FRIEDRICH-SCHILLER-
UNIVERSITÄT
JENA

Master Thesis:

RNA Nanopore Sequencing: tracking down the m^{6}A modification

to obtain the academic degree
Master of Science (M. Sc.) in Bioinformatics

FRIEDRICH-SCHILLER-UNIVERSITÄT JENA
Fakultät für Mathematik und Informatik
RNA Bioinformatics/High-Throughput Analysis

Written by:
Jannes Spangenberg
born on 25.07.1997 in Neubrandenburg

Supervised by:
Sebastian Krautwurst, Christian Höner zu Siederdissen and Manja Marz

September 3, 2021

Zusammenfassung

Studien zeigen, dass es möglich ist Basenmodifikationen in dem Oxford Nanopore Technologies Sequenzierungssignals von DNA und RNA zu detektieren. Das Signal um eine modifizierte Base weist eine Verschiebung gegenüber der unmodifizierten Variante auf. Genaue und verlässliche Tools zur Vorhersage einer RNA Modifikation sind noch sehr rar und funktionieren noch nicht gut oder nur eingeschränkt gut. In dieser Masterarbeit versuche ich aus dem Nanopore Signal von RNA *in vitro* Transkriptionsdaten die N^6-Methyladenosin (m^6A) Modifikation von einer normalen Adenosin Base zu unterscheiden. Dazu verwende ich die Skriptsprache `Python`, eigens geschriebene Skripte und adaptierte Skripte meiner Betreuer. Für die Vorhersage der m^6A Modifikation aus dem Signal verwende ich tiefe neuronale Netze, welche eine Methode des maschinellen Lernens darstellen. Das Signal verarbeite ich in drei verschiedenen Formen für die neuronalen Netze. Eine Form ist das rohe Sensorsignal, ein andere das transformierte Pikoampere (pA) Signal und die letzte das normalisierte pA Signal. Desweiteren analysiere ich die Basensegmentierung eines Nanopore Signals an und vergleiche dabei die Segmentierungs- und Resquigglingtools `Taiyaki` und `Nanopolish eventalign`. Hierfür visualisiere ich die unmodifizierten und modifizierten Signale bestimmter Motive aus den Datensätzen für das Trainieren und Evaluieren meiner neuronalen Netze. Zum Schluss vergleiche ich die neuronalen Netze untereinander auf ihre Fähigkeit die m^6A Modifikation aus den *in vitro* Transkriptionsdaten vorherzusagen.

Abstract

Studies show that it is possible to detect base modifications in the Oxford Nanopore Technologies sequencing signal from DNA and RNA. The signal around a modified base presents a shift compared to the unmodified variant. Accurate and reliable tools for predicting an RNA modification are still rare and do not perform very well or only to a limited extent. In this master thesis, I aim to distinguish the N^6-methyladenosine (m^6A) modification from a regular adenosine base from the Nanopore signal of RNA *in vitro* transcription data. For this, I use the scripting language `Python`, specially written scripts, and adapted scripts from my supervisors. For the prediction of the m^6A modification from the signal, I use deep neural networks, which are a method from machine learning. I process the signal in three different forms for the neural networks. One form is the raw sensor signal, another is the transformed pico ampere signal, and the last is the normalized picoampere signal. Furthermore, I look at the base segmentation of a Nanopore signal and compare the segmentation and resquiggling tools `Taiyaki` and `Nanopolish eventalign`. For this, I visualize the unmodified and modified signals of certain motives from the data sets for the training and evaluation of my neural networks. Finally, I compare the neural networks with each other for their ability to predict the m^6A modification from the *in vitro* transcription data.

Contents

List of Figures

List of Tables

List of Algorithms

Abbreviations

A	adenine
AUC	area under the ROC curve
CLIP	cross-linking-immunoprecipitation
C	cytosine
DNA	desoxyribonucleic acid
DNN	deep neural network
FPR	false positive rate
GELU	gaussian error linear unit
hm^{5}C	5-hydroxymethylcytidine
HDF5	Hierarchical Data Format version 5
I	inosine
IVT	*in vitro* transcription
LSTM	long short-term memory
LReLU	leaky rectified linear unit
m^{1}A	N^1-methyladenosine
m^{5}C	5-methylcytidine
m^{6}A	N^6-methyladenosine
m^{6}Am	N^6,2'O-dimethyladenosine
NGS	next-generation sequencing technology
ONT	Oxford Nanopore Technologies
pA	pico ampere
Ψ	pseudouridine
ReLU	rectified linear unit
RNA	ribonucleic acid
ROC	receiver operating characteristic
TPR	true positive rate
U	uracil

1 Introduction

1.1 Oxford Nanopore Technologies and modifications

The next-generation sequencing technology (NGS) from Oxford Nanopore Technologies (ONT) makes it possible to sequence desoxyribonucleic acid (DNA) or ribonucleic acid (RNA) directly. In Oxford Nanopore Sequencing, the DNA or RNA strand is pulled through a pore in a membrane. A voltage is applied to this membrane, and a sensor in the pore measures the electric current. The measured current in the pore changes characteristically depending on the bases passing the sensor due to the molecular structure of the DNA and RNA bases, which can be seen in Figure 1. This opens up new opportunities to detect and predict base modifications, as modified bases have a different molecular structure. Therefore, they show a shift in the measured current compared to their unmodified variant, which could look like the signal in Figure 2. The measured current is an electrical signal, also called the nanopore signal. A measured current represents a characteristic signal for five nucleotides, as five bases pass the pores sensor, called a 5-mer or event.

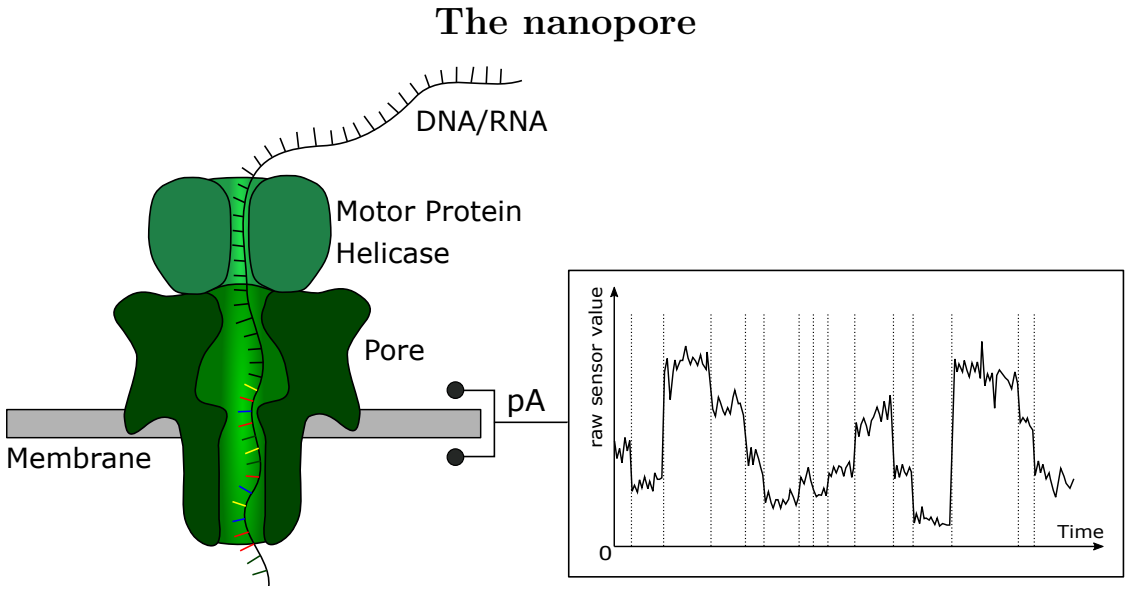

Figure 1: The DNA/RNA is pulled through the pore with the help of the motor protein. A current is applied to the membrane. The bases in the pore disrupt the electric flow, and the sensor measures this disruption. This will result in the characteristic nanopore signal.

The change of the signal between an unmodified 5-mer and a modified 5-mer can

be used to detect modified DNA or RNA [8, 17, 21, 29, 31, 32]. Detecting base modifications is essential, as it can help understand diseases and develop new drugs against lethal diseases, such as cancer. RNA base modifications regulate many post-transcriptional gene expression steps such as degradation, translation, splicing, localization, and primary microRNA processing [2, 3, 7, 25, 27, 38, 42, 43]. They influence the cellular processes, cancer risk, and the cell fate [30,37,44] and play a significant role in the early development of humans [5,9,27,39,41,44,45]. A few of these modifications are N^6-methyladenosine (m^6A), N^1-methyladenosine (m^1A), N^6,2'O-dimethyladenosine (m^6Am), inosine (I), 5-methylcytidine (m^5C), 5-hydroxymethylcytidine (hm^5C) and pseudouridine (Ψ), but much more are known today [15, 27].

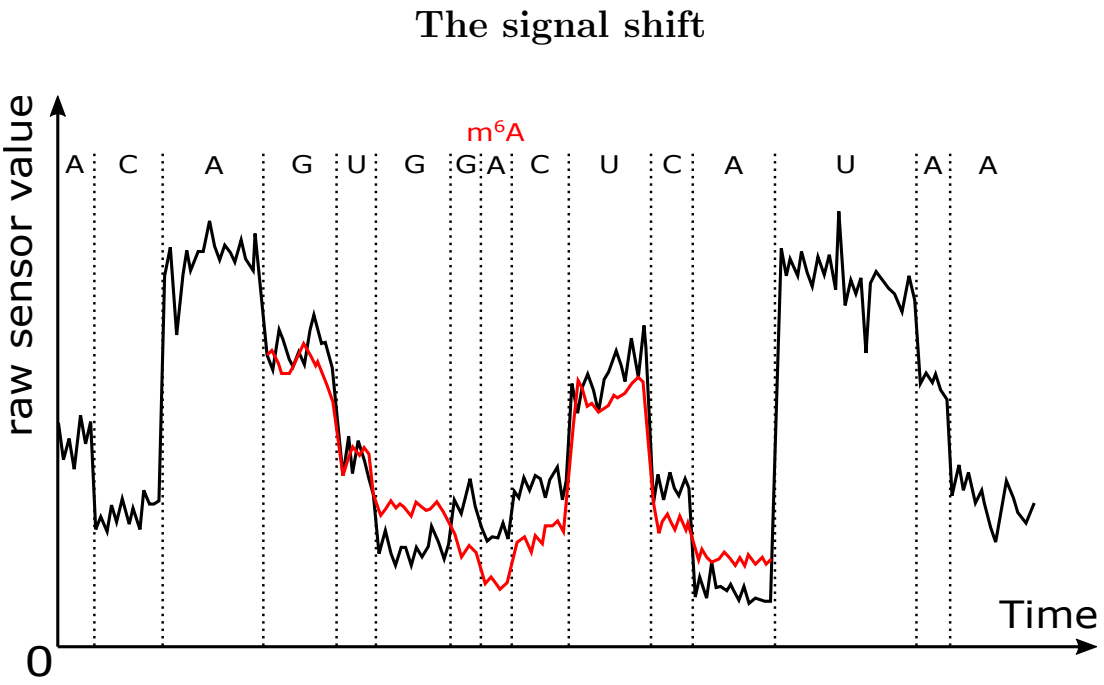

Figure 2: The black line represents a theoretical signal of unmodified bases, while the red part represents a theoretical possible signal shift, created by the m^6A modification.

1.2 Existing methods

There are already some approaches and studies to detect DNA modifications using the signal from ONT sequencing [8, 17, 21, 29, 31, 32]. However, also other methods to map specific RNA modifications exist already. For m^6A and m^6Am a method called cross-linking-immunoprecipitation (CLIP) can be used to map these modifications transcriptome wide [19]. CLIP uses antibodies that bind to m^6A sites [19].

Using ultraviolet light-induced antibody-RNA cross-linking and reverse transcription, specific mutation signatures are induced [19]. Another method is DART-seq, which is an antibody-free method to detect the m^6A modification [24]. It induces cytosine (C) to uracil (U) deamination at Cs adjacent to m^6A bases, which can be identified using RNA-seq [24].

However, tools to detect and predict RNA modifications within the nanopore signal are still rare. Existing RNA modification prediction tools use read aggregated information such as error rates [20], or Gaussian mixture models on signal aggregated information such as the normalized mean or standard deviation of the nanopore signal [27].

The m^6A is often found in specific motifs, to which methyltransferase complexes can bind [6,18,46]. One reported motif is the "DRACH" motif, which methyltransferases can recognize, and demethylases [22].

In the project work [33], I tried to predict the m^6A modification from a nanopore signal, using a deep neural network (DNN) with signal aggregated features like mean, standard deviation, median, median-absolute-deviation, skew, and kurtosis [33]. These features were extracted from different signal types, such as the raw sensor value, the unnormalized pico ampere (pA) value, and the normalized pA value [33].

In this master thesis, I aim to look much deeper into the raw nanopore signal to understand what happens with the signal when an unmodified 5-mer passes the pore compared to a modified 5-mer with the same bases. Furthermore, I use a DNN which takes the nanopore signal representing an odd number of bases, where the middle base represented by the signal is either modified or unmodified, as input. If the sequence is modified, m^6A is the middle base. If the sequence is unmodified, A is the middle base. The network is then trained to predict the modification status of the sequence.

1.3 Deep neural networks

DNNs are a subset of the machine learning area. Neurons and the human brain inspire their construction and functionality. Artificial neural networks are built from layers of artificial neurons. These artificial neurons model the brain's neurons and take one or more values, typically from other neurons, as input. The input values are combined and processed by the neuron using weights, which can be seen in Figure 3. These weights are adjusted while training the network. The neuron's output is given to other neurons as input or is interpreted as the output of the whole network.

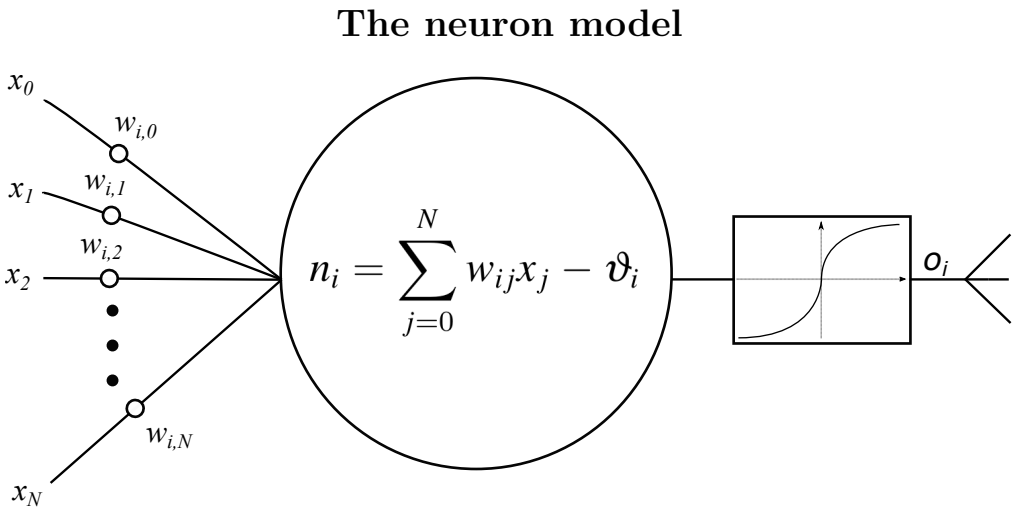

Figure 3: This picture shows a simple neuron model. Every neuron i gets the input values $x_0, x_1, x_2, \ldots, x_N$ over all inputs $N+1$ and sums them up using the weights $w_{i,0}, w_{i,1}, w_{i,2}, \ldots, w_{i,N}$ together with a bias term ϑ_i. The sum n_i is given to an activation function, here shown as a sigmoid function. The output $o_i = \text{activation}(n_i)$ is given to other neurons to process or handled as an output of the whole model.

Neurons are stacked together to form layers. Layers are divided into three different types. The first one is the input layer, shown in red in Figure 4. Neurons in this layer receive their input from the model's input samples, also called features. The output of the input layer can be sent to a hidden layer, which is shown in blue in Figure 4. One or more hidden layers can appear in a neural network. In this layer type, neurons receive information from other neurons by the previous layer. The information or input is processed as shown in Fig 3 and given to neurons of the next layer. The last layer type is the output layer, shown in green in Figure 4. This

layer's neurons act like neurons of the hidden layers, but their output is not sent to other neurons. Their output is interpreted as the output of the whole neural network, which can be of different forms like a binary classification, a multiple-choice classification, a numeric prediction, and others.

In training, the network learns to yield the correct output. Training is an optimization task on the neuronal weights of the network, typically performed using gradient descent with automatic differentiation [4]. In the case of neural networks, this automatic differentiation is called backpropagation [12]. This algorithm performs a gradient descent of the whole network, in which the weights of all neurons are updated with a learning rate to get closer to the correct output for one or more inputs. This is accomplished with a gradient or loss of the output layer neurons, which is determined using a distance or error metric in the form of a loss function. The gradients of previous neurons that contribute to the current neuron's output and, therefore, the resulting gradient are calculated recursively. This is repeated for every neuron's gradient from the output to the network's input layer, hence backpropagated through the network. The gradient of the neurons originates from the derivative of the respective activation function. The learning rate is a scale for the weight adjustment and is used to prevent over-fitting. The update or backpropagation is executed for every batch the neural network sees.

