Simulating and Generating pre-miRNA using Variational Auto-Encoders

Petković, Marko

Award date:
2022

Technische Universiteit
Eindhoven
University of Technology

Department of Mathematics and Computer Science
Data Mining Research Group

Simulating and Generating pre-miRNA using Variational Auto-Encoders

Master Thesis

Marko Petkovic

Supervisors:
dr. Vlado Menkovski
prof. dr. Anna Vilanova
prof. dr. Sofia Calero

Version 1.0

Eindhoven, November 2022

Abstract

The genome of all organisms contains the code for all the processes used by a living organism, to develop and survive. A part of the genome directly encodes proteins, which is referred to as coding RNA, while the other part, which does not directly encode proteins, is called non-coding RNA. Some of these non-coding RNAs are called micro RNAs (miRNA). These RNA sequences regulate gene expression by prohibiting the translation of messenger RNA (mRNA) into proteins. MiRNAs can be associated with diseases such as cancer, cardiovascular and neurological diseases and serve as biomarkers for them. Next to this, they can also be used in personalized medicine. Therefore, identifying the entire genome of miRNA can be of great relevance.

As experimental methods for novel miRNA detection are complex and expensive, computational methods for miRNA prediction have been developed. There is no clear way to recognize which RNAs are part of the miRNA genome, and which are not. Since manually detecting patterns which discriminate miRNA from other RNA is difficult, a solution is the use of Machine Learning (ML). In these data-driven approaches, a predictive model can be created by looking at examples of miRNA and non-miRNA. Even though these models can be highly accurate, they are commonly black boxes, which makes it difficult to explain their predictions.

In this thesis, we are the first to our knowledge to use Deep Generative Models to model miRNA. We propose a novel framework, in which we use Machine Learning in an interpretable way to potentially develop a description of miRNA. In this framework, we utilize the generative factors of the data obtained from the latent representation developed by the model. We use Variational Auto-Encoders (VAE) to model the data, and achieve excellent reconstruction performance, as well as an organized and partitioned latent space. We also present how we can use the model to sample new RNA strands by conditioning on the secondary structure of the RNA.

In our novel framework, we use Decision Trees on the latent space to develop an miRNA description. Splits in the Decision Tree are made by making separations in the latent space using a secondary classifier, based on predefined biological features of each RNA. The explainable framework obtained an accuracy of 91.2%, and made splits based on features which are known the be relevant to miRNA classification. In turn, this confirmed that the framework is successful and could potentially be used in different domains.

Preface

Back in February, at the beginning of 2022 I started writing this thesis. At first, I was making very little progress in modelling pre-miRNAs, due to the complexity of creating a proper model for this problem. This made the master project (and the year) last very long and be quite tough. However, once the model started working, the results were coming in, and I had the opportunity to learn lots of interesting things about generative models and micro RNAs. Working on this research topic broadened my horizon, and inspired me to pursue my career in research, by applying for a PhD.

I want to thank my supervisor Vlado for the guidance he has given me to solve this problem, as well as the discussions in which we talked about potential solutions. I would also like to thank him, Anna and Sofia for taking the time to grade and evaluate my master project. Lastly, I want to further thank Vlado and Sofia for offering me a PhD position in Machine Learning-based Simulation for Materials Discovery.

In addition, I am also grateful for the love and support my girlfriend Diana has provided me over the course of this project. I would especially like to thank her for pointing out where I was missing commas in this thesis.

Finally, I also wish to thank my parents Milan and Marija, and my siblings Miki and Marina. They provided me with lots of support and fun times while working on this thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In living organisms, DNA encodes all the information used by the organism to develop and survive. To make use of the information encoded in the DNA, it needs to be translated to RNA by one of the RNA polymerase enzymes. A part of the DNA encodes proteins, which becomes messenger RNA (mRNA) after translation. Then, when the mRNA reaches a ribosome, the protein is assembled. In this process, transfer RNA (tRNA) transfers the amino acids needed to assemble the protein, while the assembly of the protein is hosted by the ribosomal RNA (rRNA).

Sometimes it is desirable that the expression of certain mRNA is halted. In this process, micro RNA (miRNA) plays a big role. MiRNA is involved in the development of various parts of an organism, by inhibiting "unnecessary" mRNA. However, overexpression of certain miRNA can also lead to various diseases. Therefore, to better understand disease prevention and treatment, mapping the entire miRNA genome can be of great use.

MiRNA is a form of non-coding RNA, with a length usually between 18 to 24 nucleotides. MiRNA starts out from the transcription of primary miRNA (pri-miRNA), following which it is processed by enzymes into precursor miRNA (pre-miRNA), from which mature miRNA is processed by another enzyme [49]. The mature miRNA is then incorporated in the RNA Induced Silencing Complex (RISC). This complex then binds to a target mRNA, which is done by the miRNA being (partially) complement to the mRNA. Following this, RISC prevents the translation of the mRNA through degradation or cleavage.

Gene silencing done by miRNA affects various parts of an organism's development. It affects the development of the nervous system, muscles, blood vessel formation and apoptosis [67]. If expression of certain miRNAs is affected, for example by gene mutation, multiple diseases can result, such as cancer, neural and cardiovascular diseases [68]. Since some of these miRNAs appear in the serum of the blood, they could be used as biomarkers for these diseases [27, 32, 73]. Therefore, miRNA can be used to identify potential diseases in patients, and allow for preventive interventions. In addition, miRNA analysis can be used to predict a patient's reaction to certain drugs [3], which can be used for personalized medicine. Furthermore, miRNA can be used as a drug by itself, by amplifying or suppressing the effect of certain miRNAs. Overall, understanding the properties of the different miRNA can be of great value to medicine.

Currently, around 2654 mature human miRNAs are documented, of which only 26% are classified as mature miRNA with high confidence [43]. Analysis suggests [1] that the actual number of human miRNA is around 2300. Finding these new miRNAs can be done through experimental detection. The three main methods used for miRNA detection use Real-Time quantitative PCR (qRT-PCR), microarrays and Next-Generation Sequencing (NGS) [15]. These methods are considered reliable at finding miRNAs and their targets. However, performing these experiments often requires complex lab conditions and intensive work [48, 55]. This is made more difficult by

the experiments being error prone, as well as the quick degradation of miRNA.

On the other hand, computational methods can also be used for detection of novel miRNA. In these computational methods, the targets of prediction are pre-miRNAs since these contain more features to distinguish them from non pre-miRNAs. Almost all pre-miRNAs contain a hairpin structure [75] and a substrate for the dicer enzyme [37]. Therefore, in computational miRNA detection methods, the structure of the pre-miRNA can be used for prediction of novel miRNA strands, as it contains more information than only the miRNA itself. The structure of the pre-miRNA consists of the primary and secondary structure. Here, the primary structure is the order of the nucleotides in the pre-miRNA, while the secondary structure explains the shape of the pre-miRNA due to the bonds of the opposing nucleotides.

Some of these computational methods rely on handcrafted rules about known properties of a miRNA, which makes it hard to find potential novel miRNAs with new features. Other computational methods rely on data-driven approaches and use Machine Learning (ML) [56] for making classifications. These methods use handcrafted features from the primary and secondary structure of pre-miRNA. Since these methods use handcrafted features derived from previous knowledge, they might still suffer in performance when it comes to detecting novel miRNAs. More recently, Deep Learning (DL) approaches for pre-miRNA prediction have been proposed which make use of the entire primary and secondary structure of the miRNA. This way, DL models learn a representation of the data, and figure out by themselves which features are relevant. Some of these methods make use of Convolutional Neural Networks (CNN) [14, 17, 79], Recurrent Neural Networks (RNN) [50] or both [65] for classifying pre-miRNA. In [14], an image encoding algorithm for the hairpin structure of the pre-miRNA was proposed, which encodes the primary and secondary structure of the RNA in an image (see Section 5.1). Later, using concept whitening, limited interpretability was introduced in the model [69].

Most ML approaches for detecting pre-miRNA rely on discriminative models. Given the data and its class label, discriminative models are forced to learn to assign the correct label to each datapoint. In general, these models try to learn the decision boundaries between classes. Thus, they do not explicitly learn a representation of the data. On the other hand, generative models usually develop a representation of the data, by learning the generative factors of the data. Using a representation developed by a generative model, we can not only make label predictions for new datapoints, but also make descriptions of (non) pre-miRNA data, which we can use to make interpretations.

Mapping the entire human miRNA genome can be of great benefit for understanding diseases as well as improving personalized medicine. The complexity of current experimental methods for detection of pre-miRNA has led to the development of computational methods for their detection. ML-based computational methods often rely on handcrafted features or have limited explainability due to their complexity. In this thesis, we aim to develop a framework using a generative model, that will enable making a precise description of what a miRNA is.

1.1 Problem Formulation

In this work, we aim to develop a framework which can be used for generating interpretable descriptions of what constitutes a precursor micro RNA. An RNA with up to 200 nucleotides in length can be represented by an $\mathbb{B}^{200\times4}$ matrix, where each nucleotide is one-hot encoded. The dataspace of all possible RNAs between 40 and 200 nucleotide long would contain 3.44×10^{120} different combinations. To develop a useful description of the pre-miRNA, we first need to develop a representation of the genomic data, which we can then use for making explanations. The representation can be made by reducing the dimensionality of the data, by mapping the RNA data to a latent space of z latent variables ($\mathbb{R}^z$) using a generative model. By enriching the latent

representation of the data with biologically relevant features, we can create linear separations in the latent space based on these features. Using these separations, we can provide a description of what a miRNA is. An overview can be found in Figure 1.1.

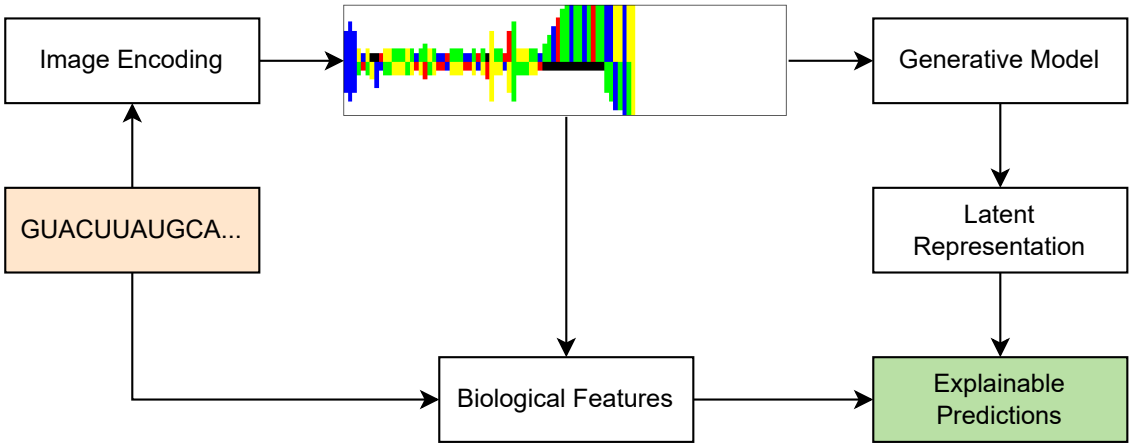


Figure 1.1: Framework for creating pre-miRNA descriptions. First, the nucleotide sequence is encoded to an image by folding the sequence and applying the image encoding algorithm. Then, the image is processed by a generative model to obtain a latent representation of the RNA. Using the sequence and image encoding, biological properties of the RNA are calculated. By using the biological features to create separations in the latent space, we can develop descriptions of miRNA using the explainable predictions made by the framework.

To create a framework for generating pre-miRNA descriptions, we will define the following research question:

Can we train a generative model to uncover the generative factors of pre-miRNA which we can use to understand what differentiates pre-miRNAs from other similar RNAs?

To answer our main research question, we will look at multiple subquestions.

1. *Can we train a Variational Autoencoder (VAE) which achieves good performance when modelling pre-miRNA?*
   The encoder-decoder architecture of VAEs allows us to learn a relevant latent space, which should model the generative factors of the pre-miRNA data well. In addition, there have been multiple extensions to the VAE architecture which can be exploited for a better organized space, which can help understanding the generative factors. We will aim to obtain a Mean Absolute Error (MAE) of less than 0.01 (1%) for reconstructions, and we want to achieve an organized latent space with regards to already established biological pre-miRNA features.
2. *Is it possible to sample potential novel pre-miRNA from the latent space of the generative model?*
   Using the decoder of a VAE, we can create new samples of (non) pre-miRNA. By introducing conditionality in the VAE, in addition to generating random samples, we can specifically try to create new samples conditional on class or other features of interest.
3. *Can we shape (a part of) the latent space with regards to the shape of the RNA molecules?*
   By conditioning the latent space on the Minimum Free Energy (MFE) of an RNA, we can (partially) control the shape of reconstructed RNAs. As many currently known biological features which differ across pre-miRNA and non pre-miRNA are related to the shape of the molecules [69], such a latent space could be used for making descriptions of pre-miRNA.

4. *How can we make use of the latent space of the generative model to make a definition of what constitutes a miRNA?*
   By organizing the latent space well, we can determine which biological features are of relevance for pre-miRNA classification. By applying domain knowledge, we can label each datapoint with biological features. Due to the organization of the latent space, import features should have a simple distribution over the latent space. Using these simple relationships, a Decision Tree can be built to develop a description for pre-miRNA.

1.2 Contribution

The contribution of this work is threefold: (i) we create a VAE which models pre-miRNA, (ii) we propose a method for reconstructing the highly structured pre-miRNA image encoding and (iii) we provide a framework for defining a rule-based definition of pre-miRNA, which can also be generalized to other domains.

To our knowledge, no other work has yet modeled miRNA using a VAE. For our first contribution, we will make use of VAEs to model pre-miRNA, using the image encoding from [14]. In addition to this, we also organize the latent space with regards to class (i.e. pre-miRNA or non pre-miRNA) and shape of the RNA molecule, using the Minimum Free Energy. This way, we are able to conditionally sample new strands of potential novel pre-miRNA while also conditioning on their shape.

For our second contribution, we will propose a way to reconstruct the pre-miRNA image encoding. Unlike in other image datasets, there is a strong vertical dependence between pixels in the RNA images. In the RNA image encoding, each nucleotide is represented by a bar, meaning that there should be full dependence between the color of pixels within a bar. To ensure that the reconstructions of RNA images respect this property, we propose a custom decoder for our model. The decoder will contain two separate modules to make reconstructions. The first module is used for reconstructing the shape of the molecule through the length of the bars, and the second module is used to reconstruct the color of each bar. Finally, the outputs of both modules are multiplied to obtain a reconstructed image.

Next to this, we will use our trained VAE to create a framework for making a rule-based definition of pre-miRNA. By using the regularized latent space, we can find meaningful generative factors of pre-miRNA. To use this framework, a domain expert can define features of pre-miRNA, such as the presence of a terminal loop, or the existence of certain combinations of nucleotides. Then, we train classifiers to find simple relationships of biological features in the latent space, which aim to maximize the difference between pre-miRNA and non pre-miRNA. This way, we create a Decision Tree which can provide a rule-based definition of what constitutes a pre-miRNA. In addition to the domain of pre-miRNA, this approach could also be generalized to make rule-based definitions for objects from different domains.

1.3 Outline

In Chapter 2, we will discuss the relevance of miRNAs further, as well as their biogenesis and experimental detection. In Chapter 3, we will introduce Machine Learning, and explain how different algorithms and neural networks work. Towards the end of the chapter, we will delve deeper into deep generative models, with the main focus on Variational Auto-Encoders and their variants. Following this, in Chapter 4, we will look at different existing interpretability approaches for Machine Learning algorithms and neural networks in Section 4.1. Then, we will look at computational pre-miRNA detection methods in Section 4.2, and how interpretability has been applied in

these methods.

In Chapter 5, we will cover the data that will be used for modelling pre-miRNA, as well as the model architectures we will use. Here, we will also introduce our framework for making explainable predictions using the latent space. We will look at the different performances of our models and explainable predictions method in Chapter 6. Finally, we will present a concluding summary in Chapter 7, where we will also discuss limitations of the current work and potential future experiments.

Chapter 2

micro RNA

2.1 RNA interference

Many RNAs (like tRNA and rRNA) are not directly coding any proteins, and it is estimated that in eukaryotes (organisms whose cells have a nucleus) around 98% of the RNA genome is non-coding [64]. Part of these non-coding RNA's are involved in RNA interference (RNAi). The main two types of RNA involved in RNAi are miRNA and small interfering RNA (siRNA). Both miRNA and siRNA are involved in the regulation of post-transcriptional gene expression. Both types form a RISC with Argonaute proteins, in which the siRNA or miRNA guides the RISC to a complementary mRNA. In the case of siRNA, the mRNA is fully complementary, meaning the siRNA's often have a single target. After binding, the RISC cleaves the mRNA, after which it is recognised as a foreign object by the cell. On the other hand, miRNA is often partially complementary to the target mRNA, meaning a single miRNA can have multiple targets. This also results in the regulation not only occurring to cleavage, but also through translational repression and degradation.

2.2 MiRNomics

The field which studies miRNAs has been named *mirnomics*. To better understand the role of miRNA and the relevance of pre-miRNA to the computational prediction task, we will first take a look at the biogenesis of miRNA.

2.2.1 Biogenesis

There exist several pathways for the biogenesis of miRNA [20]. The canonical pathway consists of the transcription of the DNA into a pri-miRNA, following operations of different enzymes to obtain a mature miRNA. For the alternative pathways, the miRNA is not directly transcribed from the gene, but mainly from introns from other RNAs [20]. Then, they are processed by different combinations of the enzymes used in the canonical pathway. A simple visual overview of this process can be found in Figure 2.1.

Canonical Pathway

The canonical biogenenis of miRNA starts with the transcription of pri-miRNA by RNA polymerase. Following the transcription, the pri-miRNA is cleaved by a microprocessor complex, from which pre-miRNA can be created. This complex consists of the Drosha enzyme and the DGCR8 binding protein (which is known as Pasha in invertebrates). The pre-miRNA is then transported from the nucleus to the cytoplasm by Exportin 5. The pre-miRNA contains a hairpin loop and is usually between 80 and 200 nucleotides long. Next to this, the pre-miRNA contains a substrate

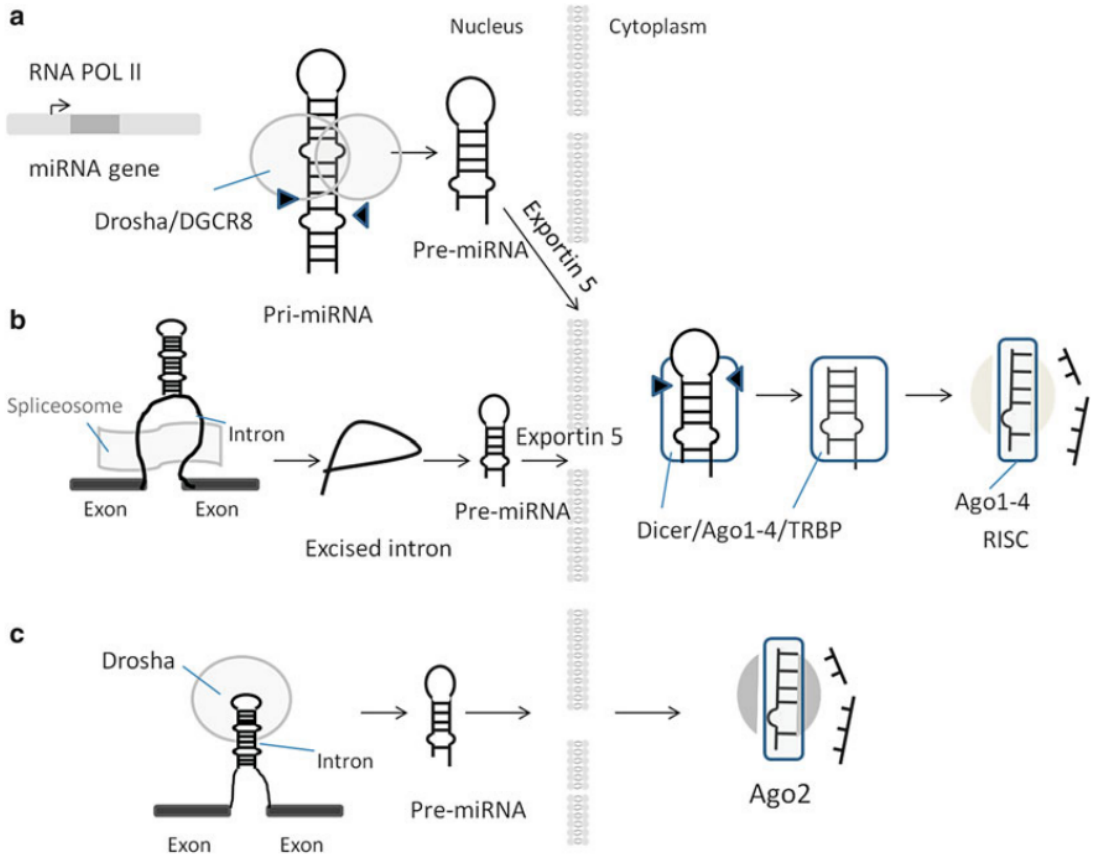


Figure 2.1: Different pathways for the biogenesis of miRNA [20]. Pathway **a** is the canonical pathway, while pathways **b** and **c** are non-canonical.

for the Dicer enzyme. The Dicer enzyme binds to this substrate, and cleaves the terminal loop of the pre-miRNA. This leaves a double-stranded RNA (dsRNA). Of this dsRNA, usually one of the strands is degraded, which leaves a mature miRNA. The miRNA then becomes part of the RNA-Induced Silencing Complex (RISC), in which it binds with Argonaute proteins. The miRNA is then used in the RISC for targeting mRNA.

Alternative Pathways

Next to the canonical pathway, multiple alternative pathways for miRNA biogenesis have been found. In one of these pathways, the pre-miRNA is produced from introns (mirtrons) of mRNA by splicing [54]. This pathway does not make use of microprocessor complex consisting of Drosha and DGCR8, but follows the canonical pathway after splicing the pre-miRNA from the intron. Other pre-miRNA are produced from short-hairpin RNA (shRNA), which is cleaved by Drosha and DGCR8 [76]. However, this pre-miRNA is not cleaved by Dicer due to being too short for a Dicer substrate, and instead needs Argonaute proteins to finish maturation into miRNA.

2.2.2 Structure of (pre-)miRNA

In contrast to pre-miRNA, a mature miRNA is typically around 22 nucleotides long, and is a single, not folded, strand. In Figure 2.2, the structure of an artificially designed pre-miRNA is shown [2]. In each pre-miRNA, it is guaranteed that there is a terminal loop present, however, this can vary in size. Furthermore, in pre-miRNA, the stem of the molecule often contains mainly

strong bonds (A-U and C-G), and sometimes wobbles (G-U). Whenever there are no strong bonds between molecules, it can result in a (asymmetric) bulge, where nucleotides go more "outward" as they are not pulling each other. Finally, at the end of an RNA molecule, the nucleotides do not always have bonds.

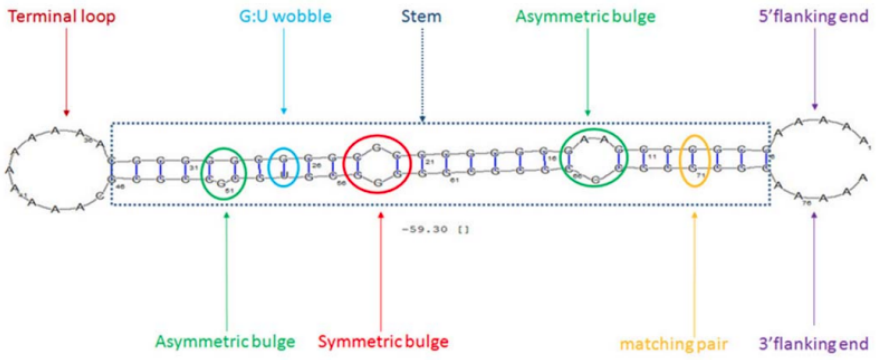


Figure 2.2: Artificial pre-miRNA strand [2], labeled with different properties which can be present in (non) pre-miRNA.

2.2.3 Function

The main function of miRNA is post-transcriptional gene regulation, by binding to mRNA to inhibit translation. When the miRNA is fully complementary to the target mRNA, cleavage by the Argonaute proteins in the RISC is induced [20]. In this case, each miRNA has one specific mRNA target, which is usually the case in plants. In animals, the miRNA is often not fully complementary to the target mRNA [49]. This disallows cleavage of the mRNA by the Argonaute proteins. Instead of this, translational inhibition is achieved through mRNA degradation. Due to the miRNA not being fully complementary to the mRNA, a single miRNA can have multiple mRNA targets. In addition, there can be multiple variants of a single miRNA (isomiRs), which makes prediction of the targets of a miRNA difficult [20].

Next to their canonical function, several other functions of miRNA have been found. One of these functions is RNA activation [20]. The miRNA can target promoter elements of protein-coding genes. For example, they can enrich the RNA polymerase binding. It is also suggested that miRNA is involved in post- and co-transcriptional gene regulation in the nucleus [49], through interaction of RISC and mRNA.

The role of miRNA of post-transcriptional gene regulation is important for multiple processes in an organism. They play a role in nervous system development through repressing non-neuronal gene activity [67]. In muscle development, miRNA plays a role in myotube formation [67]. Furthermore, miRNA plays an important role in the cell cycle, by inhibiting cell cycle regulators [67]. Next to this, miRNAs also have a role in cell signaling, apopstosis and autophagy [67].

As a result of mutations, it is possible that a miRNA cannot be produced anymore, a miRNA cannot bind to its target mRNAs anymore, or new target mRNA cannot be created [68]. For example, failure of miRNAs involved in the cell cycle can result in cancer development. Furthermore, failure can also lead to autoimmune diseases such as rheumatoid arthritis and multiple sclerosis [22], as well as other neurological diseases and cardiovascular diseases [68].

MiRNAs were also found to be present in the serum of blood [68]. Some of the miRNAs circulating in the blood have been associated with different diseases [27, 32, 73], which in turn means these miRNAs could potentially be used as biomarkers.

2.2.4 Detection

To detect miRNAs, several experimental methods exist. These methods include high throughput systems such as Real-Time quantitive PCR (qRT-PCR), microarrays and Next-Generation Sequencing (NGS). Of these techniques, qRT-PCR and microarrays can be used to identify already known miRNA, but fail to detect novel miRNAs. On the other hand, NGS can detect novel miRNAs, and is better at distinguishing similar miRNAs. A serious drawback of these methods is that they often take a long time, starting from under 6 hours for qRT-PCT, to up to 2 weeks for NGS [15]. In addition, these methods often require complex lab conditions and intensive work and can still be error prone [48, 55], making the experiments costly to perform. To improve performance, these methods can be combined with computational methods for miRNA prediction (see 4.2).

