TRs.

In our case, however, the model can already identify the starting and ending position with some precision. Therefore, we only use the additional step with smaller windows to increase the accuracy even further. Thus, the B signal is calculated only for a small region around the previously identified starting/ending position. As a result, B is not noisy since it is calculated only for a small number of triplets of windows.

In Table 6.6, the results for our models and other software and algorithms used for the annotation of TRs, are given. The performance of other software/algorithms reported in Table 6.6 is taken from [6]. Most of the software used for the annotation of repeats output many different results for a given location of interest. For that purpose, in [6], the authors define two extra measures, the average precision and the average recall. Let $T = t_1; .....; t_n$ be a dataset of TRs, and R the set of results of a given algorithm, with $R(t_i)$ being the subset of R overlapping with t_i by at least 1 bp. That is, $R(t_i)$ contains the multiple different results for a given location of interest,

	CNN			Vanilla VAE			GS VAE		
Disease	0.667	35	26	0.03	1	1	0.541	28	17
Codis	0.77	19	15	0.33	5	3	0.747	19	16
Y	0.776,	82	69	0.51	50	35	0.722	78	67
Marshfield	0.747	564	465	0. 343	238	176	0.724,	544	480
Promoters	0.587	11534	5910	0.253	5032	2432	0.545	10884	5780

Table 6.6: *Performance on Experiment III. Model performance on different datasets created in [6]. Compare with Fig. 6.12. The three columns for each model denote the following measures: [average Jaccard, #TR j=0.5, #TR j=0.7] Since our models have a single output per location, the average precision and average recall of Fig. 6.12 are both equal with the average Jaccard coefficient.*

namely the location of t_i. Then the average precision and recall for the given dataset of n TRs, are given by the following mathematical formulas:

$$\sigma_P(T,R) = \frac{1}{n}\left(\sum_{i=1}^{n}\frac{\sum_{x\in R(t_i)} jac\,(x,t_i)}{|R\,(t_i)|}\right)$$

$$\sigma_R(T,R) = \frac{1}{n}\left(\sum_{i=1}^{n}\max_{x\in R(t_i)} jac\,(x,t_i)\right)$$

However, our models give a unique result for each region of interest and, thus, the average precision and average recall are both equal with the average Jaccard coefficient. A high recall in the expense of a small precision can render the algorithm considerably less useful. For example, the extreme case of TRStalker, as shown in Table 6.6, achieves a high recall in all datasets (above 0.9), but produces an extremely high number of results covering a target locus (RPL), varying from 116 to 205 for the different datasets. Covering the same locus with many different outputs creates output redundancy and poses the problem that, since in the real case scenario the ground truth will not be known, there will be no way to select among the multiple outputs. Thus, a high recall is important when it is accompanied by a high precision. We can see that our models perform comparably with other established software and have the advantage of giving only one output for a locatino of interest.

6.3 Experiment IV

Experiment IV deals with the objective II, which is to explore if the neural networks can capture characteristics of the flanking sequence of the repeat, and use them to predict when a repeat is coming, thus associating them with the presence of repeats. This constitutes a first step towards the identification of specific genomic composition

Average precision σ_P, average recall σ_R, average number of reported results covering a target locus (RPL) and total number of TRs intersected (with Jaccard=0.5 and Jaccard=0.7) of the compared algorithms and datasets

Dataset	Measure	Dot2dot-filter	Dot2dot	TRF	MREPS	TRStalker	TRStalker*	SWAN	Troll	SciRoKo	Repeatmasker
Diseases	σ_P	0.830	0.678	0.722	0.630	0.464	0.491	0.372	0.508	0.724	0.452
	σ_R	0.835	0.899	0.763	0.686	0.901	0.960	0.575	0.571	0.752	0.475
	RPL	1.0	6.0	1.4	1.5	155.9	157.5	2.0	4.1	1.1	1.2
	#TR j=0.5	44	45	37	37	43	45	27	29	38	24
	#TR j=0.7	36	42	33	23	43	45	24	18	33	20
CODIS	σ_P	0.860	0.721	0.836	0.659	0.485	0.565	0	0.682	0.819	0.812
	σ_R	0.860	0.899	0.850	0.818	0.839	0.988	0	0.797	0.867	0.822
	RPL	1.0	6.3	1.1	2.1	188.2	172.2	0.0	3.2	1.2	1.0
	#TR j=0.5	19	19	20	20	18	20	0	19	20	19
	#TR j=0.7	18	18	15	15	18	20	0	16	16	15
Y-STR	σ_P	0.877	0.706	0.770	0.617	—	0.535	0.021	0.682	0.827	0.721
	σ_R	0.879	0.916	0.786	0.712	—	0.979	0.029	0.777	0.841	0.729
	RPL	1.0	6.1	1.1	2.1	—	205.5	2.5	2.8	1.1	1.0
	#TR j=0.5	84	86	70	74	—	86	3	79	80	67
	#TR j=0.7	74	81	65	59	—	86	1	75	69	62
Marshfield	σ_P	0.856	0.637	0.784	0.662	—	0.559	0	0.683	0.810	0.767
	σ_R	0.860	0.905	0.794	0.755	—	0.975	0	0.792	0.830	0.775
	RPL	1.0	7.1	1.1	2.0	—	154.3	0.0	3.2	1.1	1.0
	#TR j=0.5	589	609	559	577	—	619	0	583	587	554
	#TR j=0.7	503	557	447	405	—	619	0	490	471	437
Promoters	σ_P	0.825	0.667	0.663	0.575	0.422	0.422	0.663	0.517	0.808	0.743
	σ_R	0.803	0.934	0.548	0.687	0.929	0.929	0.447	0.627	0.709	0.432
	RPL	1.0	4.8	1.4	1.9	116.0	116.0	2.2	2.1	1.0	0.6
	#TR j=0.5	13 783	15 192	8864	12 490	15 264	15 264	5156	10 939	12 125	7128
	#TR j=0.7	12 329	14 792	7905	7880	15 098	15 098	3233	5723	10 492	6286

Figure 6.12: *Table taken from [6] showing the performance of a range of software on the five different curated datasets.*

or other factors as contributing factors to the initiation and, potentially, the origin of tandem repeats. The importance of repeats' flanking sequences and their effect on repeats' mutation rates have been described in Section 3.1.1.1. In this experiment, we explore whether the composition of the repeat's flanking sequence can be distinguished from random sequences by our DL models. This way, we attempt to take a first step towards exploring the flanking sequence composition and characteristics as potential contributing factors to the origin and emergence of repeats. The origin and mutational dynamics of microsatellites is yet not completely clear [27]. Details on that and the importance of their exploration have been discussed in Section 3.1.1.1.

