

UNIVERSITY OF ZAGREB  
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

MASTER THESIS num. 2466

# **Rapid Microbe Detection Using Deep Learning**

Sara Bakić

Zagreb, June 2021.

UNIVERSITY OF ZAGREB  
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

MASTER THESIS num. 2466

# **Rapid Microbe Detection Using Deep Learning**

Sara Bakić

Zagreb, June 2021.

*Umjesto ove stranice umetnite izvornik Vašeg rada.  
Kako biste uklonili ovu stranicu, obrišite naredbu \izvornik.*



# CONTENTS

<b>1. Introduction</b>	<b>1</b>
<b>2. Dataset</b>	<b>4</b>
2.1. File formats . . . . .	5
2.1.1. FASTA and FASTQ . . . . .	5
2.1.2. PAF . . . . .	6
2.2. Preprocessing . . . . .	6
2.2.1. Reference-sampled chunks . . . . .	7
2.2.2. Real reads . . . . .	8
2.2.3. Dimensionality reduction . . . . .	8
<b>3. Methods</b>	<b>10</b>
3.1. Overview . . . . .	10
3.1.1. Artificial Neural Networks . . . . .	11
3.1.2. Embedding Layer . . . . .	13
3.1.3. Attention . . . . .	15
3.1.4. Transformers . . . . .	17
3.2. Representation Models . . . . .	21
3.2.1. Triplet Network . . . . .	21
3.3. Detection from representations . . . . .	23
3.3.1. Parametric classification . . . . .	23
3.3.2. K-nearest neighbors . . . . .	24
3.4. Visualization . . . . .	26
3.5. Evaluation Metrics . . . . .	27
3.6. Technical Stack . . . . .	28
<b>4. Experiments and Results</b>	<b>30</b>
4.1. Training on reference-sampled chunks . . . . .	31
4.2. Training on a combination of reference-sampled chunks and real reads . . . . .	36
4.3. Extensions for longer reads . . . . .	40

4.3.1. Training with longer input sequences . . . . .	40
4.3.2. Averaging representation learning . . . . .	40
4.3.3. Majority voting classification learning . . . . .	43
4.4. Discussion . . . . .	44
<b>5. Conclusion</b>	<b>45</b>
<b>Bibliography</b>	<b>47</b>

# 1. Introduction

Bioinformatics is an interdisciplinary scientific field that develops acquisition, storage, analysis, and dissemination techniques to understand biological data. Biological datasets are, in general, large and complex which is why the developed techniques have to be fast and accurate. The term "biological data" usually refers to genes, DNA, RNA, or protein, and the developed techniques can be particularly useful when comparing genetic material between different organisms.

Microbes are microscopic organisms, which may exist as single cells, multiple cells, or as a colony of cells. They are too small to be seen by the naked eye and live in almost every habitat, the deserts, the ocean floor, geysers, high in the atmosphere, and deep in the Earth showing they can adapt to living conditions varying from mild conditions to extremely high or low temperatures, high pressure, and even high radiation environments. The most common types of microbes are bacteria, viruses, and fungi.

Microbes are really important in everyday life for humans, as well. A healthy human is a home to millions of microbes, also called microorganisms, making up the human microbiota, including the essential gut flora. However, some microbes are pathogens responsible for infectious diseases and are as such targets of hygiene measures. Microbes serve to ferment foods (eg. beer or wine) and treat sewage and produce fuel, enzymes, and other bioactive compounds.

By studying the genetic material of microbes, a deeper understanding of their biological components and some insight into how their genetic configuration contributes to the characteristics that distinguish one microbe species from another is gotten. Since the discovery of microorganisms during the period 1665-1885, they have been the objective of many genetic researches and applications, and were even used to study evolution. With the accelerating development of sequencing techniques, new possibilities for microbial genetics were introduced and microbial genetics has advanced tremendously.

Since the entire biological information is stored within DNA or RNA strand(s), what bioinformaticians are interested in are the sequences of nucleotides in those strands. There are four canonical DNA bases, thymine (T), adenine (A), cytosine (C), and guanine (G), and the process of determining the physical order of those bases is called sequencing. Therefore,

the first step in every research is the collection of needed biological data through sequencing. While there are different sequencing approaches, the dominant sequencing strategy nowadays is, so-called, Nanopore ("third-generation", "long-read") DNA sequencing. The Nanopore sequencing process can be separated into three main stages:

1. sample preparation - obtaining nucleotide specimen used in the experiment
2. signal collection - measuring electrical current signals using Nanopore sequencing device, e.g. MinION
3. base calling - application of base caller to current signals from the previous stage

Sequences obtained through Nanopore sequencing are long and of high quality which is essential for further research. A nanopore is a hole of nanometer size set in an electrically-resistant polymer membrane through which DNA strands are being driven by electrophoresis. As the strand is being driven through the nanopore, different magnitudes of the electric current density across a nanopore surface are generated depending on the nanopore's dimensions and the composition of DNA or RNA that is occupying the nanopore. These current signals are then converted to a sequence of bases through a procedure called base calling. Although third-generation sequencing approaches yield high-quality data, there are still errors in sequences due to DNA strands moving fast making the signals prone to background noise and base calling errors.

For the fast microbe detection problem, both current signals and base called data could be used. However, this work focuses on the ability to learn representations and distinctions between different microbic organisms based on base called data. Working with such data remains one of the most challenging problems in artificial intelligence due to high complexity of biological data. Additionally, since sequencing errors add noise to the data, it is obvious why this problem can be considered extremely hard.

The goal of this work is to research possibilities of deep learning methods for creating a compressed vector representation for a sequence representing a DNA fragment of a microbe. This representation should encapsulate features that distinguish fragments of one microbe from fragments belonging to other species. Finally, having the representations that in a manner cluster fragments belonging to the same microbe, a system for detecting the microbe based on a representation of a fragment of that microbe is developed.

The approach to the microbe detection problem presented in this work will be inspired by novel deep learning architectures which are mostly used for natural language processing tasks (NLP). These architectures are based on an "attention" mechanism and achieve state-of-the-art results on a variety of NLP problems.

In this thesis, chapter 2 gives an introduction to the dataset used for training and evaluation of the microbe detection process. Chapter 2 also provides details on preprocessing

flow. Chapter 3 will give a theoretical overview of general methods and underlying details of architecture used in this work. Then, specificities on the exact models used for microbe detection task are presented alongside visualization and evaluation methods that will be used. Chapter 3 is completed with a brief description of the technical stack. Chapter 4 presents all of the experiments done and their results and discusses them. Finally, a conclusion is done in chapter 5.

## 2. Dataset

The entire dataset originated from the Zymo mock community <sup>1</sup>, a publicly available database (Nicholls et al. ((2019  The ZymoBIOMICS Microbial Community Standard is the first commercially available standard for studies in metagenomics and microbiomics. Mock community standards can be useful for the development and validation of not only laboratory but also bioinformatics methods as well. This database consists of ten microorganisms, eight equally distributed bacteria (each holding 12% of the data): *Bacillus subtilis*, *Enterococcus faecalis*, *Escherichia coli*, *Lactobacillus fermentum*, *Listeria monocytogenes*, *Pseudomonas aeruginosa*, *Salmonella enterica*, and *Staphylococcus aureus* and two yeasts (each holding 2% of the data): *Cryptococcus neoformans* and *Saccharomyces cerevisiae*.

The entire dataset consists of:

- 4.23 M reads
- 16.59 Gb bases
- 4,620bp read length N50

Besides short reads representing DNA fragments, as a means to perform the experiments in this research, the entire genomes were needed. In our research, data from seven out of ten microbes included in the Zymo mock community was used. Full lengths of the references used in this research are shown in Table 2.1.

**Table 2.1:** Original Genome Lengths

Species	Length (bp)
<i>Bacillus subtilis</i> (bs)	4,045,677
<i>Enterococcus faecalis</i> (ef)	2,845,392
<i>Escherichia coli</i> (ec)	4,875,441
<i>Listeria monocytogenes</i> (lm)	2,992,342
<i>Pseudomonas aeruginosa</i> (pa)	6,792,330
<i>Salmonella enterica</i> (se)	4,809,318
<i>Staphylococcus aureus</i> (sa)	2,730,326

<sup>1</sup><https://github.com/LomanLab/mockcommunity>

Genomes are very long, complex, and very similar to each other which makes the distinction between different species even harder. The closer organisms are in a phylogenetic tree, the higher is the similarity between genomes. The similarity between two genomes can be calculated using the Average Nucleotide Identity (ANI). ANI is a measure of nucleotide-level genomic similarity between the coding regions of two genomes. First, bidirectional best hits (BBHs) between a genome pair are computed as pairwise bidirectional best nSimScan hits of genes having 70% or more identity and at least 70% coverage of the shorter gene. ANI is then computed using the following formula:

$$\text{ANI} = \frac{\sum_{bbh} \text{Percent identity} * \text{Alignment length}}{\text{Lengths of BBH genes}} \quad (2.1)$$

Having an ANI score of  $\geq 95\%$  is, usually, a boundary for same species genomes. The ANI scores for genomes in Table 2.1 are calculated using ORTHOAni<sup>2</sup> (Lee et al. ((2016))) algorithm and are shown in Table 2.2. The notation in the table is following the abbreviations listed in Table 2.1.

**Table 2.2:** ANI scores

ef	67.27%					
ec	64.17%	64.52%				
lm	67.70%	70.47%	65.09%			
pa	64.98%	71.68%	68.02%	68.91%		
se	63.06%	64.62%	80.75%	64.32%	68.33%	
sa	67.21%	67.68%	64.55%	67.33%	72.49%	65.65%
	bs	ef	ec	lm	pa	se

As it can be seen, ANI scores are mostly in the range between 65% and 70% which is relatively high. The biggest outlier is the ANI score between *Salmonella enterica* and *Escherichia coli* which implies it might be the most difficult to distinguish those two species.

## 2.1. File formats

### 2.1.1. FASTA and FASTQ

The most common formats biological data is stored in, are FASTA and FASTQ. Both of these formats contain textual representation of a sequence of nucleotides where each nucleotide is represented by one representation letter. The most common letters are reserved for five canonical nucleobases:

<sup>2</sup><https://www.ezbiocloud.net/tools/orthoani>

- **A**: adenine
- **C**: cytosine
- **G**: guanine
- **T**: thymine
- **U**: uracil

Apart from primary bases, there are letters for parts of the sequence that are noisy and cannot be uniquely determined (eg. M - adenine or guanine, N - any of the five canonical bases, etc.).

Each entry in FASTA format consists of two lines. The first line begins with the symbol ">" followed by a sequence identifier. The second line is the sequence itself.

An example of an entry in FASTA format:

```
>a019fb87-85b7-495a-b6dc-789a2f8c4572_Basecall_2D_000_2d
TACGCATAAGCGCCAAAAGCACAAAGATGCTCACCGCCAG...
```

Each entry in FASTQ format consists of four lines. The first line begins with the symbol "@" followed by a sequence identifier. The second line is the sequence itself. The third line begins with the symbol "+" which is optionally followed by the sequence identifier and the fourth line contains information on the quality of the sequence.

An example of an entry in FASTQ format:

```
@cluster_2:UMI_ATTCCG
TTTCCGGGGCACATAATCTTCAGCCGGGCGC...
+
9C;=;<9@4868>9:67AA<9>65<=>591
```

### 2.1.2. PAF

PAF is a text format describing approximate mapping positions between two sets of sequences. It contains this TAB-delimited information in each line 

## 2.2. Preprocessing

For this research, fragments of DNA are needed. Since sequenced reads are usually noisier and with errors, this work exploits approaches combining both fragments sampled from the references and the real reads. While some of the preprocessing is common for both types of fragments, a few preprocessing steps are separated and will, therefore, be separately described.

**Table 2.3:** PAF format

Column	Type	Description
1	string	Query sequence name
2	int	Query sequence length
3	int	Query start (0-based; BED-like; closed)
4	int	Query end (0-base; BED-like; open)
5	char	Relative strand: "+" or "-"
6	string	Target sequence name
7	int	Target sequence length
8	int	Target start on the original strand (0-based)
9	int	Target end on the original strand (0-based)
10	int	Number of residue matches
11	int	Alignment block length
12	int	Mapping quality (0-255; 255 for missing)

### 2.2.1. Reference-sampled chunks

For each of the reference genomes, an approximately equal number of chunks was sampled. The sampling algorithm went as follows:

1. choose the reference genome from which a chunk will be sampled
2. if the chosen genome consists of more than one contig, choose a contig to sample from (the probability of choosing a certain contig was proportionate to its length)
3. having a contig, randomly choose starting position and sample a chunk of a predefined length (all chunks are of the same length)

The length of the sampled chunks was determined based on the configuration of the representation network, the length of the sequence that is expected as the input to the network to be specific, but more on this later.

Apart from the sampled sequence, some additional information was memorized for each sampled chunk:

1. the starting position of the chunk on the reference
2. contig from which the sample originates
3. microbe to which the sample belongs
4. length of the sample

This information was later used for representation learning.

### 2.2.2. Real reads

The reads obtained in the sequencing process are already in a short DNA fragment format. However, there are some additional steps needed to be done to use these reads in the representation learning in the same manner as reference sampled chunks were used. All of the information that was memorized for the sampled chunks is needed for the real reads as well. While the information on contig and microbe the read originates from is available in the FASTQ file along with the sequence, reads were needed to be aligned against the reference to find to which part of the genome the read belongs to and memorize starting position of the mapping.

This information is obtained through the process called alignment. One of the most commonly used aligner is minimap2<sup>3</sup> (Li ((2018))) which aligns a set of shorter reads to a reference genome and writes the alignment in the aforementioned PAF format. The information stored in PAF format is then used to extract the relative starting position of the read and store it for future purposes.

Now that both reference-sampled dataset and dataset of real reads share the same set of features, both datasets can be preprocessed equally.

