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`.

Thanks ...

CONTENTS

List of Figures

List of Tables

1. Introduction

In recent years, deep learning and usage of deep neural networks have significantly improved the state-of-the-art in many application domains such as computer vision, speech recognition, and natural language processing[8][7]. In this thesis, we present application of deep learning in the fields of Biology and Bioinformatics for analysis of DNA sequencing data.

DNA is a molecule that makes up the genetic material of a cell responsible for carrying the information an organism needs to survive, grow and reproduce. It is a long polymer of simple units called nucleotides attached together to form two long strands that spiral to create a structure called a double helix. The order of these bases is what determines DNA's instructions, or genetic code.

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

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

1.1. Objectives

Goal of this thesis is to show that the accuracy of base calling is much dependent on the underlying software and can be improved using machine learning methods. Novel approach for base calling of raw data using convolutional neural networks is presented.

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 describe used deep learning concepts in detail used later on in later chapters.

Chapter 4 goes into implementation details, training of the deep learning model and explains methods used to evaluate obtained results.

Chapter 5 consists of the results of testing performed on different datasets as well as comparison with state-of-the-art basecallers.

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

2. Background

2.1. Sequencing

Sequencing the entire genome of an organism is a difficult problem due to limitations of technology. All sequencing technologies to date have constraints on the length of the strand they can process. These lengths are much smaller than the genome for a majority of organisms, therefore, whole genome shotgun sequencing approach is used. In this approach, multiple copies of the genome are broken up 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 accuracy of the reads produced by the sequencer. Short reads make resolving repetitive regions practically impossible.

Figure 2.1 depicts process of sequencing.

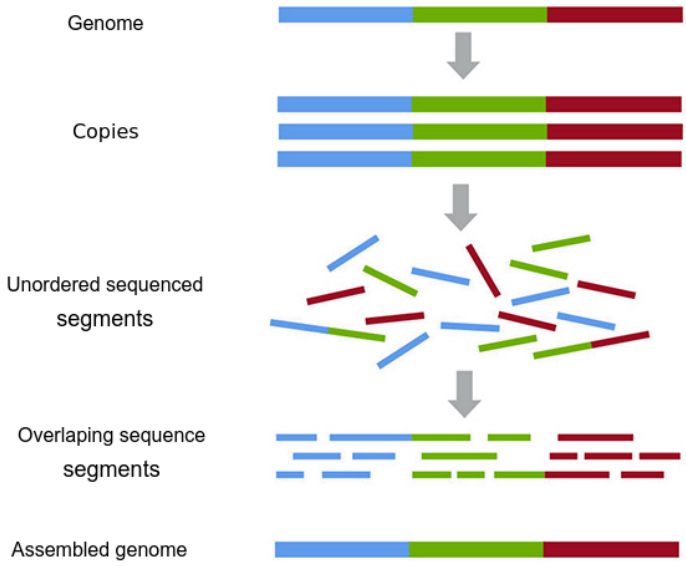

Figure 2.1: Depiction of the sequencing process

Development of sequencing started with work of Frederick Sanger[9][10]. In 1977, he developed the first sequencing method which allowed read lengths up to 1000 bases with

very high accuracy (99.9%) at a cost of 1$ per 1000 bases[mile_skripta]. 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. The need for technologies that would produce 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 a cost of accuracy. Error Rates of PacBio devices are 10-15%.

Cost makes the main obstacle stopping widespread genome sequencing. A significant breakthrough occurred in May 2015 with a release of MinION sequencer by Oxford Nanopore making sequencing drastically less expensive and even portable.

2.2. Oxford Nanopore MinION

The MinION device by Oxford Nanopore Technologies (referenca) is the first portable DNA sequencing device.

It's small weight, low cost, and long read length combined with high-throughput and decent accuracy yield promising results in various applications such as monitoring infectious disease outbreaks [2][3], characterizing structural variants in cancer[4], full human genome assembly [5] what could potentially lead to personalized genomic medicine.

Although MinION is able to produce long reads, usually tens of thousand base pairs (with reported reads above 100 thousand pairs[loman]), they have a high sequencing error rate depending of the pore model used. Older pore models, R7.3 chemistry.....

As its name says, nano-scale pores are used to sequence DNA. An electrical potential is applied over an insulating membrane in which a pore is inserted. As the DNA passes through the pore, the sensor detects changes in ionic current caused by differences in the shifting nucleotide sequences occupying the pore. Figure bla shows change of ionic current as DNA strain being pulled through a nanopore.

