

UNIVERSITY OF ZAGREB  
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

MASTER THESIS num. 1572

# **End-to-End Deep Learning Model for Base Calling of MinION Nanopore Reads**

Neven Miculinić

Zagreb, June 2018.

*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 Marko Ratković for his help with this thesis.*

# CONTENTS

<b>1. Introduction</b>	<b>1</b>
1.1. Organization . . . . .	1
<b>2. Background</b>	<b>2</b>
2.1. Oxford Nanopore MinION . . . . .	3
2.2. Related work . . . . .	4
2.2.1. Oxford Nanopore Technologies . . . . .	4
2.2.2. Third-party basecallers . . . . .	6
<b>3. Methods</b>	<b>7</b>
3.1. Neural Network . . . . .	7
3.2. Activation functions . . . . .	8
3.3. CNN . . . . .	8
3.4. RNN . . . . .	10
3.5. Pooling layer . . . . .	11
<b>4. System architecture</b>	<b>13</b>
4.1. Data Preparation . . . . .	13
4.1.1. Data correction . . . . .	13
4.1.2. Data sources . . . . .	15
4.2. Training pipeline . . . . .	15
4.3. Hyperparameter optimization . . . . .	16
4.3.1. Final model arhitecture . . . . .	19
<b>5. Results</b>	<b>20</b>
5.1. Definitions . . . . .	20
5.2. Other solutions . . . . .	20
<b>6. Conclusion</b>	<b>22</b>
<b>Bibliography</b>	<b>23</b>

# LIST OF FIGURES

2.1. Depiction of the sequencing process . . . . .	3
2.2. DNA strain being pulled through a nanopore . . . . .	4
2.3. Structure of FAST5 file and raw signal plot show in <i>HDFView</i> . . . . .	4
3.1. Simple three layer feed forward neural network . . . . .	7
3.2. Code generating the activation function plot. Also shows how tensorflow and keras could be used . . . . .	9
3.3. Plot of common activation functions between -1 and 1. PrELU is excluded since the parameter $\alpha$ is learnt during training . . . . .	9
3.4. Convolution layer, kernel size 3 with stride 1. Adapted from (Ratković, 2017) with authors permission. . . . .	10
3.5. An unrolled recurrent neural network. Adapted from (Ratković, 2017) with authors permission. . . . .	11
3.6. Example of max pool layer in convolutional neural network. Adapted from <a href="https://leonardoaraujosantos.gitbooks.io/artificial-intelligence/content/pooling_layer.html">https://leonardoaraujosantos.gitbooks.io/artificial-intelligence/ content/pooling_layer.html</a> . . . . .	12
4.1. dataset protobuf description . . . . .	14
4.2. Example of simple alignment and CIGAR string. Adapted from (Ratković, 2017) . . . . .	15
4.3. dataset protobuf description . . . . .	17
4.4. training config yaml . . . . .	18
4.5. Example of starting training command. In the config model is properly con- figures to use <code>/data</code> and <code>/model</code> paths . . . . .	18
4.6. HyperParameter options description . . . . .	19
5.1. Basecallers read identity as tested by (Wick et al., 2018) . . . . .	21
5.2. Basecallers assembly identity as tested by (Wick et al., 2018) . . . . .	21

# 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 (Lecun and Bengio, 1998; Krizhevsky et al., 2012) In this paper, we present application of deep learning for DNA basecalling problem.

Oxford Nanopore Technology's MinION nanopore sequencing platform Mikheyev and Tin (2014) is the first portable DNA sequencing device. It produces longer reads than competing technologies. In addition, it enables real-time data analysis which makes it suitable for various applications. Although MinION is able to produce long reads, even up to 882 kb Loman (a,b), they have an error rate of 10% or higher. This master thesis uses R9.4 pore model and compares previous techniques with novel auto-encoder multi-task training.

## 1.1. Organization

[TODO]: Write some fancy stuff once completed.

## 2. Background

Due to technical constraints, it's infeasible to sequence whole DNA in single strand. Every sequencing technology to date have an upper limit how big strand can it precisely sequence. This limit is considerably smaller than size of genome. For example E.Coli has 4.5 million base pairs in its DNA, while Sanger's sequencing maximum output is around 1000 base pairs max. To make DNA basecalling feasible technique called shotgun sequencing was invented. The strand is cloned number of times, then via chemical agent broken down into smaller fragments of appropriate length. 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.

If we have reference sequence we usually align the reads on the reference to aid us into genome assembly. Otherwise we have to use many de novo assembly techniques.

The right analogy would be building a puzzle. Since we cannot scan the whole puzzle because our camera is too small or imprecise, we are scanning pieces of the whole picture. Puzzle pieces would represent fragments in this analogy. If we have a map, even a rough one, it shall aid us into assembling those puzzle pieces into complete pictures. Otherwise we're fiddling in the dark and using de novo assembly techniques.