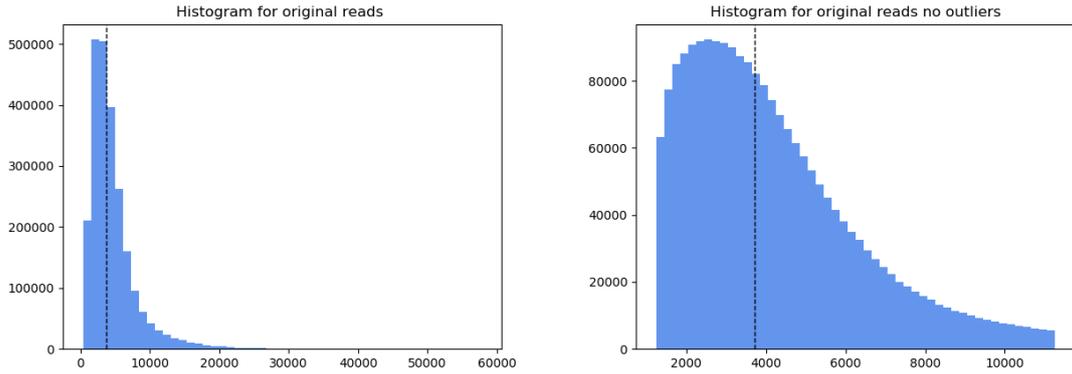
### 2.2.3. Dimensionality reduction

The length of reads obtained through the third-generation sequencing process can vary from less than a thousand to more than ten thousand bases. Long sequences can be problematic for deep models because they usually poorly handle long codependency. The usual length of sequences forwarded to state-of-the-art deep architectures does not exceed few hundreds or a thousand input points which is significantly shorter compared to sequences having few thousand bases. To reduce the dimensionality of sequences, the input samples were split into fragments of  $k$  consecutive bases, so-called  $k$ -mers. Splitting the input samples that way reduced the length of the input samples by the factor of  $k$  which means the original sample containing  $s * k$  bases would now have the maximum acceptable sequence length given that the maximum sequence length is set to  $s$  and the original sequence is split into fragments of  $k$  consecutive bases. To understand this result better, a parallel to natural language processing can be drawn. The sequence obtained through dimensionality reduction is equivalent to a sentence while each  $k$ -mer in the sequence is equivalent to a word in the sentence and each word represents a point in a sentence.

To justify slicing the original sequences into smaller fragments which would then represent one input point in a new sequence, some analysis of lengths occurring in the Zymo mock community database was done.

---

<sup>3</sup><https://github.com/lh3/minimap2>



(a) Histogram of all lengths

(b) Histogram of lengths with 10% outliers removed

**Figure 2.1:** Histograms of read lengths

Figure 2.1 shows histograms of lengths of reads present in the Zymo mock community database. In subfigure 2.1a all lengths all length can be seen and in subfigure 2.1b, 10% of outliers are removed. Additionally, the median of lengths = 3732 is labeled.

Some additional analysis regarding the lengths of the reads was to determine a feasible combination of  $k$  and  $s$ . The analysis is done on a grid of options for a set of values and results are shown in Table 2.4.

**Table 2.4:** Percentage of reads having length  $\leq k*s$

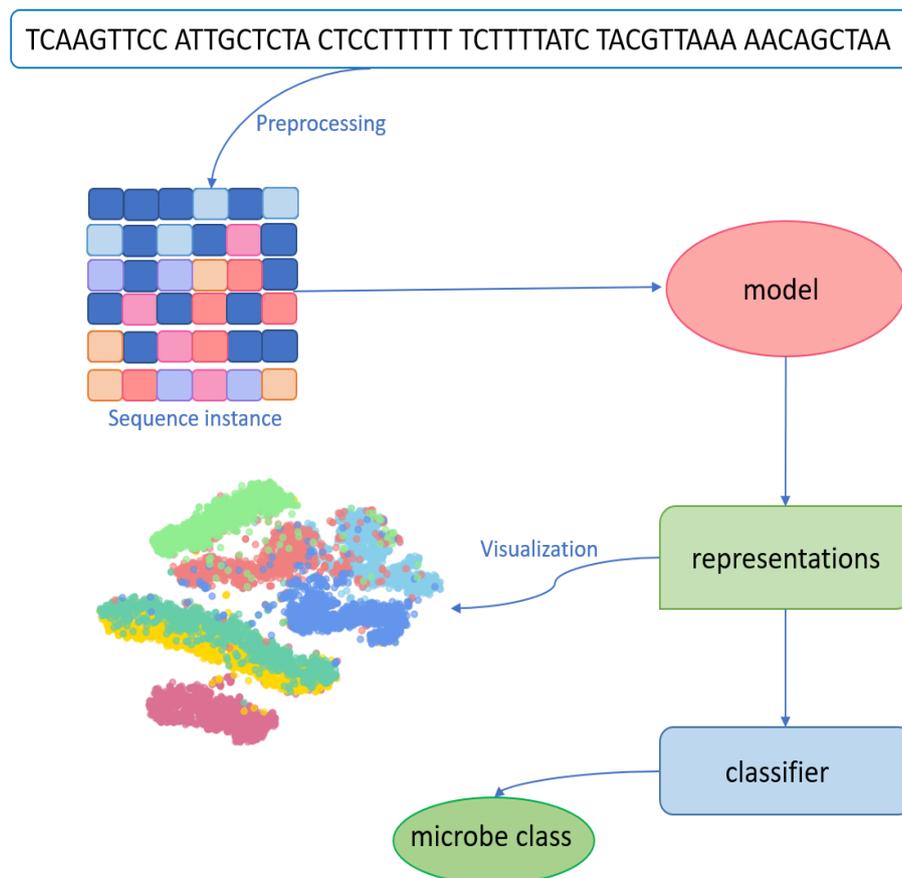
$s \backslash k$	128	256	512	1024
5	0.69%	5.42%	28.09%	<b>69.95%</b>
7	2.17%	13.53%	47.40%	<b>85.49%</b>
9	4.04%	23.13%	<b>63.58%</b>	<b>91.86%</b>
11	7.13%	33.05%	<b>75.18%</b>	<b>95.00%</b>

Table 2.4 shows that the choice of  $k$  and  $s$  can have a significant effect on the portion of reads that "fit" into the network. If the threshold is set to 50%, only six combinations (bold in the table) pass that threshold. Four of those combinations include having sequence length set to 1024 input points which is quite long. The other two feasible combinations combine sequence length of 512 and  $k$  set to either 9 or 11. Although setting  $k=11$  covers a larger portion of reads, setting  $k$  to large value results in having a vast vocabulary of  $k$ -mers. More on vocabulary, its purpose, and the effect of the value of  $k$  on the size of vocabulary later. Nevertheless, for all these reasons, most of the experiments presented in this work will be performed with  $k=9$  and  $s=512$ .

# 3. Methods

## 3.1. Overview

The main objective of this work is to research, develop and evaluate deep learning architecture that could generate a compressed representation of sequenced data. The representation should be such that representations belonging to different microbes could be segregated and the species a read originates from, could be assigned to that read based on its compressed representation with high accuracy. The overview of the work pipeline is shown in Figure 3.1.



**Figure 3.1:** Scheme of work pipeline. The initial sequence goes through the afore-explained pre-processing. The model learns compressed sequence representation, which is visualized and classified into microbial species.

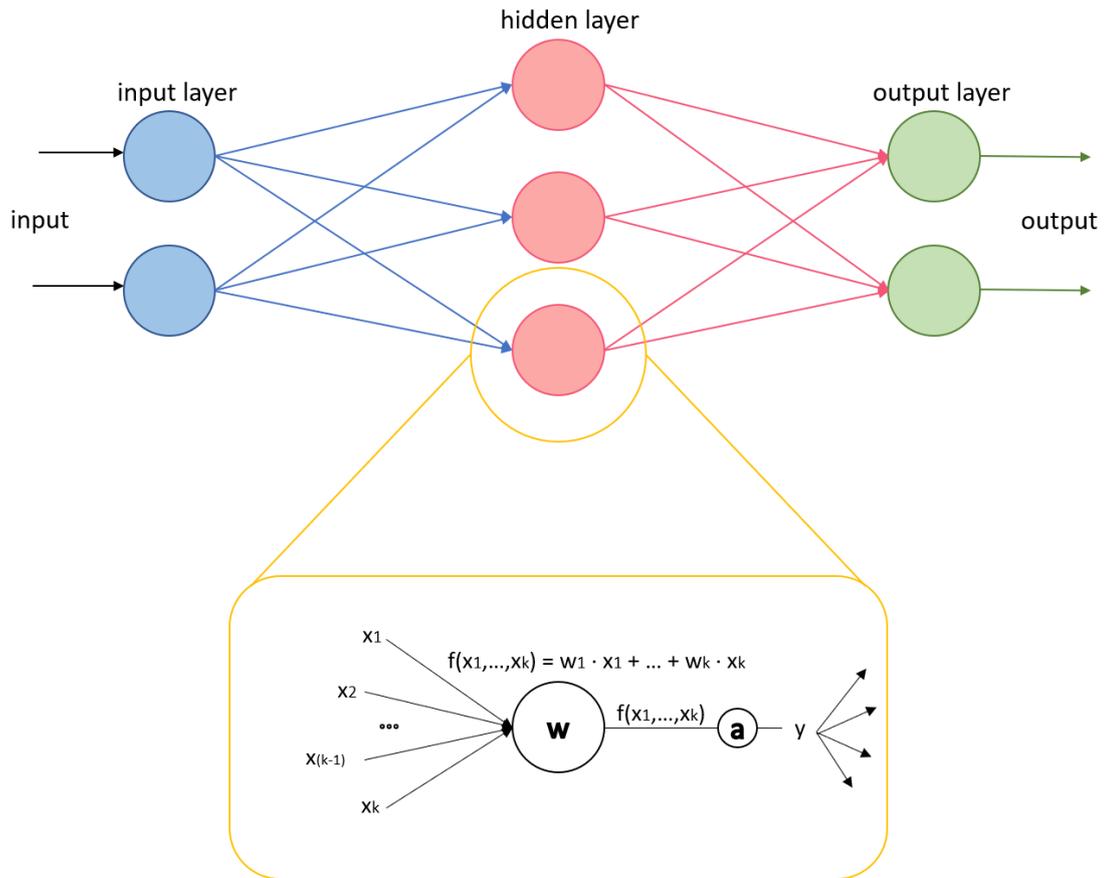
The overview of artificial neural networks is introduced in section 3.1.1. After the pre-processing steps described in section 2.2, textual representations of the k-mers need to be converted to number representations. The process of transferring textual representations of k-mers to real-valued vectors, as well as justification for such transformation, is described in section 3.1.2. Having real-valued representations, the sequences are ready to be forwarded to neural network input. Architecture components of deep neural networks being used for microbe detection problem are explained in detail in sections 3.1.3 and 3.1.4. How these architecture components are combined into a representation network is explained in section 3.2 while the process of detecting microbial species from read-level representations is explained in section 3.3. Finally, visualization and evaluation methods, as well as technical stack details are introduced in sections 3.4, 3.5, and 3.6.

### 3.1.1. Artificial Neural Networks

Artificial neural networks (ANNs) are one of the fundamental pieces of machine learning. They are inspired by biology and designed to simulate the way the human brain processes and analyzes information. The architecture of ANNs can be decomposed into layers. There are three fundamental types of layers present in (each) neural network: an input layer, hidden layers (one or more), and an output layer. Each layer is built of units called neurons. A neuron is a node in the network simulating a neuron in the human brain. It receives one or more inputs and sums them to produce an output. Usually, each input is separately weighted and the output sum is passed through a non-linear function known as the activation function. Many non-linear functions could be used as the activation function but the few most common are: sigmoid, ReLU, tanh, and several variants of those (Sharma and Sharma ((2017))). The connections between nodes are called edges and they imply the output of which node is forwarded to an input of which other node. The edges simulate synapses in the brain.

An overview of a three-layer neural network with a two-dimensional input layer, three-dimensional single hidden layer, and a two-dimensional output layer is illustrated in Figure 3.2. Figure 3.2 also demonstrates the functionality of a neuron with a  $k$ -dimensional input. As the Figure shows, a neuron has a weight vector  $w = (w_1, w_2, \dots, w_k)$  which is used to find a weighted sum of  $k$  input values. The weighted sum is then passed through an activation function and the final output is forwarded to other neurons as one of their input values.

ANNs can learn non-linear functions which is why they are called universal functional approximators. The process of learning consists of two alternating steps called the forward pass and backward pass - backpropagation. Artificial neural networks are built for a specific task and the forward pass is the process of obtaining a result that can be interpreted and evaluated for the predefined task from the inputs by forwarding them through the network. Backpropagation is a general optimization method for performing automatic differentiation



**Figure 3.2:** Scheme of an ANN with a closer look at a neuron structure

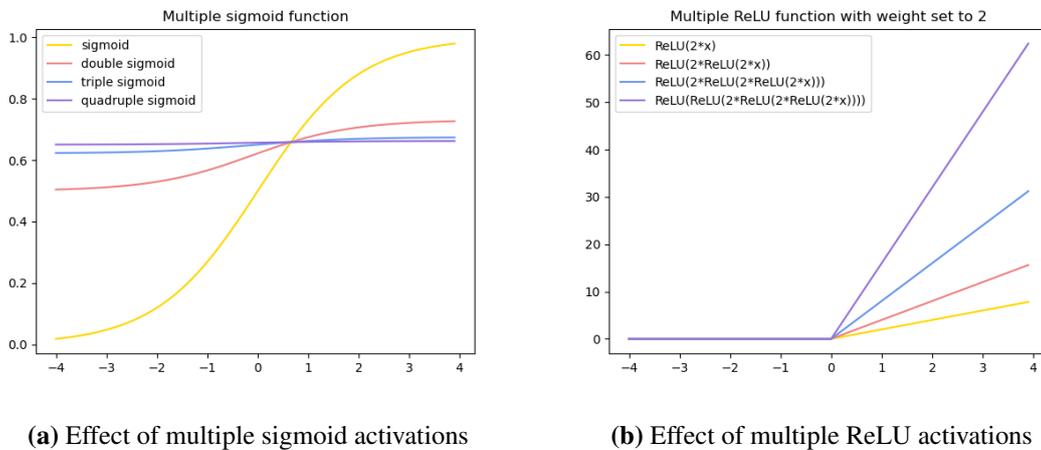
of complex nested functions. The result of the forward pass is evaluated by calculating a predetermined loss function. By computing the gradient of that loss function with respect to the neural network's weights values for weight updating are determined. The gradient is calculated using the chain rule, calculating layer by layer where the gradient of the final layer is being calculated first and the calculation proceeds backward through the network until the gradient of the first layer is calculated.

Deep neural networks (those with more than one hidden layer) often encounter problems with the backpropagation optimization method. These problems are vanishing and exploding gradients. When applying the chain rule in the backpropagation algorithm, the gradients of layers closer to output are passed to gradient calculations of the layers preceding them in the network.

Sometimes, the gradient can have a rather small value - it is common for many activation functions to have a gradient  $\in (0, 1)$ . Calculating the gradient with respect to front layer weights in an  $N$ -layer network effectively results in multiplying small numbers  $N$  times. This results in the front layer gradient being a very small value (almost 0) which is the so-called vanishing gradient problem.

On the other hand, if the activation function gradient is relatively big, it can accumulate

to enormous value through the backpropagation algorithm. This is the so-called exploding gradient problem and it brings the network into an unstable state.



