UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

MASTER THESIS num. 1417

Deep Learning Model for Base Calling of MinION Nanopore Reads

Marko Ratković

Zagreb, June 2017.

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

I would like to thank my mentor, Mile Šikić, for his patient guidance, encouragement and advice provided over the years.

I would also like to thank my family and friends for their continuous support.

In the end, honorable mentions go to Neven Miculinić for his help with this thesis.

CONTENTS

LIST OF FIGURES

List of Tables

1. Introduction

In recent years, deep learning methods significantly improved the state-of-the-art in multiple domains such as computer vision, speech recognition, and natural language processing [1][2]. In this thesis, we present application of deep learning in the field of Bioinformatics for analysis of DNA sequencing data.

DNA is a molecule that makes up the genetic material of a cell, and it is responsible for carrying the information needed for survival, growth, and reproduction of an organism. DNA is a long polymer of simple blocks called nucleotides connected together forming two spiraling strands to a structure called a double helix. Possible nucleotide bases of a DNA strand are adenine, cytosine, guanine, thymine usually represented with letters A, C, G, and T. The order of these bases is what defines genetic code.

DNA sequencing is the process of determining this sequence of nucleotides. Originally sequencing was an expensive process, but during the last couple of decades, the price of sequencing has drastically decreased. A significant breakthrough occurred in May 2015 with the release of MinION sequencer by Oxford Nanopore making DNA sequencing inexpensive and more available, even for small research teams.

Base calling is a process assigning sequence of nucleotides (letters) to the raw data generated by the sequencing device. Simply put, it is a process of decoding the output from the sequencer.

1.1. Objectives

The goal of this thesis is to show that the accuracy of sequencing data is not only limited by sequencing technology, but also by the underlying software used for base calling and can be further improved using different machine learning concepts. A novel approach for base calling of raw data using convolutional neural networks is introduced.

1.2. Organization

Chapter 2 gives more detailed explanation of the problem, background on nanopore sequencing and overview of state-of-the-art basecallers.

Chapter 3 describes in detail deep learning concepts used in later chapters.

Chapter 4 goes into implementation details, preprocessing methods and training of the deep learning model.

Chapter 5 explains the methodology used to evaluate obtained results and the results of testing performed on different datasets as well as comparison with state-of-the-art basecallers.

In the end, Chapter 6 gives a brief conclusion and possible future work and improvements of the developed basecaller.

2. Background

2.1. Sequencing

All sequencing technologies to date have constraints on the length of the strand they can process, which are much smaller than the genome for a majority of organisms, making sequencing the entire genome of an organism a difficult problem. To resolve this problem whole genome shotgun sequencing approach is used, in which multiple copies of the genome are broken randomly into numerous small fragments that can be processed by the sequencer. Sequenced fragments are called reads.

Genome assembly is the process of reconstructing the original genome from reads and usually starts with finding overlaps between reads. The quality of reconstruction heavily depends on the length and the quality (accuracy) of the reads produced by the sequencer.

Figure 2.1 depicts process of sequencing.

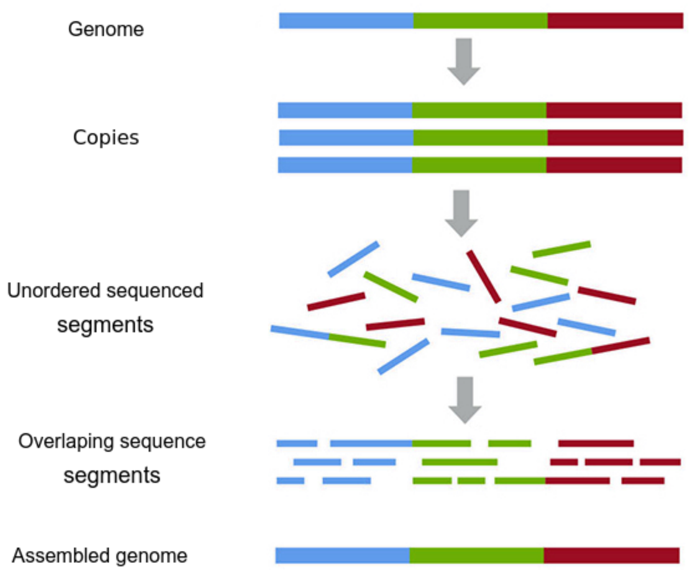


Figure 2.1: Depiction of the sequencing process

Development of sequencing started with work of Frederick Sanger [3] [4]. In 1977, he developed the first sequencing method which allowed read lengths up to 1000 bases with very high accuracy (99.9%) at the cost of 1$ per 1000 bases. Second generation sequencing

(IAN Torrent and Illumina devices) reduced the price of sequencing while maintaining high accuracy. Mayor disadvantage of these devices is read length of only a few hundred base pairs. Short reads make resolving repetitive regions practically impossible.

The need for technology able of producing longer reads led to the development of so-called third generation sequencing technologies. PacBio developed sequencing method that allowed read lengths up to several thousand bases but at the cost of smaller accuracy. Error Rates of PacBio devices are ~10-15%.

Cost makes the biggest obstacle stopping widespread genome sequencing. The release of, previously mentioned, MinION sequencer made sequencing less expensive and even portable.

2.2. Oxford Nanopore MinION

The MinION device by Oxford Nanopore Technologies is the first portable DNA sequencing device. Its small weight, low cost, and long read length combined with decent accuracy yield promising results in various applications including full human genome assembly [5] what could potentially lead to personalized genomic medicine.

2.2.1. Technology

As its name says, nanoscaled pores are used to sequence DNA. An electrical potential is applied over a membrane in which a pore is inserted. As the DNA passes through the pore, the sensor detects changes in ionic current caused by different nucleotides present in the pore. Figure 2.2 shows the change of ionic current as DNA strain is pulled through a nanopore.

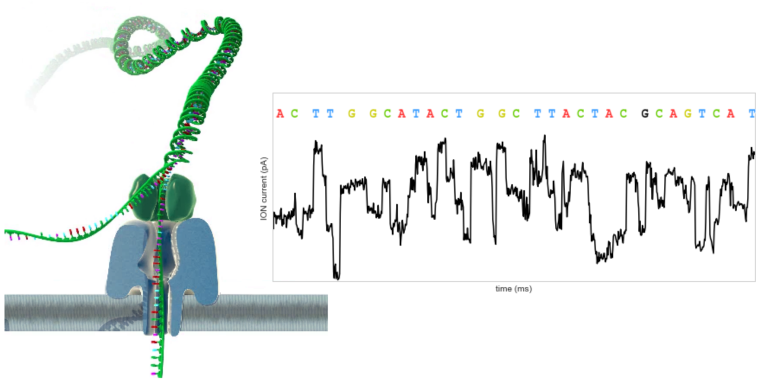


Figure 2.2: DNA strain being pulled through a nanopore [1]

[1]Figure adapted from https://nanoporetech.com/how-it-works

Official software called MinKNOW outputs sequencing data in FAST5 (a variant of the HDF5 standard) file format. It is a hierarchical file format with data arranged in a tree-structure of groups. Metadata are stored in group and dataset attributes. The same file format is during used different stages of analyses and groups, datasets and attributes are added incrementally. Figure 2.3 shows raw signal being present in the FAST5 file.

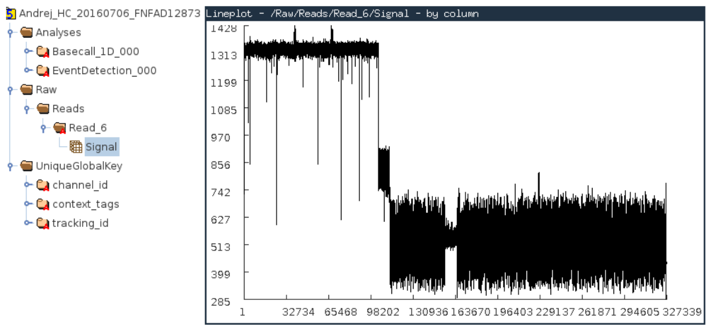


Figure 2.3: Structure of FAST5 file and raw signal line plot show in *HDFView* [2]

Minion offers the possibility of sequencing one or both strands of DNA. Sequencing both strands and combining information results in reads of higher quality. Those reads are called 2D (two-dimensional) reads. Otherwise, if the only single strand is sequenced 1D (one-dimensional) reads are produced.

MinION devices can produce long reads, usually tens of thousand base pairs (with reported reads lengths of 100 thousand [6] and even recently above 800 thousand base pairs [7]), but with high sequencing error than older generations of sequencing technologies. Switch from older R7.3 to R9 chemistry in 2016 increased accuracy of produced data. With this change, the accuracy of 1D data increased from 70% to 85% and the accuracy of 2D reads from 88% to 94% [8]. This increase of accuracy makes 1D reads usable for analysis with benefits over 2D reads being faster sample preparation and faster sequencing. Developed tool in this thesis focuses on base calling 1D reads.

2.3. Existing basecallers

2.3.1. Official

Oxford Nanopore has, with the R9 version of the platform, introduced a variety of base calling options. Some of those are production ready and some experimental. The majority of information regarding differences, specifications and similar is only available through Nanoporetech Community [3].

[2]https://support.hdfgroup.org/products/java/hdfview/

[3]https://community.nanoporetech.com/

Metrichor is an Oxford Nanopore company that offers cloud-based platform *EPI2ME* for analysis of nanopore data. Initially, base calling was only available by uploading data to the platform - that being the reason why this basecaller is often called Metrichor even though it is a name of the company.

The older version of Metrichor relied on *hidden Markov models* (HMM) to find the biological sequence corresponding to the signal. Preprocess included segmentation of the signal into smaller chunks called events defined by start location of the chunk, length, mean value and variance of the signal in the chunk. Metrichor than assumed that each event usually corresponds to a context of 6 bases being present in the pore and that the context is typically shifted by one base in each step. The states of HMM are modeled as a context present in the pore and transition correspond to change of bases in the pore. During the transition from one state to another, an event is emitted. Base calling is performed using the Viterbi algorithm which determines the most likely sequence of states for the observed sequence of events. This approach showed poor results when calling long homopolymer stretches as the context in the pore remains the same [9].

With the release of R9 chemistry, this model was replaced by a more accurate recurrent neural network (RNN) implementation. Currently, Oxford Nanopore offers several RNN-based local basecaller versions under different names [10]: Albacore, Nanonet and basecaller integrated into MinKNOW.