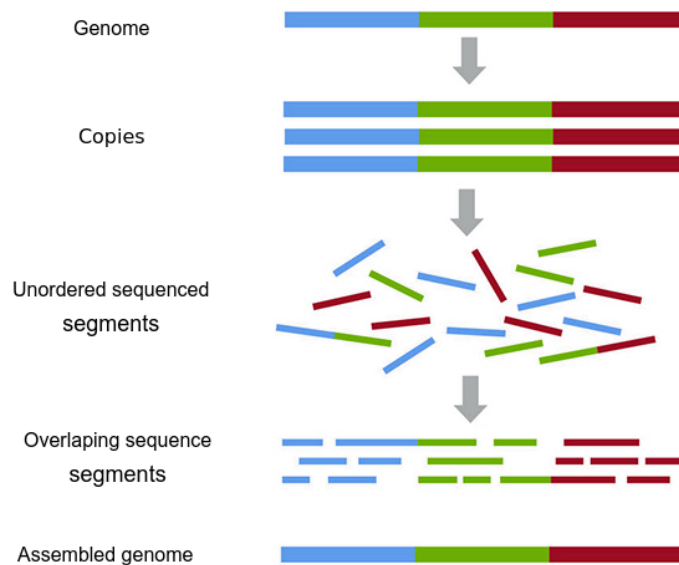
Figure 2.1 depicts process of sequencing visually.

In 1977, Frederick Sanger (Mile Šikić, 2013)(Pettersson et al., 2009) started development of sequencing technologies. It allowed read lengths up to 1000 bases with very high accuracy(99.9%) at the cost of 1\$ per 1000 bases. Later, second generation sequencing, like IAN Torrent and Illumina devices, reduced the price while keeping the accuracy high. However, they had a cost of shorter read lengths, about a few hundred base pairs, which makes resolving repetitive regions practically impossible.

Third generation sequencing technologies have longer read lengths at the accuracy's expense. PacBio, for example, developed technology with a few thousand bases with error rates of ~10-15%.

MinION sequences, which this master thesis use, made sequencing less expensive and even portable.





**Figure 2.1:** Depiction of the sequencing process

## 2.1. 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 Jain et al. (2017) what could potentially lead to personalized genomic medicine. It weighs only 87 grams, and its portability lead to uses on international space station and the antarctic among other places.

Under the hood, it has numerous nano-meter sized pores, thus a names Nanopore. Each pore has width of around 6 nucleotied. Under electric current DNA strand passed through the pore and changes its electric resistance. The sensor measures current through the pore multiple times a second <sup>1</sup>. This signal varies depending which k-mer is occupying the pore, and on its basis we're performing the basecalling. On figure 2.2 this process is visually depicted.

MinION devices can produce long reads, usually tens of thousand base pairs (with reported reads lengths of 100 thousand Loman (a) and even recently above 800 thousand base pairs Loman (b)), but with high sequencing error than older generations of sequencing technologies.

The sequecing resulting file is in FAST5 format, which is adapted HDF5 file format, popular in bioinformatics community. It stores raw signal, alongside various metadata. Unfortunately, many basecallers, including the official ones, upon executing store their results

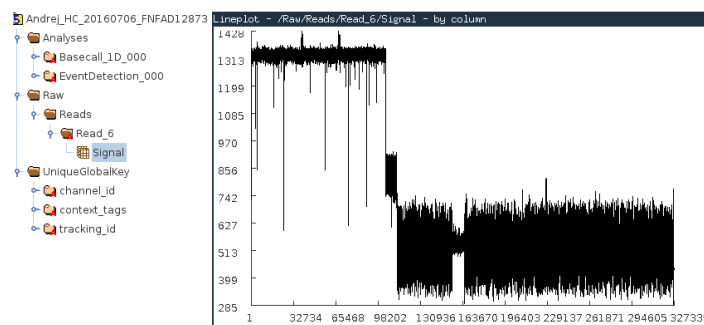
<sup>1</sup>The model we worked with had 4000 samples per seconds

<sup>2</sup>Figure adapted from <https://nanoporetech.com/how-it-works>



**Figure 2.2:** DNA strain being pulled through a nanopore <sup>2</sup>

in the FAST5 files. It leads to data and processing coupling in single file, and bloated file sizes.



**Figure 2.3:** Structure of FAST5 file and raw signal line plot show in *HDFView* <sup>3</sup>

## 2.2. Related work

### 2.2.1. Oxford Nanopore Technologies

This subsection covers various basecallers published by ONT, the MinION device maker.

#### Metrichor

Metrichor is now defunct cloud based basecaller. The older versions used *hidden Markov models* (HMM) as underlying algorithm. Preprocessing started with segmenting signal into smaller chunks called events with their start, end, length, mean signal strength and standard deviation. This events are observations in the HMM model, and the underlying generating

<sup>3</sup><https://support.hdfgroup.org/products/java/hdfview/>

hidden sequence is sequence of 6-mers. They build their HMM transition matrix with stay, skip 1 and skip 2 probabilities, that is underlying 6-mer cannot move for more than 2 nucleotides per event. Basecalling is performed using Viterbi algorithm. This approach showed poor results when calling long homopolymer stretches as the context in the pore remains the same Goodwin et al. (2016)Ip et al. (2015).

## **Nanonet**

Nanonet is ONT first generation neural network basecaller. It used to be available on github, but it's now defunct and unavailable.

## **Albacore**

Albacore is a production basecaller provided by Oxford Nanopore, and uses a command-line interface. It utilizes the latest in Recurrent Neural Network algorithms in order to interpret the signal data from the nanopore, and basecall the DNA or RNA passing through the pore. It implements stable features into Oxford Nanopore Technologies' software products, and is fully supported. It receives .fast5 files as an input, and is capable of producing:

- .fast5 files appended with basecalled information
- .fast5 files that have been processed, but basecall information present in a separate .fastq file

## **Guppy**

Guppy is ONT's new basecaller that can use GPUs to basecall much faster than Albacore. Both the GridION X5 and PromethION contain GPUs and use Guppy to basecall while sequencing. Guppy can also use CPUs and scales well to many-CPU systems, so it may run faster than Albacore even without GPUs. In the future it's intended to replace Albacore as production basecaller.