**Figure 3.3:** Vanishing and exploding gradient

Figure 3.3 illustrates problems of vanishing and exploding gradient. Subfigure 3.3a shows how applying the sigmoid function multiple times results in having a flattened function, which is equivalent to a vanishing gradient. On the other hand, applying a ReLU function with weight=2 multiple times results in a very steep function with a possible exploding gradient.

### 3.1.2. Embedding Layer

Textual representation of a k-mer needs to be converted to a real-valued vector which encodes the meaning of that k-mer. This step is important since deep networks expect the input to be in a number format but also, having a real-valued representation of k-mers allows making a comparison between different k-mers, to calculate their proximity, and to create embeddings such that similar k-mers have a similar vector representation.

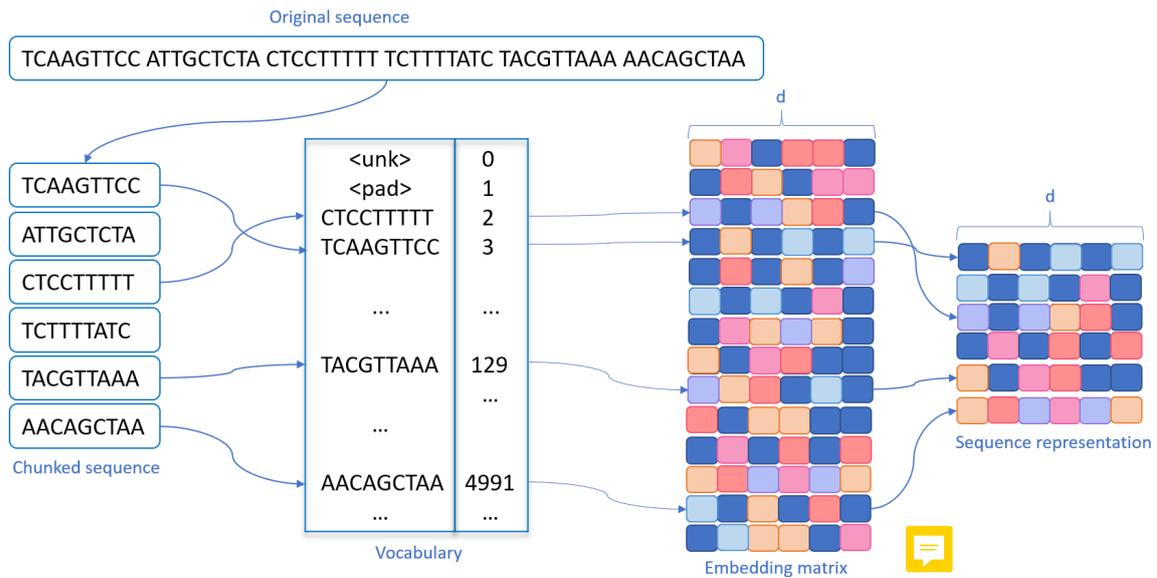
The first step in obtaining real-valued vector representations of the k-mers is building a vocabulary of the k-mers. A vocabulary is a structure in which each k-mer is assigned an index. This index is related to another structure called the embedding matrix. The embedding matrix contains the actual real-valued vector representations of the k-mers. Therefore, when a k-mer is forwarded as the input, the index of that k-mer is extracted from the vocabulary and its embedding vector is found in the embedding matrix based on its vocabulary index. The quality of the k-mer embeddings and further analysis depend on two embedding-related hyperparameters, the size of the vocabulary and the dimension of the representation vector. Assuming four canonical bases are occurring in the samples (A, C, T, G) the maximum size of vocabulary is approximately  $4^k$  since there are four potential values at each position in

the  $k$ -mer and there are  $k$  of those positions. For  $k=9$ , the size of vocabulary would be 262,144 which is acceptable. On the contrary, if  $k$  would be set to 10, the vocabulary would contain 1,048,576 inputs which is significantly bigger and more memory demanding. This difference reflects even more when observing the effect it has on the embedding matrix. The dimensions of the embedding matrix are defined by the size of the vocabulary and the dimension of the representation vector. Having the size of vocabulary  $V$  and the dimension of the representation vector  $d$ , the shape of the embedding matrix is  $V \times d$ . Having  $d$  set to a value that is too small lacks the capacity while setting  $d$  to a value that is too big results in representations having poor quality.

Since data obtained through the sequencing process has some errors which are encoded with letters other than A, C, T, G, if a  $k$ -mer containing one of those letters comes as input, the current vocabulary-embedding matrix system does not have a way to handle such  $k$ -mer. For these and some other reasons, the usual approach is to have some special tokens in the vocabulary and the representation vectors associated with those tokens. Two most common special tokens are `<unk>` and `<pad>` token. The `<unk>` token is used when a  $k$ -mer not present in the vocabulary is presented as the input. That  $k$ -mer is then replaced by the `<unk>` token and represented through a vector representation of the `<unk>` token.

As Figure 2.1 shows, read lengths can vary from a few hundred to few tens of thousands of bases. When creating a real-valued vector representation for a sequence of  $k$ -mers, each  $k$ -mer is embedded with a vector of a predefined length. While each  $k$ -mer is now represented with a vector of the same dimensionality, the dimensionality of the sequence remains different. Artificial neural networks usually expect the input to have invariable dimensions and this constraint does not hold for reads with varying lengths. This problem is solved by defining the expected dimensionality of the input when constructing the neural network and doing some additional preprocessing for those reads that have a length different from the predefined expected sequence length. While several different approaches can be taken, the two most common are trimming and padding. Trimming refers to reads that are longer than the predefined maximum sequence length. Trimming means, as the name itself implies, that reads that are too long for the model should be trimmed to predefined length and the residue of the sequence should be discarded. Padding refers to reads that are shorter than predefined length. This approach exploits the aforementioned `<pad>` token. When the input sequence is shorter than the predefined length, it is artificially augmented to that length by adding `<pad>` token at the end until the target length is achieved. `<pad>` token usually consists of zeros which is why this approach is often referred to as zero-padding.

The embedding pipeline is shown in Figure 3.4. Chunking the initial sequence into  $k$ -mers is followed by finding their index in the vocabulary and extracting their representation vector from the embedding matrix based on the vocabulary index. The  $k$ -mer embedding



**Figure 3.4:** Dimensionality reduction and embedding process

vectors are then stacked to form an input sequence that is forwarded to the network. It should be mentioned that representation vectors in the embedding matrix are initially set to random values but are being optimized through backpropagation, like any other parameters in the network.

### 3.1.3. Attention

For many years, the dominant strategy in sequence modeling included two types of neural architectures, convolutional and recurrent neural networks. These architectures often encountered problems with the aforementioned vanishing and exploding gradients due to long-term temporal dependencies. These problems imposed some length-related and other constraints to recurrent and convolutional models for sequence modeling which resulted in poor results across different sequence modeling tasks.

The biggest change in sequence modeling came in 2017 with "Attention Is All You Need" (Vaswani et al. ((2017))) paper where authors proposed a novel network architecture, the Transformer, based on a mechanism called attention where they focused on a variant of attention mechanism called self-attention.

The attention mechanism consists of three basic concepts: query, keys, and values, all real-valued vectors. The attention function maps a query and a set of key-value pairs to an output vector. The output vector is computed as a weighted sum of values where each weight in the sum is calculated as the compatibility score between the query and the corresponding key. In the self-attention setting all three vectors, query, key, and value, are created from the same original vector which corresponds to the aforementioned k-mer embedding (for

natural language processing words are being embedded to real-valued vectors). Query-, key- and value-vectors are created by multiplying embeddings by three separated matrices:  $W^q$  for queries,  $W^k$  for keys, and  $W^v$  for values. These matrices usually have a smaller target dimension (compared to embedding dimension) therefore creating query, key, and value vectors that are smaller than corresponding original embeddings and the weights of matrices are being optimized in the learning process (through backpropagation). Having query, key, and value vectors determined from the embeddings, it can be proceeded with the attention mechanism. There are a few variants of attention functions, such as Bahdanau (additive) (Bahdanau et al. ((2016))), or dot-product attention (Luong et al. ((2015))), but the most commonly used nowadays (and presented in the "Attention Is All You Need" paper) is so-called scaled dot-product attention.

The input consists of three aforementioned types of vectors, queries and keys of dimension  $d_k$  and values of dimension  $d_v$  which means the dimensions of the projection matrices  $W^q$ ,  $W^k$  and  $W^v$  are  $d_{model} \times d_k$ ,  $d_{model} \times d_k$ , and  $d_{model} \times d_v$ , in that order, where  $d_{model}$  is the target dimension. Attention function is in practice computed on a set of queries simultaneously. The queries are packed into a matrix  $Q$  as well as keys and values into matrices  $K$  and  $V$ . The matrix of outputs for input queries is computed as:

$$Attention(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (3.1)$$

As equation 3.1 shows, scaled dot-product attention consists of computing the dot-product of the query with all keys, dividing it by  $\sqrt{d_k}$ , and applying softmax to obtain the compatibility scores between queries and keys which are then used as weights for calculating the output. Applying softmax to the dot-product output essentially means keeping values that should be focused on intact, and drowning out those values that are irrelevant. Dividing the dot-product by  $\sqrt{d_k}$  is an important step, especially for large  $d_k$ .  $\sqrt{d_k}$  corresponds to the standard deviation of that dot-product, therefore, dividing the result by  $\sqrt{d_k}$  essentially means normalizing the dot-product result. For values relatively far from zero, softmax function is in regions with extremely small gradients (this is also visible in Figure 3.3a) and by normalizing the dot product, value is pushed into a region close to zero to counteract that effect.

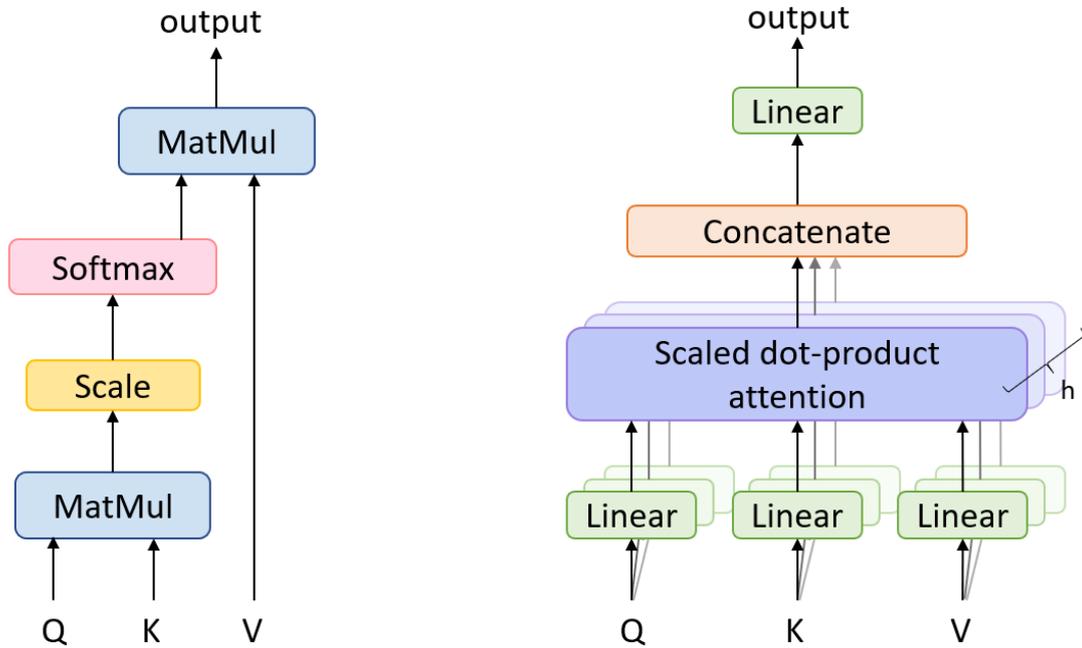
The attention mechanism is based on calculating compatibility score and consequentially focusing on certain position(s). The attention performance can be improved by a simple extension to the above-explained attention mechanism - multi-head attention. Multi-head attention is essentially the same as the self-attention mechanism explained above only repeated  $h$  times with different  $W^q$ ,  $W^k$ ,  $W^v$  matrices. This extension to the original self-attention mechanism expands the model's ability to focus on different positions. This is important since, in a single self-attention setting, weights could easily be dominated by the actual k-

mer itself and not other k-mers in the sequence. Additionally, since each attention head has independent query/key/value matrices by which the initial embeddings are projected into query/key/value representations, the attention layer is augmented with multiple representation subspaces. The final output of multi-head attention is calculated as follows:

$$MultiHead(Q, K, V) = \text{concatenate}(\text{head}_1, \text{head}_2, \dots, \text{head}_h)W^o \quad (3.2)$$

$$\text{where head}_i = \text{Attention}(QW_i^q, KW_i^k, VW_i^v) \quad (3.3)$$

The dimensions of  $W_i^q, W_i^k, W_i^v$  remain the same as in single-head attention and  $W^o \in \mathbb{R}^{hd_v \times d_{model}}$ .



**Figure 3.5:** Overview of scaled dot-product (left) and multi-head attention mechanisms (right)

The overview of the scaled dot-product attention mechanism (left) and multi-head attention (right) is illustrated in Figure 3.5.

Knowing the basic mechanisms behind the attention, more about a deep learning architecture called transformers which is based on attention mechanism can now be learned.

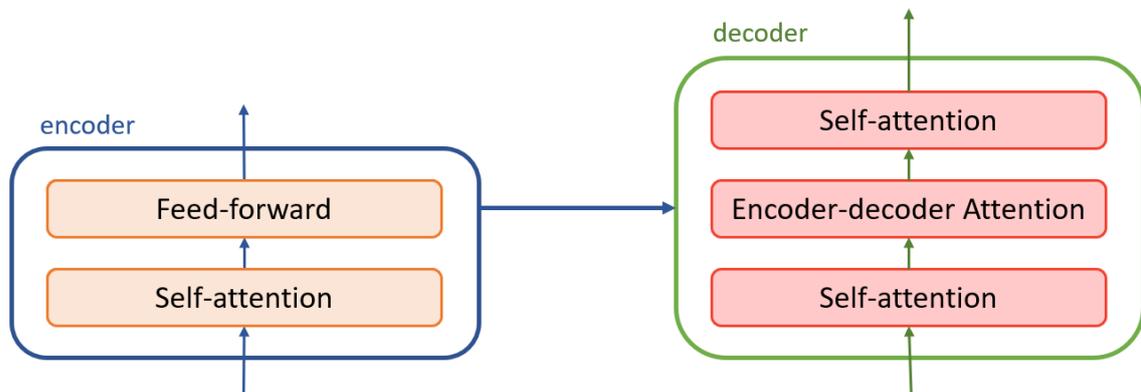
### 3.1.4. Transformers

The transformer network was originally designed for sequence transduction problems and therefore consists of an encoding and a decoding component. The input to the encoder would

be the original sequence and the output of the encoder would be forwarded to the decoder component which would use knowledge extracted from the input sequence in the encoder to compute the output sequence. This section provides a detailed overview of the transformer architecture and how this architecture incorporates attention mechanism and performs different sequence modeling tasks.