Albacore is basecaller by Oxford Nanopore Technologies ready for production and actively supported. It is available to the Nanopore Community served as a binary. The source code of Albacore was not provided and is only available through the ONT Developer Channel. Tool supports only R9.4 and future R9.5 version of the chemistry.

Nanonet[4] uses the same neural network that is used in Albacore but it is continually under development and does contain features such as error handling or logging needed for production use. It uses *CURRENNT* library for running neural networks. It supportes basecalling of both R9 and R9.4 chemistry versions.

Scrappie[5] is another basecaller by Oxford Nanopore Technologies. Similar to Nanonet, it is the platform for ongoing development. Scrappie is reported to be the first basecaller that specifically address homopolymer base calling. It became publicly available just recently in June, 2017 and supports R9.4 and future R9.5 data.

[4] `https://github.com/nanoporetech/nanonet/`

[5] `https://github.com/nanoporetech/scrappie`

2.3.2. Third-party basecallers

Nanocall [11] was the first third-party open source basecaller for nanopore data. It uses HMM approach like the original R7 Metrichor. Nanocall does not support newer chemistries after R7.3.

DeepNano [12] was the first open-source basecaller based on neural networks. It uses bidirectional recurrent neural networks implemented in Python, using the Theano library. When released, originally only supported R7 chemistry, but support for R9 and R9.4 was added recently.

3. Methods

The process of base calling can be represented as the problem of machine translation where a sentence is translated from one language to another. For base calling, the sequence of events or current measurements is *translated* to the sequence of nucleotides (letters A, C, T, and G).

This section explains some key deep learning concepts needed to understand the final model. It gives general idea behind recurrent neural networks used in a majority of existing basecallers and possible problems that serve as motivation for the different approach - usage of convolutional neural networks.

3.1. Architecture

3.1.1. RNN

Recurrent neural networks can be viewed as a simple feed-forward network with the difference that the current output does not only depend on the current input but previous inputs as well. RNNs store that information in their hidden state which is updated in each step. The figure shows simple RNN and the same RNN unfolded in time. Unrolling is a way of showing how network processes each input in the sequence and updates its hidden state (show in figure 3.1).

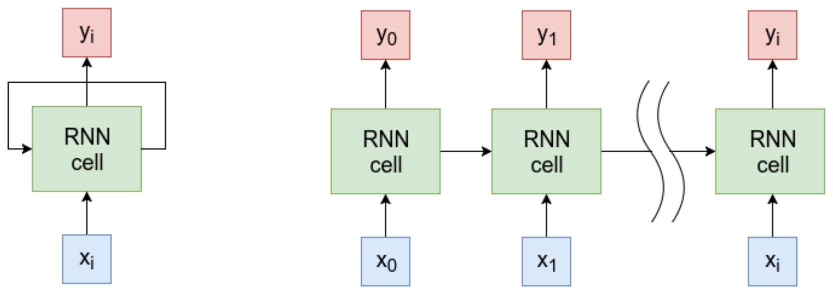


Figure 3.1: An unrolled recurrent neural network

These networks are trained using a variant of backpropagation called backpropagation through time which is essentially the same as classical backpropagation on an unfolded network. The gradient is propagated through the entire recurrence relation, and the gradient is multiplied in each step with the factor, depending on a scale it can make gradient vanish (drop to 0) or exponentially grow each step and explode. Detailed explanation can be found [13]. These issues are called the vanishing and exploding gradient and are generally resolved by a variant of RNN called *LSTM* [14].

Bidirectional Recurrent Neural (BiRNN) networks are used when the current output not only depends on the previous elements in the sequence but also future elements. The idea is to combine two RNN (one in the positive direction, one in negative time direction) and have an output of the current state expressed as a function of hidden states of both RNNs and current input. This is the approach used in DeepNano [12].

One of the major drawbacks of all recurrent networks is computation time. RNNs operate sequentially as the output for the second step depends on the first step and so on, which makes parallelization capabilities of RNNs quite limited. This especially is the case for Bidirectional RNNs.

3.1.2. CNN

Convolutional Neural Networks (CNNs) were responsible for major breakthroughs in Image Classification and are the core of most Computer Vision systems today. More recently CNNs are applied being to problems in Natural Language Processing and show promising results [15][16].

Convolution can be easily explained as a sliding window function applied to a matrix or in the case of base calling, signal. The sliding window is called a kernel or a filter. Figure 3.2 shows an example of convolution with kernel size 3 and how output is calculated as a sum of element-wise multiplication of kernel elements and input vector. Stride defines by how much filter is shifted at each step. Usually, to preserve the same dimension, padding with zeros is added to the borders.

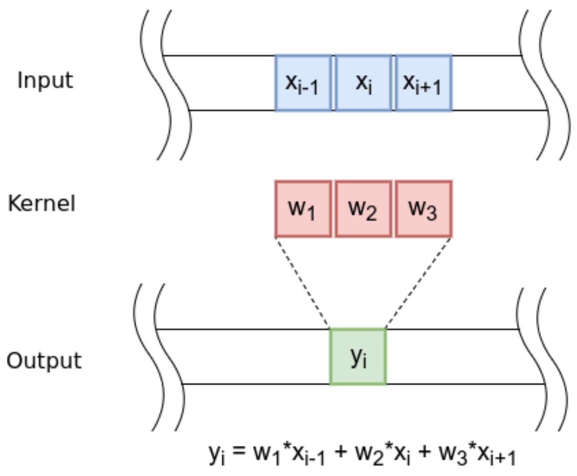


Figure 3.2: Convolution layer, kernel size 3 with stride 1.

Activations

After each convolution layer, usually nonlinear layer (know as activation layer) is applied. The purpose of this layer is to introduce nonlinearity to a system which consists of only linear operations as convolution layers are nothing more than just element-wise multiplications and summations. In classical neural networks, nonlinear functions like $tanh$ and $sigmoid$ were often used, but because of the undesirable property of saturation (at either end of 0 or 1 for $sigmoid$, -1 or 1 for $tanh$), other activations are more often used today with CNNs.

The Rectified Linear Unit (ReLU) has become very popular in the last few years. It is shown in [2] that usage of ReLU greatly accelerates the convergence of stochastic gradient descent compared to the sigmoid or tanh activations. Calculation of ReLU is much also more efficient as it is is simply thresholding at zero.

$$ReLU(x) = \begin{cases} x, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases} \tag{3.1}$$

The downside of ReLU is still saturation to the 0 on one side. Once in this state, the neuron is unlikely to recover because the function gradient at 0 is also 0, so gradient descent learning will not alter the weights. This is the problem known as *dying ReLU*. Different variants of ReLU, *PrRelu*, and *ELU* are often used to resolve this problem [17][18].

$$PrELU(x) = \begin{cases} x, & \text{if } x > 0 \\ \alpha x, & \text{otherwise} \end{cases} \tag{3.2}$$

$$ELU(x) = \begin{cases} x, & \text{if } x > 0 \\ \alpha(exp(x) - 1), & \text{otherwise} \end{cases} \tag{3.3}$$

Figure 3.3 shows different activation functions.

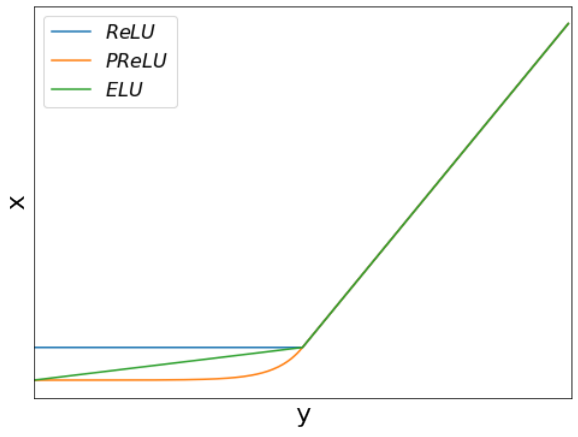


Figure 3.3: Activation functions

Pooling

The pooling layer is usually placed after the convolutional layer. Its primary utility lies in reducing the spatial dimensions of the input for the next convolution layer while preserving the most salient information. Pooling also provides basic invariance to translation.

Similar to the convolution layer described previously, the pooling layer also uses sliding window or a certain size that is moved across the input transforming the values. Usually, larger strides are used then in the convolution layers, as the purpose of this layer is subsampling. Most ofter, maximum value operation on the values in the window (max pooling) is used, but other transformations are possible (average pooling, L2-norm, or stochastic pooling). Figure 3.4 show dimensionality reduction by factor 2 using pooling with kernel size 2 with stride 2.

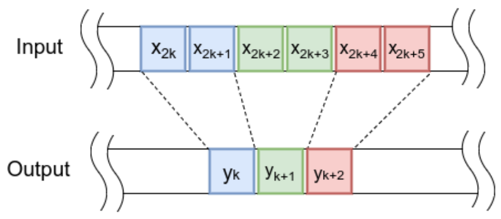


Figure 3.4: Dimensionality reduction by pooling (kernel size 2, stride 2)

Comparision with RNN

During calculation, each *patch* a convolutional kernel operates on is independent of the other, meaning that the entire input layer can be processed concurrently making CNNs usually more efficient than RNNs.

When compared with RNN in which output can depend on the entire sequence, in convolution layer, single output *sees* only limited window in the previous layer defined by kernel size. This is called the receptive field of the convolution. Figure 3.5 shows each new layers depends on larger portion of the input (z_i *sees* 5 elements of input). Lower layers see limited spatial information and are able to detect simple features like edges but through a series of convolutional layers, later layers can detect more abstract concepts using intermediate features detected from the whole input, or the signal in our case. This is the motivation behind deep convolution neural networks and why they are so popular in the field of image processing.

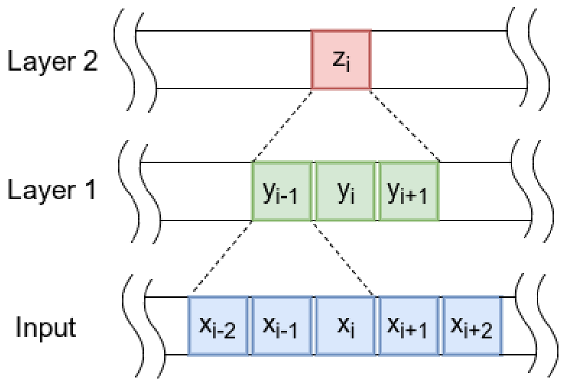


Figure 3.5: Receptive field after 2 layers of convolutions with kernel size 3

Stacking layers increases computational time as the input signal has to pass through the entire network but calculations at each layer can happen concurrently and each individual computation is small. In practice, even deep CNNs still have a big speed up over RNNS.