## **Scrappie**

Scrappie<sup>4</sup> is ONT's research basecaller. 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.

Unlike Albacore, Scrappie does not have fastq output, either directly or by writing it into the fast5 files – it only produces fasta reads.

---

<sup>4</sup><https://github.com/nanoporetech/scrappie>

## 2.2.2. Third-party basecallers

### Nanocall

Nanocall (David et al., 2016) 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

DeepNano (Boža et al., 2017) 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.

### basecRAWller

basecRAWller<sup>5</sup> is developed by Marcus Stoiber and James Brown at the Lawrence Berkeley National Laboratory.

### Chiron

Chiron (Teng et al., 2017) is developed by Haotian Teng and others in Lachlan Coin's group at the University of Queensland. They are basecalling from the raw signal, using first residual convolutional neural network, then LSTM and finally beam search or greedy decoder depending on chosen configuration.

---

<sup>5</sup><https://basecrawller.lbl.gov/>

## 3. Methods

This chapter is dedicated to explaining key deep learning concepts used throughout the master thesis. It's here primarily for completeness, and it's author recommendation to go into detail via other sources, for example Deep learning book (Goodfellow et al., 2016), google, or research papers cited for most of the techniques.. TensorFlow (Abadi et al., 2015) and Keras (Chollet et al., 2015) deep learning frameworks were used for implementation. For each deep learning concept I'll provide equivalent keras/tensorflow code whichever one is simpler and used throughout the codebase.

### 3.1. Neural Network

Feed-forward Neural network is the basic building block of any deep learning system. It's composition of multiple differentiable functions. Commonly we have input vector  $x$ , apply some linear transformation to it and add bias, and finally on the result some activation function. Details on common choices for actionvation function are in section 3.2. In mathematical language  $y = f(Ax + b)$  would be one layer of neural network transforming input  $x$  into output  $y$ . Stacking those operation we get multiple layers, hence the word deep in deep learning. Simple 3-layer neural network is depicted in figure 3.1.

$$y_1 = f(A_1x + b_1)$$

$$y_2 = f(A_2y_1 + b_2)$$

$$y = f(A_3y_2 + b_3)$$

**Figure 3.1:** Simple three layer feed forward neural network

## 3.2. Activation functions

Most neural network operations are linear transformation. Composing multiple linear transformation we get new linear transformation. Thus have non-linear behavior we use the non-linear activation functions. Originally, the most popular choice was tanh and  $\sigma(x) = \frac{1}{1+e^{-x}}$  activation functions. They are nice because of limited output domain.

However, other choices proved more effective, especially with deep neural network due to greater learning speeds, and to overcome gradient vanishing problem. Gradient vanishing refers to neural network gradient approaching zero as we back propagate through more and more layers. Exploding gradient is related phenomena in which gradient approaches infinity. Both present serious hampering to neural network training.

ReLU, The rectified linear unit,  $f(x) = \max(0, x)$ , is one hugely popular choice and decent baseline compared to other ReLU variants. ReLU greatly accelerates the convergence of stochastic gradient descent compared to  $\sigma$  and tanh activation functions (Krizhevsky et al., 2012).

Furthermore, its calculation is drastically simpler then computing transcendental functions, like  $\sigma$  or tanh.

Over time, ReLU showed its downsides, called *dying ReLU*. It still saturates the gradients when it's 0, that is when  $x \leq 0$  giving no useful gradient to back propagate. Thus several ReLU variant have been proposed: PrRelu (He et al., 2015) in equation 3.1, ELU (Clevert et al., 2015) in equation 3.2, and finally SeLU (Klambauer et al., 2017) in equation 3.3. In Selu constants  $\alpha$  and  $\lambda$  are chosen in such a way that output gravitates towards normal distribution with zero mean and unit variance. Those constants are:  $\lambda = 1.0507$  and  $\alpha = 1.6732$ . Code generating the function plot is displayed in figure ??, and the plot is in figure 3.3

$$PrELU(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{otherwise} \end{cases} \quad (3.1)$$

$$ELU(x) = \begin{cases} x & \text{if } x > 0 \\ \exp(x) - 1 & \text{otherwise} \end{cases} \quad (3.2)$$

$$\text{selu}(x) = \lambda \begin{cases} x & \text{if } x > 0 \\ \alpha e^x - \alpha & \text{otherwise} \end{cases} \quad (3.3)$$