Chapter 3

Machine Learning

In Machine Learning (ML), the main goal is to develop algorithms which can learn patterns in data by themselves, without human input. Within ML, there are three basic categories: supervised learning, unsupervised learning and reinforcement learning. The reinforcement learning category is concerned with learning intelligent agents to take actions based on their environment, which will lead to the highest future reward. In supervised learning, the ML algorithm is fed with data $\mathbf{x}$ and its label $\mathbf{y}$. Then, the algorithm is trained to learn a map f between the input data $\mathbf{x}$ and the label $\mathbf{y}$. On the other hand, in unsupervised learning, the ML algorithm only takes $\mathbf{x}$ as input. The ML algorithm then finds similarities in the data, for example through clustering or learning the probability density function of the data.

In addition to the distinction between supervised and unsupervised learning, there is also a distinction between generative and discriminative ML methods. The main difference between these two types are the tasks which the models are trained to perform. Discriminative models try to directly model the conditional distribution of the label given the data, i.e. $p(\mathbf{y}|\mathbf{x})$. This leads to the models learning decision boundaries between the data classes which separate the data in the best possible ways. Generative models learn the generative factors of the data through the joint probability distribution $p(\mathbf{x}, \mathbf{y})$. Although a generative model does not directly learn $p(\mathbf{y}|\mathbf{x})$, it is still possible to classify new observations $\mathbf{x}$. This is done by computing the marginal probability distribution $p(\mathbf{x})$ and then calculating the conditional probability distribution $p(\mathbf{y}|\mathbf{x})$ by doing $p(\mathbf{x}, \mathbf{y})/p(\mathbf{x})$.

3.1 ML Algorithms

3.1.1 Linear Regression

In linear regression, the goal is to model a dependent variable $\mathbf{y}$ against independent variable(s) X. The formula of the fitted model is $\mathbf{y} = X\beta + \epsilon$, where X is the data (with the first column being 1 in case an intercept is fitted), β the coefficients estimated by the model and ϵ the residual variance. The main method used to fit a linear model on the data is Ordinary Least Squares (OLS). In OLS, we aim to minimize the squared difference between each observation and the prediction according to the model ($||\mathbf{y} - X\beta||^2$). There exist variants of linear regression which introduce bias to the model to improve generalizability. For example, ridge regression and lasso regression do this by adding the $l1$-norm and $l2$-norm respectively to the penalty term.

3.1.2 Logistic Regression

Logistic regression makes use of the logistic function, which is defined as $\sigma(t) = \frac{1}{1-e^{-t}}$. In the simplest case, where we want to predict the probability of two classes/events, the logarithmic odds

(log-odds) of the two events are modeled by a linear combination of the dependent variables. In the logistic function, t represents these log-odds, which are estimated by $X\beta$. To fit a Logistic Regression model, Maximum Likelihood Estimation (MLE) is used. Unlike in OLS, there is no closed form solution available, and iterative methods are required for fitting the model. Logistic Regression can also be extended to predict probabilities for more than two events, as well as having different types of (categorical) variables, which are ordinal or nominal.

3.1.3 Support Vector Machine

Support Vector Machines (SVM) [13] are used for binary classification, by constructing hyperlanes that separate the data. We can define the hyperplane which is separating the data as $\mathbf{w^T x} - b = 0$. In addition to separating the two classes, SVM also aims to find a hyperplane which maximizes the margin between datapoints of the two classes. In case perfect separation is possible, any point which lies on or above $\mathbf{w^T x} - b = 1$ is classified as the positive class, while any point on or below $\mathbf{w^T x} - b = -1$ is classified as the negative class. To calculate the optimal hyperplane, we can rewrite the hyperplane equation into $y_i \mathbf{w^T x} - b \geq 1$. By minimizing $||\mathbf{w}||^2$, we will maximize the width of the hyperplanes separating the classes, and we will obtain the parameters $\mathbf{w}$ and b.

However, it is often the case that the datapoints are not perfectly separable. In this case, we optimize the hinge loss instead. The loss is calculated for each wrongly classified datapoint using the distance to the hyperplane. Thus, we calculate the loss as the maximum between 0 and $1 - y_i(\mathbf{w^T x} - b)$. In addition to minimizing this loss, $\lambda||\mathbf{w}||$ is also minimized, where λ is the weight to balance the two losses.

Along with the data not being perfectly separable, in many cases the ideal hyperplane is not linear. To alleviate this problem, the kernel trick can be used. Using the kernel trick, the data is mapped to a higher dimension, in which it is possible to find a linear separator. When minimizing the loss of an SVM, the dot product of each pair of datapoints is calculated to solve the Lagrangian dual of the loss. When using the kernel trick, the dot product is replaced with the kernel function. Common kernels include the polynomial kernel $\big(k(x_1, x_2) = (x_1 \cdot x_2)^d\big)$ and the Gaussian radial basis function $\big(k(x_1, x_2) = \exp(-\gamma||x_1 - x_2||^2)\big)$.

3.1.4 Naive Bayes

Naive Bayes is a simple, probabilistic classifier, which makes strong assumptions of independence between the features in the data. The classifier is a generative model, as it models the joint probability of the class and the features. The classifier makes predictions by factorizing the joint probability into the probability of the class, as well as the probability of each feature given the class. To estimate the probabilities for each feature given the class, a Gaussian distribution can be used to estimate the probabilities when dealing with continuous features, a multinomial distribution can be used for features representing counts and a Bernoulli distribution can be used to represent binary features. Using Naive Bayes for classification can be summed up with the following formula: $\hat{y} = \underset{k \in \{1,2...K\}}{\operatorname{argmax}} \; p(C_k) \prod_{i=1}^{n} p(x_i|C_k)$. In this formula, C_k is the k-th class and x_i is the i-th feature of an observation.

3.1.5 Decision Tree

In Decision Tree Learning, the goal is to train a Decision Tree to perform classification or regression. To use a Decision Tree to classify a new observation, we walk down the nodes of the tree until we reach a leaf. At each node, the model checks whether the observation satisfies the condition of the node, based on which the next node is visited. When a leaf is reached, the class of the leaf is assigned as the predicted class of the observation.

Decision Trees are trained by learning the rules for each node, based on the information gain of each rule. Usually, the information gain is calculated using entropy (C4.5 [52]) or the gini-index (CART [7]). Both measures calculate the impurity of the data at a node, where having only 1 class at a node is considered pure, and having a (uniform) mixture of classes is considered impure. When checking which split to make, each variable is tested, and the rule resulting in the highest information gain of the child nodes is picked. In case of continuous variables, thresholds are used to check for different possible splits. The Decision Tree stops making splits after no information gain is possible, the data is pure at the node, or if a maximum depth is reached. Once the splitting is finished, each leaf node is assigned the majority class at the node. Then, the tree is pruned by recursively removing sibling leaf nodes which predict the same class.

3.1.6 Random Forest

Random Forests [6] are ensemble Machine Learning algorithms, which consist of multiple Decision Trees. To classify an observation, it is ran through all the Decision Trees in the Random Forest, and the final class prediction is obtained through a majority vote. When training each of the individual trees in the Random Forest, two types of bagging are used. First, a random subset with the size of the training set is sampled with replacement from the original training set, ensuring that the dataset is different for each tree. In turn, this lowers the variance of the overall model, despite individual trees not being very accurate. Then, a subset of features is selected for each tree, thus disallowing them to split on all features. In this way, features which are strong predictors of the output are not used in every tree, lowering the correlation between the trees.

3.2 Deep Learning

Deep Learning is a field of Machine Learning which is concerned with representation learning using Artificial Neural Networks (ANN) with multiple layers. In DL, Deep Neural Networks (DNN) automatically learn feature maps from the raw data, by stacking multiple layers. DNNs can be applied to a variety of problems, such as computer vision, natural language processing and speech recognition.

ANNs are inspired by biological neural networks, which constitute animal brains. Each neuron (or node) in an ANN is inspired by how neurons in an animal brain work. Each neuron takes its input features ($\mathbf{x}$) and applies a linear transformation $\mathbf{z} = \mathbf{w^T x} + b$ to it. In order to be able to learn non-linear relationships in the data, an activation function $a(\mathbf{z})$ is used. An overview of common activation functions and the functions used in this thesis can be found in Table 3.1.

The sigmoid activation function is similar to logistic regression, mapping the output between 0 and 1, whereas the tangent hyperbolic is a "stretched" version of the sigmoid function, mapping the output between -1 and 1. Calculating the gradient of these functions at very high or low values of $\mathbf{z}$ will always result in a virtually non-existent gradient. During the training of ANNs, this can

Function	Function($\mathbf{z}$)
Sigmoid	$\frac{1}{1+e^{-\mathbf{z}}}$
Tangent Hyperbolic	$\frac{e^{\mathbf{z}}-e^{-\mathbf{z}}}{e^{\mathbf{z}}+e^{-\mathbf{z}}}$
ReLU	$\max(0, \mathbf{z})$
Leaky ReLU	$\max(\alpha\mathbf{z}, \mathbf{z})$
ELU	$\begin{cases} \mathbf{z}, & \text{if } \mathbf{z} > 0 \\ \alpha(e^{\mathbf{z}} - 1), & \text{otherwise} \end{cases}$
Softplus	$\ln(1 + e^{\mathbf{z}})$

Table 3.1: Overview of different activation functions used in neural networks.

lead to the vanishing gradient problem.

The ReLU function is the identity functions for positive values of $\mathbf{z}$, while it is 0 for negative values. In turn, this means that the gradient of the function is always 1 for positive values and 0 for negative values, thus solving the vanishing gradient problem. Other variations of the ReLU function also exist. The Leaky ReLU function allows for a small gradient to exist when the input value is negative, while the Softplus function is a smooth approximation of the ReLU function. Finally, the ELU function is a smoothed version of the shifted ReLU ($\max(-\alpha, \mathbf{z})$).

Depending on what the problem is, there are multiple options as to how to represent the output of an ANN. In case of a regression problem, where the value of one or multiple features needs to be predicted, we can use one neuron per output feature. In this case, the activation function can be the identity or the ReLU function, depending on the range of values the output features take. In case of binary classification, a single neuron with the sigmoid function can be used. The output can be interpreted as the probability of the positive class. In classification with any number of classes, the softmax function can be used, with the the amount of output neurons equal to the amount of classes. The softmax function is defined as $p(y = j|z) = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$, where z_k is the output of the neuron associated with the k-th class, and z_j is the output of the neuron associated with class j. A network which uses the softmax function for the output, essentially provides probabilities for each class when predicting an observation.

The simplest type of ANN is a fully connected network. Such a network has multiple layers of neurons, where each neuron takes the outputs of all previous neurons as input. In case we would have a network with 2 hidden layers h_0 and h_1, and output layer o, the network would make predictions as follows: $\hat{y} = o(h_1(h_0(\mathbf{x})))$.

3.2.1 Training

Since there is no closed form solution for optimizing these complex networks, an iterative approach called Stochastic Gradient Descent (SGD) is used. In gradient descent, the loss is calculated after passing a batch of datapoints through the network. This loss can be categorical cross-entropy in case of classification, or Mean Squared Error (MSE) in case of regression. Then, the gradient of the loss is calculated. To update the weights we make use of backpropagation. The gradient is factorised over all the operations in the neural network using the chain rule. All parameters are then updated by taking a step down the gradient, of which the size is determined by the learning rate. Once the entire dataset (in batches) has been passed through the network and the weights have been updated using backpropagation, an epoch has been completed. Then, the datapoints are shuffled to create new batches and the process is repeated, until the network has converged.

Multiple extensions to the basic SGD optimizer exist. Momentum can be added to the update of weights, to prevent the network getting stuck in local minima. Other optimizers like AdaGrad [18] and RMSProp [66] have an adaptive learning rate. In AdaGrad, parameters which received larger updates in the past receive a smaller learning rate, while parameters with smaller updates get bigger learning rates. However, this can lead to some parameters not getting updated, which is solved by introducing a decay in RMSProp. The Adam [38] optimizer combines both optimizers, and also keeps track of a cumulative history of gradients. Typically, these algorithms will achieve a faster convergence than SGD, but they will not necessarily achieve better results.

3.2.2 Convolutional Neural Networks

Convolutional Neural Networks (CNN) are used in many computer vision tasks for analyzing images. Using fully connected networks will quickly prove to be a problem, as, while image size increases, the amount of input variables also quickly increases. For example, an RGB image of 300 by 300 pixels has 270,000 input parameters (300 pixels * 300 pixels * 3 channels). In addition, fully

connected layers also struggle to learn relevant features, as each node takes the entire previous layer as input. For example, a node in the first hidden layer will receive both the pixel from the top left and bottom right corner, while these pixels will very often have no relationship. On the contrary, pixels close together in images often have meaningful relationships, as they represent a part of the image.

To find these local relationships in images, a CNN makes use of layers containing filters (kernels). A filter is usually of rectangular shape and has the same amount of channels as the input it receives. It convolves over the input image, and detects local features. Since one kernel only has input channels * kernel width * kernel height weights, this drastically reduces the amount of parameters a layer has. In a single convolutional layer multiple kernels can be present, and the amount of output channels is decided based on the amount of kernels used. After applying one layer of convolutions, a feature map of the input image is obtained, to which more convolutional layers can be applied. Following a convolution, usually the ReLU activation function is used, to reduce vanishing gradients. By applying multiple layers of convolutions, we obtain feature maps representing more abstract features in each consequent layer.

Next to the kernel size, a convolutional layer also has other parameters. One of these is padding, which is used to control the output size of the layer. Padding is done by adding zeros (other numbers/patterns are also possible) around the outside of the image, before being processed by a kernel. In case we want the output to be the same size as the input, we make use of "same" padding. Another way of controlling the output size is using the stride of kernels. Stride controls by how many pixels the kernel moves as it is convolving over an image. Dilation can be used to make the network more memory efficient. It effectively increases the size of the kernel, while the same amount of pixels are processed as usual, effectively adding spacing between kernel elements. To calculate the output shapes of a layer, equation 3.1 is used. In this equation, S_{in} and S_{out} are the width or height of the input and output, D is the amount of dilation, K is the kernel size in that direction, P the amount of pixels of padding applied, and S the stride.

$$S_{out} = \frac{S_{in} - D * (K - 1) + 2P}{S} + 1 \tag{3.1}$$

Next to padding, stride and dilation, pooling layers can also be used to reduce the size of the processed data. For a pooling operation, a kernel is specified and performs an operation over a single channel. Usually, the stride of this kernel is equal to the kernel size so no overlap is present, but the stride can also have a lower value to introduce some overlap. The most commonly used pooling operations are max-pooling and average-pooling.

In most CNN architectures, blocks consisting of convolutions, pooling layers and other regularization layers are stacked on top of each other. Here, the output of lower layers typically has less channels, but a bigger image size. Deeper in the network, the size gets reduced, while the amount of channels increases, allowing the network to learn more spatial patterns of the input image. After applying all convolutional blocks, the output is flattened. Following the flattening, a fully connected layer is used to transform the data to the correct output shape, which can be used for classification and regression.

CNNs are still prone to overfitting. One common technique used to reduce overfitting and improve generalization is dropout [30]. When dropout is applied to a layer during training, each node in that layer has a set probability of being deactivated. This way, the entire network can be seen as an ensemble of multiple networks. Next to this, using dropout does not allow neurons to rely on each other, and improves generalization performance this way.

Other methods for regularizing the network include batch normalization [34] and weight normalization [58]. Both methods aim at speeding up the training process of a neural network, and reducing overfitting. In batch normalization in CNNs, the output of each filter is normalized using

the mean and standard deviation of the filter output. During training, the batch normalization layer keeps tracks of these parameters and updates them the same as normal weights in a neural network. In weight normalization, the weight matrix is parameterized as $\mathbf{w} = \frac{g}{||\mathbf{v}||}\mathbf{v}$. Since the scalar g is equal to the norm of the weights ($||\mathbf{w}||$, the direction of the weight vector is decoupled. When performing gradient descent, the gradient is calculated with respect to both $\mathbf{v}$ and g, speeding up training and introducing regularization. Other regularization methods also exist, such as $l1$ and $l2$ regularization, which effectively limit the weights from becoming too big.

3.2.3 CNN Architectures

Since CNNs have been introduced, various different architectures have been proposed. One of the first well performing architectures were the VGG models [61]. These models consist of multiple blocks, where each block consists of 2 or 3 convolutional layers with a 3 by 3 kernel, followed by pooling.

Another popular architecture is the inception architecture [63]. In the inception architecture, the input of a single block is processed (independently) by multiple convolutional layers of different sizes, after the amount of channels is reduced by 1 by 1 convolutions. At the output of the block, the output of the multiple smaller networks is concatenated, and passed through the next block. Since the release of the first inception architecture, multiple updates have been made. For example, in inception v4 [62], parts of the block which consists of k by k convolutions have been replaced by 2 convolutional layers, with kernels of size k by 1 and 1 by k. This way, the amount of necessary computations is further reduced.

One of the drawbacks of very deep Convolutional Neural Networks is the vanishing gradient problem, where the gradient goes towards 0 at shallow layers in the network when performing backpropagation, which in turn does not allow these layers to learn. To combat this problem, Residual Networks (ResNets) have been introduced [28]. In these networks, skip connections are introduced, meaning that the input of a block gets elementwise summed with the output of the block. Since gradient flows over these skip connections, training them is easier. This allows these networks to have much more layers while still being able to train. It is also possible to combine the inception and ResNet architecture by adding skip connections between blocks, as shown in [62].

3.2.4 Deep Generative Models

While many neural networks focus on classification and regression and are thus discriminative, there also exist several deep generative models. In these models, neural networks are often used for approximating probability distributions. Some of the most popular deep generative models include Generative Adversarial Networks (GAN), Variational Auto-Encoders (VAE) and Normalizing Flow (NF). These models all have different types of latent spaces, where the data is represented in a lower dimensional space. During training, the models aim to learn lower dimensional representations of the data which contain its generative factors.

GANs consist of two networks [24], a generator which generates new samples $G(\mathbf{z})$ from a noise vector $\mathbf{z}$, and a discriminator D which evaluates whether the samples are real or fake. Both networks are trained simultaneously, meaning that the generator is learning to fool the discriminator, which in turn is trying to get better at "catching" the generator. However, since both models are being trained simultaneously, this may lead to a failure called mode collapse. Here, the generator gets stuck in a local minimum, as it only learns to generate a small part of the data. This can happen when at some point during training the discriminator prefers a certain part of the data. In addition, vanishing gradients can also be problem when the discriminator trains too fast. In that case, the gradients in the generator vanish, as the loss will always remain high no matter which step the generator takes. Next to this, GANs do not explicitly model the likelihood function of

the data, like NF or a lower bound of the likelihood in the case of VAEs.

Normalizing Flows [53] is a statistical method, where the change-of-variable law is used to transform a simple distribution, like the standard normal, to a more complex data distribution. These transformations are done using invertible functions f_i, meaning that an inverse function f_i^{-1} exists. These invertible functions are modeled using neural networks. There are two requirements for the neural network used to represent a transformation: the function should be easy to invert, and it should be easy to compute the determinant of its Jacobian. This allows us to calculate the log-likelihood of the target distribution as in equation 3.2. Here, z_0 is the initial simple distribution, z_K the target distribution after K transformations. When training an NF model, the negative log-likelihood is minimized using gradient descent. After training, it is possible to sample new points by first sampling from the simple distribution and then passing the sample through the transforming functions.

$$\log p_K(z_K) = \log p_0(z_0) - \sum_{i=1}^{K} \log \left| \det \frac{df_i(z_{i-1})}{dz_{i-1}} \right| \tag{3.2}$$

3.3 Variational Autoencoders

VAEs [39] try to model the generative factors of the data, through a latent space $\mathbf{z}$. A VAE consists of an encoder which maps the data $\mathbf{x}$ to latent space $\mathbf{z}$ and a decoder which reconstructs the data from the latent space. By using a decoder, the network is forced to learn a lower dimensional representation of the data which contains meaningful generative factors, as otherwise making reconstructions would not be possible.

In most VAEs, the prior distribution of the latent space is assumed to follow a standard normal distribution with a diagonal covariance matrix, meaning that the all the latent variables are independent. To model the latent space from an observation $\mathbf{x}$, the observation is first passed through an encoder $q_\phi(\mathbf{z}|\mathbf{x})$. This encoder tries to approximate the posterior distribution $p(\mathbf{z}|\mathbf{x})$. Then, we sample $\mathbf{z}$ from the approximate posterior distribution obtained from the encoder. Finally, $\mathbf{z}$ is passed through the decoder $p_\theta(\mathbf{x}|\mathbf{z})$ to obtain reconstruction $\hat{\mathbf{x}}$. In a VAE, the encoder and decoder are parameterized with neural networks ϕ and θ.

To train a VAE, we want to maximize the log-likelihood of the data, which is $\log(p(\mathbf{x}))$. However, it is often computationally expensive or impossible to calculate this log-likelihood. One of the parts of the model we want to optimize is approximating $p_\theta(\mathbf{z}|\mathbf{x})$ as well as possible using $q_\phi(\mathbf{z}|\mathbf{x})$. We can define the distance between the two distributions using the Kullback-Leibler Divergence, which can be rewritten as in Equation 3.3.

$$\begin{aligned}
D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}|\mathbf{x})) &= \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}|\mathbf{x})} d\mathbf{z} \\
&= \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{q_\phi(\mathbf{z}|\mathbf{x}) p_\theta(\mathbf{x})}{p_\theta(\mathbf{z},\mathbf{x})} d\mathbf{z} \\
&= \log(p_\theta(\mathbf{x})) + \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z},\mathbf{x})} d\mathbf{z} \\
&= \log(p_\theta(\mathbf{x})) + \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{x}|\mathbf{z}) p_\theta(\mathbf{z})} d\mathbf{z} \\
&= \log(p_\theta(\mathbf{x})) + E_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}(\log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z})} - \log p_\theta(\mathbf{x}|\mathbf{z})) \\
&= \log(p_\theta(\mathbf{x})) + D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z})) - E_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}(\log p_\theta(\mathbf{x}|\mathbf{z}))
\end{aligned} \tag{3.3}$$

Using this formula, we can obtain a lower bound for the log-likelihood (Equation 3.4). Since the KL Divergence is always non-negative, $D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}|\mathbf{x}))$ will always be greater or equal to 0. This function is the Evidence Lower BOund (ELBO), which we aim to maximize when training a VAE. The first term of this loss function is the reconstruction error, while the second term is the regularization term which ensures the learned latent space is close to the prior.

$$\log(p_\theta(\mathbf{x})) - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}|\mathbf{x})) = E_{\mathbf{z}\sim q_\Phi(\mathbf{z}|\mathbf{x})}(\log(p_\theta(\mathbf{x}|\mathbf{z}))) - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z})) \tag{3.4}$$

During training, the encoder and decoder are trained to maximize the ELBO using backpropagation. To enable backpropagation over sampling of the latent space from the approximate posterior, we make use of the reparameterization trick (see Equation 3.5). When using the reparameterization trick, we sample a random variable ϵ from the standard normal distribution. Then, we multiply ϵ with the standard deviation of the posterior σ and add μ, which allows us to sample from the posterior while backpropagation remains possible.

$$\begin{aligned} &\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mu, \sigma) \\ &\epsilon \sim \mathcal{N}(0, \mathbf{I}) \\ &\mathbf{z} = \mu + \sigma \odot \epsilon \end{aligned} \tag{3.5}$$

3.3.1 Beta-VAE

To control the regularization of the latent space, we can introduce a scalar term β, by which the KL-Divergence term of the loss is multiplied [29] (Equation 3.6). By setting β to a (relatively) higher number, the model is forced to learn an efficient latent space, where all latent variables are independent. This results in the model learning a disentangled latent representation.

$$\mathcal{L}_{\theta,\phi} = E_{\mathbf{z}\sim q_\Phi(\mathbf{z}|\mathbf{x})}(\log(p_\theta(\mathbf{x}|\mathbf{z}))) - \beta * D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z})) \tag{3.6}$$

In [8], the authors propose an extension to the β-VAE through an annealing scheme. If the KL term of the loss is optimized, the result is a posterior distribution which is equal to the prior distribution, which therefore will not have any capacity to encode information. Therefore, by slowly increasing the capacity (C) of the latent space, the VAE is expected to first disentangle the most relevant generative factors. To achieve this, the loss function is modified as in Equation 3.7.

$$\mathcal{L}_{\theta,\phi} = E_{\mathbf{z}\sim q_\Phi(\mathbf{z}|\mathbf{x})}(\log(p_\theta(\mathbf{x}|\mathbf{z}))) - \beta * |D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z})) - C| \tag{3.7}$$

3.3.2 DIVA

To improve disentanglement performance, the Domain Invariant Variational Autoencoder (DIVA) was introduced [33]. Contrary to a normal VAE, this method is (semi) supervised, as it takes class labels and the domain of the data into account. In the authors' implementation of the model, the latent space is factorized in three parts: a latent space corresponding with the class ($\mathbf{z}_y$), a latent space corresponding to the domain ($\mathbf{z}_d$) and a latent space for remaining variance ($\mathbf{z}_\mathbf{x}$). Each of the three latent spaces has their own separate encoder to approximate the posterior, which takes the entire data as input. In the example, the dataset used is a rotated MNIST, where the digit represents the class label and the rotation angle represents the domain. Next to this, $\mathbf{z}_y$ and $\mathbf{z}_d$ have conditional priors $p_{\theta_y}(\mathbf{z}_y|y)$ and $p_{\theta_d}(\mathbf{z}_d|d)$, which output parameters for the normal distribution, used for the prior. $\mathbf{z}_\mathbf{x}$ has a standard normal prior. When reconstructing an image, we concatenate all latent spaces, and pass them through a shared decoder $p_\theta(\mathbf{x}|\mathbf{z}_y, \mathbf{z}_\mathbf{x}, \mathbf{z}_d)$. This gives us a loss function as seen in Equation 3.8. The β terms introduced in this equation serve the same

purpose as in the β-VAE, to reduce the capacity of the latent space and improve disentanglement.

$$\begin{aligned}\mathcal{L}_s(y, \mathbf{x}, d) = \; & E_{q_{\phi_y}(\mathbf{z}_y|\mathbf{x}), q_{\phi_x}(\mathbf{z}_\mathbf{x}|\mathbf{x}), q_{\phi_d}(\mathbf{z}_d|\mathbf{x})}[\log(p_\theta(\mathbf{x}|\mathbf{z}_y, \mathbf{z}_\mathbf{x}, \mathbf{z}_d))] \\ & - \beta_y * D_{KL}(q_{\phi_y}(\mathbf{z}_y|\mathbf{x})||p_{\theta_y}(\mathbf{z}_y|y)) \\ & - \beta_x * D_{KL}(q_{\phi_x}(\mathbf{z}_\mathbf{x}|\mathbf{x})||p(\mathbf{z}_x)) \\ & - \beta_d * D_{KL}(q_{\phi_d}(\mathbf{z}_d|\mathbf{x})||p_{\theta_d}(\mathbf{z}_d|d))\end{aligned} \tag{3.8}$$

To further improve disentanglement, on $\mathbf{z}_y$ and $\mathbf{z}_d$ auxiliary classifiers $q_{\omega_y}(y|\mathbf{z}_y)$ and $q_{\omega_d}(d|\mathbf{z}_d)$ are built. This gives us the loss function from Equation 3.9.

$$\mathcal{F}_{\text{DIVA}}(y, \mathbf{x}, d) = \mathcal{L}_s(y, \mathbf{x}, d) + \alpha_y E_{q_{\phi_y}(\mathbf{z}_y|\mathbf{x})}[\log q_{\omega_y}(y|\mathbf{z}_y)] + \alpha_d E_{q_{\phi_d}(\mathbf{z}_d|\mathbf{x})}[\log q_{\omega_d}(d|\mathbf{z}_d)] \tag{3.9}$$

3.3.3 IAF

VAE with Inverse Autoregressive Flow (IAF) is a VAE which makes use of inverse flows to push the posterior distribution closer to the prior [40]. In this architecture, $\mathbf{x}$ is first passed through the encoder, which results in a posterior distribution from which we sample $\mathbf{z_0}$. In addition, the encoder also creates a context $\mathbf{h}$. After this, a chain of $\mathbf{T}$ autoregressive transformations is applied to $\mathbf{z_0}$, which results in the final latent space $\mathbf{z_T}$. Then, $\mathbf{z_T}$ is passed through a decoder, like with a normal VAE. An example of an autoregressive transformation which can be used is Masked Autoencoder for Distribution Estimation (MADE) [23].