As discussed in Chapter 4, in this experiment we use a dataset where positive samples are comprised of the 60bp flanking sequence before the repeat. Negative samples are random parts of the genome, whose subsequent 60bp do not include a repeat. We train the models in a 2-class classification setting, where the model predicts if a certain input sequence comprises a flanking sequence of a repeat. The models' performance is evaluated in terms of the Accuracy and F1 score. The dataset is balanced, and, thus, the baseling random prediction accuracy is 0.5 for in the 2-class

CNN		vanilla VAE		GS VAE	
Acc.	F1	Acc.	F1	Acc.	F1
0.797	0.79	0.65	0.451	0.759	0.76

Table 6.7: *Performance on Experiment IV. The model predict if an input sequence comprises a flanking sequence of a repeat or not.*

classification setting. In Table 6.7, the performance of the three model architectures explored in this work, CNN, vanilla VAE and Gumbel-Softmax VAE, is presented.

We observe that CNN and Gumbel-Softmax VAE, can predict which sequences are flanking sequences of a repeat with accuracy considerably above the baseline random prediction accuracy of 0.5 and that vanilla VAE performs poorly. As a sanity check, we tested the trained models on sequences that were shuffled, to distort any information. Namely, we permuted the letters in the testing sequences in order to destroy any patterns that the network had capture during training. For both the CNN and the Gumbel-Softmax VAE models the accuracy on the permuted sequences dropped around the baseline random prediction accuracy of 0.5. In Fig. 6.13 the heatmaps for the current experiment are shown. Fig. 6.13 (a) corresponds to the heatmap for the CNN, while Fig. 6.13 (b) to the heatmap for the Gumbel-Softmax VAE architecture.

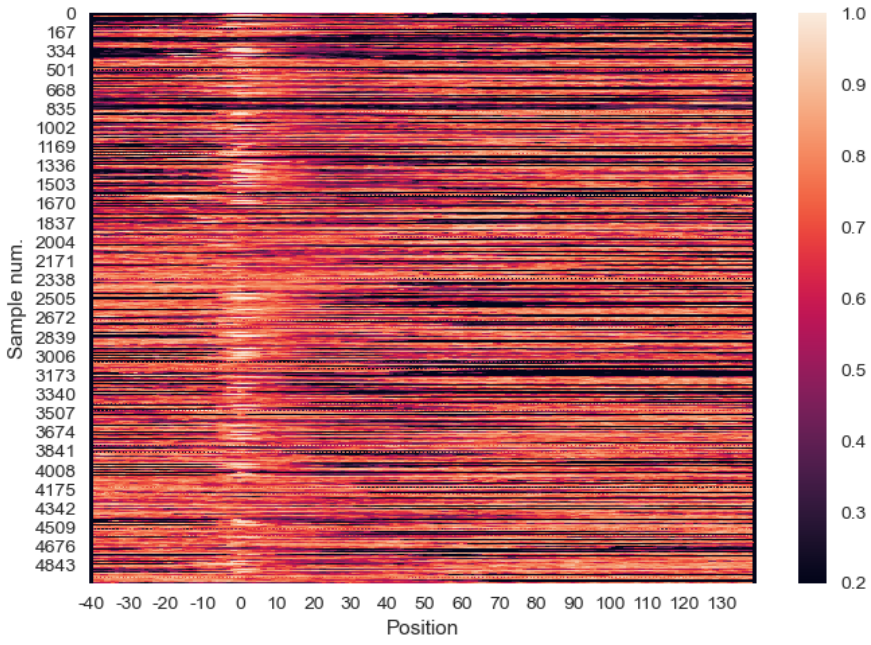
To produce these heatmaps, for each sample we slide a 60 bp window over the sequence with a step of 1 bp. At each position (x-axis) the output of the model among the 5,000 sequences (y-axis) is denoted. The samples have been created to contain 100 bp part of the flanking sequence before the repeat, while they also include the repeat and some part of the flanking sequence after the repeat. The amount of bp after the repeat varies among samples, since the number depends on the length of the repeat. To make the numbering on x-axis clear, we note that position 0 of the heatmaps corresponds to the output of the model that takes as input the 60 bp window that are immediately before the first repeat bp. That is, on position 1 the 60 bp window scanning the sequence contains 59 bp of the flanking sequence and only the first bp of the repeat etc.

In Fig. 6.14 (a) and Fig. 6.15 (a), the average binarized output of the network among the 5,000 positive sequences is shown for the CNN and the Gumbel-Softmax VAE respecitvely. That is, for each sample we binarize the predictions of the model shown in the heatmaps, using the standard 0.5 threshold for the positive prediction and, then, average among the 5,000 samples. We observe that the model output peaks when the scanning window contains the part immediately before the repeat.

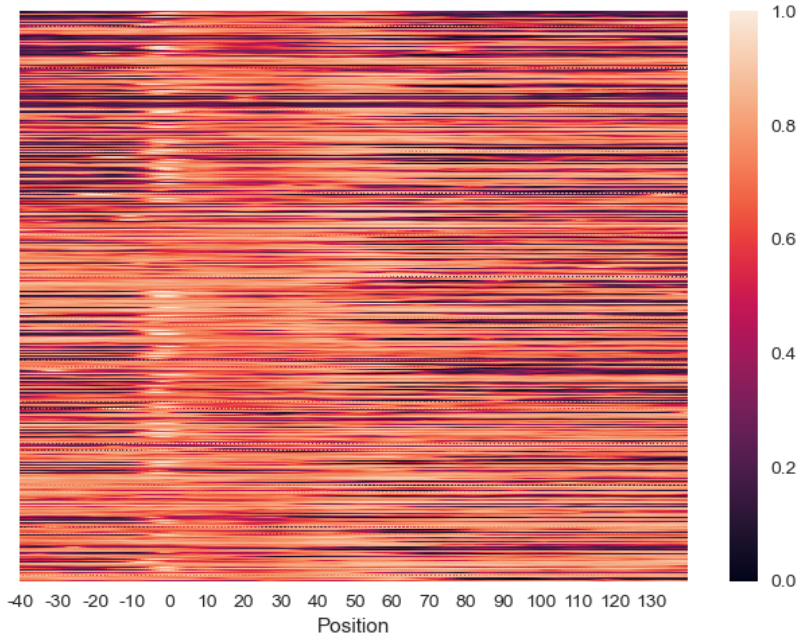
A decrease in the average binarized output occurs when the window contains the repeat. We note that the length of the repeat influences the minimum size of flanking sequence (before or after the repeat) that is contained in the input window at any point. To make this clear for the reader, we explain that, for example, for a repeat of 30 bp length, the 60bp window that scans the repeat and its flanking sequence will contain a minimum of 30 bp of the flanking sequence before or after the repeat at any point. For larger repeat sizes $\geq$ 60 bp there are points where the window does not contain any part of the flanking sequence, leading to a larger decrease in the model output. Thus, in Fig. 6.14 (a) and Fig. 6.15 (a), the decrease in positions where the scanning window contains the repeat is small accounting for the samples that contain small length repeats, since the curve is the average output among 5,000 sequences.

