

Computer vision and machine learning for R-loop detection
and classification in AFM imaging data

by

Berkant Cunnuk

A thesis submitted to the Faculty of Graduate and Postdoctoral Studies of
The University of Manitoba
in partial fulfillment of the requirements of the degree of

Master of Science

Department of Mathematics
University of Manitoba
Winnipeg

Copyright © 2026 by Berkant Cunnuk

Abstract

R-loops are three-stranded nucleic acid structures containing a DNA:RNA hybrid and an associated single DNA strand. They are usually created during the process of transcription. Although their presence can be beneficial in cellular processes, excessive formation of these structures is commonly associated with instabilities. However, their underlying contribution to genomic instability is not well-understood, and their architecture is not well-known either. As such, it is important to identify and classify different R-loop architectures. This task has been previously accomplished through manual classification. In this study, we introduce two computational pipelines to automate this task in atomic force microscopy images: a computer vision pipeline and a machine learning pipeline. The former uses a deterministic algorithm involving image segmentation and pattern-based shape recognition. The latter includes two different algorithms: a decision tree and a multi-layer perceptron neural network. We present a comparison of these two approaches and share results for multi-class classification and binary classification of R-loop imaging data.

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Dr. Margherita Maria Ferrari, for her guidance throughout the ups and downs of this project. I would also like to thank the Pyne Lab and the Chédin Lab for providing the data used in this project and for helpful discussions. I gratefully acknowledge that this research was supported in part by the Pacific Institute for the Mathematical Sciences through a Maud Menten Institute Research Accelerator Award. I acknowledge the support of the University of Manitoba Grex High Performance Computing Centre (RRID:SCR_026342) for computational resources. Lastly, I would like to thank the UM Graduate Mathematics Society for creating a supportive environment throughout this journey.

Contents

Abstract	i
Acknowledgements	i
List of Figures	x
List of Tables	xi
1 Introduction	1
2 Literature review	3
2.1 R-loops and their formation	3
2.2 The role of R-loops in genomic instability	6
2.3 Possible benefits of R-loops	9
2.4 Methods to analyze R-loops and their formation	12
3 Dataset description	16
3.1 Atomic Force Microscopy	16
3.2 Introduction to TopoStats	17
3.3 Classes in the data	21
4 Deterministic pipeline	24
4.1 Input	24

4.2	Output	24
4.3	Preprocessing	26
4.4	Thresholding	27
4.5	Segmentation	28
4.6	R-loop detection	29
4.6.1	Tracing bright spots	31
4.6.2	Calculation of degrees	32
4.7	R-loop classification	36
4.7.1	Blobs	36
4.7.2	Forks	37
4.7.3	Loops	38
5	Machine learning pipeline	40
5.1	Introduction and motivation	40
5.2	Selected features	41
5.3	Tested algorithms	44
5.3.1	Decision Trees	44
5.3.2	Multi-layer perceptron	47
5.4	Training methodology	52
6	Results	53
6.1	Evaluation metrics	53
6.1.1	Recall	54
6.1.2	Precision	54
6.1.3	F1 score	54
6.1.4	Balanced accuracy score	55
6.2	Multi-class classification	55
6.3	Binary classification	58

6.4	Feature importance	59
7	Discussion	63
8	Conclusion and future directions	73
	References	75

List of Figures

2.1	A figure depicting the DNA:RNA hybrid (brown-blue) and the associated single DNA strand (green).	3
2.2	Different types of R-loops observed in AFM images (Holleman et al. 2025).	15
3.1	An AFM image with the height colourbar extracted by TopoStats. . .	17
3.2	The same image in Figure 3.1 with the applied grain mask.	18
3.3	A visual representation of some TopoStats features, inspired by the figures in the TopoStats documentation.	20
3.4	Distribution of aspect_ratio feature across different R-loop classes. All points are slightly transparent. A more opaque point indicates the existence of multiple points in that location.	22
3.5	Distribution of total_contour_length feature across different R-loop classes. The unit for the <i>y</i> -axis is meters. All points are slightly transparent. A more opaque points indicates the existence of multiple points in that location.	22
3.6	Distribution of num_crossings feature across different R-loop classes. The unit for the <i>y</i> -axis is meters. All points are slightly transparent. A more opaque points indicates the existence of multiple points in that location.	23

3.7	Distribution of <code>average_end_to_end_distance</code> feature across different R-loop classes. The unit for the y -axis is meters. All points are slightly transparent. A more opaque points indicates the existence of multiple points in that location.	23
4.1	An overview of the deterministic pipeline.	25
4.2	(a) A raw (unedited) image of a DNA molecule. (b) The denoised version of the image in (a).	26
4.3	Transformation of an AFM image. (a) Original image of a DNA molecule. (b) Grayscale version of the image in (a). (c) Binarized version of the image in (b).	28
4.4	Detected contours drawn on top the original image in Figure 4.3(a). Boundary pixels are green. The inner contour is pink. The pipeline finds a parent-child loop.	29
4.5	A zoomed in portion of Figure 4.3(a). There are 2 bright spots inside the DNA boundary and 2 bright spots outside.	30
4.6	(a) An artifical DNA molecule in 3D. (b) The 2D projection of the same DNA molecule in (a). The loops in (b) are examples of what we call fake loops.	31
4.7	Detected bright spots in Figure 4.5. Magenta spots are inside the DNA boundary. Blue spots are omitted from the next steps in the pipeline as they are outside of the DNA molecule.	32
4.8	Light blue contour is the spline of the DNA molecule. Everything else was drawn by hand for demonstration purposes. The junction point is magenta and the intersections are orange. The boundary of the fixed radius circle is green. Note that this image also illustrates an example of a fake loop.	33

4.9	The tracing for the spline given in Figure 4.8. Magenta arrows show the orientation of the tracing. The start and endpoints have a degree of 1. Degree 4 pixels are located inside the green circle. The rest of the (intermediary) points have a degree of 2.	34
4.10	A cluster of degree 4 junction points (magenta). The medoid of the cluster is the dark blue pixel. The light blue contour is the spline. . .	36
4.11	(a) The original image of a DNA molecule. The pipeline identifies the bright spot in (b) and finds no junction points in the spline as shown in (c). The pipeline classifies the DNA molecule in (a) as a blob. . . .	37
4.12	(a) The original image of a DNA molecule. The pipeline identifies a degree 3 junction point in (b) and verifies that there is no parent-child loop nearby. The pipeline classifies the DNA molecule in (a) as a fork.	38
4.13	(a) A DNA molecule with a fake loop. (b) A DNA molecule with a real R-loop classified as loop.	39
5.1	An example of a DNA molecule that contains a blob with more than 2 leaves. (a) The original image of this DNA molecule. (b) The spline of the same molecule obtained from TopoStats.	43
5.2	An example of a leaf point not counted towards <code>number_leaves</code> because it is outside the DNA boundary. (a) The DNA molecule spline obtained from TopoStats. (b) The boundary of the same DNA molecule in (a).	44
5.3	An example decision tree that filters spam mail. The decision tree checks 5 conditions. <i>C1</i> checks if there are banned keywords. <i>C2</i> checks if the sender is in a public spam sender list. <i>C3</i> checks if flagged keywords are overused. <i>C4</i> checks if the sender is blocked by the recipient. <i>C5</i> checks if the sender is in a public trusted sender list.	46

5.4	A visual representation of an MLP neural network consisting $l + 1$ layers for a single label classification problem. b_l denotes the bias entry in layer k . n_i^l denotes the i th neuron in layer l . $W_{i,j}^l$ denotes the weight of the connection between the j th neuron in layer l and the i th neuron in layer $l + 1$	48
6.1	The left portion of the decision tree generated by the machine learning pipeline.	60
6.2	The right portion of the decision tree generated by the machine learning pipeline.	61
7.1	(a) A DNA molecule with a small bright spot that was wrongly classified as a blob by our deterministic pipeline. The bright spot is highlighted with magenta. (b) The boundary of the same DNA molecule, highlighted with green.	64
7.2	(a) A DNA molecule with an unrelated bright spot near its boundary that was wrongly classified as a blob by our deterministic pipeline. The bright spot is highlighted with magenta. (b) The boundary of the same DNA molecule, highlighted with green.	64
7.3	(a) A DNA molecule with a junction point that was wrongly classified as a fork. (b) The skeleton of the same molecule, highlighted with light blue. This is also an example of a fake loop that is hard to detect.	65

7.4	(a) A DNA molecule that was wrongly classified as a fork. (b) The skeleton of the same molecule, highlighted with light blue. The green circles are used for the degree calculation. Dark blue points are medoids, and pink points are detected junction points. The DNA molecule in (a) makes a sharp turn, and green circles intersect different points of the molecule more than 3 times, creating fake junction points.	66
7.5	The right portion of the decision tree in Figure 6.2. The decision tree checks <code>x[2]</code> (aspect ratio) feature for entries with a high number of leaves and at least one bright spot. The orange leaf node is for blobs, and the purple leaf node is for entries with no R-loop.	67
7.6	SHAP values of features for the label “No” (no R-loops). Each dot represents an entry in the validation set. The colour of a dot represents its numerical value, not its importance. The feature at the top has the most impact. The feature at the bottom has the least impact. . .	68
7.7	Mean absolute of SHAP values across different features and classes for the label “No” (no R-loops). A feature with a larger mean absolute SHAP value will have more influence over the final decision. The feature at the top has the most impact. The feature at the bottom has the least impact.	69
7.8	SHAP values of features for the label “Blob”. Each dot represents an entry in the validation set. The colour of a dot represents its numerical value, not its importance. The feature at the top has the most impact. The feature at the bottom has the least impact.	69

7.9	Mean absolute of SHAP values across different features and classes for the label “Blob”. A feature with a larger mean absolute SHAP value will have more influence over the final decision. The feature at the top has the most impact. The feature at the bottom has the least impact.	70
7.10	SHAP values of features for the label “Fork”. Each dot represents an entry in the validation set. The colour of a dot represents its numerical value, not its importance. The feature at the top has the most impact. The feature at the bottom has the least impact.	70
7.11	Mean absolute of SHAP values across different features and classes for the label “Fork”. A feature with a larger mean absolute SHAP value will have more influence over the final decision. The feature at the top has the most impact. The feature at the bottom has the least impact.	71

List of Tables

5.1	Included and removed features from TopoStats.	42
6.1	Deterministic pipeline multi-class results.	56
6.2	Decision tree multi-class results with inflated training set.	56
6.3	Decision tree multi-class results without any inflation.	57
6.4	MLP neural network multi-class results with inflated training set. . .	57
6.5	MLP neural network multi-class results without any inflation.	57
6.6	Deterministic pipeline binary classification results.	58
6.7	Decision tree binary classification results with inflated training set. .	58
6.8	Decision tree binary classification results without inflation.	59
6.9	MLP neural network binary classification results with inflated training set.	59
6.10	MLP neural network binary classification results without inflation. . .	59

1

Introduction

R-loops are three-stranded nucleic acid structures mainly formed during DNA transcription that are involved in pathological and physiological processes (Zhu et al. 2025). They contain a DNA:RNA hybrid and an associated non-template DNA strand. Studies indicate that excessive formation of R-loops can impede DNA transcription, replication, and repair mechanisms, which leads to replication stress and genomic instability (Aguilera and García-Muse 2012; Hamperl et al. 2017; Hegazy, Fernando, and Tran 2020). They have also been linked to double-stranded breaks (Hegazy, Fernando, and Tran 2020; So and Martin 2019).

A recent study have found that nicks in the non-template strand can act as an initiator for R-loop formation (Holleman et al. 2025). This had led to the discovery of distinguishing features of R-loops in Atomic Force Microscopy (AFM) images. AFM images of these nick-initiated R-loops indicate the existence of three possible R-loop structures, namely *blobs*, *forks*, and *loops*. These classifications were done manually.

Despite the existence of many studies linking excessive R-loop formation to various types of instabilities, the exact mechanisms that govern R-loop formation are still ambiguous (Zhu et al. 2025; Wells, White, and Stirling 2019). There are tools

that track recurring patterns and energy-based approaches for R-loop detection in genomic sequences (Wongsurawat et al. 2012; Stolz et al. 2019; Jenjaroenpun et al. 2015; Ferrari et al. 2025). There are also machine learning tools that can predict R-loops in genomic sequences (Li et al. 2023; Hu et al. 2024). However, none of these tools performs R-loop detection and classification in AFM images. An automated tool that can accurately identify and classify R-loops in AFM images is an unmet need.

In this study, we introduce two computational pipelines that can identify and classify the majority of R-loops in AFM images. We have developed a deterministic computer vision pipeline that can track distinguishing features of R-loops in a given AFM image with the help of another AFM analyzing tool called TopoStats (Beton et al. 2021). We have also developed a machine learning pipeline that can learn determining characteristics of different R-loop structures using the features obtained from TopoStats and our deterministic pipeline. We have considered two main problems: multi-class classification and binary classification of R-loops. The former includes R-loop structures observed in AFM images that were manually classified before (blobs, forks and loops). The latter determines whether a given AFM image includes an R-loop (“Yes” or “No”) without classifying its structure.

The remainder of this thesis is organized as follows. Chapter 2 presents the relevant literature regarding the formation and the role of R-loops together with methods that are used to analyze their formation. Chapter 3 presents the dataset we used in this study and provides an introduction to AFM and TopoStats. Chapter 4 and Chapter 5 explain how the deterministic and the machine learning pipelines were implemented, respectively. Chapter 6 shares the results we obtained from both pipelines. Chapter 7 provides a comparison of the two pipelines and shares the limitations of our work. Finally, we share our conclusions and possible future directions in Chapter 8.

2

Literature review

2.1 R-loops and their formation

R-loops are three-stranded nucleic acid structures which contain a DNA:RNA hybrid and an associated single DNA strand. They are usually created during the process of DNA transcription. The RNA and the template DNA strand form a double helix structure for the transcription process. As the facilitator enzyme RNA polymerase moves along the DNA strand, a persistent DNA:RNA hybrid can be formed (Castillo-Guzman and Chédin 2021). In this case, the non-template DNA strand is referred to as the associated single DNA strand in the R-loop.

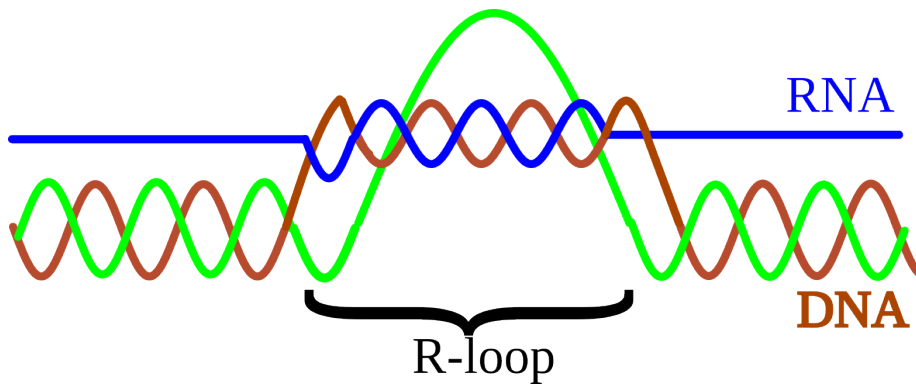


Figure 2.1: A figure depicting the DNA:RNA hybrid (brown-blue) and the associated single DNA strand (green).

There are two commonly accepted mechanisms used to explain R-loop formation. When the RNA involved in the transcription process is released from the transcription complex, it binds with the template strand behind RNA polymerase, forming an R-loop. Additionally, this RNA may also continue to stay attached to the template strand and form an R-loop inside the transcription complex (Zhu et al. 2025). Normally, R-loops are formed co-transcriptionally at the same locus as the RNA polymerase moves along the template DNA strand. These R-loops are considered to be *cis* R-loops. However, an R-loop can also be formed using RNA molecules at a remote locus. Such an R-loop is considered to be a *trans* R-loop (Zhu et al. 2025).

R-loops typically form in regions of DNA with high Guanine-Cytosine (G-rich) content (Allison and Wang 2019). Having more G-C bonds increases the stability of the DNA:RNA hybrid, making it more likely for R-loops to appear. There are also some factors that reduce the G-rich requirement and promote R-loop formation. For instance, negative supercoiling increases the likelihood of R-loop formation as it facilitates possible interactions between the involved RNA and the template strand (Allison and Wang 2019). Additionally, nicks in the non-template DNA strand can enhance R-loop formation even if it is not a G-rich region (Roy et al. 2010).

Although the DNA sequence itself is an important factor, how far it can influence R-loop formation is still a question (Zhu et al. 2025). As mentioned before, external factors that do not directly depend on the DNA sequence, such as negative supercoiling, can also affect R-loop formation. R-loop formation in regions of human and yeast mitochondria with replication initiators (sometimes referred to as heavy chain origin regions) specifically depends on negative supercoiling of the DNA template (Lee and Clayton 1996). The chromatin structure and epigenetic changes can also play a vital role (Zhu et al. 2025) as the former controls how accessible DNA is, and the latter affects gene activity. The methylation of histones (the DNA wrapping proteins in chromatin) leads to a tighter chromatin state, which is usually associated

with decreased R-loop formation (Zhu et al. 2025). For instance, decreased H3K9 dimethylation (H3K9me2) was shown to trigger R-loop formation at the ribosomal RNA (Zhou et al. 2020). On the other hand, histone acetylation leads to a more relaxed chromatin state, which is usually believed to promote R-loop formation (Zhu et al. 2025). As an example, the acetylation of H3K27 was shown to increase the likelihood of DNA:RNA hybrid formation (Chen, Koutelou, and Dent 2022).

Another external factor of significance is the presence of structures in the DNA strand. Noncanonical DNA structures, such as G4 (G-quadruplex) DNA, can also be a factor in R-loop forming mechanisms. Unlike the typical double helix DNA, G4 DNA is formed upon a base structure containing four guanines, also referred to as a G-tetrad. A G4 DNA is formed when multiple G-tetrads stack to form a tower, thus creating a square tower made out of guanines. The existence of a G4 structure promotes base pairings between G-rich RNA strands and C-rich DNA strands. If a G-rich region forms a G4 structure, the single-stranded DNA during transcription becomes more stable, leading to longer lasting R-loops (Duquette et al. 2004). R-loops with a G4 structure are referred to as G-loops in that work. The same study also verifies that the state of the strand, like supercoiled, relaxed or linear, meaningfully affects G-loop formation. The G4 structure may also increase the formation of *trans* R-loops since it introduces more distant regions (in the sense that they are further away from the actual transcription site) for a possible R-loop formation (Zhu et al. 2025).

It should be noted that these noncanonical DNA structures can also work against R-loop formation in some circumstances. For instance, PrimPol (a DNA polymerase) has the ability to skip G4 structures and some forms of DNA damage, which restarts the transcription. Skipping these regions reduces the opportunities to form R-loops (Guilliam and Doherty 2017).

2.2 The role of R-loops in genomic instability

A subset of persistent DNA:RNA hybrids associated with R-loops was linked to irreparable DNA damage and genomic instability (Costantino and Koshland 2018). Cells have evolved various mechanisms to remove existing R-loops or prevent new R-loop formation (Santos-Pereira and Aguilera 2015). Such a mechanism is desirable for a cell because R-loops can introduce genomic instability if they accumulate on a region (Aguilera and García-Muse 2012).

(Castillo-Guzman and Chédin 2021) proposes two primary classes of R-loops. Class I (promoter-paused) includes those formed frequently during transcription by RNA polymerase. Class II (elongation-associated) includes those formed occasionally during the elongation phase of transcription. Class I R-loops are usually detected near the proximity of the transcription start site, while Class II R-loops are longer and formed throughout the DNA strand (including the transcription start site).

As Class I R-loops are formed much more frequently than the others, there is a higher chance that they cause issues. Additionally, Class I R-loops are formed much closer to the transcription start site, which increases the likelihood that they play a role in gene alteration more frequently than others (Castillo-Guzman and Chédin 2021). Although Class I R-loops are the ones mainly associated with genomic instabilities, there are studies showing that excessive formation of Class II R-loops can also be problematic. For instance, (Stork et al. 2016) shows that the hormone estrogen (E2) stimulation in breast cancer cells greatly increases Class II R-loop formation, leading to DNA damage.

Just like how R-loop formation depends on a variety of factors, the level of instability caused by them also changes depending on the location and the structure of the R-loop in question. For instance, the G4 structures we mentioned before increase the formation of *trans* R-loops, which can be an issue since *trans* R-loops are more prone to causing problems compared to *cis* R-loops. This is because *cis* R-loops only

form near the transcription zone, whereas *trans* R-loops can be formed in remote locations, leading to more regions with the potential to create genomic instabilities (Zhu et al. 2025).

There are DNA and RNA helicases which specifically target DNA:RNA hybrids, in an effort to maintain genomic stability (Santos-Pereira and Aguilera 2015). Sgs1 (Chang et al. 2017), Dbp2, and Sen1 (Tedeschi et al. 2018) are some possible examples we can give amongst many. All of these use ATP to fight against excessive formation of DNA:RNA hybrids, and their removal causes R-loop accumulation and DNA damage. There are also enzymes such as RNase H, which specifically target the RNA portion of DNA:RNA hybrids (Stein and Hausen 1969).

