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 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 fields of Biology and 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 to form 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 or sequencer. 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 base calling is dependent on the underlying software and can be improved using machine learning methods. 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 discussed in later chapters.

Chapter 4 goes into implementation details, training of the deep learning model and explains the methodology 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, 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 is a difficult problem. To resolve this problem whole genome shotgun sequencing approach is used, in which 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.

Figure 2.1 depicts process of sequencing.

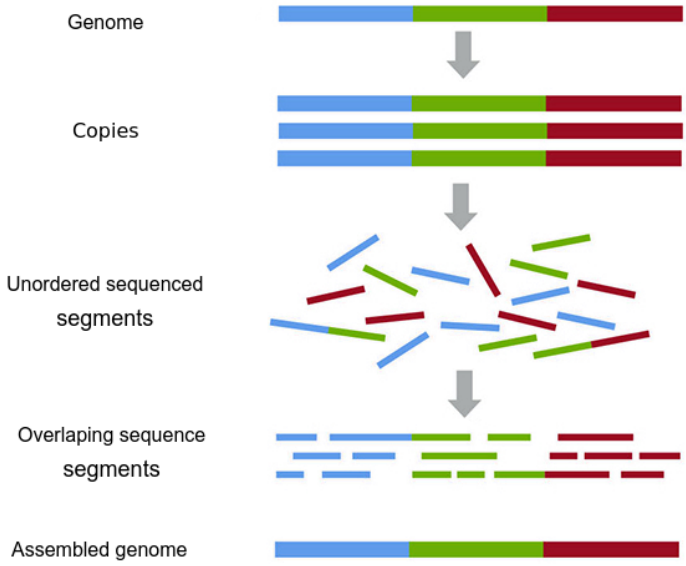

Figure 2.1: Depiction of the sequencing process

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 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. Short reads make resolving repetitive regions practically impossible.

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. The release of, previously mentioned, MinION sequencer made sequencing drastically less expensive and even portable.

2.2. Oxford Nanopore MinION

The MinION device by Oxford Nanopore Technologies is the first portable DNA sequencing device. It's 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, pores of nanoscale 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 differences nucleotides present in the pore. Figure 2.2 shows the change of ionic current as DNA strain is pulled through a nanopore.

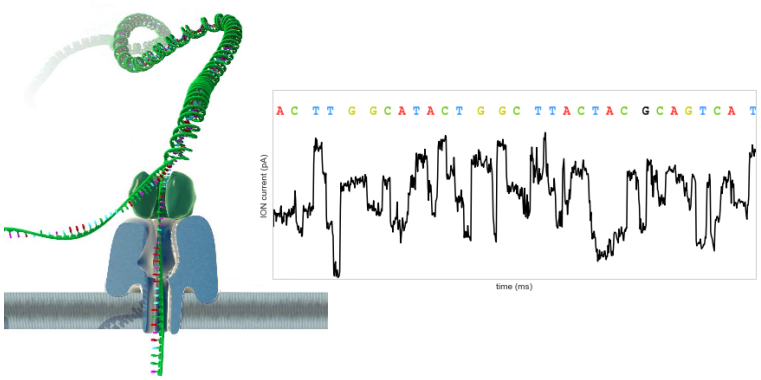

Figure 2.2: DNA strain being pulled through a nanopore [1]

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.

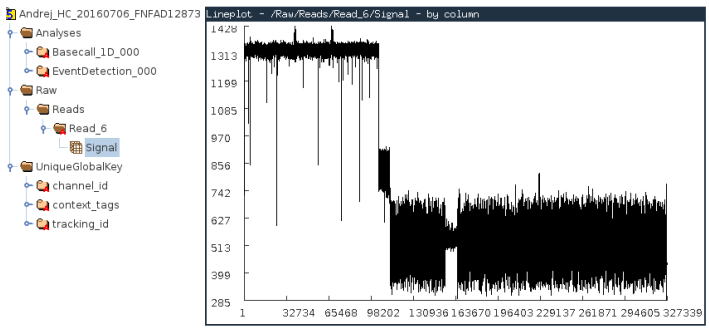

Figure 2.3: Structure of FAST5 file and raw signal line plot show in *HDFView* [2]

Minion offers the possibility of sequencing one or both strands of DNA(1D or 2D reads). Combining information from both strands is results in reads of higher quality, MinION is able to 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 will focus on base calling these kinds of 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 community nanoporetech [3] https://community.nanoporetech.com/

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

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

[3] .

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 Metrichor is actually a name of the company.

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 5 bases being present in the pore and that the context is typically shifted by one base in each step. HMM states 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. Basecalling 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.

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.

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 basecalling. It became publicly available just recently in June, 2017.

2.3.2. Third-party

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.

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

DeepNano[12] was the first open-source basecaller based on neural networks that uses bidirectional recurrent neural networks. DeepNano was written 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 basecalling 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, G).