After this small decrease, a rise in the curve denotes that the networks detect the patterns and characteristics used to predict the flanking sequence *before* the repeat, in the flanking sequence *after* the repeat. That is, even though the NNs were trained using data from the flanking sequences *before* the repeat, the characteristics that are captured by the NNs and enable the accurate prediction of a sequence as flanking/non-flanking are also present in the flanking sequence *after* the repeat in many samples. This is more evident in the case of Gumbel-Softmax VAE (Fig. 6.15 (a)). Comparing 6.14 (a) and Fig. 6.15 (a), we observe that the unsupervised Gumbel-Softmax VAE is more able to capture patterns that are present in the flanking sequences both *before* and *after* the repeat despite being trained only on data from the upstream (before) part. This could potentially mean that the unsupervised setting is more appropriate for this experiment.

We also trained the models using the same type of dataset comprising of repeat's flanking sequences (positive class) and random sequences not comprising repeats (negative class) but we created the data leaving a 10 bp gap between the repeat and the windows labelled as positive samples. This is to account for potential mistakes in annotation of the RepeatMasker, e.g. RepeatMasker misannotating the initial part of a repeat. This way, we ensure that there is no misannotated part of the repeat included in the positive samples. The results for the CNN and Gumbel-Softam VAE are shown in Table 6.8. The performance of the models drops a little but the models can still predict which sequences are flanking sequences of a repeat with high accuracy. In Fig. 6.14 (b) and Fig. 6.15 (b), the average binarized output of the network among the 5,000 positive sequences is shown for the CNN and the Gumbel-Softmax VAE respectively.

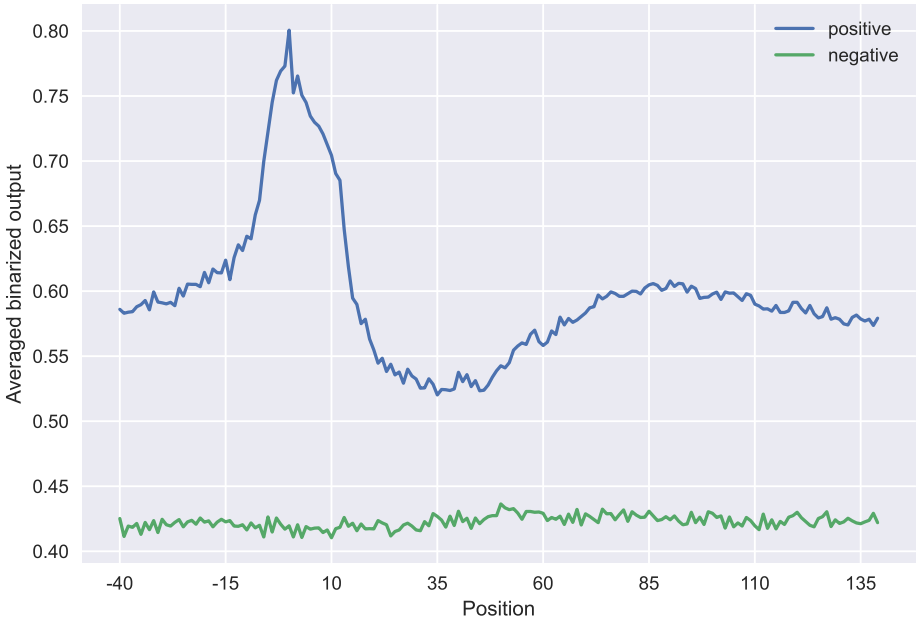


(a)

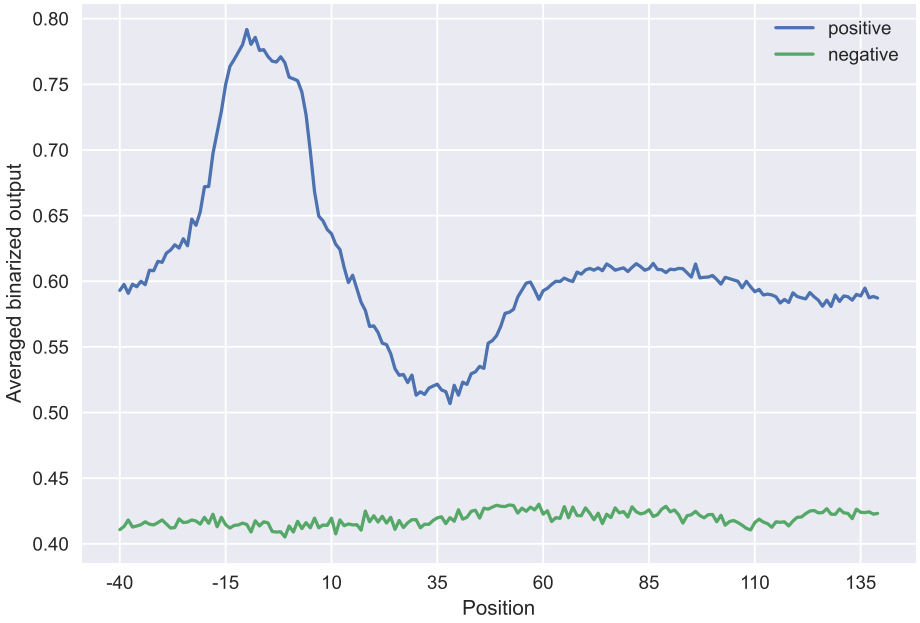


(b)

Figure 6.13: *Heatmaps of 5,000 sequences containing a repeat and its flanking region. (a) CNN (b) Gumbel-Softmax VAE*

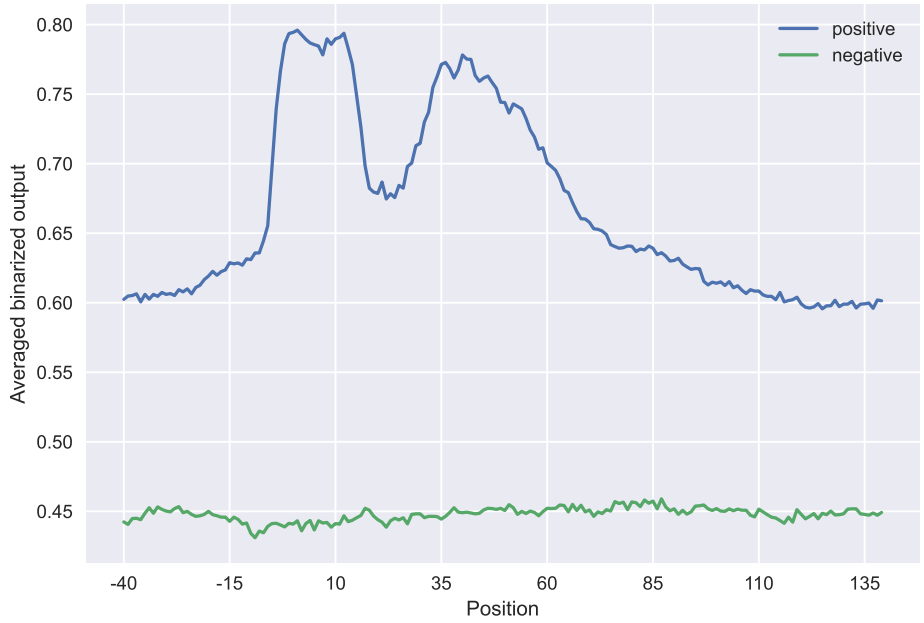


(a)