The replication protein A (an eukaryotic single-stranded DNA binding protein) binds with the RNA and works with DNA polymerases to act as a replication re-starter. A possible mutation restricting replication protein A binding to RNA will potentially cause R-loop accumulation and genomic instability (Mazina et al. 2020).

Mutations of BRCA1 (breast and ovarian cancer susceptibility protein 1) and the BRCA2 (breast and ovarian cancer susceptibility protein 2) genes were shown to be related to R-loop stabilisation. It was observed that cells where BRCA2 was removed have increased levels of R-loops, providing a possible link between R-loop-caused genomic instability and cancer (Bhatia et al. 2014). The presence of R-loops may also prevent damage-induced transcription and impede the functions of BRCA1, which further contributes to DNA damage and genomic instability.

The mechanisms we talked about so far may not be effective if the transcription complex in question is twisted or tangled. For such cases, Topoisomerase 1 (TOP1) and Topoisomerase 2 (TOP2) help relieve DNA supercoiling during transcription by cutting and resealing DNA strands, preventing R-loop formation in the process (Allison and Wang 2019). If the process of transcription (between DNA and RNA) and the process of replication (between DNA and DNA) collide on the same DNA tem-

plate, the cell experiences a potential replication stress. This is sometimes referred to as a transcription-replication conflict. The conflict can arise in two orientations: co-directional (CD), which happens when both processes occur in the same direction, and head-on (HO), which happens when processes occur in opposite directions (García-Muse and Aguilera 2016). The study by (Hamperl et al. 2017) indicates co-directional R-loops can promote this conflict by preventing the progression of the replication fork.

Excessive formation of R-loops was also linked to double-stranded breaks (DSBs) and decreased DNA repair (Hegazy, Fernando, and Tran 2020). The latter is considerably problematic since decreased DNA repair also means fewer DSBs will be dealt with, which is not ideal for a cell, as insufficient repair of DSBs has been linked to genomic instabilities (So and Martin 2019). The cell response to DNA damage is primarily controlled by ATM and ATR protein kinases, which are enzymes that attach a phosphate molecule to a protein. The latter usually regulates the DSB response, whereas the former handles replication stress response (Blackford and Jackson 2017). Replication fork stress caused by an R-loop was shown to activate ATR, and the DSBs caused by this stress activate ATM (Saldivar, Cortez, and Cimprich 2017).

R-loops can be essential for cancers involving faulty DNA repair mechanisms. Preventing the cell from fixing the damage gives rise to further cancer cell accumulation. As an example, this situation was observed in BRCA1 deficient breast cancer. Decreased BRCA1 function led to an increase in R-loop formation and DNA damage (Gorthi et al. 2018), thus reducing the cell’s ability to respond to this issue.

Ewing sarcoma cells (cancer cells which form in bones and the tissue surrounding them) were shown to modify the damage-induced transcription mechanism, which leads to excessive R-loop formation and increased replication stress. In addition, Ewing sarcoma cells have a faulty homologous recombination mechanism due to the

interaction between BRCA1 and the transcription machinery, which further contributes to genomic instabilities in cancer cells (Gorthi et al. 2018).

R-loops have been linked to histone H3 S10 phosphorylation (H3S10P), which is closely related to chromatin condensation. These histone modifications were found in yeast, *Caenorhabditis elegans* and in HeLa cells (Castellano-Pozo et al. 2013). Histone H3 S10 has an important role in preventing genomic instability, as chromatin condensation is vital for safe DNA segregation during cell division. However, overly condensed chromatin regions increase the likelihood of gene silencing and transcription arrest, which increases genomic instability in the long run and aids cancer development (Castellano-Pozo et al. 2013).

To summarise, the role of R-loops in genomic instability is certainly quite complex, including DNA damage, alterations to the DNA repair mechanism, and transcription-replication conflicts. As we discussed, their regulation is vital for a cell, and the disruption of those regulatory mechanisms can be catastrophic.

2.3 Possible benefits of R-loops

Although unregulated R-loop formation is problematic for the reasons we mentioned so far, its existence is not always negative. There is evidence suggesting the existence of regulatory R-loops (Niehrs and Luke 2020). Provided that their existence is tightly regulated, these R-loops contribute to genomic stability by facilitating DNA transcription termination, DNA repair, and gene regulation (Niehrs and Luke 2020). Additionally, there is evidence indicating that regulatory R-loops can promote chromatin decondensation, effectively making DNA more accessible for various crucial cellular processes such as replication, transcription and repair (Powell et al. 2013).

Although R-loops have been linked to various types of abnormalities in a cell,

the precise relation between R-loops and diseases is still being investigated (Wells, White, and Stirling 2019). For instance, the mechanism by which BRCA1 regulates R-loop formation and the exact regions which play a role in this regulation are still questions. As such, the link between cancer cell growth and BRCA1's R-loop regulation mechanism is not straightforward (San Martin Alonso and Noordermeer 2021). To make matters worse, R-loop-induced genomic instability was shown to reduce tumorigenesis (the transformation of normal cells to cancer cells) under certain conditions. For instance, the knockout of the protein coding gene SKI8 (also known as WDR61) was shown to reduce the size, weight, and the risk of breast tumors by triggering genomic instabilities in tumor cells (Hou et al. 2024). Their investigation reveals that the knockout of SKI8 causes R-loop accumulation in breast tumor cells, leading to DNA damage, thus limiting their growth capacity.

We have talked about issues related to R-loop-induced genomic instabilities in the previous chapter. In the context of cancer, these instabilities are emphasized by the unusual methylation of tumor suppressor gene promoters (Kulis and Esteller 2010). These genes typically slow down cell division and trigger apoptosis (a programmed cell suicide) in an attempt to prevent cancer. There is evidence that R-loops located at gene promoters can provide protection against methylation by impeding DNA methyltransferase DNMT3B, which allows these useful genes to stay active and delay cancer progression (Ginno et al. 2012).

The long non-coding RNAs (lncRNAs) in the budding yeast *Saccharomyces cerevisiae* were shown to promote the transcription of specific genes in response to environmental signals (transcriptional induction). Controlled formation of these lncRNA:DNA hybrids makes the process of metabolic tuning (in particular, carbon source shifts) faster (Cloutier et al. 2016). In this case, R-loop formation acts as a gene expression regulator, allowing the organism to respond to environmental changes.

In the previous chapter, we presented evidence that R-loops can be linked to DSBs. However, the relation between R-loops and DSBs is more complex than one might imagine. There is also evidence linking R-loop formation and DSB repair (Hegazy, Fernando, and Tran 2020). There are two commonly accepted ways DSB repair can happen. The first one is a genetic recombination between two similar or identical DNA molecules to fix DNA damage, which is also called homologous recombination (HR) (Ceccaldi, Rondinelli, and D’Andrea 2016). The second one is called nonhomologous end joining (NHEJ) (Lieber 2010). NHEJ joins broken DNA ends without the need for a homologous template. HR tries to achieve a perfect repair by using another DNA molecule that is not broken. NHEJ, on the other hand, attempts to fix DNA damage more rapidly to prevent cell death, and it is the default repair mechanism of a cell because performing HR is not always possible. NHEJ is more likely to cause genomic instabilities by its very nature (Lieber 2010). As a response to DNA break, increased R-loop formation was detected around the proximity of DNA damage sites (Lieber 2010). (Keskin et al. 2014) proves the existence of a DNA repair mechanism where the transcript RNA was used as a template for DSB repair. In the absence of RNase H, the DSB repair can happen through the template RNA. The removal of the RNA portion of the DNA:RNA hybrid by RNase H was shown to significantly impede these processes in budding yeast, which indicates that transient DNA:RNA hybrids are needed for DSB repair (Ohle et al. 2016). Similarly, the removal of the RNA portion of the DNA:RNA hybrid in human cells was shown to impede both the HR and NHEJ processes (Lu et al. 2018).

2.4 Methods to analyze R-loops and their formation

One of the first attempts to computationally model R-loop formation was done in (Wongsurawat et al. 2012). Their work utilises the sequence analysis performed in (Roy and Lieber 2009). These led to the introduction of *R-loop forming sequences* (RLFS), which were discovered by analyzing certain recurring features of a given DNA sequence. The first publicly available computer program that can identify RLFS in a genomic sequence is QmRLFS-finder, which stands for Quantitative Model for R-loop Forming Sequence finder (Jenjaroenpun et al. 2015). This program uses a previously developed quantitative structural model (Wongsurawat et al. 2012) for RLFS prediction to detect the location of RLFS from a given genomic sequence. It identifies three regions correlated to RLFS: a G-rich region for R-loop initiation, a linker region, and an elongation region that is relatively longer compared to the other two. The identification is done by finding subsequences in the input sequence that match a pre-determined set of patterns.

Additionally, there are energy-based approaches to study R-loop formation, like the study by (Stolz et al. 2019). The same authors also developed a software called “R-looper”, which considers the topology of the analysed DNA together with the associated genomic sequences to make predictions on places where there is a higher likelihood of R-loop presence. Furthermore, they also developed a database called “RLBase” (Miller et al. 2023).

R-loop formation can also be analysed by using what is called a *formal grammar*. A *formal grammar* is a system for generating words (in the combinatorial sense). As an example, (Ferrari et al. 2025) introduces a set of terminal and non-terminal symbols together with production rules to describe R-loop formation. Each R-loop is represented by a word in this system. The grammar defines regions for the initiation

and the termination of an R-loop. Each k consecutive nucleotides are assigned a terminal symbol depending on the probability that they are included inside an R-loop. The grammar also defines a set of production rules related to the different stages of R-loop formation.

Another discrete mathematical structure used to analyze R-loop formation is tree polynomials (Liu et al. 2024). The authors try to understand the connection between co-transcriptional R-loop formation and RNA folding. Co-transcriptional R-loops are formed during transcription when the RNA involved in the transcription process binds with template strand. There is a competition between the R-loop formation and the process of RNA folding where the RNA fold on itself to form a secondary structure (Belotserkovskii et al. 2018). (Liu et al. 2024) develops a computational pipeline that can detect R-loops from genomic data. This pipeline calculates the tree polynomial representation of RNA secondary structures and finds a correlation between the coefficient sums of these tree polynomials and experimental data of R-loop formation when it is applied to R-loop forming sequences.

There are also machine learning tools to understand R-loop formation. deepR-loopPre is a deep learning tool that can detect plant R-loops in genomic data (Li et al. 2023). Compared to the pattern-based QmRLFS, deepRloopPre has lower precision but a higher recall. The same authors also maintain a publicly available database called R-loopAtlas, which contains information about detected R-loops in plants. For R-loops in human genes, there is a deep learning tool called DeepER, which can predict R-loop locations in a given genomic sequence (Hu et al. 2024). Compared to DeepRloopPre and QmRLFS, DeepER has higher recall and F1 scores, but it has a lower precision score. DeepER was also used to analyze the connection between repeat expansion diseases and R-loop formation. These are diseases caused by the expansion of short tandem repeats, which are short genomic sequences that repeat across a genome. There is evidence linking these repeats to increased R-loop

formation and transcriptional silencing (Loomis et al. 2014). After analyzing a set of genomes including repeat sequences previously linked to repeat expansion diseases, DeepER was able to detect R-loops at the CAG, GGGGCC, and CGG repeats, which aligns with the experimental data. DeepER also predicted R-loop formation at the repeats in GIPC1, and VWA1 genes, but there is currently no experimental result indicating that R-loops can be formed in these genes (Hu et al. 2024).

In addition to the methods analysing genomic sequences directly, (Carrasco-Salas et al. 2019) introduced the idea that there are different classes of R-loops that differ from each other in the way they interact with the corresponding DNA, leading to a more topological approach to classify R-loops. In order to test this claim, they used a combination of chemical and AFM approaches. A subsequent study found that R-loops put constraints on the corresponding DNA, and this difference was captured by the AFM images (Holleman et al. 2025). The same authors also reveal that the properties of the associated single-strand DNA are one of the determining characteristics of R-loops. This is the main reason we are considering the analysis of R-loop imaging data as a viable method for classification, which we hope will reveal crucial information about R-loop architectures.

(Holleman et al. 2025) also shows that single-strand DNA binding proteins significantly increase R-loop formation in vitro because they impede the removal of nascent R-loops, thus making it more likely that they survive longer. This indicates that these proteins can be R-loop regulators. They also found that nicks in the non-template strands initiate the R-loop formation process. Atomic force microscopy (AFM) images they obtained show that R-loops formed as a result of these nicks have unique features. The non-template strand in these R-loops usually branch away from the main DNA:RNA hybrid, thus giving rise to R-loops with forked features. They have found three different types of R-loops called *blobs*, *forks*, and *loops* (Figure 2.2). *Blobs* include a globular element on the DNA molecule. *Forks* have an

elongated region in addition to the template DNA molecule. *Loops* have a looped element emanating from the DNA molecule.

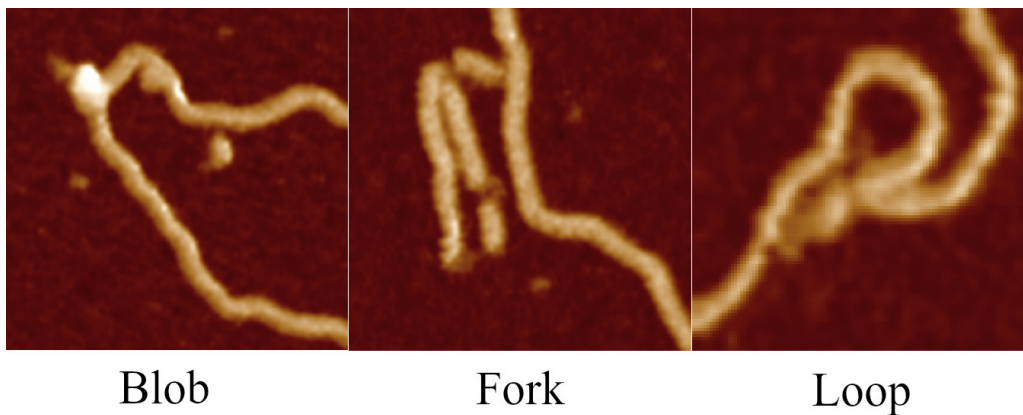


Figure 2.2: Different types of R-loops observed in AFM images (Holleman et al. 2025).

3

Dataset description

3.1 Atomic Force Microscopy

Atomic Force Microscopy (AFM) is a high-resolution version of Scanning Probe Microscopy (SPM), which can form images using a scanner probe (Binnig, Quate, and Gerber 1986). Its resolution is on the scale of nanometers, and it can produce images of objects well below the diffraction limit of a typical microscope.

AFM obtains data by moving its probe in a two-dimensional grid (x and y) and recording the information upon contact with the sample of interest. The probe is attached to a cantilever. The contact with the sample causes a deflection, which is recorded in a computer. This deflection is a measure of the force between the probe and the contact point, which includes a variety of sources, such as mechanical forces, electrostatic forces, and magnetic forces (Binnig, Quate, and Gerber 1986). Although its name includes “atomic force”, AFM does not actually use nuclear forces.

A detector turns the deflection into an electric signal, which is proportional to the deflection amount. The height of the probe can be adjusted until the deflection is no longer present. This difference in height provides a z value for that location. AFM continues this operation until all points in the grid are visited.

3.2 Introduction to TopoStats

Our dataset consists of raw AFM data, which uses the `.spm` (Scanning Probe Microscopy) file format, which is commonly used to store data from scanning probe microscopy devices. It includes metadata, such as the scanner type, the scan rate, details about sensors, and the microscopy model. It also includes the topography data of the sample (z coordinates). This is sometimes called the *height channel*. The height channel is used to create a pseudocolour image, where each pixel is assigned a colour value depending on the measured height at that location. From now on, we will call these images AFM images.

The dataset we use in this work was captured by the Pyne Lab (Holleman et al. 2025). We extract a 512x512 pixel image from this data using TopoStats (Beton et al. 2021), a tool for analysing AFM images. An example image extracted using TopoStats is presented in Figure 3.1. TopoStats also gets rid of some of the imaging artifacts during this extraction process.

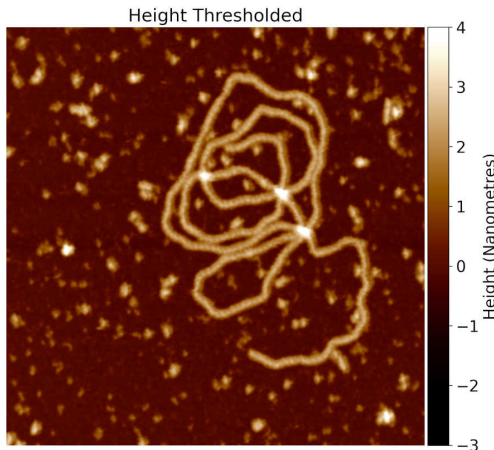


Figure 3.1: An AFM image with the height colourbar extracted by TopoStats.

On top of preprocessing AFM images, TopoStats also has the ability to trace molecules. It uses functions in the Gwyddion (a software for SPM data analysis)

libraries to identify molecules (Nečas and Klapetek 2012). Each detected molecule is assigned a mask with a unique number, which is called a *grain*. Masks are used to isolate certain parts of an image (see Figure 3.2). In this case, it tracks pixels that deviate from a pre-determined mean value. For AFM images, the pixels that satisfy this criterion are part of the DNA matter.

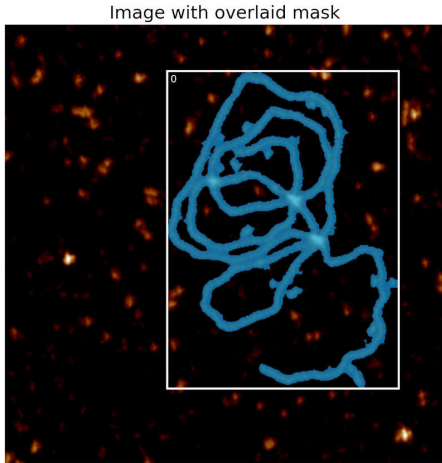


Figure 3.2: The same image in Figure 3.1 with the applied grain mask.

TopoStats calculates statistical information about each detected molecule. We combine this data with our own annotations and use that as input for our machine learning pipeline. A comprehensive list of these features is provided below. For a visual representation of some of these TopoStats features, see Figure 3.3.