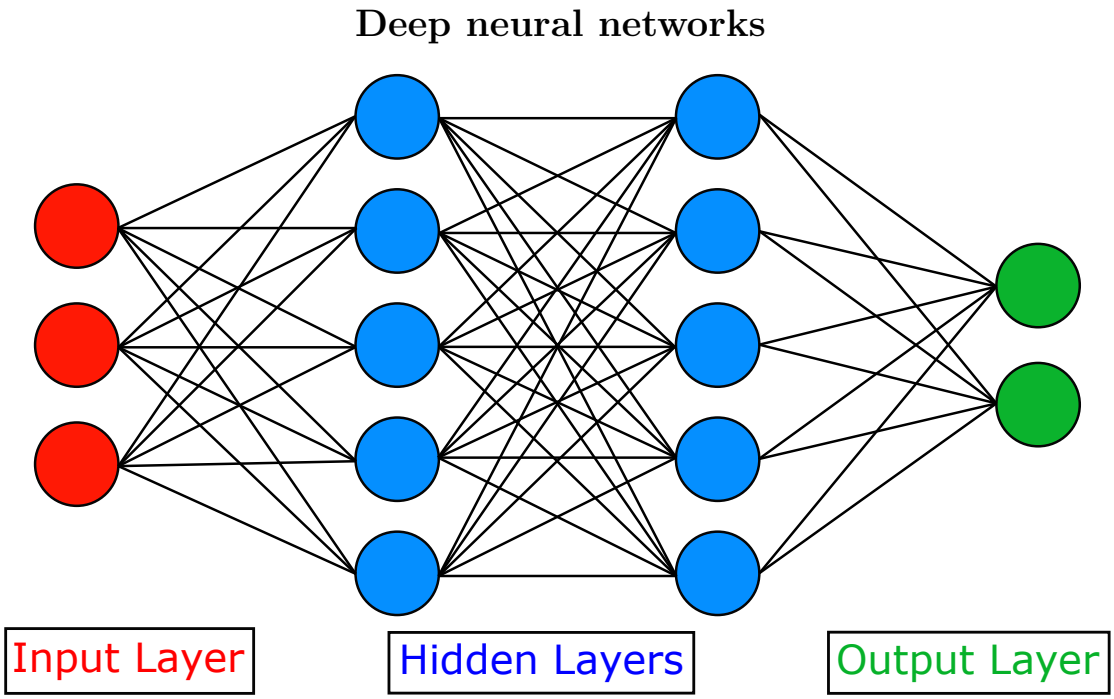

Figure 4: A neural network is built from neuron layers. There are three different types of layers. The first one, shown in red, is called the input layer. After the input layer, zero or more hidden layers, here shown in blue, can be stacked after another. The last layer is called the output layer, shown in green.

1.4 Data collection

One major issue with training a neural network to call methylation is the data collection. In the best case, I have *in vivo* datasets with a perfect ground-truth to train the neural networks and validate their predictions. Such an *in vivo* dataset with a good ground-truth validation is lacking. That is why I use an *in vitro* transcription (IVT) dataset by Novoa et al. (EpiNano) [20] for training and an IVT dataset of Marz et al. (Modbuster) [33] to validate the models. This way, I can check if the model can transfer the learned aspects from one *in vitro* dataset to another. Within these datasets, reads are either fully methylated with m^6A or not methylated at all. *In vivo* a read is not fully methylated, not even fully modified. However, this way, much data can be provided to train models to predict a modification, and it can be tested whether a modification can be predicted from the signal.

2 Materials

Table 1: Conda was used to build enviroments with the necessary packages for the Python scripts. Taiyaki and Nanopolish are segmentation and resquiggling tools. Numpy and PyTorch are used within the Python scripts to analyze and process the data.

Tool	Version
PyTorch [26]	1.9.0
Python [35]	3.8.10
Conda [1]	4.10.3
Numpy [11]	1.21.0
Taiyaki	5.0.0
Nanopolish	0.13.2

I constructed multiple DNN models for the m^6A prediction to handle different datasets and compare them by their accuracy. As mentioned in the introduction, the IVT EpiNano dataset [20] is used to train these models and the IVT Modbuster dataset for validation. The EpiNano dataset [20] was sequenced using the ONT platform GridION, and the Modbuster dataset was sequenced using the MinION Mk1B. Both datasets contain synthesized RNA reads that are either fully m^6A methylated or not methylated at all.

2.1 Resquiggler and segmentation

The first step to prepare the datasets for a neural network is to resquiggle the nanopore signal of every read with the help of a reference sequence to correct the basecalls. Resquiggling means that the nanopore signal and its basecalls are mapped to a reference sequence to correct basecalling errors and adjust the base signal segmentation. Reads, whose nanopore signal could not or only partly be mapped to the reference sequence, are not or only in parts included in the tool's output. Furthermore, the signal base segmentation is adjusted. If the signal base segmentation is incorrect, signal values from one base can protrude into the surrounding base signals.

This could lead to a more challenging prediction problem if the signal boundaries are not clear.

Table 2: The datasets were preprocessed by S. Krautwurst with the mentioned resquiggling tools `Taiyaki` and `Nanopolish`. The output of these tools is processed so that every base in a sample has the same signal length. After that, the datasets get balanced, resulting in the same amount of modified (m^6A) and unmodified (can) samples within the datasets. Due to the exclution parts or whole reads in the output of `Taiyaki` and `Nanopolish`, the number of samples for the same input data differs.

Datasets	**Resquiggler**	**m^6A**	**can**
Novoa et. al. (EpiNano) [20]	`Taiyaki`	3'302'753	3'302'753
Marz et. al. (Modbuster)	`Taiyaki`	881'010	881'010
Novoa et. al. (EpiNano) [20]	`Nanopolish eventalign`	2'444'399	2'444'399
Marz et. al. (Modbuster)	`Nanopolish eventalign`	697'038	697'038

ONT `Taiyaki`[1] and `Nanopolish eventalign`[2] are both available on GitHub.

2.1.1 Taiyaki

`Taiyaki` is a tool to prepare training datasets for basecalling ONT reads. The output of `Taiyaki` contains reads that could be mapped to the given reference together with a segmentation of the base signals, normalization, and transformation parameters. To calculate the pico ampere (pA) current from the raw signal values, called Dacs in `Taiyaki`, the parameters offset, range, and digitization are used. The `Taiyaki` GitHub page provides the equation

$$\text{current} = (\text{Dacs} + \text{offset}) \cdot \frac{\text{range}}{\text{digitization}}$$

for this transformation. It is possible to normalize this current per read with the provided shift and scale values per read by `Taiyaki`.

$$\text{norm\_current} = \frac{\text{current} - \text{read\_shift}}{\text{read\_scale}}$$

[1] https://www.github.com/nanoporetech/taiyaki
[2] https://www.github.com/jts/nanopolish

These three different types of nanopore signals can be used as the input for the DNN model. `Taiyakis` segmentation contains signals with lengths being a multiple of ten, with the smallest being ten. This resolution is a hardcoded parameter of `Taiyaki`, which is unfavorable because the original base signals have different lengths that are not just multiples of ten.

2.1.2 Nanopolish eventalign

`Nanopolish` is a software package to analyze the signal of ONT sequencing data. The tool uses a signal distribution model for each possible 5-mer at the pores sensor. These 1024 signal models contain a mean pA and a standard deviation. With the help of these distributions, the tool `Nanopolish eventalign` provides another segmentation than `Taiyaki`. Some read parts or 5-mer events of `Nanopolish eventalign` cannot be assigned to a model, which can be seen in the output file of the tool. Furthermore, the tool provides multiple non-overlapping segments for a single 5-mer event. These segments get concatenated. A chunk of five base signal segments is used as a sample, if

1. no 5-mer event is missing for the chunk,
2. the middle base of the chunk is the canonical or modified version of the base of interest (A).

If some segments provided by `Nanopolish eventalign` could not be assigned to a 5-mer event, hence its distribution model, then these segments are marked within the output file. These segments get discarded if they appear at the start or end of an event. For those, it is unclear to which 5-mer event they belong. Segments within one 5-mer event that are not assigned are used, as the signal could contain some spikes or outliers that the tool might not assign to the model distribution.

2.2 Model input format

The idea is to construct samples, called chunks, out of the nanopore signal for the model input. To do this, I choose a chunk size of five nucleotides. As already mentioned, five bases are measured at the sensor in the pore at the same time. This means that a modified base influences the signal of four bases up and downstream of itself, which can result in a signal shift as shown in Figure 2. The maximum impact could be achieved by using a chunk size of nine bases. However, this could be unfavorable, as the model might learn the sequence context or motifs instead of using the base signal to predict the methylation status. Another way would be to only take the signal of the modified base, hence a chunk of size one. In this case, much helpful information from the surrounding base signals would be discarded. A chunk size of five also matches the way the datasets were initially constructed. The EpiNano dataset contains synthetic sequences that comprised all possible 5-mers (with a median occurrence of 10) while minimizing the RNA secondary structure [20]. A read in the dataset is either fully m^6A methylated or not methylated at all. The Modbuster dataset was constructed similarly. Here every fifth base is an adenine (A) or m^6A, and every possible 5-mer with an A or m^6A in the middle appears within the dataset of the modified and unmodified reads. Nevertheless, the order of 5-mers is not random in the Modbuster dataset. Not every possible 9-mer with an A in the middle is present, which means that in the fully methylated reads, every base signal representing a C, G, or U is influenced by two m^6A methylations, which might not appear very often *in vivo*. That is why I used the EpiNano dataset for training and the Modbuster dataset for validation.
Each base in a chunk or sample contains multiple features. The most prominent feature is the base signal, Figure 5. This signal must be of the same size in every base for every sample. In section 3.1, I explain how to accomplish this. The size of this base signal will be called signal_size, which describes how many signal data points are used for every base in a sample. Additional features per base are the reference bases, which get one-hot-encoded:

- A or $m^6A \rightarrow [1, 0, 0, 0]$,

- C $\rightarrow [0, 1, 0, 0]$,
- G $\rightarrow [0, 0, 1, 0]$,
- U $\rightarrow [0, 0, 0, 1]$

and the original base signal length extracted from the segmentation provided by the tools `Taiyaki` or `Nanopolish eventalign`. The one-hot-encoding is a simple vector encoding or representation for the bases.
Additionally, the signal can be processed in three different ways. As later shown in section 2.1.1 I can provide the signal as:

- *The raw sensor value* describes the direct sensor value of the nanopore without any transformation or normalization.
- *The unnormalized pA value* is the transformed raw value. With the help of some parameters described in section 2.1.1, the raw sensor value can be transformed into the pA unit.
- *The read normalized pA value* is also described in section 2.1.1. The pA values are normalized using the whole read signal shift and scale values.

Usually, the normalization is useful to eliminate biases that could result from different pores or different sequencing timepoints. But in this case, the raw sensor values work best for the models when using IVT data. The read normalization kind of wipes the signal shift of the modification within the fully methylated reads, as the amount of modifications shifts the whole read signal mean. In those reads, the signal mean and variance are heavily influenced by the modifications, which would *in vivo* not be the case. *In vivo* reads are not fully methylated, not even fully modified. Using the read mean and variance to normalize the read signal might reduce or even remove the signal shift of the modified bases.

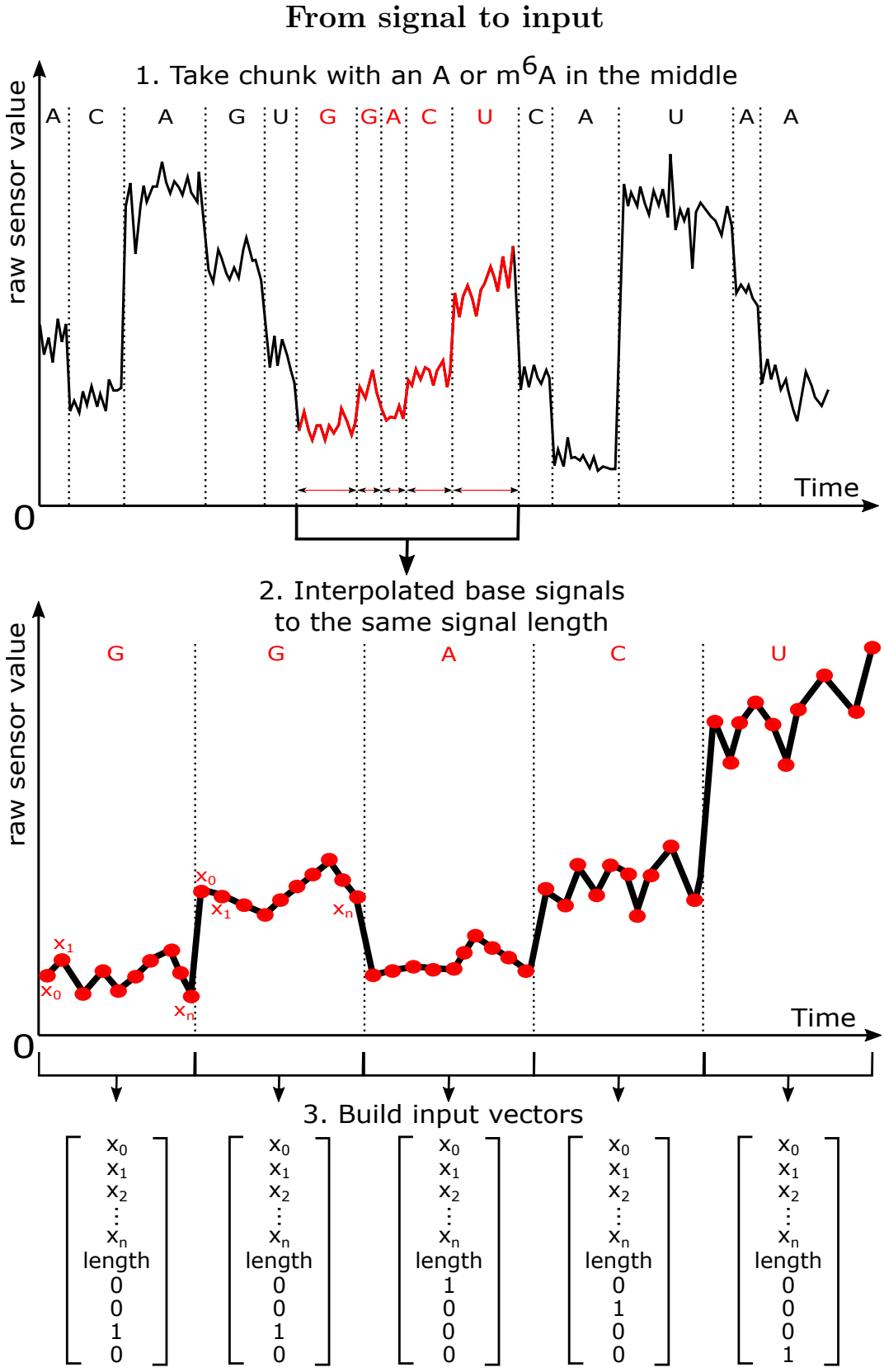

Figure 5: Samples for the network are extracted from the nanopore signal. The base signals are interpolated to the same signal size and are used as an input feature $(x_1, x_2, \ldots, x_n)$, while the original signal *length* is stored as a separated feature. The reference bases are embedded with an *one-hot-encoding*.

2.3 HDF5 format

The Hierarchical Data Format version 5 (HDF5) provides a data structure to store big, complex, and heterogeneous data to train models [16]. It is an open-source file format and uses a directory-like structure to store and access the data. HDF5 also allows metadata storage to add descriptive information about single entries, data subsets, or the whole dataset. `Taiyakis` output is provided in HDF5. The datasets that I prepared for the models are also in HDF5 and fit the design provided by the script `generic_dataset.py`, described later in section 3.2 and can be reviewed in the section 6. Each group in the datasets for the models contains 1000 samples. Grouping multiple samples together increases the writing and reading speed compared to storing them in a single group, which reduces the training time of the models.

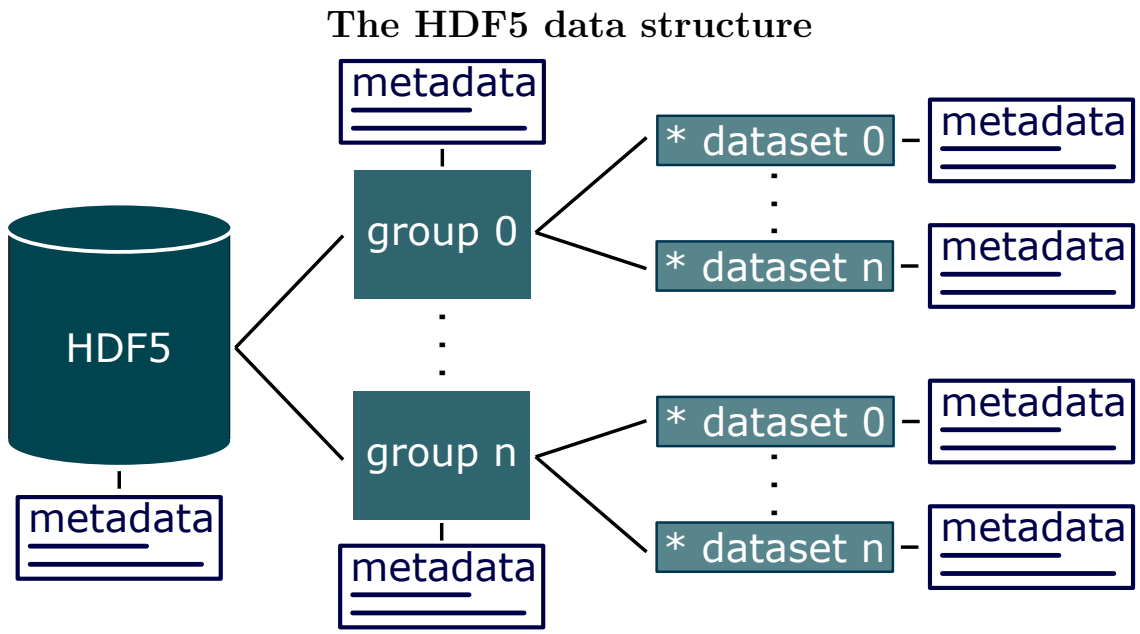

Figure 6: This picture shows a sketch of how an HDF5 file is built up. The format can organize datasets within different groups if necessary. Everything in the HDF5 format can have metadata information. HDF5 supports multiple different data types and data slicing. Data slicing enables efficient work with large datasets and prevents that the entire dataset is loaded into the memory.