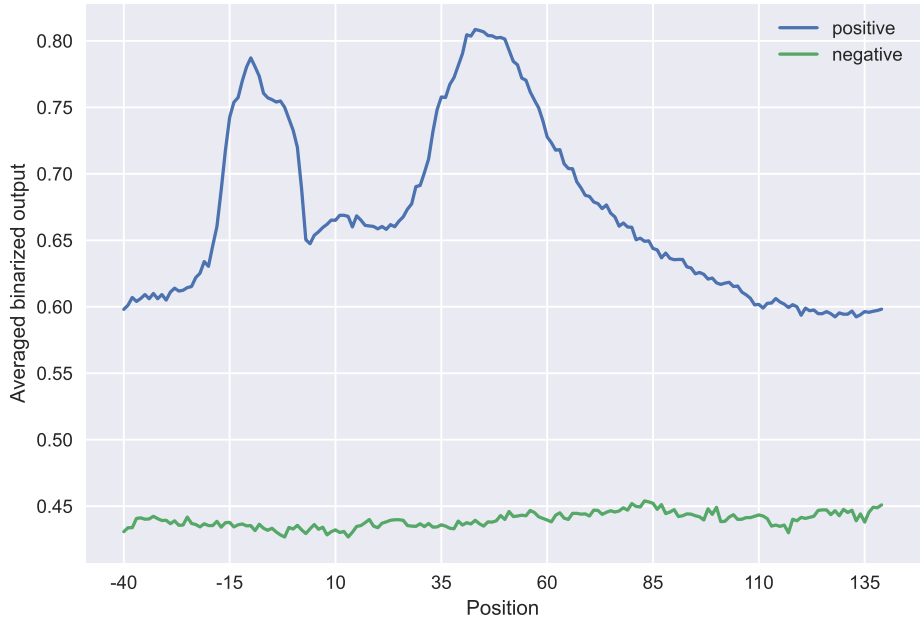


(b)

Figure 6.14: *Blue: Binarized output averaged among 5,000 sequences containing a repeat and its flanking region (positives). Green: Binarized output averaged among 5,000 sequences non containing a repeat (negatives). (a) CNN model trained immediate flanking sequences (b) CNN model trained on flanking sequences with a 10 bp gap*



(a)



(b)

Figure 6.15: Binarized output averaged among 5,000 sequences containing a repeat and its flanking region (positives). Green: Binarized output averaged among 5,000 sequences non containing a repeat (negatives). (a) GS VAE model trained immediate flanking sequences (b) GS VAE model trained on flanking sequences with a 10 bp gap

CNN		GS VAE	
Acc.	F1	Acc.	F1
0.765	0.767	0.74	0.738

Table 6.8: Performance on Experiment IV. The model predict if an input sequence comprises a flanking sequence of a repeat or not. Model trained on flanking sequences that are 10 bp away from the repeats starting point.

Chapter 7

Discussion

The aim of this project was two-fold. The first objective (Objective I) was to explore the use of deep learning models for the identification of tandem repeats. The second objective (Objective II) was to use deep learning models to correlate the genomic composition of repeats' upstream flanking sequences, namely sequences that are placed immediately before and after the repeat, with their presence. Both objectives were fueled by the importance of tandem repeats and latest evidence that reinforce their functional role and great contribution in gene expression, regulation, and phenotypes, as discussed in 1.1 and 3.1. This latest evidence has led to an increasing interest around tandem repeats, a fact that renders their identification and analysis highly important.

Objective I, which aimed at identifying the repeats was split in two subtasks, referred to as Objective I.I and Objective I.II. Objective I.I aimed for the models to predict if a certain 80bp window contains a repeat or not. Objective I.II aimed for the models to perform exact annotation of the input sequence in terms of the presence of repeats. That is, to identify the exact starting and ending positions of the repeat while scanning an input sequence that contained a repeat at a random location. Objective II, which was related to the exploration of a potential correlation of the presence of repeats with the genomic composition of their upstream flanking sequence, aimed for the models to predict if a 60bp window comprises part of the flanking sequence of a repeat or not.

To target the aforementioned objectives, we adapted and implemented state-of-the-art methods in the domain of deep learning for application in genomic DNA sequences. We explored both supervised and unsupervised methods to identify the more appropriate of these settings for our targeted tasks. In particular, we explored the application of the supervised convolutional neural networks (CNNs) and unsupervised variational autoencoders (VAEs). In the framework of VAEs, we explored

both the vanilla VAE, which is comprised of continuous latent variables, while also the Gumbel-Softmax VAE, a fully discrete latent space VAE. The exploration of discrete latent variable emerged as a natural choice taking into account the inherently discrete nature of the genomic data and also that discrete latent variable VAEs have been shown to alleviate issues, e.g. posterior collapse, in the complex procedure of training the VAE.

To train these deep learning models, we tried different optimization settings and architectures and we performed extensive hyper-parameter tuning as discussed in 5. To train and evaluate our models the whole genome sequence DNA data from all chromosomes was obtained from USCS Genome Browser. For the labelling of the repeats in training data, we used RepeatMasker, which is one of the most used software for the identification of repetitive elements. During the evaluation of the models' performance, RepeatMasker's labelling was admitted as the ground truth for the experiments targeting all objectives, with the exception of Objective I.II. For Objective I.II, the evaluation of the task of annotating the repeats, namely finding their starting and ending position, requires high precision. That is, finding the exact starting and ending position of a repeat is more difficult than predicting its presence in an input window. Thus, the evaluation of this task should involve a highly precise evaluation dataset where the annotated repeats are verified. For that purpose, a manually curated dataset created in [6], consisting of biologically non-naïve short tandem repeats was used to evaluate the models performance in the task related to Objective I.II.

7.1 Assessing the results

The results achieved by the models are promising, with the CNN architecture and the Gumbel-Softmax VAE achieving the best performances.

For the **Objective I.I** we, firstly, investigated the use of two different training datasets. Type I dataset consisted of samples where the repeat was placed on their initial part, namely the starting position was randomly placed on positions 0-20 of the samples. Type II dataset contained samples where the initial position of the repeat could be placed in any possible position. Type I dataset guarantees that the training samples contain a substantial part of the repeat pattern, while type II dataset provides no such guarantee. It was observed that models trained on the type II dataset could more accurately predict if a window contains a repeat and their accuracy depended less on the length of the repeat that is contained on the window in comparison with

the models trained on type I dataset. That is, the type I models (for all architectures) were more accurately predicted a repeat even in cases where only a small part of it was contained on the rightmost part of the window, in contrast with type I models, which achieved a low accuracy in such cases. After these comparisons, we concluded that type II dataset is more appropriate in cases where we need the models to be able to predict if any, larger or smaller, part of the repeat is contained in a window.