A MADE block consists of two hidden fully connected layers, followed by an output layer which is the same size as the input. By applying masks over the connections, the model becomes autoregressive, since the output distribution of the variables can be written as a nested product of the conditionals of each output node (Figure 3.1).

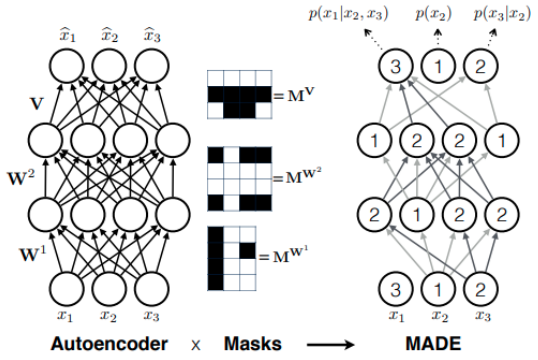


Figure 3.1: Single MADE block [23]. Due to the masked connections, the model is autoregressive.

The $\mathbf{t}$'th autoregressive transformation consists of two seperate MADE blocks which both take as input $\mathbf{z}_{t-1}$ and $\mathbf{h}$. Each block produces one of the two real valued vectors $\mathbf{m}_t$ and $\mathbf{s}_t$. Then, to calculate $\mathbf{z}_t$, we apply the transformation in Equation 3.10.

$$\begin{aligned}\sigma_t &= \text{sigmoid}(\mathbf{s}_t) \\ \mathbf{z}_t &= \sigma_t \odot \mathbf{z}_{t-1} + (1 - \sigma_t) \odot \mathbf{m}_t\end{aligned} \tag{3.10}$$

The autoregressive transformations are an inverse normalizing flow, and due to the nature of the transformations, the Jacobian $\frac{d\mathbf{z}_t}{d\mathbf{z}_{t-1}}$ is triangular with σ_t on its diagonal. Due to the simple

Jacobian, we can easily calculate the change in density as in Equation 3.2. This results in the final density of $q(\mathbf{z_T}|\mathbf{x})$ as shown in Equation 3.11. To calculate the KL-divergence, we update $q_\phi(\mathbf{z}|\mathbf{x})$ with $q(\mathbf{z_T}|\mathbf{x})$.

$$q(\mathbf{z_T}|\mathbf{x}) = q(\mathbf{z_0}|\mathbf{x}) - \sum_{t=0}^{T} \log \sigma_t \tag{3.11}$$

Chapter 4

Related Work

4.1 Interpretability in Machine Learning

When using ML models in real-life applications, interpretability of the models' decision can be of great value, or even required. For example, in the case of automatic cancer detection using CNNs, the model could be used as an aid to the diagnosing doctor. It is important for the doctor to understand why and how the model makes its prediction, to be able to verify whether the model is making an accurate prediction. In this example, the model could show an activation map, by highlighting areas of the image which were important in making the prediction.

4.1.1 Interpretable Models

Some ML methods are naturally interpretable. For example, it can be clearly seen which features differentiate between two or multiple classes. In Decision Trees, the rules the model learns are clearly interpretable. However, in a very deep Decision Tree, the rules can become very convoluted and hard to interpret. Similarly, in case of linear regression and logistic regression, the models can be interpretable with few variables. The coefficients directly indicate the effect of the variable in case of linear regression or indicate the changes of odds ratio for logistic regression. When more variables are added to these models, it can result in the coefficients losing relevance, or the model getting harder to interpret due to its complexity.

4.1.2 Black-Box Models

Other ML methods such as Random Forests, SVMs and neural networks are considered black-box models, and are harder to interpret. Due to the complex structure of these algorithms, it is not intuitive to look at the weights or coefficients of these models to get a clear understanding of how and why a prediction is made.

While a Random Forest consists of Decision Trees, there are usually too many Decision Trees to make explanations. There are methods to make a random forest interpretable. For example, the variable importance [6] can be calculated. This is done by permuting the data for each variable in the training set, and testing how much each variable influences the error, with variables which increase the error being seen as more important. This method tells us how important some variables are for making classifications, but does not directly show how these variables influence the decision of the forest. Other methods for Random Forest explainability attempt to create a single decision tree from the forest, which mimics the behaviour of the forest [57, 71].

When using an SVM with a linear kernel, the decision boundary is interpretable, as it is defined using coefficients for each variable. In most cases, SVMs use a different kernel, which maps the data to a higher dimension. The learned coefficients are not interpretable due to them being

used to create the boundary in a higher dimensional space. It is possible to interpret this bound in case the data has only two dimensions, as in that case it is possible to visually interpret the relationship.

4.1.3 Interpretability in CNNs

Naturally, Deep Neural Networks are not interpretable due to their highly complex structure of multiple layers of neurons. In CNNs, there exist multiple methods for adding interpretability to the model. Some of these methods, such as saliency maps and class activation mapping are applied after training of the network, while other methods such as β-VAE and concept whitening change the architecture of the network or how the network is trained.

Saliency maps [60, 77] highlight which input pixels are most responsible for a layer activation or classification. Class Activation Mapping (CAM) [80] is a similar method, which can be applied to networks where the fully connected layer is replaced by global average pooling over all the channels. The activation maps of the final layer are then multiplied with the weights used for classification, which shows what the network is looking for. An extension to CAM, called Grad-CAM [59] can also be used on networks using a fully connected layer instead of global average pooling.

One of the techniques which change the network itself to make it more interpretable is proposed in [78]. Here, CNNs are made more interpretable by adding an additional loss term. This loss term is the entropy of the activation map of a chosen convolutional filter. Since this entropy is being minimized, the filter is forced to detect only a single feature, in a single place in the image. The authors assume that if the filter is activated in multiple places, the specific filter is detecting a low-level feature such as an edge. By applying this technique, the output of networks can be understood in a similar way as saliency maps.

In the concept whitening technique [11], it is proposed that in addition to the dataset, the neural network is also fed with images representing concepts which are relevant to class prediction. For example, if one of the classes we are predicting are cats, we could add images of the heads, paws and tails of cats as concepts representing this class. To apply these concepts, a fully connected layer in the network is used. In this layer, individual neurons are assigned for learning concepts and additional neurons learn the remaining variance in the data. Each of these neurons is then trained to detect their assigned concepts. Simultaneously, whitening transformations are applied on this layer to disentangle the output. While training, the neural network is fed images from the training dataset as well as the concept dataset. Since each neuron corresponds to a single concept, the network predictions can be interpreted by comparing the influence each neuron has on the prediction.

4.1.4 Interpretability in Deep Generative Models

In the vanilla versions of many deep generative models, such as GANs and VAEs, the latent space learned by these models is often entangled, meaning that, as we change a single variable in the latent space, multiple features of the sampled data change. For example, in a model trained on a dataset with faces, changing a single variable would result in both orientation of the face and hair color changing. To counter this problem, multiple methods for training disentangled deep generative methods have been proposed.

In the InfoGAN [10], the input vector has been decomposed in a noise vector $\mathbf{z}$ and a latent code $\mathbf{c}$, which is responsible for capturing the variation of structured features in the data. To ensure the model learns a relevant latent code, an additional penalty term is added, consisting of the mutual information between the latent code $\mathbf{c}$ and the output of the generator. The authors have shown that the InfoGAN was able to disentangle factors on datasets like MNIST [44] and

3D Faces [51], such as digit type and rotation for MNIST, and azimuth and elevation for 3D Faces.

To make the vanilla VAE more interpretable, a simple modification was proposed in [29], by adding a penalty term β (>1) for the KL-divergence. This forces the VAE to learn an efficient representation of the data. In turn, this leads to the latent factors being independent of each other, meaning that a well trained model has latent factors directly related to a generative factor. Compared to InfoGAN, it has been shown that β-VAE can disentangle even more factors. Furthermore, by controlling the capacity of the latent space, it is possible to disentangle the data, while also keeping high quality reconstructions [8]. By further decomposing the KL-term of the loss, it is possible to additionally penalize the part of the term representing the Total Correlation (TC) between the latent variables [9]. This method is called β-TCVAE, and the authors have shown it has better disentanglement performance compared to β-VAE.

4.2 Computational pre-miRNA Detection

Most methods for computational miRNA prediction use pre-miRNA instead of miRNA. This is due to the pre-miRNA containing more biologically relevant characteristics which can differentiate it from other RNA, such as the terminal loop [75] and a substrate for the dicer enzyme [37].

Many algorithms use both the primary and secondary structure of the pre-miRNA for making predictions. The primary structure consists of the order of nucleotides in the molecule, which can be Adenine (A), Uracil (U), Cytosine (C) and Guanine (G). The secondary structure describes the shape of the RNA molecule, by how it is folded. This is determined by the thermodynamic bonds between the nucleotides [4]. In this structure, opposing C and G nucleotides (base pair) have the strongest thermodynamic bond, followed by A and U (base pair), followed by C and U (wobble). The other potential pairs of nucleotides are mismatches, and have no bonds or weak bonds. When a nucleotide does not have a bond with an opposing nucleotide, it results in a bulge in the molecule.

To computationally obtain the secondary structure of a pre-miRNA, the raw nucleotide sequence is passed through a folding algorithm called RNAFold [31]. This algorithm determines how the RNA molecule can best fold, by minimizing the Minimum Free Energy (MFE). RNAFold indicates which nucleotides have a bond with opposing nucleotides, from which the shape of the molecule can be inferred.

In 2005, [75] developed an SVM based method to distinguish pre-miRNA from pseudo pre-miRNA, which are RNAs with similar characteristics to pre-miRNA. They only consider stem portions of the molecules, and for each nucleotide in the stem, they take note of its bond and the bond of the nucleotides next to it. This gives rise to 32 possible features, which are counted and normalized for each RNA. On this data, an SVM was trained which achieved an accuracy of 90% on human data, as well as correctly identifying 90% of the pre-miRNAs from other species. In [35], they added two features: the MFE value from the RNAFold algorithm and a p-value, indicating whether the MFE is significantly different than a random RNA sequence with similar nucleotides. Both an SVM and RF were trained using these new features, and both algorithms outperformed the SVM from [75], with the RF being slightly better.

Another approach was proposed in [48]. They proposed a feature vector containing 29 variables, such as dinucleotide frequencies and folding measures. The SVM achieved a 94% accuracy on their test set. A similar approach was proposed in [5], where they achieved 90% sensitivity and 97% specificity. Another approach using an SVM [16] used a combination of features from [75] and [48]. The methods achieved a 93% sensitivity and a 96% specificity rate on their test set.

Other ML approaches for pre-miRNA prediction mainly use SVM and RF [56]. These algorithms were combined in [56], to create a good performing ensemble model.

More recently, Deep Learning based approaches for pre-miRNA detection were proposed. In [50], an RNN approach was introduced. The data was one-hot encoded based on the nucleotide and its secondary structure. Then this data was passed through multiple Long Short-Term Memory (LSTM) layers. A different approach was proposed in [17], where CNNs were used to represent the data. In this representation, the image has 9 channels and is of size L by L, where L indicates the length of the RNA in nucleotides. 8 of these channels are used to represent the nucleotide sequence twice, as the nucleotide sequence is one-hot encoded, and is repeated once horizontally and once vertically. The final channel indicates the bonds between nucleotides. Another CNN approach [79] simply takes the primary sequence and one-hot encodes it. A similar method was proposed in [65], where following convolutional layers also recurrent layers were applied.

To improve utilization of the secondary structure by models, a special image encoding algorithm was developed in [14] (see 5.1). Using the images generated from this algorithm, a CNN with state-of-the-art performance was built, outperforming many other ML models on various datasets and achieving an accuracy of 95%. Later, concept whitening was applied to the model, which resulted in a slight accuracy drop (92%). However, using this model, it was possible to make explainable explanations using the defined miRNA concepts, which were the presence of a large asymmetric bulge, and the stem of the RNA containing at least 90% base pairs and wobbles.

Chapter 5

Methods

5.1 Data

To model the (non) pre-RNA, we make use of the *modmiRBase* dataset [12], which consists of image encodings of folded RNA molecules. The RNA sequences used to generate the image encodings are taken from various datasets. The positive pre-miRNA sequences were taken from `mirbase.org` [25] and `mirgenedb.org` [21]. The non pre-miRNA sequences were obtained using various methods. Some negative datasets consist of existing non pre-miRNAs with similar characteristics as pre-miRNA [26, 48, 74], while other datasets consist of shuffled or not optimally folded pre-miRNAs [56].

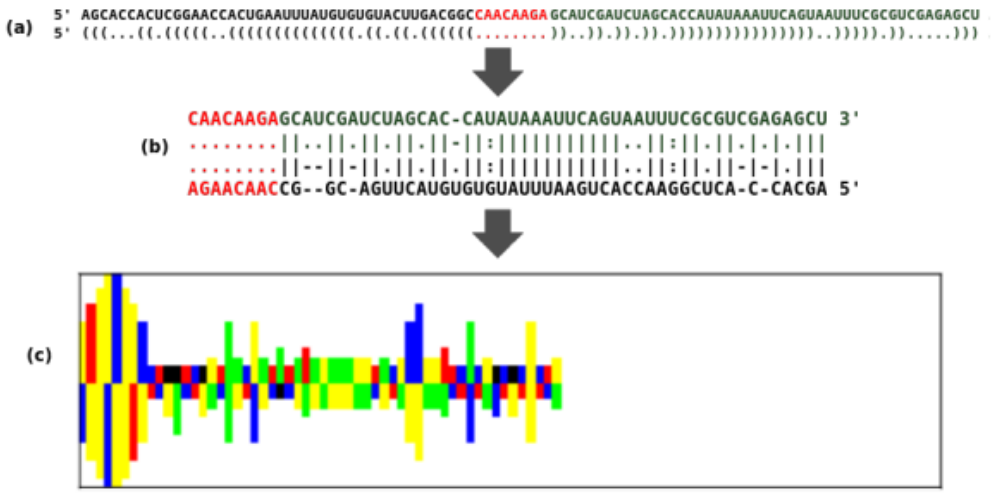


Figure 5.1: Taken from [14]. Encoding process of going from RNA sequence **(a)** with fold indication **(b)** to RGB encoding **(c)**.

To fold the RNA sequences, the RNAFold algorithm is used [46], which indicates for each nucleotide whether they have a bond with an opposing nucleotide. Then, the sequences are arranged in a top and bottom part, and gaps are introduced for nucleotides which have no opposing nucleotide.

Following the folding, each nucleotide and gap is assigned a color (A: blue, C: yellow, G: green, U: red and Gap: black). Next, the height of each bar representing nucleotides is determined by the bond strength, and an additional algorithm from [14] is used to increase the height of bars

when multiple weak bonds are next to each other. This way, the image encoding has a slight resemblance with the actual physical shape of pre-miRNA. The resulting encoding is a 25 by 100 pixels image encoding of the RNA, where the top half consists of 13 pixels and the bottom half of 12. An example of the encoding process can be seen in Figure 5.1.

Since we want to use a VAE to model this data, it is important that we can calculate the reconstruction loss well. Using mean squared error loss would result in unfair penalization. For example, the squared distance between a yellow pixel (1,1,0) and red pixel (1,0,0) is smaller than the distance between a yellow pixel and black pixel (0,0,0). Intuitively, these should be the same, as in both cases the wrong color of pixel has been used. Therefore, we model the encoded images as having 5 channels, where each channel is one-hot encoded with regards to pixel color. For white pixels, the value of each channel is 0, since a white pixel indicates the absence of any structure in the RNA encoding.

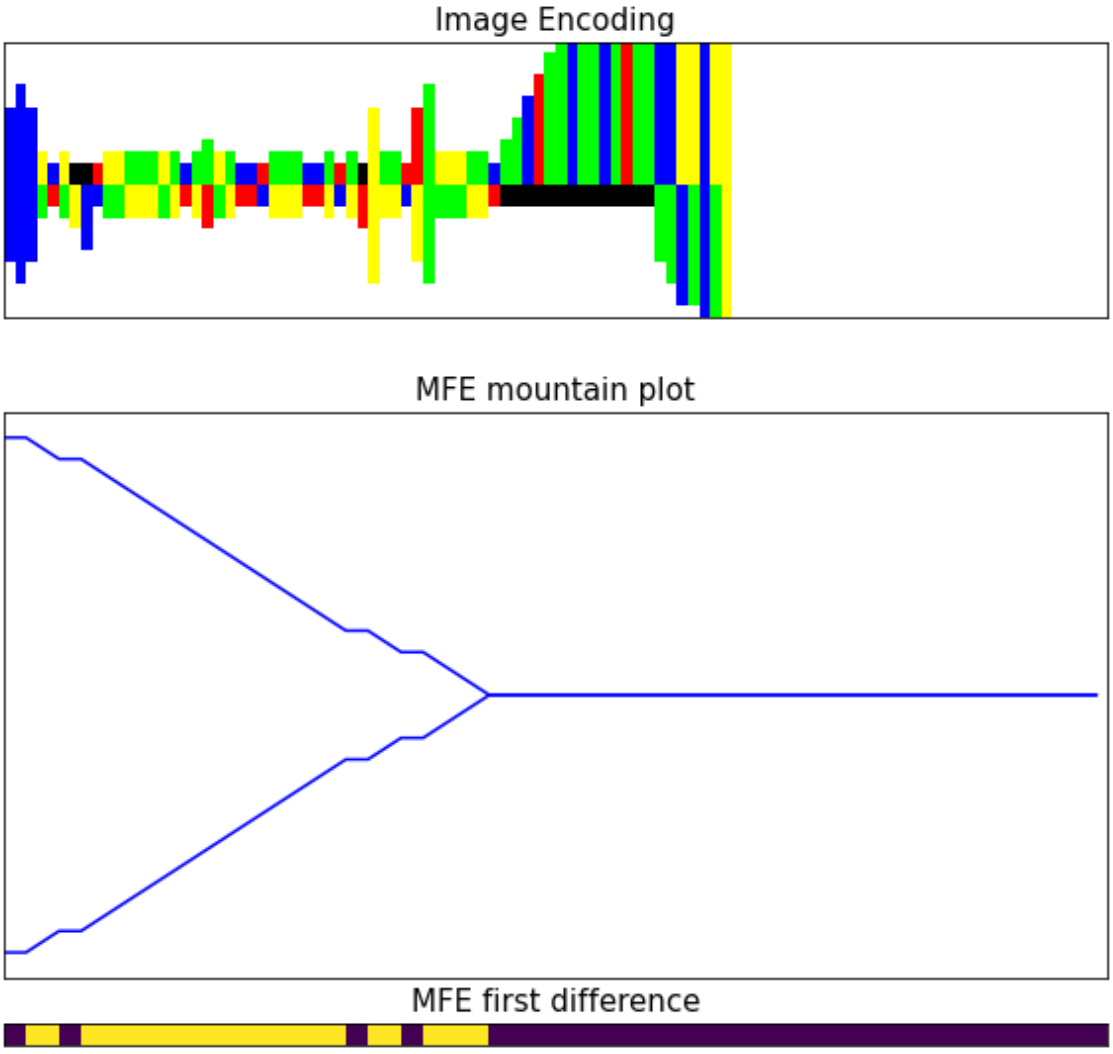


Figure 5.2: RNA image encoding with corresponding mountain plot of the MFE and the (absolute) first difference of the MFE. Yellow pixels indicate a value of 1, while purple pixels indicate a value of 0.

Next to the image encodings, we also make use of the Minimum Free Energy of the folded RNA molecules, which is also provided by RNAFold [46]. In the mountain plot of the MFE, slopes indicate that a nucleotide has a strong bond, while plateaus indicate loop structures or gaps. To make use of this data, we take the absolute value of the first difference of the mountain plot, meaning that we assign a value of 1 to strong bonds, while we assign a value of 0 to nucleotides without bonds or gaps. Then, we rearrange this data to follow the same shape as the RNA follows in the image encodings. In Figure 5.2, an example of an RNA image encoding can be found with the corresponding mountain plot of the MFE along with the first difference.

5.2 Models

To model the (non) pre-miRNA, we used multiple VAEs. For each VAE, the encoder, decoder and latent space can differ in architecture and techniques used. In general, each model follows a structure like in Figure 5.3. In the model, the encoded image of RNA is passed through the convolutional encoder, and is mapped to the latent space. In the decoder, the latent space is followed by a fully connected layer. This is followed by two separate decoders for the color and length of each bar, which can consist of either deconvolutional or fully connected layers. In the sections below, we will present an overview of the different architectures and their components in more detail and explain how they are used in the context of modeling miRNA.

5.2.1 Encoders

We make use of two types of encoders. The models, which have a latent space where IAF is applied, have an encoder based on the ResNet inspired architecture from [40]. The encoder for the remaining models is VGG-isnpired, based on the miRNA classification model used in [14]. In addition, models with IAF make use of the ELU activation functions, while the other models use the ReLU activation function in all their modules.

VGG-inspired

The VGG-inspired encoders have an architecture consisting of blocks. Each block contains 2 convolutional layers, each followed by a batch normalization layer with a stride of 1 and same padding, followed by a 2 by 2 max-pooling layer. For each convolutional layer, we use the ReLU activation function. An overview of the architecture of these blocks is in Table 5.1a. For each encoder, we stack 3 of these blocks, followed by a fully connected layer mapping to the parameters of the latent space (μ and σ). The fully connected layer for μ has no activation, while the fully connected layer for σ uses softplus activation. To sample $\mathbf{z}$ from this latent space, we make use of the reparametrization trick from Equation 3.5. The full architecture of the network can be found in Table 5.1.

Table 5.1: Network architecture for VGG encoders

(a) Convolutional Block architecture

Layer	Params	Input
Conv1	$k_1 \times k_2, f$	Input
BatchNorn1	f	Conv1
RELU1		BatchNorm1
Conv2	$k_1 \times k_2, f$	RELU1
BatchNorm2	f	Conv2
RELU2		BatchNorm2
MaxPool	$p_1 \times p_2$	RELU2

(b) Full VGG network architecture

Layer	Params	Input
ConvBlock1	$k_1 \times k_2, f_1, p_1 \times p_2$	Input
ConvBlock2	$k_3 \times k_4, f_2, p_3 \times p_4$	ConvBlock1
ConvBlock3	$k_5 \times k_6, f_3, p_5 \times p_6$	ConvBlock2
Flatten		ConvBlock3
z_μ (FC)	$size(z)$	Flatten
z_σ (FC)	$size(z)$	Flatten

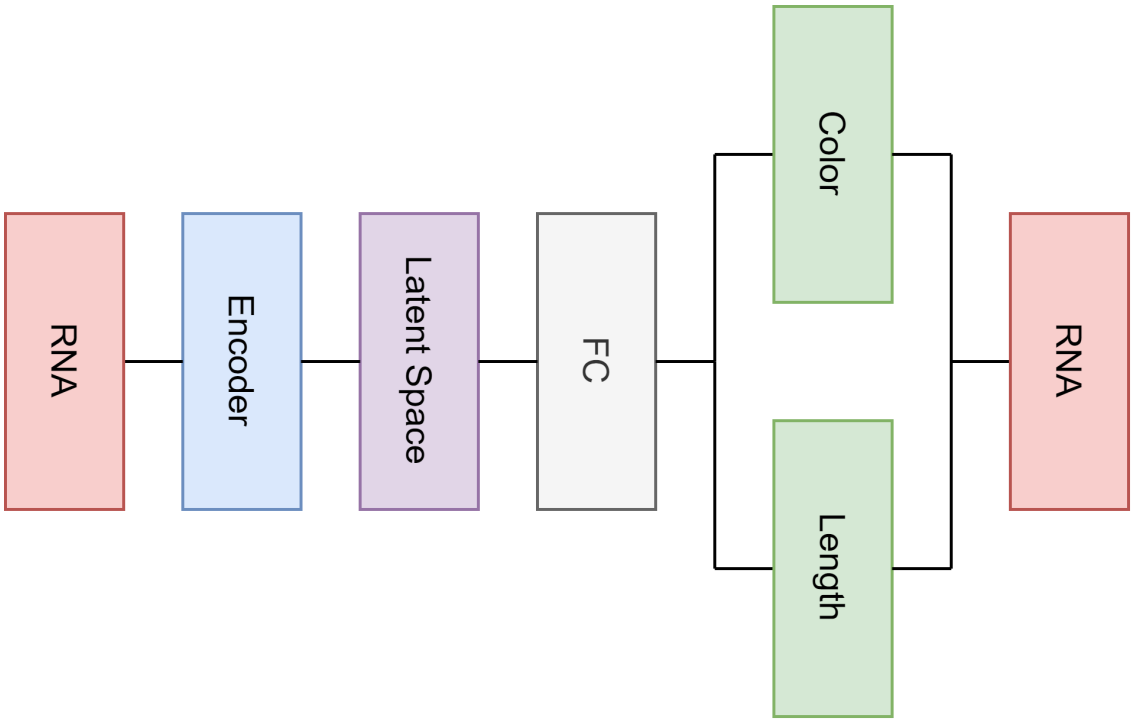


Figure 5.3: General overview of the model architecture used to model pre-miRNA.