Feature name	Description
center_x	Grain centre x coordinate
center_y	Grain centre y coordinate
height_min	Minimum height value for the grain mask
height_max	Maximum height value for the grain mask
height_mean	Mean height value for the grain mask

height_median	Median height value for the grain mask
radius_min	Minimum distance between center and perimeter
radius_max	Maximum distance between center and perimeter
radius_median	Median distance between center and perimeter
radius_mean	Mean distance between center and perimeter
area	Area of the grain mask
volume	Volume of the grain mask
area_cartesian_bbox	Area of the x - y axis-aligned bounding box
smallest_bounding_width	Width of the smallest bounding box
smallest_bounding_length	Length of the smallest bounding box
smallest_bounding_area	Area of the smallest bounding box
aspect_ratio	Ratio of the bounding width to length
min/max_feret	Minimum and maximum feret diameter
num_crossings	Number of crossing regions in the grain
total_contour_length	Length of the spline (skeleton) of a molecule
average_end_to_end_distance	Average distance between endpoints of splines

TopoStats can be configured by providing a custom `.yaml` configuration file to tune various parameters. For our analysis, we used the TopoStats version 2.4.0 and made the following changes to the default configuration. We have adjusted the standard deviation value for the grain mask and adjusted the area threshold for all grains. The minimum and maximum height values in Figure 3.1 is also set here.

```
filter:
  threshold_std_dev:
    below: 9.0 # Threshold for data below the image background
  remove_scars: # attempt to remove horizontal streaks or lines
    caused by scanning errors
```

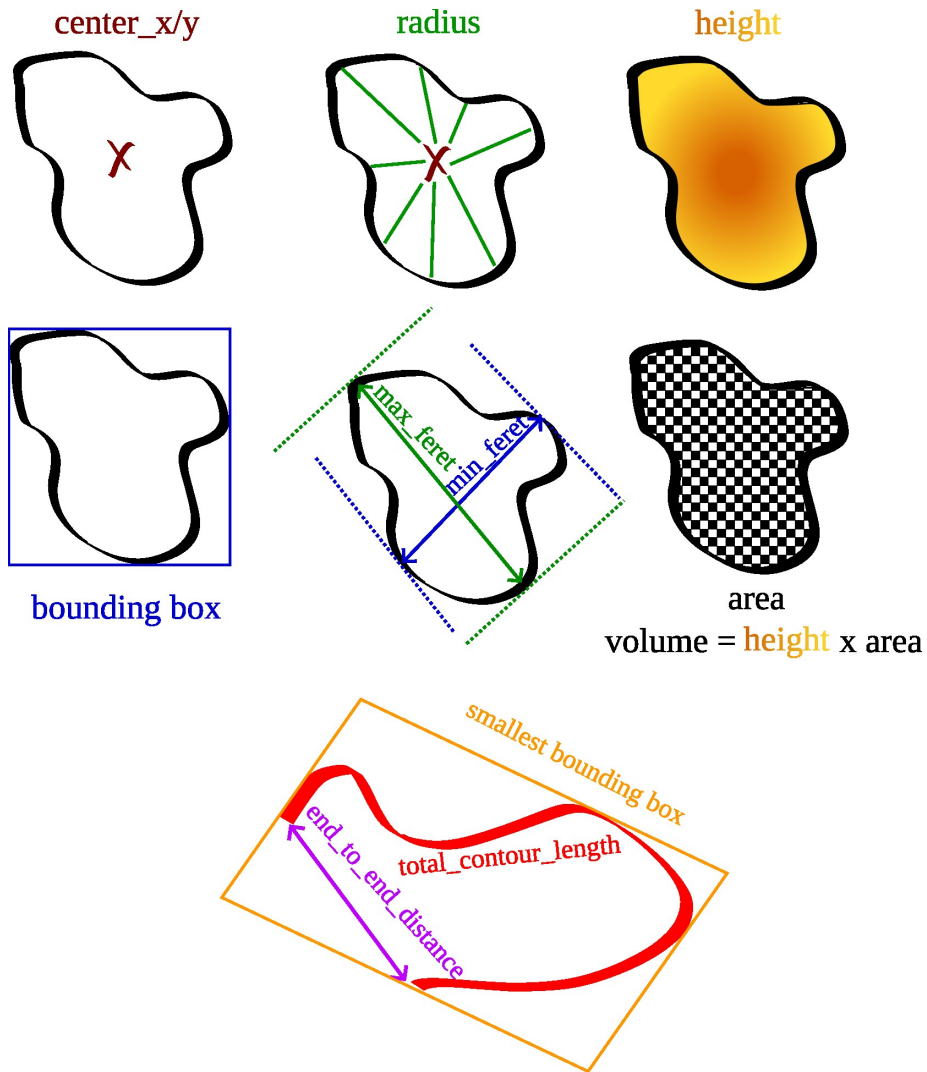


Figure 3.3: A visual representation of some TopoStats features, inspired by the figures in the TopoStats documentation.

```

run: true

grains:
  area_thresholds:
    above: [1000, 20000] # above surface [Min, Max] in nm^2

plotting:
  savefig_dpi: 200 # dots per inch value
  zrange: [-3,4] # min and max height range for images
  colorbar: false # disable colourbar
  axes: false # disable matplotlib axes

```

3.3 Classes in the data

We have 258 images in our dataset. Some of these images can contain multiple molecules. In these 258 images, there are 339 different molecules in total. The Pyne Lab manually annotated this data and gave a label to each of these molecules. Possible labels are [N] (none, means no R-loop present), [B] (blob), [F] (fork), [L] (loop). A detailed description of these classes is given in Chapter 2. For an example of AFM images of each class, see Figure 2.2.

We have observed that blobs in our dataset have a higher `aspect_ratio` on average compared to other classes, as seen in Figure 3.4. Forks, on the other hand, have a higher `contour_length` on average compared to other classes, as seen in Figure 3.5. None of these is a distinguishing characteristic, as there are outliers in both cases. To further illustrate how our dataset looks, we also provide the distribution of `num_crossings` and `average_end_to_end_distance` features in Figure 3.6 and Figure 3.7, respectively.

Our dataset has a class imbalance. The overwhelming majority of the entries have a label [N], meaning that they have no R-loops present. The overall distribution is as follows: 24 [B], 26 [F], 16 [L], and 273 [N]. This imbalance was not an issue for our deterministic pipeline (see Chapter 4), as it targets specific patterns in a given image and does not use the data itself to learn how to make decisions. However, the same can not be said for our machine learning pipeline (see Chapter 5) due to its heavy reliance on data. In that case, we had to inflate our dataset to get reliable results.

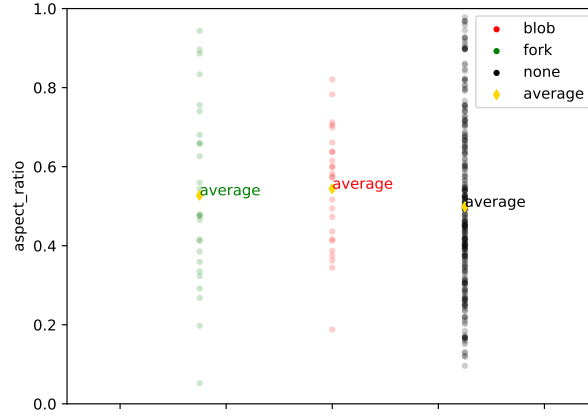


Figure 3.4: Distribution of `aspect_ratio` feature across different R-loop classes. All points are slightly transparent. A more opaque point indicates the existence of multiple points in that location.

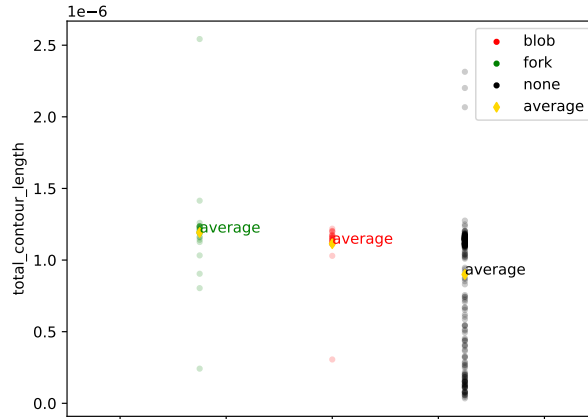


Figure 3.5: Distribution of `total_contour_length` feature across different R-loop classes. The unit for the y -axis is meters. All points are slightly transparent. A more opaque points indicates the existence of multiple points in that location.

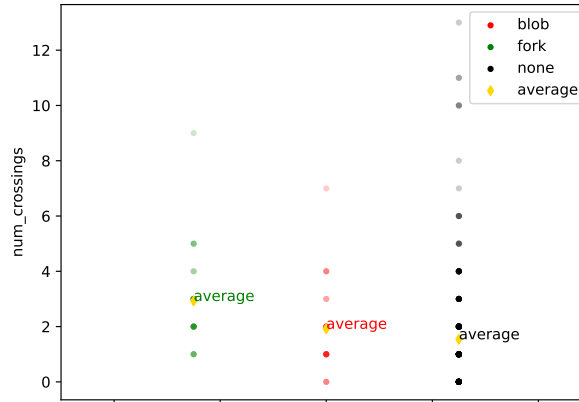


Figure 3.6: Distribution of `num_crossings` feature across different R-loop classes. The unit for the y -axis is meters. All points are slightly transparent. A more opaque points indicates the existence of multiple points in that location.

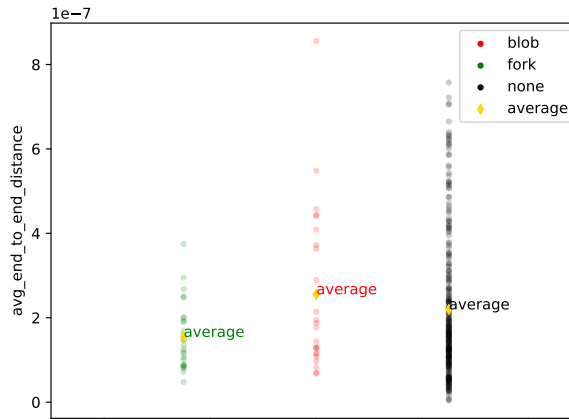


Figure 3.7: Distribution of `average_end_to_end_distance` feature across different R-loop classes. The unit for the y -axis is meters. All points are slightly transparent. A more opaque points indicates the existence of multiple points in that location.

4

Deterministic pipeline

4.1 Input

This pipeline takes AFM images in `.png` format as input. Images in our dataset have a resolution of 512x512. The input may contain multiple images. In that case, each image is analyzed separately. Images are processed using functions in `OpenCV`.

Although the pipeline is designed to work with AFM images, any other image would also work, provided that it uses raster graphics (images using a grid of colour values to represent graphics). `RGB` (`Red`, `Green`, `Blue`) is the only supported colour scale. If the input uses `RGBA`, the `A` (`Alpha`, for transparency) channel is omitted. Processing time will be longer for higher resolution images, but there is no hard-coded maximum image resolution that can be used as an input.

4.2 Output

The output of the pipeline is an image with the exact same resolution as the input. Depending on the user input, the pipeline returns an image in which:

- the boundaries (see Section 4.5) of the DNA molecules are highlighted.

- detected junction points (see Section 4.6.2) are highlighted.
- only the molecules containing an R-loop are highlighted.
- detected R-loops are highlighted with a different colour depending on their classification.

The user may also request access to a hierarchical tree containing all the coordinates of detected R-loops and the associated DNA molecules. The locations of all regions of interest, which are used to decide which class a detected R-loop belongs to, are also stored. The program can export all this information to a `.csv` file, if desired. An overview of the pipeline is provided in Figure 4.1.

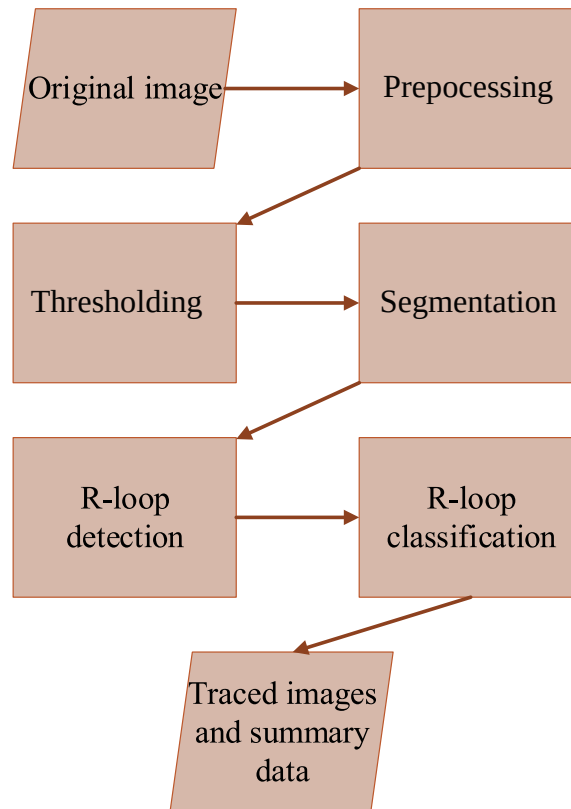


Figure 4.1: An overview of the deterministic pipeline.

4.3 Preprocessing

Noise around the DNA molecules is often problematic. The contour detection algorithm we are using (see Section 4.5 for more details) relies on a clear distinction between background and foreground pixels. Tracing is significantly harder to do if there is a lot of noise, especially around the boundary of the DNA molecules. The difference between the raw image of an example DNA molecule and its denoised version is presented in Figure 4.2. The noise also increases the likelihood of getting disconnected edges. The purpose of preprocessing is to eliminate these issues.

The input is denoised using an extension of Gaussian blurring called *non-local means denoising* (Buades, Coll, and Morel 2011). Gaussian blurring normally involves calculating the average colour value around a pixel and assigning that value to the same pixel. This makes the image less detailed, with the benefit of getting smoother edges. Non-local means denoising does the same, while also taking into account the patterns throughout the entire image, resulting in a denoised image without losing too much detail.

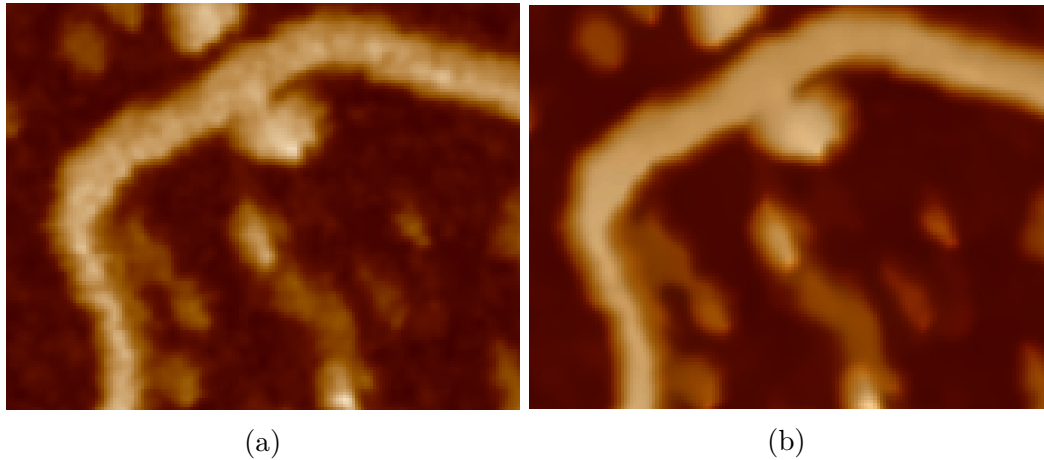


Figure 4.2: (a) A raw (unedited) image of a DNA molecule. (b) The denoised version of the image in (a).

4.4 Thresholding

The purpose of thresholding is to create a clear distinction between the DNA matter and the background. In other words, we would like to assign a different colour value for the parts of the image that need to be traced.

The pipeline starts by converting the colour scale of the input from **RGB** to **Grayscale**. The **Grayscale** is a colour scale consisting of different shades of gray, ranging from 0 (black) to 255 (white). This is done to reduce complexity as **RGB** colour values are not important for edge detection.

At this point in the pipeline, the background is assumed to have a darker colour compared to the material we are trying to trace. We consider the threshold $\lambda = 150$ out of the maximum value of 255. This corresponds to a mid-light gray pixel in **Grayscale** (see Figure 4.3(b)). The user may change this value or ask the program to determine an appropriate λ value. One should still verify that the selected λ value works reasonably well for the given image. Every pixel with a colour value above λ is considered to be a part of the DNA matter. We assign the value 1 to these pixels. Every other pixel is assigned the value 0. This makes it possible to completely binarize the image. In the binarized image, the background is pitch black, and DNA molecules are white, as seen in Figure 4.3(c). This process effectively provides a matrix consisting of only zeros and ones, which are easier to analyze.

In the event that we get disconnected edges, the threshold value λ is adjusted. This can be done automatically by the software or manually through user input. In the former case, we calculate the average value of the light coloured pixels over the entire image and decrease that value by 5 until there are no disconnected edges.

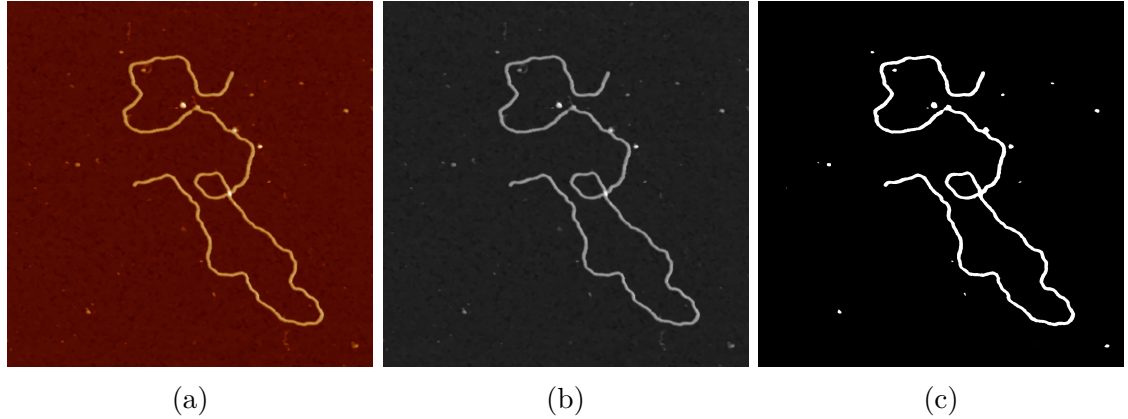


Figure 4.3: Transformation of an AFM image. (a) Original image of a DNA molecule. (b) **Grayscale** version of the image in (a). (c) Binarized version of the image in (b).

4.5 Segmentation

The purpose of segmentation is to obtain a multi-dimensional array containing all the different groups of pixels in a given input image. This makes it possible to provide a DNA molecule as an input to an R-loop detection or classification algorithm.

We trace contours in the input image by using Suzuki’s contour detection algorithm (Suzuki et al. 1985). This algorithm expects a binarized input. Binarization is performed automatically by the pipeline as explained in Section 4.4.

A boundary pixel is defined to be a pixel with a value of 1 which is also adjacent to at least one background pixel (which has a value of 0). The process starts from the top left corner of the input image and tries to find a boundary pixel. We call this the *index* pixel. Then, the algorithm checks whether the neighbour pixels of the index are also part of the boundary. We start a recursive process for any detected boundary pixel and check its neighbours as well. This process is repeated until the pixel in question is either an isolated boundary pixel or a pixel which is part of the background. Additionally, we stop the recursive process if all the neighbours of a pixel were already visited before. All the detected pixels are labelled (each gets a label 0 or 1). The set of points that have the same label creates a group of pixels

in the image (a DNA molecule in this case). Each group of pixels creates a contour in an image. An R-loop for our purposes is a contour or just a continuous set of points (see Figure 4.4). A contour may be contained inside another contour. The outer contour is considered to be the parent, and the inner one is considered to be the child. Note that a contour that passes the same point twice can be traced both from inside and outside. We consider such contours to be a *parent-child* loop, as they contain a child contour.

In the event that a detected contour has an inner contour, all the points in the inner contour are given a sub-label. The sub-label makes reference to the label of the parent, allowing the algorithm to build a hierarchical tree containing all the individual groups in the input image. The process is completed once all the pixels with a value of 1 are visited.

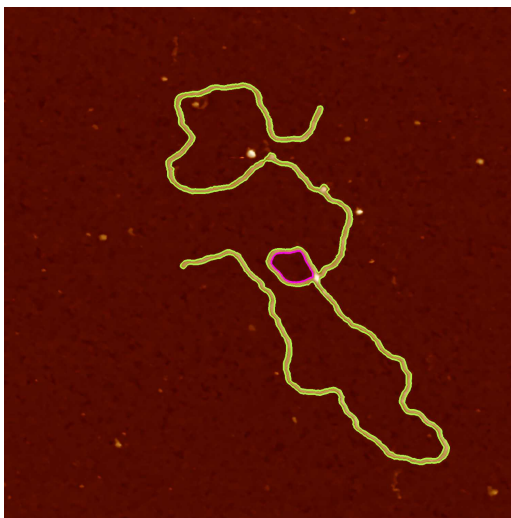


Figure 4.4: Detected contours drawn on top the original image in Figure 4.3(a). Boundary pixels are green. The inner contour is pink. The pipeline finds a parent-child loop.

4.6 R-loop detection

This part of the pipeline selects candidates for the R-loop classification algorithm, explained in Section 4.7. The groups captured in the segmentation step can include

unwanted artifacts, which normally appear as a distinctively bright spots. We trace all bright spots in the input image and check whether they are located inside the boundary of a DNA molecule. If a bright spot does not satisfy this condition, it is assumed to be an artifact and not a part of the DNA matter.

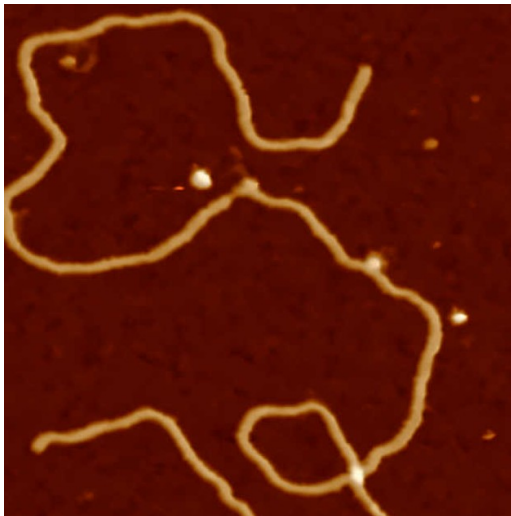


Figure 4.5: A zoomed in portion of Figure 4.3(a). There are 2 bright spots inside the DNA boundary and 2 bright spots outside.

As shown in Figure 4.5, these bright spots can also appear inside a DNA molecule naturally. A bright spot indicates that the height value of that spot is larger compared to the surrounding region. To determine whether this spot is a region of interest, one needs additional topological information.

We note that some bright spots appear due to the limitations of the AFM. The points which are not in close proximity may look very close when they are projected to a 2D plane (see Figure 4.6). Two points with the same x and y values with different z values will look identical if we were to take their projection. Similarly, the way a DNA molecule folds may look like a loop in a 2D image, but they are not useful for our purposes. We call such loops *fake* loops. Fake loops give rise to bright spots as seen in Figure 4.5 since the DNA molecule passes the same point twice and the detected height value by AFM increases around that point. Points with a higher value are assigned a lighter colour as explained in Chapter 3. Fake loops are also

parent-child loops. Therefore, not every parent-child loop is related to an actual R-loop structure. The feature that determines whether a parent-child loop is linked to an R-loop is the degree of points (see Section 4.6.2) in its proximity. The usage of these degrees is explained in detail in Section 4.7.