## 3.3. CNN

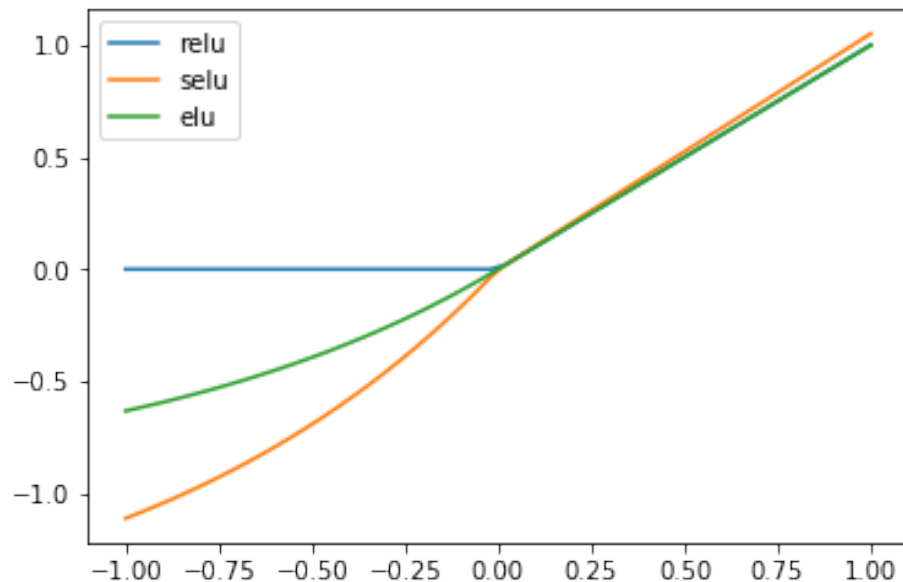
Convolutional Neural Networks(CNNs) are the bread and butter of almost all computer vision system today, and they lunched the deep learning hype with unprecedented results on

```

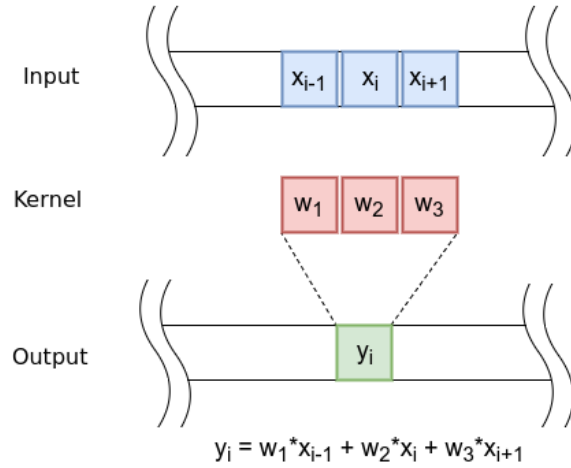
1 import tensorflow as tf
2 import keras
3 import seaborn as sns
4 import numpy as np
5 import matplotlib.pyplot as plt
6
7 for act in ["relu", "selu", "elu"]:
8     with tf.Graph().as_default():
9         x = tf.placeholder(shape=(None, ), dtype=tf.float32)
10        y = keras.layers.Activation(act)(x)
11
12        with tf.Session() as sess:
13            xx = np.linspace(-1, 1)
14            yy = sess.run(y, feed_dict={
15                x: xx
16            })
17            plt.plot(xx, yy, label=act)
18            plt.legend()

```

**Figure 3.2:** Code generating the activation function plot. Also shows how tensorflow and keras could be used



**Figure 3.3:** Plot of common activation functions between -1 and 1. PrELU is excluded since the parameter  $\alpha$  is learnt during training



**Figure 3.4:** Convolution layer, kernel size 3 with stride 1. Adapted from (Ratković, 2017) with authors permission.

Image Classification problems. Lately, their scope is expanded to Natural language processing (NLP) tasks with promising results (Kalchbrenner et al., 2016; Gehring et al., 2017).

Convolution is a type of feed-forward neural network layer where the weights are tied together. We have a sliding window over which we apply linear transformation of the input, and get the output element. This operation is repeated to get the full output tensor. For 1D case <sup>1</sup> figure 3.4 depicts it visually.

There's also atrous convolution Chen et al. (2017), also known as dilated convolution where spaced input tensor elements are taken depending on the dilation factor. Ordinary convolution is a special case with dilation = 1.

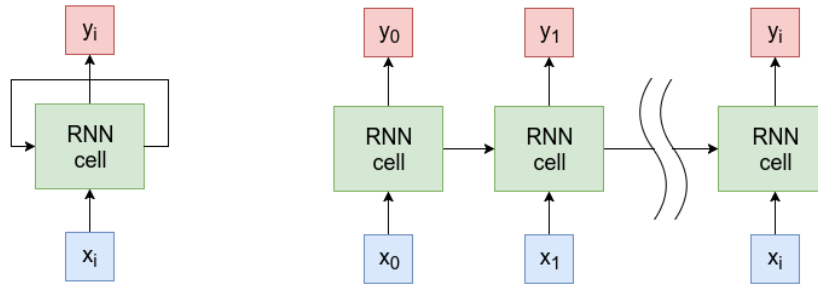
### 3.4. RNN

RNN, residual neural networks are one of the basic building blocks in sequence models. They are feed-forward neural network spanned in time domain. The same neural network is applied to two inputs, hidden state and input at time  $t$  and gives two outputs, new hidden state and output at time  $t$ . The historic information remains saved in the hidden state, and it's propagated to the future. It's visual description is depicted in figure 3.5.

They can be unrolled in one massive feed-forward neural network. They are trained with backpropagation through time, on this unrolled graph. Usually the unrolling is capped and fixed number or time steps. Common issue to vanilla RNNs is vanishing and exploding gradients problem. It's solved by redesigning the basic RNN cell. There are two common approaches called LSTM (Hochreiter and Schmidhuber, 1997) and GRU (Chung et al., 2014).