During the forward pass, input flows and is transformed, hopefully becoming a representation that is more suitable for the task. During the back phase, the gradient is propagated back through the network. Just like in RNNs, this signal gets multiplied and depending on the scales it can vanish resulting in no gradient flow to lower layers and no parameter upgrades. This limits the depth of the network. Resnet arhitecture [19] with its residual layers address this issue and allows deep architectures with steady gradient flow.

3.1.3. Residual Networks

A Residual Network or ResNet is a neural network architecture which solves the problem of vanishing gradients using a simple trick. Figure 3.6 shows on the left classical CNN that takes input and transforms it using convolution layers and activations. This can be represented as some nonlinear function $H(x)$. $H(x)$ can be written as a sum of some other nonlinear function $F(X)$ and linear member x. $F(X)$ is called the residual. Detailed explanation and comparisons are included in the original paper.

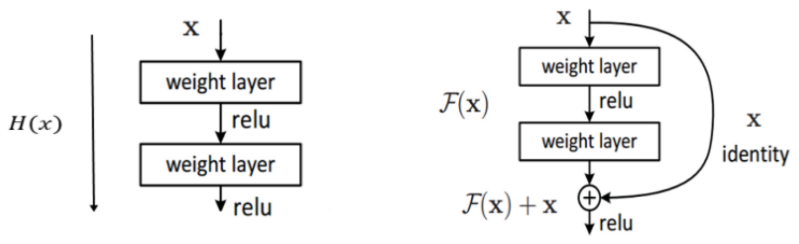


Figure 3.6: Comparison between classical CNN and CNN with the residual connection [1]

Instead of learning $H(X)$, network learns residual and at the output x is simply summed up to the $F(x)$ as shown in the figure. By stacking these layers, the gradient could theoretically *skip* over all the intermediate nonlinear layers and reach the bottom without vanishing.

3.2. CTC Loss

As mentioned previously, the goal of this thesis is to design model which can convert from a sequence of current measurements into a sequence of base pairs.

Suppose that we have an input sequence X (signal data) and the desired output sequence Y (nucleotides). X and Y will be of different lengths as the sequence of base pairs is always shorter than the length of the signal.

Instead of having a variable size of the output from the neural network, we can limit it to the length m and have sequences of variable length *decoded* from those outputs. The neural network can be considered to be simply a function that takes in some input sequence X (of length n) and generates sequence O (of length m). Output sequence of variable length Y is later *decoded* from O.

3.2.1. Definition

They key idea behind Connectionist Temporal Classification(CTC) [20] is that instead of directly generating output sequence Y as output from the neural network, we generate a probability distribution at every output length (from t=1 to t=m) that after *decoding* gives maximum likelihood output sequence Y. Finally, the network is trained using training dataset $D = \{(X_i, Y_i)\}$ by creating an objective function that restricts the maximum likelihood decoding for a given sequence X_i to correspond to our desired target sequence Y_i.

Given an input sequence X of length n, the network generates probabilities over all possible labels (A, C, T, and G) with an extra symbol "-" representing a *blank* at each timestep.

$$\Sigma = \{A, C, T, G\} \cup \{-\} \tag{3.4}$$

[1]Figure adapted from the original paper [19]

Possible output generated by the network is called *path*. Path is defined by the sequence of its elements $\pi = (\pi_1, \pi_2, ..., \pi_m)$ where π_i is from Σ. The probability of a given path π, given input sequence X, can then be expressed as the product of probabilities for each of its forming elements.

$$P(\pi|X) = \prod_{t=1}^{m} o_t(\pi_t), \tag{3.5}$$

where $o_t(\pi_t)$ is probability of element π_t being t^{th} element on path π

Real output sequence, for given path, is obtained by traversing the path and removing all blanks and duplicate letters. Let $decode(\pi)$ be the output sequence corresponding to a path π. As seen in expression 3.6 multiple path correspond to the same sequence $Y = "ACT"$.

$$ACT = \begin{cases} decode(A, A, A, C, T) \\ decode(A, A, C, -, T) \\ decode(-, A, C, T, T) \\ decode(-, -, A, C, T) \\ decode(A, C, C, C, T) \\ \vdots \\ decode(A, C, T, -, -) \end{cases} \tag{3.6}$$

The probability of output sequence Y is then the sum of probabilities of all paths that decode to Y:

$$P(Y|X) = \sum_{\pi \in decode^{-1}(Y)} P(\pi|X) \tag{3.7}$$

3.2.2. Objective

Given the dataset $D = \{(X_i, Y_i)\}$, training objective is the maximization of the likelihood of each training sample which is the same as the minimization of negative log likelihood:

$$L(D) = - \sum_{(X,Y) \in D} lnP(Y|X) \tag{3.8}$$

3.2.3. Output decoding

Given the probability distribution $P(Y|X)$ and given input sequence X, most likely Y^* can be computed.

$$Y^* = \underset{Y \in L^m}{\operatorname{argmax}} P(Y|X) = \underset{Y \in L^m}{\operatorname{argmax}} \sum_{\pi \in decode^{-1}(Y)} P(\pi|X), \tag{3.9}$$

where L^m set of all possible sequences over alphabet L

with length less than or equals to m

The probability of a single output sequence Y is the sum of probabilities of all paths that decode to Y and the most probable sequence is selected as the output. Calculation of all possible sequences is computationally intractable but exist several algorithms that approximate this decoding.

The naive possibility is to take the most probable path and say that output sequence corresponds to that path. This is not necessarily correct. For example, suppose we have one path with probability 0.1 corresponding to sequence A, and ten paths with probabilities 0.05 each corresponding to sequence B. Label B is preferable one since it has an overall probability of 0.5; however, this naive best path decoding would select label A, whose single path has higher probability the those for label B. This method is called *best path decoding*.

Better approximations can be calculated using *beam search decoding* proposed in paper [21]. The idea is an incremental construction of the most probable sequences in step t using N most probable sequences from the previous step. Parameter N is called beam width.

Probability of Y being extended with character c in step t is expressed like:

$$P(c, Y, t) = P(Y, t-1) * p(c, t|X), \tag{3.10}$$

where $p(c, t|X)$ is probability of c being t-th output from the network in t-th step.

Lets label with $\hat{Y}$ prefix of Y without last symbol and Y^e as last symbol in Y.

$$Y = \hat{Y} + Y^e, \text{ where } + \text{ is concatanation operator} \tag{3.11}$$

We can get Y by having $\hat{Y}$ in previous step and network output Y^e or simply by already having Y in previous step and network output blank($-$) or Y^e as blanks and repeats are merged during decoding.

$$P(Y, t) = P(Y, t-1)p(-, t|X) + P(Y, t-1)p(Y^e, t|X) + P(\hat{Y}, t-1)p(Y^e, Y, t) \tag{3.12}$$

Using this expression, using dynamic programing can efficiently be determine probability of some sequence Y.

Beam search decoder keeps set B of N most probable sequences. B is initially a set consisting of a single blank. In each time step t (from t=1 to t=m) all sequences from B are expanded and probabilities of new sequences are calculated using recursive relation 3.12. Expanded sequences are placed in the new set B'. At the end of each step, B is replaced by B' and truncated by keeping N most probable sequences. After last time step, the sequence in B with the highest probability is chosen as the final output. Full pseudocode can be found in original paper [21]. Detailed explanation and calculation of gradient can be found in original CTC paper [20], or this very detailed blog post [22].

3.3. Batch normalization

Batch normalization is method proposed in paper [23] that accelerates learning process. During training, parameters are updated, and distribution of outputs of each layer keep changing. A small change in the distribution of outputs in early layers can cause a drastic change in later layers, and those layers need to adapt to the new scale of their inputs. This change of distribution is called the internal covariate shift and results in slows learning. This is solved by centering each output from activations of the training batch to zero-mean and unit variance. After that learned scale and offset are applied. This process is called batch normalization. After training, mean and variance for each activation are computed on the whole training dataset rather than on mini-batches during training.

Batch normalization offers several advantages other than reducing internal covariant shift including more robust learning process by reducing reliance on the scale of the parameters and their initial values allowing the usage of larger learning rates and faster learning altogether. It is shown in the original paper that batch normalization also regularizes the model that could potentially improve the performance of the model.

4. Implementation

4.1. Data

Both used datasets show in table 4.1 be previously have passed through MinKNOW and had been basecalled by Metrichor. As 1D read analysis was the focus of this thesis, only those reads were used.

Table 4.1: Used datasets

	Number of reads	Total bases [bp][1]	Whole genome size [bp]
E. Coli[2]	164471	1 481 687 490	4 639 675
lambda[3]	86	466 465	48 502

Figure 4.1 shows data present inside FAST5 file after being base calling by Metrichor. For each basecalled event, $model\_state$ field contains the most likely sequence of bases in the pore. How many bases have passed through the pore between two consecutive events is defined by the $move$ field.

4.1.1. File formats

Descriptions of various file formats used later in descriptions of preprocessing of training data and evaluation are given in this section.