The encoding component of the transformer network is essentially a stack of encoder layers that are identical in structure. An encoder layer consists of two components: a self-attention layer followed by a feed-forward network. When the self-attention layer outputs the weighted value vectors as it is explained above, the same feed-forward network is applied to each position in the sequence independently and the final output of the encoder layer is created. The output of the preceding encoder layer is then forwarded to the input of the following encoder layer and the initial input for the first encoder layer consists of the initial embedding vectors.

The decoder component has a structure similar to the encoder component. It is essentially a stack of decoder layers (the same number as the encoder) that have an architecture similar to encoder layers but with an additional component. Three components of the decoder layer are the self-attention layer, encoder-decoder attention layer, and feed-forward neural network. The encoder-decoder attention layer helps the decoder to focus on relevant parts of the input sentence by performing multi-head attention over the output of the encoder. The other two components of the decoder layer have the same role as they do in the encoder component.



**Figure 3.6:** Illustration of the encoder and decoder layer

The architecture of the encoder and decoder layer is illustrated in Figure 3.6.

The transformer network does not include any recurrent nor convolutional components. All of the inputs are simultaneously forwarded to the network. That approach provides no information on the order of inputs in the sequence. This information is really important for sequential data, especially biological data, since the ordering of the nucleotides (or k-mers)

defines the functionality of the genome fragment. To solve this problem, information on the relative ordering in the sequence is injected through a mechanism called positional encoding.

### Positional encoding

Positional encoding essentially consists of adding a vector to each of the input embedding vectors to enrich the embeddings with the information on the relative position of the token in sequence. Positional encoding vectors follow a certain pattern that can be fixed or learned by the model. Commonly used positional encoding approach includes forming the positional encoding vector from two periodic functions, sin, and cos:

$$PE(pos, 2i) = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right) \quad (3.4)$$

$$PE(pos, 2i + 1) = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right) \quad (3.5)$$

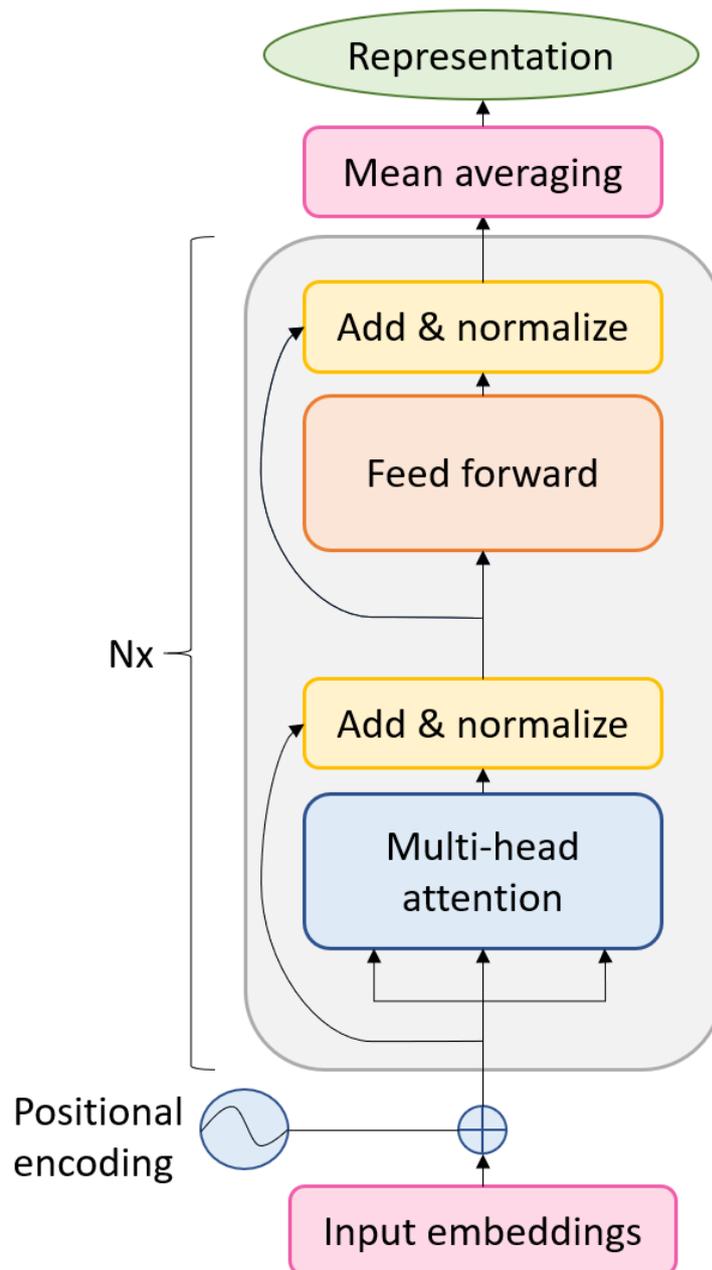
where  $d_{model}$  is the dimension of the embeddings and  $i$  is the position.

Knowing all of the specificities of the transformer network, the overall architecture of the network can be further inspected. As it was mentioned earlier, the transformer network is originally built as a two-part network with an encoder for knowledge extraction and a decoder for output sequence creation. However, the microbe detection task is not a sequence-to-sequence problem and there is no need (nor the possibility) for creating an output sequence. The transformer network built for this task should only extract knowledge from the input sequence and create a sequence representation. This is done by using only the encoding component of the transformer network.

Transformer-based model with the architecture for the microbe detection task is illustrated in Figure 3.7. This model has the underlying architecture of the transformer model described above. A sequence of the aforementioned k-mer embeddings is presented as the input, information on the relative positions of k-mers in the sequence is injected by positional encoding and position-corrected embedding vectors are then forwarded to the transformer network. The transformer component is built out of N encoder layers. Each encoder layer has the two-component structure explained above. The figure, however, shows the presence of a residual connection around each sublayer in the encoder layer followed by layer normalization. The output of each sublayer in the encoder layer can be formulated as:

$$\text{SublayerOutput}(x) = \text{LayerNorm}(x + \text{SublayerFunc}(x)) \quad (3.6)$$

The output of the final encoder layer is essentially a list of vectors where the length of the list is determined by the length of the input sequence and each vector represents an updated real-valued vector representation of each k-mer in the input sequence. These vectors are often referred to as dynamic embeddings (while the initial embedding is referred to as static



**Figure 3.7:** Transformer model for sequence representation

embeddings). However, this does not complete the representation task, since the final goal is to have a single vector that would be the representation of the entire sequence. There are several approaches for obtaining the sequence-level representation from a list of token-level representations. One approach would be to concatenate all of the token-level vector representations and create a sequence-level representation. This approach has one significant disadvantage and that is the high dimensionality of the sequence representation. The next approach was used for sequence classification tasks for a long time. The idea was to introduce an additional special token and add it at the beginning of the input sequence. This token would be initialized from the embedding matrix like the other tokens and it would participate

in the encoding process. Once the output of the encoder is obtained, the vector representation corresponding to the special token would be used as a sequence representation and the rest of the output would be discarded. The main disadvantage of this approach was that it solely relied on artificially added token and the rest of the output was simply discarded. Therefore, a rather simple yet efficient approach to creation of sequence representation from token-level embeddings would be to mean average all of the token-level embeddings into a single vector of the same dimensionality. That vector would then be the sequence representation which is the chosen approach for this work and is illustrated in Figure 3.7.

By performing mean averaging of encoder output, the final sequence representation is obtained. How is the goodness of a sequence representation evaluated, and how is such network trained will be explained in detail in the following section.

## 3.2. Representation Models

As was already mentioned, the main goal of this work is to develop a mechanism that creates compressed representations of sequenced data such that reads originating from the same species are more similar than those of reads originating from different species. An unsupervised approach to this problem is the triplet concept which will be explained in the following subsection.

### 3.2.1. Triplet Network

A triplet network (Hoffer and Ailon ((2018))) is a neural network of an arbitrary architecture (convolutional, recurrent, transformer, ...). The defining feature of a triplet network is the training mechanism. The triplet network training is an unsupervised feature learning approach utilizing triplets  $(x, x^+, x^-)$  such that:

- $x$  is an arbitrary sample called anchor
- $x^+$  is a positive sample semantically similar to anchor
- $x^-$  is a negative sample semantically dissimilar to anchor

Triplets are forwarded through the network and their representations are obtained. Having their representations, the network is trained by optimizing so-called triplet loss defined as:

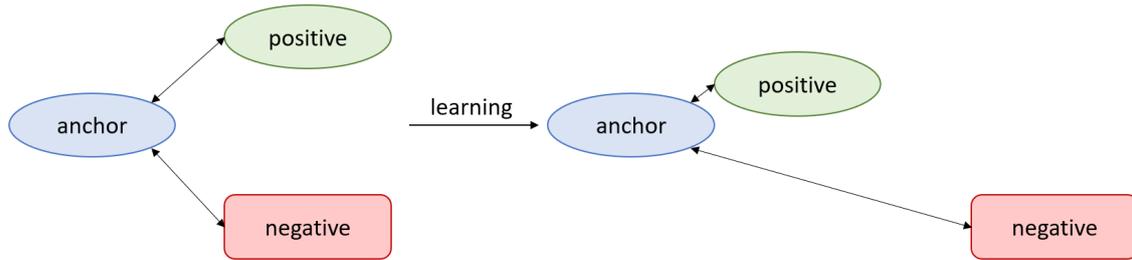
$$\mathcal{L} = \max(d(f(x), f(x^+)) - d(f(x), f(x^-)) + margin, 0) \quad (3.7)$$

The formula in equation 3.7 shows how the triplet loss is calculated for anchor  $x$ , positive sample  $x^+$ , and negative sample  $x^-$ . First, they are forwarded through the network to obtain their representations:  $f(x)$ ,  $f(x^+)$ , and  $f(x^-)$ . Having the representations, distances

$d$  between the anchor and positive sample, and anchor and negative sample are calculated. The distance between representations can be defined as any distance measure suitable for real-valued vectors but most commonly used is the L-norm distance measure. There are many L-norm distances but one of the most commonly used is L2-norm distance - Euclidean distance which is, for n-dimensional data, calculated as equation 3.8 states and was used for this purpose.

$$d(a, b) = \sqrt{\sum_{i=0}^n (a_i - b_i)^2} \quad (3.8)$$

To have similar representations for what are considered to be similar reads - anchor and positive sample, and dissimilar representations for what are considered to be dissimilar reads - anchor and negative sample, the distance between the anchor and positive sample representation should be as close to zero as possible and the distance between the anchor and negative sample should be as large as possible. If the expression  $d(f(x), f(x^+))$  has a value close to zero and expression  $d(f(x), f(x^-))$  has a large value, then the difference between those two expressions is negative. However, since loss cannot be negative and a positive learning outcome usually means the loss is equal to 0, the maximum value between the difference of distances and 0 is taken. There is an additional element in the triplet loss formulation called *margin*. Margin is a positive real-valued number to push the difference between anchor-positive and anchor-negative distances to larger values. This addition pushes the original distance constraint,  $d(f(x), f(x^+)) \leq d(f(x), f(x^-))$ , to  $d(f(x), f(x^+)) \leq d(f(x), f(x^-)) - margin$  where  $margin > 0$  in order to have  $\mathcal{L} = 0$ .



**Figure 3.8:** Illustration of triplet learning effect

The effect of triplet learning on distances between similar and dissimilar samples is shown in Figure 3.8.

Choice of a positive and a negative sample for the sampled anchor depends on the problem that is being modelled through the triplet network. For this work, potential positive samples for a sampled anchor are those reads that belong to the same microbe as the anchor. Since some additional information on reads was extracted, the list of potential positive samples can be narrowed down to reads that share more than species with the anchor. As section

2.2 explains, alongside the sequence itself, the specific contig the read originates from, and the relative starting position on the reference are memorized for each read. Therefore, a read is a candidate for being a positive sample for a chosen anchor if it originates from the same contig and its starting position on the reference is relatively close to the starting position of the anchor. Additional information for narrowing the list of negative candidates can be utilized as well. A read is a candidate for the negative sample if it does not originate from a different species does not belong to a similar region of the microbe which is determined by taking the relative starting position on the reference into account for these purposes as well.

### **3.3. Detection from representations**

After creating compressed representations from the input sequences such that reads from the same microbe and belonging to the same region of the reference have similar representations and those belonging to different species have rather dissimilar representations, a mechanism of deciding from which microbe the read originates based on its representation has to be developed. Two approaches can be taken for the final microbe detection, one being a parametric classification approach and the other being a non-parametric classification approach such as k-nearest neighbors. Details on these approaches, their advantages, disadvantages, and main differences, but also the possibility of cooperation between these two approaches are presented in the following subsections.