Table 5.2: Network architecture for ResNet encoders

(a) ResNet block architecture

Layer	Params	Input
Conv1	$k_1 \times k_2, f$	Input
BatchNorm1	f	Conv1
ELU1		BatchNorm1
Conv2	$k_1 \times k_2, f$	ELU1
SUM	α	Input, Conv2
BatchNorm2	f	SUM
ELU		BatchNorm2

(b) ResNet downsample block architecture

Layer	Params	Input
Conv1	$k_1 \times k_2, f, s_1$	Input
BatchNorm1	f	Conv1
ELU1		BatchNorm1
Conv2	$k_1 \times k_2, f, s_2$	ELU1
Conv0	$k_3 \times k_4, f, s_1$	Input
SUM	α	Conv0, Conv2
BatchNorm2	f	SUM
ELU		BatchNorm2

(c) ResNet stem architecture

Layer	Params	Input
Conv1	$k_1 \times k_2, f_1$	Input
BatchNorm1	f_1	Conv1
ELU1		BatchNorm1
Conv2	$k_1 \times k_2, f_2$	ELU1
BatchNorm2	f_2	Conv2
ELU2		BatchNorm2
MaxPool	$p_1 \times p_2$	ELU2

(d) Full ResNet network architecture

Layer	Params	Input
Stem	$k_1 \times k_2, f_{1,2}, p_1 \times p_2$	Input
RB1	$k_3 \times k_4, f_3$	Stem
RBdown1	$k_3 \times k_4, f_4$	RB1
RB2	$k_3 \times k_4, f_4,$	RBdown1
RBdown2	$k_3 \times k_4, f_5$	RB2
Flatten		RBdown2
z_μ (FC)	$size(z)$	Flatten
z_σ (FC)	$size(z)$	Flatten
Context (FC)	$size(context)$	Flatten

ResNet-Inspired

Similarly to the VGG-Inspired encoders, the ResNet-Inspired encoders also consist of blocks. In the ResNet inspired network, we make use of two types of residual blocks: regular blocks and

downsample blocks. In each of the blocks, we pass the input through a convolutional layer, followed by batch normalization and an ELU activation. We then pass it through another convolutional layer, following which we perform a weighted elementwise sum between the input and the output, where we give the output a weight of α. An overview of this architecture is in Table 5.2a. The downsampling blocks follow a similar architecture. However, in the first convolutional layer we use a stride of 2. To make elementwise summation possible, we also need to downsample the input. We do this by applying a convolutional layer with a 1 by 1 kernel and a stride of 2. We then perform the same summation as in the normal residual block and perform batch normalization and ELU activation afterwards. See Table 5.2b for more details. Finally, to build the entire encoder with residual blocks, we first have a stem (Table 5.2c) which consists of two convolutional layers followed by batch normalization and ELU activation. We then have two pairs of regular residual and residual downsampling blocks, where we increase the amount of filters at each downsampling block. Finally, we flatten the output, which is followed by dense layers to calculate the posterior as well as the context $\mathbf{h}$ which is later used for the autoregressive transformations. The full architecture can be found in Table 5.2.

5.2.2 Latent Space

Depending on which model configuration we use, there are different ways in which the latent space is calculated and how the KL Divergence is obtained.

Normal Latent Space

In the case of a vanilla VAE, or for the $\mathbf{z_x}$ latent space of the DIVA, the distribution of the posterior is obtained by passing $\mathbf{x}$ through the encoder. The prior is standard normal distribution, i.e. $\mathcal{N}(0, I)$.

DIVA latent space

For a DIVA model, we partition the latent space in a latent space for the class of the RNA ($\mathbf{z}_y$), the MFE of the RNA ($\mathbf{z}_m$) and the remaining variability in the data ($\mathbf{z_x}$). The latent spaces for y and m function differently than a normal latent space. For these latent spaces, we make use of a conditional prior, which is parameterized by a neural network.

The conditional prior of y is calculated by a neural network with a hidden layer of the same size as the dimension of the latent space, which is followed by batch normalization and ReLU/ELU activation. On this output, we apply a fully connected layer (without activation) to calculate the mean of the prior distribution, and a fully connected layer (with softplus activation) to calculate the standard deviation of the prior distribution.

Since the MFE follows a similar shape as the RNA images, we make use of 1-dimensional convolutional layers to calculate the prior. We stack 6 convolutional layers, followed by 2 separate fully connected layers to calculate the mean and the standard deviation, in the same way as for the conditional prior for y. Following each convolutional layer, we apply batch normalization and ReLU/ELU activation depending on the model.

In addition to conditional priors, these latent spaces also have auxiliary classifiers built on top of them. The classifier for y is simple, using only fully connected layers on top of the latent space with softmax activation. On the latent space of the MFE, we build an auxiliary reconstructor, which follows the same architecture as the conditional prior, only reversed, where the convolutional layers are replaced with transposed convolutional layers. In addition, an auxiliary predictor for y is also built on this latent space, to encourage further separation between classes.

IAF

When applying IAF, we first sample $\mathbf{z_0}$ and $\mathbf{h}$ from the encoder. For the inverse flow, we stack 2 MADE layers to make one flow block. Multiple flow blocks are stacked to improve the transformation of the latent variables. The output of flow block t is $\mathbf{m}_t$, $\mathbf{s}_t$ and $\log \sigma_t$. The first two allow us to calculate the transformation from Equation 3.10, while the last output is used to calculate the density as in Equation 3.11, which is used in the KL-Divergence term of the loss function.

Naturally, we can also apply these flow layers on top of the latent spaces in a DIVA model. In this case, the reconstructors and predictors are built on the final outputs of the flow layers instead of on the initial samples from $q_\phi(\mathbf{z_0}|\mathbf{x})$.

5.2.3 Decoders

In the decoder, we have a separate module for decoding the color and the length of bars of the RNA. Before passing $\mathbf{z}$ through the decoder, in the case of a DIVA model, all latent spaces are concatenated. We make use of two different architectures for the decoder: one where the length and color of bars is reconstructed using fully connected layers, and one where this is done using deconvolutional layers.

To reconstruct the length of bars, we make use of a stamp matrix [72], which contains the shape of all possible bars. In both fully connected and deconvolutional architectures, we calculate for each bar the probability for each length. In other words, we calculate the probability of the bar having a length of 0, a length of 2, a length of 3, all the way to having a length of 13. We then multiply the stamp matrix with the probability distribution of the lengths to obtain a "distribution" of the shape of the RNA. For the colors, we simply calculate the probability for each color per nucleotide, which we multiply with the bar distribution to obtain reconstruction. When sampling from the decoder, we sample from the probability distributions before taking the same steps.

In Figure 5.4, a visual overview is represented of how the decoder works, using artificially data, only for the top half of the reconstruction. At (1), we have the probability distribution of the bar lengths of the top half of our image. We multiply this with (2), which is the stamp containing all the possible bars. From this multiplication, we obtain (3), which is a "distribution" of the shape. This operation is summarized in equation 5.1, for the top half. Here, S is the 13 by 13 stamp matrix (2), P is the matrix containing the probabilities for each length per bar (1), and D is the output of the operation (3). For this example, we took a sample of the color at (4), while in reality it would be the probabilities per color. We multiply (3) and (4), from which we obtain (5), which is used to calculate the reconstruction loss. When sampling from the decoder, we take a sample per bar at (1), instead of using the full distribution.

$$S = \begin{bmatrix} 0 & 0 & \dots & 0 & 1 \\ 0 & 0 & \dots & 1 & 1 \\ \vdots & & \ddots & & \vdots \\ 0 & 1 & \dots & 1 & 1 \\ 0 & 1 & \dots & 1 & 1 \end{bmatrix} \qquad P = \begin{bmatrix} l_{1,1} & \dots & l_{1,100} \\ \vdots & \ddots & \vdots \\ l_{13,1} & \dots & l_{13,100} \end{bmatrix} \tag{5.1}$$

$$D = SP$$

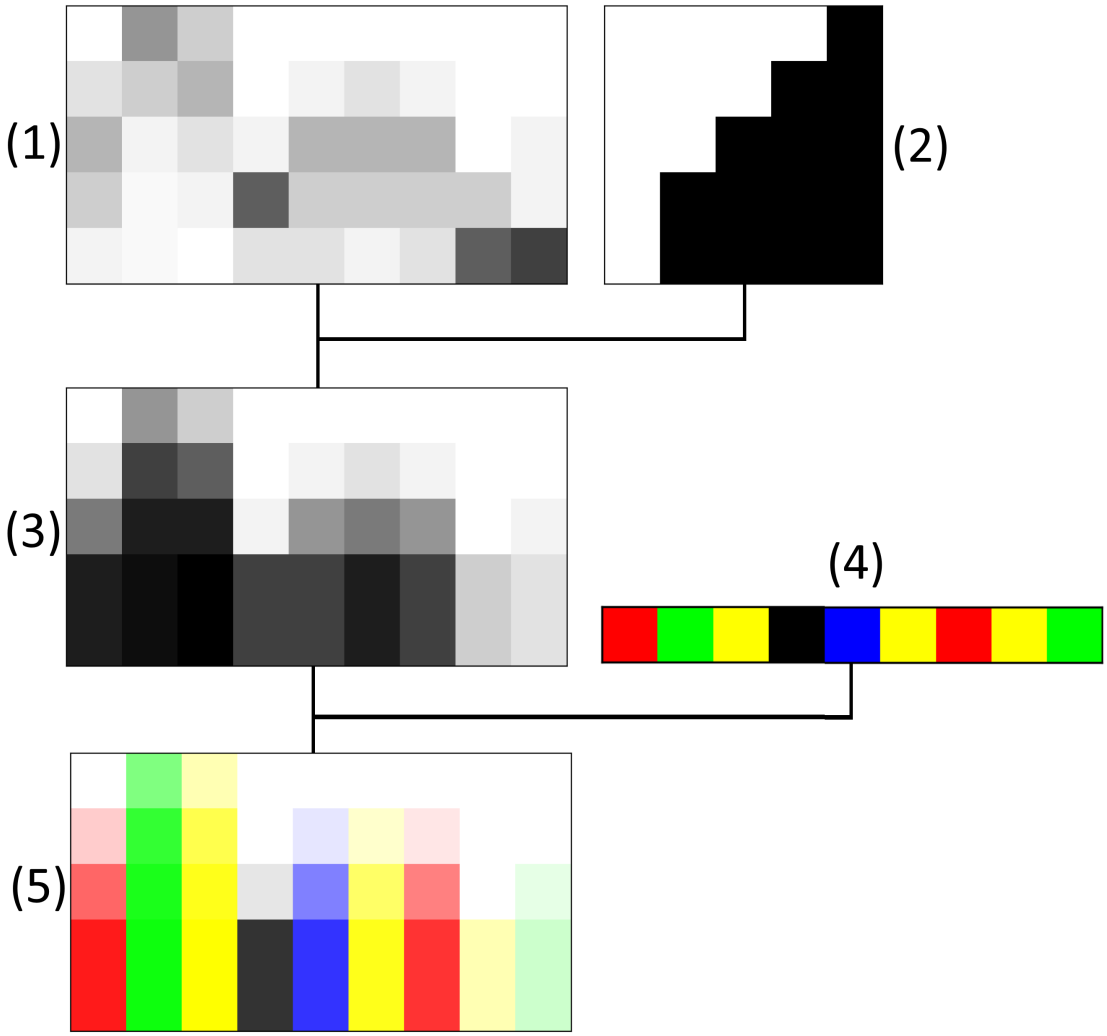


Figure 5.4: Process of going from the output of the bar length decoder **(1)** and bar color decoder **(4)** to reconstructions **(5)**.

Fully Connected Decoder

In this architecture, the latent space is followed by 2 fully connected layers of 512 dense units, which are both followed by batch normalization and (low) dropout layers (Table 5.3c). The output of these layers is used by two other fully connected layers, which calculate the color of bars and length of bars respectively.

The fully connected layer for bar length calculation has an output of shape $(2, 100, 13)$, after which a softmax layer is applied on the last dimension. The first index represents the top and bottom bars, the second index represents the length of the image (100 pixels) and the final image represents the probability distribution of length per bar. To obtain the lengths of bars, we perform (batched) matrix multiplication between this output tensor and our stamp matrix[72], which results in a $(2, 100, 13)$ tensor. We then reshape the matrix in such a way that it represents the original image with a shape of 25 by 100, by removing the final row from the bottom bars. An overview of this part of the decoder can be found in Table 5.3b.

The layer for color calculation has an output of shape $(2, 100, 5)$, on which we apply a softmax along the last dimension. We repeat elements of this tensor such that they cover the entire image, and reshape it such that it matches the original image shape, resulting in a shape of $(25, 100, 5)$. The architecture of this part of the decoder can be found in Table 5.3a.

We then element-wise multiply the color output with length output, to obtain the reconstruction. Before doing this, we add an extra dimension to the output of the bars, and repeat this dimension 5 times. The reconstruction loss is then calculated by taking the Mean Squared Error (MSE) between the original input and the reconstruction. An overview of this architecture can be found in Figure 5.3.

Table 5.3: Network architecture for Fully Connected Decoders

(a) Fully Connected Decoder Color

Layer	Params	Input
FC	1000	Input
Reshape	(2,100,5)	FC
Softmax		Reshape
Expand	(25,100,5)	Softmax

(b) Full Connected Decoder Length

Layer	Params	Input
FC	2600	Input
Reshape	(2,100,13)	FC
Softmax		Reshape
Stamp		Softmax
Rearrange	(26,100)	Stamp
Slice	25,100	Rearrange

(c) Fully Connected Decoder Stem

Layer	Params	Input
FC1	512	Input
BatchNorm1	512	FC1
DropOut1	do_1	BatchNorm1
FC2	512	DropOut1
BatchNorm2	512	FC2
DropOut2	do_2	BatchNorm2

(d) Full FC decoder network architecture

Layer	Params	Input
Stem	do_1, do_2	Input
Color		Stem
Length		Stem
Multiply		Color, Length

Deconvolutional Decoder

Next to this, we also modeled the decoder for the length and color of the bars using transposed convolutional layers. In this case, we first have a dense layer mapping the latent space to 1152 neurons, after which batch normalization and a ReLU activation are applied.

For the reconstruction of the length of the bars (Table 5.4b), we take the output of 1152 neurons, and reshape it into a 3 by 12 matrix with 32 channels. On this matrix, we apply deconvolutional blocks, which consist of a transposed convolutional layer, followed by batch normalization and ReLU activation. The architecture of these block can be found in Table 5.4c. We stack 6 of this blocks, where on 2 of the blocks we perform upsampling, by using a stride of 3.

For the color reconstruction (Table 5.4a), the output of 1152 neurons is also reshaped in a 3 by 12 matrix with 32 channels. We then take a sum over the height (3), to obtain a 1-dimensional matrix with 32 channels. On this matrix, we perform 1-dimensional transposed convolutions, using a similar architecture as for the length of bars. Following the output of both these networks, the output is combined in a similar way as in the fully connected decoder. The full architecture can be found in Table 5.4d.

Table 5.4: Network architecture for Deconvolutional Decoders

(a) Deconvolutional Decoder Color

Layer	Params	Input
SUM	$height$	Input
CTB1	f_1, k_1, s_1, p_1	SUM
CTB2	f_1, k_1, s_2, p_2	CTB1
CTB3	f_2, k_1, s_1, p_1	CTB2
CTB4	f_2, k_1, s_2, p_2	CTB3
CTB5	f_1, k_1, s_2, p_2	CTB4
ConvT	f_3, k_2, s_2, p_1	CTB5
Softmax		ConvT
Reshape	2,100,5	Softmax
Expand	25,100,5	Reshape

(b) Deconvolutional Decoder Length

Layer	Params	Input
CTB1	$f_1, k_1 \times k_2, s_1, p_1$	Input
CTB2	$f_1, k_1 \times k_2, s_2, p_2$	CTB1
CTB3	$f_2, k_1 \times k_2, s_1, p_1$	CTB2
CTB4	$f_2, k_1 \times k_2, s_2, p_2$	CTB3
CTB5	$f_1, k_1 \times k_2, s_2, p_2$	CTB4
ConvT	$f_3, k_3 \times k_4, s_2, p_1$	CTB5
Softmax		ConvT
Stamp		Softmax
Slice	25,100	Stamp

(c) Deconvolutional Decoder Block

Layer	Params	Input
ConvT	$f, k_1(\times k_2), s, p$	Input
BatchNorm	f	ConvT
(R)ELU		BatchNorm

(d) Full Deconvolutional decoder network architecture

Layer	Params	Input
FC	1152	Input
Reshape	3,12,32	FC
BatchNorm	32	Reshape
(R)ELU		BatchNorm
Color	$f_{1-3}, k_{1,3}, s_{1,2}$	(R)ELU
Length	$f_{1,2,4}, k_{1-4}, s_{1,2}$	(R)ELU
Multiply		Color, Length

5.2.4 Full Model Architectures

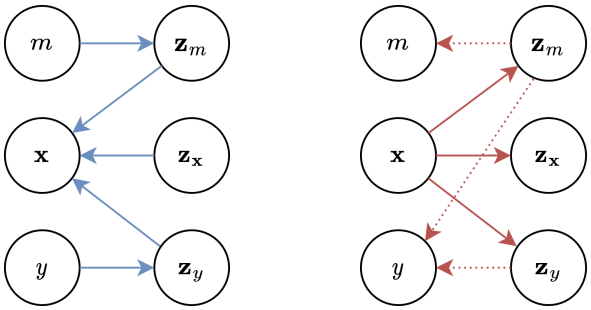
In terms of the global architecture, there are two different types of models. While both models are types of VAEs, there are a few differences between the models which use the standard VAE architecture and models using an architecture based on DIVA.

In the probabilistic graph model of the standard VAE architecture in Figure 5.5, $\mathbf{x}$ is generated by sampling from the prior distribution and passing the sample through the decoder. Inference is done by passing $\mathbf{x}$ through the decoder, and sampling from the obtained posterior distribution.

The DIVA architecture is slightly more complicated, as it can be seen in the probabilistic graph model in Figure 5.6. For generation, $\mathbf{z}_m$ and $\mathbf{z}_y$ are sampled from conditional priors, which are parameterized through neural networks. To obtain the conditional prior, both y and m are needed as input. $\mathbf{z}_\mathbf{x}$ is sampled from a standard normal prior, following which the three partitions of the latent space are concatenated and passed through the decoder to obtain $\mathbf{x}$ again. In inference mode, the three posteriors are obtained by passing $\mathbf{x}$ through three separate decoders. Following the sampling of the posteriors, inference on y is separately performed using $\mathbf{z}_m$ and $\mathbf{z}_y$, while inference on m is performed using only $\mathbf{z}_m$.

(a) Generation (b) Inference

Figure 5.5: Probabilistic graph model for inference and generation using a standard VAE architecture.



(a) Generation (b) Inference

Figure 5.6: Probabilistic graph model for inference and generation using a DIVA architecture.

Sampling

To sample RNAs using the decoder, we take the outputs of the color and bar length predictions following their softmax layers. We then have a choice whether we want to sample from these distributions, or take the most likely length/color for each bar. We then can reconstruct these in an RGB image of an RNA strand, in a similar way to how the decoder works.

5.3 Interpretable Latent Space

To make interpretable explanations using the latent space, we propose a framework using Decision Trees to make interpretable classifications using the latent space. To make these classifications, each datapoint should be labelled with multiple features, which are potential explanatory variables for the classification task. The datapoints are labelled using the algorithms for calculating concept values from [69]. Using the labelled datapoints and their representations in the latent space, a Decision Tree is constructed. Using this tree, it is then possible to make explainable predictions from the learned latent space.

The algorithm for learning our Decision Tree is similar to CART and C4.5, with some modifications. To calculate a split based on a variable, we make use of a secondary (simple) ML algorithm. Using this second algorithm, we train a classifier for the current variable using the latent space. Then, to evaluate the split, we check if the classifier achieves a good performance and how big the information gain is. When making splits, the classifier with the highest information gain and performance above a threshold is used. The data is then split according to the predicted classes by the classifier. A complete overview of the algorithm can be found in Algorithm 1.

In the latent space, we expect that our model found distributions of the labelled features. For relevant features, we expect that there are strong correlations with the latent variables, meaning that in one side of the latent space we expect the feature to have a high value, while in the other side we expect it to have a low value. In our Decision Tree algorithm, we make use of classifiers which find a "simple" decision boundary, such as an SVM with linear kernel. These classifiers are then expected to perform well on the relevant features, as these should be easier to separate in this case. Furthermore, by using a simple classifier, we prevent the classifier from "ignoring" the latent space, since a complex classifier could find a direct map between the latent space and the features. In turn, this would allow the Decision Tree to only learn from the feature data, rendering the latent space meaningless.

We apply the proposed Decision Tree to make a description of pre-miRNA, using the algorithm for calculating concepts defined in [69]. We applied the concept calculating algorithm on both training and test sets, and trained the Decision Tree on (a part of) the latent space with these

Algorithm 1 MakeSplit($\mathbf{z}, y, concepts, depth$)

Require: $max\_depth, min\_samples, thresholds, min\_acc$
$depth \leftarrow depth + 1$
Node $\leftarrow Node(\mathbf{z}, y, concepts)$
if $depth \geq max\_depth$ **or** $len(y) < min\_samples$ **or** Node is pure **then**
  return Node
end if
entropy $\leftarrow entropy(y)$
gain $\leftarrow 0$
cls $\leftarrow None$
for i in $columns(concepts)$ **do**
  for j in splits using thresholds of $concepts[i]$ **do**
    train classifier ($X = \mathbf{z}, y = j$)
    split $\mathbf{z}$ based on classifier predictions in $\mathbf{z_0}$, $\mathbf{z_1}$
    $new\_entropy \leftarrow$ weighted_mean($entropy(\mathbf{z_0}), entropy(\mathbf{z_1})$)
    $info\_gain \leftarrow$ entropy $-$ $new\_entropy$
    if $info\_gain >$ gain **and** acc(classifier)$> min\_acc$ **then**
      gain $\leftarrow info\_gain$
      cls $\leftarrow$ classifier
    end if
  end for
end for
Split data into $\mathbf{z_0}$, $\mathbf{z_1}$, $concepts_0$, $concepts_1$, y_0, y_1 according to cls
Node.left_child $\leftarrow$ **MakeSplit**($\mathbf{z_0}$,y_0,$concepts_0$,$depth$)
Node.right_child $\leftarrow$ **MakeSplit**($\mathbf{z_1}$,y_1,$concepts_1$,$depth$)
return Node

concepts. The tree was trained using an SVM with a linear kernel as the concept classifier, which needs to achieve an accuracy of at least 90% to be acceptable. We used a minimum amount of samples of 10 per leaf and a maximum depth of 4. For continuous variables and other variables with a large number of unique values, we have defined multiple thresholds, which split the data in a class greater than the threshold, and a class smaller or equal to the threshold. The results of the Decision Tree were compared to the biological knowledge about pre-miRNA. The results of this method should only be considered as a proof of concept, as for proper performance the used features should be defined using expert domain knowledge.

5.4 Experiments

5.4.1 Model

Using the previously defined model components, we use a DIVA model with 3 latent spaces and IAF to model the pre-miRNA. In addition, we use the deconvolutional variant of the decoder. Since we use IAF, the model uses the ResNet encoder architecture in all 3 encoders, with the same hyperparameters for each encoder, using the architecture from Table 5.2d.

In the stem, the model uses a kernel size of 5 by 5 (k_1, k_2), 32 filters (f_1) in the first convolutional layer and 48 filters (f_2) in the second convolutional layer. Finally, 2 by 2 max pooling (p_1, p_2) is applied at the end of the stem.

The filters used by the ResNet network are increasing from 48 (f_3), to 64 (f_4), to 80 (f_5). In the ResNet blocks (Tables 5.2a and 5.2b), a kernel size of 3 by 3 (k_1, k_2) is used, with exception from the identity downsampling, where a kernel size of 1 by 1 (k_3, k_4) is used. In all layers, we use

"same" padding and a stride of 1, with exception of the downsampling blocks. The first convolutional layers of these blocks, as well as the identity convolution (Conv0), use a stride of 2 (s_2). The elementwise sum operation of each block uses an α of 0.2.

After the ResNet blocks, the output is flattened. The flattened output is followed by 2 separate fully connected layers, used to calculate the distribution of the posterior. For each latent space, we use a size of 64. In addition, there is a separate fully connected layer for the context **h**, which has 32 nodes. The latent space is followed by 8 stacked blocks of 2 MADE layers. Each MADE layer has a hidden size of 1080 neurons.

In the decoder (Table 5.4d), we use a kernel size (k_1, k_2) of 3 (by 3) for all operations before the final layer, both in the decoder for color and length. At layers without upsampling, we use a stride of 1 (s_1) and "same" padding, while at layers with upsampling we use a stride of 3 (s_2) without padding. The final layer in both decoders uses a 1 (by 1) kernel (k_3, k_4), with 1 filter ($f4$) for the bar length and 10 filters ($f3$) for the bar color.

For training the model, we used the Adam optimizer with a learning rate of 0.0005 and a batch size of 32. As we include multiple auxiliary classifiers, we modify the loss function from Equation 3.9. The new function can be found in Equation 5.2. The auxiliary losses for predicting y on $\mathbf{z}_y$ (α_{y_1}) and $\mathbf{z}_m$ (α_{y_2}) both have a penalty term of 12, and are calculated using categorical cross-entropy, while the reconstruction loss of m on $\mathbf{z}_m$ (α_m) has a penalty term of 1 and is calculated using MSE.

$$\begin{aligned}\mathcal{F}_{\text{DIVA}}(y, \mathbf{x}, m) = \mathcal{L}_s(y, \mathbf{x}, m) \\ &+ \alpha_{y_1} E_{q_{\phi_y}(\mathbf{z}_y|\mathbf{x})}[\log q_{\omega_y}(y|\mathbf{z}_y)] \\ &+ \alpha_{y_2} E_{q_{\phi_y}(\mathbf{z}_m|\mathbf{x})}[\log q_{\omega_y}(y|\mathbf{z}_m)] \\ &+ \alpha_m E_{q_{\phi_m}(\mathbf{z}_m|\mathbf{x})}[\log q_{\omega_m}(m|\mathbf{z}_m)]\end{aligned} \tag{5.2}$$

We investigated the latent spaces of the model, to show that our model was able to find meaningful generative factors of the data. Furthermore, we also investigated the model's capability to reconstruct pre-miRNAs and its (conditional) sampling and interpolation quality.