2.2.1. Technology

As its name says, nano-scale pores are used to sequence DNA. An electrical potential is applied over an insulating membrane in which a pore is inserted. As the DNA passes through the pore, the sensor detects changes in ionic current caused by differences in the shifting nucleotide sequences occupying the pore. Figure bla shows change of ionic current as DNA strain being pulled through a nanopore.

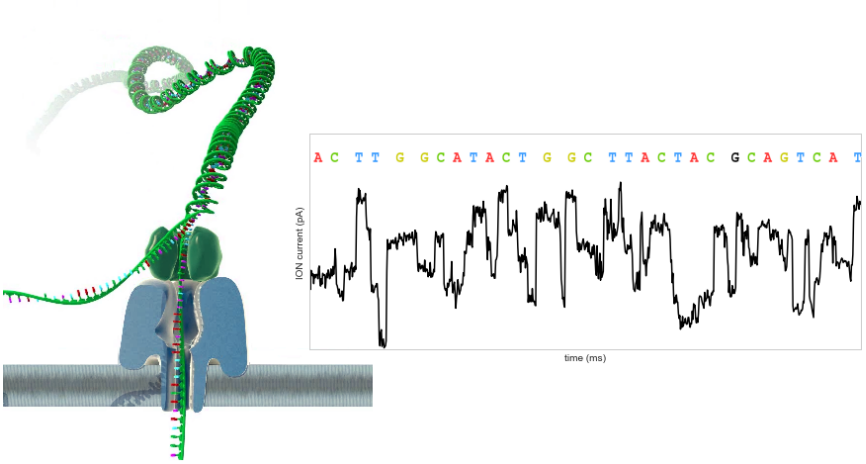

Figure 2.2: DNA strain being pulled through a nanopore[ref na video])

2.3. Existing basecallers

Output of the sequencer is fast5 file. Various fields, including signal. Picture bla shows structure of fast5 file. Analysis fields are added if file is passed to MinKnow for signal segmentation.

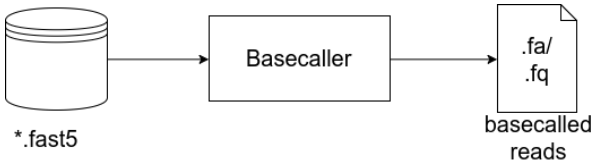

Figure 2.3: Basecalling

2.3.1. Official

Albacore is basecaller by Oxford Nanopore Technologies ready for production and activly supported. The source code of Albacore was not provided and is only available as a binary through the ONT Developer Channel to users who have signed the Developer terms and conditions.

Nanonet uses the same neural network that is used in Albacore but it is continually under development. As such, it does not contain production code features such as error handling or logging. It uses CURRENNT library for running neural networks.

Scrappie is an other basecaller by Oxford Nanopore Technologies. Simillar to NanoNet, it is platform for ongoing development. Scrappie was the first basecaller reported to specifically address homopolymer basecalling. It became publicly available just recently in June, 2017.

2.3.2. Third-party

Nanocall[3] was the first third-party open source basecaller for nanopore data. It uses HMM approch like the original R7 Metrichor. Metrichor. It uses the segmented signal from minKNOW and assigns k-mers to the events using a hidden Markov model. Nanocall does not support newer chemistries after R7.3 directly, it can be used but its accuracy is significantly lower than other basecallers.

DeepNano[1] was the first open-source basecaller based on neural networks that uses bidirectional recurent neural networks. DeepNano was written in Python, using the Theano library. When released, originaly supported R7 chemistry, but support for R9.4 and R9.5 was added recently.

3. Methods

Process of basecalling can be represented as problem of machine translation where sentence is translated from one language (sequence of events or sequence of current measurements) to an other (sequence of nucleotides).

Instead of opting for the traditional path using RNN (recurent neural networks) we tried using CNN(Convolutional neural networks).

3.1. Arhitecture

This section gives general idea behind recurrent neural networks and possible problems that serve as motivation for the different approach - usage of convolutional neural networks.

3.1.1. RNN

Recurrent neural networks can be viewed as a simple feed-forward network with the twist 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 and that state is updated each step. The figure shows simple RNN and the same RNN unfolded in time. Unrolling is simple way of showing how network processes each input in the sequence and updates it's hidden state and is shown in figure 3.1.