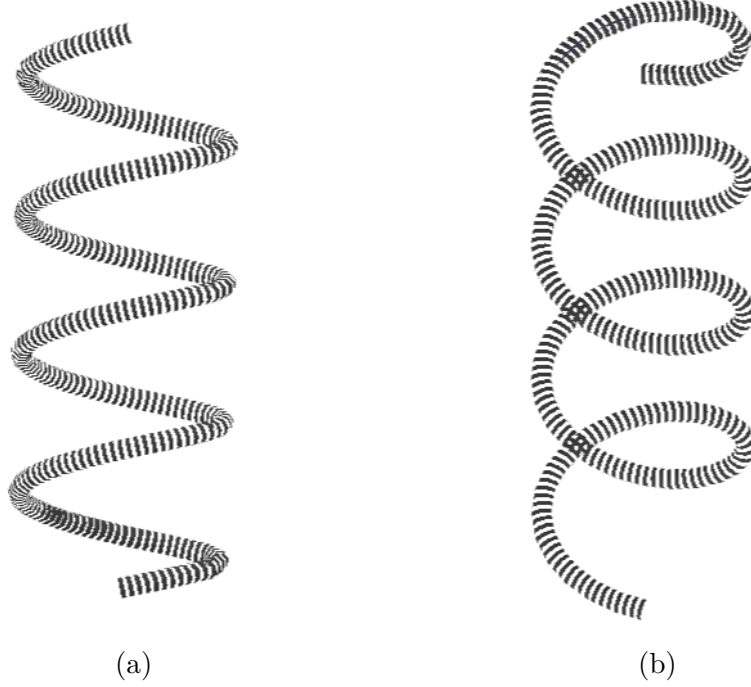


Figure 4.6: (a) An artificial DNA molecule in 3D. (b) The 2D projection of the same DNA molecule in (a). The loops in (b) are examples of what we call fake loops.

4.6.1 Tracing bright spots

We are tracing bright spots to avoid false positives as explained in Section 4.6. The location of these points is also used during classification (see Section 4.7). The tracing is done in this part of the pipeline in preparation for the R-loop classification. Bright spots are traced using the same method we utilized for boundary tracing in Section 4.5). The only difference is the threshold value λ . We pick $\lambda = 200$, which corresponds to a very bright white colour in the **Grayscale** colour code. The binarization in this case specifically targets bright spots instead of the DNA molecule itself. For an example, see Figure 4.7.

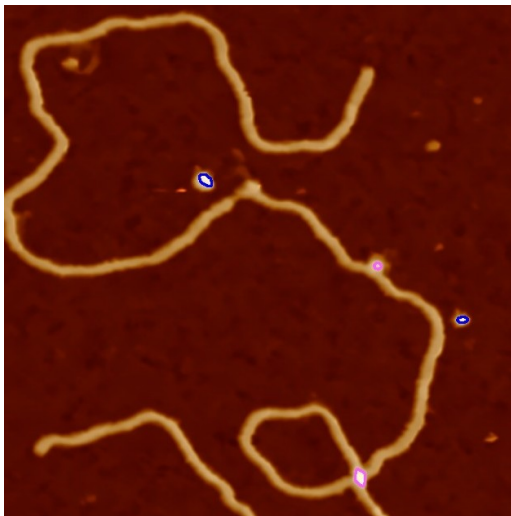


Figure 4.7: Detected bright spots in Figure 4.5. Magenta spots are inside the DNA boundary. Blue spots are omitted from the next steps in the pipeline as they are outside of the DNA molecule.

4.6.2 Calculation of degrees

The purpose of calculating degrees is to identify junction points in a given molecule. These points have a degree greater than or equal to 3. Junction points are crucial as relevant changes in a molecule happen around these points. This information is also used to classify R-loops, in Section 4.7. This process also allows us to identify parent-child loops that are not fake loops. We have observed that fake loops create a junction point of degree at least 4. An example of a fake loop with a degree 4 junction point is presented in Figure 4.8.

To detect these points, we make use of splines, which are traced by TopoStats. TopoStats provides a spline for each DNA molecule. It acts as the skeleton of that individual molecule. This is useful for detecting junction points. Our pipeline captures the entire boundary as a whole, but junction points are almost never a part of the boundary since they are located in the inner regions of a DNA molecule. This is the reason why we are using the splines traced by TopoStats. We consider a circle of fixed radius around each point in a given spline and count the number of times

that the boundary of that circle intersects the spline. This number is considered to be the degree of a pixel. See Figure 4.8 for an example.

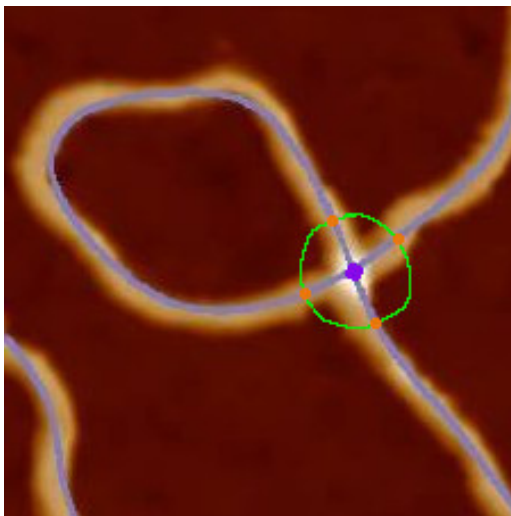


Figure 4.8: Light blue contour is the spline of the DNA molecule. Everything else was drawn by hand for demonstration purposes. The junction point is magenta and the intersections are orange. The boundary of the fixed radius circle is green. Note that this image also illustrates an example of a fake loop.

After iterating through all the pixels in a given spline and calculating the degree for each of them, we form a degree sequence. The endpoints of a spline will have degree 1. Intermediary points will have a degree of at least 2. Those with a degree of 3 and above are considered to be junction points. The reduced degree sequence for the spline in Figure 4.8 is given below. The original degree sequence is much longer. We took out some repeating values to fit it on this page, for demonstration purposes.

$$(1, 2, 2, 2, 2, 2, \underline{4}, \underline{4}, \underline{4}, 2, 2, 2, 2, 2, \underline{4}, \underline{4}, \underline{4}, 2, 2, 2, 2, 1)$$

The underlined elements are junction points. The degree sequence above is obtained by following the tracing in Figure 4.9. Although there are 2 different groups of $\{4, 4, 4\}$, their location are physically close in the image. All of these points are located inside the green circle in Figure 4.9. Therefore, there is no point keeping all duplicates of 4. It is more efficient to pick a representative among these points to

use in future calculations.

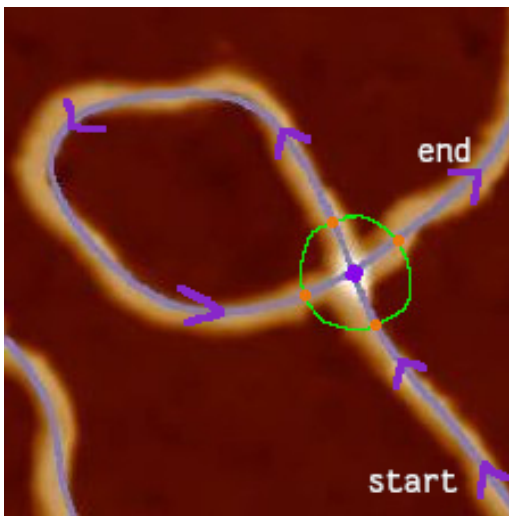


Figure 4.9: The tracing for the spline given in Figure 4.8. Magenta arrows show the orientation of the tracing. The start and endpoints have a degree of 1. Degree 4 pixels are located inside the green circle. The rest of the (intermediary) points have a degree of 2.

To pick a representative, we will first find clusters in the labelled dataset obtained by combining the spline coordinates and the corresponding degree sequence. We used the DBSCAN (Density-Based Spatial Clustering of Applications with Noise) algorithm (Ester et al. 1996) implemented in the `scikit-learn` library. We opted for DBSCAN over the more commonly used k -means for two important reasons. The first one is the unusual shapes of the clusters we are trying to detect. k -means assumes that clusters will be a convex shape, as it sees a cluster as a set of points whose distance to the cluster centroid is less than or equal to some pre-determined radius. However, the junction points we detect do not always form a convex shape. For instance, junction points in Figure 4.8 and Figure 4.10 form a concave “X” shape. On the other hand, DBSCAN clusters can be practically any shape. The second reason is that DBSCAN checks the distances between nearest points instead of picking a cluster centroid and working with a fixed search radius. If a junction point is in close proximity to another junction point in a cluster, DBSCAN will add

this point to the already existing cluster. This means that the cluster will continue to grow, even if the search radius is set to a small value, as long as junction points continue to appear along the spline of a DNA molecule. In other words, DBSCAN will keep going until none of the points in the cluster is in the proximity of another point that does not belong to the cluster. This is very useful for our use case, because it bundles all related junction points in a single cluster without needing to dynamically adjust the search radius.

DBSCAN provides us with clusters of points, where each cluster contains points with the same label. In our case, this gives us clusters of points having the same degree. We are only interested in clusters which have points with degree above or equal to 3. The rest is omitted.

Then, we calculate the *medoid* of each cluster and pick that point as the representative for a cluster. The medoid of a set of points is defined to be the point whose sum of dissimilarities (distance to other points) is minimal in the set. Let $X = \{x_1, x_2, \dots, x_n\}$ be a set of points. The medoid of X is

$$\arg \min_{x \in X} \sum_{i=1}^n d(x, x_i)$$

where d is a distance function. In our case, d is the typical Euclidean distance. This is similar to calculating the more commonly used *centroid*, but they are not the same. The centroid of a set is not guaranteed to be an element of that set, while medoid of a set will always be an element of that set (see Figure 4.10). In our case, all pixel coordinates consist of whole integers, and floating value coordinates would not be appropriate (without any rounding). We picked the medoid to not deal with this issue.

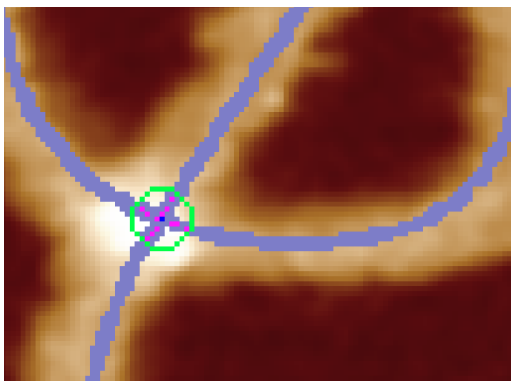


Figure 4.10: A cluster of degree 4 junction points (magenta). The medoid of the cluster is the dark blue pixel. The light blue contour is the spline.

4.7 R-loop classification

This chapter explains how the pipeline classifies regions of interest obtained in Section 4.6. Any object or region in a given image that makes it until this part of the pipeline is considered an R-loop candidate. A candidate that does not belong to any of the following classes is considered not to be an R-loop.

4.7.1 Blobs

The distinctive feature of R-loops of this type is that they have a bright spot somewhere along the DNA molecule. The reasoning is that the globular elements increase the height of the region they are located in. This difference creates brighter spots in AFM images. However, not all bright points count. An R-loop candidate should also satisfy the following properties.

- The bright spot should be contained inside the boundary of a DNA molecule.
- The degree of all spline points inside the bright spot should be at most 2.

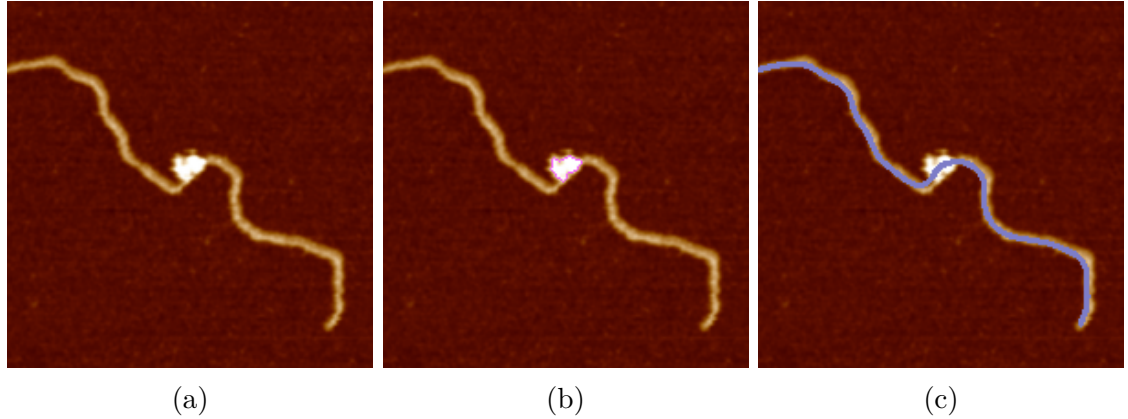


Figure 4.11: (a) The original image of a DNA molecule. The pipeline identifies the bright spot in (b) and finds no junction points in the spline as shown in (c). The pipeline classifies the DNA molecule in (a) as a blob.

The first property eliminates possible artifacts, and the second property makes sure that the bright spot is caused by a globular element instead of a fake loop. These loops will also have bright spots on top of their junction points. However, they do not correspond to R-loops.

The second property is equivalent to saying that the bright spot in question does not include any junction point inside its boundaries. The degrees of points in the spline of each DNA molecule are already calculated in Section 4.6.2. To check whether the second property is satisfied, it suffices to only consider medoids found in Section 4.6.2. If any of those medoids are located inside a given bright spot, we omit it as it can not be a blob. A DNA molecule that satisfies all the conditions to be a blob is presented in Figure 4.11.

4.7.2 Forks

The feature of R-loops of this type is a degree 3 junction point which resembles an actual fork. However, not all degree 3 junctions count. An R-loop candidate should also satisfy the following properties.

- The junction point should not be in the proximity of a parent-child loop.

- If the first property does not hold, the junction point should be contained inside a bright spot.

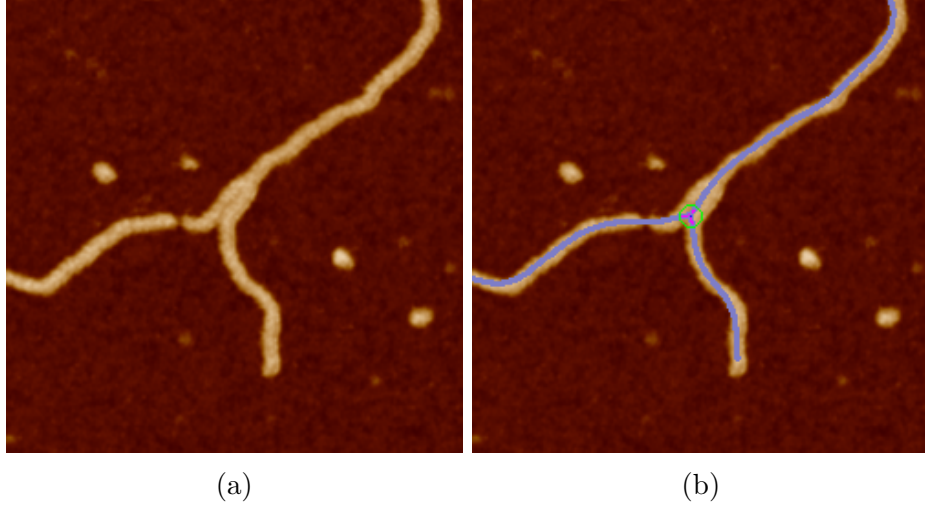


Figure 4.12: (a) The original image of a DNA molecule. The pipeline identifies a degree 3 junction point in (b) and verifies that there is no parent-child loop nearby. The pipeline classifies the DNA molecule in (a) as a fork.

The first property guarantees that the elongated region does not form a loop. R-loops of this kind are investigated in the next section. In the event that the first property is not satisfied, the second property makes sure that the loop in question (if there is one) is a fake loop. In other words, if the elongated regions went below or above some part of the DNA molecule, which looks like a loop due to the limitations of AFM explained in Section 4.6, we still consider that a fork. A DNA molecule that satisfies all the conditions to be a fork is presented in Figure 4.12.

4.7.3 Loops

Similar to forks, the distinctive feature of these R-loops is a degree 3 junction point. However, unlike forks, we expect this junction point to be associated with an actual loop. In other words, a possible candidate should satisfy the following properties.

- The junction point should be in the proximity of a parent-child loop.
- Assuming the first property holds, the loop in question should not be a fake loop.

Although the terminology is deceptive, one should not confuse this class with fake loops explained Section 4.6. We should mention that identifying this class of R-loops is extremely challenging. The looped element, in many cases, has a colour very similar to the background plasmid as seen in Figure 4.13(b). Neither our segmentation pipeline nor the splines provided by TopoStats properly traces these objects. Therefore, the results we will present do not include loops. The deterministic pipeline aims to detect forks and blobs.

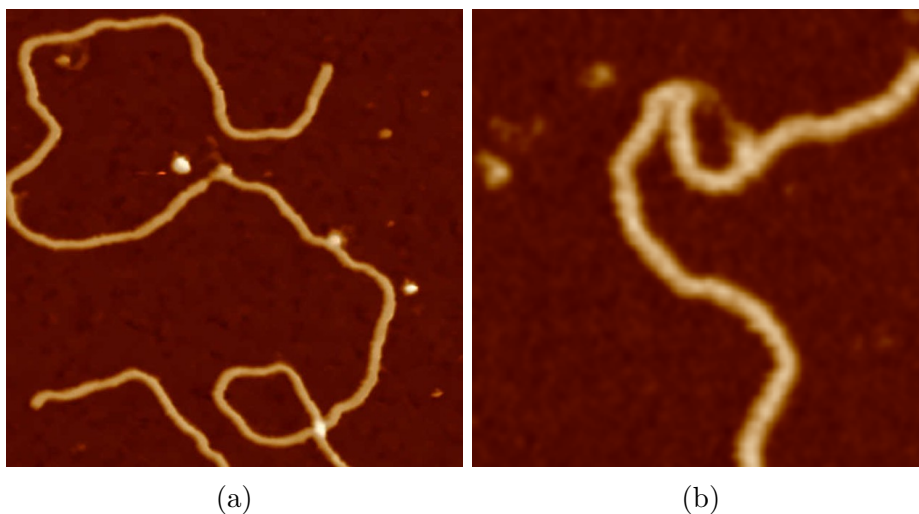


Figure 4.13: (a) A DNA molecule with a fake loop. (b) A DNA molecule with a real R-loop classified as loop.

5

Machine learning pipeline

5.1 Introduction and motivation

The deterministic pipeline tracks pre-determined patterns to detect and classify R-loops. This approach proved better than anticipated, but it had its limitations. Dealing with edge cases required constantly adding new code. An algorithm that can learn from the data and adapt itself to future changes is more desirable.

TopoStats generates statistical output for each molecule it finds, which they call `grain_statistics`. A comprehensive list of these features is provided in Chapter 3. We combined the features from TopoStats with extra features counted by our deterministic pipeline and created a labelled dataset. This chapter explains how we fed this data to two different supervised learning algorithms in an attempt to classify R-loops and understand which features can be a determining characteristic. To run these algorithms, we used the implementations in the `scikit-learn` library.

Since the machine learning pipeline heavily relies on the statistics generated by TopoStats, we are primarily targeting forks and blobs. R-loops that were classified as loops were removed from the dataset, and our final results do not include them. We initially tried using all the classes but failed to get reliable results. As men-

tioned previously in Chapter 4, neither TopoStats nor our deterministic pipeline was consistently able to trace these types of R-loops. Therefore, there is no reason to believe that the statistical values obtained from L (loop) entries properly represent topological properties of these objects. Although a minority of these L entries with proper tracing were successfully identified by the deterministic pipeline, we decided to not include those at all since the overall reliability of detection in this class is low.

5.2 Selected features

TopoStats provides 21 features for each detected molecule. Many of them are not useful for our purposes. Some features do not give information about the topology of the molecule, such as the centre point or location coordinates. Some features give information about the masks applied during the tracing of the molecule, which is also not useful for us. The table below explains which features were removed.

Feature name	Removed
center_x	Yes
center_y	Yes
height_min	Yes
height_max	Yes
height_median	Yes
height_mean	Yes
radius_min	Yes
radius_max	Yes
radius_median	Yes
radius_mean	Yes
area	Yes
volume	Yes

area_cartesian_bbox	Yes
smallest_bounding_width/length	Yes
smallest_bounding_area	Yes
aspect_ratio	No
min/max_feret	Yes
num_crossings	No
total_contour_length	No
average_end_to_end_distance	No

Table 5.1: Included and removed features from TopoStats.