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

3.1. Arhitecture

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 and that state is updated in each step. The figure shows simple RNN and the same RNN unfolded in time. Unrolling is a simple way of showing how network processes each input in the sequence and updates its hidden state (show in figure 3.1.(

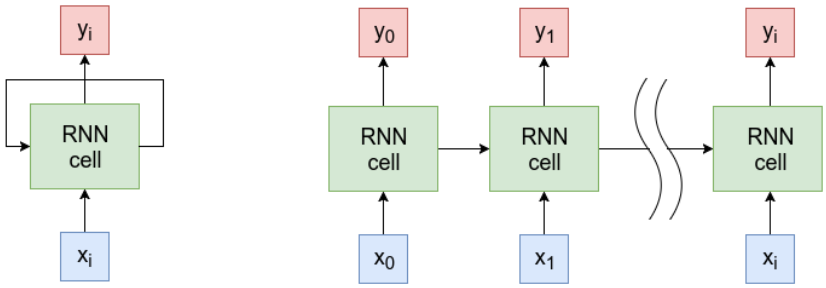

Figure 3.1: An unrolled recurrent neural network

These networks are trained using a variant of backpropagation called backpropagation through time which is essentially the same as classical backpropagation on an unfolded net-

work. 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. Detailed explanation can be found [13]. These issues are called the vanishing and exploding gradient[13] 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. 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 goes 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 have promising results [15][16].

The easiest way to understand a convolution is by thinking of it 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.

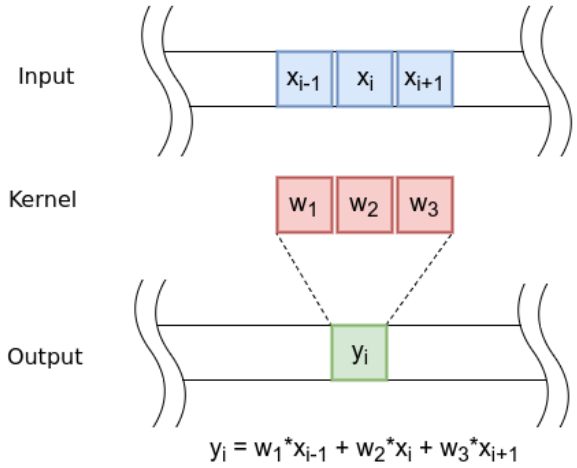

Figure 3.2: Convolution layer, kernel size 3 with with stride 1.

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.

ReLU. The Rectified Linear Unit has become very popular in the last few years. It computes the function f(x)=max(0,x). 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] with .

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

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

Figure 3.3 shows different activation functions.

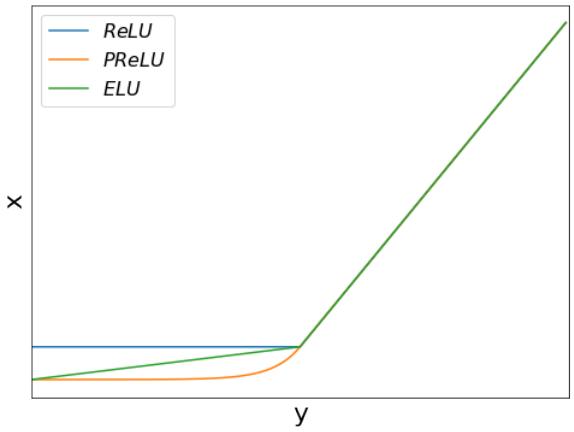

Figure 3.3: Activation functions

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 stride larger stride is used then in the convolution layers, as the purpose of this layer is subsampling. Most ofter, maximum value on the values in the window (max pooling) but other transformations are possible (average pooling, L2-norm, and even stochastic pooling). Figure 3.4 show dimensionality reduction by factor 2 using pooling with kernel size 2 with stride 2.

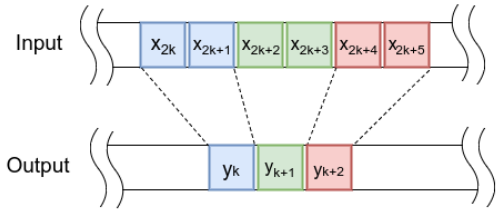

Figure 3.4: Dimensionality reduction by pooling (kernel size 2, stride 2)

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 which makes CNNs more efficient than RNNs.

When compared with RNN, in which output can depend on the entire sequence, but 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). This is the motivation behind deep convolution neural networks and why they are so popular in the field of image processing. Lower layers see limited spatial information and are able to detect simple features like edges and through a series of convolutional layers, later layers can detect more abstract concepts using intermediate features detected from the whole image, or the signal in our case.

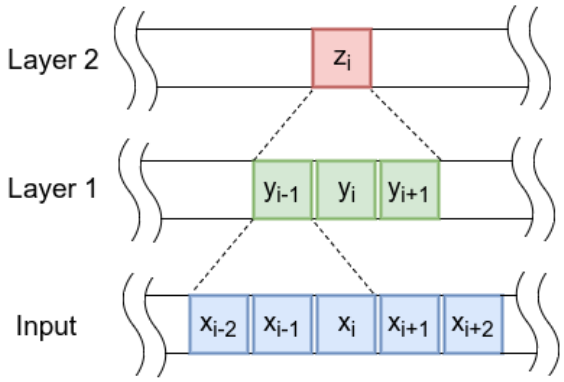

Figure 3.5: Receptive field after 2 layers of convolutions with kernel size 3

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, CNNs have a big speed up over RNNS. This especially goes when dealing with larger sequences[16].

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 allow 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 the 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 other nonlinear function $F(X)$ and linear member X. $F(X)$ is called the residual.