#### **3.3.1. Parametric classification**

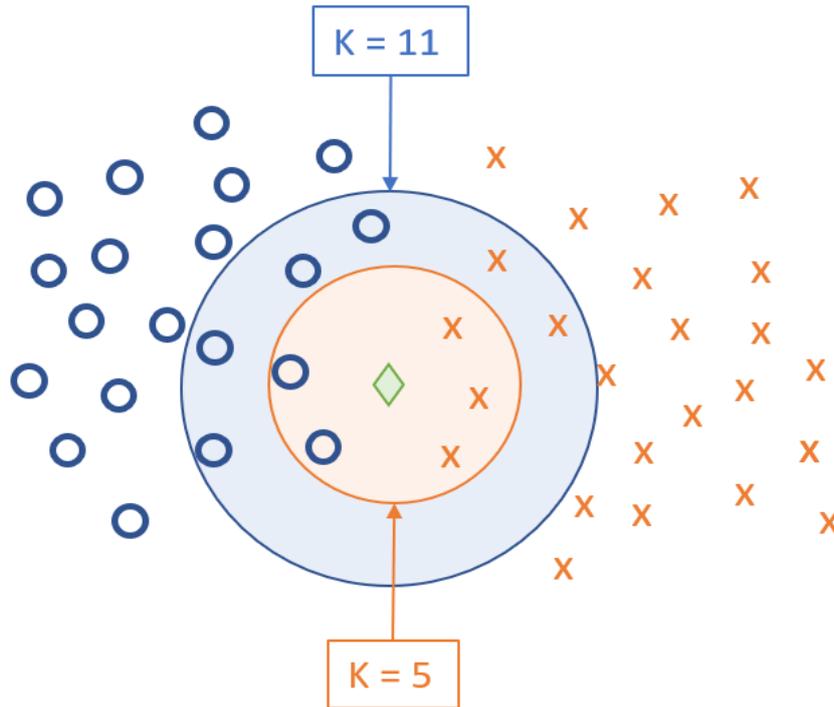
A parametric classification is an approach that requires some architecture extensions for the afore-explained representation network. After obtaining a representation of the read, the knowledge embedded in that representation is used to determine the microbe that read belongs to. This is done by adding one or more artificial neural layers on top of the representation network. These layers are usually referred to as classification head. The input to the classification head is the representation vector and the outputs are probabilities of a sample belonging to each of the predefined classes. Since this approach includes adding some parameters to the network, some additional training is required for the model to be able to accurately classify samples. The additional training can include only those layers that belong to the classification head or the entire network. If the weights of the representation component remain fixed and only the classification head is being trained, the learning algorithm is by using the entire network for the forward pass but backpropagate the gradient only through the layers of the classification head. On the other hand, both the pretrained representation component as well as the newly initialized classification head can be trained

together on the classification task. This approach is often referred to as fine-tuning and it is a common practice in deep learning. Fine-tuning implies that the entire network participates in both forward and backward pass of the learning algorithm. Training a classification network requires labels for provided samples. When it comes to microbe detection, each sample should be assigned a label denoting to which microbe the sample belongs. Since the output of the classification head denotes the probabilities of the input sample belonging to a certain microbe, the number of possible microbe classes has to be defined before classification learning. This imposes some constraints. Once the classification network is successfully trained using a set of  $N$  microbial species, it can accurately classify any sample belonging to one of the species that were a part of the training set. However, if a sample originating from a completely new species occurs, this network still classifies that sample as one of the  $N$  microbial species from the training set. To have the ability to classify samples from the new microbial species, a set of samples from that microbe would be needed for training a new network with  $N+1$  target microbes. This is quite unfortunate since the number of target microbes can increase repeatedly. However, it has been shown many times that fine-tuning a pretrained representation network on a downstream task, such as classification, has benefits not only for the final classification accuracy but also improves the representation results.

### 3.3.2. K-nearest neighbors

K-nearest neighbors (KNN) (Kramer ((2013))) is non-parametric classification method. This implies there is no need for any architectural changes or extensions to the existing representation network. Having a set of sample representations alongside labels denoting microbe classes of the samples, the K-nearest neighbors classifier can determine the class of any new sample based on its representation vector. K-nearest neighbors algorithm is a part of a general method known as instance-based where specific training instances are used to make predictions without having an actual model derived from data. Therefore, the training part of the K-nearest neighbors is simply storing training data along with class labels of training samples. Having this information stored, new observation is assigned to the most frequent class amongst K-nearest neighbors of the new observation. The success of K-nearest neighbors classification depends on data points in the training set but also on a few hyperparameters. The hyperparameters that can affect the prediction of the K-nearest neighbors classifier are the number of nearest neighbors being observed -  $K$ , and the distance measure determining which data points are the nearest to the new observation. While the aforementioned L2 norm distance - Euclidean distance is common for continuous data,  $K$  is more of a problem-specific parameter.

The effect of different values for hyperparameter  $K$  is shown in Figure 3.9. Setting  $K$  to 5 results in new observation being classified as 'x' while setting it to 11 results in new



**Figure 3.9:** Effect of different K on classification results

observation being classified as 'o'.

K-nearest neighbors classification method has the obvious advantage of requiring no additional training nor architecture modifications. The classification process is quite straightforward and the information on number of nearest neighbors belonging to each of the possible classes is available. When a sample belonging to a new microbe appears, that can be easily detected by looking at the distances between that sample and samples from known microbes (if the representation network works as expected, this sample should be very distant from samples belonging to different species) and to include a new species in the KNN method is simple and easy. Also, some additional constraints can be easily imposed. These constraints can assure a higher level of certainty for classification. With the simplest KNN approach and  $N$  possible classes a minimum fraction of neighbors belonging to the same class for a sample to be classified as that class is  $\frac{1}{N}$ . Having a large number of classes effectively means that the class of observation is determined by relying on a relatively small fraction of nearest samples. This can be changed by adding a more strict condition for the classification, e.g.  $\geq 50\%$  of neighbors belonging to the same class. If the condition is not met, the new observation can be classified as an outlier.

The main disadvantage of this approach is that it completely relies on a representation network that is usually pretrained on some general task and it might not be fully adapted to the classification task.

For all these reasons, a hybrid approach to classification could utilize some advantages of parametric as well as non-parametric approach. The hybrid approach would include training a classification network consisted of representation network and classification head on a downstream classification task. However, once the training is done, the classification head is removed and the updated representation network is used for the KNN classification approach. This way, a fine-tuned representation network that is further adapted to the classification task is obtained and can be used to classify new observations in a fast and simple manner while retaining the possibility of easily detecting a sample that potentially does not belong to any of the microbes present in the current training dataset. Of course, this approach does not eliminate the need for a new fine-tuning session once a new microbial species is included in the database but it shows advantages over both individual approaches.

### 3.4. Visualization

There are several methods for visualizing high-dimensional data in a low-dimensional space (usually two- or three-dimensional space). T-distributed stochastic neighbor embedding (t-SNE) (van der Maaten and Hinton ((2008))) method is chosen for visualizing representations of reads. t-SNE is a statistical visualization method based on Stochastic Neighbor Embedding. The entire method is built around the idea of modeling each high-dimensional point by a two- or three-dimensional point in such a way that similar objects are modeled by nearby points and dissimilar objects are modeled by distant points with high probability.

t-SNE consists of two stages. In the first part of the algorithm, a probability distribution over pairs of original high-dimensional points is constructed in such a way that similar data points are assigned high probability while dissimilar data points are assigned low probability. This is done by computing two conditional probabilities for each pair of data points following this formula:

$$p_{j|i} = \frac{\exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2/2\sigma_i^2)}{\sum_{k \neq i} \exp(-\|\mathbf{x}_i - \mathbf{x}_k\|^2/2\sigma_i^2)} \quad (3.9)$$

$p_{i|i} = 0$  for each  $i=1, \dots, N$  where  $N$  is the number of data points in the dataset.  $p_{i|j}$  is calculated likewise and probability  $p_{ij}$  that is proportional to similarity between  $\mathbf{x}_i$  and  $\mathbf{x}_j$  is defined as:

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2N} \quad (3.10)$$

Now, since t-SNE aims to learn a d-dimensional map that reflects calculated probabilities  $p_{ij}$ , the second phase of the t-SNE algorithm is constructing a probability distribution over pairs of points in a d-dimensional map in such a way that the Kullback-Leibler divergence

(KL divergence) between these two distributions is minimal. The probability distribution of low-dimensional data is constructed following this approach:

$$q_{ij} = \frac{(1 + \|\mathbf{y}_i - \mathbf{y}_j\|^2)^{-1}}{\sum_k \sum_{l \neq k} (1 + \|\mathbf{y}_k - \mathbf{y}_l\|^2)^{-1}} \quad (3.11)$$

where  $\mathbf{y}_i, i=1, \dots, N$ , are the same data points but in  $d$ -dimensional space.

Once the probability distribution between low-dimensional data pairs is calculated, the difference between these two distributions is calculated using KL divergence:

$$KL(P||Q) = \sum_{i \neq j} p_{ij} \log \left( \frac{p_{ij}}{q_{ij}} \right) \quad (3.12)$$

By minimizing the KL divergence using gradient descent with respect to  $\mathbf{y}_i$  the final distribution of points in low-dimensional space is determined. The quality of t-SNE clustering depends heavily on algorithm parametrization and can be misleading but it is simply used as a visualization aid - other forms of evaluation are calculated using the original representations.

### 3.5. Evaluation Metrics

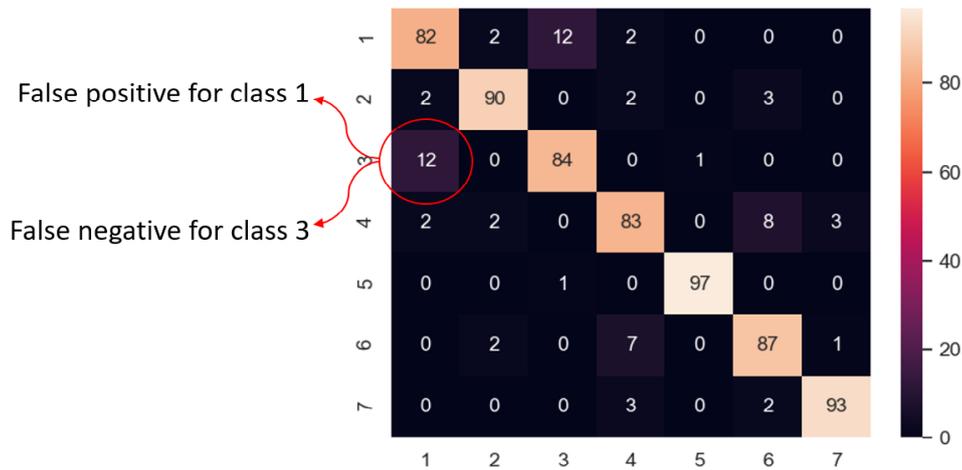
Evaluation metrics provide feedback on how well does a model perform on a target task. There are several standard evaluation metrics for classification problem: accuracy, recall, precision, and F1 score (M and M.N ((2015))). All of the mentioned metrics are closely related to a specific table layout that allows a straightforward visualization of model performance called confusion matrix. A confusion matrix is a square matrix with the number of rows and columns determined by the number of target classes of the classification model. Each row in the confusion matrix represents instances of the actual class while each column represents instances of the predicted class (in some literature it is vice versa). This effectively means that numbers on the diagonal of the matrix represent the number of correctly predicted samples per class, also called true positives. On the other hand, numbers off the diagonal represent the number of incorrectly classified samples, also called false positives and false negatives. E.g., the number in the first row and fifth column of a confusion matrix stands for the number of samples that belong to first class but were classified as fifth class. These samples are considered false negative for class 1 but false positive for class 5.

An example of a confusion matrix is shown in Figure 3.10.

Having a confusion matrix, all of the aforementioned metrics can be easily calculated.

Accuracy is the ratio of correctly classified observations to all observations:

$$accuracy = \frac{\sum_{class} true\ positive_{class}}{\sum confusion\ matrix} \quad (3.13)$$



**Figure 3.10:** Example of a confusion matrix

Recall (sensitivity) is the ratio of correctly classified observations to all observations belonging to the actual class (it is computed for each class independently):

$$recall = \frac{true\ positive}{true\ positive + false\ negative} \quad (3.14)$$

Precision (positive predictive value) is the ratio of correctly classified observations to all observations in the predicted class (it is computed for each class independently):

$$precision = \frac{true\ positive}{true\ positive + false\ positive} \quad (3.15)$$

F1 score is the weighted average, or rather a harmonic mean, of precision and recall (it is computed for each class independently):

$$F1 = 2 * \frac{recall * precision}{recall + precision} \quad (3.16)$$

All of the metrics that are being calculated per class are averaged to obtain so-called macro metrics.

These metrics are very informative even for imbalanced datasets - those with the majority of data belonging to one class. Although our dataset is not like that, all these metrics are important for getting an insight into the model's performance.

## 3.6. Technical Stack

The solution for the microbe detection problem is implemented using the programming language Python. Alongside standard Python libraries, one of the most important computational libraries for this work was Pytorch.

Pytorch is an open-source machine learning framework that includes a two-fold usage - tensor computation with GPU acceleration and a machine learning research platform with maximum flexibility and speed. PyTorch defines a class called Tensor (`torch.Tensor`) to store and operate on homogeneous multidimensional rectangular arrays of numbers. PyTorch Tensors are similar to NumPy arrays, but can also be operated on a CUDA-capable Nvidia GPU. The most important PyTorch modules include the Autograd module which provides an automatic differentiation method, the Optim module with a variety of optimization algorithms, and the nn module with basic building blocks for graphs.

The evaluation of the model relied on Python's sklearn library while the visualization was done using utilities of matplotlib library.

To align a set of reads to a reference, minimap2 was used. minimap2 is a versatile sequence alignment program that aligns DNA or mRNA sequences against a large reference database.

The entire implementation is publicly available at [GitHub](#).

## 4. Experiments and Results

This section provides details on different experiments done on microbe detection problem and the results of these experiments.

For all of the experiments, the basic architecture was afore-explained transformer-based architecture for the representation pretraining but the hyperparameters of the network and dataset setting were different. After the pretraining was done, a classification head was added on top of the representation module, and classification fine-tuning was done.

Some of the hyperparameters were kept constant across different experiments. These hyperparameters are shown in Table 4.1.

**Table 4.1:** Constant parameters

Parameter	Constant
k	9
encoder layers	6
representation dim	192
optimizer	SGD
scheduler	stepLR
scheduler step	2500
gamma	0.99

SGD (Stochastic Gradient Descent) (Ketkar ((2017))) is a basic iterative method for optimizing an objective function. Although there are different advanced optimizing methods, SGD showed as suitable for this task.

StepLR is a learning rate scheduler that decays the learning rate of each parameter group by a predefined value - gamma, every "scheduler steps". After a predefined number of steps, the current learning rate is multiplied by the gamma value to obtain a new learning rate for the next scheduler steps.

Parameter  $k$  refers to the size of k-mers the original sequence is being fragmented into. As the read length analysis in section 2.2.3 showed, setting  $k=9$  seems like a reasonable choice so that parameter was held fixed throughout all experiments.

When deciding on K for the KNN classifier, each model was testing with K ranging from 2 to 15 and one achieving the best results was chosen.

Details on experiment settings and the results will be provided in the following subsections.

## 4.1. Training on reference-sampled chunks

This experiment is done in somewhat idealized conditions. As it was mentioned earlier, real data is noisy and of various lengths. On the other hand, the closest it can get to perfect data is by sampling the genome references. Additionally, when sampling the references, the length of sampled chunks can be predefined and set to a value that perfectly fits the input dimensions of the representation network.

For this experiment, 300 000 samples were sampled from references of seven microbial species.

For this experiment, the maximum sequence length is set to 512 and all of the output vectors are mean averaged to obtain a final 192-dimensional sequence representation. Before returning, the output is batch normalized.

24 epochs of representation learning with triplet loss were done. The margin was set to 2.5 and the loss in the 24th epoch was 0.21 (the initial value of the loss was 2.5 meaning that the distance between an anchor sample and a positive sample was the same as the difference between an anchor sample and a negative sample).

Evaluation of the network was done on a different set of chunks sampled from the same references containing 7 000 samples ( $\sim 1\ 000$  samples per reference). KNN results were very similar irrespective of the number of nearest neighbors taken into account but the best result was obtained for K=10. The classification results are shown in Figure 4.1.