3 Methods

3.1 Signal interpolation

The base signals can have different lengths. However, the model always takes a chunk with a given number of bases, in my case, five. As the number of input values is fixed, I implemented a simple linear interpolation (algorithm 1) to resize the base signals to a given length. Different signal lengths for the `Taiyaki` prepared data were tested: 10, 20, and 30. It seems that the longer the signal length, the better the performance of the model. Therefore, I only used the signal length 30 for the datasets prepared with `Nanopolish eventalign`. Because of time constraints, I could not yet try out other signal lengths. I will present models trained on datasets with signal length 30 in my results section 4.

The algorithm 1, which is used to interpolate a given signal, applies one of the following operations. It can shrink a long signal or can inflate a short signal to a given signal length. To accomplish this, the ratio calculated from the size of the given input signal is used, which is the number of data points, and the desired signal length for the model input. This ratio is calculated so that the input signal's first and last value is used as the first and last value of the output signal, which I will call *inter_signal*. The ratio is used as an increment to step through the input signal. If the current step position is not an integer, then the value for the *inter_signal* is calculated by using the index-wise bordering data points from the input signal. These values are weighted with the distance of the step position to the data points. Using these weights, I linearly interpolate the desired data point like a weighted mean. This works because the sum of the weights, hence the distances to the bordering data points, is one. The bordering data points have the indices $\lfloor \text{step position} \rfloor$ and $\lceil \text{step position} \rceil$ in the signal array. Figure 7 and 8 illustrate how the algorithm 1 works. If the current step position is an integer, then the value from my input signal is taken. Another approach to get the same linear interpolation would be to calculate the slope of the bordering data points together with the distance of the step position to get the new data point.

Algorithm 1: Linear interpolation (pythonlike pseudocode)

```
import numpy as np
input: array signal, int signal_length
output: array inter_signal
begin
    inter_signal ← np.zeros(signal_length)
    ratio ← (len(signal)−1)/(size−1)
    ratio_pos, signal_pos ← 0, 0

    while signal_pos < len(inter_signal):

        # case: ratio_pos is an integer: take this value
        if ratio_pos % 1 == 0:
            inter_signal[signal_pos] ← signal[int(ratio_pos)]

        # continuously interpolate at the missing positions
        else:
            # calculate bot and top border value positions
            bot ← int(np.floor(ratio_pos))
            top ← int(np.ceil(ratio_pos))

            # interpolate with bordering values
            # weights are the inverted distances
            inter_signal[signal_pos] ← signal[bot] * (top − ratio_pos)
                                     + signal[top] * (ratio_pos − bot)

        signal_pos ← signal_pos + 1
        ratio_pos ← ratio_pos + ratio

    end
    return inter_signal
end
```

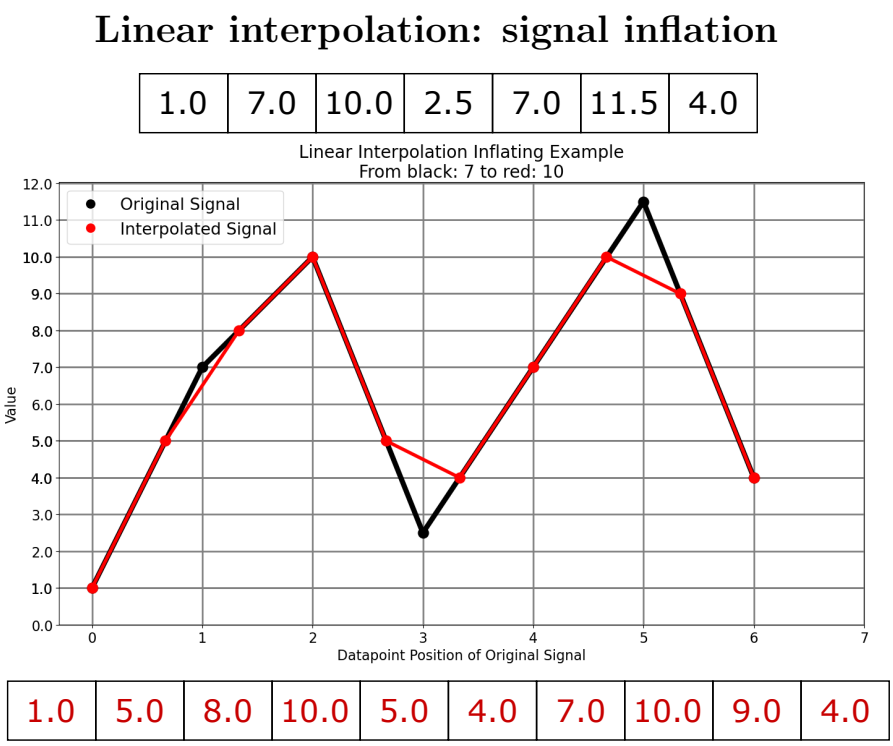

Figure 7: I used Algorithm 1 on the array written in black above the graph of size 7 to inflate it to the array in red below the graph of size 10.

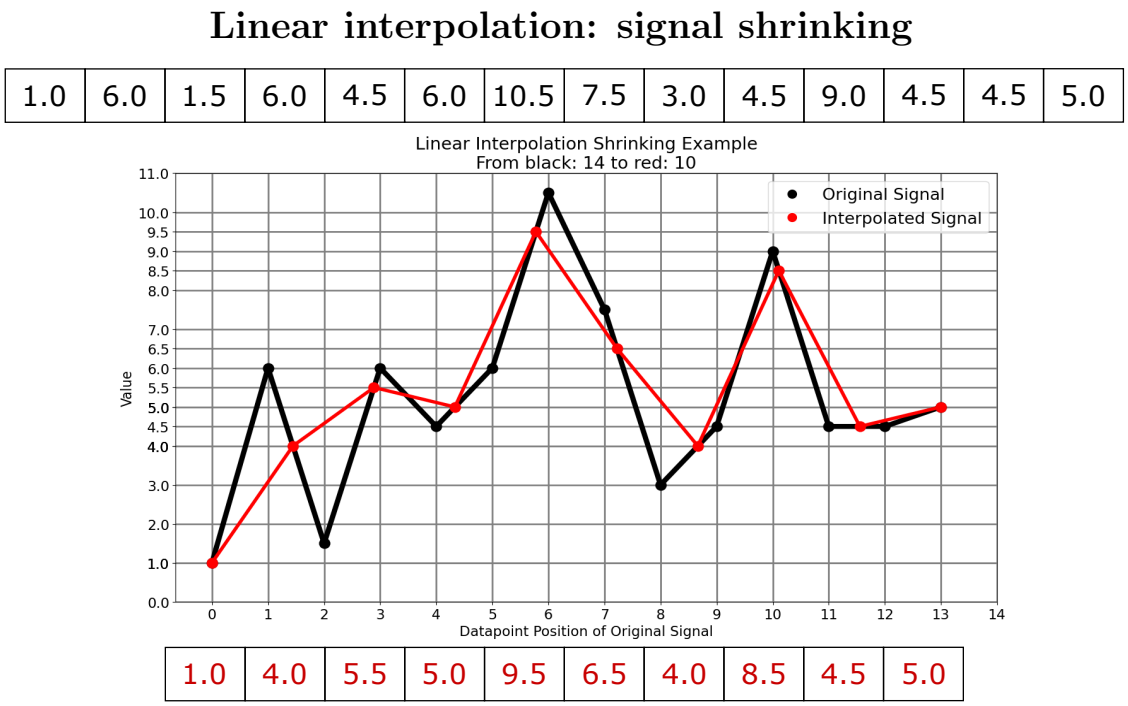

Figure 8: I used Algorithm 1 on the array written in black above the graph of size 14 to shrink it to the array in red below the graph of size 10.

3.2 Python scripts

`Python` scripts are used to learn more about the data and to preprocess the data for the neural networks. These scripts can be found on the USB device in the attachment section 6. The scripts `prepare_dataset.py`, `train_model.py`, `generic_dataset.py` and `test_model.py` were provided by S. Krautwurst and then further edited and changed by myself. Essential scripts are presented and shortly described below:

- `taiyaki_prepare.py`

  is used to prepare the input samples from the read signals in the `Taiyaki` output file. This script builds chunks of a given size with a given base (A) in the middle. It extracts the corresponding base signals of this chunk. `Taiyakis` segmentation resolution is bound to multiple of ten: 10, 20, 30, ... There are no segments of other lengths.

- `nanopolish_prepare.py`

  is used to prepare the input samples from the `Nanopolish eventalign` output file. The `Nanopolish eventalign` tool heavily over-segments the nanopore signal of read parts that the tool could map to a given reference sequence. The script searches for chunks of a given size and base (A) in the middle of the chunk. While doing this, over-segmented events are concatenated together, and a chunk is formed out of them if a given number (five) of consecutive events are found. To keep it simple, the script currently uses the `Taiyaki` output to extract the raw nanopore signal instead of searching multiple Fast5 files for the signal. The `Nanopolish eventalign` output file itself does not provide the nanopore signal, only aggregated features like mean or standard deviation, among the signal segmentation.

- `balance_dataset.py`

  takes a built dataset from `prepare_dataset.py` or `nanopolish_prepare.py` and balances them. After balancing the datasets contain the same amount of samples per provided class (0 = unmodified, 1 = modified). All samples of the minor class are included. The same amount of samples from the major class are randomly drawn. The classes are sorted within the balanced dataset.

- `generic_dataset.py`

  was designed to provide a general data structure for the training and evaluated datasets. The script is used to load and handle the HDF5 files in multiprocessing and create the samples directly on the desired device.

- `train_model.py`

  is used to train the neural networks with different hyperparameters.

  - One is the batch size, which is the number of samples for which the loss is calculated before the model's weights are adjusted via backpropagation. This mainly affects how fast the model is trained, as backpropagation is very time-consuming. It can prevent over-fitting, as the model sees many different random samples at once. For all models that will be presented later, I used a batch size of 16.
  - The learning rate is a scalar for the weight adjustment in backpropagation. The default learning rate is 0.0001.
  - Another parameter is the number of training epochs, which is at least 80. In some models, the number of epoch goes up to 200 to see if the model can still learn and improve by training for more epochs.
  - The number of transformer encoder layers within the transformer part of the models can be changed as well. The default here is two.
  - The last much-used parameter is the number of workers. This parameter determines how many subprocesses are used by the dataloader module of `PyTorch` in the training loop. These subprocesses preload the sample batches in parallel in the background to increase the training speed. They are prepared on the given device (CPU or GPU), where the model should be trained to reduce data traffic. The default device is the GPU (cuda:0).

- `test_model.py`

  is the script to evaluate trained models with a given dataset. It is structured similarly to `train_model.py` without the training part.

- `model.py`

  contains different model classes. These classes define the model's architecture. There is currently a superclass for every model that contains the methods for the sample preparation to build the training and evaluation datasets. This includes signal interpolation.

- `plot_signals.py`

  plots the nanopore signals from a given prepared dataset for all found motifs of a given size. A subset of 200 signals is plotted to represent every motif found in the dataset and to keep the plots clear and not overloaded. This script was used to examine the data and learn more about the nanopore signal and the shift resulting from the m^6A modification. It also helped to show the importance of signal segmentation.

- `plot_curves.py`

  plots the precision-recall and receiver operating characteristic (ROC) curves seen in section 4 from the classification data of up to eight models.

3.3 Model architecture

The models are built using the transformer encoder and linear layers from `PyTorch`. First, the input samples are split. One part is the interpolated base signals, and the other part is the additional features like original signal length and reference bases one-hot encoded. The base signals are sent to the transformer encoder, whose output will be concatenated with the additional features to be processed by the linear layers. Finally, a linear layer reduces the input to one value. After that, a sigmoid function transforms the value from an interval of $[-\infty, \infty]$ to an interval of $[0, 1]$. This output value represents the methylation status prediction of the input sample with the A in the middle. An activation function and a layer normalization separate every linear layer, and the transformer encoder also uses an activation function. The models use the rectified linear unit (ReLU) and leaky rectified linear unit (LReLU) activation functions. The details are described and can be seen in following figures 10 - 12.

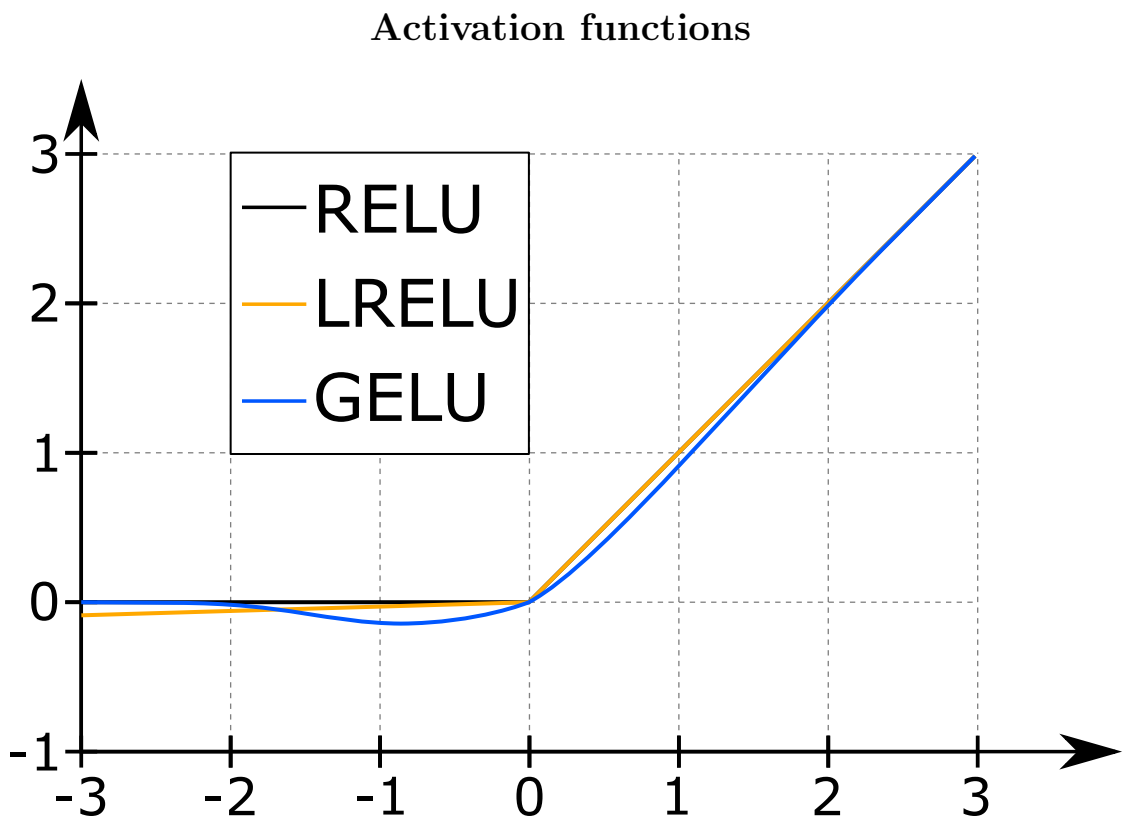

Figure 9: The black graph shows the standard ReLU activation function. This function is constant 0 from $x \in [-\infty, 0)$ and x for $x \in [0, \infty]$. Simply put ReLU $= \max(0, x)$. LReLU, here in orange, is different, the negative part is multiplied with a slope, so it is not constant 0. `PyTorch` provides the formula LReLU $= \max(0, x) +$ negative_slope $* \min(0, x)$. The LReLU in the models has the default value for the negative_slope, which is 0.01. The gaussian error linear unit (GELU) function shown in blue is another possible default activation function integrated in `PyTorch`.

3.3.1 Transformer layer stacks

A transformer model uses the multi-head self-attention mechanism, instead of recurrence or convolution, to solve the sequence transduction problem and learn long-range dependencies within input sequences [10, 36]. Recurrent neural networks are used in time series processing, but due to the recurrence and the vanishing gradient problem, they have an increased learning time [13]. The vanishing gradient problem describes that the gradient to update the weight of a neuron will be vanishingly small in some cases, which also depends on the derivatives of the activation functions. The more weight adjustments a network executes, the higher the chance is that this problem occurs. The transformer model reduces this problem while keeping the ability

to solve the sequence transduction problem with the help of the self-attention mechanism instead of recurrence. Using this mechanism, the transformer architecture is more parallelizable than recurrent neural networks and requires less training time. The multi-head self-attention mechanism of the transformer model is built out of multiple attention units also called attention heads. An attention unit learns weights for each input value with the help of a scaled dot-product. These weights are divided into the categories query (Q), key (K), and value (V) and stored as matrices. Each input sample is multiplied with each matrix. The attention of an input sample is calculated with:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_K}}\right)V \text{ [36]}$$

Using two matrices Q and K as QK^T allows the attention to be non-symmetric. So, if value i attends to value j, it does not necessarily mean that value j attends to value i. d_K is the dimensionality of the key matrix, and $\sqrt{d_K}$ stabilizes the gradients during training. The softmax function normalizes the weights. The attention describes which input values attend other input values. Multi-head attention describes that multiple attention units are calculated in parallel for a given input sample. These attention calculations are combined to produce an overall attention score for an input sample [36].