<sup>1</sup><https://keras.io/layers/convolutional>





**Figure 3.5:** An unrolled recurrent neural network. Adapted from (Ratković, 2017) with authors permission.

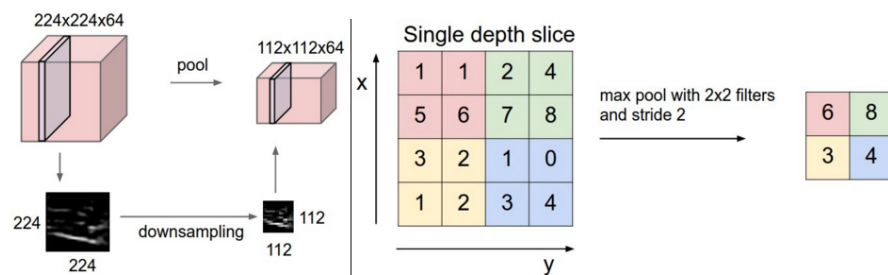
Bidirectional Recurrent Neural (BiRNN) networks are used when the current output not only depends on the previous elements in the sequence but also future elements. Basically we're stacking two RNNs, one in forward direction and another in the backward and concatenating the hidden states/outputs. This approach was used in DeepNano (Boža et al., 2017).

Despite their modeling power, main drawback is their speed. Since they are processing data sequentially, it's hard to parallelize those operations.

### 3.5. Pooling layer

Pooling layers refer to tensor dimensionality reduction. Tensor is squeezed by some factor, and each block of factor width is aggregated using some aggregation function. In other words, we divide our tensor into equally shaped hyperrectangles, and each aggregate using aggregation function yielding new hyperrectangle. For example, maxPool2d for image sized 50x50 shall with pool size (2,2) shall divide it into rectangles of size (2, 2) max those 4 elements and result in image sized 25x25. This is visually depicted in figure 3.6. Common choices are AvgPool, and MaxPool (Scherer et al., 2010) <sup>2</sup>.

<sup>2</sup><https://keras.io/layers/pooling/>



**Figure 3.6:** Example of max pool layer in convolutional neural network. Adapted from [https://leonardoaraujosantos.gitbooks.io/artificial-intelligence/content/pooling\\_layer.html](https://leonardoaraujosantos.gitbooks.io/artificial-intelligence/content/pooling_layer.html)

## 4. System architecture

### 4.1. Data Preparation

In previous project incarnation, we used fast5 with additional custom .ref files as our dataset. This setup had some issues. First, it was IO heavy since we were reading the whole fast5 file which contained non-raw data section <sup>1</sup>, data management with separated raw files and labels, and lastly it was a magic to parse it every time. Which exactly group or subgroup within fast5, or is the signal start normalized to 0, or is it some arbitrary begging written at some other place in fast5 files changes from version to version.

Therefore, for this project I've decided to make my own custom format for training described via protobuf (Google). The issue from previous versions weren't the only reason. Another was ease of importing other peoples data in their formats into this one. If I can read it, I can easily write plugin converting between formats. As of writing this thesis 4 such plugins were written since I wasn't sure which data I shall use and in which format am I going to import them in.

Ease of machine encoding/decoding as well as other previously mentioned factors guided this decision. Also I've written datapoint format inspector operating on this common format, making my life easier. The source code for conversion procedures can be found on minion-data <sup>2</sup> github repository as well as pypi. The specific interface description language(IDL) can be seen in figure 4.1.

#### 4.1.1. Data correction

Each data point has been previously basecalled, and some resulting sequence was obtained. Then each sequence was corrected by aligning to the reference genome. By aligning to the reference we get the reference genome region and the CIGAR strings describing the alignment. The CIGAR string is list ordered sequence of insertions(I), deletions(D), matches(=) and mismatches(X). Insertions are bases present in the query string, but absent from reference, and deletions are the opposite.

---

<sup>1</sup>like basecalling from other basecallers

<sup>2</sup><https://github.com/nmiculnic/minion-data>

```

1 syntax = "proto3";
2
3 package dataset;
4
5 enum BasePair {
6     A = 0;
7     C = 1;
8     G = 2;
9     T = 3;
10    BLANK = 4;
11 }
12
13 enum Cigar {
14     MATCH = 0;
15     MISMATCH = 1;
16     INSERTION = 2; // Insertion, soft clip, hard clip
17     DELETION = 3;  // Deletion, N, P
18 }
19
20 message DataPoint {
21     message BPConfidenceInterval {
22         uint64 lower = 1;
23         uint64 upper = 2;
24         BasePair pair = 3;
25     }
26     repeated float signal = 1;
27     repeated BasePair basecalled = 2; // What we basecalled
28     repeated BPConfidenceInterval labels = 3; // labels describe ←
        corrected basecalled signal for training
29 }
30

```

**Figure 4.1:** dataset protobuf description

	Alignment
Reference	...C A G A - A C C T G T...
Query	C A G A A A C A T - T
Alignment operations	= = = = I = = X = D =
CIGAR string	4= 1I 2= 1X 1= 1D 1=

**Figure 4.2:** Example of simple alignment and CIGAR string. Adapted from (Ratković, 2017)

Example alignment is depicted in figure 4.2. We take the error corrected labels and place them in dataset protobuf file for further training processing.

### 4.1.2. Data sources

Data has been downloaded from [https://data.genomicsresearch.org/Projects/online\\_dataset/train\\_set\\_all/](https://data.genomicsresearch.org/Projects/online_dataset/train_set_all/). The following species were provided there by the Chiron team:

- Human
- E. Coli
- Lambda Phage

For the concrete Chiron dataset, the re-squiggled preparation method was used. (The data was re-squiggled, that is after aligning the read on the reference, the read data is improved and each base pairs place on the raw signal is calculated.)