To showcase our latent space interpretation methods, we trained a Decision Tree on the latent space. We analyzed the pre-miRNA description developed by the Decision Tree, by investigating the relationships it found in the latent space. In addition, we compared these findings to biological features of pre-miRNA.

5.4.2 Experiment Structure

To show how and why design choices for our model were made, we performed additional experiments. We show how a vanilla VAE is unsuitable for the task, and how other design choices improve performance of the model. We also show how using β-VAE improves model performance, as well as how adding IAF, using a DIVA architecture and using a deconvolutional decoder helps increase the performance of the model. For models without IAF, which use a VGG-inspired encoder, we increase the amount of filters the same way as in the ResNet encoders, starting from 48, to 64 and finishing with 80. However, unlike the ResNet encoder, the VGG encoder does not have a stem. In Table 5.5, a concrete overview of the experiments can be found. The code used for all experiments and results can be found on `https://github.com/merkelmauer/mirna`.

Table 5.5: Experiments for generative pre-miRNA modeling. Bold row indicates our final model.

Model Name	β	DIVA	IAF	DeConv	$\mathbf{z}(\mathbf{z}_y, \mathbf{z}_\mathbf{x}, \mathbf{z}_m)$
VAE	1	No	No	No	192 (-)
β-VAE	0.05	No	No	No	192 (-)
β-IAF-VAE	0.5	No	Yes	No	192 (-)
DC-β-IAF-VAE	0.5	No	Yes	Yes	192 (-)
DC-IAF-DIVA	**0.5**	**Yes**	**Yes**	**Yes**	**192 (64,64,64)**

Chapter 6

Results

6.1 Model Performance

We first investigated how the performances of the different models compare quantitatively. For this, we looked at the different losses recorded during training and testing (Table 6.1). We calculated the reconstruction loss in terms of the sum of the mean squared error per pixel, while the KL loss was calculated as the sum of KL losses per latent variable. In Appendix A, loss curves for the different models can be found. In Table 6.2, we further investigated the reconstruction performances of the different models. Here, we took the most likely reconstruction of each image and calculated the MAE for the entire image, nucleotide color and the length of the bars.

We also assessed the quality of the reconstructions in more detail. In Figure 6.1, the MAE for each reconstruction per pixel can be found. In addition, we also looked at how reconstructions for some images compare between models in Figure 6.2. More reconstructions can be found in Appendix B.

Table 6.1: Training statistics of different model types. Bold entries represent lowest obtained losses. Entries in brackets for DC-IAF-DIVA indicate KL losses per latent space.

	Loss		Reconstruction Loss		KL loss ($\mathbf{z}_y$,$\mathbf{z}_\mathbf{x}$,$\mathbf{z}_m$)	
Model Name	Train	Test	Train	Test	Train	Test
VAE	365.1	375.8	332.9	343.6	32.3	32.3
β-VAE	186.8	154.6	166.5	134.8	405.3	395.6
β-IAF-VAE	144.3	142.2	135.1	132.8	18.9	18.3
DC-β-IAF-VAE	**16.9**	**24.1**	**14.7**	**21.8**	**4.4**	**4.6**
DC-IAF-DIVA	117.9	126.1	26.2	32.1	154.2	155.4
					(69.6, 35.7, 48.9)	(69.5, 35.1, 50.8)

Table 6.2: Reconstruction statistics for different model types. Bold statistics represent lowest obtained errors.

Model Name	MAE	MAE nucleotide	MAE length
VAE	0.1363	0.3053	0.7844
β-VAE	0.0550	0.0919	0.4779
β-IAF-VAE	0.0531	0.0824	0.4443
DC-β-IAF-VAE	0.0085	0.0150	0.0670
DC-IAF-DIVA	**0.0068**	**0.0122**	**0.0567**

In the vanilla VAE with a fully connected decoder, we can see that the network did not seem to learn to model the data. The reconstruction error seems high, and throughout training the KL divergence has constantly been going up (Figure A). By inspecting some of the reconstructions from the model (Figure 6.2), we can see that the model is able to reconstruct the terminal loop, as well as the rough length of each RNA. However, past the terminal loop the reconstruction quality drops off severely, which is also backed up by the statistics from Table 6.2, showing that, on average, around 30% of the nucleotides are reconstructed wrongly. Simultaneously, the MAE for bar length is 0.78. While this seems like a low number, it should be noted that the error was also calculated for "non-existing" bars at the end of each image. Similarly, we see in Figure 6.1 that there are reconstruction errors in almost the entire stem region of most (non) pre-miRNAs, which further indicates a poor reconstruction performance of the model.

A possible explanation for the poor performance of the model can be explained by the nature of the dataset. While the image data is very structured, it is much more complex than common datasets used in literature, such as MNIST (handwritten digits) and CelebA (faces) [45]. In these datasets, most datapoints are similar in nature. For example, in CelebA, each image has common features such as eyes, noses, ears and mouths, all being in roughly similar locations between the images. Similarly, in MNIST, each number is written in roughly the same way with no major differences. On the other hand, the stems of the modeled RNA molecules can be vastly different in terms of structure and nucleotide sequence. Having a too high penalty term on the KL-divergence in turn makes it harder to differ between points in the latent space, which does not enable the model to make good reconstructions.

By lowering the β parameter to 0.05 (β-VAE), the model achieved a drastically better reconstruction performance. As can be seen in Figure 6.2, the reconstructions are more accurate. It should be noted that the model still made mistakes in bar lengths and sometimes made mistakes in terms of color. In Figure 6.1, we can also see that the model frequently made mistakes in the length of bars with strong bonds, making them slightly too long or too short. In addition, due to setting β to 0.05, the KL divergence increased more than tenfold compared to the normal VAE (Table 6.1).

In the β-IAF-VAE, we introduced IAF on top of the latent space, and increased the β parameter to 0.5. Compared to the β-VAE, this model achieved a slightly better reconstruction performance, while it achieved a significantly more regularized latent space (Table 6.1). However, like with the normal β-VAE, the reconstructions this model produces still contain visible mistakes, and the errors are distributed somewhat similarly.

By using a deconvolutional decoder for bar length and color, the reconstruction performance increased significantly. Reconstructions in Figure 6.2 show few mistakes, which is further backed up by the statistics in Table 6.2. Furthermore, this model also achieved the lowest overall loss, as well as lowest reconstruction loss and KL loss (Table 6.1). Looking at Figure 6.1, we can see that the model produces fairly few errors in the left part of the image, while it produces somewhat more errors in the latter part of the image. This is likely a result of the dataset having relatively few long (non) pre-miRNAs, which limited the model's ability to learn how to reconstruct that part of the image.

In terms of the MAE for the different reconstruction statistics, the DIVA model performed better, despite having a worse reconstruction loss than the DC-β-IAF-VAE. Since the reconstruction loss is calculated based on the distribution of the bar length and color, it is possible that the DIVA model has more uncertainty in the reconstruction of the bar length and color. However, if the most likely choice was still the correct one, it can result in the lower MAE statistics as seen in Table 6.2. We can also see this in Figure 6.1, that the DIVA model made less mistakes than the DC-β-IAF-VAE. On the other hand, the DIVA model has a significantly higher KL Loss than the DC-β-IAF-VAE, as a result of its partitioned latent space.

To compare the latent space between the two models, we compared $\mathbf{z}_m$ from the DIVA model and the entire latent space of DC-β-IAF-VAE. For this, we reduced the dimensionality of both latent spaces to 2 using t-SNE [70]. In this dimensionality reduction method, the distance between points is preserved in such a way that similar points are close together, while more dissimilar points are further apart. Therefore, we expect that in a well organized latent space, points close together will have similar properties. Likewise, we expect that when relevant properties change, the change will be happening gradually in its cluster of similar points. Using the class labels and concepts defined in [69], we colored each datapoint according to their label or concept value.

In Figure 6.3, we compared the latent spaces of the two models, and colored the datapoints according to class label, the presence of the terminal loop, the vertical size of the loop (in pixels), the length of the stem and the fraction of base pairs in the stem. For the class labels and the presence of a terminal loop, we can see clear separations in the latent space of the DIVA model, while especially for the class label this separation is not present in the VAE. For the size of the terminal loop, there is a clear trend in the DIVA latent space. On the other hand, in the VAE latent space some datapoints with similar loop sizes are clustered together. However, there does not seem to be a clear between datapoints with different loop sizes. Likewise, we see that the stem length and the fraction of base pairs is smoothly changing in the DIVA latent space, while in the VAE latent space datapoints with similar values are scattered throughout the latent space. While there are still datapoints with similar values clustered together, it is not to the same extent as for the DIVA latent space.

Overall, we can conclude that the additions of IAF, separate latent spaces through DIVA and using a deconvolutional decoder all provided performance gains for the modelling of pre-miRNA. We have shown that each addition improves the reconstruction performance, which resulted in a final model achieving a high performance, with a MAE of 0.0068. Finally, we have also shown that partitioning the latent space allowed our model to learn biological features, without showing them explicitly to the model.

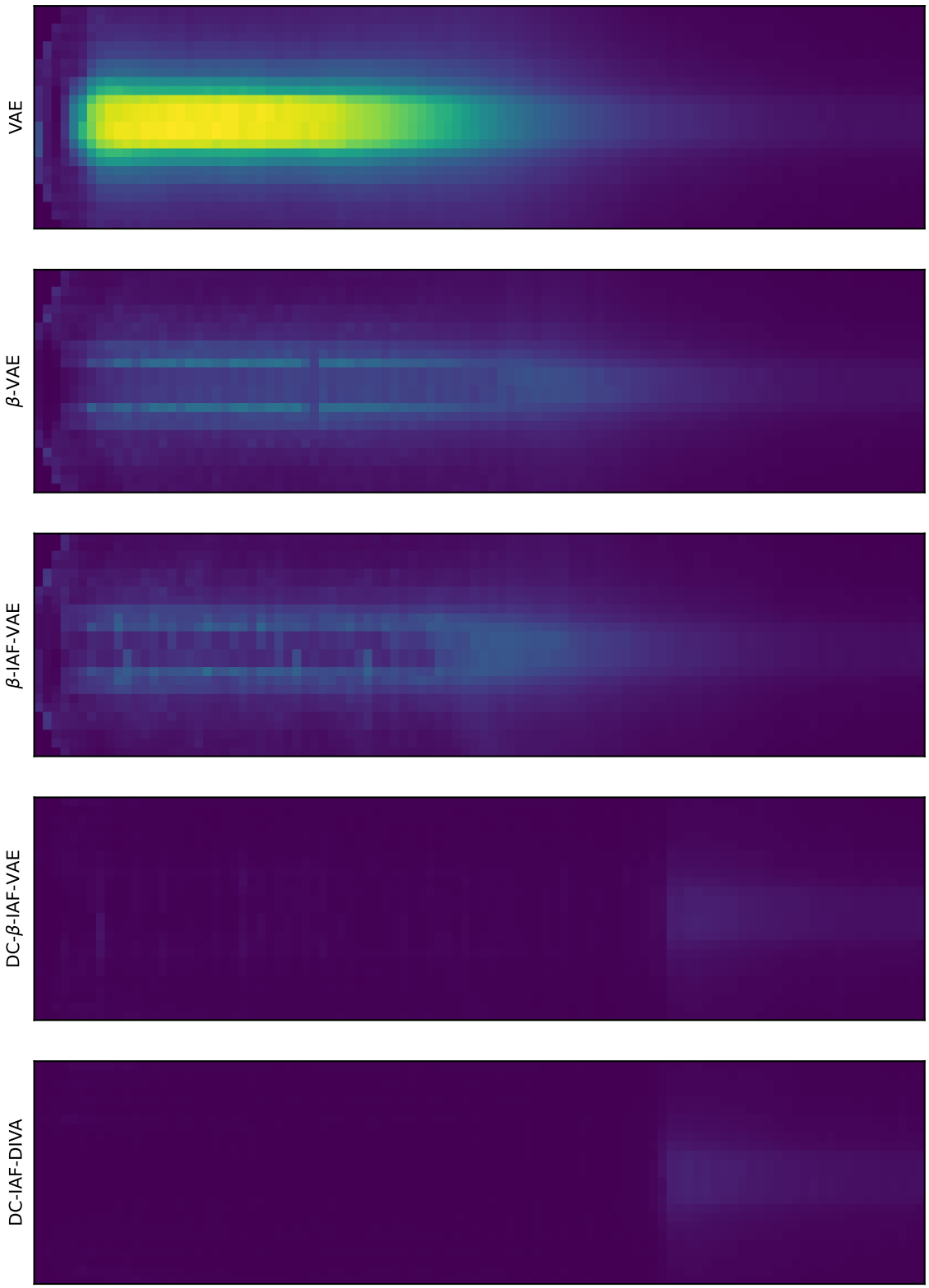


Figure 6.1: Distribution of errors for the different models. The brightest yellow indicates that in 64% of the cases, the reconstruction of that pixel was wrong.

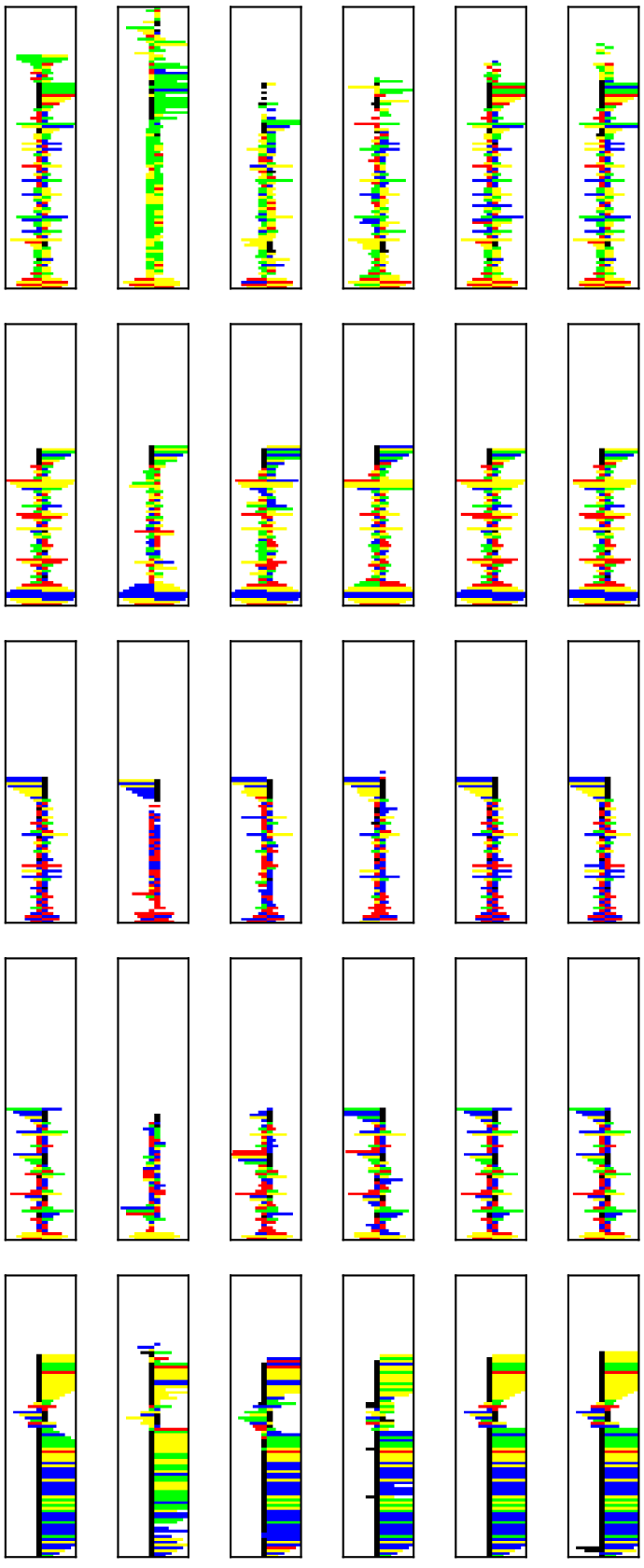
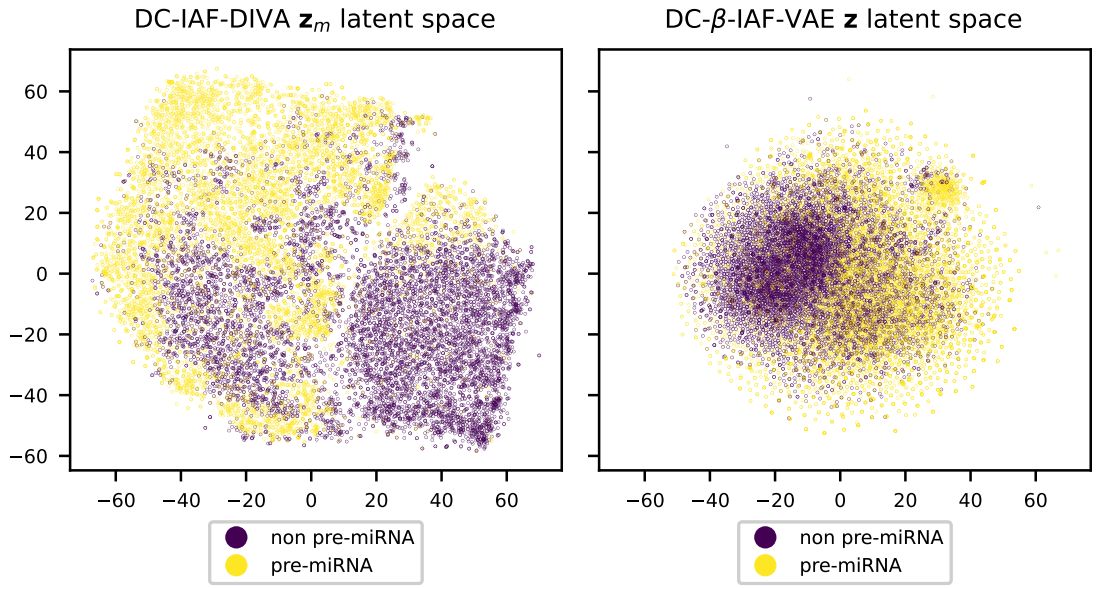
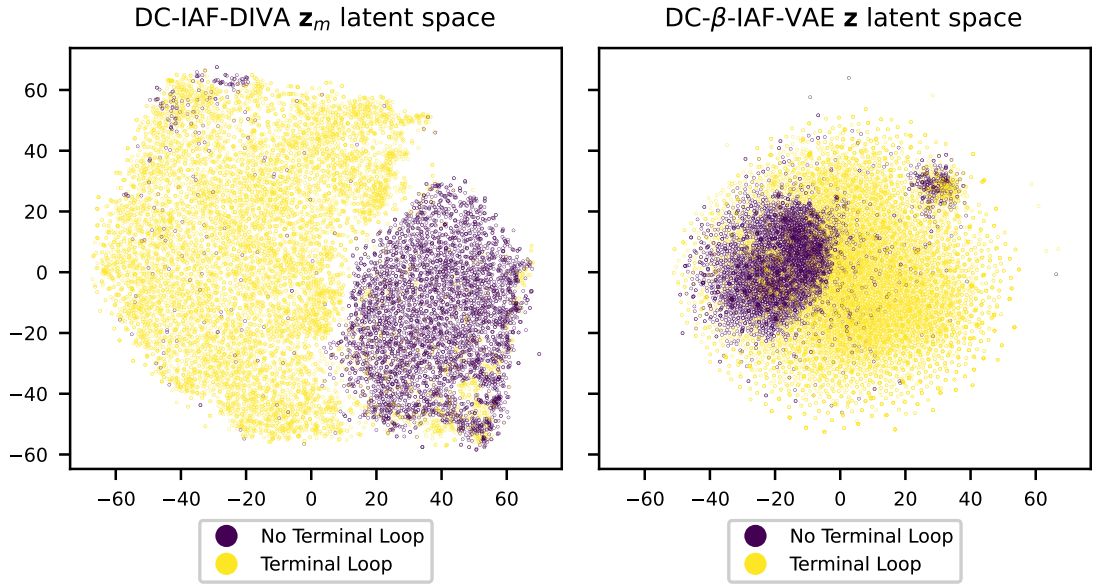


Figure 6.2: Reconstructions of 5 random samples from the test set using each model variant. From top to bottom: original, VAE, β-VAE, β-IAF-VAE, DC-β-IAF-VAE, DC-IAF-DIVA.

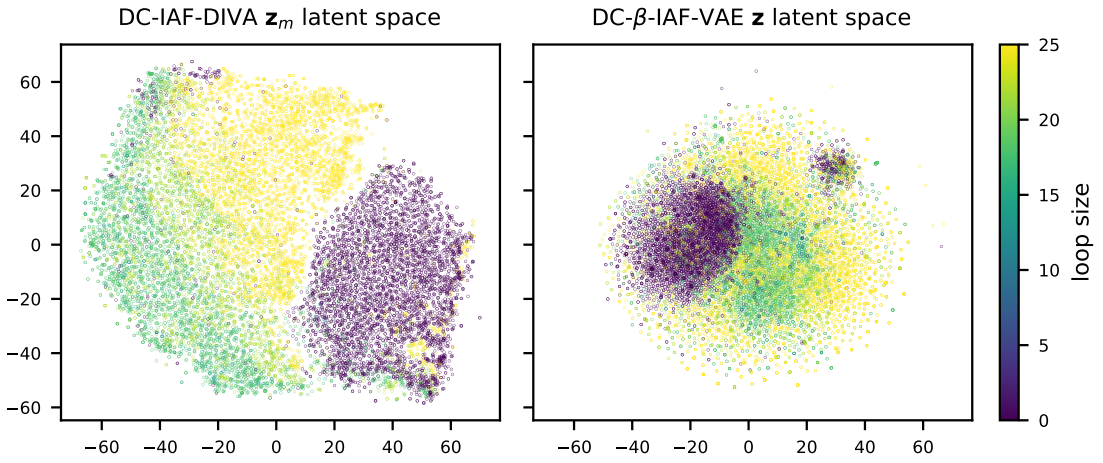


(a) Class values

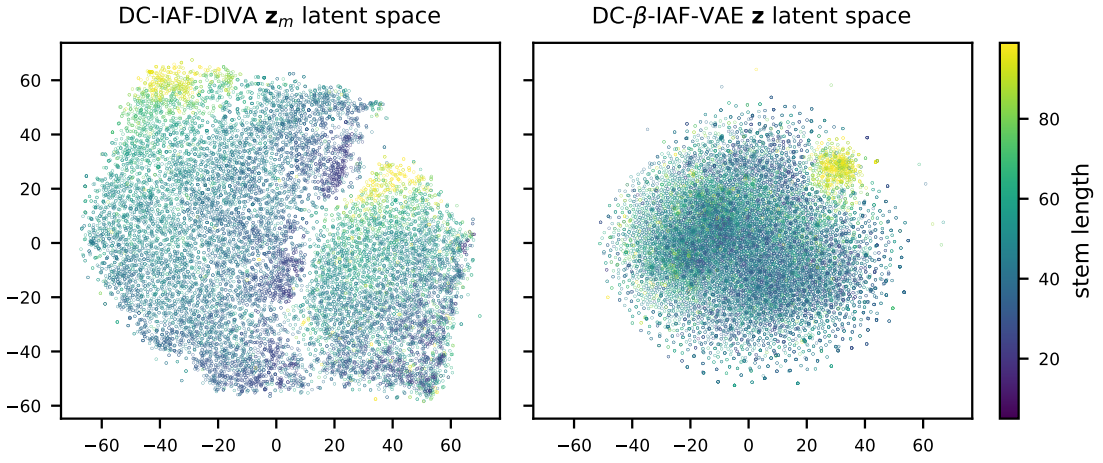


(b) Presence Terminal Loop

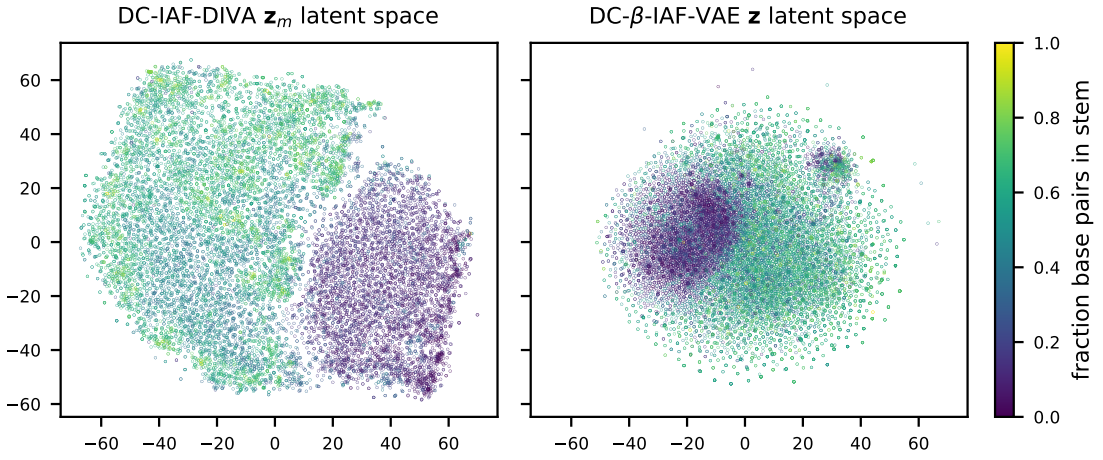
Figure 6.3: Distribution of different biological concepts over the entire latent space of DC-β-IAF-VAE and $\mathbf{z}_m$ of DC-IAF-DIVA. For labels with continuous values, the colorbar to the right of the figures indicates which color corresponds to which value.