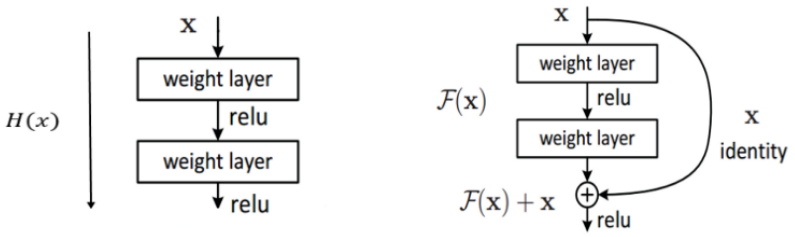

Figure 3.6: Comparison between classical CNN and CNN with residual connection [1]

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

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 and from direct output of the network in some way decode our desired output sequence Y. m is the maximal allowed length of output sequence Y. Idea is that the network outputs is fixed width and the variable length sequence is derived from them. 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)[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, 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.

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

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:

$$P(\pi|X) = \prod_{t=1}^{m} o_t(\pi_t), \tag{3.4}$$

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:

$$P(Y|X) = \sum_{\pi \in decode^{-1}(Y)} P(\pi|X) \tag{3.5}$$

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:

$$L(D) = -\sum_{(X,Y) \in D} lnP(Y|X) \tag{3.6}$$

3.2.3. Output decoding

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

$$Y^* = \underset{Y \in L^m}{\operatorname{argmax}} P(Y|X) = \underset{Y \in L^m}{\operatorname{argmax}} \sum_{\pi \in decode^{-1}(Y)} P(\pi|X), \tag{3.7}$$

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 needed to be found. Calculation of all possible sequences is computationally intractable but exist algorithms that approximate decoding.

One naive possibility is to take the most probable path and say that output sequence corresponds to that path. This is not necessarily true: suppose we have one path with probability 0.1 corresponding with sequence A, and ten paths with probabilities 0.05 each corresponding

to sequence B. Clearly, label B is preferable overall, since it has an overall probability of 0.5; however, this naive best path decoding would select label A, which has a higher probability than any single path for label B.

Better approximations can be calculated using beam search algorithm proposed in paper[21]. Idea behind this approach is tu

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[20] or various 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 layers keep changing. Small change in distribution of outputs in early layers can cause 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. Proposed solution for this is to center each output from activations of the mini-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 is computed on the whole training dataset rather than on mini-batches during training. Pros of using batch normalization Batch normalization offers several advantages other than reducing internal covariant shift. If offers more robust learning process by reducing dependance on scale of the parameters and their initial values. This allows usage of larger learning rates and faster learning all together. It is show in the original paper that batch normalisation also regularizes the model that could potentially improve performance of the model.

4. Implementation

4.1. Datasets

Datasets show in table ref are used. Ecoli as a sample of larger number of reads and lambda as a small example used for testing purposes. Both datasets were previously have passed to MinKNOW and have been basecalled by Metrichor. Baseced data by Metrichor are used to train our model.

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 show data preset in FAST5 file after basecalling by Metrichor. For basecalled event, $model\_state$ field contains possible sequence of bases in the pore and $move$ field defines by how many bases have passed through the pore between two consequitive events.

[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`

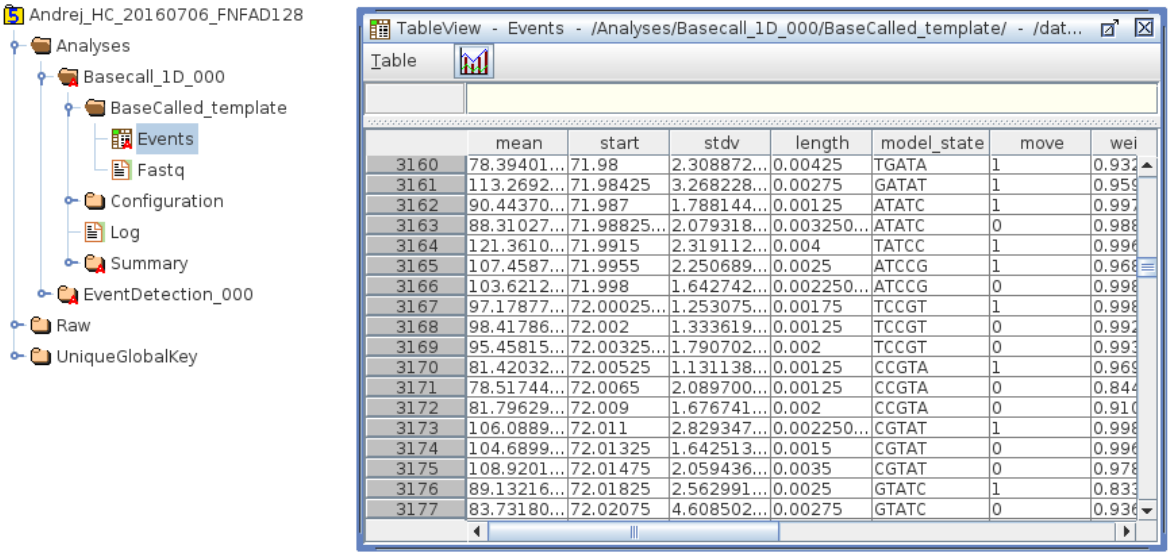

Figure 4.1: Basecall information produced by Metrichor show in HDFView

4.2. 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 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[4] 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 same file format (SAM).

SAMTools[5] and it's Python bindings PySam[6] 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*[7] keeping all dependencies in single container thus eliminating possible conflict between packages on host OS. Nvidia Docker[8] was used for GPU support.