The confusion matrix in subfigure 4.1a shows that there are two pairs of species often mistaken, *Salmonella enterica* and *Escherichia coli*, and *Enterococcus faecalis* and *Listeria monocytogenes*.

t-SNE plot of the representations can be seen in Figure 4.2.

The distribution of chunk representations shows that data is not clustered very well. While four out of seven microbe species are separated quite well, *Bacillus subtilis* is partially overlapped, *Salmonella enterica* is hidden behind *Escherichia coli* and *Enterococcus faecalis* is hidden behind *Listeria monocytogenes*. It is quite interesting to see that cluster belonging to one microbe species is additionally separated into subclusters. This could be due to forming positive samples for an anchor based on the contig and mapping position on the reference.

The representation network was then fine-tuned on a classification task. The classi-

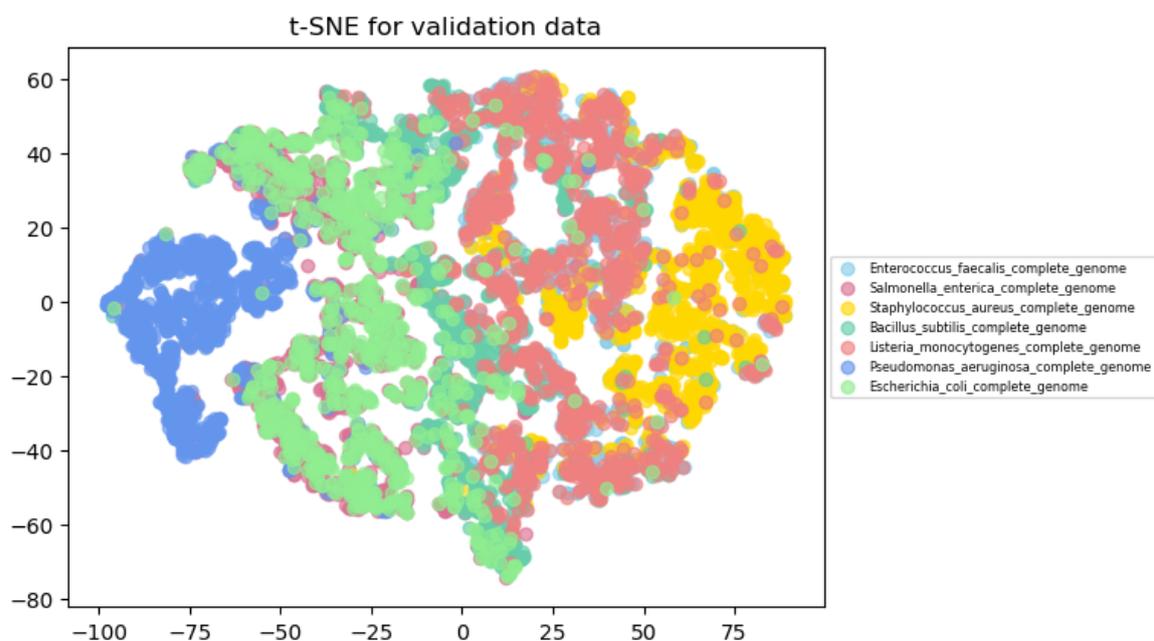


(a) Confusion matrix for K=10 on the reference-sampled dataset

Metric	Value
Accuracy	66.54%
Precision	66.58%
Recall	66.55%
F1	66.33%

(b) Classification metrics

**Figure 4.1:** KNN classification results for K=10 after representation pretraining



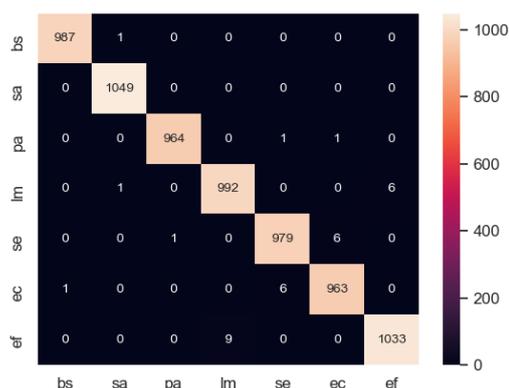
**Figure 4.2:** t-SNE visualization of reference-sampled chunks after representation pretraining

fication head was a simple fully connected layer with a 192-dimensional input and a 7-dimensional output.

KNN results were similar irrespective of the number of nearest neighbors taken into account but the best result was obtained for K=6 in this case as well. The classification results are shown in Figure 4.3.

Results in Figure 4.3 show significant improvement of results after the fine-tuning. The chunks are classified almost perfectly.

After taking a look at the distribution of reads in Figure 4.4, it is obvious that classifica-

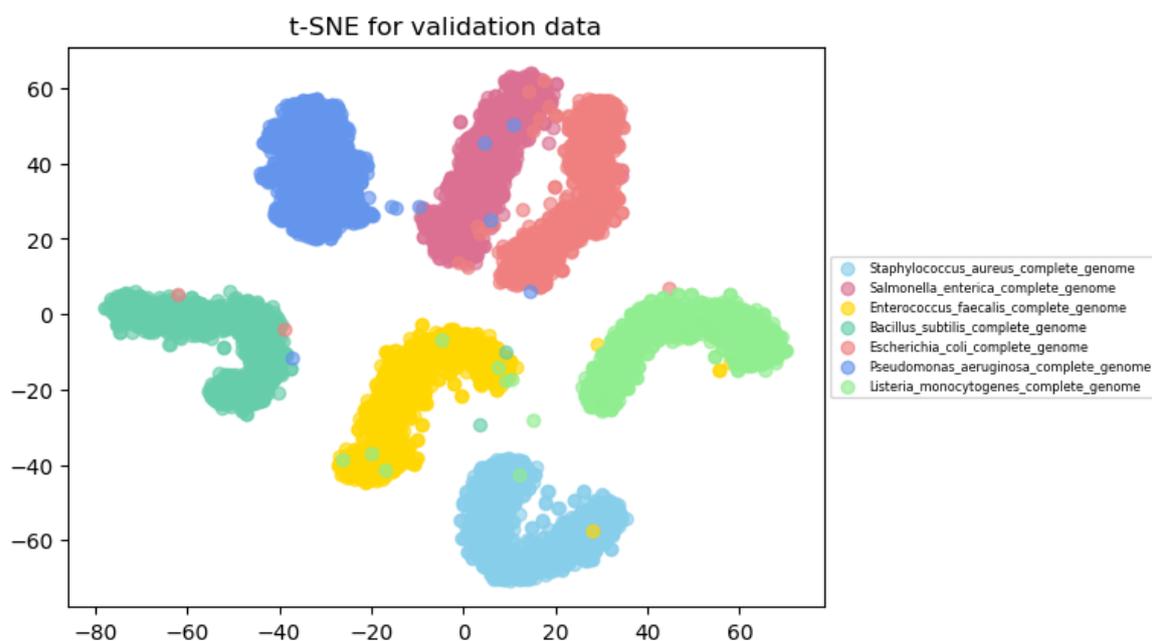


Metric	Value
Accuracy	99.53%
Precision	99.53%
Recall	99.53%
F1	99.53%

(b) Classification metrics

(a) Confusion matrix for K=6 on reference-sampled dataset

**Figure 4.3:** KNN classification results for K=6 after classification fine-tuning



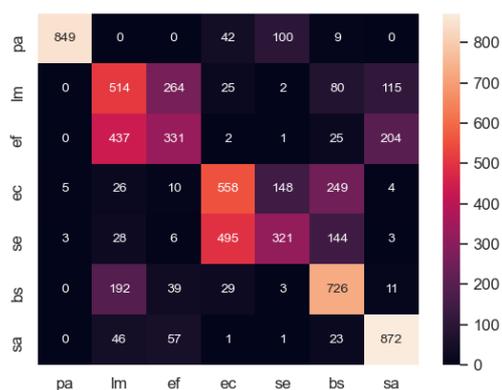
**Figure 4.4:** t-SNE visualization of reference-sampled chunks after classification fine-tuning

tion fine-tuning contributed to cluster separation.

However, these evaluations were done on reference-sampled chunks which are somewhat idealized. To truly evaluate this model, it should be tested on a set of real reads. For this purpose, a test set of real reads was sampled from the original Zymo mock community database. 7 000 samples were sampled from 7 microbe species (1 000 per species).

The results of testing on a network that was pretrained on a representation task are shown in Figure 4.5.

As Figure 4.5 shows, the results of the model pretrained on the triplet problem are poor.



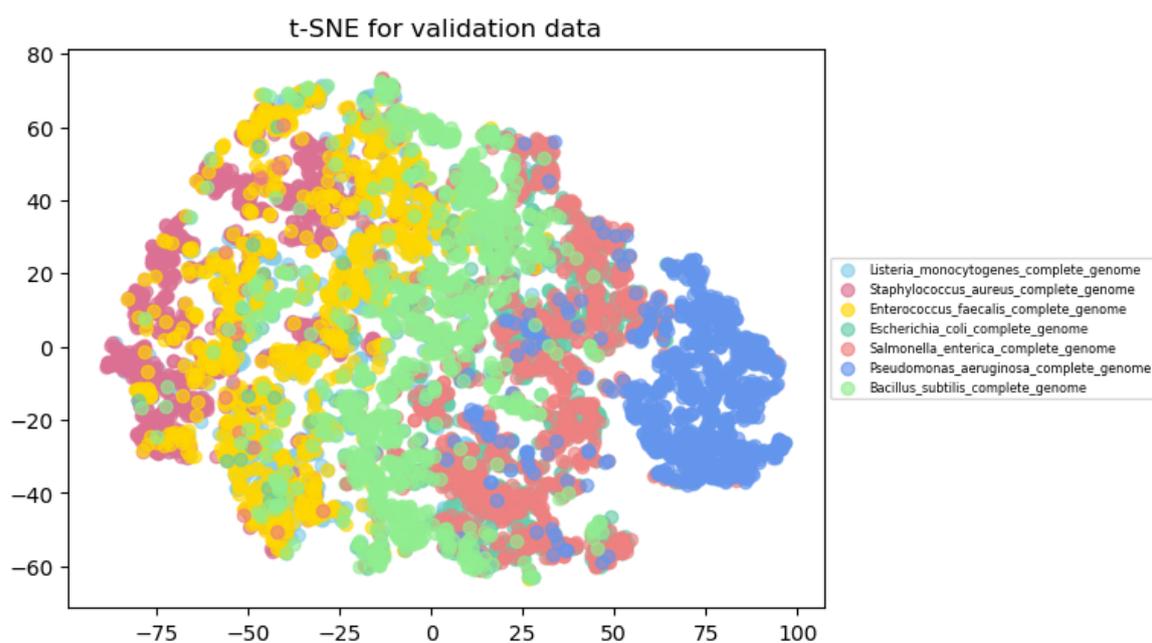
Metric	Value
Accuracy	59.59%
Precision	59.59%
Recall	60.19%
F1	58.86%

(b) Classification metrics

(a) Confusion matrix for K=11 on dataset of real reads

**Figure 4.5:** KNN classification results for K=11 after representation pretraining

Less than 60% of reads are classified correctly.

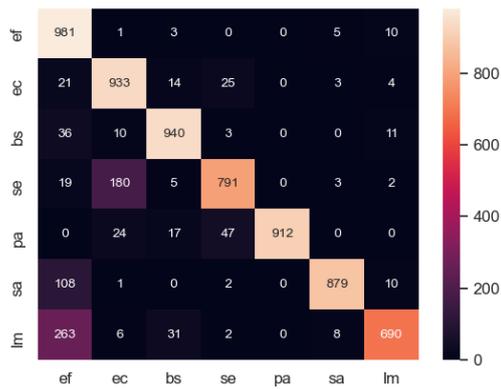


**Figure 4.6:** t-SNE visualization of real reads after representation pretraining

These results are supported by Figure 4.6 which shows that different species are not separated very well. The results on the same dataset after fine-tuning the representation model on the classification task are shown in Figure 4.7.

Figure 4.7 shows significant improvement of the results compared to results after representation pretraining. These results are supported by Figure 4.8.

Although the results on real reads are significantly improved by performing classification fine-tuning, the results on real reads are still not comparable to those on reference-sampled chunks. This is expected since the model is trained solely on reference-sampled chunks

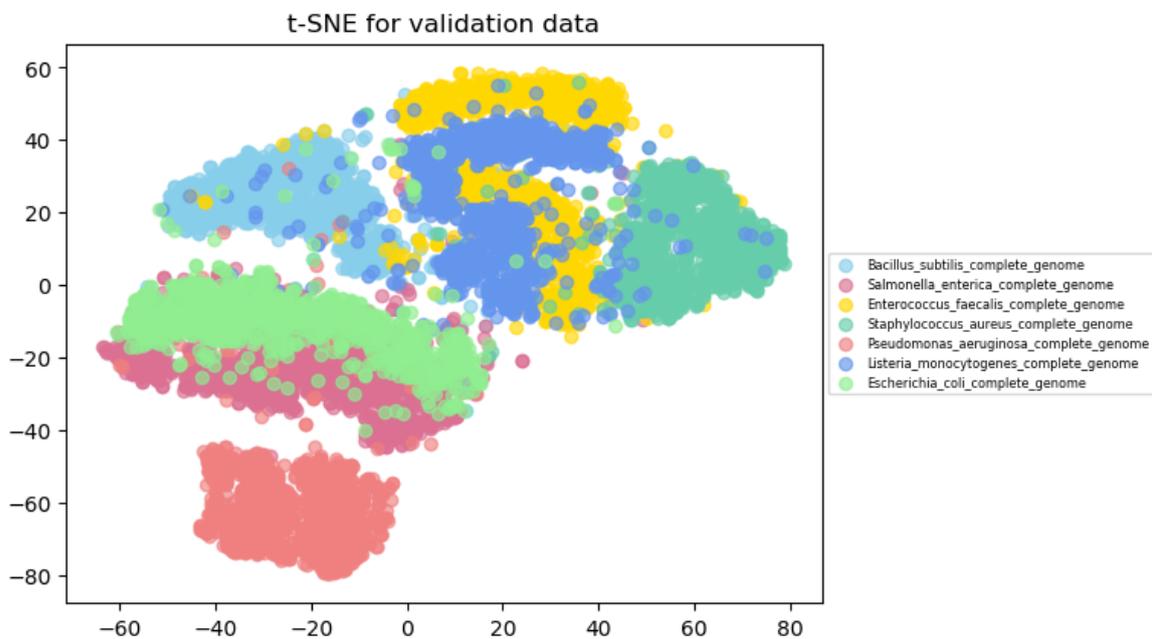


Metric	Value
Accuracy	87.51%
Precision	87.51%
Recall	89.47%
F1	87.64%

(b) Classification metrics

(a) Confusion matrix for K=9 on dataset of real reads

**Figure 4.7:** KNN classification results for K=9 after classification fine-tuning



**Figure 4.8:** t-SNE visualization of real reads after classification fine-tuning

which are idealized samples of data. For these reasons, real reads should be included in the training. There are two possibilities, training solely on real reads or training on a combination of reference-sampled chunks and real reads. While training solely on real data would make the setting most truthful, it would also include having only noisy data. Reference-sampled chunks bring stability and in a way augment the real data. For these reasons, it is decided to train models with a combination of real data and reference-sampled chunks.