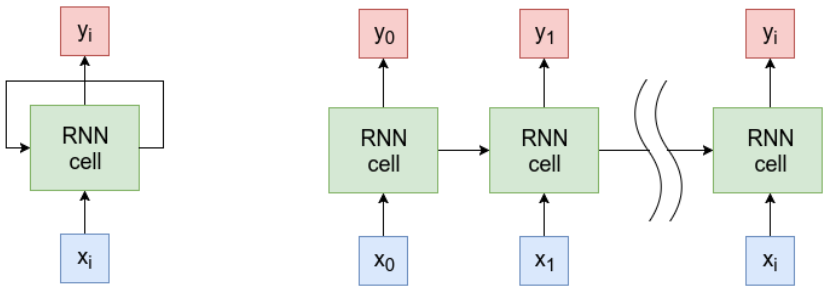

Figure 3.1: An unrolled recurrent neural network

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 same factor, depending on a scale it can make gradient vanish (drop to 0) or exponentially grow each step and explode. These issues are called the vanishing and exploding gradient[2] and are generally resolved by a variant of RNN called *LSTM*[6]. To take into account dependencies of both previous inputs and next inputs in the sequence bidirectional networks (*BRNN*) are used. 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 RNN and current input. This is the approach used in DeepNano[1].

One of the major drawbacks of this kind of networks is computation time. RNN operate sequentially, the output for the second step depends on the first step and so on, which makes parallelization capabilities of RNN quite limited. This especially goes for Bidirectional RNN.

3.1.2. CNN

$$y_i = w_1 * x_{i-1} + w_2 * x_i + w_3 * x_{i+1}$$

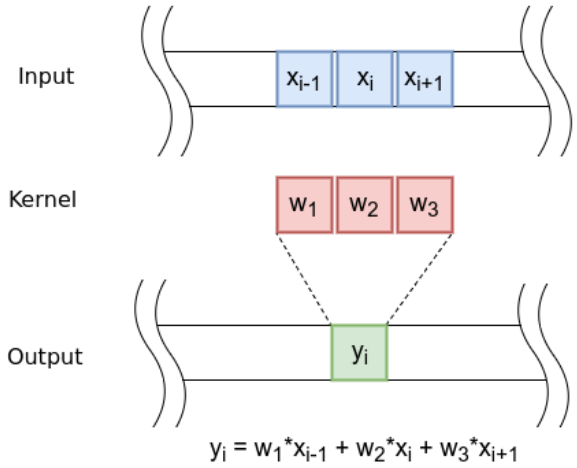

Figure 3.2: Convolution layer, kernel size 3

3.1.3. Residual Networks

Vanishing gradient problem

3.1.4. Gated Residual Networks

Why not use only needed - distribution of gates

3.2. CTC Loss

The goal is to design model which can convert from a sequence of events 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(the length of base pairs is always smaller than the length of the raw signal), which may pose a problem.

Instead of having a variable size of the output from the neural network, we can limit it to length m (maximal length of output sequence Y). The neural network can be considered to be simply a function that takes in some input sequence X (of length n) and generate some sequence O (of length m). Note that this generated sequence is not same as output sequence Y.

3.2.1. Definition

They key idea behind Connectionist Temporal Classification(CTC)[5] 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, network is trained by creating an objective function that restricts the maximum likelihood decoding for a given sequence X to correspond to our desired target sequence Y.

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

The output generated by the network is called *path*. Path is defined by the sequence of it's elements $\pi = (\pi_1, \pi_2, ..., \pi_m)$. The probability of a given path π, given inputs X, can then be written as the product of all its forming elements:

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

where $o_t(\pi_t)$ is probability of 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. The probability of output sequence Y is then the sum of probabilities of all paths that

decode to Y:

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

3.2.2. Objective

Given the dataset $D = \{(X, Y)\}$, training objective is the maximization of the likelihood of each training sample which corresponds to the minimization of negative log likelihood:

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

3.2.3. Output decoding

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

$$Y^* = \underset{Y \in L^m}{\operatorname{argmin}} P(Y|X),$$

where L^m set of all possible sequences over alphabet L with length less than or equals to m

This problem is computationally intractable but exist algorithms that approximate decoding. One of those algorithms is beam search.

This serves as a brief overview of the method and explains key concepts why it is used. More detailed explanation can be found in original paper[5] or various blog post[4].

4. Implementation

4.1. Deep Learning model

4.2. Training

To train described model we need to obtain dataset that consist of sample pairs (X_i, Y_i) where (X_i) is input signal and Y_i is output sequence. One of the mayor issues is determining correct output for given signal. One option is to use existing basecaller like Metrichor to determine output sequences.

Supervised learning so for need to specify dataset that for each input signal we need specify desired sequence of bases. Xi, Yi where len of X and len of Yi are not specified.