[4] https://github.com/baidu-research/warp-ctc
[5] http://www.htslib.org/
[6] https://github.com/pysam-developers/pysam
[7] https://en.wikipedia.org/wiki/Dependency_hell
[8] https://github.com/NVIDIA/nvidia-docker

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.

4.3. File formats

This section will give brief overview of file formats used and information present in them.

FASTA

FASTA is widely used file format for reference sequences. It is a text-based format for representing either sequences, in which nucleotides or amino acids are represented using single-letters. Most usual file extensions are *.fasta* or *.fa*). Sequence representation consists of header line containing description starting with character >, folowed followed by line(s) of sequence data. Full file specification can be found at *NCBI* sited[9]. 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 usualy stored in FASTQ file format. Each read in the file is stored in followig 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. quality score for each base

Quality score is generated during basecalling that can be used by aligners. 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.

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

```
@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 reference sequences. It is a TAB-delimited text format consisting of header section, which is optional, and an alignment section. Detailed information about SAM specification can be found on *SAMTools* website[10]. Among other information, alignment start position is included, flag stating is read aligned as template of as reverse complement and CIGAR string that describes 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 with letters "=", "X", "I" and "D".

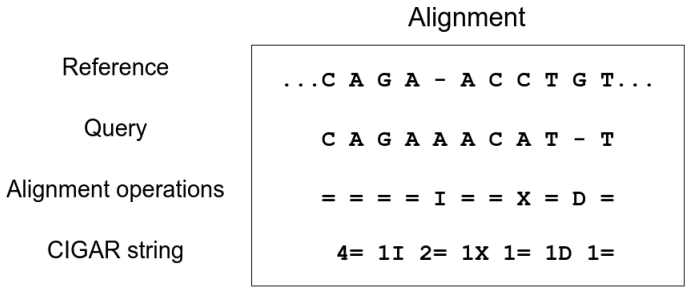

Figure 4.4: Example of simple alignment and CIGAR string

4.4. Data preprocess

To help training process, raw signal is split into smaller blocks (signals of length 1000) that are used as inputs. For Metrichor basecalled event using $start$ field it is easy to calculate the block it falls into. With this, for each block we have output by Metrichor. To correct errors produced by Metrichor and possibly increase quality of data, each read aligned to the reference using aligner GraphMap thet returns best position in the genome, hopefully the part of the genome from which read came from. Alignment part in the genome is used as

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