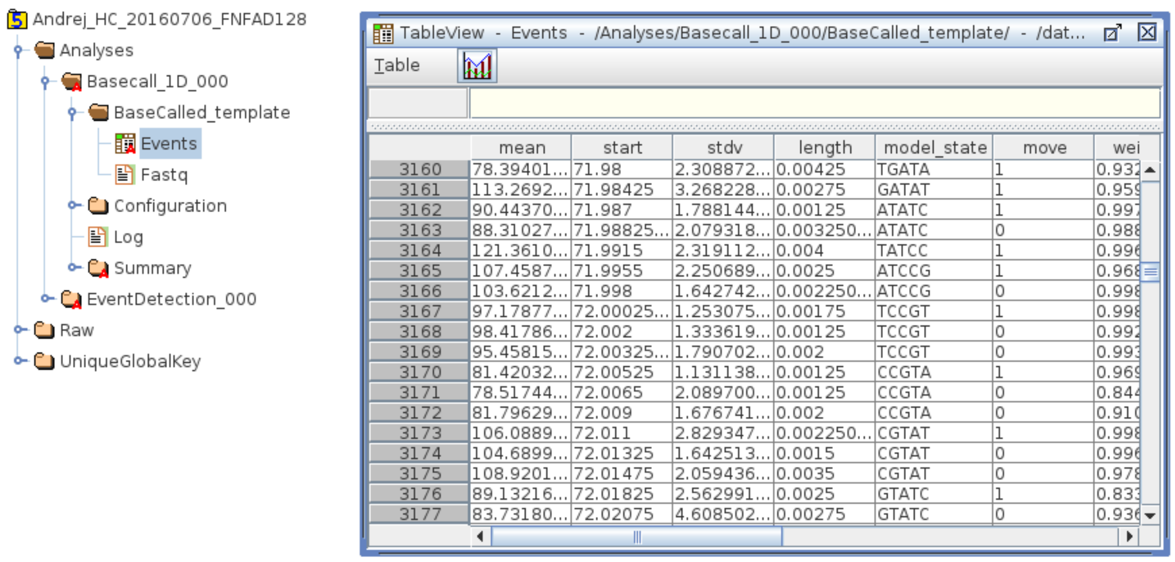
FASTA

FASTA is widely used file format for reference sequences. Usual file extensions are *.fasta* or *.fa*). Sequence representation consists of header line containing description starting with

[1]Total number of bases calle by Metrichor

[2]R9 sequencing data from `http://lab.loman.net/2016/07/30/nanopore-r9-data-release/`, reference taken from `https://www.ncbi.nlm.nih.gov/nuccore/48994873`

[3]Internal dataset, reference taken from `https://www.ncbi.nlm.nih.gov/nuccore/NC_001416.1`



	mean	start	stdv	length	model_state	move	wei
3160	78.39401...	71.98	2.308872...	0.00425	TGATA	1	0.932
3161	113.2692...	71.98425	3.268228...	0.00275	GATAT	1	0.959
3162	90.44370...	71.987	1.788144...	0.00125	ATATC	1	0.997
3163	88.31027...	71.98825...	2.079318...	0.003250...	ATATC	0	0.988
3164	121.3610...	71.9915	2.319112...	0.004	TATCC	1	0.996
3165	107.4587...	71.9955	2.250689...	0.0025	ATCCG	1	0.968
3166	103.6212...	71.998	1.642742...	0.002250...	ATCCG	0	0.998
3167	97.17877...	72.00025...	1.253075...	0.00175	TCCGT	1	0.998
3168	98.41786...	72.002	1.333619...	0.00125	TCCGT	0	0.992
3169	95.45815...	72.00325...	1.790702...	0.002	TCCGT	0	0.993
3170	81.42032...	72.00525	1.131138...	0.00125	CCGTA	1	0.969
3171	78.51744...	72.0065	2.089700...	0.00125	CCGTA	0	0.844
3172	81.79629...	72.009	1.676741...	0.002	CCGTA	0	0.910
3173	106.0889...	72.011	2.829347...	0.002250...	CGTAT	1	0.998
3174	104.6899...	72.01325	1.642513...	0.0015	CGTAT	0	0.996
3175	108.9201...	72.01475	2.059436...	0.0035	CGTAT	0	0.978
3176	89.13216...	72.01825	2.562991...	0.0025	GTATC	1	0.833
3177	83.73180...	72.02075	4.608502...	0.00275	GTATC	0	0.936

Figure 4.1: Basecall information produced by Metrichor show in HDFView

character >, followed by line(s) of sequence data represented by letters. Full file specification can be found at *NCBI* site[4]. Figure 4.2 shows example of sequence stored in FASTA file format.

```
>gi|48994873|gb|U00096.2|_E.coli_str._K-12_substr._MG1655,_complete_genome
AGCTTTTCATTCTGACTGCAACGGGCAATATGTCTCTGTGTGGATTAAAAAAAGAGTGTCTGATAGCAGC
TTCTGAACTGGTTACCTGCCGTGAGTAAATTAAAATTTTATTGACTTAGGTCACTAAATACTTTAACCAA
TATAGGCATAGCGCACAGACAGATAAAAATTACAGAGTACACAACATCCATGAAACGCATTAGCACCACC
ATTACCACCACCATCACCATTACCACAGGTAACGGTGCGGGCTGACGCGTACAGGAAACACAGAAAAAAG
CCCGCACCTGACAGTGCGGGCTTTTTTTTTCGACCAAAGGTAACGAGGTAACAACCATGCGAGTGTTGAA
GTTCGGCGGTACATCAGTGGCAAATGCAGAACGTTTTCTGCGTGTTGCCGATATTCTGGAAAGCAATGCC
AGGCAGGGGCAGGTGGCCACCGTCCTCTCTGCCCCCGCCAAAATCACCAACCACCTGGTGGCGATGATTG
AAAAAACCATTAGCGGCCAGGATGCTTTACCCAATATCAGCGATGCCGAACGTATTTTTGCCGAACTTTT
```

Figure 4.2: Example of FASTA file

FASTQ format

Reads are usually stored in FASTQ file format. Each read in the file is stored in following way:

1. character "@" followed by a sequence identifier and an optional description
2. sequence
3. character "+" character and is optionally followed by the same sequence identifier and description.

[4]http://www.ncbi.nlm.nih.gov/BLAST/blastcgihelp.shtml

4. quality score for each base

Quality is represented by ASCII printable characters where the character "!" represents the lowest quality while "~" is the highest. Figure 4.3 shows example of read stored in FASTQ file format.

```
@S1_2
GTGGGGGAAACGCAACTGGGAGGCCCTATTCCGGCGGCAGGG
+S1_2
-',+#%",#%(&,*.'-(!&!"'%$*$-)#$%*--)$!$&&.
```

Figure 4.3: Example of FASTQ file

SAM format

The Sequence Alignment/Map (SAM) format is a generic format for storing reads alignments against the reference sequences. It is a TAB-delimited text format consisting of header section and an alignments section. Detailed information about SAM specification can be found on *SAMTools* website[5]. Among other information, alignment start position is included, flag stating if read aligned as the template of as the reverse complement and CIGAR string describing the alignment. Figure 4.4 shows simple alignment and CIGAR string. There are several possible letters that can appear in CIGAR string but most importantly matches, mismatches, insertions and deletions are represented by letters "=", "X", "I" and "D".

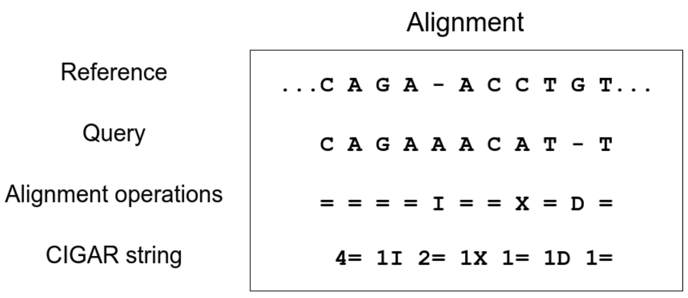


Figure 4.4: Example of simple alignment and CIGAR string

4.2. Data preprocess

To help training process, the raw signal is split into smaller blocks that are used as inputs. For each Metrichor basecalled event is easy to determine the block it falls into using *start*

[5]https://samtools.github.io/hts-specs/SAMv1.pdf

field. Using this information output given by Metrichor can be determined for each block. To correct errors produced by Metrichor and possibly increase the quality of data, each read is aligned to the reference. This is done using aligner GraphMap [24] that returns the best position in the genome, hopefully, the part of the genome from which read came from. Alignment part in the genome is used as a target. Using CIGAR string returned by aligner we can correct Metrichor data and get target output for each block. This process is shown in figure 4.5.

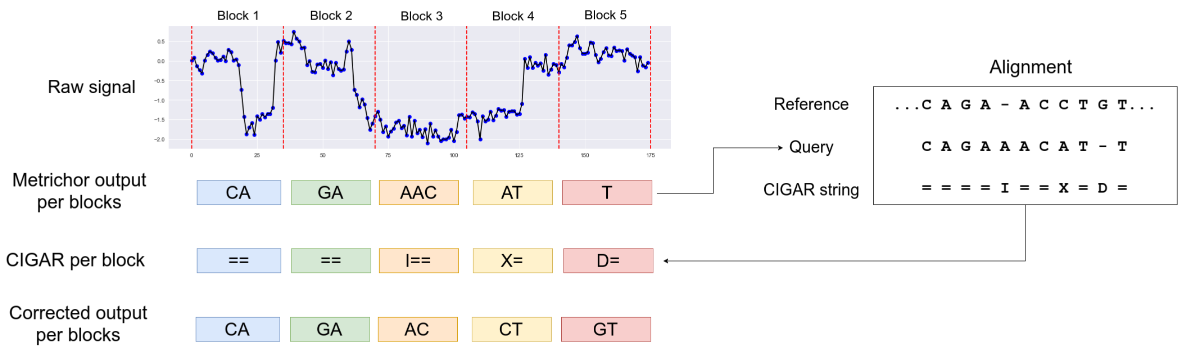


Figure 4.5: Dataset preparation

To eliminate the possibility of overfitting to the known reference, the model is trained and tested on reads from different organisms. Due to limited amount of public available raw nanopore sequence data, ecoli was *divided* into two regions. Reads were split into train and test portions, depending on which region of ecoli they align. If read aligns inside first 70% of the ecoli, it is placed into train set, and if it aligns to the second portion, it is placed into test set. Reads whose alignment overlaps train and test region are not used. Important to note that ecoli genome, and genomes of the majority of other bacteria, is cyclical, so reads with alignments that wrap over edges are also discarded. Total train set consist of over 110 thousand reads. Overview of the entire learning pipeline is shown in figure 4.6.