All the height-related features, area and volume were removed because they provide information about the applied grain mask, not the molecule itself. Radius related features, Feret diameters, and `area_cartesian_bbox` features were removed after trial and error. We observed that their removal did not meaningfully affect the outcome. The smallest bounding width, length, and area were removed as they are all related to the aspect ratio feature. Amongst these features, we only kept the aspect ratio.

On top of the features above, we also added 2 additional features obtained from the deterministic pipeline, namely `number_bright` and `number_leaves`, which stand for the number of detected bright spots and the number of endpoints in a molecule, respectively. The term leaf is used in a graph theoretical sense here. It represents the points with a degree of 1.

The `number_leaves` feature is useful for identifying forks. A DNA molecule with no additional branches will usually have 2 leaves, namely the start and the endpoint of the spline. On the other hand, forks will usually have at least 3 leaves due to the extra branch needed for a degree 3 junction point. However, this feature is not a distinguishing characteristic of forks. Blobs can also have more than 2 leaves, though

it is rarer (Figure 5.1). If a branch in a DNA molecule is not traced by the spline provided by TopoStats, `number_leaves` will be smaller than its correct value. In other words, the accuracy of this feature heavily depends on TopoStats' tracing. To improve the accuracy, leaves outside the DNA boundaries were not counted (Figure 5.2).

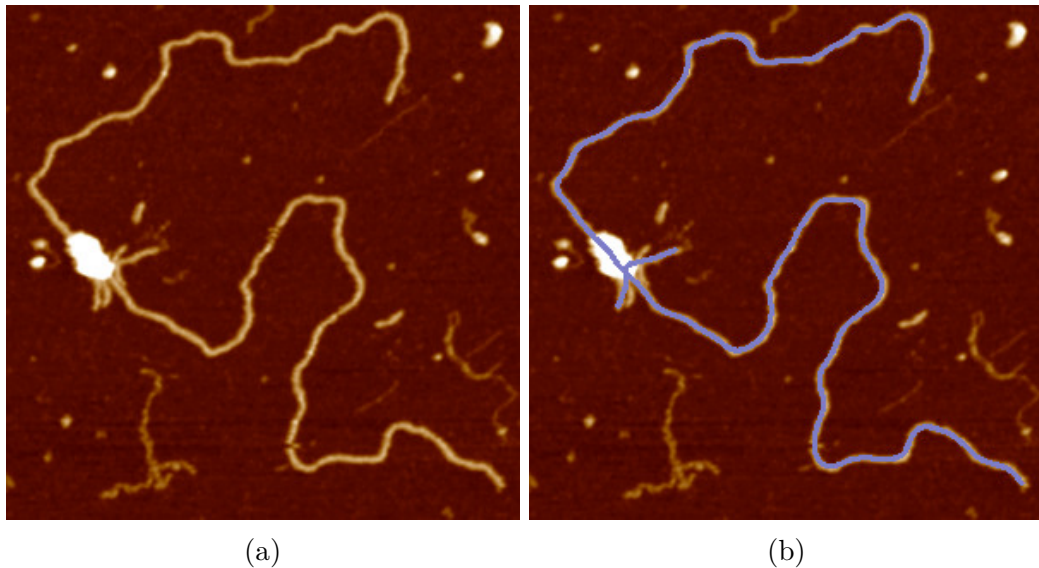


Figure 5.1: An example of a DNA molecule that contains a blob with more than 2 leaves. (a) The original image of this DNA molecule. (b) The spline of the same molecule obtained from TopoStats.

The `number_bright` feature is useful for identifying blobs. The existence of a bright spot does not necessarily indicate the presence of an R-loop. The bright spot in question can appear on top of a junction point or be an artifact. The opposite, however, is more useful. A DNA molecule without any bright spots can not include a blob. While counting the number of bright spots in a molecule, we only checked if a detected bright spot stays inside the boundary of that molecule. Unlike the deterministic pipeline, we have not performed any other checks. In other words, we did not filter out artifacts and bright spots located on top of junction points. The machine learning pipeline has no context regarding the location or the validity of these bright spots. The values obtained from our deterministic pipeline only play a

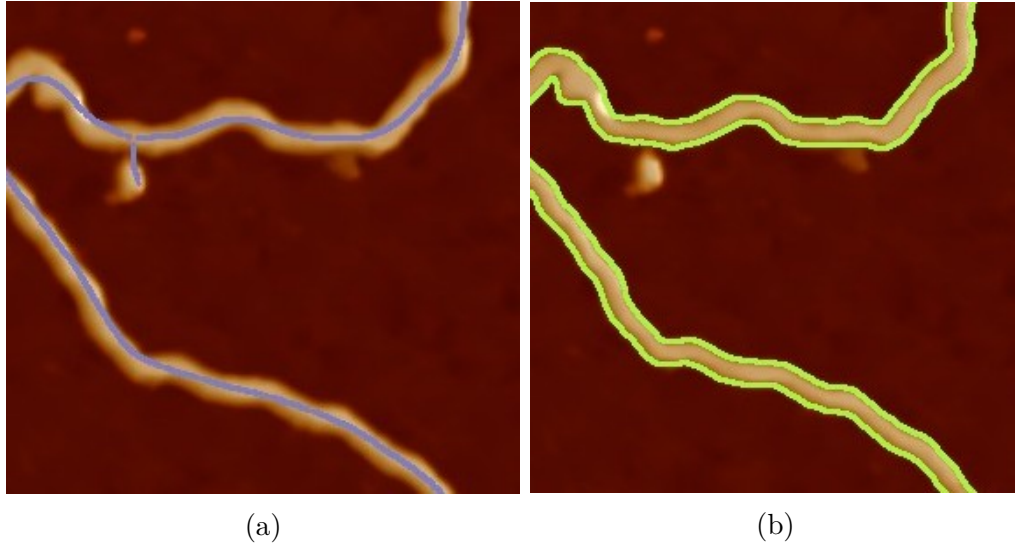


Figure 5.2: An example of a leaf point not counted towards `number_leaves` because it is outside the DNA boundary. (a) The DNA molecule spline obtained from TopoStats. (b) The boundary of the same DNA molecule in (a).

supporting role. We expect the machine learning algorithms to do the heavy lifting and make a decision using all the features.

5.3 Tested algorithms

This chapter presents the machine learning algorithms that were able to produce reliable results for our use case. Unsupervised learning algorithms, such as k -means (and its derivative clustering algorithms) and principal component analysis (PCA) did not perform well in our dataset. We have not observed a strong connection between the location of a data point in the space and its actual label. In other words, groupings made by unsupervised learning algorithms did not correlate with R-loop labels. Therefore, we only included supervised learning algorithms.

5.3.1 Decision Trees

Decision trees are a supervised learning method that tries to make predictions based on a collection of decision rules obtained from features. It is a whitebox

algorithm, meaning that its steps can easily be traced back. Understanding how decisions are made is as simple as following the nodes in a tree.

In a decision tree, leaves represent the labels. The root node contains all the entries in the training set. After every decision, entries are separated into two groups: those that satisfy a given condition and those that do not. This continues until all the entries in the training set reach a leaf node with a matching label. The set of decisions for classifying an entry forms a path from the root node to the associated leaf node in the tree. The trees used in this section are classification trees. If the predictions can be picked from real numbers (or some other continuous set of values), decision trees can be used for regression problems as well.

Decision trees excel at problems where classifications are made based on recurring patterns. One common use case is spam email detection (Subasi et al. 2018). Spam emails usually contain a set of words that are not commonly used for normal communication. These could be insults, threats or requests for confidential information. Overuse of words such as “urgent”, “lottery”, “crypto”, “prize”, “free” and many others can indicate that an email is spam. The mail address of the sender may also already be in a public spam sender list. All of these conditions can be checked by a decision tree to determine whether an email is possibly spam. An example decision tree that can filter out spam emails is provided in Figure 5.3. Decision trees can also be used for diagnosing diseases (Ghiasi, Zendehboudi, and Mohsenipour 2020; Shouman, Turner, and Stocker 2011), detecting credit card fraud (Sahin and Duman 2011) and fraudulent transactions in financial institutions (Işık, Kefe, and Sağlar 2023), sales forecasting to respond to changing market conditions (Thomassey and Fiordaliso 2006), and performing risk assessment for chronic diseases (Aljaaf et al. 2015).

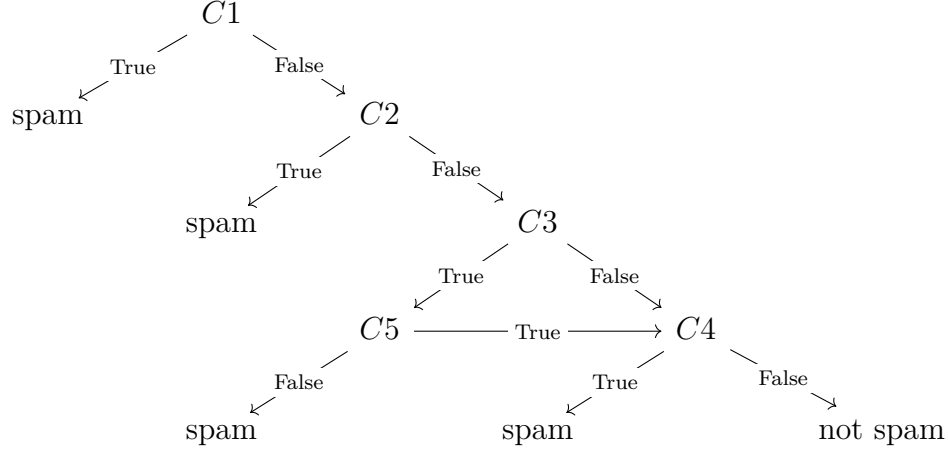


Figure 5.3: An example decision tree that filters spam mail. The decision tree checks 5 conditions. $C1$ checks if there are banned keywords. $C2$ checks if the sender is in a public spam sender list. $C3$ checks if flagged keywords are overused. $C4$ checks if the sender is blocked by the recipient. $C5$ checks if the sender is in a public trusted sender list.

Mathematically, given a set of feature vectors $f_i \in \mathbb{R}^n$, $i \in \{1, \dots, \ell\}$ and a label vector $l \in \mathbb{R}^\ell$, a decision tree attempts to find a partition that puts entries with the same label in the same group. The tree will recursively search for the best partition. What is considered “best” depends on the implementation of the tree.

Let D_γ be the data at a node γ in the decision tree and e_γ be the number of entries at a node γ . Every partition is a decision. For instance, $f < 0.5$, where f is a feature, can be a partition. Let t_γ be the threshold for a partition. Then, a partition S can be represented as $S = (t_\gamma, f)$. The partition S will separate the entries into two sets, namely $D_\gamma^{left}(S)$ and $D_\gamma^{right}(S)$.

Let $Q(S, D_\gamma)$ be the quality of the partition S at node γ . The values of the function Q are calculated with the help of a pre-determined impurity function I as follows:

$$Q(S, D_\gamma) = \frac{e_\gamma^{left}}{e_\gamma} I(D_\gamma^{left}(S)) + \frac{e_\gamma^{right}}{e_\gamma} I(D_\gamma^{right}(S)).$$

The goal of a decision tree is to find a partition $\bar{S}$ such that

$$\bar{S} = \arg \min_{\bar{S}} Q(S, D_{\gamma}).$$

In other words, a decision tree will attempt to find a partition that minimizes the impurity. The method for finding this partition also depends on the implementation of the tree. In our decision tree, we opted for the greedy exhaustive search method, which calculates impurities for any possible feature and threshold pair and picks the one with the minimum impurity. For the impurity function, we chose the commonly used Gini function. It is defined to be

$$\text{Gini}(D_{\gamma}) = \sum_l \frac{e_{\gamma}^l}{e_{\gamma}} \left(1 - \frac{e_{\gamma}^l}{e_{\gamma}}\right)$$

where e_{γ}^l denotes entries at node γ with label l .

5.3.2 Multi-layer perceptron

Multi-layer perceptron (MLP) is a supervised learning method. It is a feedforward neural network (Figure 5.4), which consists of layers of neurons and a nonlinear activation function. The first layer is the input layer. Neurons in this layer represent the features (input). Intermediary layers are called hidden layers. The final layer is the output layer (Pal, Mitra, et al. 1992).

MLPs excel at finding nonlinear relationships that are often missed by simpler models due to the overwhelming number of features needed to represent them. They can be used for typical classification problems mentioned in Section 5.3.1. In addition, MLPs can be used for handwritten digit recognition (Bellili, Gilloux, and Galinari 2003), preventing information loss from digital documents (Guha et al. 2021), face recognition (Boughrara, Chtourou, and Amar 2012; Bhattacharjee et al. 2010), and personalized content ranking (Hu et al. 2021).

In this pipeline, we used a neural network with 3 hidden layers. Each hidden layer has 20 neurons. In total, the neural network has 5 layers, including the input and output layer. By default, the implementation in `scikit-learn` uses a single hidden layer of 100 neurons. This did not perform well in our relatively small dataset. Therefore, we reduced the number of neurons in a single layer and added additional hidden layers. For our classification problem, the output layer has a single neuron, as each entry in our dataset has a single label. Multi-label problems can have more than one neuron in the output layer.

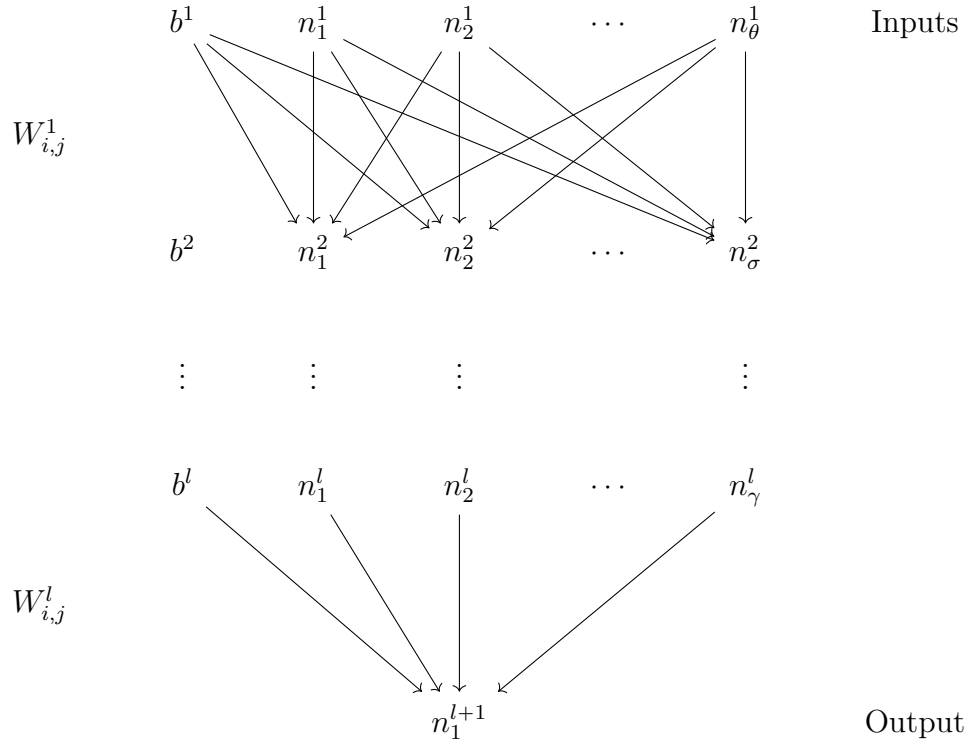


Figure 5.4: A visual representation of an MLP neural network consisting $l + 1$ layers for a single label classification problem. b_l denotes the bias entry in layer k . n_i^l denotes the i th neuron in layer l . $W_{i,j}^l$ denotes the weight of the connection between the j th neuron in layer l and the i th neuron in layer $l + 1$.

Every layer except the output layer has a bias entry for every neuron. This is added at the end of any calculation performed by a neuron. Let n_i^l denote the i th

neuron in layer l . Let $W_{i,j}^l$ denote the weight of the connection between n_j^l and n_i^{l+1} (Figure 5.4). Every neuron in a hidden layer takes values from the previous layer and transforms these values using a weighted summation (Pal, Mitra, et al. 1992). For example, the neuron n_i^l would calculate

$$W_{i,1}^{l-1}n_1^{l-1} + W_{i,2}^{l-1}n_2^{l-1} + \dots + W_{i,\ell}^{l-1}n_\ell^{l-1} + b_i^l$$

where ℓ is the number of neurons in the layer $l - 1$ and b_i^l is the bias entry for the neuron n_i^l . The first ℓ entries in this summation come from all the neurons in the previous layer that are connected to the neuron n_i^l . The value calculated by n_i^l is used for calculations performed in the next layers. One can put all the weight-related entries in a matrix and do these calculations at once for an entire layer using linear algebra. To be more compact, one could say that the neurons in layer l calculate

$$A(W^{l-1}n^{l-1} + b^l)$$

where W^{l-1} , n^{l-1} and b^l are the matrices containing weight entries in layer $l - 1$, neurons in layer $l - 1$ and bias entries in layer l , respectively. Values calculated by neurons are fed into an activation function A . The purpose of this function is to introduce nonlinearity to the neural network. As mentioned above, each neuron calculates a linear summation of inputs. Every layer performs a linear transformation on the input provided by the previous layer (also called forward propagation). This is not always ideal, as real data often contains nonlinear and complex relations. The image of a matrix under the activation function A is calculated in an element-wise manner. For our neural network, we are using the rectified linear unit (ReLU) activation function. It is defined to be

$$\text{ReLU}(x) = \frac{x + |x|}{2}.$$

Other common activation functions include the logistic sigmoid function and the hyperbolic tan (tanh) function. These are more expensive to calculate. We did not observe a meaningful difference between these activation functions for our use case. Therefore, we opted for the less complex ReLU function.

MLP learns the training set using the backpropagation algorithm (Pal, Mitra, et al. 1992). After a prediction is made, the neural network calculates the loss (or impurity). This is a metric of how successful the prediction was. We are using the Cross-Entropy loss function, which is defined to be

$$-\frac{1}{E} \sum_{i=1}^E \sum_{j=1}^C (y_{ij} \cdot \log(p_{ij}))$$

where E is the number of entries, C is the number of classes, and p_{ij} is the probability of entry i being a part of class j determined by the model. The function y_{ij} equals 1 if entry i is in class j ; otherwise, it equals 0. This summation iterates over all entries in the dataset and sums the logarithm of the model-predicted probability for each entry. If the model is 100% sure for all entries, the loss function obtains its smallest value. On the other hand, if the model is unsure and makes a lot of mistakes, the value of the loss function increases.

Backpropagation tries to understand how each weight in the neural network should change in order to decrease the value of the loss function, thus making the result better. To minimize the loss function, backpropagation uses the gradient descent method. Let $f : \mathbb{R}^m \rightarrow \mathbb{R}$ be the loss function. The gradient of f ($\nabla f : \mathbb{R}^m \rightarrow \mathbb{R}^m$) is defined to be

$$\nabla f(x) = \begin{bmatrix} \frac{\partial f}{\partial x_1}(x) \\ \frac{\partial f}{\partial x_2}(x) \\ \vdots \\ \frac{\partial f}{\partial x_m}(x) \end{bmatrix}$$

where $x = (x_1, x_2, \dots, x_m)$ is a point in $\mathbb{R}^m$. $\nabla f(x)$ gives the direction of the fastest increase. The gradient descent method goes the exact opposite direction to minimize a given function. Let $c_0 \in \mathbb{R}^m$ be a candidate for a local minimum of f . Let us pick

$$c_1 = c_0 - \mu \nabla f(c_0)$$

where $\mu \in \mathbb{R}^+$ is the learning rate of the method. One can see that $f(c_0) \geq f(c_1)$. In general, define

$$c_{n+1} = c_n - \mu \nabla f(c_n)$$

to form a sequence of values. The image of these values under f will create a monotonically non-increasing sequence.

$$f(c_0) \geq f(c_1) \geq f(c_2) \cdots$$

After finding a point that minimizes the loss function, the algorithm goes back to the weight matrix and updates it accordingly. For this weight optimization, we are using a limited-memory implementation of the Broyden–Fletcher–Goldfarb–Shanno (L-BFGS) algorithm (Liu and Nocedal 1989) in `scipy`. We also tried to use stochastic gradient descent (Robbins and Monro 1951) and adaptive moment estimation (Kingma and Ba 2014) for weight optimization, but observed that they did not perform well for our small dataset. Therefore, we opted for L-BFGS.

After the weights are optimized, the neural network repeats this entire process until the maximum number of iterations is met or the improvement of the loss function becomes negligible after a pre-determined number of steps. For our neural network, a maximum of 3000 iterations is allowed. If the value of the loss function does not decrease by at least 10^{-4} for 10 iterations, the process is considered complete, and the iteration process stops.