target. Using cigar string returned by aligner we can correct Metrichor data and with that our test data per blocks. This process is shown in figure 4.5.

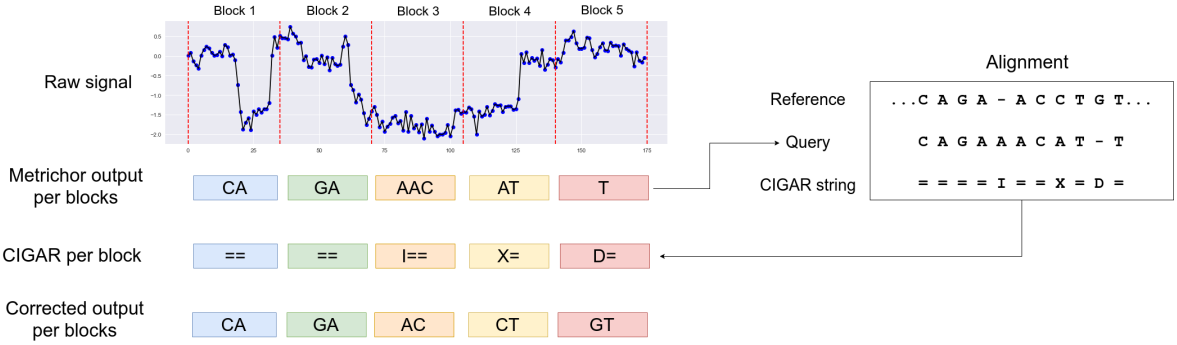

Figure 4.5: Dataset preparation

To eliminate possibility of overfitting to the know reference, model is trained and tested on reads from different organisms. Due to limited ammount of public available nanopore sequence data, ecoli was *devided* into two regions. Reads were split into into train and test portions depending in which region of ecoli they align to. If read aligns inside first 60% of the ecoli it is placed into train set and if it aligns in second portion it is placed into test set. Reads whose aligment overlaps train and test region are not used. Important to note that ecoli,and genome of majority of other bacteris, has cyclical genome so reads with alignment that wraps over edges are also discarded. Total train set consist of bla bla reads. Overview of the entire learning pipeline is shown in figure 4.6.

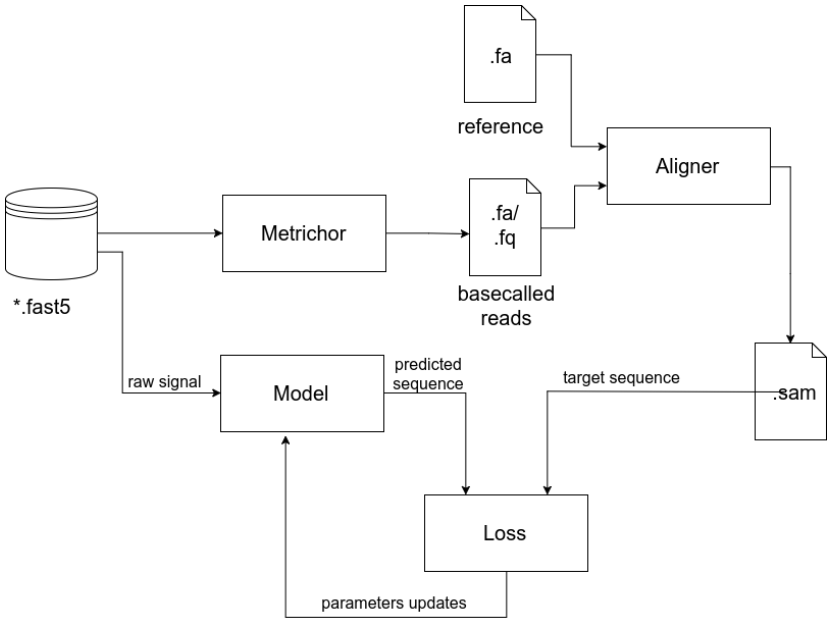

Figure 4.6: Overview of training pipeline

4.5. Deep Learning model

Final model is residual neural network consisting of 72 residual blocks depicted in figure 4.7. This is a variant of arhitecture proposed in paper [24] with the difference of ELU being used as activation instead of ReLU as it is reported[25] to speeds up learning process and improve accuracy as the depth increase. without running more detailed analysis is difficult if this this actually results in performance benefit over simple ReLU .

Each residual block contains 2 convolution layers making total number of convolution layers 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 2 is used every 48 layers resulting in reduction of dimensionality by factor 8. This is used to help training by reducing number of required blank labels in the output and computation effort. This network has 2 milion peremeters that are learned during training.

Training the model is minimization of previously described CTC loss. It was done using Adam, stochastic gradient descent algorithm based on estimation of 1st and 2nd-order moments. It is often used as it offers fast and stable convergention. Default parameters of Adam were used ($\beta_1 = 0.9$ and $\beta_2 = 0.999$. Initial earning was set to 1e-3 with exponential decay. Batch size was set to 8 mostly due to limited hardware resources. As noisy batch that would cause gradients to explode, gradient is clipped to range [-2, 2]. Figure 4.8 shows learning curves during training. Occasional spikes of training loss be explained small batch size and presence of noise in the signal.