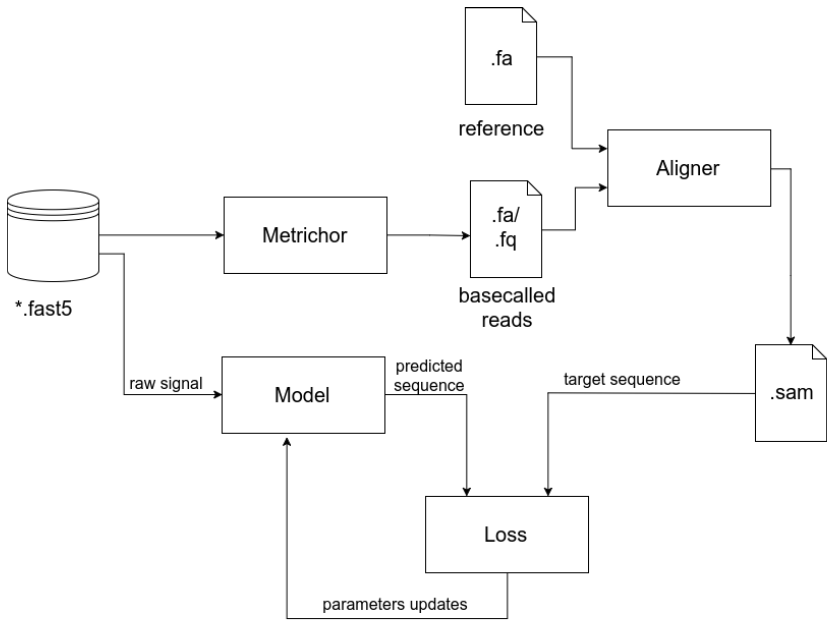


Figure 4.6: Overview of training pipeline

4.3. Deep Learning model

The final model is a residual neural network consisting of 72 residual blocks that are depicted in figure 4.7. The used model is a variant of architecture proposed in paper [25] with the difference of ELU being used as activation instead of ReLU as it is reported [26] to speeds up the learning process and improve accuracy as the depth increase.

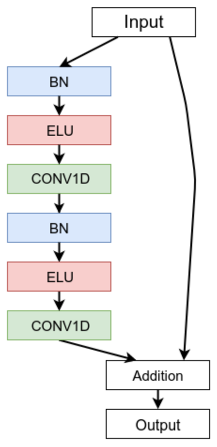


Figure 4.7: Used residual block

Each residual block contains two convolution layers making a total number of convolutions 144. Each convolutional layer in this models uses 64 kernels of 3. Because sequenced read is always shorter than the raw signal, pooling with kernel size two is used every 48 layers resulting in a reduction of dimensionality by factor 8. This is used reduce computation effort and to help training by reducing the number of required blank labels outputed by the network. This network has two million parameters that are learned during training.

Training the model is the minimization of previously described CTC loss. It was done using Adam [27], stochastic gradient descent algorithm based on an estimation of first and second-order moments. It is often used as it offers fast and stable convergence. Default parameters of Adam were used ($\beta_1 = 0.9$ and $\beta_2 = 0.999$). Initial learning rate was set to 1e-3 with exponential decay. Batch size was set to 8 mostly due to limited hardware resources. As noisy batches could potentially cause gradients to explode, gradient clipped to a range [-2, 2] was used. Learning curve is show on figure 4.8. Occasional spikes of training loss are explained by small batch size and presence of noise in the signal. Learning curve shows no sign of overfitting as modes shows similar performance on both train and validation sets.

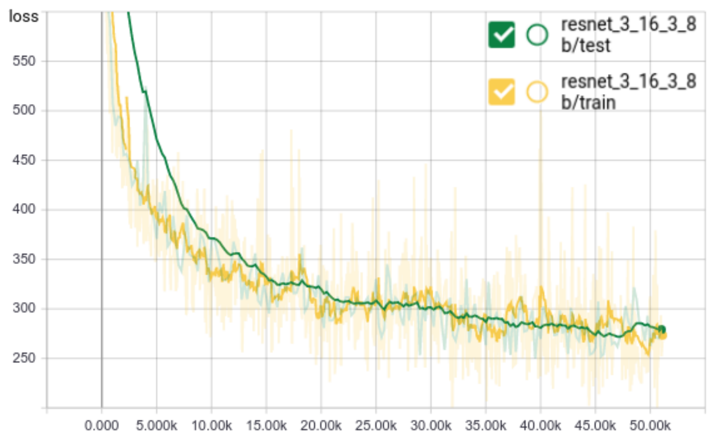


Figure 4.8: Learning curve in TensorBoard

Training process is implemented as *producers/consumer* pattern with communication done using FIFO[6] queue. Multiple producer threads, running on the CPU, load FAST5 and alimnment batch data, preprocess it and convert to required objects, while single GPU worker takes batch from the queue and computes forwards and backward passes on the network. This is done to reduce time between batches and maximize GPU utilization during training.

4.4. Technologies

Overall solution was implemented in Python programing language. Described model is implemented using TensorFlow. It is an open source software library for numerical computation using data flow graphs developed by Google. TensorFlow, even though is considered low-level framework, offers implementations of many higher level concepts (layers, losses, and

[6]FIFO is an acronym for first in, first out

optimizers) which makes it great for prototyping while keeping it modular and extensible for highly specific tasks as well.

TensorFlow offers efficient GPU implementations of various layers and losses but as of version 1.2 still lacks GPU implementation of used CTC loss, so WARP-CTC[7] was used. It offers both GPU and CPU implementations as well as bindings for TensorFlow.

For alignment tasks, developed tool offers support for GraphMap and BWA but can easily be extended with support any other aligner that outputs results in SAM file format.

SAMTools[8] and Python bindings PySam[9] were used for conversions between various file formats used in Bioinformatics.

Docker was used for automating the deployment on different machines. It helps to resolve problem know as *dependency hell*[10] keeping all dependencies in single container thus eliminating possible conflict between packages on host OS. Nvidia Docker[11] was used for GPU support inside docker containers.

Training and all evaluations were done on the server with *Intel(R) Xeon(R) E5-2640 CPU*, 600 GB of RAM and *NVIDIA TITAN X Black* with 6GB of GDDR5 memory and 2880 CUDA cores.

All developed and used code, including utility scripts is publicly available on *GitHub*[12] under the *MIT Licence*.

[7]`https://github.com/baidu-research/warp-ctc`

[8]`http://www.htslib.org/`

[9]`https://github.com/pysam-developers/pysam`

[10]`https://en.wikipedia.org/wiki/Dependency_hell`

[11]`https://github.com/NVIDIA/nvidia-docker`

[12]`https://github.com/mratkovic/masters-thesis`

5. Results

Developed tool was compared with other available basecallers that support R9 chemistry. This includes third-party basically DeepNano and official basecallers by Oxford Nanopore (cloud-based Metrichor and Nanonet). The fact that ground truth is not known makes evaluation difficult. Different methods for evaluation were used to get clearer information about each basecaller. In all tables and figures, developed model is addressed as *resdeep* simply due to the fact it is a deep residual neural network.

5.1. Error rates per read

Basecalled reads are aligned to the reference using GraphMap and alignments are analyzed. If the whole sequencing is done correctly and quality basecaller is used, all reads should align to the reference. Mismatches, insertions, and deletions, in that case, should be due to limitations of sequencing technology and noise in the signal.

A portion of the read length that aligns as correctly is called match_rate. Same goes for mismatches and insertions. Sum of all matches, mismatches, and insertions is equal to the reads length 5.1.

$$read\_len = n\_matches + n\_mismatches + n\_insertions \tag{5.1}$$

$$match\_rate = \frac{n\_matches}{read\_length} \tag{5.2}$$

$$missmatch\_rate = \frac{n\_mismatches}{read\_length} \tag{5.3}$$

$$insertion\_rate = \frac{n\_insertions}{read\_length} \tag{5.4}$$

$$match\_rate + snp\_rate + insertion\_rate = 1 \tag{5.5}$$

Deletion rate is defined as a total number of deletions in the alignment over the length of the aligned read.

$$deletion\_rate = \frac{n\_deletion}{read\_length} \tag{5.6}$$

To get reliable results, this is done on both ecoli test dataset and lambda dataset. For each basecaller, median, mean and variance of all aligned reads are calculated. To summarize the results, the median is used as a single value as it is robust and even more informative in the case of skewed distributions like these. Results expressed as percentages are shown in the table 5.1 for ecoli and table 5.3 for lambda dataset.

Developed tool shows promising results by having better match rate and smaller mismatch rate than the others. Both datasets show all basecallers being biased towards deletions than insertions but this possibly is the bias of the used aligner. To eliminate that possibility, tests were repeated using *BWA mem* aligner[1] with almost identical results. Results using BWA aligner are shown in tables 5.2 and 5.4. Results are consistent on both datasets using both aligners.

Table 5.1: Alignment specifications of Ecoli R9 basecalled reads using GraphMap

	Match % (median)	**Mismatch % (median)**	**Insertion % (median)**	**Deletion % (median)**
DeepNano	90.254762	6.452852	**3.274420**	11.829965
Metrichor	90.560455	5.688105	3.660381	8.328271
Nanonet	90.607674	5.608912	3.652791	8.299046
resdeep	**91.408591**	**5.019141**	3.477739	**7.471608**

Table 5.2: Alignment specifications of Ecoli R9 basecalled reads using BWA mem

	Match % (median)	**Mismatch % (median)**	**Insertion % (median)**	**Deletion % (median)**
DeepNano	90.254762	6.452852	3.274420	11.829965
Metrichor	90.595441	6.869543	2.531646	7.567381
Nanonet	90.988989	6.674760	**2.348552**	7.698530
resdeep	**91.470588**	**5.929204**	2.477283	**6.970362**

[1]https://github.com/lh3/bwa

Table 5.3: Alignment specifications of Ecoli R9 basecalled reads using GraphMap

	Match % (median)	**Mismatch % (median)**	**Insertion % (median)**	**Deletion % (median)**
DeepNano	86.997687	9.623494	3.442490	16.052830
Metrichor	87.714988	7.835052	4.093851	**10.757491**
Nanonet	88.415611	8.178372	3.629653	11.793022
resdeep	**89.694482**	**7.238095**	**3.078796**	13.450292

Table 5.4: Alignment specifications of lambda R9 basecalled reads using BWA mem

	Match % (median)	**Mismatch % (median)**	**Insertion % (median)**	**Deletion % (median)**
DeepNano	86.625973	11.288361	2.098225	14.648308
Metrichor	87.294093	10.109186	2.376476	**9.645323**
Nanonet	87.767037	10.017598	2.354248	10.597232
resdeep	**89.049870**	**9.480883**	**1.615188**	12.962441

Distribution of these percentages per reads are shown using histogram plot on figure 5.3 and the KDE (*kernel density estimate*) plot on igures 5.2 and 5.4. Like the histogram, the KDE plot encodes the density of observations, but curve approximation is used instead of bins resulting in less cluttered comparison. It is important to note that lambda is small dataset and more samples are needed to get a better approximation of distribution.