5.4 Training methodology

We used 70% of our dataset for training. The rest 30% was used for validation. The training and validation sets have $323 \cdot 0.7 \approx 226$ and $323 \cdot 0.3 \approx 97$ entries, respectively. The dataset is split in a stratified manner, meaning that the label proportions in the original dataset are preserved. The dataset is shuffled in each run, and entries in each group are picked randomly. This is done to check whether the pipeline produces consistent results that do not solely depend on a specific choice of training entries.

Due to the class imbalance in our dataset, both of the algorithms we tried default to picking the label N (none) as the dataset is dominated by these entries. There are not enough R-loop entries for the algorithm to properly learn. To deal with this issue, we inflated the training set by adding a copy of each R-loop entry. We did not pick specific entries to duplicate to avoid creating dense point masses in the feature space and prevent overfitting. The original dataset has 24 B (blob), 26 F (fork), and 273 N (none) entries. R-loop entries make approximately $34/226 \approx 15\%$ of the training set. The inflation increases this proportion to approximately $68/260 \approx 26\%$. Inflation was only performed in the training set. The validation set does not contain duplicate entries from the training set.

We also tried using SMOTE (Chawla et al. 2002) instead of creating exact duplicates. In our case, using SMOTE was not effective, as it often generated R-loop entries with very unusual features. Therefore, we opted for creating exact duplicates in the end.

MLP is sensitive to the scaling of input features. Therefore, we scaled the training features so that the training set has a mean of 0 and a standard deviation of 1. We also scaled the validation set by using the mean value and standard deviation obtained from the training set. The mean value and standard deviation calculations were only performed on the training set to prevent data leakage.

6

Results

In this chapter, we present multi-class and binary classification results for both pipelines. The multi-label problem uses the same labels explained in Chapter 3. The setup for the binary classification problem is explained in Section 6.3.

6.1 Evaluation metrics

Due to the class imbalance in our dataset, the typical accuracy score (the proportion of correctly predicted entries) does not truly represent how well a pipeline has performed. If we were to have an algorithm that attaches the label **N** (none) to any entry by default, it would technically have a success ratio of 85% as the R-loop entries are a minority group. Needless to say, such an algorithm would not provide any real-life value. For completeness, we will still include the accuracy score of both pipelines in this chapter. On top of that, the results will include recall, precision, F1, and balanced accuracy scores for each class. The aim is to put extra weight on R-loop entries and punish the algorithm when these entries are not predicted and classified properly.

6.1.1 Recall

Recall (sometimes referred to as sensitivity) measures the ability of an algorithm to correctly predict entries in a given class. Correctly predicted entries in a given class are considered to be *true positives*. We will denote the number of true positives by TP . Wrongly classified entries are considered to be *false positives* for the predicted class. The same entries are considered to be *false negatives* for the class they actually belong to. We will denote the number of false positives and false negatives by FP and FN , respectively. We shall use TP_X , FP_X , and FN_X to denote the number of true positives, false positives and false negatives for class X . Naturally, $TP_X + FN_X$ gives the total number of entries in class X . The recall score does not take false positives into account. An algorithm can have a high recall score while having high FP . The formula below calculates the recall score for class X :

$$\text{recall}(X) = \frac{TP_X}{TP_X + FN_X}.$$

6.1.2 Precision

Precision measures the ability of an algorithm to predict entries in a given class without having a high FP . An algorithm does not need to correctly predict all entries in a given class to have a high precision score. The formula below calculates the precision score for class X :

$$\text{precision}(X) = \frac{TP_X}{TP_X + FP_X}.$$

6.1.3 F1 score

The F1 score is defined to be the harmonic mean of recall and precision. It is a representation of recall and precision in one place. It combines them by giving each

metric equal weight. The highest F1 score possible is 1, which is obtained if the algorithm has a perfect recall and precision score. The F1 score approaches zero if either the recall or the precision score is low. The formula below calculates the F1 score for class X :

$$\text{F1}(X) = \frac{2}{\text{recall}(X)^{-1} + \text{precision}(X)^{-1}}.$$

6.1.4 Balanced accuracy score

Balanced accuracy score takes the average of the recall scores of each class. The effect of minority classes can be represented with this metric because each class has equal weight regardless of its size. A high balanced accuracy score requires good performance across the entire dataset. The formula below calculates the balanced accuracy score in a situation where n classes are present, namely $\{c_1, c_2, \dots, c_n\}$:

$$\text{Balanced accuracy} = \frac{\sum_{i=1}^n \text{recall}(c_i)}{n}.$$

At first glance, one might think that false positives are not accounted for in this metric because it is just an average of recall scores. However, that is not correct. Although a false positive will not reduce the recall score of the predicted class, it will reduce the recall score of its true class. Therefore, it will indirectly affect the balanced accuracy score.

6.2 Multi-class classification

This section presents the classification results for the multi-class problem. Table 6.1 presents results for the deterministic pipeline. For the machine learning pipeline, we present the average of 5 different runs. We included results for both the original and inflated datasets. Tables 6.2 and 6.3 show results for the decision tree. Tables 6.4 and 6.5 show results for the MLP neural network. For a detailed comparison of

the two pipelines and their limitations, please see Chapter 7.

The deterministic pipeline performs classification for all detected molecules in a given image. Therefore, unlike the machine learning pipeline, a single entry can hold multiple labels at the same time, though this is not very common. For instance, if an image contains a blob and a fork, it gives positive points to both classes, assuming the pipeline was able to identify both R-loops.

	Recall	Precision	F1
Blob	70%	64%	66%
Fork	71%	41%	52%
None	69%	93%	79%
Accuracy score	69.44%		
Balanced accuracy score	70.42%		

Table 6.1: Deterministic pipeline multi-class results.

	Recall	Precision	F1
Blob	62%	83%	71%
Fork	88%	100%	93%
None	99%	95%	97%
Accuracy score	94.84%		
Balanced accuracy score	82.92%		

Table 6.2: Decision tree multi-class results with inflated training set.

	Recall	Precision	F1
Blob	25%	40%	31%
Fork	62%	50%	56%
None	93%	91%	92%
Accuracy score	84.53%		
Balanced accuracy score	60.03%		

Table 6.3: Decision tree multi-class results without any inflation.

	Recall	Precision	F1
Blob	75%	86%	80%
Fork	75%	86%	80%
None	98%	95%	96%
Accuracy score	93.81%		
Balanced accuracy score	82.51%		

Table 6.4: MLP neural network multi-class results with inflated training set.

	Recall	Precision	F1
Blob	50%	57%	53%
Fork	62%	42%	50%
None	90%	94%	92%
Accuracy score	84.53%		
Balanced accuracy score	67.54%		

Table 6.5: MLP neural network multi-class results without any inflation.

6.3 Binary classification

The setup for binary classification problem is straightforward. Everything else except the labels stays the same. Class labels B (blob), F (fork), and N (none) are replaced with Y (yes) and N (no). In other words, we answer the question “Does this molecule contain an R-loop?” without classifying which type of R-loop it is. For the deterministic pipeline, no extra change is needed because it first performs R-loop detection and then classifies them. The former can be considered a binary classification. Similar to Section 6.2, the machine learning results here show the average of 5 runs.

Table 6.6 shows binary classification results for the deterministic pipeline. Tables 6.7 and 6.8 show binary classification results for the decision tree. Tables 6.9 and 6.10 show binary classification results for the MLP neural network.

	Recall	Precision	F1
No	66%	92%	76%
Yes	80%	50%	61%
Accuracy score	69.04%		
Balanced accuracy score	73%		

Table 6.6: Deterministic pipeline binary classification results.

	Recall	Precision	F1
No	95%	94%	94%
Yes	60%	73%	71%
Accuracy score	90.72%		
Balanced accuracy score	81.90%		

Table 6.7: Decision tree binary classification results with inflated training set.

	Recall	Precision	F1
No	90%	89%	90%
Yes	44%	47%	45%
Accuracy score	82.47%		
Balanced accuracy score	66.93%		

Table 6.8: Decision tree binary classification results without inflation.

	Recall	Precision	F1
No	95%	96%	96%
Yes	81%	76%	79%
Accuracy score	92.78%		
Balanced accuracy score	88.15%		

Table 6.9: MLP neural network binary classification results with inflated training set.

	Recall	Precision	F1
No	85%	92%	88%
Yes	62%	45%	53%
Accuracy score	81.44%		
Balanced accuracy score	73.84%		

Table 6.10: MLP neural network binary classification results without inflation.

6.4 Feature importance

In this chapter, we take a closer look at the decision tree generated by the machine learning pipeline to find the core features used by the tree and the decisions it is making. All entries start at the root node. If the given condition is satisfied, the entry

moves to the child node on the left. Otherwise, the entry moves to the child node on the right. $x[0]$, $x[1]$ and $x[2]$ represent the `number_leaf`, `number_bright` and `aspect_ratio` features, respectively (Figure 6.1). The `value` array shows the class distribution at that node. The order is [B, F, N]. For instance, the root node has 34 blobs, 38 forks, and 190 entries with no R-loop. Each node also contains the value of the impurity function, which is the Gini function in our case. One can observe that the impurity goes down as the tree progresses.

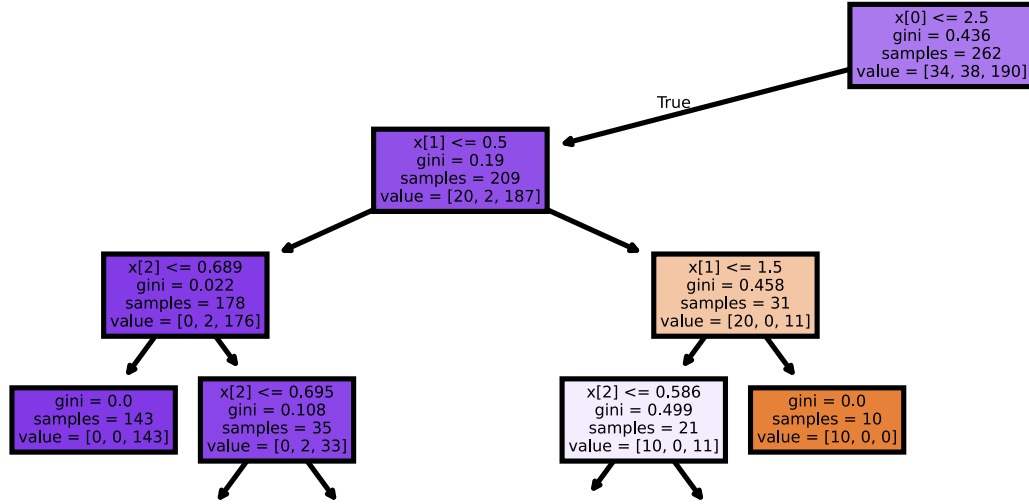


Figure 6.1: The left portion of the decision tree generated by the machine learning pipeline.

In Figure 6.1, we see that the decision tree starts by checking whether entries have strictly less than 3 leaves (the root node in the upper right corner). If that is the case, the tree checks if the entry contains any bright spots. If there are no bright spots (bottom left corner), the decision tree checks the aspect ratio of these entries and makes a decision. If the entry in question has a bright spot (bottom right corner), then this entry is either a blob or contains no R-loop. The tree labels such entries as a blob if they have more than 2 bright spots (the orange coloured leaf node on the right side). Otherwise, it checks the aspect ratio feature to make a final decision. The majority of N labelled entries were caught in the purple coloured leaf node on the left side. The fact that aspect ratio is used by the decision tree to

make the final decision should not be surprising, considering that blobs on average have a larger aspect ratio, as shown in Figure 3.4.

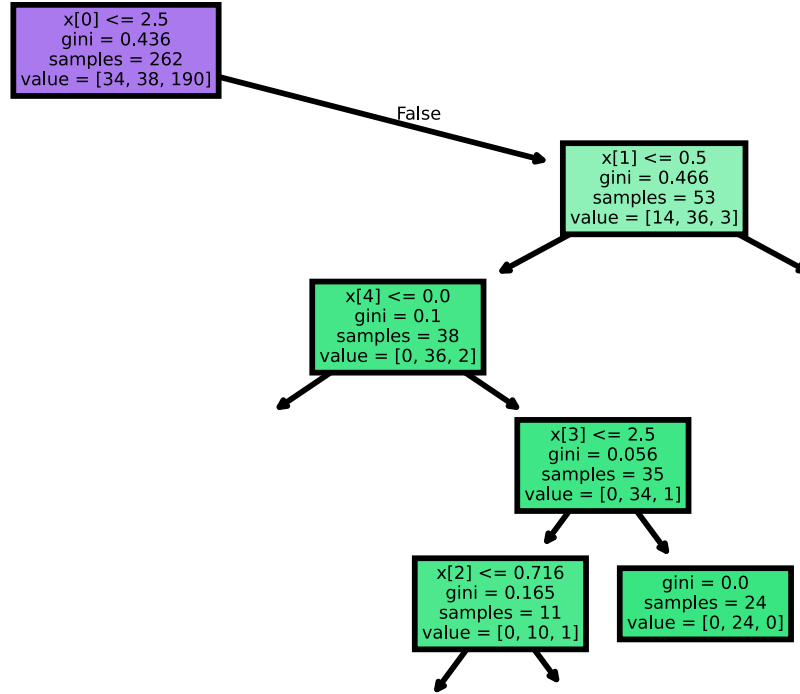


Figure 6.2: The right portion of the decision tree generated by the machine learning pipeline.

Figure 6.2 shows the right side of the same decision tree in Figure 6.1. In addition to the features explained before, Figure 6.2 uses $x[3]$ and $x[4]$, which stand for `num_crossings` and `total_contour_length`, respectively. We would like to point out that the condition $x[4] \leq 0$ in Figure 6.2 is in fact $x[4] \leq 0.0000013$. This value is rounded to 0 as it does not fit inside the rectangle representing the node.

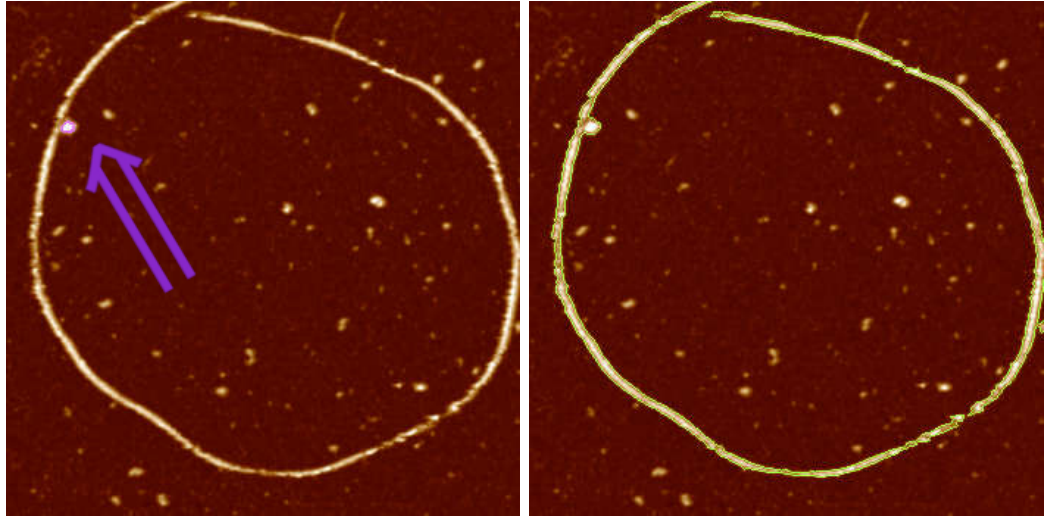
If an entry has more than two leaves (the root node in the top left corner of Figure 6.2) and it has strictly less than 1 bright spots, this entry will most likely be a fork. However, non R-loop entries may also satisfy these conditions, though it is rarer. The decision tree checks the total contour length of an entry to deal with such cases. This decision is not surprising considering that forks on average have larger total contour length, as shown in Figure 3.5. The final decision is made after checking whether the entry in question has strictly more than 2 junction points (bottom right

corner of Figure 6.2). If it does, the tree labels them as a fork (the green leaf node on the bottom right side of Figure 6.2). If it does not, it can still be a fork, but further analysis is required. The decision tree checks the aspect ratio feature again for such entries. Forks, on average, have a smaller aspect ratio compared to blobs, as shown in Figure 3.4.

7

Discussion

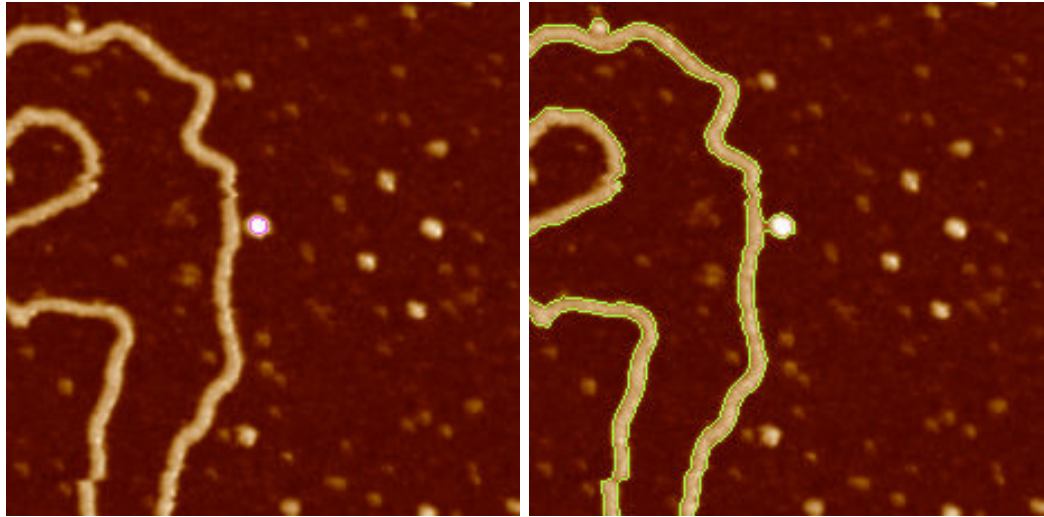
The deterministic pipeline has a balanced accuracy score of 70.42% (Table 6.1) for the multi-class problem, but it has a high number of false positives. The precision score for blobs in this case is 64%. The biggest obstacle for properly detecting blobs is the varying size of the globular element (the bright spot). We keep any bright spot with an arc length of greater than 18 pixels to catch the majority of bright spots, while skipping unrelated tiny objects and artifacts. However, this approach also traces many small sized bright spots that have no connection to an R-loop (Figure 7.1). Increasing this threshold is not a solution either because that causes the pipeline to miss many bright spots when they are relevant. The location of a bright spot can also cause confusion in the deterministic pipeline. Objects in the background are normally omitted, even if they contain a bright spot, because they are not contained within the boundary of a DNA molecule. However, if an irrelevant object (or an artifact) is very near the boundary of the DNA, they can be traced together (Figure 7.2), leading to a false classification. In any case, increasing precision leads to a smaller recall score, and having a higher recall score reduces the precision score. The F1 score is low in either case, though it is reasonable for a deterministic pipeline.



(a)

(b)

Figure 7.1: (a) A DNA molecule with a small bright spot that was wrongly classified as a blob by our deterministic pipeline. The bright spot is highlighted with magenta. (b) The boundary of the same DNA molecule, highlighted with green.



(a)

(b)

Figure 7.2: (a) A DNA molecule with an unrelated bright spot near its boundary that was wrongly classified as a blob by our deterministic pipeline. The bright spot is highlighted with magenta. (b) The boundary of the same DNA molecule, highlighted with green.

The deterministic pipeline has a precision score of 41% for forks (see Table 6.1) in the multi-class problem. The biggest obstacle to properly detecting forks is the large number of ways a junction point can be formed. Although we typically associate a degree 3 junction point with a fork, such points can be created naturally by the way a DNA molecule folds (Figure 7.4). It does not have to be created by an elongated branch. We may also see such points when the DNA molecule makes a sharp turn somewhere, and different parts of the same molecule come near one another, leading to a wrong classification (Figure 7.4).

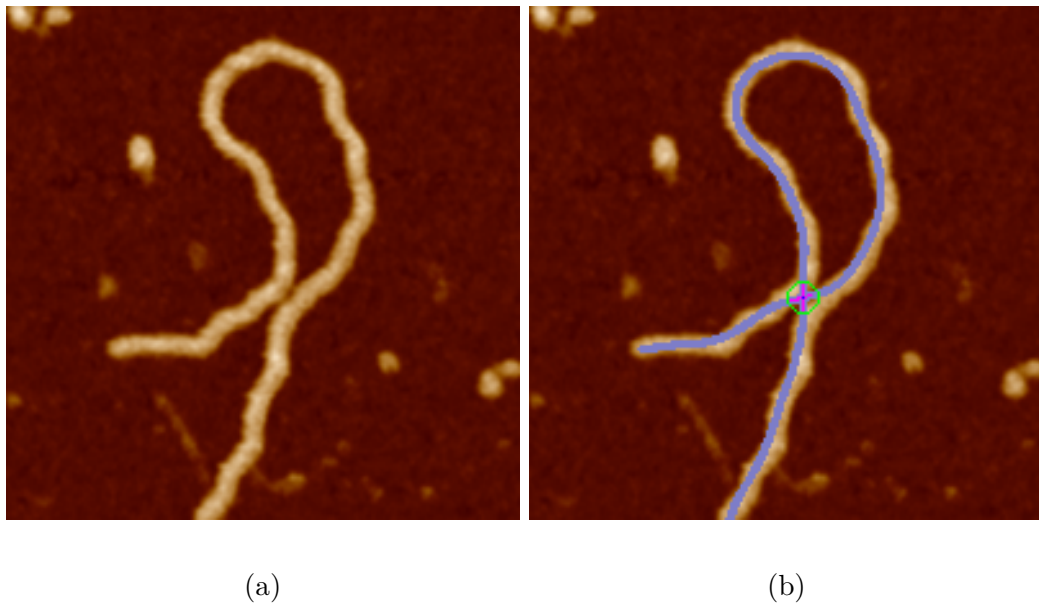


Figure 7.3: (a) A DNA molecule with a junction point that was wrongly classified as a fork. (b) The skeleton of the same molecule, highlighted with light blue. This is also an example of a fake loop that is hard to detect.