Training process is implemented in *producers/consumer* pattern with synchronization

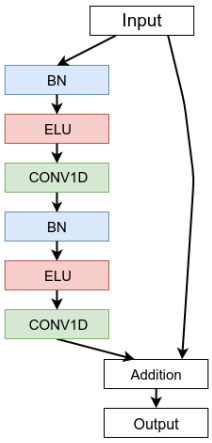

Figure 4.7: Used residual block

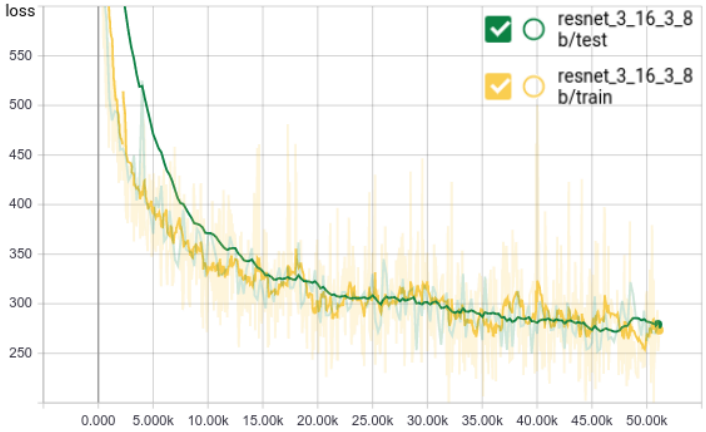

Figure 4.8: Learning curve shown in TensorBoard

and communication done using FIFO[11] queue. Multiple producer threads, running on the CPU, load FAST5 and aligments batch data, preprocess it and convert to Tensorflow 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.

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

5. Results

Developed was compared with other available basecallers that support R9 chemisty. This includes third-party basically DeepNano and official basecallers by Oxford Nanopore (cloud-based Metrichor and NanoNet). Fact that ground truth is not actually known makes evaluation difficult. Two approaches were used to get clearer information about each basecaller.

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. Missmatches, insertions and deletions, in that case, should be due to limitations of sequencing technology and noise in the signal.

To get conclusive results this is done on both test set of the ecoli dataset as well as lambda dataset. 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. Deletions are not part of the read as they are the absence of base that occurs in the reference. Deletion_rate is defined as a number of deletions in the alignment over the length of the aligned read. All expressions can be found bellow.

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

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

$$missmatch\_rate = \frac{n\_missmatches}{read\_lenght} \tag{5.3}$$

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

$$deletion\_rate = \frac{n\_deletion}{read\_lenght} \tag{5.5}$$

$$match\_rate = \frac{n\_matches}{read\_lenght} \tag{5.6}$$

For each basecaller, these rates are expressed as median, mean and variance of all aligned reads. To summarize the results, the median is used as a single value as it is more robust