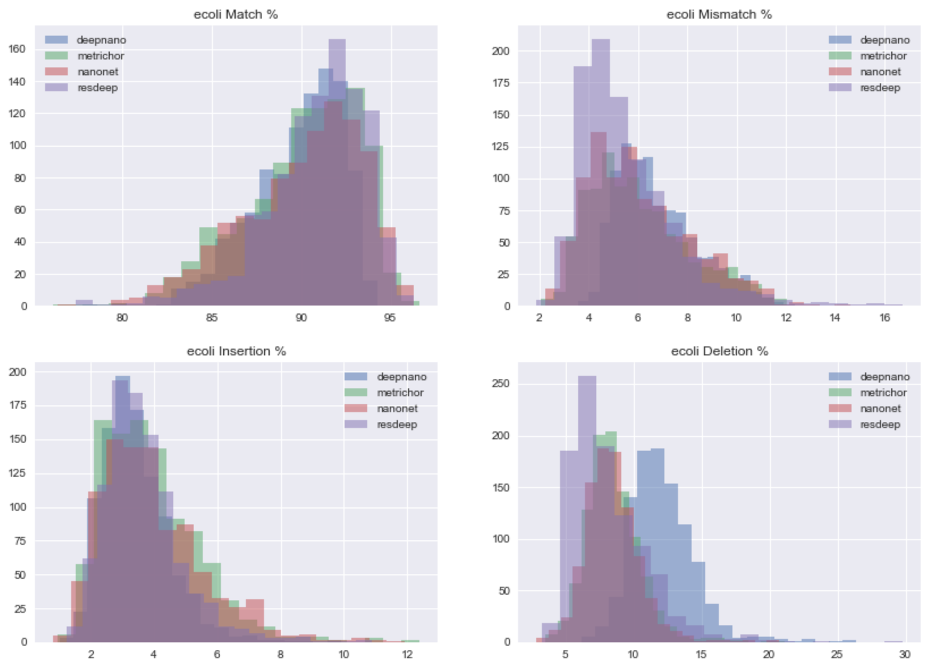


Figure 5.1: Histogram showing distribution of percentage of alimnment operations for ecoli

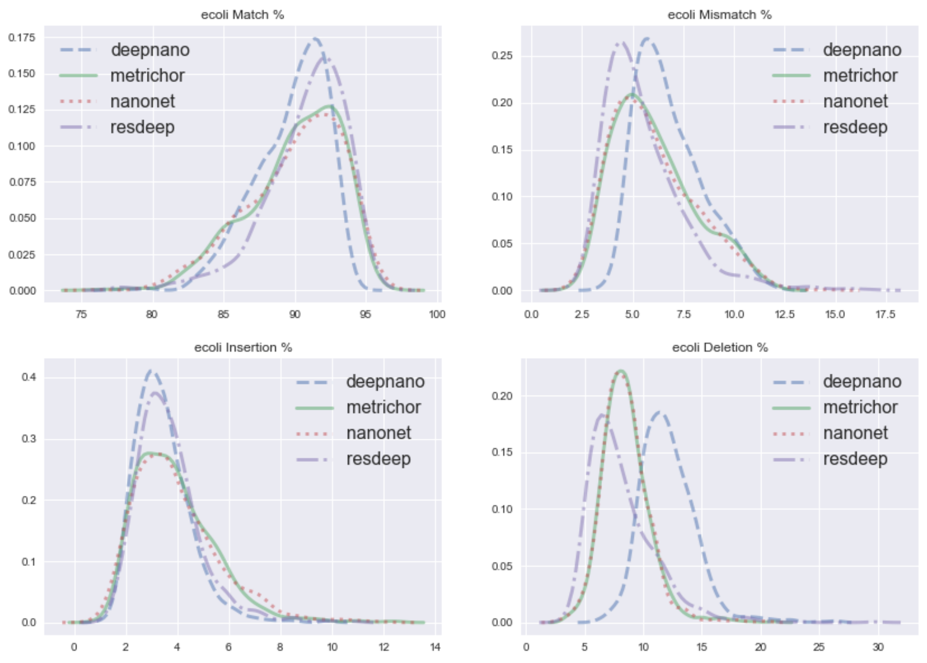


Figure 5.2: KDE plot for distribution of percentage of alimnment operations for ecoli

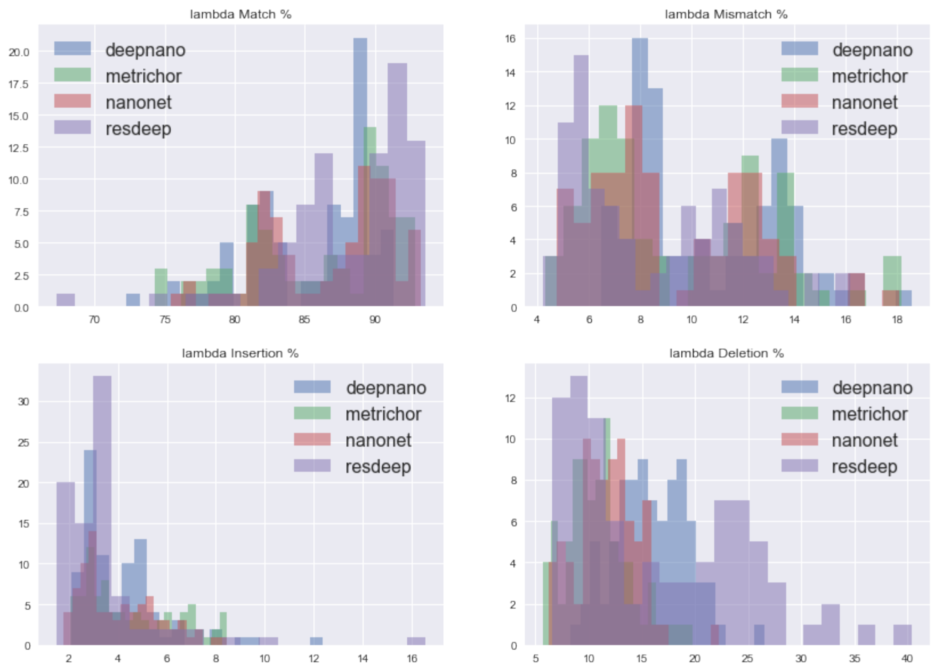


Figure 5.3: Histogram showing distribution of percentage of alimnment operations for lambda

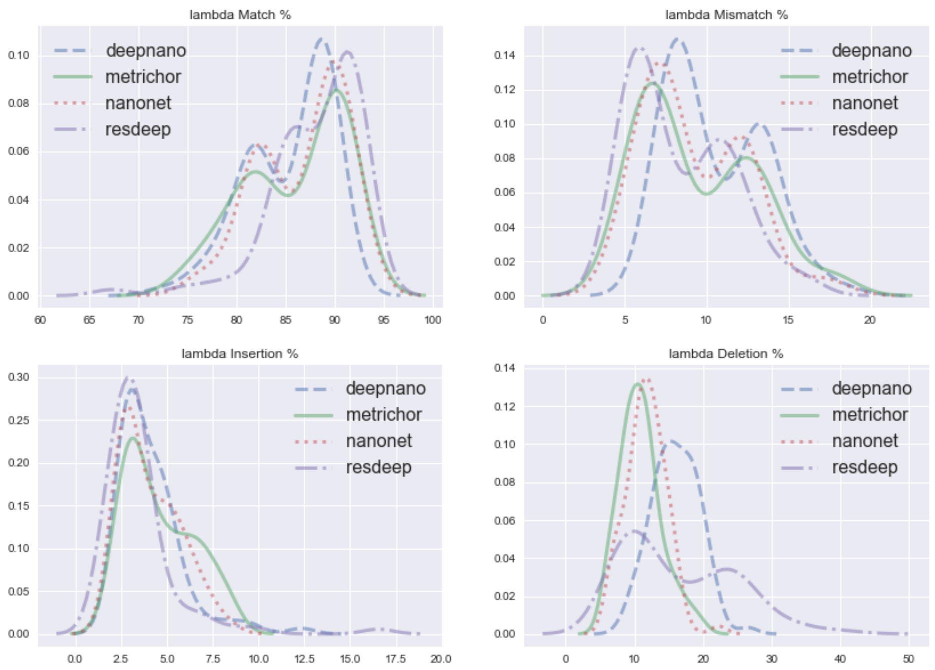


Figure 5.4: KDE plot for distribution of percentage of alimnment operations for lambda

The histogram on figure 5.5 shows how matches, mismatches, insertions, and deletions are distributed across the read. It is shown that mismatches and insertion occur more frequently at the beginnings and the ends of the reads. This is not only the case for the developed basecaller, but all other show the same property. This could be due to lack of context information from both sides when edges are base called.

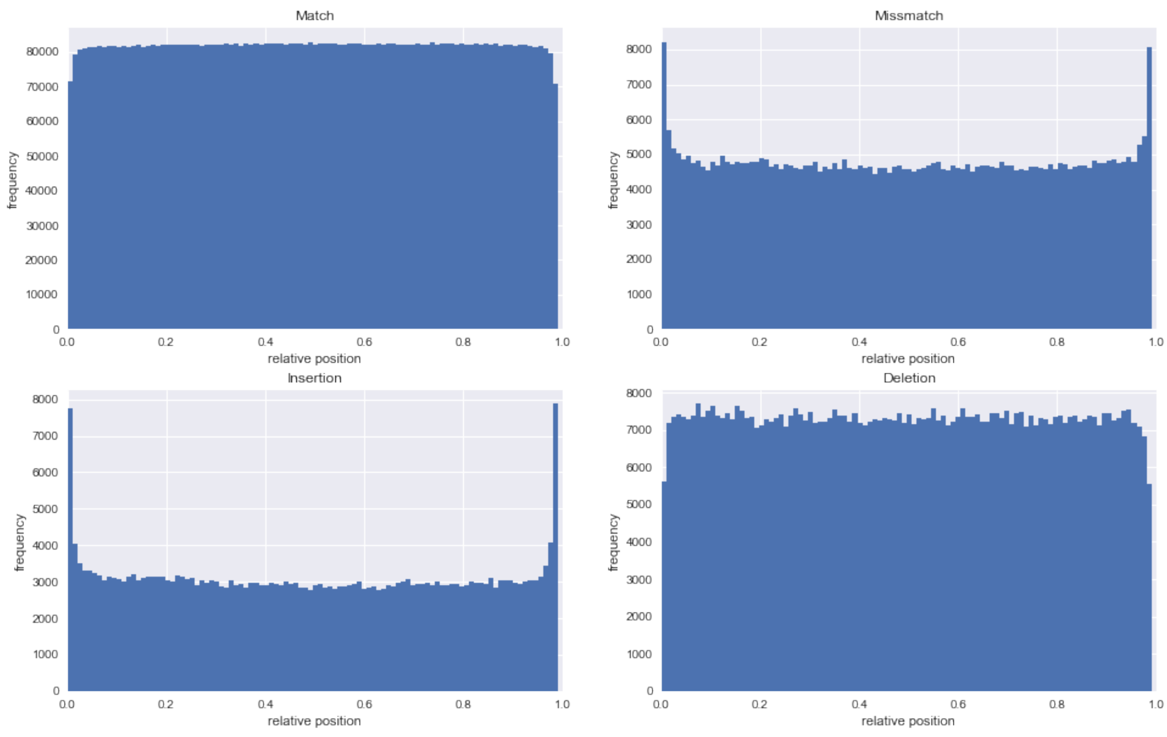


Figure 5.5: Histogram of alignment operations over relative position inside of read