The re-squiggled basecalled data is located at `/Analyses/RawGenomeCorrected_000/BaseCa`. The interesting code fragments are in function `processDataPoint` of file `minion_data/preperation/` from `minion-data` python package.

After the gzipped dataset is prepared, it goes into the training pipeline. The whole training & testing pipeline is available open source on <https://github.com/nmiculinic/minion-basecaller>

## 4.2. Training pipeline

After getting data in uniform format, the training part comes next. In aiming this as observable and simple as possible I've modeled config in using python3 `typing.NameTuple`<sup>3</sup> and `volptous` for data validation. With those two tools I've created statically typed configuration file format and data validation with human readable error. As serialization format I've

<sup>3</sup><https://docs.python.org/3/library/typing.html>

chosen YAML for its readability and terseness. The config file definition is in figure 4.3 and example config file for training is in figure 4.4. The source code can be found in module `mincall.train._train.py` and we're recommending reading it for extended details.

Input data is read via background thread which processes it, packages it into signal and label slices, and pushed into tensorflow space queues. Inside those queues, data is shuffled, and composed into batches for further training. This part of the code has extensive test suite, including end2end test with golden file <sup>4</sup>

From configuration rest of the pipeline is initialized. This included model, with its hyperparameters, loading config defined train and test sets, learning rate schedule, sequence lengths, batch size and other settings. Keras is used for model definition due to its simplicity in design.

Additionally I've embedded edlib (Šošić and Šikić, 2017) in the tensorflow's `tf.py_func` operator which enabled me having match, mismatch, insertion, deletion and identity rates observer during training on both train and test set.

Standard python logging module was used with multiple backends. One backend saved to log file for further inspection. The other one shipped logs to graylog system which I've setup on faculty servers. This system enabled me log searching and greater efficiency in finding bugs.

Example command for starting the training procedure is shown in figure 4.5.

### 4.3. Hyperparameter optimization

Our models have various hyperparameters. For example, learning rate, receptive width size, number of blocks, etc. To gain a better model we're not only optimizing models parameters, but also hyperparameters on the validation set. For this I've implemented hyperparam search capabilities in the programming solution.

The design rationally do as little things as possible. It takes the training configuration with injected hyperparams, and it creates new directory and `config.yml` for each new hyperparam assignment. Then it releases control to the training code part, as if we had run training on the command like previously described in figure 4.5 making this maximally observable. After the training completes it reports back to the hyperparam controller the results and according to backing hyperparameter optimization procedure new assignment is calculated.

[TODO] – assignment procedures.

---

<sup>4</sup>golden file is test technique where you save function output to golden file, verify manually it's correct, and after each successful test you're checking that nothing changed. This way we can be sure refactoring didn't influence correctness.

```

1 class TrainConfig(NamedTuple):
2     model_name: str
3     train_data: List[DataDir]
4     test_data: List[DataDir]
5     logdir: str
6     seq_length: int
7     batch_size: int
8     surrogate_base_pair: bool
9
10    train_steps: int
11    init_learning_rate: float
12    lr_decay_steps: int
13    lr_decay_rate: float
14
15    model_hparams: dict = {}
16    grad_clipping: float = 10.0
17    validate_every: int = 50
18    run_trace_every: int = 5000
19    save_every: int = 2000
20
21    tensorboard_debug: str = "" # Empty string is use CLI debug
22    debug: bool = False
23    trace: bool = False
24
25    @classmethod
26    def schema(cls, data):
27        return named_tuple_helper(
28            cls, {
29                'train_data': [DataDir.schema],
30                'test_data': [DataDir.schema],
31            }, data
32        )
33
34

```

**Figure 4.3:** dataset protobuf description

```

version: "v0.1"
train:
  train_data:
    - name: "r9.4"
      dir: "./mincall/example"
  test_data:
    - name: "r9.4"
      dir: "./mincall/example_test"
model_name: "dummy"
model_hparams:
  num_layers: 5
surrogate_base_pair: false
train_steps: 60
init_learning_rate: !!float 1e-4
lr_decay_steps: 10000
lr_decay_rate: 0.5
seq_length: 40
batch_size: 10
logdir: "./logs"

```

**Figure 4.4:** training config yaml

```

1 docker run --rm -it -u="$(id -u):$(id -g)" \
2 -v $DATA_DIR:/data:ro \
3 -v $MODEL_DIR:/model \
4 nmiculnic/mincall:latest-py-gpu train --config /model/config.yml
5

```

**Figure 4.5:** Example of starting training command. In the config model is properly configured to use /data and /model paths



```

1 class HyperParamCfg(NamedTuple):
2     model_name: str
3     train_data: List[DataDir]
4     test_data: List[DataDir]
5     seq_length: Param
6     batch_size: Param
7     surrogate_base_pair: Param
8
9     train_steps: Param
10    init_learning_rate: Param
11    lr_decay_steps: Param
12    lr_decay_rate: Param
13
14    model_hparams: Dict[str, Param]
15
16    work_dir: str
17    grad_clipping: float = 10.0
18    validate_every: int = 50
19    run_trace_every: int = 5000
20    save_every: int = 2000
21

```

**Figure 4.6:** HyperParameter options description

It's config is similar to the training and its NamedTuple definition can be seen in figure 4.6. The param is flexible entry; it can be a single scalar value, and then it's not hyperoptimized, or it can be dictionary with `type` field and other fields depending on the parameter types, e.g. int, float, categorical, etc. Example YAML hyperparameter config is in figure