## 4.2. Training on a combination of reference-sampled chunks and real reads

For this experiment, a new training dataset was formed out of 100 000 chunks sampled from the references and 200 000 reads sampled from the original Zymo mock community database. In the initial experiments, it was shown that reads of significantly shorter length than the target length ( $9 \times 512 = 4608$  bases) brought too much noise into training which is why reads shorter than  $0.65 \times 4608 = 2995$  bases were filtered out of the training set.

The architecture of the network remained the same and representation training ran for 24 epochs with triplet loss converging from 2.5 (which was the margin) to 0.36. The performance of the model was tested on a set of 7 000 real reads (1 000 from each microbe) and the results can be seen in Figure 4.9.



Metric	Value
Accuracy	62.58%
Precision	62.39%
Recall	62.59%
F1	62.35%

(b) Classification metrics

(a) Confusion matrix for K=14 on dataset of real reads

**Figure 4.9:** KNN classification results for K=14 after representation pretraining

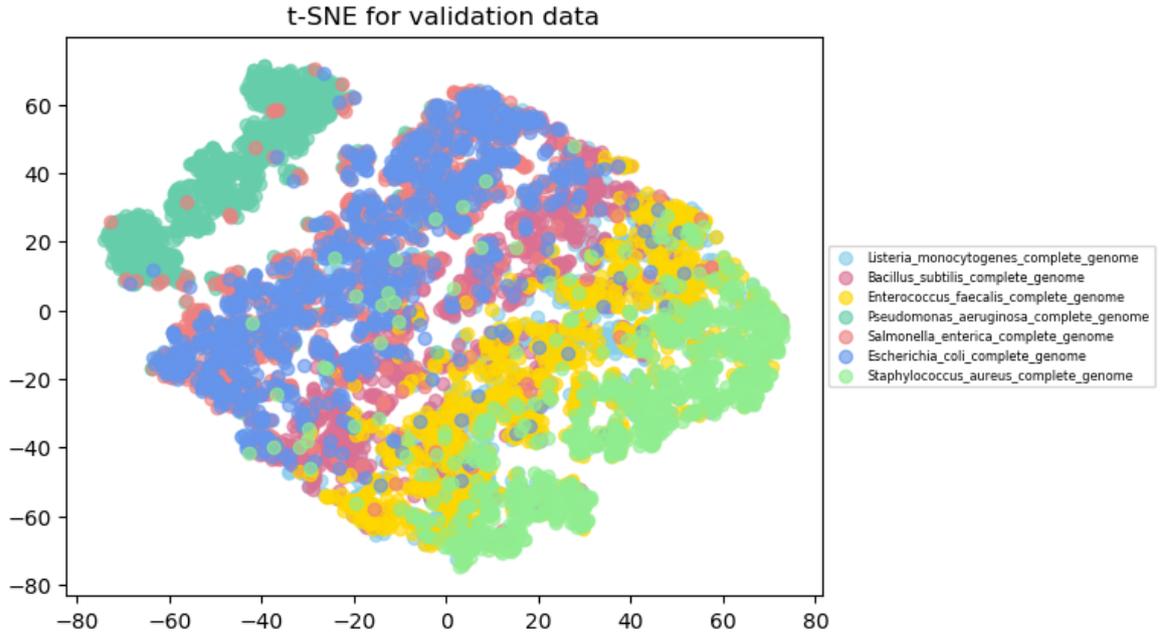
Figure 4.9 shows slight improvement on real reads once the training is done combining reference-sampled chunks and real reads. However, the biggest problem remain two pairs of species, *Escherichia coli* and *Salmonella enterica*, and *Listeria monocytogenes* and *Enterococcus faecalis*.

These results are supported by the distribution of reads per class shown in Figure 4.10. *Escherichia coli* and *Salmonella enterica*, and *Listeria monocytogenes* and *Enterococcus faecalis* are completely overlapped while *Bacillus subtilis* is partially overlapped by the same two clusters.

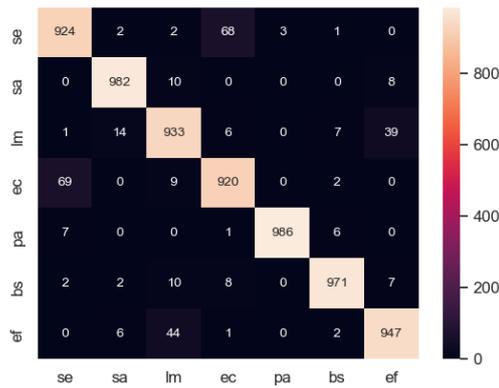
The results after classification fine-tuning are shown in Figure 4.11.

The results show significant improvement in performance after classification fine-tuning. These results are supported by the data distribution plot in Figure 4.12.

Figure 4.12 shows decent separation between different microbe clusters. The most prob-



**Figure 4.10:** t-SNE visualization of real reads after representation pretraining



Metric	Value
Accuracy	95.19%
Precision	95.19%
Recall	95.20%
F1	95.19%

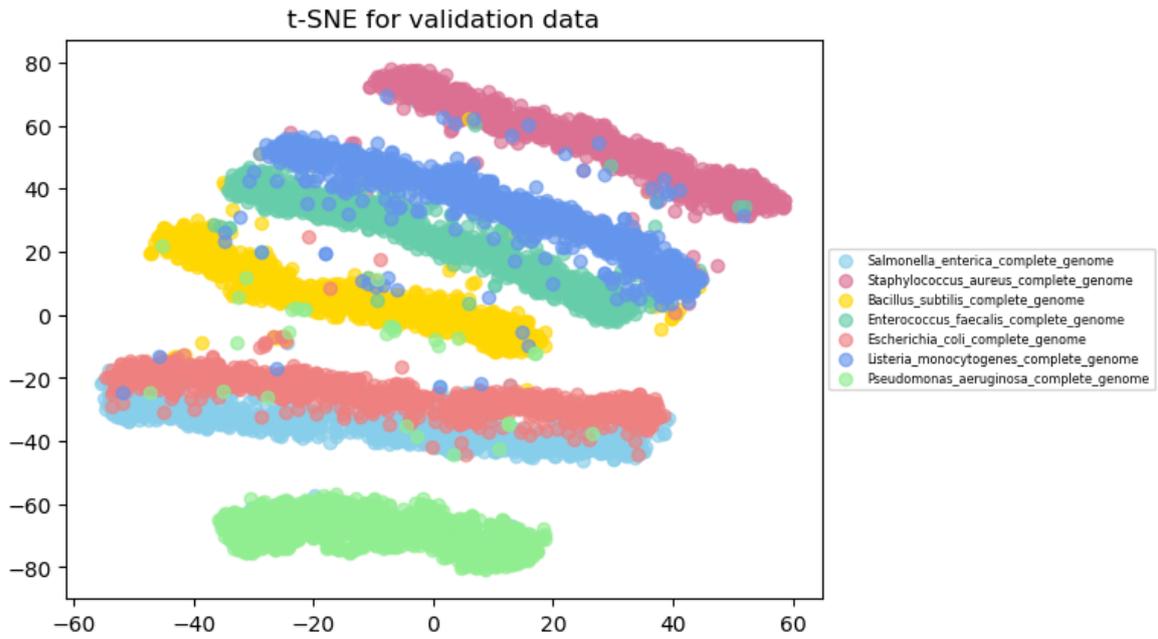
**(b)** Classification metrics

**(a)** Confusion matrix for K=9 on dataset of real reads

**Figure 4.11:** KNN classification results for K=9 after classification fine-tuning

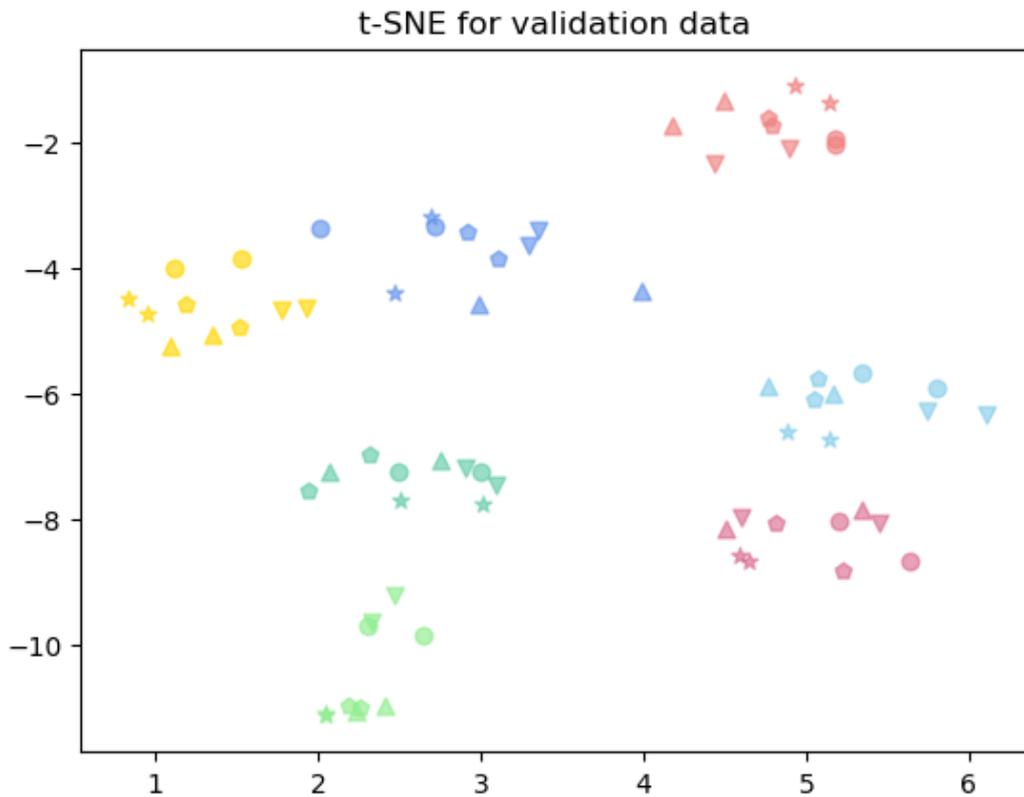
lematic remain *Salmonella enterica* and *Escherichia coli* that also share the highest ANI score which makes them the most similar species.

Now that KNN results show a decent microbe detection performance, it is interesting to inspect how distant are representations of two chunks sampled from the same reference and within <9 positions from each other (since the size of k-mers is set to 9). By choosing starting positions in such a manner, a pair of very similar chunks with a completely different sequence of 9-mers is sampled. For this purpose, a set of anchor chunks was sampled from each of the seven references. Additionally, for each anchor, a chunk starting 5 bases from the anchor start was sampled as well. That way, pairs of closely related chunks with no shared



**Figure 4.12:** t-SNE visualization of real reads after classification fine-tuning

9-mer embeddings were determined and forwarded to the network.

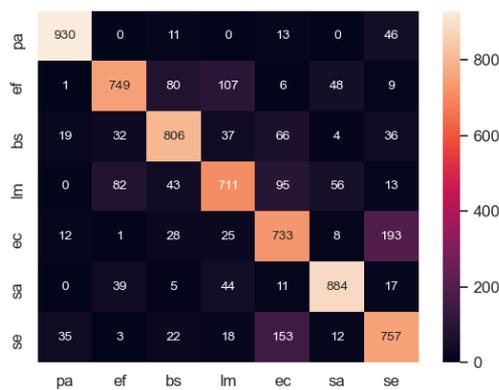


**Figure 4.13:** t-SNE for close reference-sampled chunks

Representations of such pairs of chunks are visualized in Figure 4.13. All samples belonging to the same microbe are visualized in the same color while each pair of neighbor

chunks is represented using the same shape. The figure shows that samples that are sampled from the same reference and with a distance of <9 bases often have representations that are closer to each other than to any other sample despite not sharing a sequence of k-mers. The evaluation is done on a model that was fine-tuned on a classification task which shows that fine-tuning on such task does not interfere with the representation task.

The length of the input sequence is limited to 512 input points (which corresponds to the length of 4608 bases in the input read) and those reads that are longer than the set maximum length are trimmed to maximum length. On the other hand, reads under 2995 bases are considered to be extremely short, and noisy and are, therefore, removed from the training dataset. This makes evaluating the performance of the model on a set of reads that are shorter than 2995 bases and a set of reads that are longer than 4608 bases interesting.



Metric	Value
Accuracy	79.57%
Precision	79.79%
Recall	79.57%
F1	79.62%

(b) Classification metrics

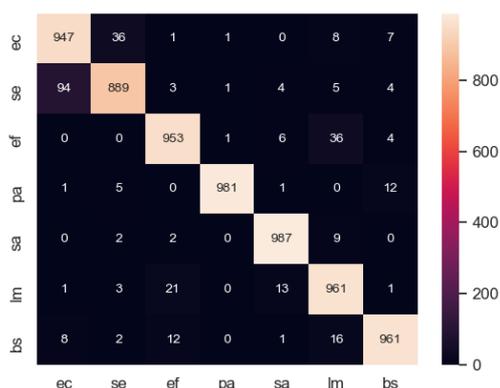
(a) Confusion matrix for K=9 on dataset of short reads

**Figure 4.14:** KNN classification results for K=9 on a set of short reads

Results on reads that are shorter than 2995 bases are shown in Figure 4.14. The biggest problem remain Escherichia coli and Salmonella enterica since these are the two most similar species.

Results on reads longer than 4608 bases are comparable to those on all reads longer than 2995 bases. This is expected since longer reads are being trimmed to the length of 4608 bases and then forwarded through the network.

However, since all of the previous results show that pairs of species with high ANI score impose a significant problem to microbe detection, once there are more microbes in the database, and many of them are quite similar to one another, taking only longer reads into account (in their full length) might be necessary for successful microbe detection. There are three options for including longer reads in their full length (or near their full length) in microbe detection: training on longer input sequences, modifying representation learning, and modifying classification learning, all three of which will be explored and presented in



(a) Confusion matrix for K=9 on dataset of long reads

Metric	Value
Accuracy	95.41%
Precision	95.47%
Recall	95.41%
F1	95.41%

(b) Classification metrics

**Figure 4.15:** KNN classification results for K=9 on a set of long reads

the following section.