measure and in the case of skewed distributions like these even more informative. Results are show in the table 5.1 for ecoli dataset and table 5.4 for lambda. Developed model show promising results by having better match rate than the others, smaller mismatch rate. Both dataset show all basecallers being a bit bias towards deletions than insertions but this can be bias of aligner used (GraphMap). To eliminate that possibility, tests were repeated with BWA with almost same results. Results for BWA are show in tables **??** and 5.2. All results are consistent on both datasets using both aligners.

Table 5.1: Alignment specifications of Ecoli R9 basecalled reads

	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: AlignmenBWAA t specifications of Ecoli R9 basecalled reads

	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**

Table 5.3: Alignment specifications of lambda R9 basecalled reads

	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 bwa lambda R9 basecalled reads

	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 rates are shown using histogram plot on figure 5.1 and the KDE (kernel density estimate) plot (figures 5.2 and 5.3). Like the histogram, the KDE plot encodes the density of observations, but curve approximation is used instead of bins. This results in less cluttered comparison. histogram plots for lambda are show on plot **??**. It is important to note that lambda is small dataset (around 80 reads) and more samples are needed to get a better approximation of distribution.

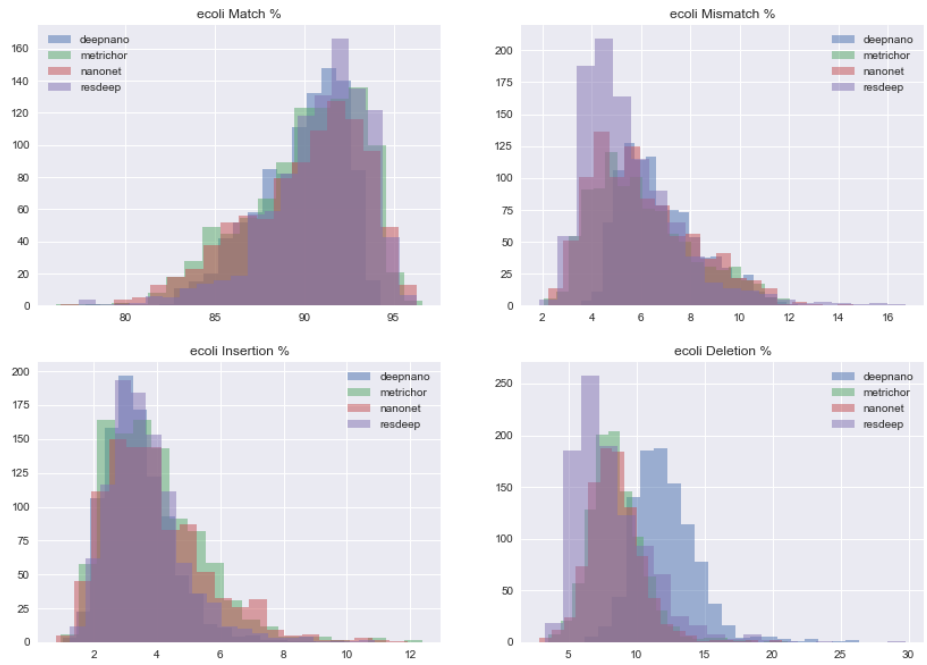

Figure 5.1: Cigar operations histogram over relative position inside read

Histogram on figure 5.4 shows how matches, mismatches, insertions, and deletions are distributed across the reads. It is shown that mismatches and insertion occur more frequently at the ends. This is not only the case for the developed basecaller, but other basecallers show same property. This could be because context information is not available from both sides when edges are base called.

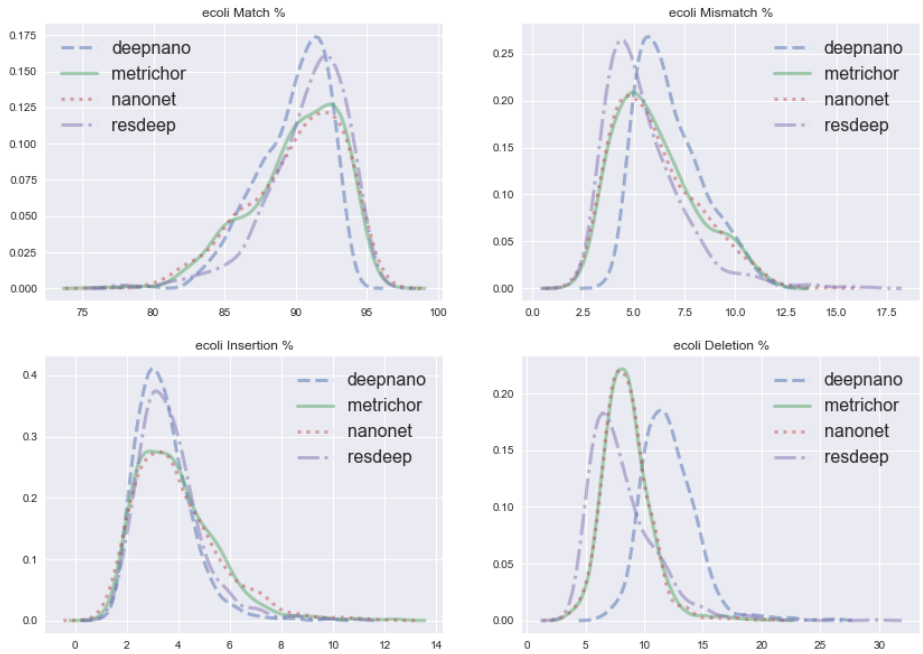

Figure 5.2: Cigar operations histogram over relative position inside read

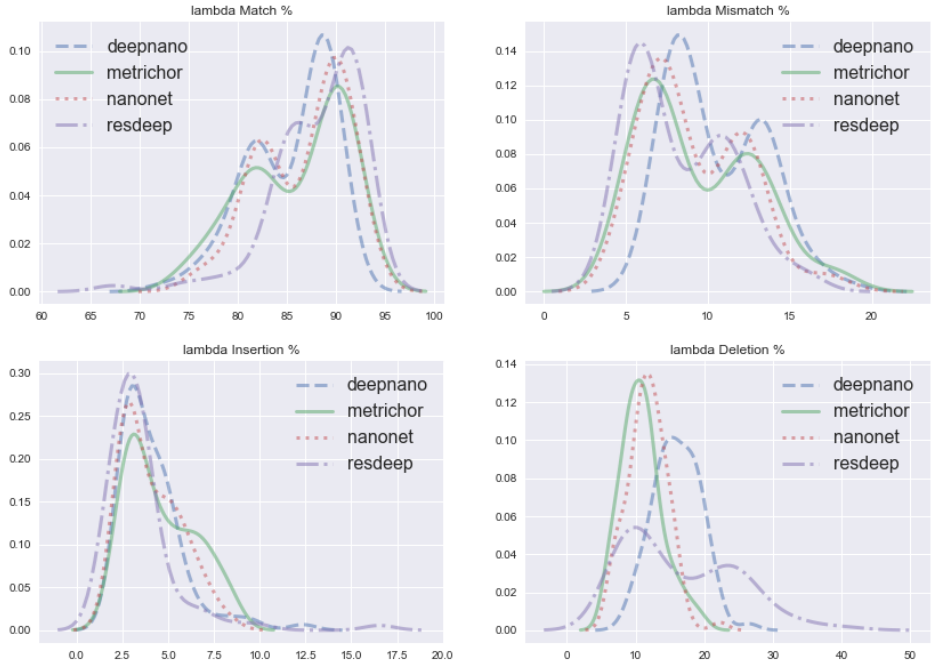

Figure 5.3: Cigar operations histogram over relative position inside read

5.1.1. Consensus

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: the proportion of bases in a sequencing 'read' that align to a matching base in a reference sequence

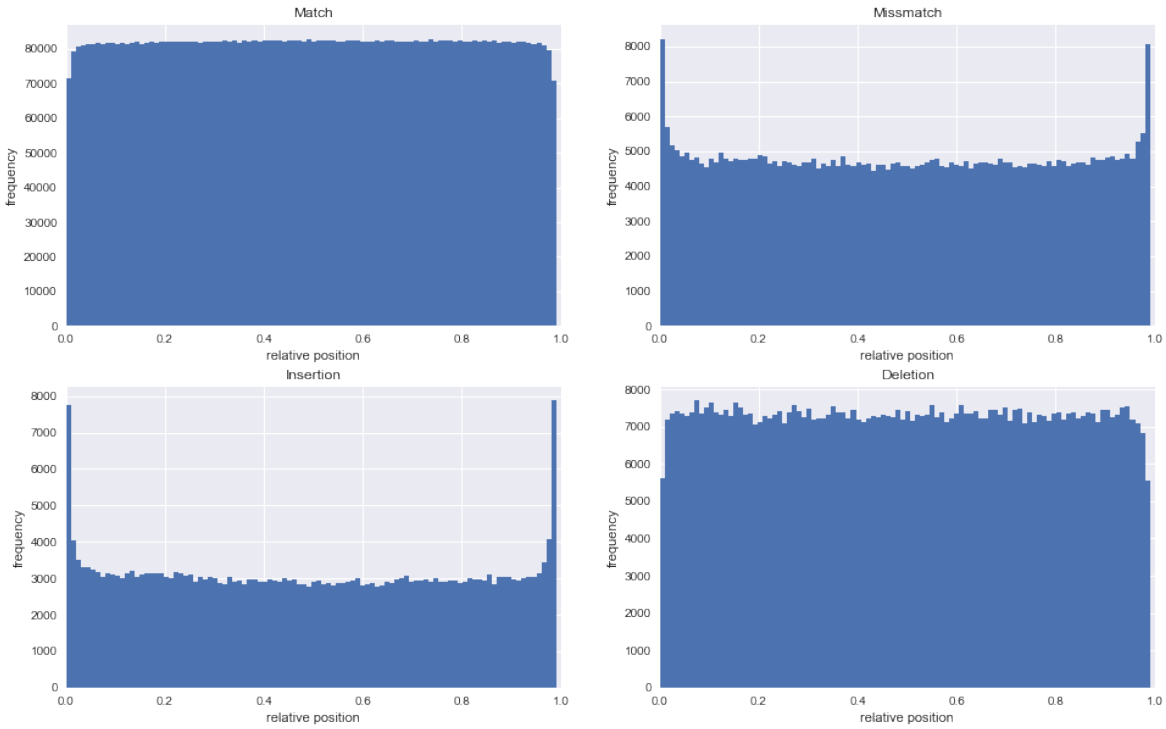

Figure 5.4: Cigar operations histogram over relative position inside of read

Results are calculated for each read in test dataset and median value, mean 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:

$$identity\_percentage = 100 * \frac{n\_correct\_bases}{reference\_lenght} \tag{5.7}$$

$$match\_rate = \frac{n\_correct\_bases}{consensus\_lenght} \tag{5.8}$$

$$snp\_rate = \frac{n\_snp}{consensus\_lenght} \tag{5.9}$$

$$insertion\_rate = \frac{n\_insertions}{consensus\_lenght} \tag{5.10}$$

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

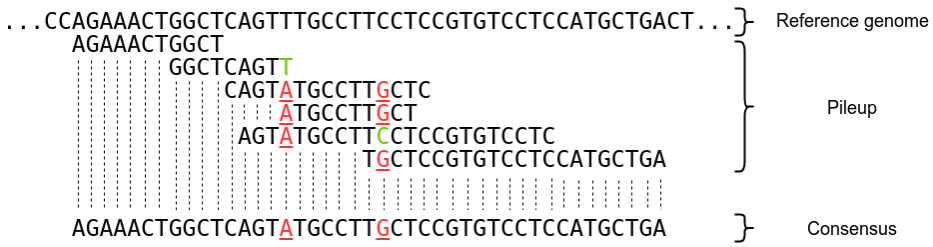

Figure 5.5: Consensus from pileup

5.2. Error rates per read

Table 5.5: Consensus specifications of Ecoli R9 basecalled reads

	Match %	Snp %	Insertion %	Deletion %
deepnano	98.874222	1.004407	0.121371	0.904092
metrichor	99.122300	0.746359	0.131342	0.629992
nanonet	97.969082	1.570034	0.460885	1.515800
resdeep	99.236079	0.647438	0.116482	0.550985

Table 5.6: Consensus specifications of lambda R9 basecalled reads

	Match %	Snp %	Insertion %	Deletion %
deepnano	99.344256	0.643333	0.012412	0.264780
metrichor	99.562607	0.418824	0.018569	0.146485
nanonet	99.442586	0.540898	0.016516	0.196127
resdeep	99.541180	0.440219	0.018601	0.297613

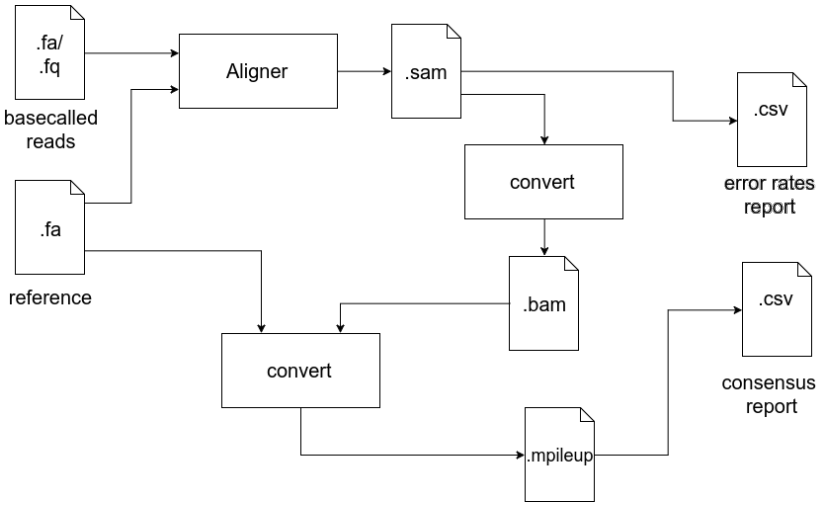

Figure 5.6: Overview of evaluation pipeline

5.3. Read lengths

Interesting thing to analyze are the lengths of basecalled reads for each tool. Developed tool output reads of lengths similar to metrhihor while other tools such as NanoNet for instance, basecall drastically shorter reads. More detail analysis of read length distributions is show using KDE plots on figure 5.7 for both lambda and ecoli.

Table 5.7: 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 5.8: 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

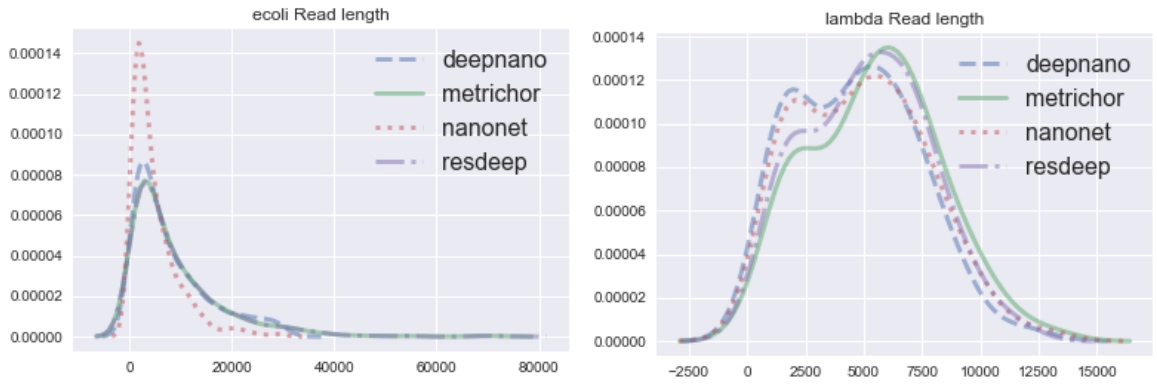

Figure 5.7: Overview of evaluation pipeline

6. Conclusion

Conclusion.

The main reason you might want to consider convolutions in your work is because they are fast. I think that's important to make research and exploration faster and more efficient. Faster networks shorten our feedback cycles.

Most of the tasks I've encountered with text end up having the same requirement of the architecture: Maximize the receptive field while maintaining an adequate flow of gradients

When I set out to write this I only had my own experience and Google's ByteNet to back this claim up. Just this week, Facebook published their fully convolutional translation model and reported a 9X speed up over LSTM based models.

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

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

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.