For both model types, we analyzed how the performance varies for testing samples containing different lengths of a repeat and how the detection of repeats is influenced when we contaminate them with noise. All models were able to identify the repeats even when these were subject of considerable contamination. This is of high importance since, as discussed in 3.1, tandem repeats are hypermutable, meaning that they are usually subject of subsequent mutations, which leads to their impurity. That is, the repeated motif is usually not perfect, a fact that renders the identification of the repeats more complicated. However, the mutated repeats are critical to detect since they are associated with various diseases, as discussed in Section 3.1. Thus, the ability of our models to discern the existence of repeats in sequences, even when these are contaminated with noise (corresponding to artificial mutations) is critical and attaches extra importance to our models' performance.

In the more difficult task of annotating the input sequence in terms of the presence of repeats, constituting **Objective I.II**, our models performed comparably with other established software. The sequence annotation was performed by scanning the input sequence, namely by moving the input window along it with a step equal to 1 bp. At any point, the position corresponding to the rightmost window position was annotated. Thus, since we aim to accurately predict the starting point of the repeat, e.g. when ideally only the first, but realistically only a few bp of the repeat are included in the input window, we used type II models. This choice is motivated by the conclusions of experiments related to Objective I.I mentioned above. These indicate that type II models have been identified as the ones which are appropriate for the detection of repeats in cases when only a small part of them is contained in the window. The curated dataset used for the evaluation of the repeat annotation contains short length tandem repeats, namely repeats where the repeated unit is repeated some tens of times. As a result, extreme precision is needed in the identification of the starting/ending point of the repeats. To increase the accuracy of our models, we added an extra processing step, aiming in adjusting the models' prediction. That is, we analyzed the similarity of subsequent substrings (k-mers) around this position. This extra step is based on the assumption that the window containing the initial

part of the repeat will be much more similar to subsequent windows, also containing parts of the repeat, than to previous windows containing random sequences.

CNN and Gumbel-Softmax VAE are performing comparable with other software, while vanilla VAE performance is quite low in some of the datasets. The CNN has a consistent performance among all five datasets, and achieves a performance comparable with other software even for the Diasease Datasets, which can be identified as the more difficult one based on the software performances. Most of the software cover a certain location with many outputs, namely for every given location various potential repeats are output by the software. For that reason, the authors in [62] introduced two different measures, recall corresponding to the maximum Jaccard coefficient among the multiple outputs for the locus and precision corresponding to the average Jaccard coefficient of the output for the given locus. As our models give one output for each location, both these measures correspond to the average Jaccard coefficient. Thus, the comparison with software that have a high recall and low precision is not exactly fair. A high recall in the expense of a small precision can render the algorithm considerably less useful, since there is no way of selecting one among the potential outputs.

For the **Objective II**, we tested the ability of the models to predict the emergence of a repeat only by having as input their flanking sequence. CNN achieved an accuracy and F1-score of over 79%, Gumbel-Softmax VAE an accuracy and F1-score around 76%, while vanilla VAE performed poorly achieving an accuracy of 65% but an F1-score of 45%. The ability of models to predict when an input sequence comprises a flanking sequence of a repeat indicates that the networks are able to capture certain genomic composition and underlying characteristics of the flanking sequence of repeats that correlate with their presence.

The influence that flanking sequences have on repeats have been reported in literature, e.g. on the repeats mutation rates [61], [27]. The relation between the flanking sequence and the repeat is a complex one that is not yet completely clear and as suggested in the literature should be further explored. [27]. Also, the origin of tandem repeats is a complex issue and many theories have been suggested for their genesis. The investigation of the tandem repeats origin could lead to a better understanding of the function and organization of the genome, the evolution of species and the population genetic studies [26]. The fact that our models are able to predict with high accuracy which sequences constitute flanking sequences of the repeats indicates a positive correlation between certain genomic compositions or underlying characteristics of the flanking sequences and their presence. That could potentially comprise

a first step towards exploring the flanking sequence composition and characteristics as potential contributor to the origin and emergence of repeats.

7.2 Limitations and Future work

In the current work, we have for the first time explored the use of deep learning for the analysis of tandem repeats. Our results are promising, indicating that further work would be valuable in that direction. However, our current work has limitations that should be addressed in future work.

Our models have performed well in terms of predicting fixed-length windows containing the repeats. This is a really useful task that can be used to locate the repeats with relative accuracy and also indicates the capability of deep learning models to analyse tandem repeats. However, the ability to find the exact starting and ending position of the repeats is a property that is needed to render our models useful tools for the annotation of the genome. The task of finding tandem repeats is complex, and this is evident by the fact that software that have been introduced years ago and are frequently used in multiple applications for the annotation of the sequence, like RepeatMasker, are imperfect and in cases inaccurate. As seen in Experiment III, our models have performed comparably with some of the other software, however, some of them have surpassed our models' performance on all datasets used for the evaluation. This is on some extent related to the nature of the curated datasets used for the evaluation of the exact annotation task (Experiment III). As we described in Section 6.2, these datasets contain mostly small length repeats and, thus, the precision of the models on finding the exact starting and ending position greatly influences the evaluation metric. Since our models work on a fixed-length window basis, i.e. they are trained to predict if a window contains a repeat, finding the exact starting and ending position is not their strength. The fixed-length window characteristic of our models, also makes them incapable of distinguishing repeats that are close together and are separated only by a few base pairs. In relation to the above problems, we propose the following two different routes.

1. The first one is to focus on using our models for the prediction of longer short tandem repeats, e.g. repeats in which the small motif is repeated, for example, hundreds of times. That is because in such cases finding the exact starting and ending position of the repeat is not of great importance. Thus, in an experiment similar to Experiment III, but with a dataset that would contain longer repeats, our models could potentially perform better than the other software.

A justified comment at this point would be that the other software could also potentially perform better in finding larger length repeats. However, most of the other's software problem in finding the repeats is not related to their imprecision in finding the starting and ending position, since they don't work on a window-basis as our models. The variability of the repeats due to mutations is what, in most cases, renders their detection a challenge. In the case of our models, we have tested their ability to detect mutated repeats by artificially introducing noise to the sequence and, we observed that our models are able to detect a substantial part of the repeats despite the fact that they were subject of extensive mutations (Fig. 6.4b, Fig. 6.5b). Thus, we believe that our models have the potential to overpass the performance of other software in detecting mutated repeats. For the software that work on a candidate search base, the difficulty to detect mutated repeats has been noted in the literature. However, for other de novo algorithms, this ability would depend on the exact algorithm they use to find the repeats. Thus, although the ability of our models to detect mutated repeats has been explored in this work, their comparative performance in this task against other software should be tested.

2. The second route involves the combination of our models with another extra processing step, which would allow our overall algorithm comprised by the deep learning model and the extra processing step to be more precise. A first step toward this direction has already been implemented by incorporating the k-mers-related processing step in Experiment III. This is a relatively simple processing step. Further work should be done on this direction, in terms of exploring different processing steps. As potential alternatives for the processing steps, the algorithms proposed by other software, e.g. the algorithm of [6], could be explored. In other words, we could explore the potential combination of deep learning models aiming at identifying the broader region containing the repeat with the other algorithms that have been proposed in the literature. That could potentially lead to an improvement in performance by exploiting each model's advantages. Connecting that again with the the fact that the NNs showcased a flexibility in detecting mutated repeats, the combination of NNs with an algorithm that is precise at finding the starting and ending position of repeats but is not good at detecting mutated repeats could potentially increased their respective performances.