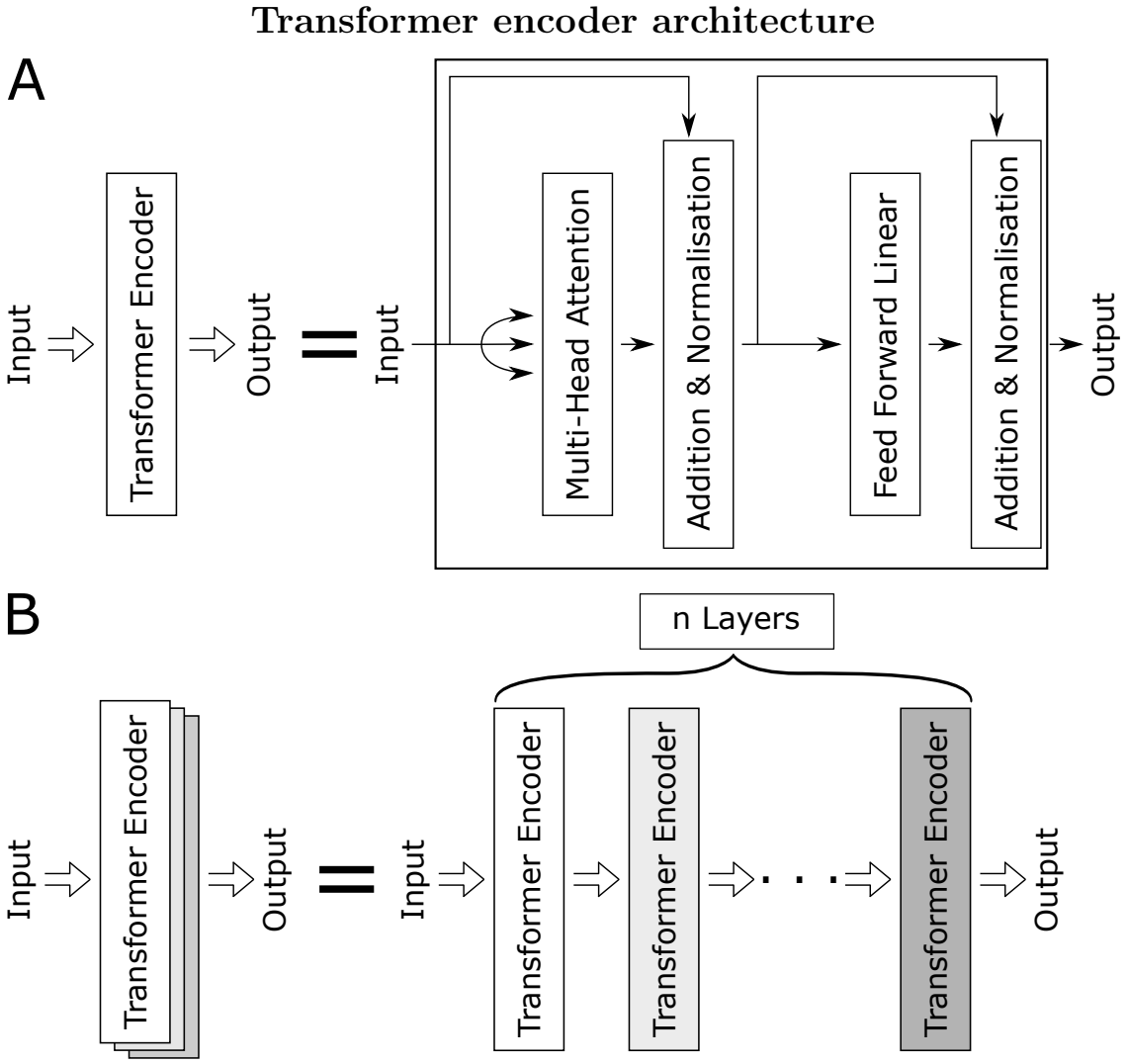

Figure 10: **A** shows the transformer encoder architecture as introduced by Polosukhin et al. [36] in "Attention is all you need".
B describes how multiple a transformer encoder stack is illustraded for the models architecture in Figure 12.

The transformer encoders in my models take the interpolated nanopore signal of a given signal length as input. They use $\lfloor \frac{\text{signal\_size}}{2} \rfloor$ heads for the multi-head self-attention mechanism and process this signal using a given number of encoder layers, the ReLU activation function, and the layer normalization. Additionally, the feed-forward linear layers of the transformer encoder have a dropout of zero and a size of 256 neurons. A dropout can help prevent the over-fitting of the model to data. The output of this part of the model has the same size as the input signal.

3.3.2 Linear layer stacks

The linear layers of the model have an input and output size of (signal_size + add_feats) × chunk_size, which represents the size of all features for every base concatenated together. In this case, the input contains the transformer output plus the additional features of the sample as shown in Figure 12. Add_feats defines the number of features apart from the signal: the original signal length before the interpolation and the reference bases one-hot-encoded. Within the linear stack, the LReLU function and the layer normalization is used as shown in Figure 11. The used LReLU is an adapted ReLU that has a negative slope of 0.01. This was done to prevent the neurons from dying while training the model, as they might when using the ReLU function. A neuron that uses the ReLU activation function might end up stuck in the negative part of the function. Here the function always returns zero. The gradient in the negative area is also zero, which means that the gradient descent learning does not alter the weights anymore. The neuron is now stuck and can not recover. Therefore, the LReLU function uses a negative slope greater than zero so that the gradient is not zero and the neuron's weights can still be adjusted.

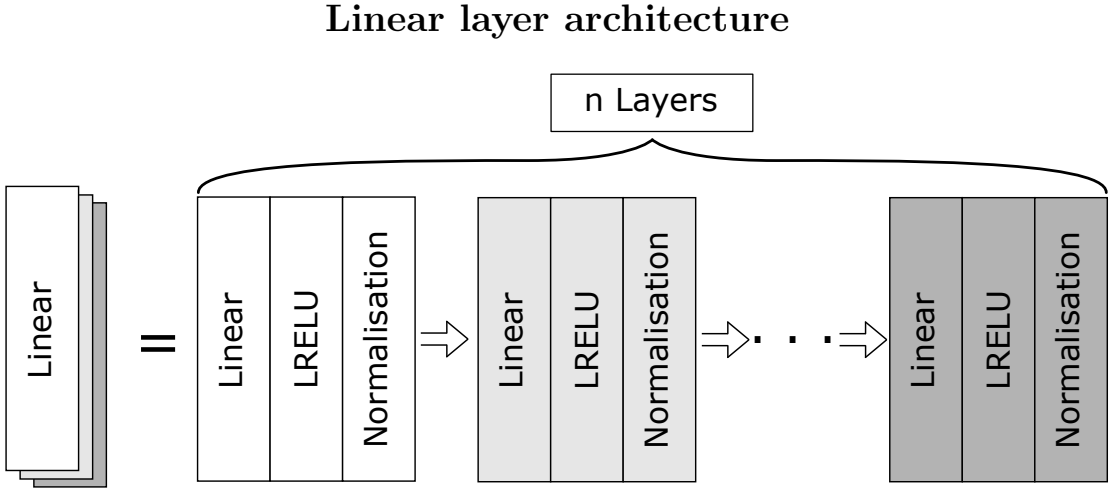

Figure 11: The linear layers are stacked after each other. These linear stacks are built together with the LReLU activation function and the layer normalization. The number of stacked layers differs slightly between the models trained. The exact hyperparameters can be seen in table 3. How the linear stacks are used in the model's architecture is shown in Figure 12.

3.3.3 The network structure

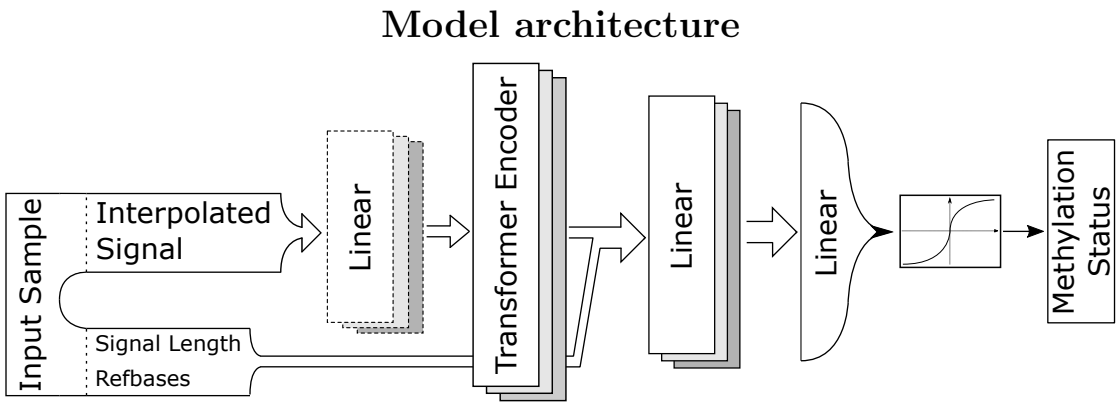

Figure 12: As described before, the input sample is split by the interpolated signal and the additional features. The transformer encoder layers process the interpolated base signals. For the raw sensor signal, a linear layer stack is installed in front of the transformer. These layers should preprocess the signal for the transformer encoder, as they are not normalized and do not have small values around zero. The assumption here is that the transformer model handles values near zero better. The transformer encoder uses the default ReLU activation function. `PyTorch` only allows to use the ReLU or GELU activation function. After the second linear layer stack, a single linear layer reduces the input to a single value. A sigmoid function converts this value to an interval of $[0, 1]$. The output represents the methylation status for the input chunk. 0 means the middle base of the chunk is an unmodified A. 1 means it is m^6A methylated.

Every model follows the architecture shown in Figure 12. Some of the models are shown in the following tables. All models were trained using the EpiNano datasets and validated on the Modbuster datasets, table 2. For the training, I used `PyTorch's` binary cross-entropy with default settings. It is a loss function to calculate the loss between the methylation labels (0 or 1) and the model's outputs, which is the predicted methylation status prediction for a sample. The loss function is a metric and calculates a distance or error score [23]. This score penalizes the prediction based on the difference from the prediction to the label.

Table 3: These tables show the models (**A**) and the corresponding hyperparameters used while training them (**B**).
A is a list of all numbers of different layers used in the architecture in Figure 12.
B contains the models I will present and compare in this thesis. Every model gets an ID to map the corresponding evaluation results in other tables. norm = normalized signal, unorm = unnormalized signal, raw = raw sensor signal

A

Model architecture	Number of linear layers before transformer	Number of transformer encoders	Number of linear layers after transformer
transformer_slim	0	2	2
trans_slim_raw	1	2	2
trans_big_raw	2	5	3

B

ID	Model architecture	EpiNano trainingset	Chunk size	Signal size	Signal type	Number of epochs
0	transformer_slim	Nanopolish	5	30	norm	80
1	transformer_slim	Taiyaki	5	30	norm	80
2	transformer_slim	Nanopolish	5	30	unorm	80
3	transformer_slim	Taiyaki	5	30	unorm	80
4	trans_slim_raw	Nanopolish	5	30	raw	200
5	trans_slim_raw	Taiyaki	5	30	raw	200
6	trans_big_raw	Nanopolish	5	30	raw	200

The models that use the norm and unorm signal type were trained 80 epochs. The models trained on the raw signal type perform better than the norm and unorm counterparts on the Modbuster evaluationset, table 4. Therefore, I focused on the raw signal type and tried to increase the model's performances with other architectures or more extended training periods, as seen in Figure 12 and table 3. In section 4 table 4, I compare the accuracies of the models after 80 epochs. Later, I compare the models with the best accuracy, regardless of the number of epochs used while training.

3.3.4 Training strategy

The models get trained using the EpiNano datasets. These sets in table 2 are prepared so that the amount of samples for both classes in a set is the same. While training, the `Dataloader` from `PyTorch` draws these samples without replacement randomly and prepares them in a batch for training. The model gets trained with a batch of training samples. It predicts the methylation status of every sample in the batch. The loss of this batch is calculated to perform backpropagation. The mean binary cross-entropy function (`BCELoss`) of `PyTorch` calculated the loss for every batch. The ground-truth is the binary label of the samples in the batch representing unmodified or m^6A methylated. Using this loss, the model's weights can be adjusted via backpropagation and a learning rate. The model learns to predict the methylation status from the signal and additional features. Every model gets trained using a learning rate of 0.0001. The trainingset gets split into the actual set for training (80% of the samples) and a testset (20% of the samples) to monitor the training results. The model never sees the samples of this test split, but they originate from the same data as the training samples. With the help of the test split, it can be observed if my model learns to predict the methylation status and if it starts to over-fit on the training split. In the latter case, the performance on the testset would be low or decrease, while the performance on the training split would increase. In the first case, the performance on the testset would increase too. Performance describes the accuracy and the loss of the model for a given input.

3.4 Sample processing

As described earlier, the models see the samples in a batch, meaning multiple samples are used to calculate a loss before adjusting the neuron's weights. These input samples are chunks of size five whose features are processed differently by the models. First, the interpolated signal is split from the additional features within the chunk, as seen in Figure 12. The interpolated base signals are passed to the transformer encoders, which are vectors with the same length of the given signal length. The output of the transformer encoders is then concatenated with the additional features of the input sample. Currently, additional features are the length of the original base signal and the reference base as an one-hot-encoding. The concatenated features are the input of the linear layers of the model. These layers process the output of the transformer part and the additional features together. To calculate the model's prediction loss, every sample is labeled with a 0 (unmodified) or 1 (m^6A methylated).

4 Results

In the following sections, I want to present visualizations of the data that I used to train and evaluate my models. This will provide a deeper look into the segmentation of `Nanopolish` and `Taiyaki`, the base signals, and the shift of the signal caused by the modification. Furthermore, I will evaluate the performance of my models to predict the m^6A methylation on the Modbuster IVT datasets. I used a ROC curve and precision-recall curve to compare the models.

4.1 The segmentation is the foundation

The `plot_signals.py` script is used to analyze the segmentation and samples of the prepared datasets. This script takes a prepared dataset as the input and creates base signal plots for all found motifs. The datasets contain samples of motifs with an A as the central base, and the base signal lengths are interpolated to a size of 30 data points. The following plots show these 30 data points from the motif GGACT (T = U in RNA) over a subset of 200 samples per class. The GGACT motif shows a relatively good separation between the unmodified and modified signal compared to other motifs. Additionally, a signal distribution for each base is shown on the right-hand side of the segments. A histogram can be seen on the x-axis of these plots. This histogram shows a relative distribution of the original signal lengths of the plotted subset. Each bar of the histogram covers a range of lengths. The y-axis left shows the value of the data points, while the y-axis on the right shows the height of the histogram. In the case of the unnormalized signal plots, a k-mer model from nanoporetech provides distributions for comparison. These models can be found unter the name *r9.4_180mv_70bps_5mer_RNA* under github.com/nanoporetech/kmer_models/tree/master/.

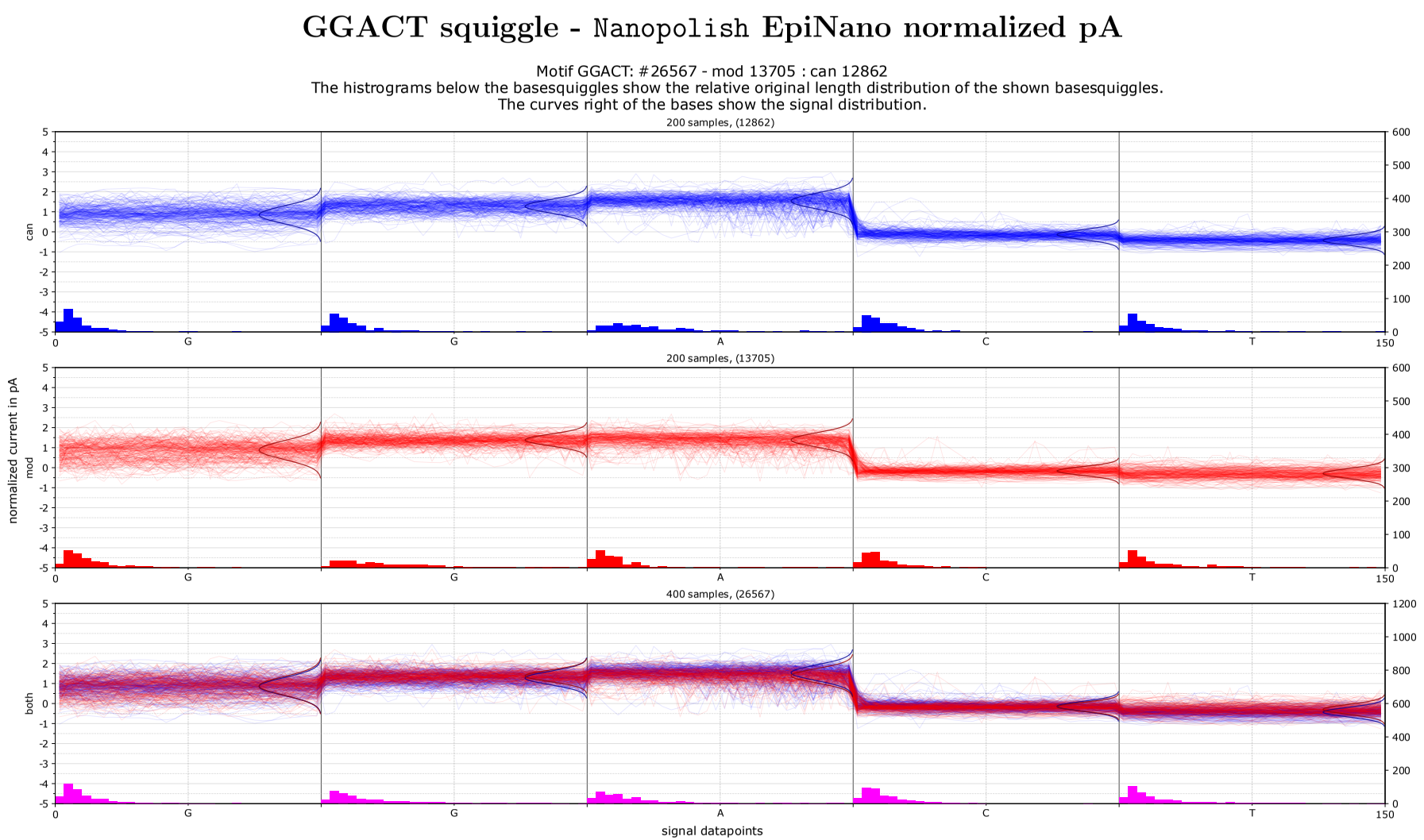

Figure 13: This picture shows 200 interpolated normalized pA signals per class from the EpiNano dataset, using the `Nanopolish` segmentation of the 5' GGACT 3' motif. The unmodified motifs are called *can* and are shown in blue, while the m6A motifs are colored red and called *mod*.