Figures 7.1, 7.2, 7.3, and 7.4 highlight a recurring issue with the deterministic pipeline: a slight mistake in the segmentation we perform or in the splines provided by TopoStats leads to a wrong classification. This is not to say that this pipeline provides no value. It will still identify R-loops in most cases, and the regions of interest that it highlights (such as the bright spots and the junction points) are still useful for a manual reviewer, even if the classification is not right.

On the other hand, the machine learning pipeline produces much more reliable

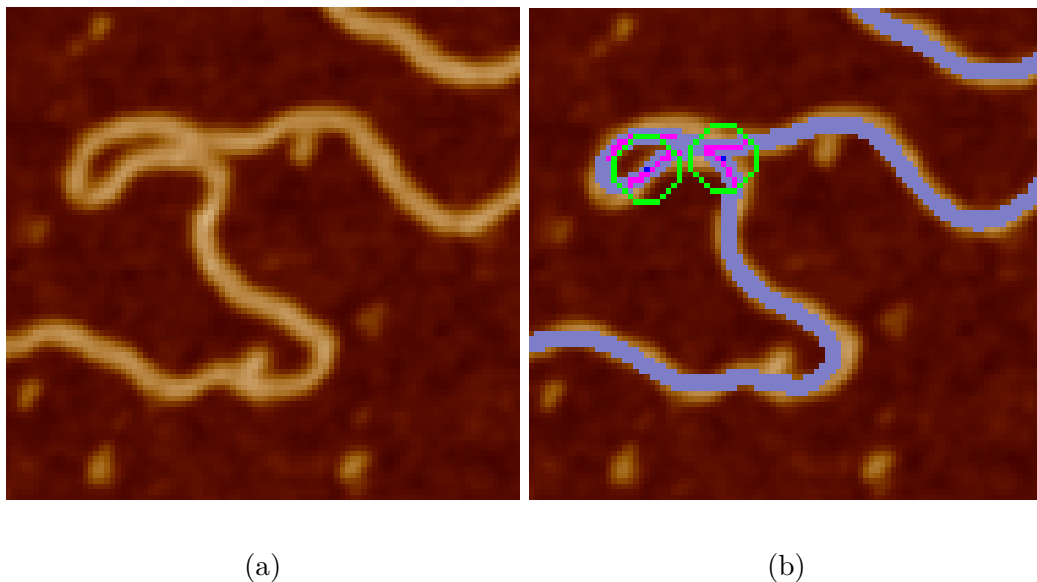


Figure 7.4: (a) A DNA molecule that was wrongly classified as a fork. (b) The skeleton of the same molecule, highlighted with light blue. The green circles are used for the degree calculation. Dark blue points are medoids, and pink points are detected junction points. The DNA molecule in (a) makes a sharp turn, and green circles intersect different points of the molecule more than 3 times, creating fake junction points.

results. The decision tree and the MLP neural network have a balanced accuracy score of 82.92% (Table 6.2) and 82.51% (Table 6.4), respectively. Both have higher recall scores compared to the deterministic pipeline, while also having a better precision score. The DNA molecules in Figures 7.1, 7.2, 7.3, and 7.4 were all classified correctly by the machine learning pipeline. In fact, this pipeline handles edge cases much better. The primary reason is that the machine learning pipeline uses multiple features to make a final decision. This does not guarantee that it will be successful all the time, but it certainly has better chances than the deterministic pipeline. If one feature is inaccurate or out of the norm, then it can still use other features to make a classification. For instance, our dataset contains entries with a high number of leaf points (above 2) while also having bright spots. The former is a common characteristic of forks, and the latter is usually observed in blobs. Such entries do not fit into the usual trend we observe in our dataset. However, the decision tree

can still correctly classify them (Figure 7.5).

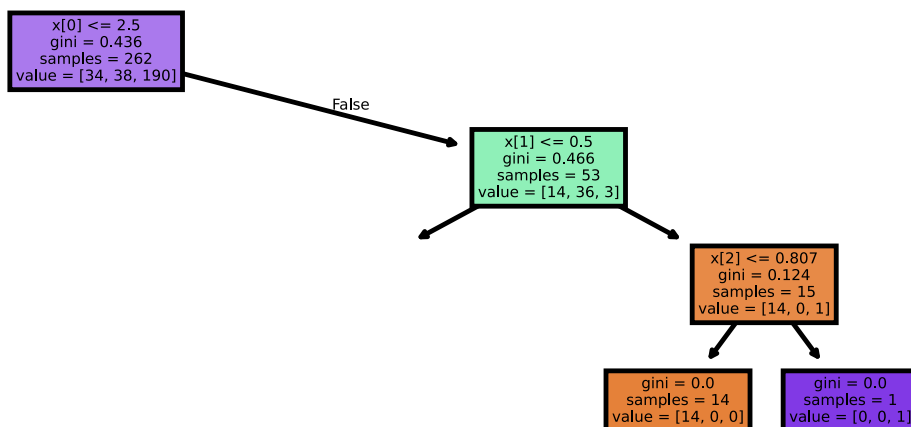


Figure 7.5: The right portion of the decision tree in Figure 6.2. The decision tree checks $x[2]$ (aspect ratio) feature for entries with a high number of leaves and at least one bright spot. The orange leaf node is for blobs, and the purple leaf node is for entries with no R-loop.

The binary classification results stay relatively the same for the deterministic pipeline, as seen in Table 6.6. The decision tree performs worse on the binary classification problem, with a recall score of 60% and a precision score of 73% for the R-loop class (see Table 6.7). One might think that the decision tree should perform better on the binary classification problem, considering that this problem is expected to be less complex than the multi-class classification. However, we see the exact opposite in reality. This happens because putting all fork and blob entries under the same label (“Yes”) makes classification harder. These entries have contradicting properties. For instance, the feature `number_leaf` tends to be higher for forks and lower for blobs. If we label these entries the same way, it is harder to determine distinguishing characteristics of R-loop-related entries, and the decision tree performs worse as a result.

The MLP neural network, on the other hand, performs better with a balanced accuracy score of 88.15% on the binary classification problem, as seen in Table 6.9. This is the highest score we observed in all of our results. Unlike decision trees, MLPs are not considered to be whitebox models. It is hard for a human to interpret the way

an MLP neural network makes decisions without any tools. To better understand our MLP results, we used a model-agnostic tool based on game theory called SHAP (Lundberg and Lee 2017). The exact mathematical formulation of SHAP is beyond the scope of this thesis. Therefore, we only present its general idea. SHAP is based on Shapley values (Shapley 1953) in cooperative game theory. Features are considered to be players in a coalition. The prediction of the model represents the total payout. Each feature in the coalition contributes its share to the total payout, which is based on how much each feature affects the model outcome. The SHAP value of a feature represents the average contribution of that feature across all possible subsets of features. In other words, SHAP calculates how the model behaves in any possible combination of subset of features and takes their average to determine the contribution of each feature. The SHAP values are calculated separately for each label. If a feature has a positive SHAP value for a label, knowing that feature will increase the likelihood that the model predicts that label. If the SHAP value is negative, the exact opposite occurs.

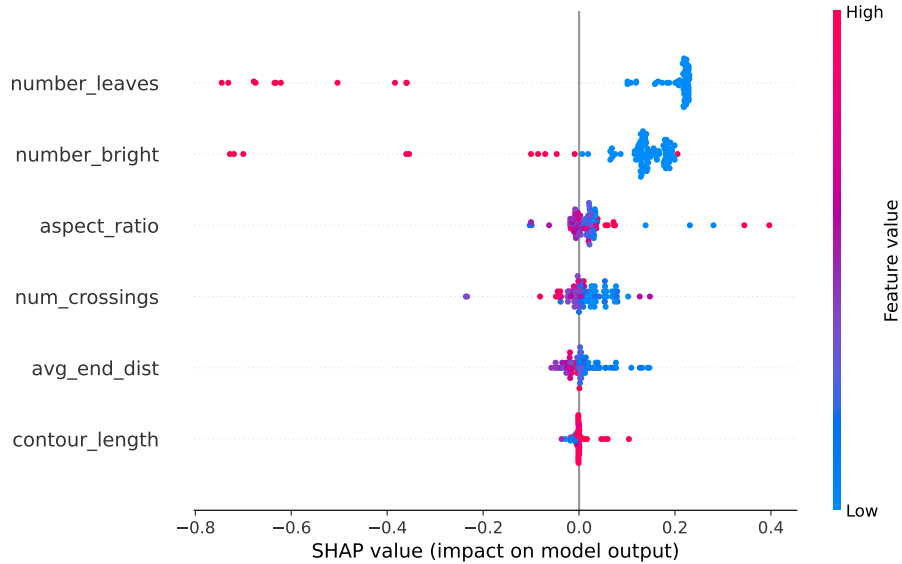


Figure 7.6: SHAP values of features for the label “No” (no R-loops). Each dot represents an entry in the validation set. The colour of a dot represents its numerical value, not its importance. The feature at the top has the most impact. The feature at the bottom has the least impact.

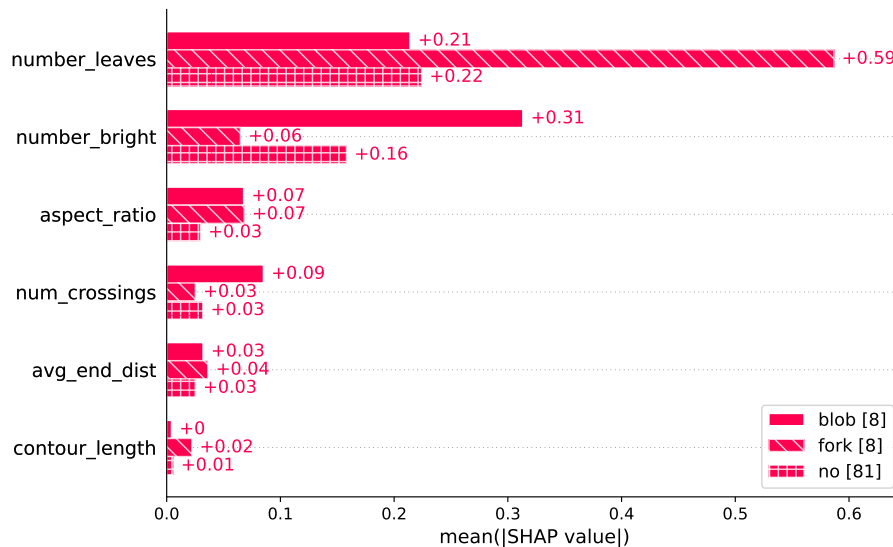


Figure 7.7: Mean absolute of SHAP values across different features and classes for the label “No” (no R-loops). A feature with a larger mean absolute SHAP value will have more influence over the final decision. The feature at the top has the most impact. The feature at the bottom has the least impact.

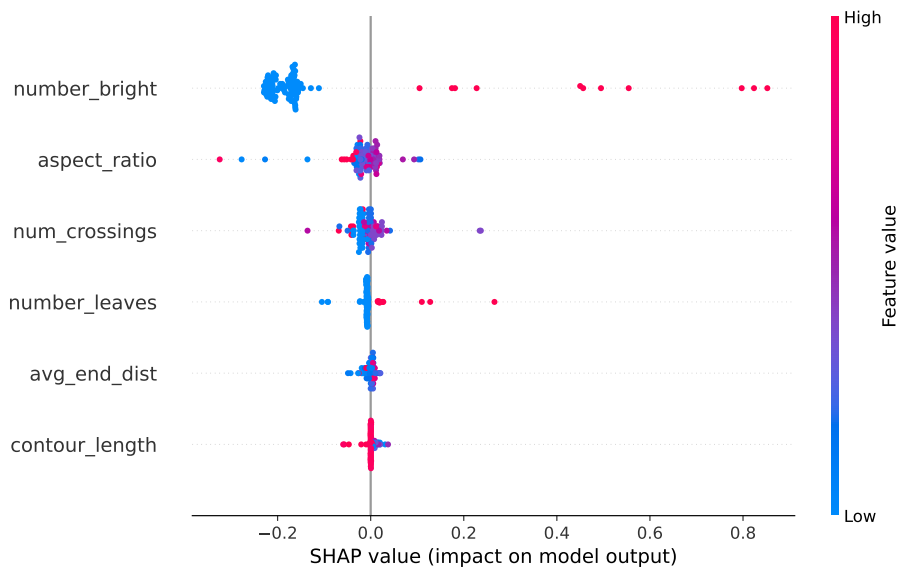


Figure 7.8: SHAP values of features for the label “Blob”. Each dot represents an entry in the validation set. The colour of a dot represents its numerical value, not its importance. The feature at the top has the most impact. The feature at the bottom has the least impact.

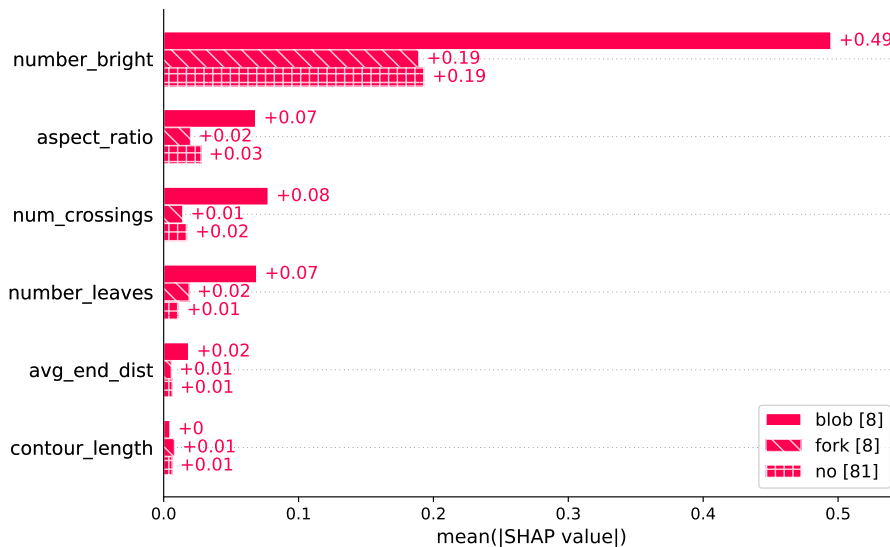


Figure 7.9: Mean absolute of SHAP values across different features and classes for the label “Blob”. A feature with a larger mean absolute SHAP value will have more influence over the final decision. The feature at the top has the most impact. The feature at the bottom has the least impact.

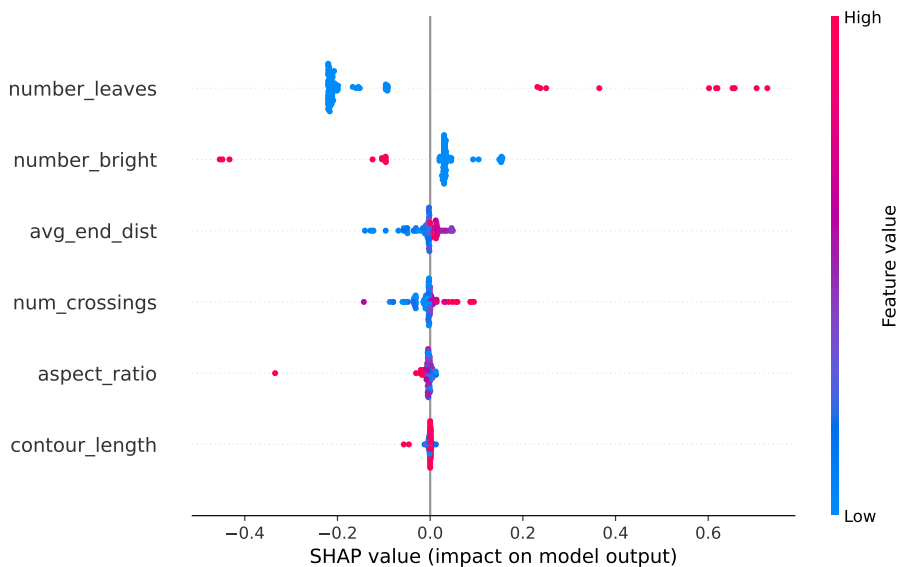


Figure 7.10: SHAP values of features for the label “Fork”. Each dot represents an entry in the validation set. The colour of a dot represents its numerical value, not its importance. The feature at the top has the most impact. The feature at the bottom has the least impact.

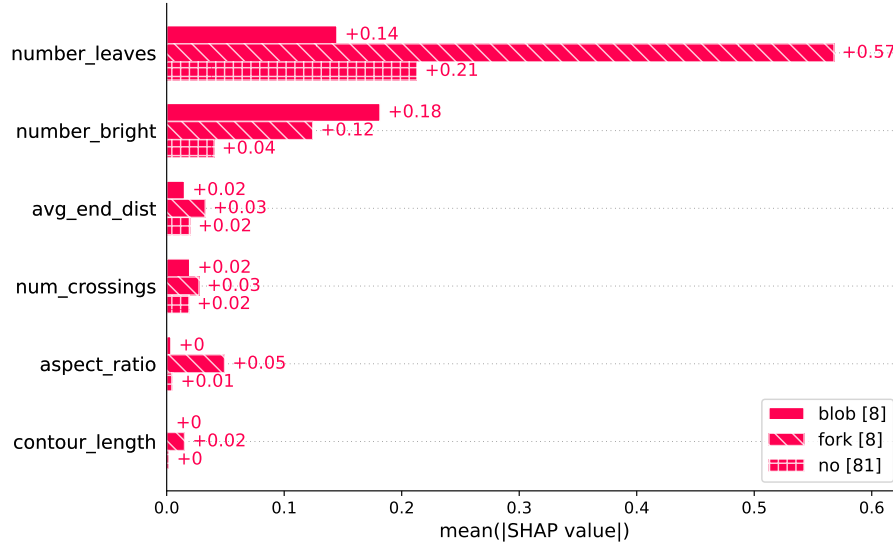


Figure 7.11: Mean absolute of SHAP values across different features and classes for the label “Fork”. A feature with a larger mean absolute SHAP value will have more influence over the final decision. The feature at the top has the most impact. The feature at the bottom has the least impact.

Looking at the Figures 7.6 and 7.7, we see that the number of leaves and bright spots are the most important features for the label “No”. If these two features’ value are low, the SHAP value is positive, which means that the entry is more likely to contain no R-loop. Similarly, if these two features’ value are high, then the entry has a higher chance to be an R-loop because the SHAP value is negative. Having a lower aspect ratio, number of junctions, and end-to-end distance leads to a positive SHAP value, but these features contribute less compared to the first two. The contribution from the total contour length feature is negligible.

For blobs, the number of bright spots and aspect ratio are the most impactful features, as seen in Figures 7.8 and 7.9. We observe that entries with more bright spots and higher aspect ratio have positive SHAP values, which means that the model is more likely to classify them as blobs. We should note that some of the entries with a high number of leaves have positive SHAP values, even though this is not a typical characteristic of blobs.

In Figures 7.10 and 7.11, we observe that the number of leaves and bright spots

are the most important features for the label “Fork”. Entries with a higher number of leaves and fewer bright spots have positive SHAP values. We also note that the model considers entries with higher average end-to-end distance and a high number of junction points to be forks. Having a high aspect ratio leads to a negative SHAP value, which reduces the likelihood that the entry in question is a fork.

The information we obtained from SHAP values should not be surprising as they match what was previously found by our decision tree. Both machine learning algorithms agree on the determining features of both blobs and forks.

8

Conclusion and future directions

We have developed two computational pipelines for identifying and classifying R-loops in AFM images. Both pipelines managed to achieve reasonably accurate results, but the machine learning pipeline performed better overall. Many of the false positives triggered by the deterministic pipeline were properly handled in the machine learning pipeline.

Our decision tree results indicate that blobs tend to have a higher aspect ratio, lower total contour length, fewer crossing points, at least 1 bright spot, and they generally have only 2 leaf points. On the other hand, forks tend to have a higher contour length, lower aspect ratio, higher number of crossing points, at least 3 leaf points, and they generally do not have a bright spot. Although these criteria hold the majority of the time, both classes have outliers.