### 4.3.1. Final model architecture

[TODO]

## 5. Results

### 5.1. Definitions

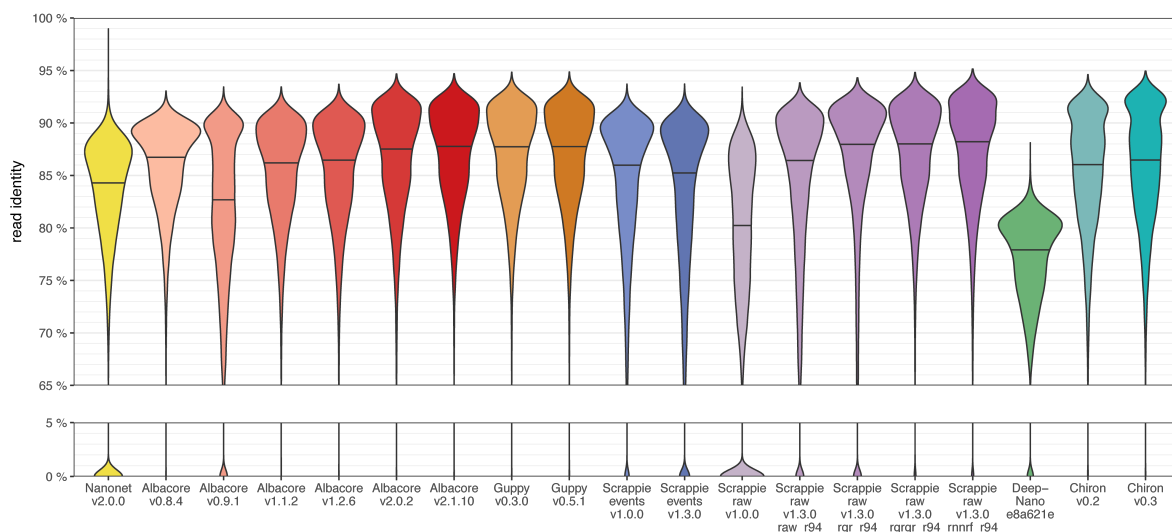
### 5.2. Other solutions

On his github repo, Ryan Wick composed multiple basecaller comparasion (Wick et al., 2018) <sup>1</sup>. This section briefly summarizes his work and it's recommended to check original authors work, it's quite good! Due to resource constraints both time and compute, I'm unable to run my own testing on my data, however I can use subset of Ryan Wick's dataset to perform my model evaluation, thus it's somewhat relevant. This summarization is composed of two figures, read identity in figure 5.1 and consensus identity in figure 5.2. All figures ara pre-polished states.

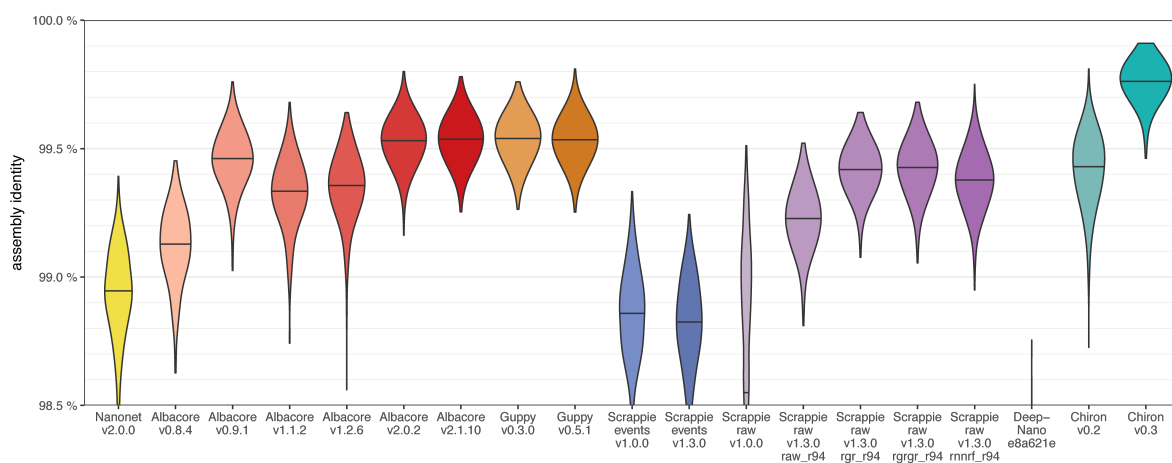
All definitions defined in previous section are valid.

---

<sup>1</sup><https://github.com/rrwick/Basecalling-comparison>



**Figure 5.1:** Basecallers read identity as tested by (Wick et al., 2018)



**Figure 5.2:** Basecallers assembly identity as tested by (Wick et al., 2018)

## 6. Conclusion

[TODO] Write something smart here depending on the results.