Apart from combining our models with an extra processing algorithmic step, we could also attempt to increase their precision in detecting the exact starting and ending points by training them in a different problem setting more appropriate to the task. For example, we could train the models to predict the exact point where a repeat starts in an input window. This comprises a different training scheme than the 2-class classification training scheme we attempted in this work.

Apart from the issue of the exact annotation of the sequence, further work should be done towards exploring the potential improvement of the performance of our models. Towards that we can:

1. Investigate different ways of creating the dataset. Given the fact that there is no previous work on deep learning models used to predict tandem repeats, the most appropriate way to create the training dataset from the available raw human genome sequence is unexplored. We have taken an initial step on exploring that in the current work by designing two different ways to create the training dataset from the raw genome data obtained from the USCS Genome Browser. According to the experiments designed and performed in this work, we have concluded that the nature of the training dataset greatly affects the performance of the models. The reasoning behind that is that we observed considerably different performance between the models trained on the two different dataset we created. Thus, we believe that the further exploration of this aspect of that work is of high importance. During this exploration, we could potentially attempt to combine different types of datasets, e.g. type I and type II datasets used in this work. Also, apart from the way of creating the training dataset, we should experiment with different input lengths.

2. Explore different architectures and combinations of them. Deep learning offers a range of different architectures with some more appropriate than others for specific tasks. In this work we have covered both supervised and unsupervised methods. A further exploration of architectures could potentially increase the model performance. Train our models using labelled data from other algorithms. Experiment III has shown that even though RepeatMasker is a widely used software, other more recently proposed algorithms outperform it. Thus, using labelled data from these algorithms to train our models could potentially increase their performance.

In relation to the VAE model, we can further explore the latent space encodings. Since the model is able to create meaningful representation of the data that enable their classification in the latent space with high accuracy, it could potentially have also incorporated other useful information in the latent space. For example, we could explore if samples with repeats of different lengths are encoded in a different way or if the position where the repeat starts in the input sample influences its the latent space encoding. In addition, different VAEs could be explored., e.g. beta-VAE [86], which has been proven to improve the disentanglement of different classes in the latent space. Another idea would be to train the VAE in a semi-supervised setting to attempt to exploit the existence of labelled data [87]. That could potentially help in creating a more structured latent space.

In relation to the results from Experiment IV, our results are promising. The high accuracy with which our models predict which sequences are flanking sequences of the repeats indicates a positive correlation between certain genomic compositions or underlying characteristics of the flanking sequences and the repeats' presence. However, this does not by itself proves that elements in the flanking sequence act as contributors to the genesis of repeats. Since the genesis/origin of the repeats is a matter that concentrates a lot of research interest, further experiments attempting to correlate of the flanking sequence composition with repeat's genesis should be designed.

We hope to investigate these ideas systematically in future work.

Bibliography

[1] Machine learning course notes, university of oxford, michaelmas term 2020. https://www.cs.ox.ac.uk/teaching/materials20-21/ml/lectures/lecture14.pdf. Accessed: 2021-08-25.

[2] Gökcen Eraslan, Žiga Avsec, Julien Gagneur, and Fabian J Theis. Deep learning: new computational modelling techniques for genomics. *Nature Reviews Genetics*, 20(7):389–403, 2019.

[3] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[4] Diederik P Kingma and Max Welling. An introduction to variational autoencoders. *arXiv preprint arXiv:1906.02691*, 2019.

[5] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

[6] Loredana M Genovese, Marco M Mosca, Marco Pellegrini, and Filippo Geraci. Dot2dot: accurate whole-genome tandem repeats discovery. *Bioinformatics*, 35(6):914–922, 2019.

[7] Interspersed repeats. https://en.wikipedia.org/wiki/Interspersed$_r epeat. Accessed$: $2021 - 08 - 10$.

[8] Hao Fan and Jia-You Chu. A brief review of short tandem repeat mutation. *Genomics, proteomics & bioinformatics*, 5(1):7–14, 2007.

[9] Maximilian O Press, Keisha D Carlson, and Christine Queitsch. The overdue promise of short tandem repeat variation for heritability. *Trends in Genetics*, 30(11):504–512, 2014.

[10] Russell L Margolis, Melvin G McInnis, Adam Rosenblatt, and Christopher A Ross. Trinucleotide repeat expansion and neuropsychiatric disease. *Archives of general psychiatry*, 56(11):1019–1031, 1999.

[11] Henry Paulson. Repeat expansion diseases. *Handbook of clinical neurology*, 147:105–123, 2018.

[12] Anthony J Hannan. Tandem repeats mediating genetic plasticity in health and disease. *Nature Reviews Genetics*, 19(5):286, 2018.

[13] Monogenic Diseases. https://www.nature.com/scitable/topicpage/rare-genetic-disorders-learning-about-genetic-disease-979/. Accessed: 2021-08-10.

[14] David Jakubosky, Matteo D'Antonio, Marc Jan Bonder, Craig Smail, Margaret KR Donovan, William W Young Greenwald, Hiroko Matsui, Agnieszka D'Antonio-Chronowska, Oliver Stegle, Erin N Smith, et al. Properties of structural variants and short tandem repeats associated with gene expression and complex traits. *Nature communications*, 11(1):1–15, 2020.

[15] Melissa Gymrek, Thomas Willems, Audrey Guilmatre, Haoyang Zeng, Barak Markus, Stoyan Georgiev, Mark J Daly, Alkes L Price, Jonathan K Pritchard, Andrew J Sharp, et al. Abundant contribution of short tandem repeats to gene expression variation in humans. *Nature genetics*, 48(1):22–29, 2016.

[16] Javier Quilez, Audrey Guilmatre, Paras Garg, Gareth Highnam, Melissa Gymrek, Yaniv Erlich, Ricky S Joshi, David Mittelman, and Andrew J Sharp. Polymorphic tandem repeats within gene promoters act as modifiers of gene expression and dna methylation in humans. *Nucleic acids research*, 44(8):3750–3762, 2016.

[17] Matthew T Maurano, Richard Humbert, Eric Rynes, Robert E Thurman, Eric Haugen, Hao Wang, Alex P Reynolds, Richard Sandstrom, Hongzhu Qu, Jennifer Brody, et al. Systematic localization of common disease-associated variation in regulatory dna. *Science*, 337(6099):1190–1195, 2012.

[18] Brett Trost, Worrawat Engchuan, Charlotte M Nguyen, Bhooma Thiruvahindrapuram, Egor Dolzhenko, Ian Backstrom, Mila Mirceta, Bahareh A Mojarad, Yue Yin, Alona Dov, et al. Genome-wide detection of tandem dna repeats that are expanded in autism. *Nature*, 586(7827):80–86, 2020.

[19] J Craig et al. Complex diseases: Research and applications. *Nature Education*, 1(1):184, 2008.