For each input file, ground truth is not specified. If we use output of Metrihor or any other basecaller we limit our model to obtain accuracy of used basecaller in best case. We limit our train data to only sequencing data of know organisms (organisms with know reference genome) and try to correct data by aligning the read produced by metrichor or any other basecaller to reference genome. Alignment destination is used as target sequencing in training. Using Metrichor basecalled data we can determine for each called event values such as start in signal, length of event, k-mer state in the pore and using move field change of k-mer from previous state.

Used dataset for training consists of raw signal from fast5 files split raw data into blocks of size l. For each block it is easy to determine basecalled events from Metrichor that belong to particular block using start information and from those events basecalled sequence.

$block\_index = \frac{event_{start} * sampling\_rate}{block\_size}$

Full basecalled sequence is aligned to the reference genome and alignment was obtained. For each block target sequence is determined from alignment information.

Figure X shows how for given signal block

Figure bla shows augmentation of data.

To eliminate possibility of overfitting to the know reference, model is trained and tested on reads that align to different regions of reference genome and those regions should not overlap. Also test is conducted on separate set of sequencing data for different organism

than the one used for training.

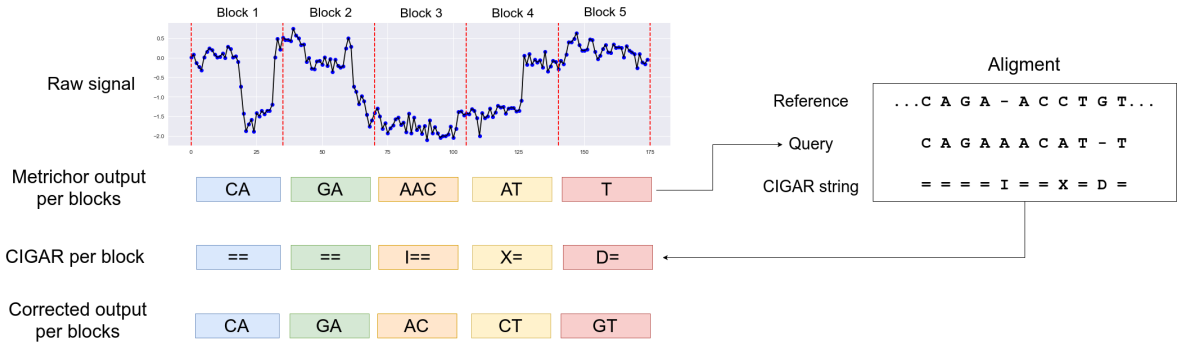

Figure 4.1: Dataset preparation

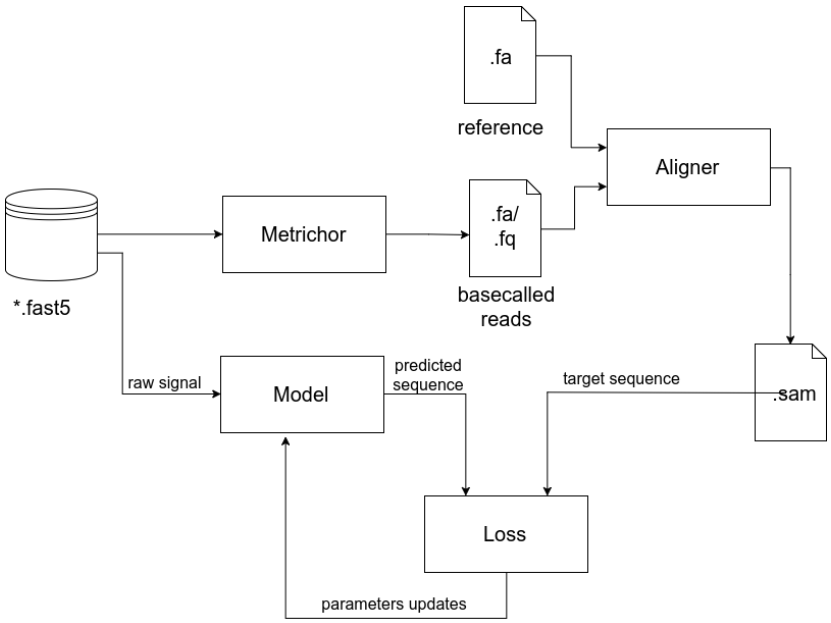

Figure 4.2: Overview of training pipeline

4.3. Evaluation methods

To evaluate trained model we base call test set, align those reads to reference and calculate various statistics on align data. From cigar string it is easy to calculate following:

$$read\_len = n\_matches + n\_missmatches + n\_insertions$$

$$match\_rate = \frac{n\_matches}{read\_lenght}$$

$$missmatch\_rate = \frac{n\_missmatches}{read\_lenght}$$

$$insertion\_rate = \frac{n\_insertions}{read\_lenght}$$

$$deletion\_rate = \frac{n\_deletion}{read\_lenght}$$

$$match\_rate = \frac{n\_matches}{read\_lenght}$$

Results are calculated for each read in test dataset and mean value and standard deviation is expressed for the whole dataset.