5.2. Consensus

Described error rates and match rates calculated from alignments of individual reads to the reference could be misleading as it is simple to produce a naive model that obtains excellent results. Let us consider a model that basecalls single base "A" for every input signal. Aligning "A" to the reference (supposing that the aligner does not discard short and highly ambiguous reads) will always return perfect alignment as nucleotide "A" is certainly present in the reference. This model would have a perfect match rate with non-existent mismatches, insertions, and deletions. Two other approaches are used, in addition to the first, to give a more rigorous comparison.

The idea behind other approaches is checking if the reconstruction of the reference is possible from the basecalled reads and closely does it match the original reference.

5.2.1. Consensus from pileup

Instead of going through the whole assembly process, as we know the reference genome of the data used in these tests, we simply align all the reads to the genome, stack them on top of each other forming pileup of read bases. Using majority vote, dominant bases are called on each position. The resulting sequence is called consensus. When calling consensus for deletions, there has to be a majority of deletions of the same length. Calling insertions has the additional condition, the majority has to agree on both length and the bases of insertion. Figure 5.6 shows how consensus is called from pileup created from aligned reads. Pileup is stored in mpileup format.

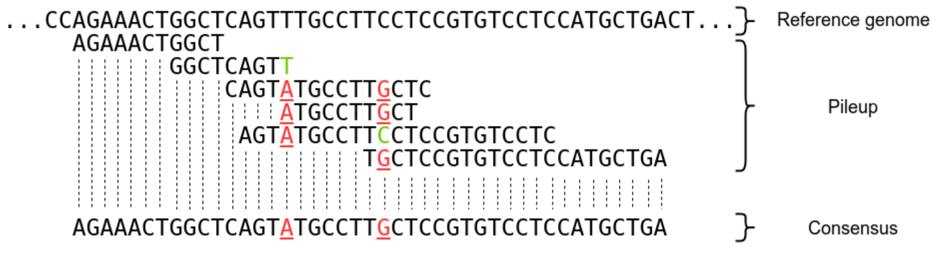


Figure 5.6: Consensus from pileup

Similarly, as before, match rate, mismatch rate, insertion and deletion rates are calculated but this time for whole consensus sequence. In this context, mismatches are called *single nucleotide polymorphisms* (snp).

$$match\_rate = \frac{n\_correct\_bases}{consensus\_length} \tag{5.7}$$

$$snp\_rate = \frac{n\_snp}{consensus\_length} \tag{5.8}$$

$$insertion\_rate = \frac{n\_insertions}{consensus\_length} \tag{5.9}$$

$$match\_rate + snp\_rate + insertion\_rate = 1 \tag{5.10}$$

$$deletion\_rate = \frac{n\_deletion}{consensus\_length} \tag{5.11}$$

Results are shown in the tables 5.5 for ecoli and 5.6 for lambda. Developed model shows results comparable with Metrichor in all aspects (matches, mismatches, insertions, deletions and the total length of the consensus sequence) for lambda and show even better results from ecoli.

All models show a slight bias towards deletions than insertions, but this may be the limitation of technology as it has been reported that deletion and mismatch rates for nanopore data are ordinarily higher than insertion rates [24].

Table 5.5: Consensus specifications of Ecoli R9 basecalled reads

	Total called [bp]	**Correctly called [bp]**	**Match %**	**Snp %**	**Insertion %**	**Deletion %**
DeepNano	1510244.0	1493242.0	98.8742	1.0044	0.1214	0.9041
Metrichor	1515893.0	1502588.0	99.1223	0.7464	0.1313	0.6300
Nanonet	1414237.0	1385515.0	97.9691	1.5700	0.4609	1.5158
resdeep	1517828.0	1506233.0	**99.2361**	**0.6474**	**0.1165**	**0.5510**

Table 5.6: Consensus specifications of lambda R9 basecalled reads

	Total called [bp]	**Correctly called [bp]**	**Match %**	**Snp %**	**Insertion %**	**Deletion %**
DeepNano	48342.0	48025.0	99.3443	0.6433	**0.0124**	0.2648
Metrichor	48469.0	48257.0	**99.5626**	**0.4188**	0.0186	**0.1465**
Nanonet	48438.0	48168.0	99.4426	0.5409	0.0165	0.1961
resdeep	48385.0	48163.0	99.5412	0.4402	0.0186	0.1976

5.2.2. Assembly

In this step consensus sequence is not calculated from pileup, but using *de novo* genome assembly. For this task, fast and accurate *de novo* genome assembler *ra*[2] [28] was used and obtained consensus sequence is compared to the reference using *dnadiff* present in the *Mumer*[3]. The length of the reference, consensus sequence, number of contigs and percentages of aligned bases from the reference to the query and vice versa are shown in the table **??**. Average identity summarizes how closely does the assebled sequence match the reference. This is run on full ecoli sequence run for 1D template reads (~160k reads), for both developed tools and Metrichor. Developed tool has shown a small increase in quality of the assembled sequence over Metrichor by offering longer consensus, higher identity percentage and overall smaller edit distance[4].

[2]`https://github.com/rvaser/ra`

[3]`https://github.com/garviz/MUMmer`

[4]Calculated using `https://github.com/isovic/racon/blob/master/scripts/edcontigs.py`

Table 5.7: Assembly and consensus results for ecoli

	Metrichor	resdeep
Ref. genome size (bp)	4639675	4639675
Total bases (bp)	4604806	**4614354**
Contigs [#]	1	1
Aln. bases ref. (bp)	4639641(100.00%)	4639612(100.00%)
Aln. bases query (bp)	4604787(100.00%)	4614351(100.00%)
Avg. Identity	98.76	**99.06**
Edit distance	60418	**46686**

5.3. Read lengths

The lengths of basecalled reads for each tool are interesting to analyze. Developed tool output reads of lengths similar to Metrichor while other tools, such as Nanonet, for instance, basecall reads that are significantly shorter. Detailed analysis of read length distributions is shown using KDE plots on figure 5.7 for both lambda and ecoli.

Table 5.8: Ecoli R9 basecalled read lengths in base pairs

	median	**mean**	**std**
DeepNano	5526.5	8126.694000	7406.554786
Metrichor	5809.5	8933.275000	9189.709720
Nanonet	3286.5	4874.406582	4803.182344
resdeep	5784.0	8990.988989	9297.972688

Table 5.9: Lambda R9 basecalled read lengths in base pairs

	median	mean	std
DeepNano	4740.0	4664.750000	2628.512543
Metrichor	5491.0	5482.952941	2748.446253
Nanonet	4931.5	4925.804878	2739.987512
resdeep	5229.0	5138.764706	2605.958080

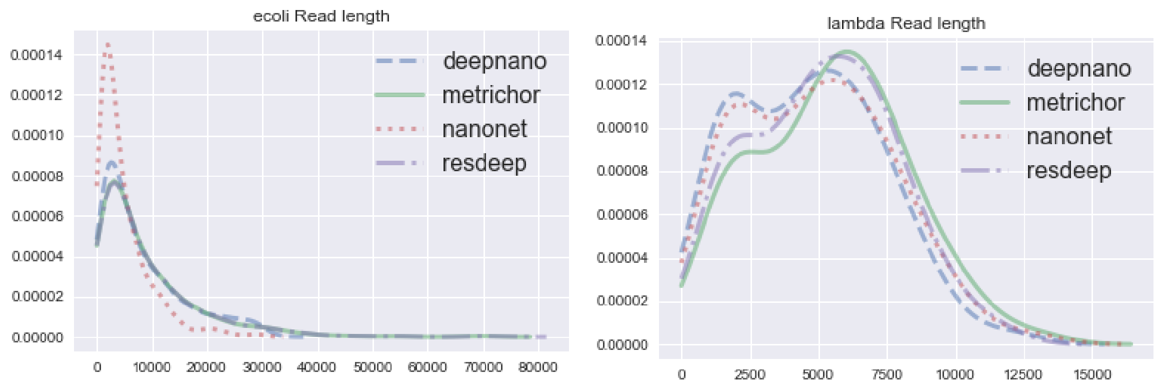


Figure 5.7: Overview of evaluation pipeline

5.4. Base calling speeds

Table 5.10 shows base calling speeds of all tools. Metrichor is present in the table as it is cloud-based service and real execution time is unknown. Tests for all other basecallers were run under same conditions on hardware described in section 4.4. All tested tools offer parallelized base calling so a number of jobs(threads) during testing was set to 32. Both the developed model and Nanonet provide GPU support for base calling, while DeepNano is limited for CPU only.

Developed tool has shown faster base calling times in both CPU and GPU group even though it is very deep network of 144 layers. This shows efficiency of CNNs compared with RNNs. Differences in base calling speeds for ecoli and lambda datasets do exist, but they are not substantial and may be contributed to the different length of the reads and the different total number of reads in datasets.

Table 5.10: Base calling speeds measured in *base pairs per second*

	ecoli (bp/s)	**lambda (bp/s)**
resdeep (CPU)	**1174.28**	**1363.340**
Nanonet (CPU)	856.01	897.499
DeepNano (CPU)	626.99	692.370
resdeep (GPU)	**6571.76**	**6140.300**
Nanonet (GPU)	3828.39	3787.510

5.5. Evaluation pipeline

Entire evaluation pipeline is shown in figure 5.8. GraphMap was used for alignment purposes and SAMTools for conversion between SAM and its binary variant BAM as well as a generation of mpileup.