[20] Teri A Manolio, Francis S Collins, Nancy J Cox, David B Goldstein, Lucia A Hindorff, David J Hunter, Mark I McCarthy, Erin M Ramos, Lon R Cardon, Aravinda Chakravarti, et al. Finding the missing heritability of complex diseases. *Nature*, 461(7265):747–753, 2009.

[21] Jian Zhou and Olga G Troyanskaya. Predicting effects of noncoding variants with deep learning–based sequence model. *Nature methods*, 12(10):931–934, 2015.

[22] David R Kelley, Yakir A Reshef, Maxwell Bileschi, David Belanger, Cory Y McLean, and Jasper Snoek. Sequential regulatory activity prediction across chromosomes with convolutional neural networks. *Genome research*, 28(5):739–750, 2018.

[23] Jian Zhou, Chandra L Theesfeld, Kevin Yao, Kathleen M Chen, Aaron K Wong, and Olga G Troyanskaya. Deep learning sequence-based ab initio prediction of variant effects on expression and disease risk. *Nature genetics*, 50(8):1171–1179, 2018.

[24] Ziga Avsec, Vikram Agarwal, Daniel Visentin, Joseph R Ledsam, Agnieszka Grabska-Barwinska, Kyle R Taylor, Yannis Assael, John Jumper, Pushmeet Kohli, and David R Kelley. Effective gene expression prediction from sequence by integrating long-range interactions. *bioRxiv*, 2021.

[25] Mathys Grapotte, Manu Saraswat, Chloé Bessière, Christophe Menichelli, Jordan A. Ramilowski, Jessica Severin, Yoshihide Hayashizaki, Masayoshi Itoh, Michihira Tagami, Mitsuyoshi Murata, Miki Kojima-Ishiyama, Shohei Noma, Shuhei Noguchi, Takeya Kasukawa, Akira Hasegawa, Harukazu Suzuki, Hiromi Nishiyori-Sueki, Martin C. Frith, FANTOM consortium, Clément Chatelain, Piero Carninci, Michiel J.L. de Hoon, Wyeth W. Wasserman, Laurent Bréhélin, Charles-Henri Lecellier, Michael Detmar, Lothar C. Dieterich, Filip Roudnicky, and et al. Discovery of widespread transcription initiation at microsatellites predictable by sequence-based deep neural network. *Nature Communications*, 12(1):3297, 2021.

[26] Deepti Srivastava, Malik Mobeen Ahmad, Md Shamim, Rashmi Maurya, Neha Srivastava, Pramila Pandey, Saba Siddiqui, and Mohd Haris Siddiqui. Modulation of gene expression by microsatellites in microbes. In *New and Future*

Developments in Microbial Biotechnology and Bioengineering, pages 209–218. Elsevier, 2019.

[27] Atul Bhargava and FF Fuentes. Mutational dynamics of microsatellites. *Molecular biotechnology*, 44(3):250–266, 2010.

[28] Gary Benson. Tandem repeats finder: a program to analyze dna sequences. *Nucleic acids research*, 27(2):573–580, 1999.

[29] Marco Pellegrini, M Elena Renda, and Alessio Vecchio. Trstalker: an efficient heuristic for finding fuzzy tandem repeats. *Bioinformatics*, 26(12):i358–i366, 2010.

[30] Roman Kolpakov, Ghizlane Bana, and Gregory Kucherov. mreps: efficient and flexible detection of tandem repeats in dna. *Nucleic acids research*, 31(13):3672–3678, 2003.

[31] Adalberto T Castelo, Wellington Martins, and Guang R Gao. Troll—tandem repeat occurrence locator. *Bioinformatics*, 18(4):634–636, 2002.

[32] Alfred V. Aho and Margaret J. Corasick. Efficient string matching: An aid to bibliographic search. *Commun. ACM*, 18(6):333–340, June 1975.

[33] Martin C Frith. A new repeat-masking method enables specific detection of homologous sequences. *Nucleic acids research*, 39(4):e23–e23, 2011.

[34] Alfredo Velasco, Benjamin T James, Vincent D Wells, and Hani Z Girgis. Look4trs: a de novo tool for detecting simple tandem repeats using self-supervised hidden markov models. *Bioinformatics*, 36(2):380–387, 2020.

[35] Shinichi Morishita, Kazuki Ichikawa, and Eugene W Myers. Finding long tandem repeats in long noisy reads. *Bioinformatics*, 37(5):612–621, 2021.

[36] Nucleobase. https://en.wikipedia.org/wiki/Nucleobase. Accessed: 2021-08-10.

[37] P Myers and M Sebaihia. Tandem repeats and morphological variation. *Nature Education*, 1(1):1, 2007.

[38] Ole K Tørresen, Bastiaan Star, Pablo Mier, Miguel A Andrade-Navarro, Alex Bateman, Patryk Jarnot, Aleksandra Gruca, Marcin Grynberg, Andrey V Kajava, Vasilis J Promponas, et al. Tandem repeats lead to sequence assembly

errors and impose multi-level challenges for genome and protein databases. *Nucleic acids research*, 47(21):10994–11006, 2019.

[39] Rita Gemayel, Marcelo D Vinces, Matthieu Legendre, and Kevin J Verstrepen. Variable tandem repeats accelerate evolution of coding and regulatory sequences. *Annual review of genetics*, 44:445–477, 2010.

[40] Melissa Gymrek. A genomic view of short tandem repeats. *Current opinion in genetics & development*, 44:9–16, 2017.

[41] Robert I Richards and Grant R Sutherland. Dynamic mutations: a new class of mutations causing human disease. *Cell*, 70(5):709–712, 1992.

[42] Recombination. https://www.nature.com/scitable/definition/recombination-226/. Accessed: 2021-08-10.

[43] Guy-Franck Richard and Frédéric Pâques. Mini-and microsatellite expansions: the recombination connection. *EMBO reports*, 1(2):122–126, 2000.

[44] Unnur A Valdimarsdóttir, Donghao Lu, Sigrún H Lund, Katja Fall, Fang Fang, órur Kristjánsson, Daníel Gubjartsson, Agnar Helgason, and Kári Stefánsson. The mother's risk of premature death after child loss across two centuries. *Elife*, 8:e43476, 2019.

[45] Marcelo D Vinces, Matthieu Legendre, Marina Caldara, Masaki Hagihara, and Kevin J Verstrepen. Unstable tandem repeats in promoters confer transcriptional evolvability. *Science*, 324(5931):1213–1216, 2009.

[46] Leslie E Orgel and Francis HC Crick. Selfish dna: the ultimate parasite. *Nature*, 284(5757):604–607, 1980.

[47] W Ford Doolittle and Carmen Sapienza. Selfish genes, the phenotype paradigm and genome evolution. *Nature*, 284(5757):601–603, 1980.

[48] Patricia Martin, Katherine Makepeace, Stuart A Hill, Derek W Hood, and E Richard Moxon. Microsatellite instability regulates transcription factor binding and gene expression. *Proceedings of the National Academy of Sciences*, 102(10):3800–3804, 2005.

[49] Stefan Rothenburg, Friedrich Koch-Nolte, Alexander Rich, and Friedrich Haag. A polymorphic dinucleotide repeat in the rat nucleolin gene forms z-dna and inhibits promoter activity. *Proceedings of the National Academy of Sciences*, 98(16):8985–8990, 2001.