(c) Terminal Loop Size



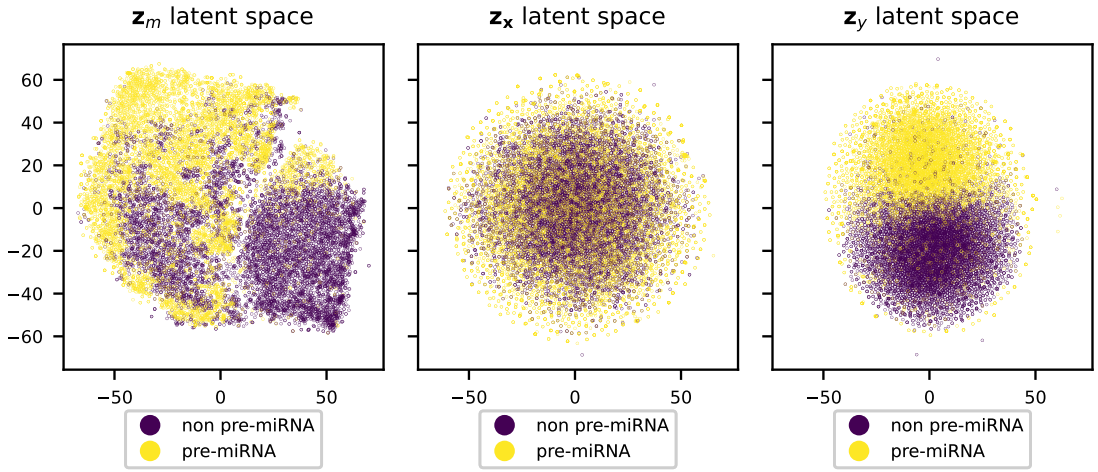
(d) Stem Length



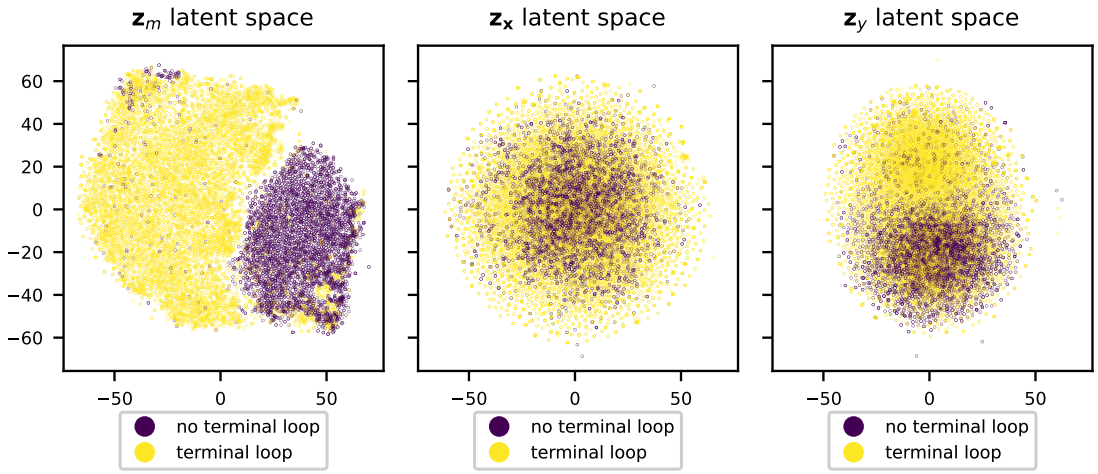
(e) Fraction Base Pairs in Stem

Figure 6.3: Distribution of different biological concepts over the entire latent space of DC-β-IAF-VAE and $\mathbf{z}_m$ of DC-IAF-DIVA. For labels with continuous values, the colorbar to the right of the figures indicates which color corresponds to which value.

6.2 Latent Space Analysis



(a) Class Values

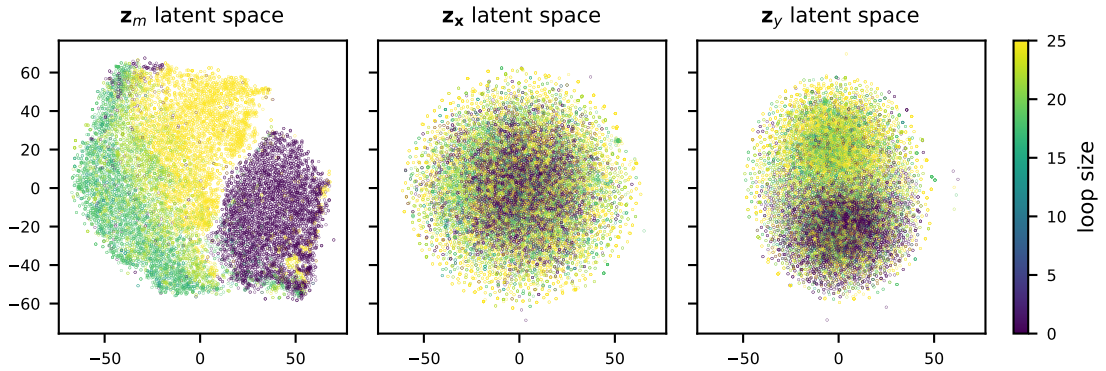


(b) Presence Terminal Loop

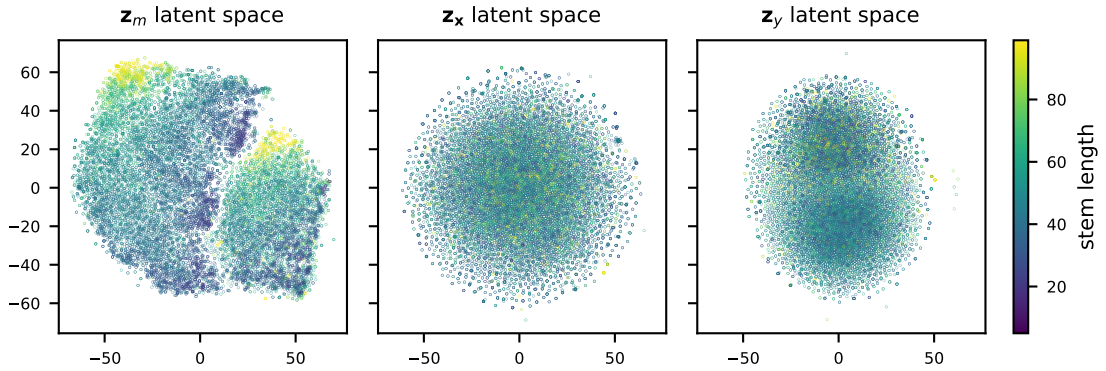
Figure 6.4: Partitioned latent spaces colored by class labels and other concepts following dimensionality reduction using t-SNE. For labels with continuous values, the colorbar to the right of the figures indicates which color corresponds to which value.

To investigate whether the DIVA model learned relevant latent spaces, we performed dimensionality reduction on all 3 latent spaces of the model. In Figure 6.4, visualizations of the 3 different latent spaces can be found after applying t-SNE and coloring the datapoints by their class label as well as different concepts. As expected, we see clear separations between classes in $\mathbf{z}_y$ and $\mathbf{z}_m$, indicating that for both latent spaces the model seemed to have learned a relevant representation. On the other hand, there does not seem to be any separation between class labels in $\mathbf{z}_\mathbf{x}$. From this, we can assume that the model has indeed used this latent space to capture any remaining variance in the data.

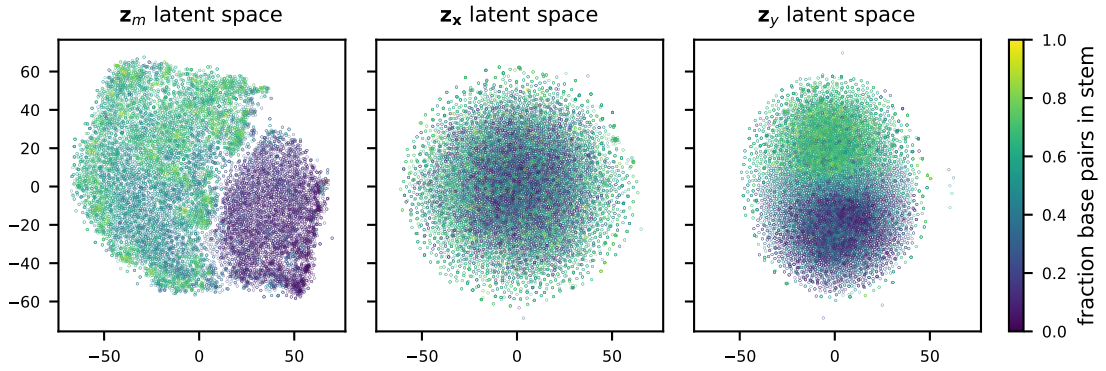
We can observe similar patterns for the different biological concepts. In the $\mathbf{z}_m$ latent space, there seems to be a clear distribution of all the different concepts over the latent space. In the latent space for $\mathbf{z}_y$, we see that the distribution of the different concept values over the latent



(c) Terminal Loop Size



(d) Stem Length



(e) Fraction Base Pairs in Stem

Figure 6.4: Partitioned latent spaces colored by class labels and other concepts following dimensionality reduction using t-SNE. For labels with continuous values, the colorbar to the right of the figures indicates which color corresponds to which value.

space is not as clearly separated. This is likely due to no concept having a full correlation with class labels, which likely did not force the model to separate these concepts in this latent space. Finally, we see that the latent space for the remaining variance in the data does not capture these concepts to any degree, showing that it indeed captures remaining variance.

When investigating the quality of the latent space, there are often no quantitative metrics

which can be used. Interpolating between pairs of points in the latent space is a possible qualitative measure which can be used instead. If the reconstructions of the interpolated points show a smooth transition, it indicates that the probability mass is not centered exclusively around training datapoints and that the learned latent factors generalize well [47, 19].

To make interpolations, we sampled five pairs of datapoints, which we put through the encoders and IAFs to obtain the latent representation of each datapoint. Then, we linearly interpolated between each pair, and decoded multiple steps from the interpolation. The results can be seen in Figure 6.5. In this figure, the top and bottom rows represent the original images, while the images second from top and bottom represent their reconstructions. During the interpolation, the images slowly change from the initial image to the other image, suggesting that the coverage of the data manifold is good. However, some steps result in images which are impossible, by either containing gaps (column 4, row 4 & 5) or by containing bar lengths which should not be possible (column 5, row 5 & 6).

These impossible images could be caused by (slight) gaps in the latent manifold. Due to the large latent space of 192 variables, it is possible that some combinations of variables were never learned by the model. When an interpolation passes through one of these gaps, it can result in producing unrealistic reconstruction. However, this does not imply that the model has a poor performance. Another explanation for the failed reconstructions of the interpolations is the decoder not being perfect. In the final column, the reconstruction of the second picture contains an error after the terminal loop (red rectangle). We see a similar error in the reconstruction of the interpolations. Likewise, the actual gaps appearing in the interpolation of column 4 can also be the results of the model struggling with reconstructing longer RNA strands.

Another qualitative measure for the latent space is the models' performance in conditional generation [47]. Here, for a datapoint from the test set, the latent space representation is calculated. Then, we conditioned on (a part of) the latent space, and investigated if the reconstructions indeed follow the conditioning.

To investigate the conditional generation, we analyzed how reconstructions were changing by modifying the MFE of the molecule. We sampled a random RNA from the test set, and passed it through the encoders to obtain $\mathbf{z_x}$ and $\mathbf{z}_y$. In addition, we defined multiple changes to the base MFE of the RNA. To sample $\mathbf{z}_m$, we passed the MFE through the conditional prior network, from which we sampled multiple times, where we once sampled the mean of the prior (first column), and three times randomly from the prior (second to fourth column).

In Figure 6.6, we have the original RNA image in the top left. In the row below, we have the reconstruction of the original RNA by sampling from the prior. In the rows below, we have conditionally generated RNAs, where we condition on the MFE. In these pictures, the conditional MFE is visualized below each generated picture. We can see that the change of MFE is reflected in the pictures. For example, the MFE for the RNA in the third row was modified such that the first strong bond appears later. We can see that this change in MFE is indeed reflected in the reconstructions, since the terminal loop grew bigger and a part of the asymmetric bulge disappeared. Similarly, in the RNA in the sixth row, the asymmetric bulge completely disappeared, as we conditioned on a lot of strong bonds early in the molecule. On the other hand, in the RNAs in the fifth and last row, we see that the molecules increased in length. However, the model seems to struggle with making the "new" extension of the RNA look realistic. This can possibly be caused by the model in general not being able to reconstruct the end of long molecules well, due to a lack of these in the dataset.

Since these reconstructions are adapting as expected when changing the MFE, we can conclude that the model understands the significance of $\mathbf{z}_m$ well. While some of the changes are not perfect, this could again be explained by a gap in the manifold of the latent space. Potentially, this issue

could be alleviated by increasing the regularization parameter (β) of this part of the latent space, to ensure the distribution sampled from the encoder is closer to the prior.

Through the analysis on the latent space, we have seen that our DIVA model has learned a meaningful representation of the latent space for pre-miRNA. We have seen that several relevant pre-miRNA characteristics seem aligned with the latent space (Figure 6.3), which implies that the model has been successful in learning generative factors of (non) pre-miRNA. In addition, we have seen that the model learned the influence of the MFE on the structure well, as, when changing the MFE, the newly sampled molecules look as expected.

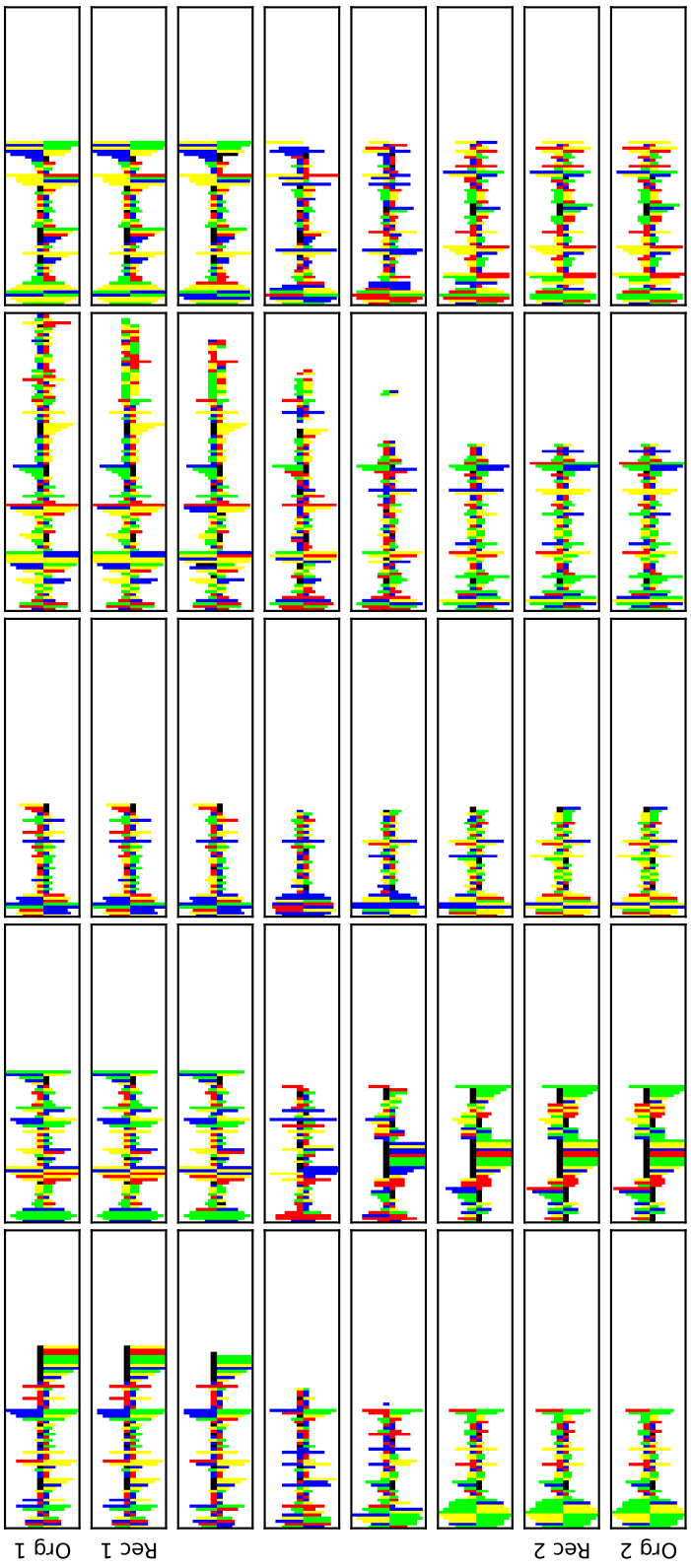


Figure 6.5: Linear Interpolation of the latent space. The top and bottom row represent the original images. The second rows from top and bottom represent their reconstructions. The points in between were interpolated, with each step having an equal size.

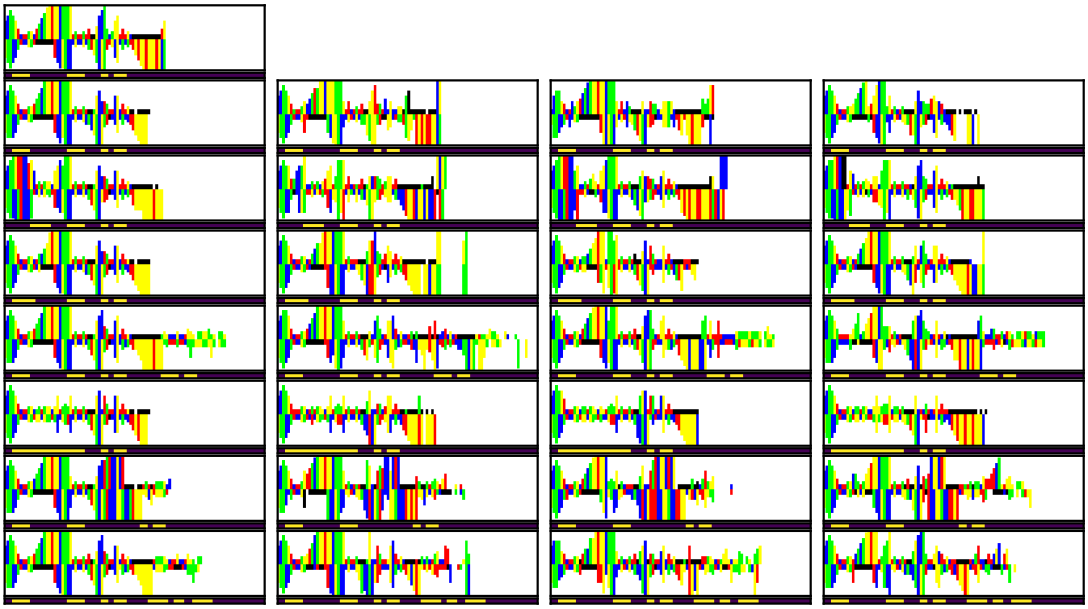


Figure 6.6: Reconstructions by sampling from the prior of the MFE. Colored bars below images represent their MFE, with 1 being yellow. Top left image is the original. The second row contains reconstructions where we sampled from the original prior of $\mathbf{z}_m$. Rows below contain conditional samples after changing the MFE.

6.3 Interpretable Latent Space

As we have seen previously, the latent space for the MFE ($\mathbf{z}_m$) was organized well with regards to the different concepts we checked for. We trained the Decision Tree on the latent space $\mathbf{z}_m$ of the training set. From this, we obtained the tree in Figure 6.7. Despite setting the maximum depth to 4, the Decision Tree only reached a depth of 3. We also plotted the approximate decision boundaries on the latent space following dimensionality reduction by t-SNE. We applied 3-nearest neighbors classifying on the classifications from the tree rules to obtain the boundary, visible in Figures 6.8, 6.9, 6.10.

On the training set, the tree achieved an accuracy of 93.9%, while on the testing set it achieved an accuracy of 91.2%. In addition, the tree achieved a sensitivity of 0.923 and a specificity of 0.955 on the test set. Meanwhile, the auxiliary classifier for y on $\mathbf{z}_m$ obtained a training accuracy of 95.9% and a testing accuracy of 92.2%. As a result, we can see that despite the minor drop in accuracy, the explainabilty increases massively. Furthermore, the drop in accuracy is also fairly minor compared to the concept whitening method [69], which achieved a testing accuracy of 92.4%.

The very first split the Decision Tree makes is based on the fraction of the amount of base pairs in the stem. This is indeed a biologically relevant characteristic of pre-miRNA [2], as the dicer enzyme is more effective on pre-miRNAs containing few bulges. Similarly, short RNA sequences with a stem having length of less than 70 and less than 50% base pairs in the stem, are unlikely to be pre-miRNA. This can again be explained by the dicer enzyme being more effective on RNAs containing mainly base pairs in the stem, since (in absolute values) the stem will have few base pairs in this case. We then see that RNAs which contain asymmetric bulges, are more likely to be non pre-miRNA. This is in line with pre-miRNA typically not having asymmetric bulges [36].

In terms of accuracy of the rules, the SVM used for making the first split had an accuracy of 92.7% on the test set. The second rule had an accuracy of 98.9%, and the final rule had an accuracy of 90.4%. Since the rules achieved a high accuracy, we can safely conclude that the rules learned by the Decision Tree are relevant and correct.

As we have seen, our Decision Tree method has successfully made an interpretable method for classifying pre-miRNA using the latent space of our model. The method achieves an accuracy slightly lower than non-interpretable methods (auxiliary classifier) and methods with limited interpretability (concept whitening [69]). The method learned decision rules which are in line with domain knowledge and has a high confidence in these rules. In turn, the method could be used in other domains for interpretable classifications using the latent space.

However, the biological features used for training the model should be evaluated with a domain expert, as the features we are using might not be optimal. Furthermore, the dataset should also be evaluated, as parts of the dataset do not consist of naturally occurring RNA, but pre-miRNA of which the nucleotide order is shuffled. This can affect the latent space learned by our model, which in turn can affect the performance of our algorithm.

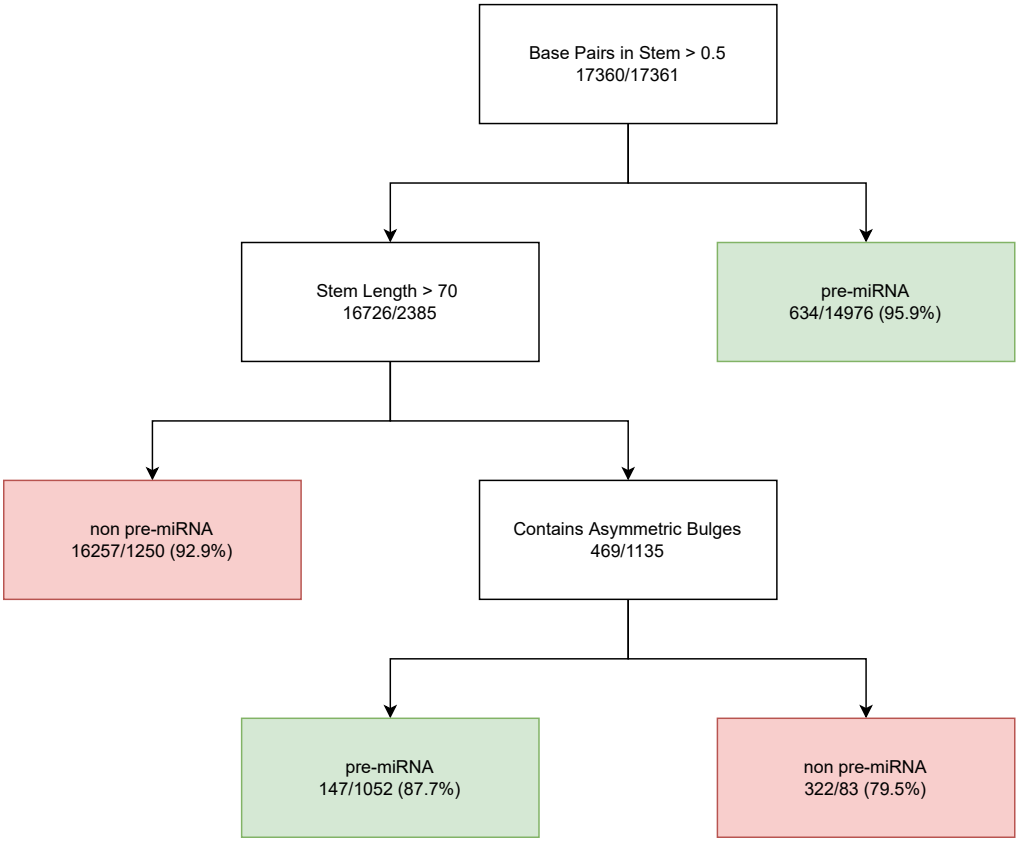


Figure 6.7: Decision Tree trained on the $\mathbf{z}_m$. First number at each node represents the amount of pre-miRNA at each node, the second number the amount of non pre-miRNA. The number in percentages represents the accuracy at each leaf, while a red leaf indicates the non pre-miRNA is predicted, while a green leaf indicates that pre-miRNA is predicted.

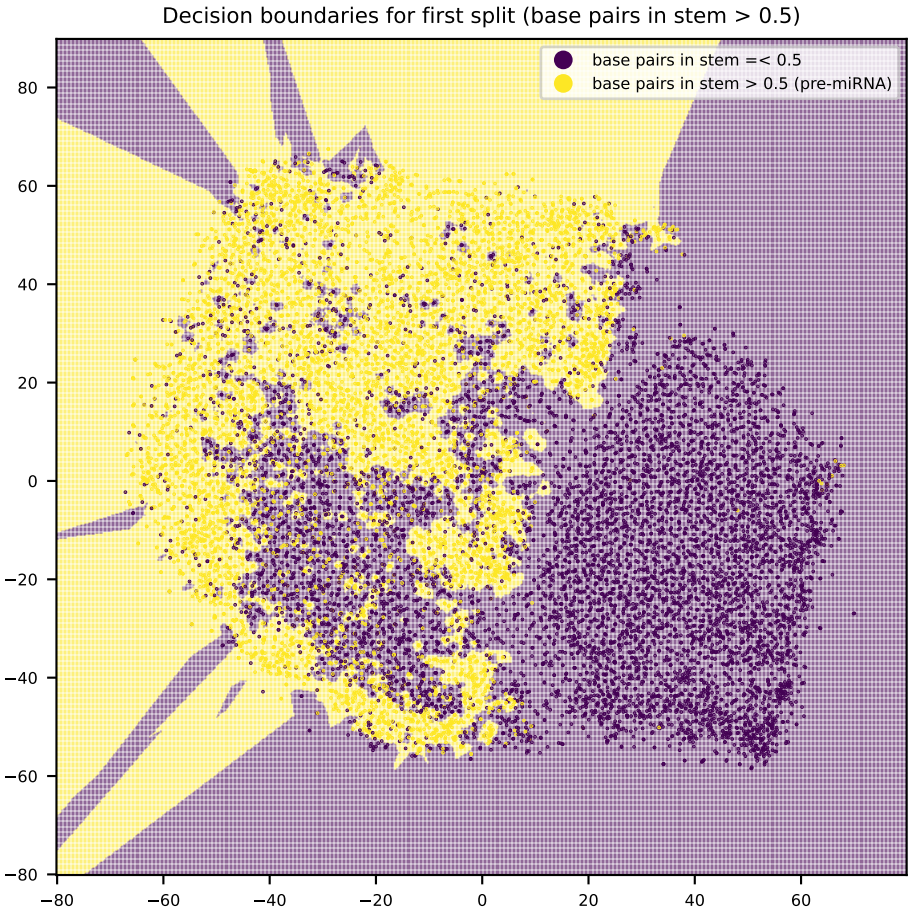


Figure 6.8: Approximate decision boundary of the first split visualized on the test set.