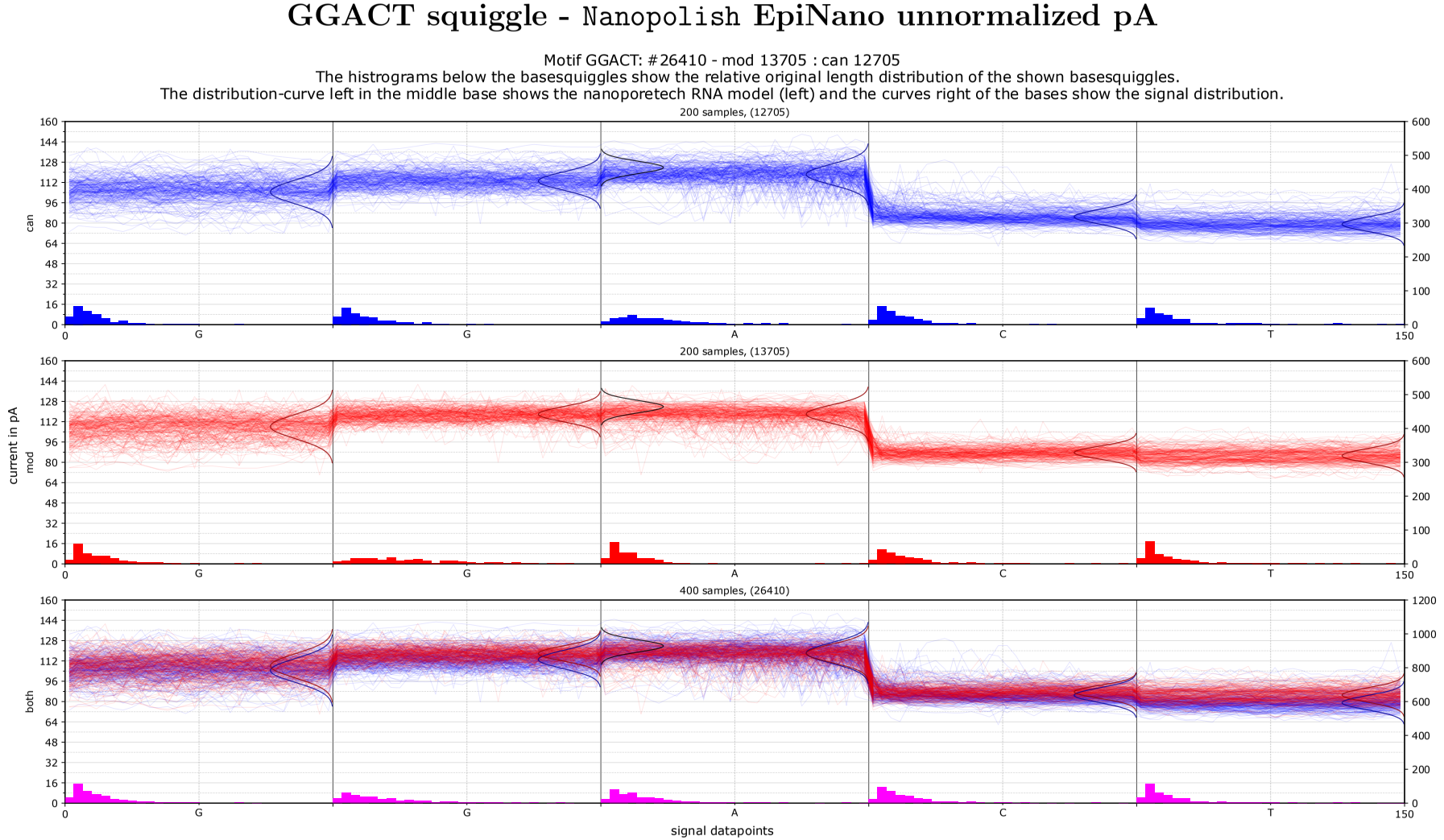

Figure 14: This picture shows 200 interpolated unnormalized pA signals per class from the EpiNano dataset, using the `Nanopolish` segmentation of the 5' GGACT 3' motif. The unmodified motifs are called *can* and are shown in blue, while the m6A motifs are colored red and called *mod*.The curves on the left-hand side of the middle base shows the distribution models from nanoporetech.

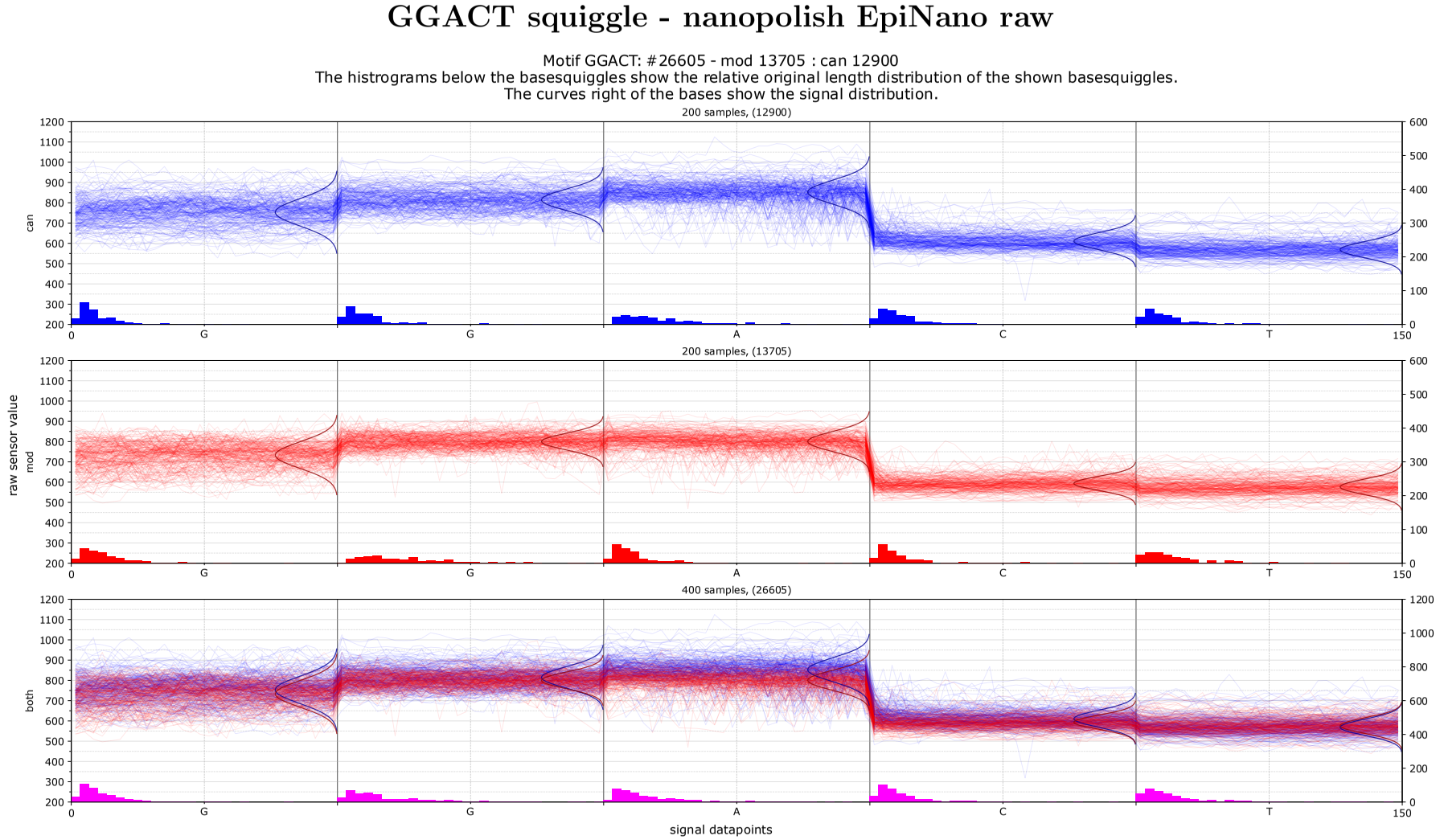

Figure 15: This picture shows 200 interpolated raw signals per class from the EpiNano dataset, using the `Nanopolish` segmentation of the 5' GGACT 3' motif. The unmodified motifs are called *can* and are shown in blue, while the m6A motifs are colored red and called *mod*.

The figures 13 - 15 show the three different signal types for the motif GGACT of the prepared dataset from EpiNano using the `Nanopolish` segmentation. Two hundred signals of both classes are plotted. These signal samples are drawn from the randomized and balanced dataset. The raw and unnormalized signals separate more at and around the modified base compared to the normalized variant. This can also be seen in other motifs of the same dataset and the Modbuster dataset, The normalized signal in Figure 13 shows nearly no separation between the unmodified or canonical signal in blue and the modified m^6A signal in red. The signal separates just slightly at the second G, A, and T. The separation is much stronger when looking at the unnormalized pA signal in Figure 14 and also appears at other bases in the motif. Here the 400 are randomly picked samples separate in both G bases and the T at the 3' end. This is again much different from the raw signal in Figure 15. The middle base A separates from the m^6A variant, and the surrounding base signals overlap much more. All three plots show a difference in the original signal length at the second G and the A base. In the unmodified signal, the A signal is longer than the m^6A signal, as seen in the histogram. The second G shows the opposite picture. Here the G of the unmodified signal is generally shorter than the G of the modified motifs. This indicates that using the length of the base signals can be a strong predictor for the methylation prediction.
In addition, the signal is relatively well segmented by the `Nanopolish` segmentation. Some outliers, fluctuations, or spikes within the signals could make the prediction very hard for these samples. Normalization can help with making the signal much more concise, but the read normalization, as seen in these pictures with fully or unmethylated reads, seems to destroy the signal shift caused by the m^6A modification. In the case of IVT data, this normalization does not seem to be useful, which might also lead to lower performances of the models trained with the normalized signal data, which can be seen in section 4.2.

GGACT squiggle - Taiyaki EpiNano raw

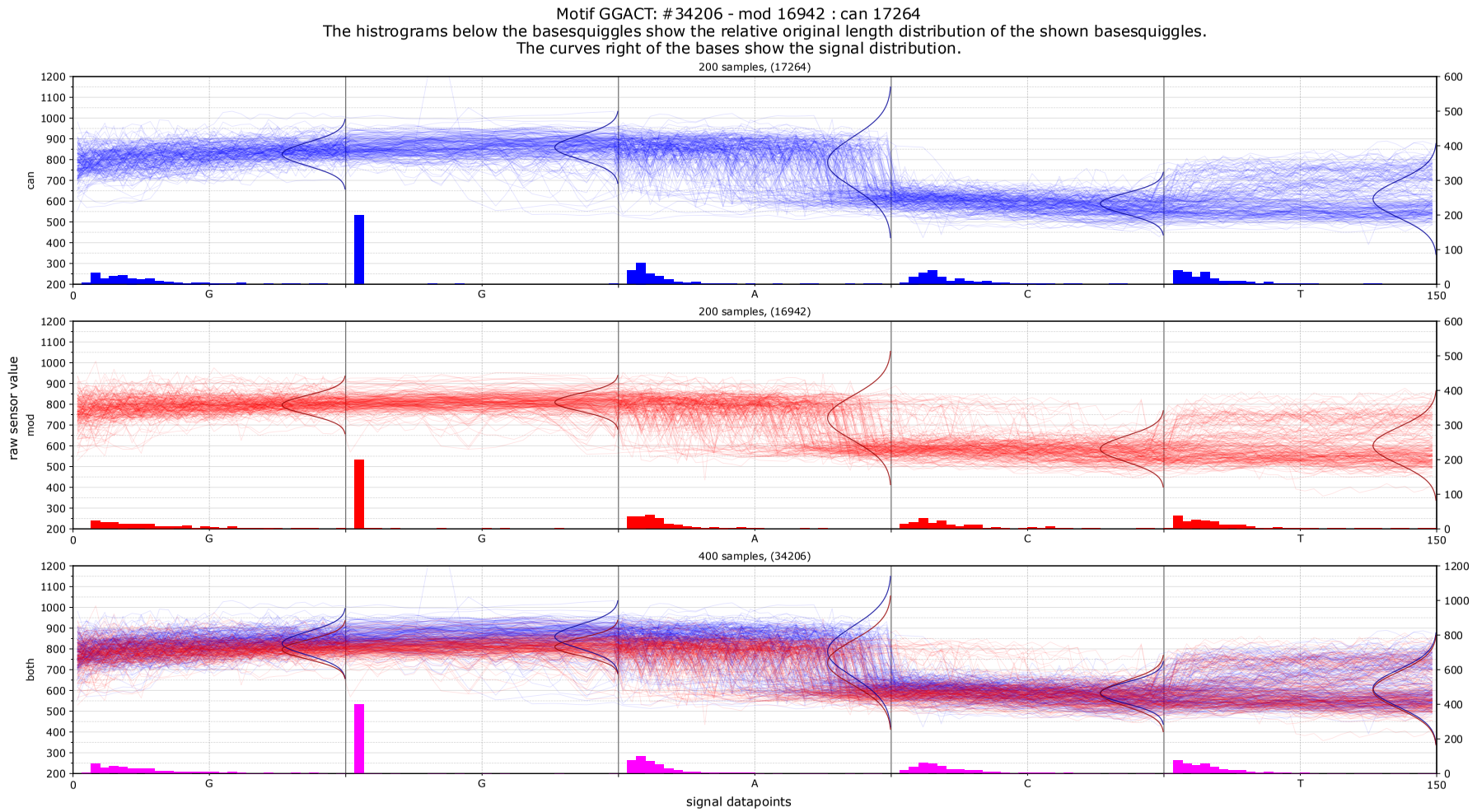

Figure 16: This picture shows 200 interpolated raw signals per class from the EpiNano dataset, using the Taiyaki segmentation of the 5' GGACT 3' motif. The unmodified motifs are called *can* and are shown in blue, while the m6A motifs are colored red and called *mod*.

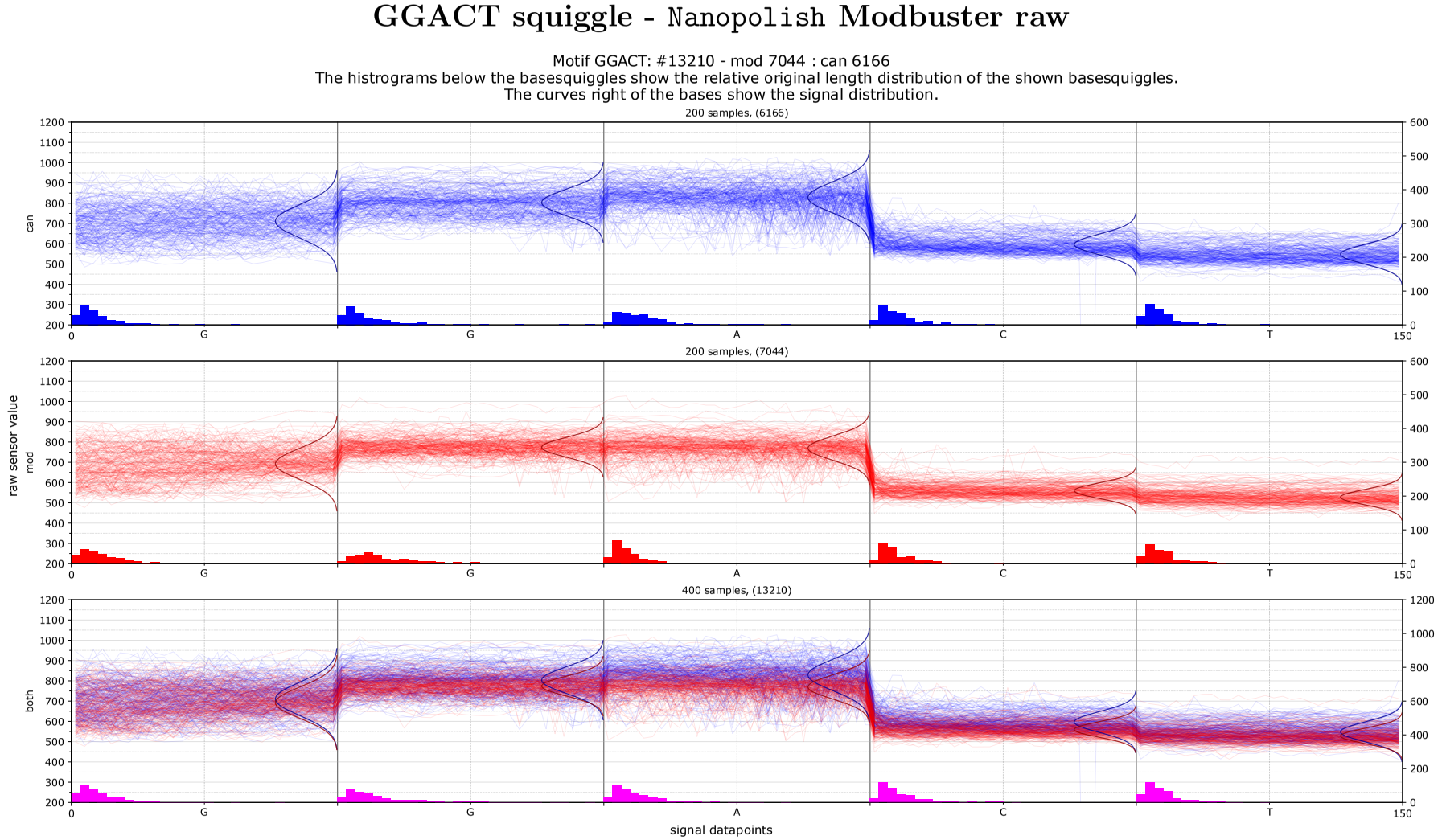

Figure 17: This picture shows 200 interpolated raw signals per class from the Modbuster dataset, using the `Nanopolish` segmentation of the 5' GGACT 3' motif. The unmodified motifs are called *can* and are shown in blue, while the m6A motifs are colored red and called *mod*.