Despite the notable performance of both pipelines, neither is perfect. The deterministic pipeline struggles to deal with cases that go out of the norm, and the machine learning pipeline suffers from a lack of available data. Although we tested training our machine learning algorithms on different subsets of the dataset and validated that they can still predict the validation set, they will benefit from being trained and tested on a larger set of data. Due to the small size of our dataset, the

results we presented might be susceptible to high variance. Although we randomly pick entries in the training set each time, this does not ensure that all entries are used for training at least once. We intend to perform k -fold cross-validation in the future to deal with this issue.

Neither pipeline can consistently predict R-loops classified as loops. Obvious cases can be detected, but the majority of loops are not traced properly. All of our attempts at properly detecting loops have led to a substantial increase in false positives in forks and blobs. One of the future goals of this study is to find a way around this problem and bring both pipelines to a state where they can accurately predict all three classes.

Another limitation of our work is that the machine learning pipeline is limited by the quality of molecule tracings of TopoStats and our deterministic pipeline. This is because all the features that our machine learning algorithms use are obtained from those two sources. Although the machine learning pipeline performs well despite these sub-optimal circumstances, it is more desirable not to have these limitations. Therefore, another future goal of this study is to develop a machine learning pipeline that can directly work on AFM images, such as one that makes use of Mask R-CNN (He et al. 2017), instead of relying on features obtained from other programs.

References

- Aguilera, Andrés, and Tatiana García-Muse. 2012. “R loops: from transcription byproducts to threats to genome stability.” *Molecular Cell* 46 (2): 115–124.
- Aljaaf, Ahmed J, Dhiya Al-Jumeily, Abir Jaafar Hussain, T Dawson, P Fergus, and M Al-Jumaily. 2015. “Predicting the likelihood of heart failure with a multi level risk assessment using decision tree.” In *2015 third international conference on technological advances in electrical, electronics and computer engineering (TAECE)*, 101–106. IEEE.
- Allison, David F, and Gang Greg Wang. 2019. “R-loops: formation, function, and relevance to cell stress.” *Cell stress* 3 (2): 38.
- Bellili, Abdel, Michel Gilloux, and Patrick Gallinari. 2003. “An MLP-SVM combination architecture for offline handwritten digit recognition: Reduction of recognition errors by Support Vector Machines rejection mechanisms.” *Document Analysis and Recognition* 5 (4): 244–252.
- Belotserkovskii, Boris P, Silvia Tornaletti, Alicia D D’Souza, and Philip C Hanawalt. 2018. “R-loop generation during transcription: Formation, processing and cellular outcomes.” *DNA Repair* 71:69–81.

- Beton, Joseph G, Robert Moorehead, Luzie Helfmann, Robert Gray, Bart W Hoogenboom, Agnel Praveen Joseph, Maya Topf, and Alice LB Pyne. 2021. “TopoStats—A program for automated tracing of biomolecules from AFM images.” *Methods* 193:68–79.
- Bhatia, Vaibhav, Sonia I Barroso, María L García-Rubio, Emanuela Tumini, Emilia Herrera-Moyano, and Andrés Aguilera. 2014. “BRCA2 prevents R-loop accumulation and associates with TREX-2 mRNA export factor PCID2.” *Nature* 511 (7509): 362–365.
- Bhattacharjee, Debotosh, Dipak K Basu, Mita Nasipuri, and Mohantapash Kundu. 2010. “Human face recognition using fuzzy multilayer perceptron.” *Soft Computing* 14 (6): 559–570.
- Binnig, Gerd, Calvin F Quate, and Ch Gerber. 1986. “Atomic force microscope.” *Physical Review Letters* 56 (9): 930.
- Blackford, Andrew N, and Stephen P Jackson. 2017. “ATM, ATR, and DNA-PK: the trinity at the heart of the DNA damage response.” *Molecular Cell* 66 (6): 801–817.
- Boughrara, Hayet, Mohamed Chtourou, and Chokri Ben Amar. 2012. “MLP neural network based face recognition system using constructive training algorithm.” In *2012 international conference on multimedia computing and systems*, 233–238. IEEE.
- Buades, Antoni, Bartomeu Coll, and Jean-Michel Morel. 2011. “Non-local means denoising.” *Image Processing on Line* 1:208–212.

- Carrasco-Salas, Yeraldinne, Amélie Malapert, Shaheen Sulthana, Bastien Molcrette, Léa Chazot-Franguiadakis, Pascal Bernard, Frédéric Chédin, Cendrine Faivre-Moskalenko, and Vincent Vanoosthuyse. 2019. “The extruded non-template strand determines the architecture of R-loops.” *Nucleic acids research* 47 (13): 6783–6795.
- Castellano-Pozo, Maikel, José M Santos-Pereira, Ana G Rondón, Sonia Barroso, Eloisa Andújar, Mónica Pérez-Alegre, Tatiana García-Muse, and Andrés Aguilera. 2013. “R loops are linked to histone H3 S10 phosphorylation and chromatin condensation.” *Molecular Cell* 52 (4): 583–590.
- Castillo-Guzman, Daisy, and Frédéric Chédin. 2021. “Defining R-loop classes and their contributions to genome instability.” *DNA repair* 106:103182.
- Ceccaldi, Raphael, Beatrice Rondinelli, and Alan D D’Andrea. 2016. “Repair pathway choices and consequences at the double-strand break.” *Trends in Cell Biology* 26 (1): 52–64.
- Chang, Emily Yun-Chia, Carolina A Novoa, Maria J Aristizabal, Yan Coulombe, Romulo Segovia, Richa Chaturvedi, Yaoqing Shen, Christelle Keong, Annie S Tam, Steven JM Jones, et al. 2017. “RECQ-like helicases Sgs1 and BLM regulate R-loop-associated genome instability.” *Journal of Cell Biology* 216 (12): 3991–4005.
- Chawla, Nitesh V, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. 2002. “SMOTE: synthetic minority over-sampling technique.” *Journal of Artificial Intelligence Research* 16:321–357.
- Chen, Ying-Jiun C, Evangelia Koutelou, and Sharon YR Dent. 2022. “Now open: Evolving insights to the roles of lysine acetylation in chromatin organization and function.” *Molecular Cell* 82 (4): 716–727.

- Cloutier, Sara C, Siwen Wang, Wai Kit Ma, Nadra Al Husini, Zuzer Dhoondia, Athar Ansari, Pete E Pascuzzi, and Elizabeth J Tran. 2016. “Regulated formation of lncRNA-DNA hybrids enables faster transcriptional induction and environmental adaptation.” *Molecular Cell* 61 (3): 393–404.
- Costantino, Lorenzo, and Douglas Koshland. 2018. “Genome-wide map of R-loop-induced damage reveals how a subset of R-loops contributes to genomic instability.” *Molecular Cell* 71 (4): 487–497.
- Duquette, Michelle L, Priya Handa, Jack A Vincent, Andrew F Taylor, and Nancy Maizels. 2004. “Intracellular transcription of G-rich DNAs induces formation of G-loops, novel structures containing G4 DNA.” *Genes & development* 18 (13): 1618–1629.
- Ester, Martin, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. “A density-based algorithm for discovering clusters in large spatial databases with noise.” In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, 226–231. AAAI Press.
- Ferrari, Margherita Maria, Svetlana Poznanović, Manda Riehl, Jacob Lusk, Stella Hartono, Georgina Gonzalez-Isunza, Frédéric Chédin, Mariel Vázquez, and Nataša Jonoska. 2025. “The R-loop Grammar predicts R-loop formation under different topological constraints.” *PLOS Computational Biology* 21 (8): e1013376.
- García-Muse, Tatiana, and Andrés Aguilera. 2016. “Transcription–replication conflicts: how they occur and how they are resolved.” *Nature reviews Molecular Cell Biology* 17 (9): 553–563.
- Ghiasi, Mohammad M, Sohrab Zendehboudi, and Ali Asghar Mohsenipour. 2020. “Decision tree-based diagnosis of coronary artery disease: CART model.” *Computer Methods and Programs in Biomedicine* 192:105400.

- Ginno, Paul A, Paul L Lott, Holly C Christensen, Ian Korf, and Frédéric Chédin. 2012. “R-loop formation is a distinctive characteristic of unmethylated human CpG island promoters.” *Molecular Cell* 45 (6): 814–825.
- Gorthi, Aparna, July Carolina Romero, Eva Loranc, Lin Cao, Liesl A Lawrence, Elicia Goodale, Amanda Balboni Iniguez, Xavier Bernard, V Pragathi Masamsetti, Sydney Roston, et al. 2018. “EWS–FLI1 increases transcription to cause R-loops and block BRCA1 repair in Ewing sarcoma.” *Nature* 555 (7696): 387–391.
- Guha, Abhijit, Debabrata Samanta, Amit Banerjee, and Daksh Agarwal. 2021. “A deep learning model for information loss prevention from multi-page digital documents.” *IEEE Access* 9:80451–80465.
- Guilliam, Thomas A, and Aidan J Doherty. 2017. “PrimPol—prime time to reprime.” *Genes* 8 (1): 20.
- Hamperl, Stephan, Michael J Bocek, Joshua C Saldivar, Tomek Swigut, and Karlene A Cimprich. 2017. “Transcription-replication conflict orientation modulates R-loop levels and activates distinct DNA damage responses.” *Cell* 170 (4): 774–786.
- He, Kaiming, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. 2017. “Mask r-cnn.” In *Proceedings of the IEEE international conference on computer vision*, 2961–2969.
- Hegazy, Youssef A, Chrishan M Fernando, and Elizabeth J Tran. 2020. “The balancing act of R-loop biology: The good, the bad, and the ugly.” *Journal of Biological Chemistry* 295 (4): 905–913.

- Holleman, Ethan, Thomas E Catley, Tadas Sereiva, Stella R Hartono, Alice LB Pyne, and Frédéric Chédin. 2025. “Protein-mediated stabilization and nicking of the nontemplate DNA strand dramatically affect R-loop formation in vitro.” *Proceedings of the National Academy of Sciences* 122 (38): e2509309122 is available under Creative Commons Attribution License 4.0 (CC BY).
- Hou, Yayan, Chunyong Zhang, Ling Liu, Ying Yu, Lei Shi, and Yan Qin. 2024. “WDR 61 ablation triggers R-loop accumulation and suppresses breast cancer progression.” *The FEBS Journal* 291 (15): 3417–3431.
- Hu, Jiyun, Zetong Xing, Hongbing Yang, Yongli Zhou, Liufei Guo, Xianhong Zhang, Longsheng Xu, Qiong Liu, Jing Ye, Xiaoming Zhong, et al. 2024. “Deep learning-enhanced R-loop prediction provides mechanistic implications for repeat expansion diseases.” *Iscience* 27 (8).
- Hu, Zhibin, Jiachun Wang, Yan Yan, Peilin Zhao, Jian Chen, and Jin Huang. 2021. “Neural graph personalized ranking for Top-N Recommendation.” *Knowledge-Based Systems* 213:106426.
- Işık, Yusuf, İlker Kefe, and Jale Sağlar. 2023. “Detection of fraudulent transactions using artificial neural networks and decision tree methods.” *Business & Management Studies: An International Journal* 11 (2): 451–467.
- Jenjaroenpun, Piroon, Thidathip Wongsurawat, Surya Pavan Yenamandra, and Vladimir A Kuznetsov. 2015. “QmRLFS-finder: a model, web server and stand-alone tool for prediction and analysis of R-loop forming sequences.” *Nucleic Acids Research* 43 (W1): W527–W534.
- Keskin, Havva, Ying Shen, Fei Huang, Mikir Patel, Taehwan Yang, Katie Ashley, Alexander V Mazin, and Francesca Storici. 2014. “Transcript-RNA-templated DNA recombination and repair.” *Nature* 515 (7527): 436–439.

- Kingma, Diederik P, and Jimmy Ba. 2014. “Adam: A method for stochastic optimization.” *arXiv preprint arXiv:1412.6980*.
- Kulis, Marta, and Manel Esteller. 2010. “DNA methylation and cancer.” *Advances in Genetics* 70:27–56.
- Lee, Daniel Y, and David A Clayton. 1996. “Properties of a primer RNA-DNA hybrid at the mouse mitochondrial DNA leading-strand origin of replication.” *Journal of Biological Chemistry* 271 (39): 24262–24269.
- Li, Kuan, Zhenzhen Wu, Jincong Zhou, Wei Xu, Ling Li, Chao Liu, Wei Li, Chengjun Zhang, and Qianwen Sun. 2023. “R-loopAtlas: an integrated R-loop resource from 254 plant species sustained by a deep-learning-based tool.” *Molecular Plant* 16 (3): 493–496.
- Lieber, Michael R. 2010. “The mechanism of double-strand DNA break repair by the nonhomologous DNA end-joining pathway.” *Annual Review of Biochemistry* 79 (1): 181–211.
- Liu, Dong C, and Jorge Nocedal. 1989. “On the limited memory BFGS method for large scale optimization.” *Mathematical Programming* 45 (1): 503–528.
- Liu, Pengyu, Jacob Lusk, Nataša Jonoska, and Mariel Vázquez. 2024. “Tree polynomials identify a link between co-transcriptional R-loops and nascent RNA folding.” *PLOS Computational Biology* 20 (12): e1012669.
- Loomis, Erick W, Lionel A Sanz, Frédéric Chédin, and Paul J Hagerman. 2014. “Transcription-associated R-loop formation across the human FMR1 CGG-repeat region.” *PLOS Genetics* 10 (4): e1004294.

- Lu, Wei-Ting, Ben R Hawley, George L Skalka, Robert A Baldock, Ewan M Smith, Aldo S Bader, Michal Malewicz, Felicity Z Watts, Ania Wilczynska, and Martin Bushell. 2018. “Drosha drives the formation of DNA: RNA hybrids around DNA break sites to facilitate DNA repair.” *Nature communications* 9 (1): 532.
- Lundberg, Scott M, and Su-In Lee. 2017. “A unified approach to interpreting model predictions.” *Advances in Neural Information Processing Systems* 30.
- Mazina, Olga M, Srinivas Somarowthu, Lyudmila Y Kadyrova, Andrey G Baranovskiy, Tahir H Tahirov, Farid A Kadyrov, and Alexander V Mazin. 2020. “Replication protein A binds RNA and promotes R-loop formation.” *Journal of Biological Chemistry* 295 (41): 14203–14213.
- Miller, Henry E, Daniel Montemayor, Janet Li, Simon A Levy, Roshan Pawar, Stella Hartono, Kumar Sharma, Bess Frost, Frederic Chedin, and Alexander JR Bishop. 2023. “Exploration and analysis of R-loop mapping data with RLBase.” *Nucleic acids research* 51 (D1): D1129–D1137.
- Nečas, David, and Petr Klapetek. 2012. “Gwyddion: an open-source software for SPM data analysis.” *Central European Journal of Physics* 10 (1): 181–188.
- Niehrs, Christof, and Brian Luke. 2020. “Regulatory R-loops as facilitators of gene expression and genome stability.” *Nature reviews Molecular Cell Biology* 21 (3): 167–178.
- Ohle, Corina, Rafael Tesorero, Géza Schermann, Nikolay Dobrev, Irmgard Sinning, and Tamás Fischer. 2016. “Transient RNA-DNA hybrids are required for efficient double-strand break repair.” *Cell* 167 (4): 1001–1013.
- Pal, Sankar K, Sushmita Mitra, et al. 1992. “Multilayer perceptron, fuzzy sets, and classification.” *IEEE Transactions on Neural Networks* 3 (5): 683–697.

- Powell, Weston T, Rochelle L Coulson, Michael L Gonzales, Florence K Crary, Spencer S Wong, Sarrita Adams, Robert A Ach, Peter Tsang, Nazumi Alice Yamada, Dag H Yasui, et al. 2013. “R-loop formation at Snord116 mediates topotecan inhibition of Ube3a-antisense and allele-specific chromatin decondensation.” *Proceedings of the National Academy of Sciences* 110 (34): 13938–13943.
- Robbins, Herbert, and Sutton Monro. 1951. “A stochastic approximation method.” *The annals of mathematical statistics*, 400–407.
- Roy, Deepankar, and Michael R Lieber. 2009. “G clustering is important for the initiation of transcription-induced R-loops in vitro, whereas high G density without clustering is sufficient thereafter.” *Molecular and cellular biology* 29 (11): 3124–3133.
- Roy, Deepankar, Zheng Zhang, Zhengfei Lu, Chih-Lin Hsieh, and Michael R Lieber. 2010. “Competition between the RNA transcript and the nontemplate DNA strand during R-loop formation in vitro: a nick can serve as a strong R-loop initiation site.” *Molecular and Cellular Biology* 30 (1): 146–159.
- Sahin, Y, and Ekrem Duman. 2011. “Detecting credit card fraud by decision trees and support vector machines.” In *Proceedings of the international multiconference of engineers and computer scientists*, 1:1–6.
- Saldivar, Joshua C, David Cortez, and Karlene A Cimprich. 2017. “The essential kinase ATR: ensuring faithful duplication of a challenging genome.” *Nature Reviews Molecular Cell Biology* 18 (10): 622–636.
- San Martin Alonso, Marta, and Sylvie M Noordermeer. 2021. “Untangling the crosstalk between BRCA1 and R-loops during DNA repair.” *Nucleic Acids Research* 49 (9): 4848–4863.

- Santos-Pereira, José M, and Andrés Aguilera. 2015. “R loops: new modulators of genome dynamics and function.” *Nature Reviews Genetics* 16 (10): 583–597.
- Shapley, Lloyd S. 1953. “A Value for n-Person Games.” In *Contributions to the Theory of Games II*, edited by Harold W. Kuhn and Albert W. Tucker, 307–317. Princeton: Princeton University Press.
- Shouman, Mai, Tim Turner, and Rob Stocker. 2011. “Using Decision Tree for Diagnosing Heart Disease Patients.” In *Australasian Data Mining Conference*, 121:23–30. Australian Computer Society.
- So, Clare C, and Alberto Martin. 2019. “DSB structure impacts DNA recombination leading to class switching and chromosomal translocations in human B cells.” *PLOS Genetics* 15 (4): e1008101.
- Stein, Hans, and Peter Hausen. 1969. “Enzyme from calf thymus degrading the RNA moiety of DNA-RNA Hybrids: effect on DNA-dependent RNA polymerase.” *Science* 166 (3903): 393–395.
- Stolz, Robert, Shaheen Sulthana, Stella R Hartono, Maika Malig, Craig J Benham, and Frederic Chedin. 2019. “Interplay between DNA sequence and negative superhelicity drives R-loop structures.” *Proceedings of the National Academy of Sciences* 116 (13): 6260–6269.
- Stork, Caroline Townsend, Michael Bocek, Madzia P Crossley, Julie Sollier, Lionel A Sanz, Frederic Chedin, Tomek Swigut, and Karlene A Cimprich. 2016. “Co-transcriptional R-loops are the main cause of estrogen-induced DNA damage.” *Elife* 5:e17548.

- Subasi, Abdulhamit, Sara Alzahrani, Afnan Aljuhani, and Maha Aljedani. 2018. “Comparison of decision tree algorithms for spam e-mail filtering.” In *2018 1st International Conference on Computer Applications & Information Security (ICCAIS)*, 1–5. IEEE.
- Suzuki, Satoshi, et al. 1985. “Topological structural analysis of digitized binary images by border following.” *Computer Vision, Graphics, and Image Processing* 30 (1): 32–46.
- Tedeschi, Frank A, Sara C Cloutier, Elizabeth J Tran, and Eckhard Jankowsky. 2018. “The DEAD-box protein Dbp2p is linked to noncoding RNAs, the helicase Sen1p, and R-loops.” *Rna* 24 (12): 1693–1705.
- Thomassey, Sébastien, and Antonio Fiordaliso. 2006. “A hybrid sales forecasting system based on clustering and decision trees.” *Decision Support Systems* 42 (1): 408–421.
- Wells, James P, Justin White, and Peter C Stirling. 2019. “R loops and their composite cancer connections.” *Trends in Cancer* 5 (10): 619–631.
- Wongsurawat, Thidathip, Piroon Jenjaroenpun, Chee Keong Kwoh, and Vladimir Kuznetsov. 2012. “Quantitative model of R-loop forming structures reveals a novel level of RNA–DNA interactome complexity.” *Nucleic acids research* 40 (2): e16–e16.
- Zhou, Hong, Le Li, Qing Wang, Yan Hu, Weiwei Zhao, Mayank Gautam, and Lijia Li. 2020. “H3K9 demethylation-induced R-loop accumulation is linked to disorganized nucleoli.” *Frontiers in Genetics* 11:43.
- Zhu, Min, Xinyu Wang, Hongchang Zhao, and Zhenjie Wang. 2025. “Update on R-loops in genomic integrity: Formation, functions, and implications for human diseases.” *Genes & Diseases* 12 (4): 101401.