# BIBLIOGRAPHY

- Marko Ratković. Deep learning model for base calling of minion nanopore reads, 2017.
- Ryan Wick, Louise M Judd, and Kathryn E Holt. Comparison of oxford nanopore basecalling tools, March 2018. URL <https://doi.org/10.5281/zenodo.1188469>.
- 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>.
- 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>.
- Alexander S Mikheyev and Mandy MY Tin. A first look at the oxford nanopore minion sequencer. *Molecular ecology resources*, 14(6):1097–1102, 2014.
- Nick Loman. *Nanopore R9 rapid run data release*, a. URL <http://lab.loman.net/2016/07/30/nanopore-r9-data-release/>. [Online; posted 30-July-2016].
- Nick Loman. *Thar she blows! Ultra long read method for nanopore sequencing*, b. URL <http://lab.loman.net/2017/03/09/ultrareads-for-nanopore/>. [Online; posted 9-March-2017].
- Mirjana Domazet-Lošo Mile Šikić. *Bioinformatika*. Bioinformatics - course materials, Faculty of Electrical Engineering and Computing, University of Zagreb, 2013.
- Erik Pettersson, Joakim Lundberg, 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>.
- 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>.
- 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.
- Camilla L.C. Ip, Matthew Loose, John R. Tyson, Mariateresa de Cesare, Bonnie L. Brown, Miten Jain, Richard M. Leggett, David A. Eccles, Vadim Zalunin, John M. Urban, Paolo Piazza, Rory J. Bowden, Benedict Paten, Solomon Mwaigwisya, Elizabeth M. Batty, Jared T. Simpson, Terrance P. Snutch, Ewan Birney, David Buck, Sara Goodwin, Hans J. Jansen, Justin O’Grady, and Hugh E. Olsen and. Minion analysis and reference consortium: Phase 1 data release and analysis. *F1000Research*, oct 2015. doi: 10.12688/f1000research.7201.1. URL <https://doi.org/10.12688/f1000research.7201.1>.
- 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>.
- 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>.
- Haotian Teng, Minh Duc Cao, Michael B. Hall, Tania Duarte, Sheng Wang, and Lachlan Coin. Chiron: Translating nanopore raw signal directly into nucleotide sequence using deep learning. *bioRxiv*, 2017. doi: 10.1101/179531. URL <https://www.biorxiv.org/content/early/2017/09/12/179531>.
- Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.
- Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian

- Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL <https://www.tensorflow.org/>. Software available from tensorflow.org.
- François Chollet et al. Keras. <https://keras.io>, 2015.
- Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, 2015.
- Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus), 2015.
- Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks, 2017.
- Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, Aaron van den Oord, Alex Graves, and Koray Kavukcuoglu. Neural machine translation in linear time, 2016.
- Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. Convolutional sequence to sequence learning, 2017.
- Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *CoRR*, abs/1706.05587, 2017. URL <http://arxiv.org/abs/1706.05587>.
- Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
- Junyoung Chung, Çağlar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014. URL <http://arxiv.org/abs/1412.3555>.
- Dominik Scherer, Andreas Müller, and Sven Behnke. Evaluation of pooling operations in convolutional architectures for object recognition. In *Proceedings of the 20th International Conference on Artificial Neural Networks: Part III*, ICANN’10, pages 92–101, Berlin, Heidelberg, 2010. Springer-Verlag. ISBN 3-642-15824-2, 978-3-642-15824-7. URL <http://dl.acm.org/citation.cfm?id=1886436.1886447>.

Google. Protocol buffers. <https://github.com/google/protobuf>.

Martin Šošić and Mile Šikić. Edlib: a c/c++ library for fast, exact sequence alignment using edit distance. *Bioinformatics*, 33(9):1394–1395, 2017. doi: 10.1093/bioinformatics/btw753. URL <http://dx.doi.org/10.1093/bioinformatics/btw753>.



## **End-to-End Deep Learning Model for Base Calling of MinION Nanopore Reads**

### **Abstract**

In the MinION device, single-stranded DNA fragments move through nanopores, which causes drops in the electric current. The electric current is measured at each pore several thousand times per second. Each event is described by the mean and variance of the current and by event duration. This sequence of events is then translated into a DNA sequence by a base caller. Develop a base-caller for MinION nanopore sequencing platform using a deep learning architecture such as convolutional neural networks and recurrent neural networks. Instead of events, use current waveform at the input. Compare the accuracy with the state-of-the-art basecallers. For testing purposes use publicly, available datasets and Graphmap or Minimap 2 tools for aligning called reads on reference genomes. Implement method using TensorFlow or similar library. The code should be documented and hosted on a publicly available Github repository.

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

### **S kraja na kraj model dubokog učenja za određivanje očitanih baza dobivenih uređajem za sekvenciranje MinION**

#### **Sažetak**

Unutar uređaja MinION, fragmenti jednostruke DNA prolaze kroz nanopore, što uzrokuje promjene u električnoj struji. Struja proizvedena na svakoj nanopori mjeri se nekoliko tisuća puta u sekundi. Svaki događaj opisan je srednjom vrijednosti i varijancom struje te svojim trajanjem. Postupak kojim se takav slijed događaja prevodi u niz nukleotida naziva se određivanje očitanih baza. Razviti alat za prozivanje baza za uređaj za sekvenciranje MinION koristeći modele dubokog učenje kao što su konvolucijske i povratne neuronske mreže. Umjesto događaja na ulazu koristi valni oblik struje. Usporediti dobivenu točnost s postojećim rješenjima. U svrhu testiranja koristiti javno dostupne skupove podataka i alate GraphMap ili Minimap 2 za poravnanje očitavanja na referentni genom. Alat implementirati koristeći programsku biblioteku TensorFlow (ili neku sličnu). Programski kod treba biti dokumentiran i javno dostupan preko repozitorija GitHub. **Ključne riječi:** određivanje

baza, Oxford Nanopore Technologies, MinION, duboko učenje, prevođenje, konvolucijske neuronske mreže, rezidualne mreže, CTC gubitak