[50] Albert de la Chapelle and Heather Hampel. Clinical relevance of microsatellite instability in colorectal cancer. *Journal of Clinical Oncology*, 28(20):3380, 2010.

[51] Tae-Min Kim, Peter W Laird, and Peter J Park. The landscape of microsatellite instability in colorectal and endometrial cancer genomes. *Cell*, 155(4):858–868, 2013.

[52] Ronald J Hause, Colin C Pritchard, Jay Shendure, and Stephen J Salipante. Classification and characterization of microsatellite instability across 18 cancer types. *Nature medicine*, 22(11):1342–1350, 2016.

[53] S Aslani, A Hossein-Nezhad, K Mirzaei, Z Maghbooli, SN Asgarabad, and F Karimi. Tandem repeats of the catt element of macrophage migration inhibitory factor gene may predict gestational diabetes mellitus severity. *European Journal of Inflammation*, 9(2):185–191, 2011.

[54] Owen Rose and Daniel Falush. A threshold size for microsatellite expansion. *Molecular biology and evolution*, 15(5):613–615, 1998.

[55] Walter Messier, Shou-Hsien Li, and Caro-Beth Stewart. The birth of microsatellites. *Nature*, 381(6582):483–483, 1996.

[56] Patrick C Gallagher, Teri L Lear, Linda D Coogle, and Ernest Bailey. Two sine families associated with equine microsatellite loci. *Mammalian Genome*, 10(2):140–144, 1999.

[57] LJ Alexander, GA Rohrer, RT Stone, and CW Beattie. Porcine sine-associated microsatellite markers: evidence for new artiodactyl sines. *Mammalian Genome*, 6(7):464–468, 1995.

[58] Eyal Nadir, Hanah Margalit, Tamar Gallily, and Shmuel A Ben-Sasson. Microsatellite spreading in the human genome: evolutionary mechanisms and structural implications. *Proceedings of the National Academy of Sciences*, 93(13):6470–6475, 1996.

[59] Xiaoying Lin, Samir Kaul, Steve Rounsley, Terrance P Shea, Maria-Ines Benito, Christopher D Town, Claire Y Fujii, Tanya Mason, Cheryl L Bowman, Mary Barnstead, et al. Sequence and analysis of chromosome 2 of the plant arabidopsis thaliana. *Nature*, 402(6763):761–768, 1999.

[60] Christian Schlötterer. Evolutionary dynamics of microsatellite dna. *Chromosoma*, 109(6):365–371, 2000.

[61] Emmanuel Buschiazzo and Neil J Gemmell. The rise, fall and renaissance of microsatellites in eukaryotic genomes. *Bioessays*, 28(10):1040–1050, 2006.

[62] Travis C Glenn, Wolfgang Stephan, Herbert C Dessauer, and Michael J Braun. Allelic diversity in alligator microsatellite loci is negatively correlated with gc content of flanking sequences and evolutionary conservation of pcr amplifiability. *Molecular Biology and Evolution*, 1996.

[63] Doris Bachtrog, Martin Agis, Marianne Imhof, and Christian Schlötterer. Microsatellite variability differs between dinucleotide repeat motifs—evidence from drosophila melanogaster. *Molecular Biology and Evolution*, 17(9):1277–1285, 2000.

[64] François Balloux, Eric Ecoffey, Luca Fumagalli, Jérôme Goudet, Andreas Wyttenbach, and Jacques Hausser. Microsatellite conservation, polymorphism, and gc content in shrews of the genus sorex (insectivora, mammalia). *Molecular Biology and Evolution*, 15(4):473–475, 1998.

[65] Karpathy, a. (2021). c2231n convolutional neural networks for visual recognition. https://cs231n.github.io/convolutional-networks/. Accessed: 2021-08-25.

[66] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[67] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *International conference on machine learning*, pages 1278–1286. PMLR, 2014.

[68] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

[69] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[70] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32:8026–8037, 2019.

[71] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[72] Yaxiong Wu, Craig Macdonald, and Iadh Ounis. A hybrid conditional variational autoencoder model for personalised top-n recommendation. In *Proceedings of the 2020 ACM SIGIR on International Conference on Theory of Information Retrieval*, pages 89–96, 2020.

[73] Sean M Colby, Jamie R Nuñez, Nathan O Hodas, Courtney D Corley, and Ryan R Renslow. Deep learning to generate in silico chemical property libraries and candidate molecules for small molecule identification in complex samples. *Analytical chemistry*, 92(2):1720–1729, 2019.

[74] Samuel R Bowman, Luke Vilnis, Oriol Vinyals, Andrew M Dai, Rafal Jozefowicz, and Samy Bengio. Generating sentences from a continuous space. *arXiv preprint arXiv:1511.06349*, 2015.

[75] Artidoro Pagnoni, Kevin Liu, and Shangyan Li. Conditional variational autoencoder for neural machine translation. *arXiv preprint arXiv:1812.04405*, 2018.

[76] James Lucas, George Tucker, Roger Grosse, and Mohammad Norouzi. Understanding posterior collapse in generative latent variable models. 2019.

[77] Yuqi Gu and David B Dunson. Identifying interpretable discrete latent structures from discrete data. *arXiv preprint arXiv:2101.10373*, 2021.

[78] Qiao Liu, Shengquan Chen, Rui Jiang, and Wing Hung Wong. Simultaneous deep generative modelling and clustering of single-cell genomic data. *Nature Machine Intelligence*, 3(6):536–544, 2021.

[79] Francesco Bartolucci, Francesca Chiaromonte, Prabhani Kuruppumullage Don, and Bruce G Lindsay. Composite likelihood inference in a discrete latent variable model for two-way "clustering-by-segmentation" problems. *Journal of Computational and Graphical Statistics*, 26(2):388–402, 2017.

[80] Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning. *arXiv preprint arXiv:1711.00937*, 2017.

[81] Arash Vahdat, William Macready, Zhengbing Bian, Amir Khoshaman, and Evgeny Andriyash. Dvae++: Discrete variational autoencoders with overlapping transformations. In *International Conference on Machine Learning*, pages 5035–5044. PMLR, 2018.

[82] Jason Tyler Rolfe. Discrete variational autoencoders. *arXiv preprint arXiv:1609.02200*, 2016.

[83] Yang Zhao, Ping Yu, Suchismit Mahapatra, Qinliang Su, and Changyou Chen. Improve variational autoencoder for text generationwith discrete latent bottleneck. *arXiv preprint arXiv:2004.10603*, 2020.

[84] Tiancheng Zhao, Kyusong Lee, and Maxine Eskenazi. Unsupervised discrete sentence representation learning for interpretable neural dialog generation. *arXiv preprint arXiv:1804.08069*, 2018.

[85] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

[86] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. 2016.

[87] Diederik P Kingma, Shakir Mohamed, Danilo Jimenez Rezende, and Max Welling. Semi-supervised learning with deep generative models. In *Advances in neural information processing systems*, pages 3581–3589, 2014.