Figure 16 visualizes the segmented signal of the `Taiyaki` tool. Within all motifs of the `Taiyaki` prepared datasets, the signal looks similar regarding the segmentation. The signal segment of the second G in the motif is concise, as indicated by the histogram in Figure 16 and the signal protrudes into the middle A base. Within A, the signal drops to another level. This level is most likely the actual signal of the C when comparing to the `Nanopolish` segmentation. `Taiyaki` seems to have significant issues with the signal segmentation. One major problem with the segmentation is the limited resolution with multiples of ten data points. The segmentation might not be a big problem for the basecallers, as they take a big chunk of the signal and do not take any segmentation into account. Big convolutional neural networks take the big signal chunk as their input to call the base sequence. This way, they work independently of the base segmentation. However, in my case, an optimal signal segmentation that represents the 5-mer events of the bases is favorable and needed. It enables a more detailed analysis of the signal and provides a better basis to build chunks. The models are built to process chunks of bases.

Another issue of the data can be seen in picture 17. It shows the raw signal of the GGACT motif in the Modbuster IVT dataset. The canonical and modified signals seem to split up much more than in the EpiNano dataset. This is most likely the result of the way the dataset was created. Every arrangement of five bases with a single-A as the central base appears within reads in this dataset, with some exceptions within adapter regions. Nevertheless, not every combination of these five base sequences is present. Also, not every nine base sequence with just one A in the middle exists in this dataset, as already mentioned before. A better way to construct such an IVT dataset would be to use every combination and arrangement of nine bases with a single-A at the central position. This prevents the outer base signals of a five base long motif showing shifts caused by neighboring modified m^6As. Currently, the signals of the G, C, and T bases in Figure 17 are also influenced by the previous and following m^6A and therefore, the signals separate much more. In the EpiNano dataset, not every fifth base is an A or m^6A and not only single-A sequences are present. When training on the EpiNano and evaluating the Modbuster dataset, this must be taken into account. More shifts within the Modbuster samples could

influence the prediction, as the model would not expect so many shifts because they were not present in the trainingset. Either way, the performance of the models must be interpreted carefully, mainly because the sample originates from IVT datasets rather than *in vivo* datasets. Nevertheless, it is feasible to compare the performances of the different models qualitatively and find out which data works best to transform the knowledge from one dataset to the other.

4.2 Model performances

I used the `test_models.py` script to check the accuracy of my models on the Modbuster IVT dataset. The models trained on the normalized and unnormalized datasets were just trained for 80 epochs because previous projects [33] and the results during this thesis show that the raw signal works better than the normalized or unnormalized one. Also, the signal plots in the Figures 13 - 17 above show, that within the raw signal the canonical and modified motifs separate much better and the different datasets are somewhat comparable. This is why the raw signal type models are trained for up to 200 epochs instead of 80 on the raw EpiNano datasets and have slightly different model architecture. The following key figures provide a measurement for the performance of the models:

1. accuracy = $\frac{\text{number of true predictions}}{\text{number of all predictions}}$
2. precision = $\frac{\text{number of true positive predictions}}{\text{number of positive predictions}}$
3. recall or true positive rate (TPR) = $\frac{\text{number of true positive predictions}}{\text{number of all positive samples}}$
4. false positive rate (FPR) = $\frac{\text{number of false positive predictions}}{\text{number of all negative samples}}$

After training, the model is checked for every tenth epoch from 80 to 200 for its accuracy on the respective Modbuster IVT dataset. The accuracy of every model after 80 epochs and the best performing model after every tenth epoch from epoch 80 to 200 can be seen in the following table.

Table 4: This table shows the accuracy of models 0 to 6 on the respective Modbuster evaluationset. The accuracy increases when using more training epochs on the models trained with the `Nanopolish` segmentation on the raw signal, as seen in models 4 and 6. The model trained on the raw signal with the `Taiyaki` segmentation shows a much higher increase when trained longer than 80 epochs. A reason for this behavior can be that `Taiyaki` has a systematic segmentation pattern in the case of m^6A that differs from a canonical A base, which the model learns to use when predicting the methylation status.

Model ID	Modbuster evaluation set	Acc. at epoch 80	Best acc. & epoch
0	`Nanopolish` & norm	0.629	0.629 & 80
1	`Taiyaki` & norm	0.662	0.662 & 80
2	`Nanopolish` & unorm	0.557	0.557 & 80
3	`Taiyaki` & unorm	0.610	0.610 & 80
4	`Nanopolish` & raw	0.705	0.708 & 190
5	`Taiyaki` & raw	0.687	0.729 & 190
6	`Nanopolish` & raw	0.692	0.694 & 110

4.2.1 High accuracy in training

The models with the best accuracy in the evaluation are compared in multiple key figures. The following plots are generated with `Tensorboard`, which is a tool to visualize and track metrics of neural networks [34]. They show the accuracy of the models while training on the 20% testset (Figure 18) and the loss per sample (Figure 19) for every epoch they were trained. `Tensorboard` does not plot the last epoch value in the graph.

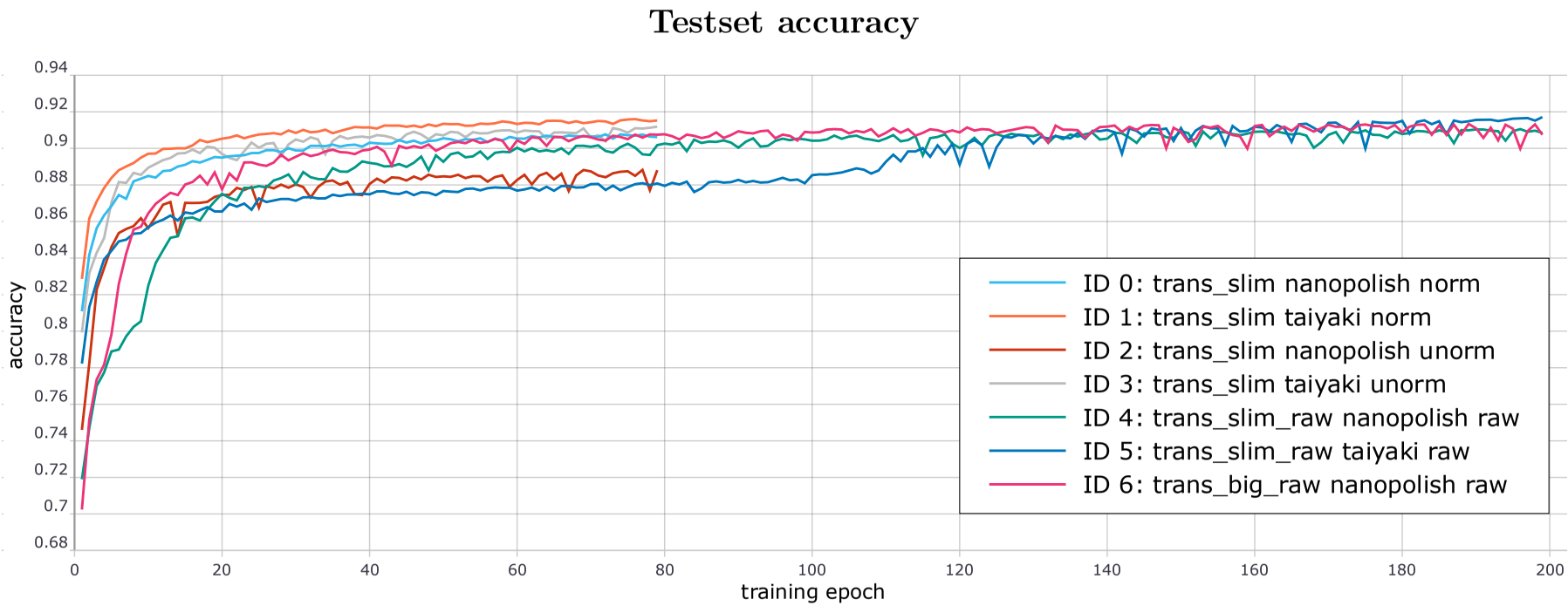

Figure 18: The graphs visualize the accuracy of the respective model on the 20% test split for every epoch while training. They were plotted using `Tensorboard`. `Tensorboard` does not show the last value of the training runs in this graph. Models trained on the normalized signal learn very fast and reach an overall high accuracy while training. This strongly indicates that a normalization of the signal is generally beneficial. Models trained on the unnormalized signal do not have such high accuracies on the 20% test split. The same behavior can be seen in models trained on the raw signal. When training for up to 200 epochs, the models can also reach similar accuracies as the models trained on the normalized signal on the 20% test split. Judging from this graph alone future steps for my work are to find a better normalization and keep the comparability between different datasets as seen before in the raw signal plots. Another idea would be to use even more training epochs to study the behavior of the models.

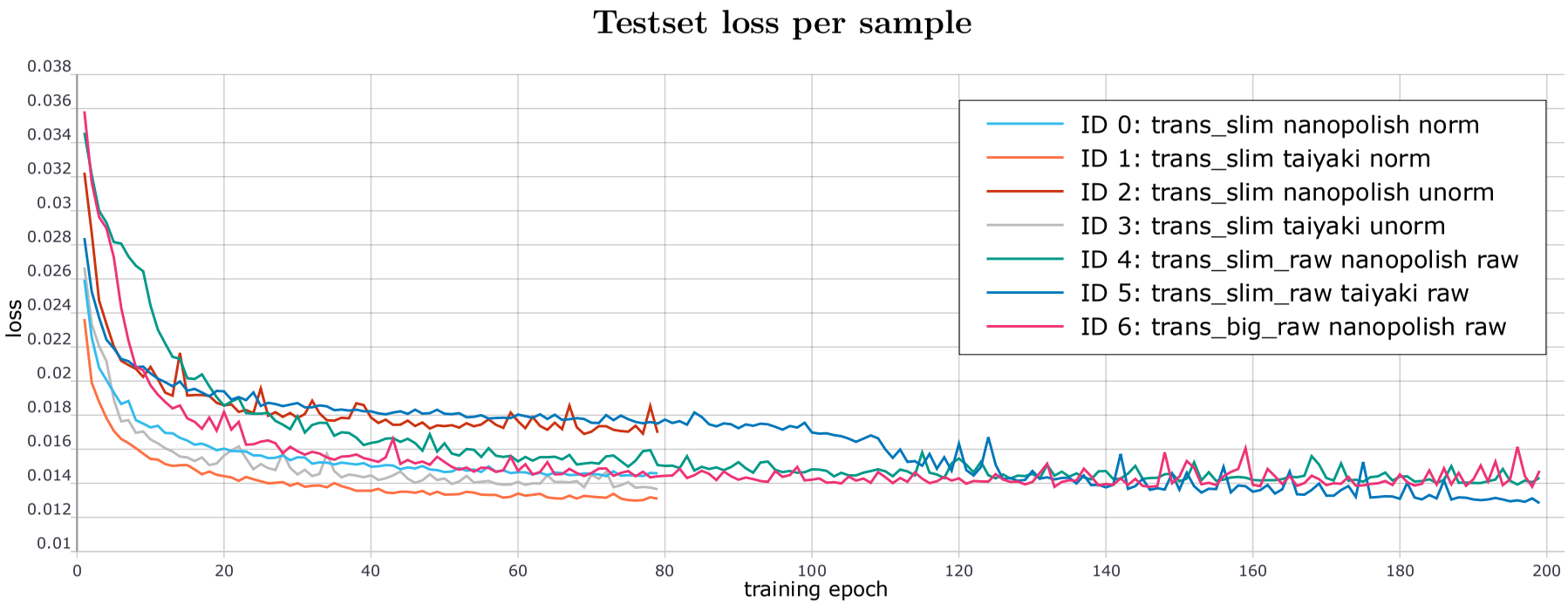

Figure 19: This plot is also generated by `Tensorboard` and shows the average loss per sample per epoch. The graphs show the same behavior as in Figure 18. Again the normalized signal is favorable for the models. The unnormalized and raw signals are harder to train a model. Nevertheless, the raw signal is better to transfer learned aspects between dataset, which can be seen in Figure 15, 16 and table 4. The graph does not show over-fitting behavior yet. If the models would over-fit on the 80% training split, then the loss on the 20% test split would rise again. Model 4 and 6 shows slight outliers going up and down, might be stuck in a local or global minimum. Model 5 trained on the `Taiyaki` prepared EpiNano set might still improve with more epochs.

4.2.2 The models yield promising evaluation results

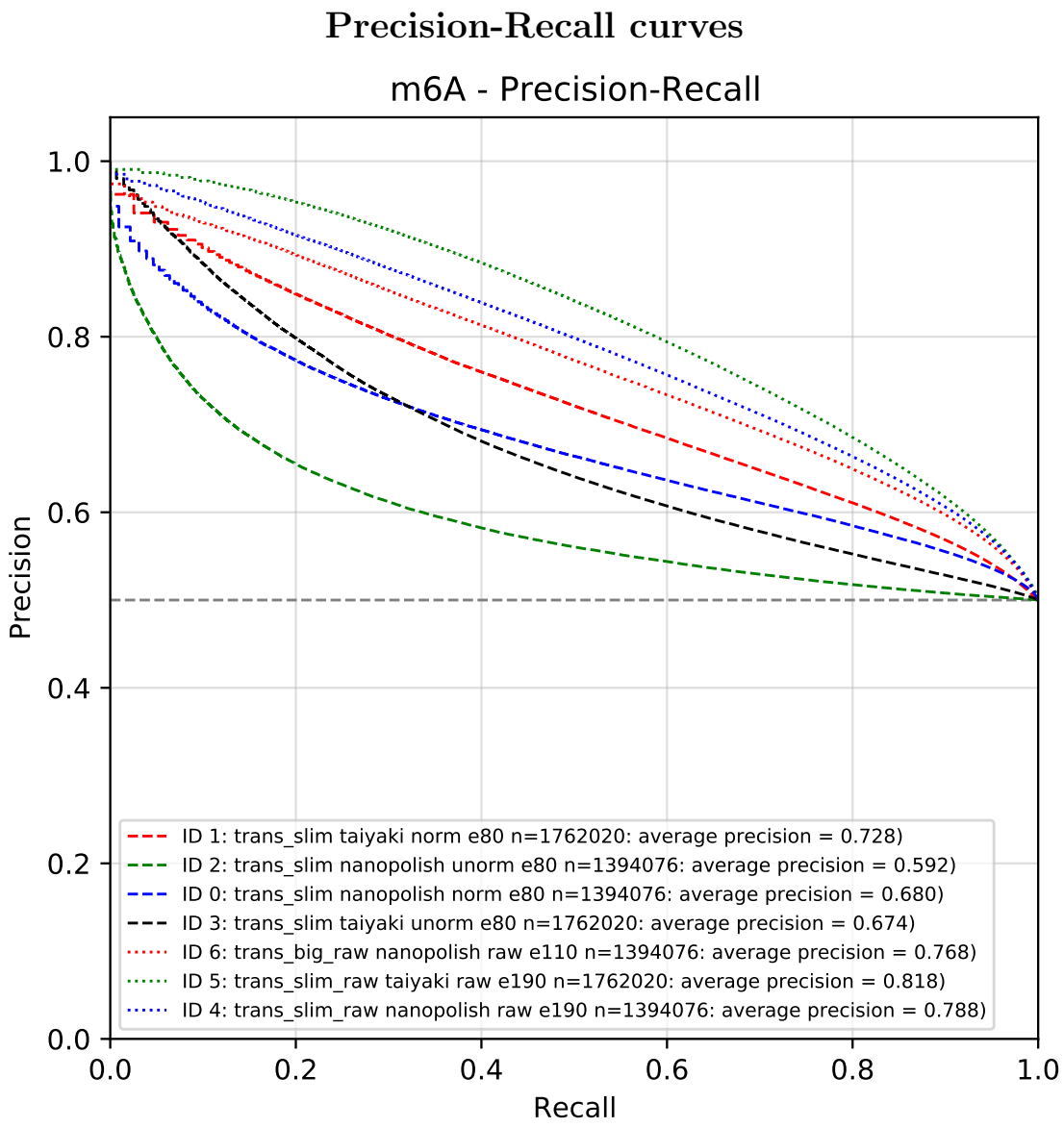

Figure 20: The precision-recall curves show the tradeoff between precision and recall for different thresholds for binary classification. If the classification threshold starts near one and decreases, then more samples are classified as positive by the model, thus increasing the TPR or recall and decreasing the precision. The grey line marks the performance of a random guesser.