## 4.3. Extensions for longer reads

### 4.3.1. Training with longer input sequences

This experiment required one significant architectural change and that is the length of the input sequence. While previous experiments were done on sequences with 512 input points, this experiment was done on input sequences with a maximum length of 1024 input points which, in combination with 9-mer as single-point inputs, results in network being able to process reads of maximum length  $\leq 9216$  bases. As Table 2.4 shows, almost 92% of reads fall into this category. The margin was set to 5, 36 representation, and 50 classification training epochs were done. The performance of this model will be evaluated on reads that are longer than 4806 bases since these are the reads that are expected to benefit from training on longer sequences.

The results in figures 4.16 and 4.17 show that the model that was only pretrained on the representation task still does not yield satisfactory results, especially if compared to the model that was fine-tuned on the classification task. This model shows slight improvement compared to results in Figure 4.15 but requires the input sequence to be twice as long to gain small improvement.

### 4.3.2. Averaging representation learning

Having a model that expects longer sequences as the input creates heavier time and memory demands and slows down the learning and inference process. Therefore, a different approach

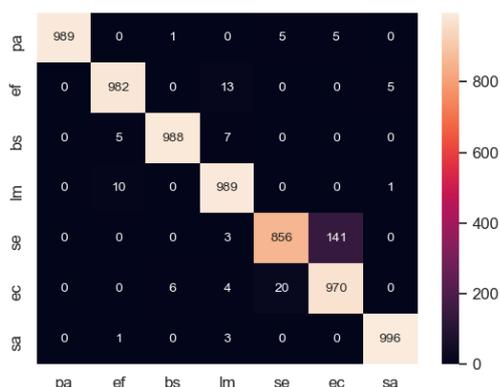


(a) Confusion matrix for K=9 on dataset of long reads

Metric	Value
Accuracy	65.60%
Precision	65.44%
Recall	65.60%
F1	65.21%

(b) Classification metrics

**Figure 4.16:** KNN classification results for K=9 after representation pretraining with max sequence len = 9216



(a) Confusion matrix for K=9 on dataset of long reads

Metric	Value
Accuracy	96.71%
Precision	96.89%
Recall	96.71%
F1	96.71%

(b) Classification metrics

**Figure 4.17:** KNN classification results for K=9 after classification fine-tuning

to including longer reads in their full length was explored. Averaging representation learning implies training the model on a sequence of 512 input points, which corresponds to the usual setting presented in this work. However, while reads longer than the maximum 4608 bases were trimmed in previous experiments, in this experiment they are treated differently. Reads longer than the expected maximum length are being chunked into pieces of length  $\leq 4608$ . The only chunk that is potentially shorter than 4608 bases is the last one. That chunk was treated the same as reads shorter than 4608 bases were. If the last chunk is longer than  $0.65 \cdot 4608 = 2995$  bases it is padded and forwarded to the network, otherwise, it was discarded. All of these chunks were forwarded into the network and once representation vectors of all of the chunks were obtained, representation vectors of chunks belonging to the same read were mean averaged to obtain a single representation vector for the entire read.



(a) Confusion matrix for K=14 on dataset of long reads

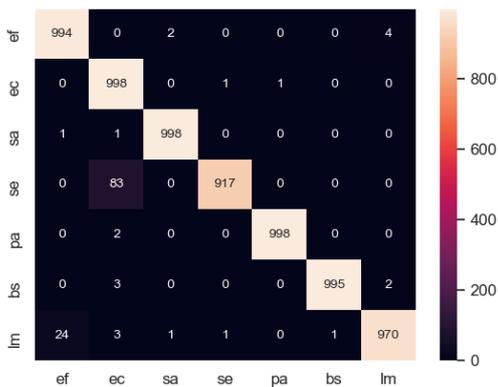
Metric	Value
Accuracy	67.46%
Precision	70.34%
Recall	67.46%
F1	66.03%

(b) Classification metrics

**Figure 4.18:** KNN classification results for K=14 after representation averaging pretraining

The results on a set of reads longer than 4608 bases after representation pretraining are shown in Figure 4.18. The results show slight improvement over training with longer input sequences.

The results are, however, still not quite satisfactory which is why classification fine-tuning was done here, as well.



(a) Confusion matrix for K=9 on dataset of long reads

Metric	Value
Accuracy	98.14%
Precision	98.25%
Recall	98.14%
F1	98.15%

(b) Classification metrics

**Figure 4.19:** KNN classification results for K=14 after classification fine-tuning

The results after classification fine-tuning shown in Figure 4.19 show a significant improvement over representation pretraining. With an accuracy score over 98%, these are the best results obtained on reads longer than the input sequence.

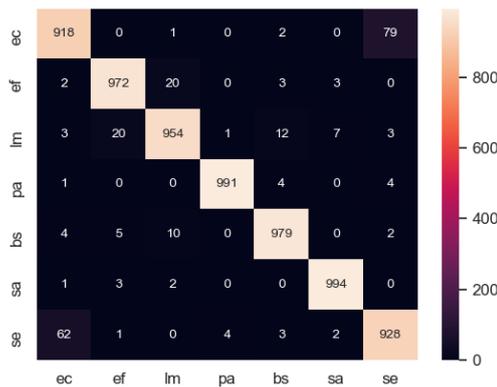
Since both representation pretraining, and classification fine-tuning results show some improvement over previous methods, averaging representations for longer reads seems like a valid approach to microbe detection task, especially when focusing on really long reads.

### 4.3.3. Majority voting classification learning

For majority voting classification learning, a network that was already pretrained on representation task. That network was then used for majority voting classification learning. Reads longer than 4608 bases are the focus of this approach, as well. Reads longer than the expected maximum length are being chunked into pieces of length  $\leq 4608$ . The only chunk that is potentially shorter than 4608 bases is the last one. That chunk was treated the same as reads shorter than 4608 bases were. If the last chunk is longer than  $0.65 \cdot 4608 = 2995$  bases it is padded and forwarded to the network, otherwise, it was discarded. Representation was created for each chunk and that representation was forwarded to the classification head. Probabilities of belonging to each microbe class are output for all chunks that were forwarded to the network. Logits for chunks belonging to the same read are then mean-averaged and final read-level microbe prediction is done.

This experiment is done without combining the parametric classification fine-tuning with non-parametric classification since the difference between this and other approaches is in choosing the prediction based on averaged probabilities the classification head outputs per chunk of read. The non-parametric approach requires removing the classification head from the model, therefore eliminating the majority voting extension.

Majority voting classification fine-tuning ran for 35 epochs and was evaluated on a set of reads longer than 4608 bases as well. The testing dataset consisted of 7 000 samples (1 000 samples per microbe). The results are shown in Figure 4.20.



Metric	Value
Accuracy	96.23%
Precision	96.23%
Recall	96.23%
F1	96.23%

(b) Classification metrics

(a) Confusion matrix for majority voting classification on long reads

**Figure 4.20:** Majority voting classification results

The results show that the majority voting approach comparable to previous results and with a slight improvement over the original trimming approach. However, the results did not exceed results obtained by mean averaging the chunk-level representations. That might be

a consequence of introducing high level of noise by mean-averaging the logits before performing a single prediction. The results may be improved by performing a simple majority voting on predictions rather than on logits.

## 4.4. Discussion

All of the previous results show that training a model on the representation task utilizing triplet loss yields results that solely do not fulfill the need for an accurate microbe detection. However, it provides a strong ground for the classification fine-tuning. Once the representation model is modified for the microbe classification task and fine-tuned on that task, the microbe detection results improve significantly.

It is also showed that the accuracy is mostly compromised on a set of reads that are relatively short. This is understandable taking into account that all of the microbes are quite similar meaning they share a lot of regions, especially the short ones. All of these experiments are done a set of reads and chunk belonging to one of the seven microbial species included in this work. However, the microbe detection would in a realistic setting be applied to a much bigger number of species which is when the similarity problem between different species would intensify, especially for relatively short reads/regions.

On the other hand, performance on reads that are longer than the expected input sequence is shown to be quite good, even in the initial setting where long reads were trimmed to maximum sequence length. All of the additionally explored training extensions designed to utilize longer reads in their full (or near full) length have, however, shown slight improvements over the initial trimming approach.

All of these results show that the focus of the future work should be on longer reads. By focusing on these reads and further developing representation and classification techniques, the system could be scalable to a significantly larger number of microbial species.

Of course, ideally, the representation training would be sufficient for an accurate non-parametric microbe detection. Therefore, the accent of future work would primarily be on improving representation learning methods by modifying triplet construction or finding a completely new learning method for this task.

## 5. Conclusion

The main goal of this work was to find a fast and accurate method for microbe detection. Different deep learning settings were explored and evaluated on sets of reads originating from seven microbial species: *Bacillus subtilis*, *Enterococcus faecalis*, *Escherichia coli*, *Listeria monocytogenes*, *Pseudomonas aeruginosa*, *Salmonella enterica*, and *Staphylococcus aureus*. The datasets of reads are obtained through the third-generation - Nanopore, sequencing and simulated chunks are obtained from genome references. Since the usual length of biological data obtained through Nanopore sequencing has length that can go up to few thousand bases, the dimensionality of the original data needed to be reduced because deep learning model usually perform badly on sequences of such length. This was done by chunking the original sequences in fragments of length  $k$ , so-called k-mers. Each k-mer then presented an input point in the sequence.

The representation network architecture was based on a novel deep learning architecture called the transformer. The transformer network is based on a deep learning mechanism called attention and it revolutionized sequence modelling problems. For this work, the transformer network originally developed for natural language processing tasks was adapted to microbe detection problem. First, the input sequences of k-mers needed to be converted into a number representation which was through an embedding process where k-mer was represented by real-valued vector. The sequence of embedding vectors was then forwarded to the representation network that output a sequence of updated embedding vectors for each k-mer, so-called dynamic embeddings (whereas the initial embedding are called static embeddings). Final read-level representations were obtained by averaging dynamic k-mer embeddings. Representation learning process was followed by a classification part where each representation was assigned a microbe.

The affect of different training settings on the microbe detection ability of a deep learning model was explored by mainly focusing on variants of data used for training and possibilities for extracting knowledge from long reads (since they showed a higher distinction potential). For this purposes, apart from the initial tests where long reads were trimmed to maximum input length, experiments with longer input sequences such as: representation averaging, and majority voting classification were done to explore possible performance improvements for

long reads.

While current results show a satisfactory accuracy of microbe detection after the classification fine-tuning, the microbe detection ability of the network that was only trained on representation task remains insufficient. The focus of the future work would, therefore, be on exploring different variants of representation training process with an accent on longer reads which provide more microbe-specific information. Additionally, since sequencing process produces current signals that are afterwards being base called, future work could also explore the possibility of including information extracted from the signals in the microbe detection process.

# BIBLIOGRAPHY

- D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate, 2016.
- E. Hoffer and N. Ailon. Deep metric learning using triplet network, 2018.
- N. Ketkar. Stochastic gradient descent. In *Deep Learning with Python*, pages 113–132. Apress, 2017. doi: 10.1007/978-1-4842-2766-4\_8. URL [https://doi.org/10.1007/978-1-4842-2766-4\\_8](https://doi.org/10.1007/978-1-4842-2766-4_8).
- O. Kramer. K-nearest neighbors. In *Dimensionality Reduction with Unsupervised Nearest Neighbors*, pages 13–23. Springer Berlin Heidelberg, 2013. doi: 10.1007/978-3-642-38652-7\_2. URL [https://doi.org/10.1007/978-3-642-38652-7\\_2](https://doi.org/10.1007/978-3-642-38652-7_2).
- I. Lee, Y. O. Kim, S.-C. Park, and J. Chun. OrthoANI: An improved algorithm and software for calculating average nucleotide identity. *International Journal of Systematic and Evolutionary Microbiology*, 66(2):1100–1103, Feb. 2016. doi: 10.1099/ijsem.0.000760. URL <https://doi.org/10.1099/ijsem.0.000760>.
- H. Li. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics*, 34(18):3094–3100, 05 2018. ISSN 1367-4803. doi: 10.1093/bioinformatics/bty191. URL <https://doi.org/10.1093/bioinformatics/bty191>.
- M.-T. Luong, H. Pham, and C. D. Manning. Effective approaches to attention-based neural machine translation, 2015.
- H. M and S. M.N. A review on evaluation metrics for data classification evaluations. *International Journal of Data Mining & Knowledge Management Process*, 5(2):01–11, Mar. 2015. doi: 10.5121/ijdkp.2015.5201. URL <https://doi.org/10.5121/ijdkp.2015.5201>.
- S. M. Nicholls, J. C. Quick, S. Tang, and N. J. Loman. Ultra-deep, long-read nanopore sequencing of mock microbial community standards. *GigaScience*, 8(5), 05 2019. ISSN

2047-217X. doi: 10.1093/gigascience/giz043. URL <https://doi.org/10.1093/gigascience/giz043>. giz043.

S. Sharma and S. Sharma. Activation functions in neural networks. *Towards Data Science*, 6(12):310–316, 2017.

L. van der Maaten and G. Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008. URL <http://jmlr.org/papers/v9/vandermaaten08a.html>.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need, 2017.

## **Rapid microbe detection using deep learning**

### **Abstract**

Microbes are microscopic organisms invisible to the naked eye with a significant role in everyday life. The ability to detect and accurately classify them is essential to discover diseases, prescribe medication, keep a healthy lifestyle in general. The main goal of this thesis is to develop a deep learning method that can in a fast and accurate way detect a microbial species from short fragments of that species. The architecture was based on a novel NLP architecture called the Transformer, adapted to the microbe detection task and used to obtain compressed representations of the sequenced DNA fragments. Once the compressed representations were found, different classification methods were applied to output predictions of microbial species.

**Keywords:** bioinformatics, microbe detection, deep learning, attention, transformers, triplet loss, classification

## **Brza detekcija mikroba uporabom dubokog učenja**

### **Sažetak**

Mikrobi su mikroskopski organizmi nevidljivi golom oku, ali s vrlo važnom ulogom u svakodnevnom životu. Mogućnost njihovog prepoznavanja i točne klasifikacije osnova je u otkrivanju bolesti, prepisivanju lijekova i održavanju zdravog načina života generalno. Glavni cilj ovog rada bio je razviti metodu baziranu na dubokom učenju koja na brz i točan način otkriva mikrob na temelju kratkih fragmenata te vrste. Arhitektura mreže bazirana je na novoj arhitekturi prezentiranoj u sklopu obrade prirodnog jezika, transformator, koja je za potrebe detekcije mikroba prilagođena i korištena u svrhe dobivanja sažetih reprezentacija sekvencioniranih fragmenata DNA. Kada su sažete reprezentacije pronađene, različite klasiifikacijske metode korištene su za finalnu predikciju vrste mikroba.

**Ključne riječi:** bioinformatika, detekcija mikroba, duboko učenje, pozornost, transformator, trojni gubitak, klasifikacija