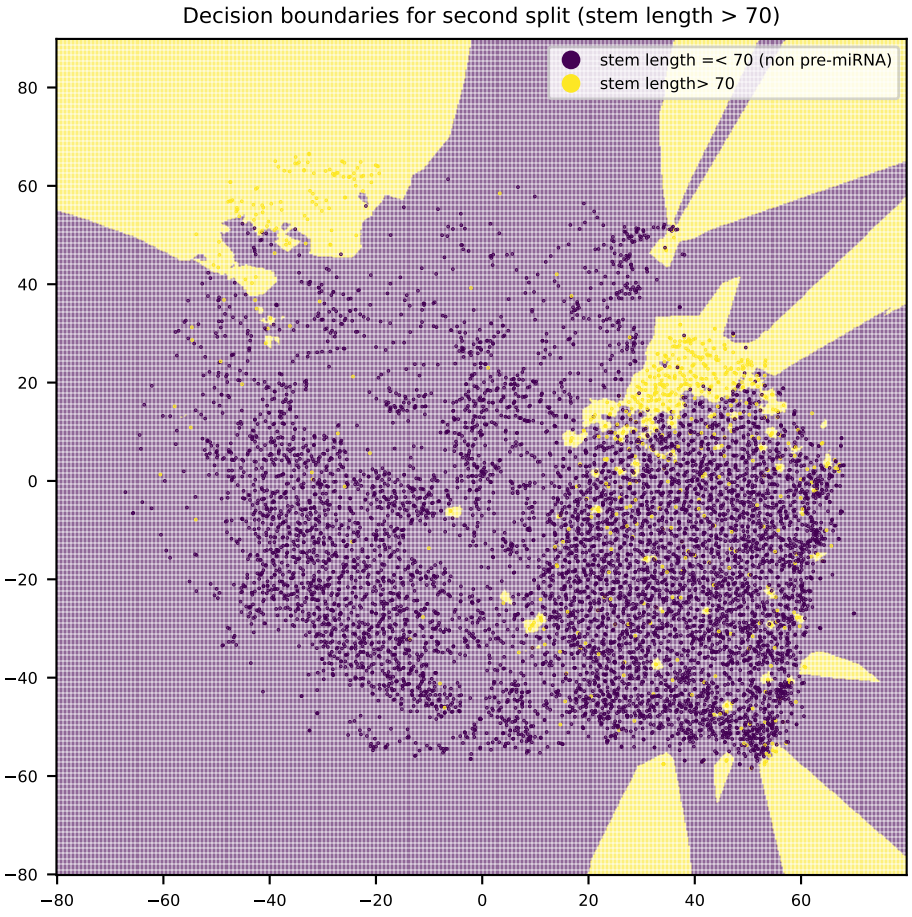


Figure 6.9: Approximate decision boundary of the second split visualized on the test set.

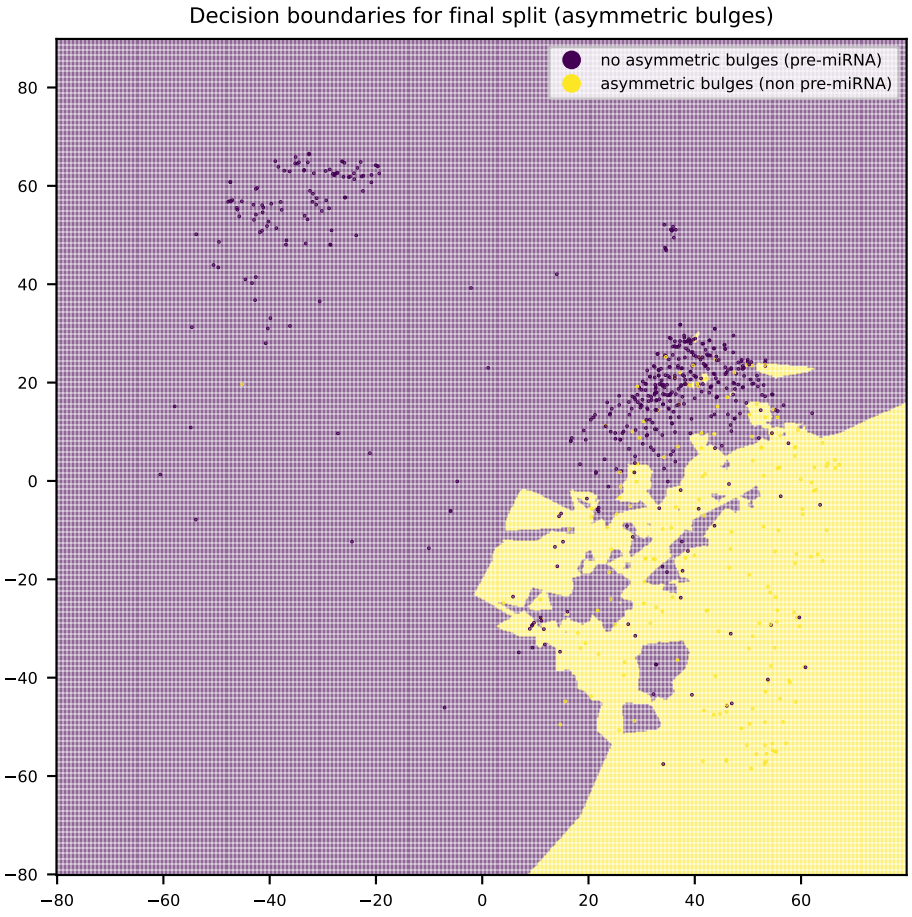


Figure 6.10: Approximate decision boundary of the third split visualized on the test set.

Chapter 7

Conclusions

As we have shown in Table 6.2 and Figure 6.1, the DC-IAF-DIVA model has managed to obtain near perfect reconstruction performance with an MAE of less than 0.01, by achieving an MAE of 0.0068. In addition, we have also shown that the latent space of the model is well organized, by investigating how the classes and different concepts are distributed over the latent space. From this, we can conclude that our model has achieved a good perfomance in modelling pre-miRNA.

Furthermore, our model is able to interpolate between different pairs of RNAs (Figure 6.5), with decent looking interpolations. While some of these interpolations are not realistic or possible, more attention should be paid to whether the decoder or incomplete coverage of the data manifold is the cause of some errors in interpolations. Overall, the model seems to have a decent sampling performance. It should be noted that, to be used for sampling in a practical context, the model should likely be trained with a higher constraint on the latent spaces.

In addition, we have shown that we can conditionally sample new RNA strands by manipulating the MFE (Figure 6.6) and that the new samples are indeed conditioned by the MFE we provided. In turn, this led to a latent space that was organized well and could thus be used for developing our framework for making interpretable predictions.

Finally, it has been demonstrated that our framework for interpretable predictions on the latent space works well. The Decision Tree we trained managed to achieve a performance (91.2% accuracy, 0.923 sensitivity, 0.955 specificity) comparable to the state-of-the-art for interpretable pre-miRNA classification (92.4% accuracy [69]), while having a higher degree of (potential) interpretability. Furthermore, the description of pre-miRNA developded by the framework is based on biological features which are known to be be properties of (non) pre-miRNA. However, the method should still be evaluated with a domain expert, to gain more insight into which biological characteristics should be used for training the Decision Tree. Due to the high performance of our method, we are certain that it has potential to be used for interpretable predictions for other domains.

In this thesis, we have shown that deep generative methods can be used to model pre-miRNA to uncover generative factors and potentially obtain new conditioned samples. We have also presented a novel method which makes interpretable classifications on the latent space using a Decision Tree. It achieved a high performance on the pre-miRNA classification task, and could be extended to different domains.

7.1 Limitations & Future Work

The main limitation of this work was (computational) time. For this reason, an extensive hyper parameter search for the best performing model was not possible. For example, the architecture of the encoders and decoders could have been further optimized. Furthermore, different kernel sizes could be used with an inception-like network containing wide and tall rectangular kernels. In addition, the parameters surrounding the latent space could also be further optimized. Here, the constraints on the latent space could be further investigated, by performing more experiments with different and higher βs for each latent space. Another part which could have been further investigated are the sizes of each partition of the latent space, and whether all partitions are required. However, given the performance displayed by the model, we are likely not far off from the best possible architecture.

Another potential limitation of our proposed model is the use of the RNA image encoding. A main drawback is that the encoding itself contains a lot of white space, which should be ignored by the model as it does not carry any information. Almost 80% of the pixels of an image is on average white, and thus irrelevant information. On the other hand, in the case of some RNAs the image is either too short or not tall enough, leading to the image representation of the RNA being cut off.

A possible solution is to represent the RNA as a graph instead of an image. In this case, each nucleotide could be represented by a node in the graph, having connections to its two adjacent nodes. To represent the bonds, (different) connections could be made between these nucleotides. Furthermore, the position of each node can be represented in its feature vector, by using the (actual) height of bars provided by the image encoding. By representing the RNA as graphs, we could make use of Graph Convolutional Networks (GCN) [41]. Using GCN, we could then build (variations) of a Variational Graph Auto-Encoder (VGAE) [42] to model the pre-miRNA.

Another potential approach of modelling pre-miRNA is modelling the mRNA targets simultaneously with the (pre-)miRNA. A potential approach could be a (Variational) Auto-Encoder which takes as input the target mRNA and needs to produce multiple potential miRNAs which could target the mRNA, or vice-versa. By modeling the target mRNAs simultaneously as the miRNA, we could not only identify potential novel miRNA, but also directly identify the mRNA which it can affect.

Finally, our method for making interpretable predictions on the latent space should be investigated further, and its performance should be verified on a different dataset. Further work could also be done on the method itself. For example, a different approach could be taken instead of defining thresholds for variables with many values, by using a regression method instead of a classifier for these variables. In addition, further investigation could be done in using different secondary classifiers instead of SVMs with linear kernels.

Bibliography

[1] Julia Alles, Tobias Fehlmann, Ulrike Fischer, Christina Backes, Valentina Galata, Marie Minet, Martin Hart, Masood Abu-Halima, Friedrich A Grässer, Hans-Peter Lenhof, et al. An estimate of the total number of true human mirnas. *Nucleic acids research*, 47(7):3353–3364, 2019. 1

[2] Jens Allmer. Computational and bioinformatics methods for microrna gene prediction. In *MiRNomics: MicroRNA biology and computational analysis*, pages 157–175. Springer, 2014. 7, 8, 49

[3] Çiğir Biray Avci and Yusuf Baran. Use of micrornas in personalized medicine. In *miRNomics: MicroRNA Biology and Computational Analysis*, pages 311–325. Springer, 2014. 1

[4] Youhuang Bai, Xiaozhuan Dai, Andrew Harrison, Caroline Johnston, and Ming Chen. Toward a next-generation atlas of rna secondary structure. *Briefings in bioinformatics*, 17(1):63–77, 2016. 22

[5] Rukshan Batuwita and Vasile Palade. micropred: effective classification of pre-mirnas for human mirna gene prediction. *Bioinformatics*, 25(8):989–995, 2009. 22

[6] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001. 12, 20

[7] Leo Breiman, Jerome H Friedman, Richard A Olshen, and Charles J Stone. *Classification and regression trees*. Routledge, 2017. 12

[8] Christopher P Burgess, Irina Higgins, Arka Pal, Loic Matthey, Nick Watters, Guillaume Desjardins, and Alexander Lerchner. Understanding disentangling in beta-vae. *arXiv preprint arXiv:1804.03599*, 2018. 17, 22

[9] Ricky TQ Chen, Xuechen Li, Roger B Grosse, and David K Duvenaud. Isolating sources of disentanglement in variational autoencoders. *Advances in neural information processing systems*, 31, 2018. 22

[10] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. *Advances in neural information processing systems*, 29, 2016. 21

[11] Zhi Chen, Yijie Bei, and Cynthia Rudin. Concept whitening for interpretable image recognition. *Nature Machine Intelligence*, 2(12):772–782, 2020. 21

[12] Jorge Cordero, Vlado Menkovski, and Jens Allmer. Detection of pre-microrna with convolutional neural networks. *bioRxiv*, 2019. 24

[13] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995. 11

[14] Jorge Alberto Cordero Cruz. Detection of pre-micrornas with convolutional neural networks. Master's thesis, Eindhoven University of Technology, 2019. 2, 4, 23, 24, 26

[15] Bala Gür Dedeoğlu. High-throughput approaches for microrna expression analysis. In *miRNomics: microRNA biology and computational analysis*, pages 91–103. Springer, 2014. 1, 9

[16] Jiandong Ding, Shuigeng Zhou, and Jihong Guan. Mirensvm: towards better prediction of microrna precursors using an ensemble svm classifier with multi-loop features. *BMC bioinformatics*, 11(11):1–10, 2010. 22

[17] Binh Thanh Do, Vladimir Golkov, Göktuğ Erce Gürel, and Daniel Cremers. Precursor microrna identification using deep convolutional neural networks. *BioRxiv*, page 414656, 2018. 2, 23

[18] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011. 13

[19] Vincent Dumoulin, Ishmael Belghazi, Ben Poole, Olivier Mastropietro, Alex Lamb, Martin Arjovsky, and Aaron Courville. Adversarially learned inference. *arXiv preprint arXiv:1606.00704*, 2016. 46

[20] Ayse Elif Erson-Bensan. Introduction to microrna in biological systems. *MiRNomics: MicroRNA biology and computational analysis*, pages 1–14, 2014. 6, 7, 8

[21] Bastian Fromm, Diana Domanska, Eirik Høye, Vladimir Ovchinnikov, Wenjing Kang, Ernesto Aparicio-Puerta, Morten Johansen, Kjersti Flatmark, Anthony Mathelier, Eivind Hovig, Michael Hackenberg, Marc R Friedländer, and Kevin J Peterson. MirGeneDB 2.0: the metazoan microRNA complement. *Nucleic Acids Research*, 48(D1):D132–D141, 10 2019. 24

[22] Victoria Furer, Jeffrey D Greenberg, Mukundan Attur, Steven B Abramson, and Michael H Pillinger. The role of microrna in rheumatoid arthritis and other autoimmune diseases. *Clinical immunology*, 136(1):1–15, 2010. 8

[23] Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. Made: Masked autoencoder for distribution estimation. In *International conference on machine learning*, pages 881–889. PMLR, 2015. 18

[24] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020. 15

[25] Sam Griffiths-Jones. The microrna registry. *Nucleic acids research*, 32(suppl_1):D109–D111, 2004. 24

[26] Adam Gudyś, Michał Wojciech Szcześniak, Marek Sikora, and Izabela Makałowska. Huntmi: an efficient and taxon-specific approach in pre-mirna identification. *BMC bioinformatics*, 14(1):1–10, 2013. 24

[27] Josie Hayes, Pier Paolo Peruzzi, and Sean Lawler. Microrna in cancer: biomarkers, functions and therapy. *Trends in molecular medicine*, 20(8):460–469, 2014. 1, 9

[28] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 15

[29] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-VAE: Learning basic visual concepts with a constrained variational framework. In *International Conference on Learning Representations*, 2017. 17, 22

[30] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012. 14

[31] Ivo L Hofacker, Walter Fontana, Peter F Stadler, L Sebastian Bonhoeffer, Manfred Tacker, and Peter Schuster. Fast folding and comparison of rna secondary structures. *Monatshefte für Chemie/Chemical Monthly*, 125(2):167–188, 1994. 22

[32] Weili Huang. Micrornas: biomarkers, diagnostics, and therapeutics. *Bioinformatics in MicroRNA research*, pages 57–67, 2017. 1, 9

[33] Maximilian Ilse, Jakub M Tomczak, Christos Louizos, and Max Welling. Diva: Domain invariant variational autoencoders. In *Medical Imaging with Deep Learning*, pages 322–348. PMLR, 2020. 17

[34] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015. 14

[35] Peng Jiang, Haonan Wu, Wenkai Wang, Wei Ma, Xiao Sun, and Zuhong Lu. Mipred: classification of real and pseudo microrna precursors using random forest prediction model with combined features. *Nucleic acids research*, 35(suppl_2):W339–W344, 2007. 22

[36] Wenjing Kang and Marc R Friedländer. Computational prediction of mirna genes from small rna sequencing data. *Frontiers in bioengineering and biotechnology*, 3:7, 2015. 49

[37] V Narry Kim. Microrna precursors in motion: exportin-5 mediates their nuclear export. *Trends in cell biology*, 14(4):156–159, 2004. 2, 22

[38] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 13

[39] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013. 16

[40] Durk P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improved variational inference with inverse autoregressive flow. *Advances in neural information processing systems*, 29, 2016. 18, 26

[41] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016. 55

[42] Thomas N Kipf and Max Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016. 55

[43] Ana Kozomara, Maria Birgaoanu, and Sam Griffiths-Jones. mirbase: from microrna sequences to function. *Nucleic acids research*, 47(D1):D155–D162, 2019. 1

[44] Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998. 21

[45] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Large-scale celebfaces attributes (celeba) dataset. *Retrieved August*, 15(2018):11, 2018. 38

[46] David H Mathews, Matthew D Disney, Jessica L Childs, Susan J Schroeder, Michael Zuker, and Douglas H Turner. Incorporating chemical modification constraints into a dynamic programming algorithm for prediction of rna secondary structure. *Proceedings of the National Academy of Sciences*, 101(19):7287–7292, 2004. 24, 26

[47] Michael F Mathieu, Junbo Jake Zhao, Junbo Zhao, Aditya Ramesh, Pablo Sprechmann, and Yann LeCun. Disentangling factors of variation in deep representation using adversarial training. *Advances in neural information processing systems*, 29, 2016. 46

[48] Kwang Loong Stanley Ng and Santosh K Mishra. De novo svm classification of precursor microrna from genomic pseudo hairpins using global and intrinsic folding measures. *Bioinformatics*, 23(11):1321–1330, 2007. 1, 9, 22, 24

[49] Jacob O'Brien, Heyam Hayder, Yara Zayed, and Chun Peng. Overview of microrna biogenesis, mechanisms of actions, and circulation. *Frontiers in endocrinology*, 9:402, 2018. 1, 8

[50] Seunghyun Park, Seonwoo Min, Hyun-Soo Choi, and Sungroh Yoon. Deep recurrent neural network-based identification of precursor microrna. *Advances in Neural Information Processing Systems*, 30, 2017. 2, 23

[51] Pascal Paysan, Reinhard Knothe, Brian Amberg, Sami Romdhani, and Thomas Vetter. A 3d face model for pose and illumination invariant face recognition. In *2009 sixth IEEE international conference on advanced video and signal based surveillance*, pages 296–301. Ieee, 2009. 22

[52] J Ross Quinlan. *C4. 5: programs for machine learning*. Elsevier, 2014. 12

[53] Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *International conference on machine learning*, pages 1530–1538. PMLR, 2015. 16

[54] J Graham Ruby, Calvin H Jan, and David P Bartel. Intronic microrna precursors that bypass drosha processing. *Nature*, 448(7149):83–86, 2007. 7

[55] Müşerref Duygu Saçar and Jens Allmer. Machine learning methods for microrna gene prediction. In *miRNomics: MicroRNA Biology and Computational Analysis*, pages 177–187. Springer, 2014. 1, 9

[56] Müşerref Duygu Saçar Demirci, Jan Baumbach, and Jens Allmer. On the performance of pre-microrna detection algorithms. *Nature communications*, 8(1):1–9, 2017. 2, 22, 24

[57] Omer Sagi and Lior Rokach. Explainable decision forest: Transforming a decision forest into an interpretable tree. *Information Fusion*, 61:124–138, 2020. 20

[58] Tim Salimans and Durk P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *Advances in neural information processing systems*, 29, 2016. 14

[59] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision*, pages 618–626, 2017. 21

[60] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013. 21

[61] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 15

[62] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-first AAAI conference on artificial intelligence*, 2017. 15

[63] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015. 15

[64] Ryan J Taft, Ken C Pang, Timothy R Mercer, Marcel Dinger, and John S Mattick. Non-coding rnas: regulators of disease. *The Journal of Pathology: A Journal of the Pathological Society of Great Britain and Ireland*, 220(2):126–139, 2010. 6

[65] Abdulkadir Tasdelen and Baha Sen. A hybrid cnn-lstm model for pre-mirna classification. *Scientific reports*, 11(1):1–9, 2021. 2, 23

[66] Tijmen Tieleman, Geoffrey Hinton, et al. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012. 13

[67] Kemal Uğur Tüfekci, Ralph Leo Johan Meuwissen, and Şermin Genç. The role of micrornas in biological processes. *miRNomics: microRNA biology and computational analysis*, pages 15–31, 2014. 1, 8

[68] Kemal Uğur Tüfekci, Meryem Gülfem Öner, Ralph Leo Johan Meuwissen, and Şermin Genç. The role of micrornas in human diseases. In *miRNomics: MicroRNA biology and computational analysis*, pages 33–50. Springer, 2014. 1, 8, 9

[69] Irma Van den Brandt. Towards concept-based interpretability of pre-mirna detection using convolutional neural networks. Master's thesis, Eindhoven University of Technology, 2021. 2, 3, 33, 39, 49, 50, 54

[70] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008. 39

[71] Thibaut Vidal and Maximilian Schiffer. Born-again tree ensembles. In *International conference on machine learning*, pages 9743–9753. PMLR, 2020. 20

[72] Joost Visser, Alessandro Corbetta, Vlado Menkovski, and Federico Toschi. Stampnet: unsupervised multi-class object discovery. In *2019 IEEE International Conference on Image Processing (ICIP)*, pages 2951–2955. IEEE, 2019. 29, 30

[73] Jin Wang, Jinyun Chen, and Subrata Sen. Microrna as biomarkers and diagnostics. *Journal of cellular physiology*, 231(1):25–30, 2016. 1, 9

[74] Leyi Wei, Minghong Liao, Yue Gao, Rongrong Ji, Zengyou He, and Quan Zou. Improved and promising identification of human micrornas by incorporating a high-quality negative set. *IEEE/ACM transactions on computational biology and bioinformatics*, 11(1):192–201, 2013. 24

[75] Chenghai Xue, Fei Li, Tao He, Guo-Ping Liu, Yanda Li, and Xuegong Zhang. Classification of real and pseudo microrna precursors using local structure-sequence features and support vector machine. *BMC bioinformatics*, 6(1):1–7, 2005. 2, 22

[76] Jr-Shiuan Yang, Thomas Maurin, Nicolas Robine, Kasper D Rasmussen, Kate L Jeffrey, Rohit Chandwani, Eirini P Papapetrou, Michel Sadelain, Dónal O'Carroll, and Eric C Lai. Conserved vertebrate mir-451 provides a platform for dicer-independent, ago2-mediated microrna biogenesis. *Proceedings of the National Academy of Sciences*, 107(34):15163–15168, 2010. 7

[77] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014. 21

[78] Quanshi Zhang, Ying Nian Wu, and Song-Chun Zhu. Interpretable convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8827–8836, 2018. 21

[79] Xueming Zheng, Shungao Xu, Ying Zhang, and Xinxiang Huang. Nucleotide-level convolutional neural networks for pre-mirna classification. *Scientific reports*, 9(1):1–6, 2019. 2, 23

[80] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2921–2929, 2016. 21

Appendix A

Loss Curves

In Figures A.1, A.2 and A.3 the total, reconstruction and KL loss curves can be found. We can see that for most models, the KL loss curves for the training and testing data are mostly equal. On the other hand, we see that models using a deconvolutional decoder are able to achieve a better reconstruction performance than models with a fully connected decoder.

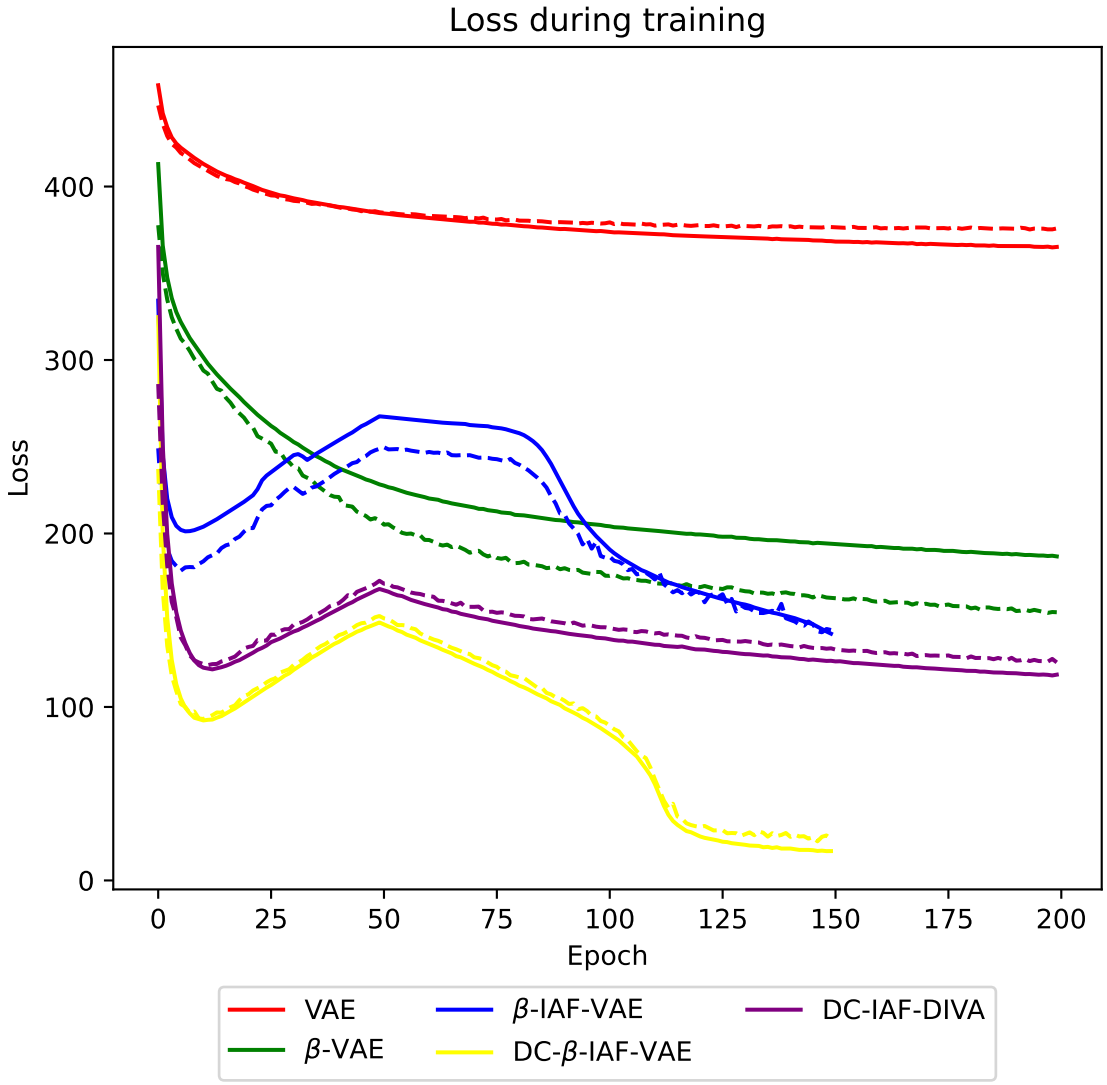


Figure A.1: Training and testing losses for different models at the end of each epoch. Dotted lines represent the test losses.