Figure 20 compares the models shown in table 3 on the respective Modbuster evaluationset regarding precision and recall. Models trained on a `Taiyaki` prepared EpiNano dataset were also evaluated on a `Taiyaki` prepared Modbuster dataset. The same applies to models trained and evaluated on a `Nanopolish` prepared dataset. The models 4, 5, and 6 that were trained on the EpiNano raw signals

separate from the rest and reach higher precision values than the other models. After these three models, the model trained on the `Taiyaki` normalized signal performs the best. Also, the best overall model was trained and tested on the `Taiyaki` prepared raw signal. I suspect that the `Taiyaki` prepared signal presents some segmentation pattern in a small number of samples that the model uses to reach better prediction performances. One such pattern can be seen in Figure 16, where the red m^6A signal protrudes into the segmentation of the following C, which does not often appear in the unmodified variant. These patterns might enable model 5 to reach 0.729% accuracy instead of just 0.708% as model 4 in table 4. Unexpectedly, model 5 can predict the methylation status better than other models with a bad signal segmentation as if the segmentation does not matter that much. Alternatively, the transformer model and the attention mechanism could filter out the necessary signal parts regardless of the segmentation quality. Figure 21 supports the performance on the respective Modbuster evaluationsets seen in the precision-recall plot. The models have the same order of performance, this time regarding the TPR and FPR, with models 4, 5, and 6 being the best. They have the highest area under the ROC curve (AUC). The models trained on the unnormalized signal perform the worst on the Modbuster evaluationset and also not very well on the 20% test split. The trans_slim architecture was used for these models, which does not include a linear layer stack in front of the transformer part to preprocess the values. However, the unnormalized signal values are like the raw signal values not spread around 0. Hence, it would be favorable to include a better normalization or preprocessing by linear layers to improve the performance here. They might perform better with a more fitting architecture or another preprocessing because the unnormalized signal, like the raw signal, shows a little bit of a separation, seen in Figure 14. In the shown plot, the separation does not appear in the middle modified base. It appears in the bases around it, which are also in the sphere of influence (five bases) of the modification within the pore sensor.

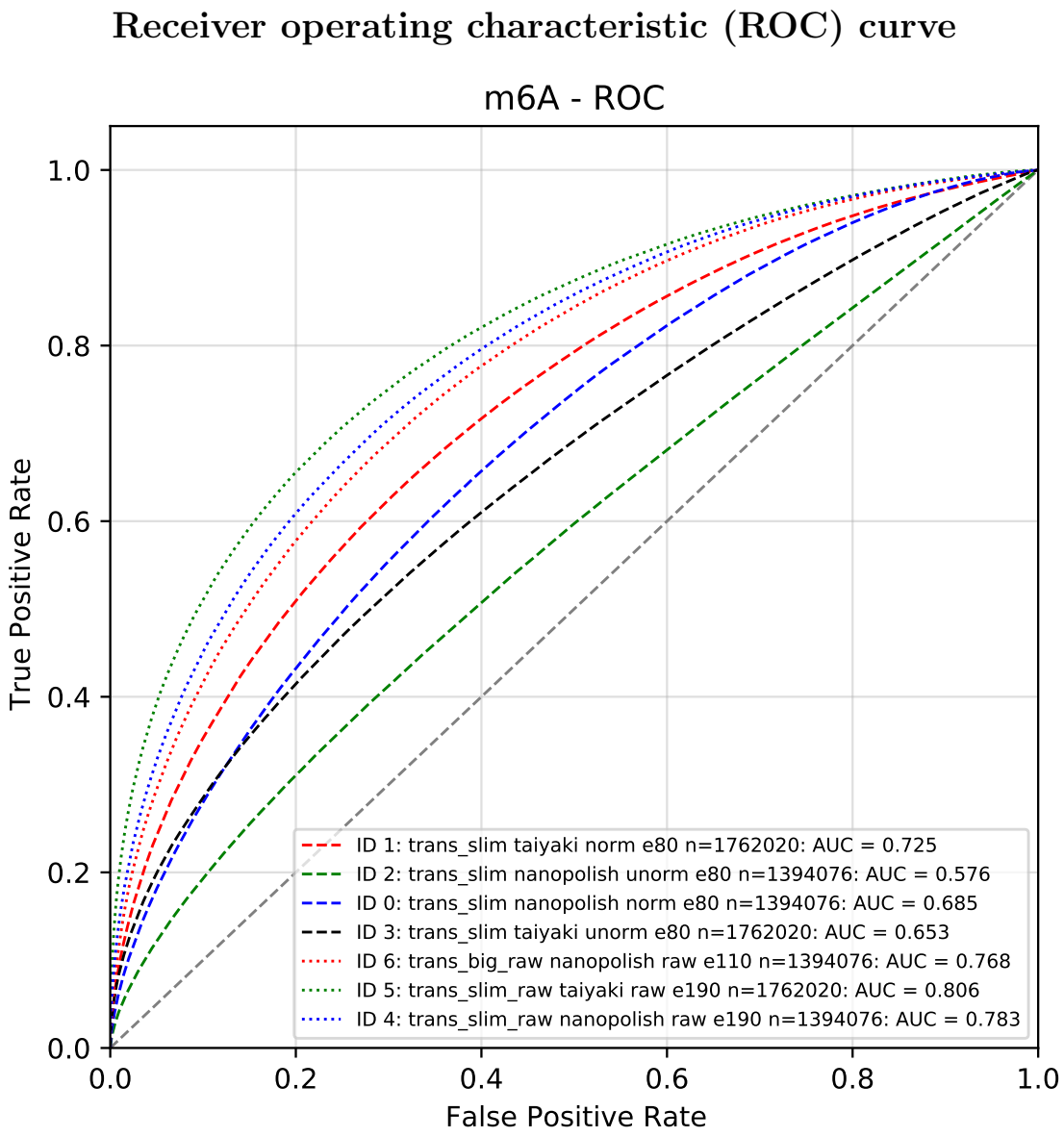

Figure 21: The ROC curve plots the TPR vs. the FPR on different classification thresholds for binary classification. If the classification threshold starts near one and decreases, then more samples are classified as positive by the model, thus increasing the TPR and FPR. Again, the grey line marks the performance of a random guesser.

Overall, models 4 and 5 perform much better than any other models of this work and from my project work on this topic [33]. The result shows that using the actual signal works better than aggregated features when using deep neural networks. The models are reasonably capable of predicting m^6A predictions in another IVT dataset on the single-nucleotide resolution on a set containing all possible modified and unmodified single-A 5-mer motifs with the A in the middle of the sequence.

5 Discussion

This thesis demonstrates a new way to predict the RNA m^6A methylation status using the nanopore signal of RNA reads with the help of neural networks. The prediction is feasible on *in vitro* transcription data, but it has yet to be tested on *in vivo* data. It is essential to have a good signal preprocessing to increase the performance for the m^6A prediction, as different signal types like the raw sensor signal, unnormalized pA, and normalized pA signal show different levels of signal separation, also called shift, between the unmodified and m^6A modified sequences. There are still some issues with the data and many ideas to improve the prediction that I want to discuss shortly. Future steps should also be to validate my models on datasets of other groups to have results for multiple datasets. Additionally, I could look at the results on individual motifs or groups of motifs. As mentioned in the introduction, the "DRACH" motif is targeted by methyltransferases and demethylases and therefore of particular interest to current research. It would be of advantage if the models can confidently predict the m^6A methylation within these motifs.

5.1 The data problem

One issue of the datasets is the way they are constructed. IVT data provides training and test data to test if methylation prediction from the nanopore signal is possible. However, the reads that are used to get the nanopore signal do not appear this way *in vivo*. The datasets that I used in my project are either entirely- or unmethylated at all adenines. *In vivo* a transcript is not methylated at all adenines. There are different modifications within one read at different positions that might even appear all the time. This means the results of the models must be interpreted carefully, as the models can predict the methylation status on an IVT dataset. However, it is unclear if it can do so on an *in vivo* dataset. This means that in future work, a ground-truth *in vivo* dataset is necessary to see if the models can keep up their performance.

5.2 Preprocessing of the signal

Another point of improvement is the signal (pre-) processing. As seen in the signal plots and already explained, the signal read normalization suggested by `Taiyaki` could be unfavorable with many m^6A methylations. *In vivo* many different modifications could appear within a single transcript. This means that another signal normalization might be favorable. One such normalization could be to take the mean and standard deviation of completely unmodified 5-mer events and use them to normalize each base signal independently. That also requires perfect signal segmentation. `Nanopolish eventalign` provides a reasonably good segmentation but cannot segment every read or misses some parts of reads. `Taiyaki` should not be used to get a signal segmentation in future work. Here, it could be of advantage to search for other segmentation and resquiggling tools. Another future step could be to create a new segmentation tool or refine the existing segmentation of, for example, `Nanopolish eventalign`.

One major influence on my data is the interpolation method. Currently, I interpolate linearly, which compresses the signal and reduces the variance. Another idea is to change the implementation of the interpolation algorithm to choose between different interpolation methods, for example, a sigmoidal interpolation between two data points instead of linear interpolation. Another would be to use distribution models to interpolate the signal. The interpolation method should keep most of the original signal's properties while changing the length or amount of data points to a given number.

Another approach could be to use existing signal processing methods to preprocess the nanopore signal. Such a preprocessing could be to reduce the noise within the signal with filters like the median filter. The Fourier-Transformation or the Wavelet-Transformation could also be helpful to analyze different frequencies of the signal and use these as input parameters.

5.3 More features for the prediction

It might also be beneficial to change the model architecture and add more features. The activation function could be changed, for example. There are other activation functions that I could try in the future, for example, the GELU activation function, which is widely used in natural language processing models like GPT-2 [28].

Additional features like some pore properties or information from the basecaller could also help improve the methylation prediction. Pore properties describe, for example, the state of the pore that was used to sequence the read or produced the signal of the read. Such properties could be the applied electrical voltage on the membrane, sequencing time within the whole sequencing run, temperature, and others. The basecaller has some additional information as well. `Guppy` for example, uses transition probabilities to generate the base sequence from the nanopore signal [40]. These transition probabilities could also be used as input features, as they might show a pattern when the `Guppy` basecaller sees the signal of a modified base. Another feature could be if `Guppy` had a basecalling error near the modification that was corrected using one of the resquiggler tools. I can also change the signal length of the input base signals. The easiest future improvement for the m^6A prediction could be to combine all three signal types in the input of a single neural network instead of looking at them separately, as I did in this thesis. This way, the network could extract the advantageous properties of each signal type for the prediction.

Adjusting the model architecture could also help to increase the performance. I already tried to make the model overall bigger, but the model did not show such a good performance as the normal one. I also tried out long short-term memory (LSTM) layers instead of the transformer part, but they did not yield better accuracies. LSTMs are artificial recurrent neural networks that are built to learn long and short-term sequence dependencies [14]. A dropout in combination with bigger models while training could also help to prevent over-fitting.

Another adjustment should be to adapt the architecture to the input. In the case of the unnormalized signal, a linear layer stack in front of the transformer part could help to preprocess the signal, similar to the architecture for the raw signal. I did not do this, as I adjusted the architecture afterward and had little time to retrain

every model on the unnormalized values with another architecture.
Apart from feature engineering and the architecture adaptation, a future step could be to combine the methylation predictions for multiple reads that belong to the same gene. This is introduced in methods that use the basecaller error rate for modification predictions, which can refine the prediction. If the prediction is correct, one could figure out the modification rate and the modification position of a transcript.

5.4 More exploration of the data

We need a better insight into which features are essential for the prediction. Other approaches to predict the methylation could be beneficial to get this knowledge. It is unclear what the deep neural networks do with the input features and which features are essential to predict the methylation. They act as a black box in this case. Approaches like general linear models or support vector machines could provide a better insight into the data and reveal essential features to distinguish a methylated signal from an unmodified one. By explorative investigations into the data, as I did with the signal plots, essential features can be estimated. This explorative work is necessary to improve models, as well as to understand how the data looks.

6 Conclusion

My master thesis proves that it is possible to predict the m^6A methylation status on RNA level within IVT data using only the nanopore signal and information about the reference sequence. A good base signal segmentation is the foundation for further processing steps and a more detailed analysis of the RNA base signals. The signal shift depends strongly on the sequence content and also on the signal type. Signal processing methods, like the Fourier transformation, Wavelet transformation, or signal filters, could open up new insights into the nanopore signal. These methods could help to reduce signal noise and filter the signal to find the essential signal parts produced by the 5-mers passing the pore's sensor that contribute the most to the modification prediction. There are also other ideas to improve the RNA modification prediction, such as including more features and changing the model architecture. The gained knowledge of this work is helpful to develop methods and tools for the RNA modification detection in ONT sequencing reads. It also provides workflows and strategies to investigate other modifications apart from m^6A, which I want to study in the future. The major question to answer will be whether the models also perform well on *in vivo* data. It would be groundbreaking if a reliable tool can be developed that can call many different RNA modifications in *in vivo* reads from the nanopore signal alone.

Appendix

Used commands and parameters:

- python3 raw_signal/nanopolish_prepare.py <nanopolish_summary.csv> <nanopolish_result.csv> <taiyaki_output.hdf5> <mod_read.ids> <nomod_read.ids> <output_prepared_dataset.hdf5> <model_architecture> [--signal_size <signal_size>] [--mod <nomodbase,modbase>] [--datamode <datamode>]

  Modbase defines the name of the modified base and nomodbase the regular counterpart. Datamode defines the signal type to prepare.

- python3 raw_signal/taiyaki_prepare.py <model_architecture> <taiyaki_output.hdf5> <output_prepared_dataset.hdf5> [--signal_size <signal_size>] [--modbase <nomodbase,modbase>] [--datamode <datamode>]

- python3 raw_signal/balance_dataset.py <dataset.hdf5> <balanced_dataset_name>

- python3 utils/plot_signals.py <balanced_dataset.hdf5> <output_parent_folder> [--mod <modbase>] [--datamode <datamode>] [--kmer_models <kmer.model>]

  The kmer model can be provided with the unnormlized datamode. Datamode defines the signal type to choose the correct axis limits.

- python3 raw_signal/train_model.py <model_architecture> <traningsset> [--num_epochs <num_epochs>] [--batch_size <batch_size>] [--output_parent_folder <output_path>]

  Num_epochs defines the number of training epochs. Batch_size defines the number of samples per training cycle.

- python3 raw_signal/test_model.py <model.torch> <model.config> <testset.hdf5>

The following USB device contains:

- additional figures for the GGACT signal types and segmentations,

- all major scripts described in this thesis and additional scripts.

GGACT squiggle - `Taiyaki` Epinano normalized

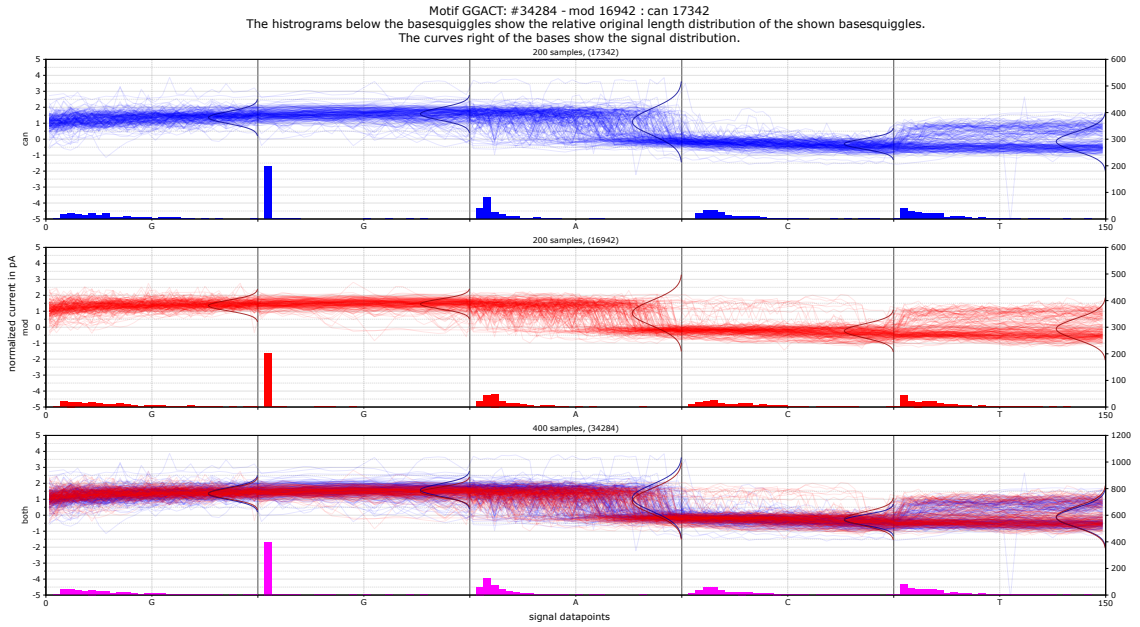

Figure S1: This picture shows 200 interpolated normalized pA signals per class from the EpiNano dataset, using the `Taiyaki` segmentation of the 5' GGACT 3' motif. The unmodified motifs are called *can* and are shown in blue, while the m6A motifs are coloured red and called *mod*.

GGACT squiggle - `Taiyaki` Modbuster normalized

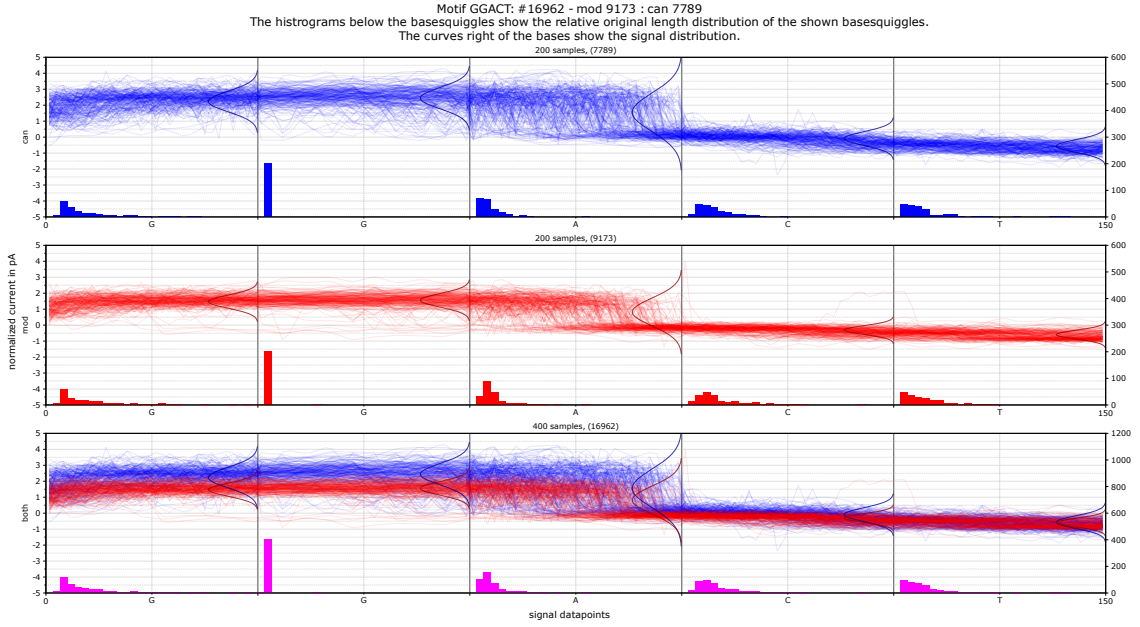

Figure S2: This picture shows 200 interpolated normalized pA signals per class from the Modbuster dataset, using the `Taiyaki` segmentation of the 5' GGACT 3' motif. The unmodified motifs are called *can* and are shown in blue, while the m6A motifs are coloured red and called *mod*.