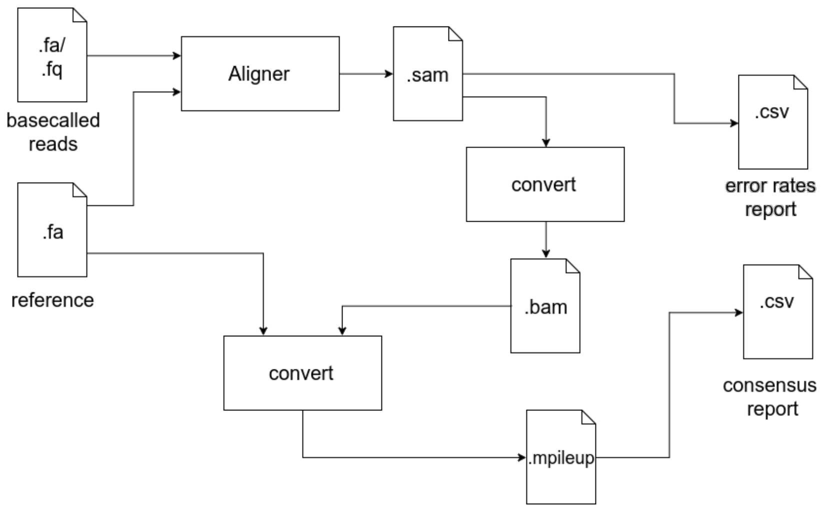


Figure 5.8: Overview of evaluation pipeline

6. Conclusion

The goal of this thesis was to show that usage of convolution neural networks can potentially replace RNNs in the analysis of sequencing data by offering better results as well as faster execution times.

On all tests, proposed model has shown improvement in the accuracy of basecalled data as well as faster basecalling speeds of over both official (Matrichor and experimental Nanonet) and third-party DeepNano while having. To provide definite proof of this claim, the model needs to be tested on larger datasets from different sequencing runs for multiple organisms.

All test are done on data for R9 chemistry, but the developed code could easily be adjusted and trained on R9.4 and newest R9.5 data when it becomes publicly available. It would be interesting to see how well this approach works compare to basecaller Scrappie by Oxford Nanopore that addresses detection of homopolymers.

Currently, without support for newer sequencing data, this model has limited application. It can be used as a demonstration of a different approach to base calling which yields promising results. As newer versions of basecallers by Oxford Nanopore do not offer any support for data sequenced with previous version of chemistries, this tool can be used to re-basecall that data and improvement of the quality of reads.

Future work includes experiments with recently proposed *scaled exponential linear units* (SELU) [29] that eliminates the need for normalization techniques such as used batch normalization. Possible improvements of the model include the combination of convolutions and attention mechanism proposed just recently in the paper [16] showing excellent results in both speed and accuracy, for tasks of language translation.

BIBLIOGRAPHY

[1] Yann LeCun and Yoshua Bengio. The handbook of brain theory and neural networks. chapter Convolutional Networks for Images, Speech, and Time Series, pages 255–258. MIT Press, Cambridge, MA, USA, 1998. ISBN 0-262-51102-9. URL `http://dl.acm.org/citation.cfm?id=303568.303704`.

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. URL `https://goo.gl/UpFBv8`.

[3] Mirjana Domazet-Lošo Mile Šikić. *Bioinformatika*. Bioinformatics - course matherials, Faculty of Electrical Engineering and Computing, University of Zagreb, 2013.

[4] Erik Pettersson, Joakim Lundeberg, and Afshin Ahmadian. Generations of sequencing technologies. *Genomics*, 93(2):105–111, feb 2009. doi: 10.1016/j.ygeno.2008.10.003. URL `https://doi.org/10.1016/j.ygeno.2008.10.003`.

[5] Miten Jain, Sergey Koren, Josh Quick, Arthur C Rand, Thomas A Sasani, John R Tyson, Andrew D Beggs, Alexander T Dilthey, Ian T Fiddes, Sunir Malla, Hannah Marriott, Karen H Miga, Tom Nieto, Justin O'Grady, Hugh E Olsen, Brent S Pedersen, Arang Rhie, Hollian Richardson, Aaron Quinlan, Terrance P Snutch, Louise Tee, Benedict Paten, Adam M. Phillippy, Jared T Simpson, Nicholas James Loman, and Matthew Loose. Nanopore sequencing and assembly of a human genome with ultra-long reads. *bioRxiv*, 2017. doi: 10.1101/128835. URL `http://biorxiv.org/content/early/2017/04/20/128835`.

[6] Nick Loman. *Nanopore R9 rapid run data release*, . URL `http://lab.loman.net/2016/07/30/nanopore-r9-data-release/`. [Online; posted 30-July-2016].

[7] Nick Loman. *Thar she blows! Ultra long read method for nanopore sequencing*, . URL `http://lab.loman.net/2017/03/09/ultrareads-for-nanopore/`. [Online; posted 9-March-2017].

[8] C Brown. *YouTube Technology Focus Live Stream No thanks, I've already got one*. URL `https://www.youtube.com/watch?v=nizGyutn6v4`. [Online; posted 8-March-2016].

[9] Sara Goodwin, John D. McPherson, and W. Richard McCombie. Coming of age: ten years of next-generation sequencing technologies. *Nat Rev Genet*, 17(6):333–351, Jun 2016. ISSN 1471-0056. URL `http://dx.doi.org/10.1038/nrg.2016.49`. Review.

[10] Nanopore Community. *Basecalling overview*. URL `https://community.nanoporetech.com/technical_documents/data-analysis/v/datd_5000_v1_reve_22aug2016/basecalling-overvi`. [Accessed; 12-July-2017].

[11] Matei David, Lewis Jonathan Dursi, Delia Yao, Paul C Boutros, and Jared T Simpson. Nanocall: An open source basecaller for oxford nanopore sequencing data. *bioRxiv*, 2016. doi: 10.1101/046086. URL `http://biorxiv.org/content/early/2016/03/28/046086`.

[12] Vladimír Boža, Broňa Brejová, and Tomáš Vinař. DeepNano: Deep recurrent neural networks for base calling in MinION nanopore reads. *PLOS ONE*, 12(6):e0178751, jun 2017. doi: 10.1371/journal.pone.0178751. URL `https://doi.org/10.1371/journal.pone.0178751`.

[13] Denny Britz. *Recurrent Neural Networks Tutorial - Backpropagation Through Time and Vanishing Gradients*. URL `https://goo.gl/Qkv9gC`. [Online; posted 15-September-2015].

[14] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[15] Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, Aaron van den Oord, Alex Graves, and Koray Kavukcuoglu. Neural machine translation in linear time, 2016.

[16] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. Convolutional sequence to sequence learning, 2017.

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, 2015.

[18] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus), 2015.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[20] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pages 369–376, New York, NY, USA, 2006. ACM. ISBN 1-59593-383-2. doi: 10.1145/1143844.1143891. URL `http://doi.acm.org/10.1145/1143844.1143891`.

[21] Alex Graves and Navdeep Jaitly. Towards end-to-end speech recognition with recurrent neural networks. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 1764–1772, Bejing, China, 22–24 Jun 2014. PMLR. URL `http://proceedings.mlr.press/v32/graves14.html`.

[22] Andrew Gibiansky. *Speech Recognition with Neural Networks*. URL `http://andrew.gibiansky.com/blog/machine-learning/speech-recognition-neural-networks/`. [Online; posted 23-April-2014].

[23] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[24] Ivan Sović, Mile Šikić, Andreas Wilm, Shannon Nicole Fenlon, Swaine Chen, and Niranjan Nagarajan. Fast and sensitive mapping of nanopore sequencing reads with GraphMap. *Nature Communications*, 7:11307, apr 2016. doi: 10.1038/ncomms11307. URL `https://doi.org/10.1038/ncomms11307`.

[25] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks, 2016.

[26] Anish Shah, Eashan Kadam, Hena Shah, Sameer Shinde, and Sandip Shingade. Deep residual networks with exponential linear unit. 2016. doi: 10.1145/2983402.2983406.

[27] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[28] Robert Vaser, Ivan Sovic, Niranjan Nagarajan, and Mile Sikic. Fast and accurate de novo genome assembly from long uncorrected reads. *bioRxiv*, 2016. doi: 10.1101/068122. URL `http://biorxiv.org/content/early/2016/08/05/068122`.

[29] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks, 2017.

Deep Learning Model for Base Calling of MinION Nanopore Reads

Abstract

MinION by Oxford Nanopore Technologie is affordable and portable sequencing device suitable for various applications. The device produces very long reads, however, it suffers from high sequencing error rate. The goal of this thesis is to show that the reported accuracy of the sequencing data is not only limited by sequencing technology, but also by the current software tools used for base calling and can be further improved by using different deep learning concepts. Approach for base calling of raw data using convolutional neural networks is proposed as an alternative to recurrent neural networks used by other basecallers offering improvements both in speed and accuracy. A detailed comparison of the developed tool with the existing tools for base calling R9 data is given.

Keywords: base calling, Oxford Nanopore Technologies, MinION, deep learning, seq2seq, convolutional neural network, residual network, CTC loss

Model dubokog učenja za određivanje očitanih baza dobivenih uređajem za sekvenciranje MinION

Sažetak

Uređaji za sekvenciranje MinION tvrtke Oxford Nanopore Technologies su pristupačni i prenosivi što ih čini pogodnim za razne primjene. Uređaj omogućuje sekvenciranje očitanja velikih duljina ali većeg postotka greške u odnosu na prethodne tehnologije. Cilj ovog diplomskog rada je pokazati da trenutna pogreška nije uzrokovana isključivo metodom sekvenciranja, već i programskim alatima koji se koriste za očitavanje baza te je pogrešku moguće smanjiti korištenjem metoda dubokog učenja.

Predstavljen je novi alat za očitavanje baza temeljen na konvolucijskim neuronski mrežama koji pruža napredak u preciznost i brzinu u odnosu na trenutno korištene rekurzivne neuronske mrežama. U radu je dana detaljna analiza razvijenog alata i usporedba s postojećim rješenjima za očitvanje baza.

Ključne riječi: određivanje baza, Oxford Nanopore Technologies, MinION, duboko učenje, prevođenje, konvolucijske neuronske mreže, rezidualne mreže, CTC gubitak