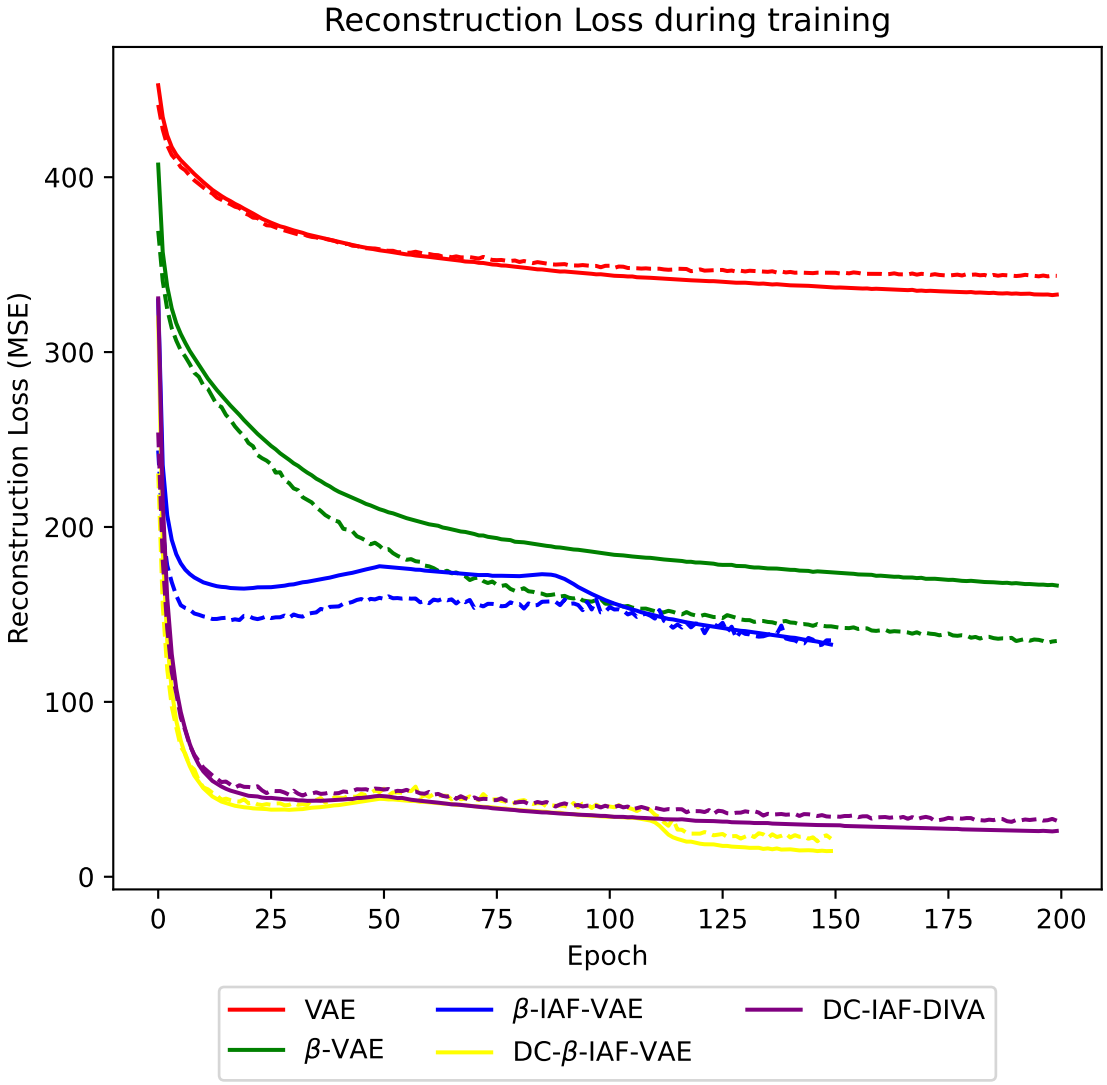


Figure A.2: Training and testing reconstruction losses for different models. Dotted lines represent the test losses.

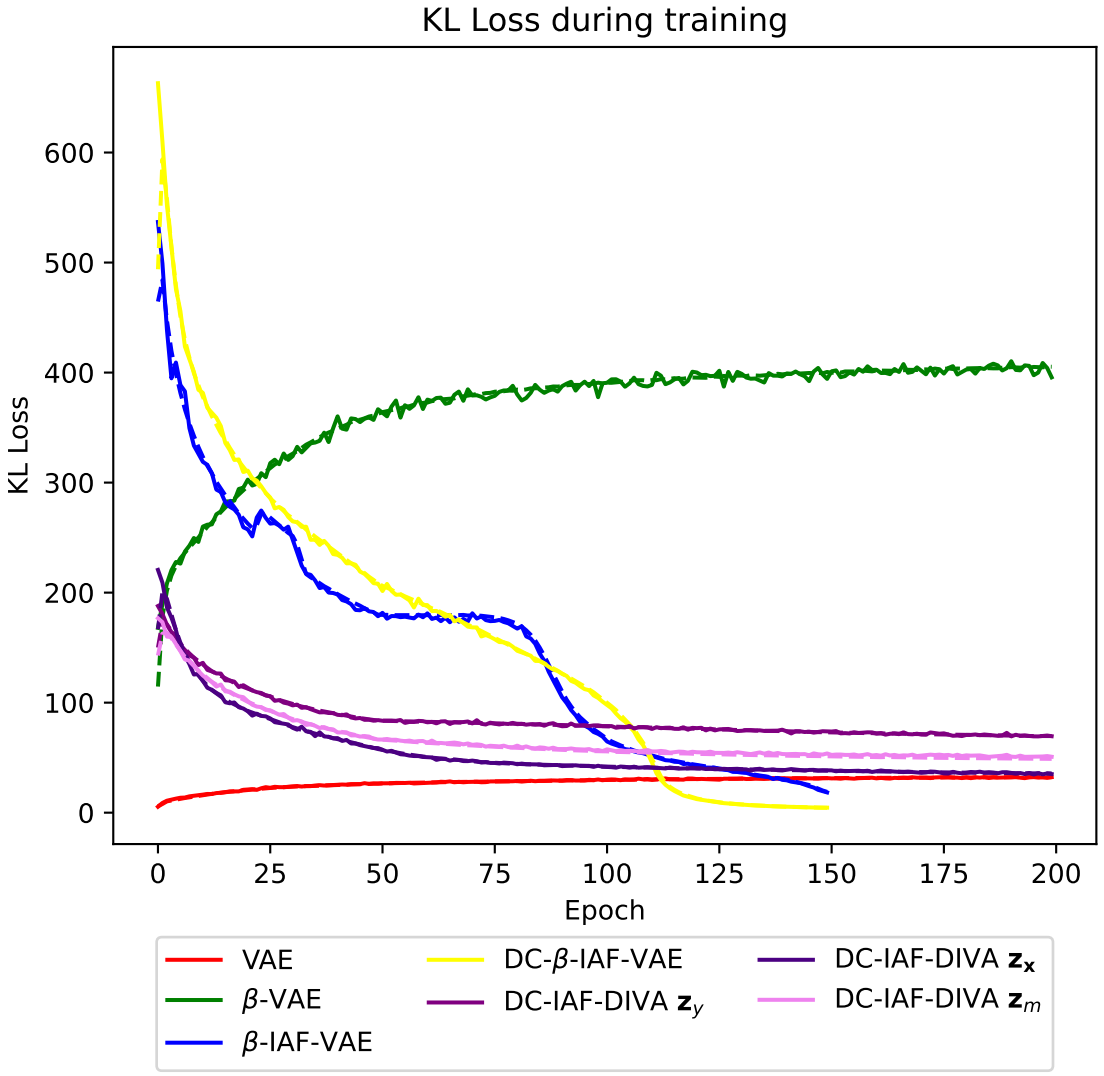


Figure A.3: Training and testing KL losses for different models. Dotted lines represent the test losses. Since these were mostly equal during training, the difference is not visible in most cases.

Appendix B

Additional Reconstructions

In this Appendix chapter, additional reconstructions from the different models can be found. In these reconstructions, it is again visible that DC-β-IAF-VAE and DC-IAF-DIVA obtain the best reconstructive performance, with near perfect reconstructions.

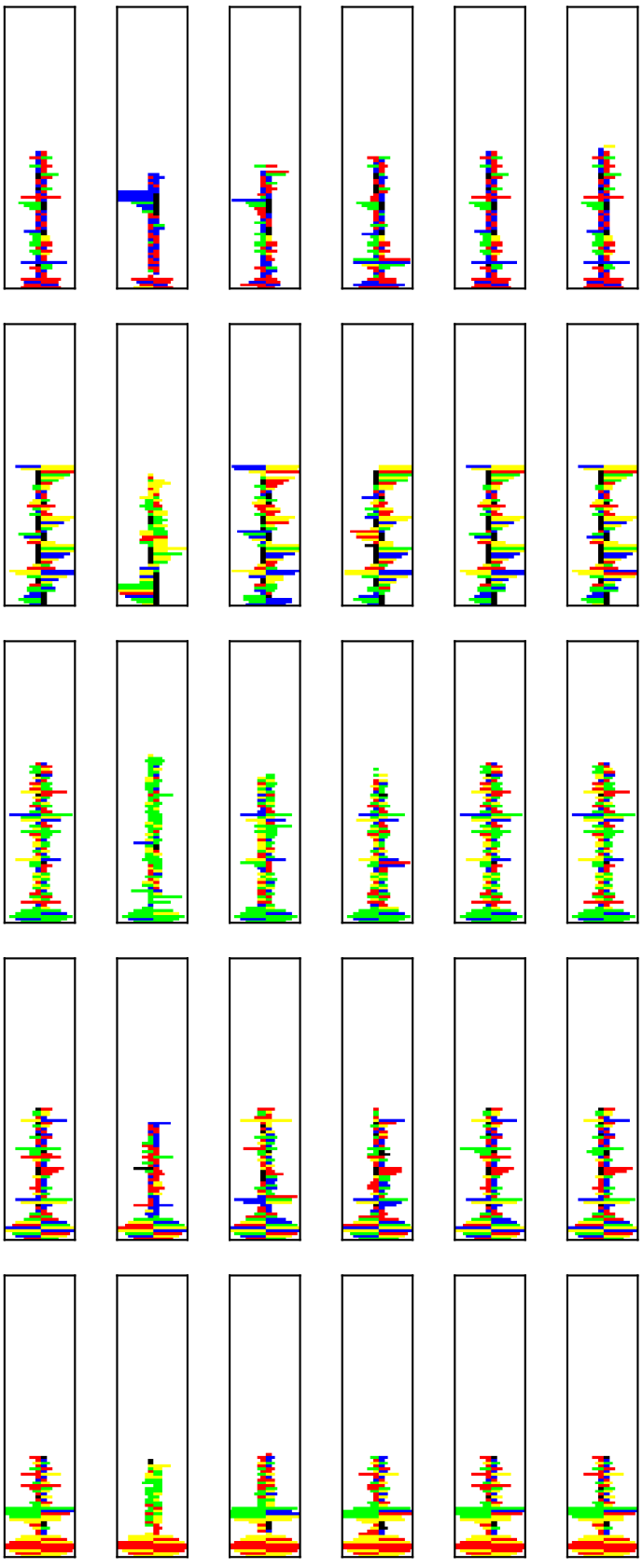


Figure B.1: Reconstructions of 5 random sample from the test set using each model variant. From top to bottom: original, VAE, β-VAE, β-IAF-VAE, DC-β-IAF-VAE, DC-IAF-DIVA

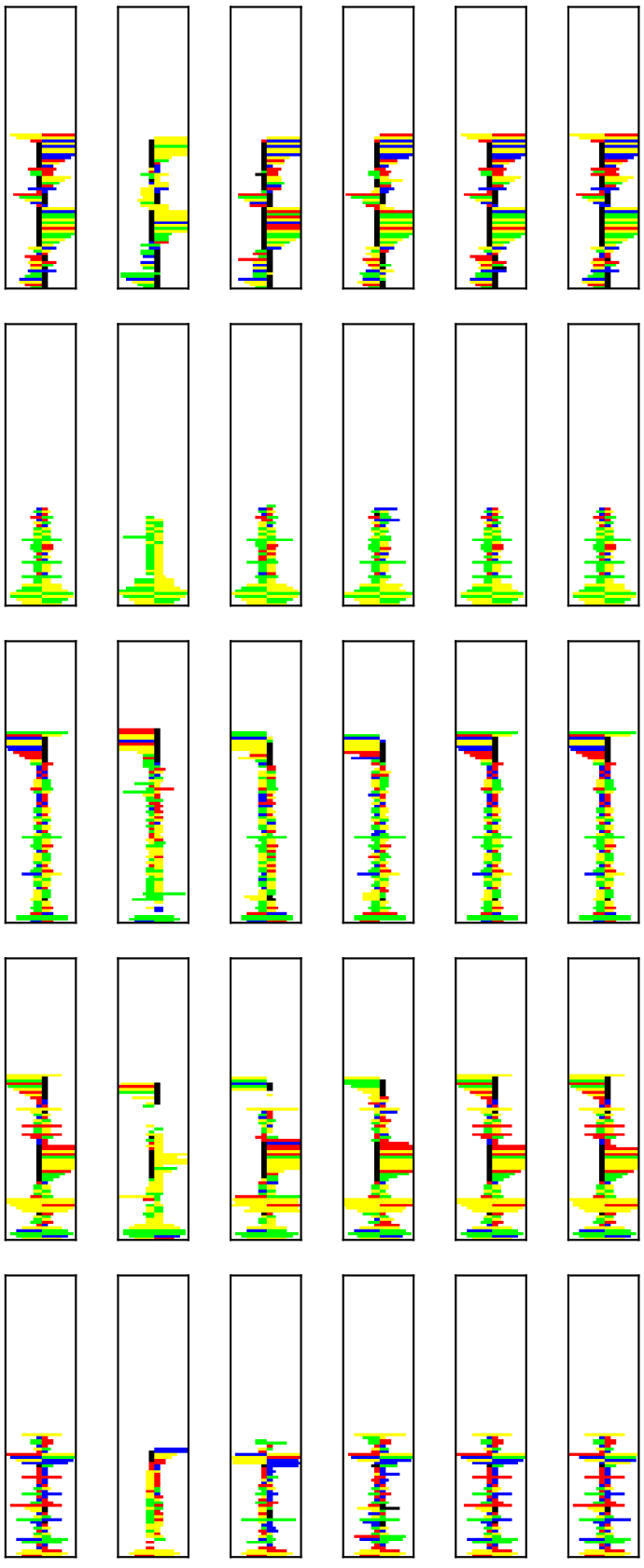


Figure B.2: Reconstructions of 5 random sample from the test set using each model variant. From top to bottom: original, VAE, β-VAE, β-IAF-VAE, DC-β-IAF-VAE, DC-IAF-DIVA

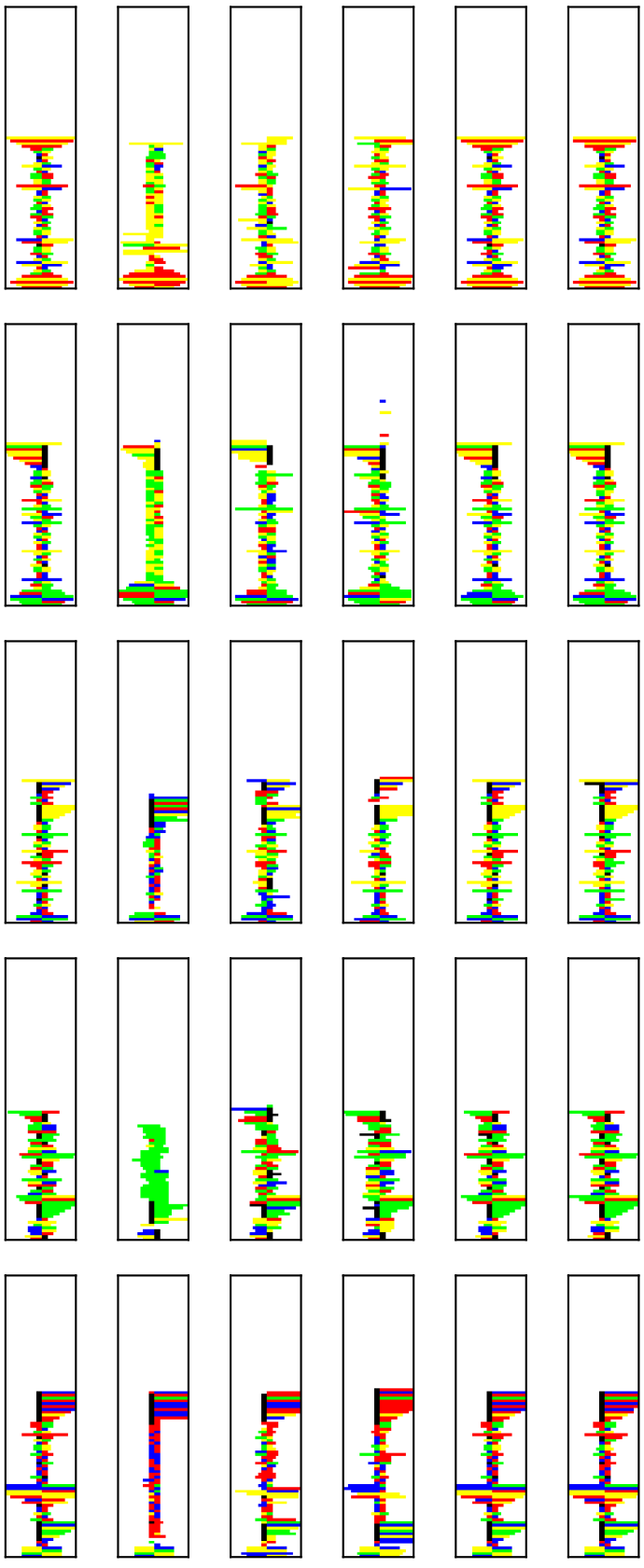


Figure B.3: Reconstructions of 5 random sample from the test set using each model variant. From top to bottom: original, VAE, β-VAE, β-IAF-VAE, DC-β-IAF-VAE, DC-IAF-DIVA

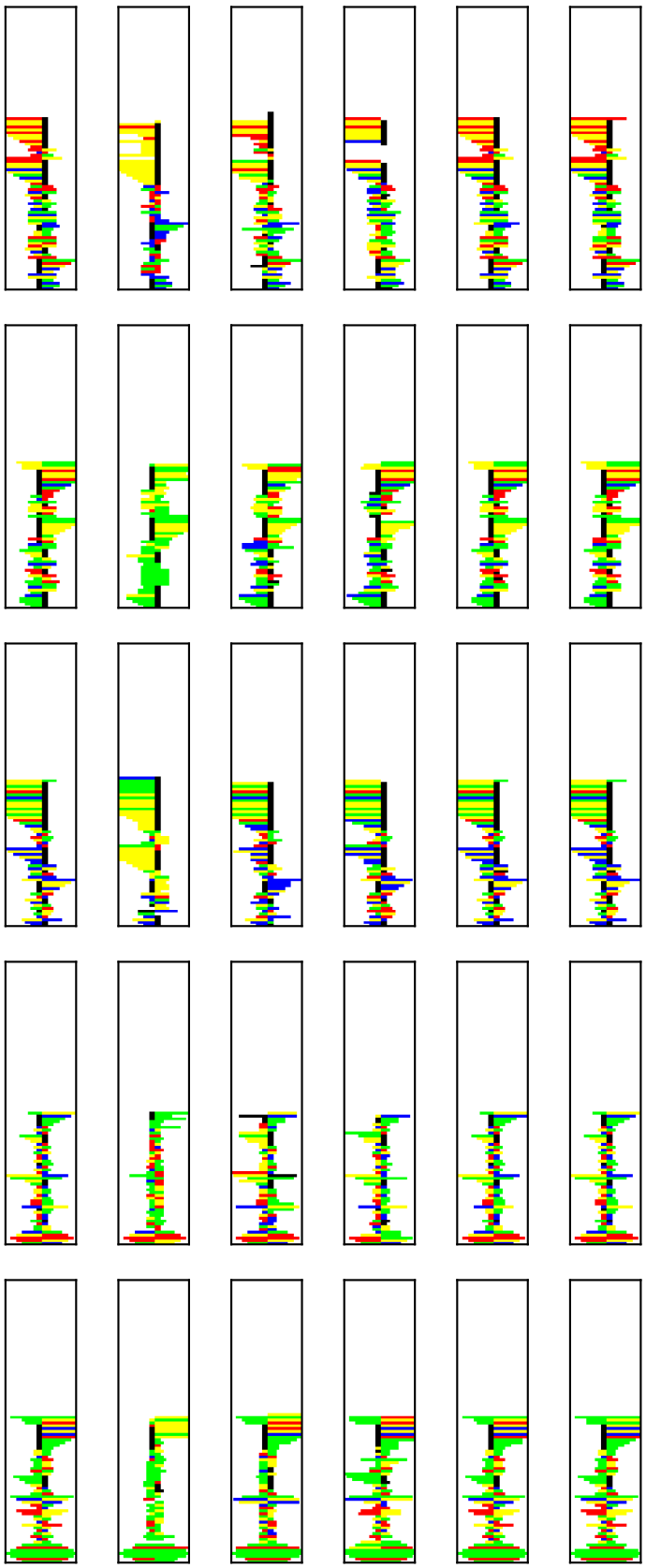


Figure B.4: Reconstructions of 5 random sample from the test set using each model variant. From top to bottom: original, VAE, β-VAE, β-IAF-VAE, DC-β-IAF-VAE, DC-IAF-DIVA

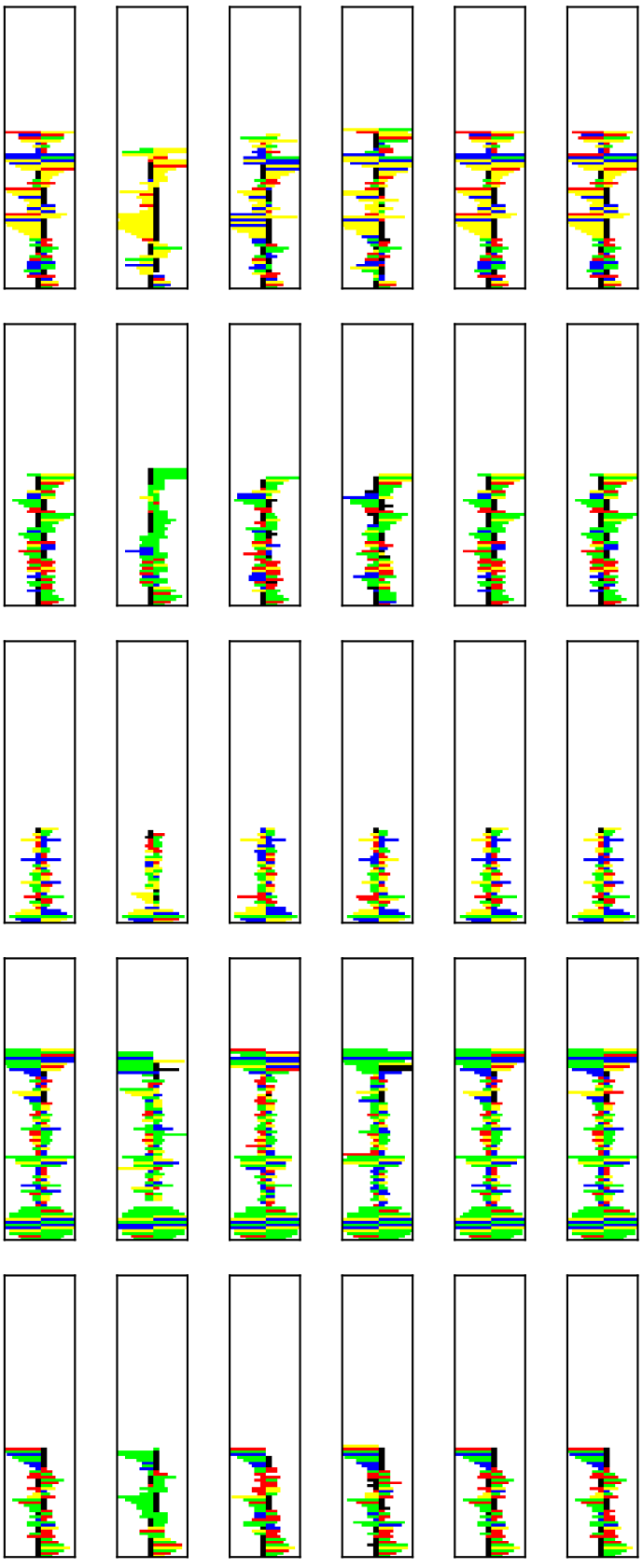


Figure B.5: Reconstructions of 5 random sample from the test set using each model variant. From top to bottom: original, VAE, β-VAE, β-IAF-VAE, DC-β-IAF-VAE, DC-IAF-DIVA