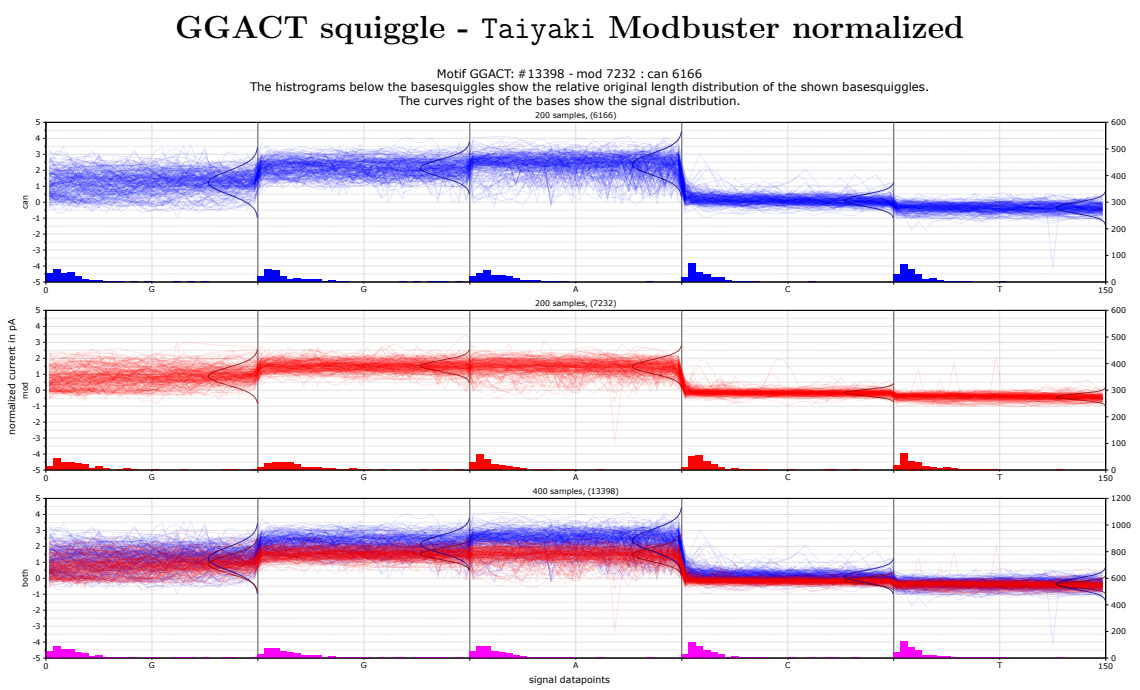

Figure S3: This picture shows 200 interpolated normalized signals per class from the Modbuster dataset, using the `Nanopolish` segmentation of the 5' GGACT 3' motif. The unmodified motifs are called *can* and are shown in blue, while the m6A motifs are coloured red and called *mod*.

Bibliography

[1] Anaconda software distribution, 2020.

[2] C. R. Alarcón, H. Goodarzi, H. Lee, X. Liu, S. Tavazoie, and S. F. Tavazoie. HNRNPA2b1 is a mediator of m6a-dependent nuclear RNA processing events. *Cell*, 162(6):1299–1308, Sept. 2015.

[3] C. R. Alarcón, H. Lee, H. Goodarzi, N. Halberg, and S. F. Tavazoie. N6-methyladenosine marks primary microRNAs for processing. *Nature*, 519(7544):482–485, Mar. 2015.

[4] A. G. Baydin and B. A. Pearlmutter. Automatic differentiation of algorithms for machine learning. *CoRR*, abs/1404.7456, 2014.

[5] T. Chen, Y.-J. Hao, Y. Zhang, M.-M. Li, M. Wang, W. Han, Y. Wu, Y. Lv, J. Hao, L. Wang, A. Li, Y. Yang, K.-X. Jin, X. Zhao, Y. Li, X.-L. Ping, W.-Y. Lai, L.-G. Wu, G. Jiang, H.-L. Wang, L. Sang, X.-J. Wang, Y.-G. Yang, and Q. Zhou. m6a RNA methylation is regulated by MicroRNAs and promotes reprogramming to pluripotency. *Cell Stem Cell*, 16(3):338, Mar. 2015.

[6] H. Coker, G. Wei, and N. Brockdorff. m6a modification of non-coding RNA and the control of mammalian gene expression. *Biochimica et Biophysica Acta (BBA) - Gene Regulatory Mechanisms*, 1862(3):310–318, Mar. 2019.

[7] M. Frye, B. T. Harada, M. Behm, and C. He. Rna modifications modulate gene expression during development. *Science*, 361(6409):1346–1349, 2018.

[8] D. R. Garalde, E. A. Snell, D. Jachimowicz, B. Sipos, J. H. Lloyd, M. Bruce, N. Pantic, T. Admassu, P. James, A. Warland, M. Jordan, J. Ciccone, S. Serra, J. Keenan, S. Martin, L. McNeill, E. J. Wallace, L. Jayasinghe, C. Wright, J. Blasco, S. Young, D. Brocklebank, S. Juul, J. Clarke, A. J. Heron, and D. J. Turner. Highly parallel direct RNA sequencing on an array of nanopores. *Nature Methods*, 15(3):201–206, Jan. 2018.

[9] S. Geula, S. Moshitch-Moshkovitz, D. Dominissini, A. A. Mansour, N. Kol, M. Salmon-Divon, V. Hershkovitz, E. Peer, N. Mor, Y. S. Manor, M. S. Ben-Haim, E. Eyal, S. Yunger, Y. Pinto, D. A. Jaitin, S. Viukov, Y. Rais, V. Krupalnik, E. Chomsky, M. Zerbib, I. Maza, Y. Rechavi, R. Massarwa, S. Hanna, I. Amit, E. Y. Levanon, N. Amariglio, N. Stern-Ginossar, N. Novershtern, G. Rechavi, and J. H. Hanna. m6a mRNA methylation facilitates resolution of naïve pluripotency toward differentiation. *Science*, 347(6225):1002–1006, Jan. 2015.

[10] A. Graves. Sequence transduction with recurrent neural networks. *CoRR*, abs/1211.3711, 2012.

[11] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. Fernández del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant. Array programming with NumPy. *Nature*, 585:357–362, 2020.

[12] R. Hecht-Nielsen. Theory of the backpropagation neural network - based on "nonindent" by robert hecht-nielsen, which appeared in proceedings of the international joint conference on neural networks 1, 593–611, june 1989. © 1989 IEEE. pages 65–93, 1992.

[13] S. Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6:107–116, 04 1998.

[14] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, Nov. 1997.

[15] N. Jonkhout, J. Tran, M. A. Smith, N. Schonrock, J. S. Mattick, and E. M. Novoa. The RNA modification landscape in human disease. *RNA*, 23(12):1754–1769, Aug. 2017.

[16] Q. Koziol and D. Robinson. Hdf5, 2018.

[17] A. H. Laszlo, I. M. Derrington, H. Brinkerhoff, K. W. Langford, I. C. Nova, J. M. Samson, J. J. Bartlett, M. Pavlenok, and J. H. Gundlach. Detection and mapping of 5-methylcytosine and 5-hydroxymethylcytosine with nanopore MspA. *Proceedings of the National Academy of Sciences*, 110(47):18904–18909, Oct. 2013.

[18] M. Lee, B. Kim, and V. Kim. Emerging roles of rna modification: m6a and u-tail. *Cell*, 158(5):980–987, 2014.

[19] B. Linder, A. V. Grozhik, A. O. Olarerin-George, C. Meydan, C. E. Mason, and S. R. Jaffrey. Single-nucleotide-resolution mapping of m6a and m6am throughout the transcriptome. *Nature Methods*, 12(8):767–772, June 2015.

[20] H. Liu, O. Begik, M. C. Lucas, J. M. Ramirez, C. E. Mason, D. Wiener, S. Schwartz, J. S. Mattick, M. A. Smith, and E. M. Novoa. Accurate detection of m6a RNA modifications in native RNA sequences. *Nature Communications*, 10(1), Sept. 2019.

[21] Q. Liu, L. Fang, G. Yu, D. Wang, C.-L. Xiao, and K. Wang. Detection of DNA base modifications by deep recurrent neural network on oxford nanopore sequencing data. *Nature Communications*, 10(1), June 2019.

[22] D. A. Lorenz, S. Sathe, J. M. Einstein, and G. W. Yeo. Direct RNA sequencing enables m6a detection in endogenous transcript isoforms at base-specific resolution. *RNA*, 26(1):19–28, Oct. 2019.

[23] S. Mannor, D. Peleg, and R. Rubinstein. The cross entropy method for classification. In *Proceedings of the 22nd international conference on Machine learning - ICML '05*. ACM Press, 2005.

[24] K. D. Meyer. DART-seq: an antibody-free method for global m6a detection. *Nature Methods*, 16(12):1275–1280, Sept. 2019.

[25] K. D. Meyer, D. P. Patil, J. Zhou, A. Zinoviev, M. A. Skabkin, O. Elemento, T. V. Pestova, S.-B. Qian, and S. R. Jaffrey. 5′ UTR m6a promotes cap-independent translation. *Cell*, 163(4):999–1010, Nov. 2015.

[26] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[27] P. N. Pratanwanich, F. Yao, Y. Chen, C. W. Q. Koh, Y. K. Wan, C. Hendra, P. Poon, Y. T. Goh, P. M. L. Yap, J. Y. Chooi, W. J. Chng, S. B. Ng, A. Thiery, W. S. S. Goh, and J. Göke. Identification of differential RNA modifications from nanopore direct RNA sequencing with xPore. *Nature Biotechnology*, July 2021.

[28] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners. 2019.

[29] F. J. Rang, W. P. Kloosterman, and J. de Ridder. From squiggle to basepair: computational approaches for improving nanopore sequencing read accuracy. *Genome Biology*, 19(1), July 2018.

[30] Y. Saletore, K. Meyer, J. Korlach, I. D. Vilfan, S. Jaffrey, and C. E. Mason. The birth of the epitranscriptome: deciphering the function of RNA modifications. *Genome Biology*, 13(10):175, 2012.

[31] J. Schreiber, Z. L. Wescoe, R. Abu-Shumays, J. T. Vivian, B. Baatar, K. Karplus, and M. Akeson. Error rates for nanopore discrimination among cytosine, methylcytosine, and hydroxymethylcytosine along individual DNA strands. *Proceedings of the National Academy of Sciences*, 110(47):18910–18915, Oct. 2013.

[32] J. T. Simpson, R. Workman, P. C. Zuzarte, M. David, L. J. Dursi, and W. Timp. Detecting DNA methylation using the oxford nanopore technologies MinION sequencer. Apr. 2016.

[33] J. Spangenberg. Project: Modbuster thesis. 2021.

[34] TensorFlow Developers. Tensorflow, 2021.

[35] G. Van Rossum and F. L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.

[36] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[37] L. P. Vu, B. F. Pickering, Y. Cheng, S. Zaccara, D. Nguyen, G. Minuesa, T. Chou, A. Chow, Y. Saletore, M. MacKay, J. Schulman, C. Famulare, M. Patel, V. M. Klimek, F. E. Garrett-Bakelman, A. Melnick, M. Carroll, C. E. Mason, S. R. Jaffrey, and M. G. Kharas. The n6-methyladenosine (m6a)-forming enzyme METTL3 controls myeloid differentiation of normal hematopoietic and leukemia cells. *Nature Medicine*, 23(11):1369–1376, Sept. 2017.

[38] X. Wang, Z. Lu, A. Gomez, G. C. Hon, Y. Yue, D. Han, Y. Fu, M. Parisien, Q. Dai, G. Jia, B. Ren, T. Pan, and C. He. N6-methyladenosine-dependent regulation of messenger RNA stability. *Nature*, 505(7481):117–120, Nov. 2013.

[39] Y. Wang, Y. Li, J. I. Toth, M. D. Petroski, Z. Zhang, and J. C. Zhao. N6-methyladenosine modification destabilizes developmental regulators in embryonic stem cells. *Nature Cell Biology*, 16(2):191–198, Jan. 2014.

[40] R. R. Wick, L. M. Judd, and K. E. Holt. Performance of neural network basecalling tools for oxford nanopore sequencing. *Genome Biology*, 20(1), June 2019.

[41] K. Xu, Y. Yang, G.-H. Feng, B.-F. Sun, J.-Q. Chen, Y.-F. Li, Y.-S. Chen, X.-X. Zhang, C.-X. Wang, L.-Y. Jiang, C. Liu, Z.-Y. Zhang, X.-J. Wang, Q. Zhou, Y.-G. Yang, and W. Li. Mettl3-mediated m6a regulates spermatogonial differentiation and meiosis initiation. *Cell Research*, 27(9):1100–1114, Aug. 2017.

[42] X. Yang, Y. Yang, B.-F. Sun, Y.-S. Chen, J.-W. Xu, W.-Y. Lai, A. Li, X. Wang, D. P. Bhattarai, W. Xiao, H.-Y. Sun, Q. Zhu, H.-L. Ma, S. Adhikari, M. Sun, Y.-J. Hao, B. Zhang, C.-M. Huang, N. Huang, G.-B. Jiang, Y.-L. Zhao, H.-L. Wang, Y.-P. Sun, and Y.-G. Yang. 5-methylcytosine promotes mRNA export — NSUN2 as the methyltransferase and ALYREF as an m5c reader. *Cell Research*, 27(5):606–625, Apr. 2017.

[43] S. Zaccara and S. R. Jaffrey. A unified model for the function of YTHDF proteins in regulating m6a-modified mRNA. *Cell*, 181(7):1582–1595.e18, June 2020.

[44] X. Zhao, Y. Yang, B.-F. Sun, Y. Shi, X. Yang, W. Xiao, Y.-J. Hao, X.-L. Ping, Y.-S. Chen, W.-J. Wang, K.-X. Jin, X. Wang, C.-M. Huang, Y. Fu, X.-M. Ge, S.-H. Song, H. S. Jeong, H. Yanagisawa, Y. Niu, G.-F. Jia, W. Wu, W.-M. Tong, A. Okamoto, C. He, J. M. R. Danielsen, X.-J. Wang, and Y.-G. Yang. FTO-dependent demethylation of n6-methyladenosine regulates mRNA splicing and is required for adipogenesis. *Cell Research*, 24(12):1403–1419, Nov. 2014.

[45] G. Zheng, J. A. Dahl, Y. Niu, P. Fedorcsak, C.-M. Huang, C. J. Li, C. B. Vågbø, Y. Shi, W.-L. Wang, S.-H. Song, Z. Lu, R. P. Bosmans, Q. Dai, Y.-J. Hao, X. Yang, W.-M. Zhao, W.-M. Tong, X.-J. Wang, F. Bogdan, K. Furu, Y. Fu, G. Jia, X. Zhao, J. Liu, H. E. Krokan, A. Klungland, Y.-G. Yang, and C. He. ALKBH5 is a mammalian RNA demethylase that impacts RNA metabolism and mouse fertility. *Molecular Cell*, 49(1):18–29, Jan. 2013.

[46] K. I. Zhou and T. Pan. Structures of the m 6 a methyltransferase complex: Two subunits with distinct but coordinated roles. *Molecular Cell*, 63(2):183–185, July 2016.

Danksagung

Ich möchte mich sehr herzlich bei meinen Betreuern Sebastian Krautwurst, Christian Höner zu Siederdissen und Manja Marz bedanken, die mir bei der Ideenfindung zur Lösung von Problemen intensiv zur Seite standen und sich bei Rückfragen gerne viel Zeit genommen haben diese zu beantworten und mir hoffentlich auch im weiteren Verlauf meiner akademischen Laufbahn zur Seite stehen werden.

Jakob Taulin und Alexander Henoch möchte ich besonders danken, da sie mir auch während des COVID-19 Lockdowns und bei geistigen Flauten ein Lächeln auf das Gesicht zauber konnten.

Ich möchte mich bei der gesamten RNA Jena Bioinformatics & High-Throughput Analysis Gruppe für die gemeinsame Zeit, das Mittagessen und besonders bei Muriel für ihre Schätzfragen des Tages bedanken.

Für die konstruktive Kritik zu meiner Arbeit möchte ich Sandra Triebel, Alexander Henoch, Jakob Taulin, Sebastian Krautwurst und Christian Höner zu Siederdissen herzlich danken.

Und ein Dankeschön an Sie, lieber Leser, dass Sie meine Arbeit gelesen haben!

Eigenständigkeitserklärung

Ich erkläre, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe. Ich habe keine Einwände die vorliegende Masterarbeit für die öffentliche Benutzung im Universitätsarchiv zur Verfügung zu stellen.

Declaration of authorship

I declare that I have prepared this master thesis independently and only using the referenced sources and resources. I have no objection to making this master thesis available for public use in the university archive.

Jena,

.....