To validate consistency of a basecaller, basecalled data is aligned to the reference genome and consensus sequence is called from all reads covering single position. Consensus sequence is compared with the reference genome and following measures are calculated:

$$identity\_percentage = 100 * \frac{n\_correct\_bases}{reference\_lenght}$$

$$match\_rate = \frac{n\_correct\_bases}{consensus\_lenght}$$

$$snp\_rate = \frac{n\_snp}{consensus\_lenght}$$

$$insertion\_rate = \frac{n\_insertions}{consensus\_lenght}$$

$$deletion\_rate = \frac{n\_deletion}{consensus\_lenght}$$

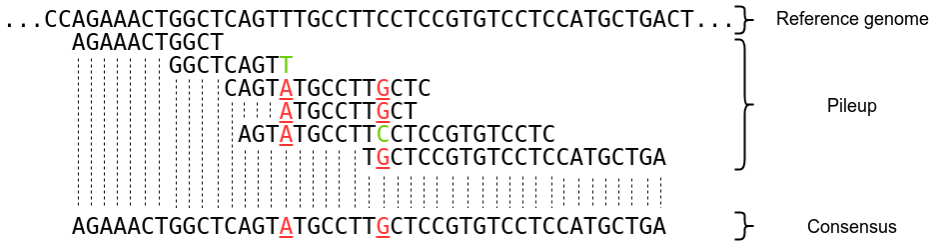

Figure 4.3: Consensus from pileup

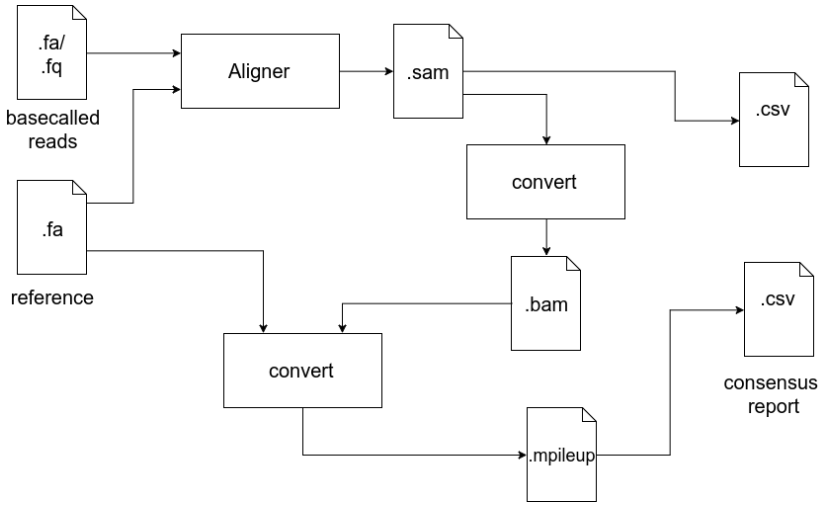

Figure 4.4: Overview of evaluation pipeline

4.4. Technologies

Overall solution was implemented in Python programing language. Described model is implemented using TensorFlow. Tensorflow is an open source software library for numerical computation using data flow graphs developed by Google. TensorFlow, even tho is considered low-level framework offers implementations of higher level concepts (layers, losses, and 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 lacks GPU implementation of used CTC loss, so WARP-CTC (https://github.com/baidu-research/warp-ctc) 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 any other aligner that outputs results in Sam file format.

SAMTools(link) and its python bindings PySam(link) were used for conversions between various file formats used in Bioinformatics.

Docker was used for automating the deployments on different machines. It helps us resolve problem know as "dependency hell"(link) keeping all dependencies in single container thus eliminating possible conflict between packages on host OS. Nvidia docker was used for GPU support.

All training was 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.

5. Results

5.1. Data

5.2. Error rates per read

5.3. Consensus analysis

6. Conclusion

Conclusion.

BIBLIOGRAPHY

[1] 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.

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

[3] 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.

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

[5] 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.

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

[7] 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.

[8] 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.

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

[10] 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.

Deep Learning Model for Base Calling of MinION Nanopore Reads

Abstract

Abstract.

Keywords: Keywords.

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

Sažetak

Sažetak na hrvatskom jeziku.

Ključne riječi: Ključne riječi, odvojene zarezima.