University of Ghana http://ugspace.ug.edu.gh

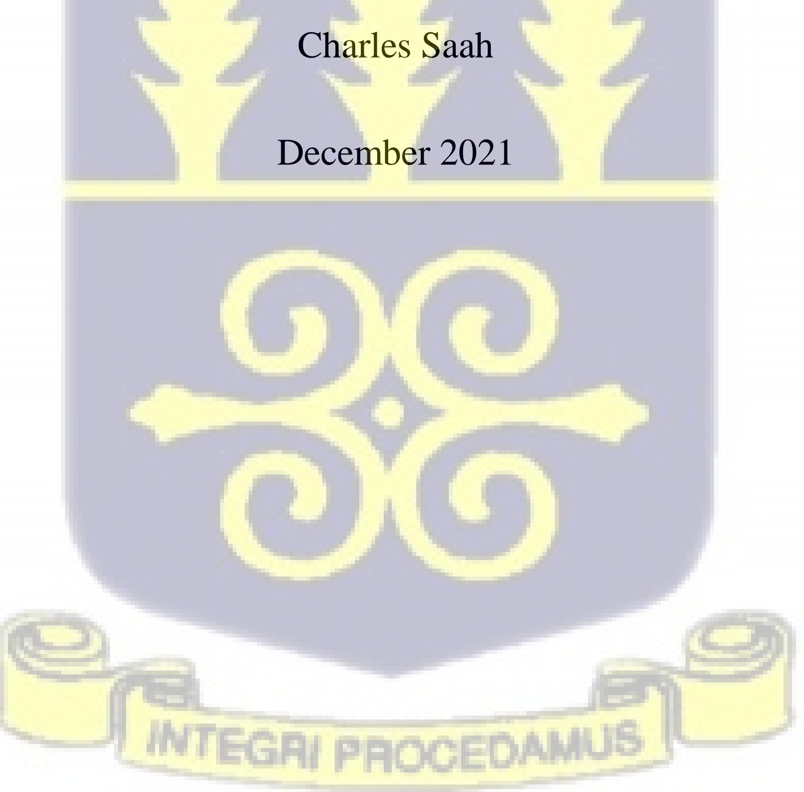
GENOME SEQUENCING ERROR CORRECTION USING DEEP CONVOLUTIONAL NEURAL NETWORK

A THESIS SUBMITTED TO THE DEPARTMENT OF COMPUTER ENGINEERING AND THE COMMITTEE ON GRADUATE STUDIES OF THE UNIVERSITY OF GHANA

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY (COMPUTER ENGINEERING)

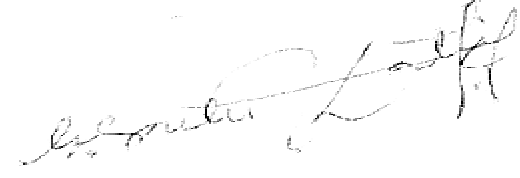
Charles Saah

December 2021



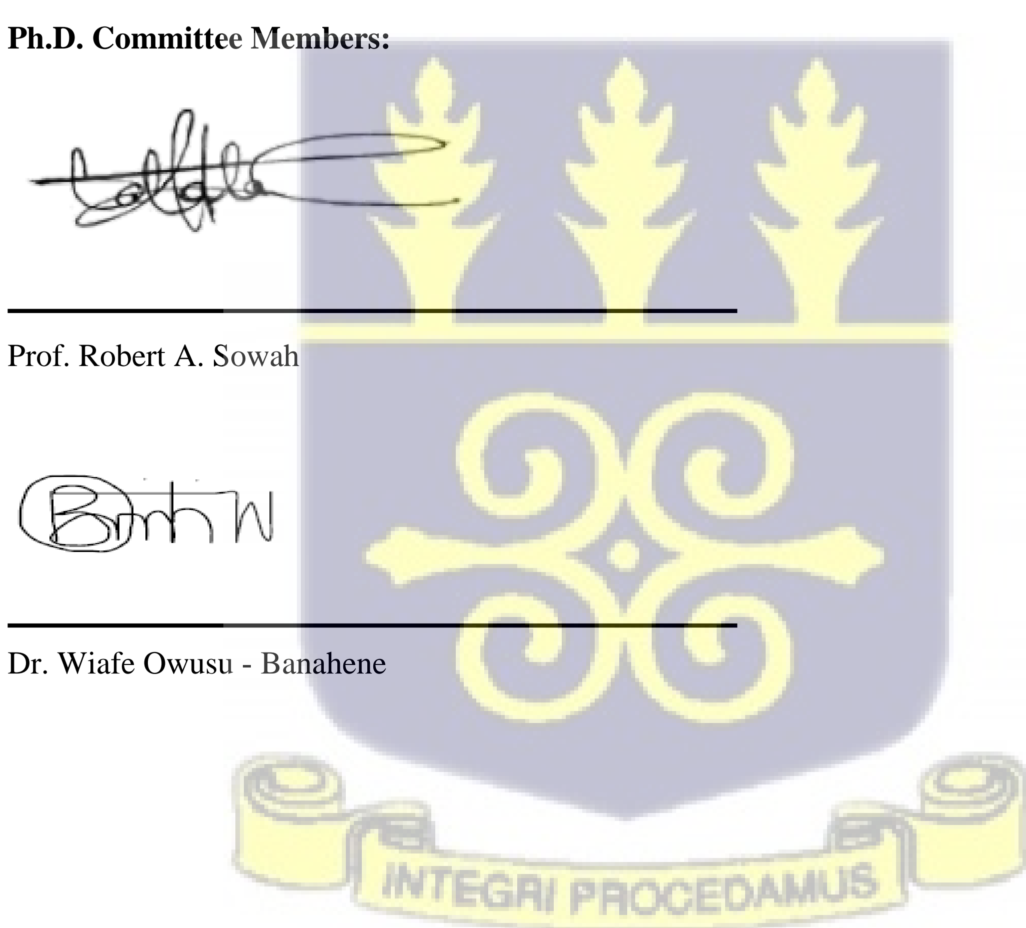
University of Ghana http://ugspace.ug.edu.gh

Supervisors List



Prof. Ferdinand Katsriku – Principal Supervisor

Ph.D. Committee Members:



Prof. Robert A. Sowah

Dr. Wiafe Owusu - Banahene

University of Ghana http://ugspace.ug.edu.gh

Declaration

I, Charles Saah, wish to confirm that the work presented in this thesis is my own. Where information from other sources has been used, I want to state that I have clearly indicated that.

University of Ghana http://ugspace.ug.edu.gh

Acknowledgment

The commencement of my Ph.D. studies to its completion has been very challenging yet insightful for me. This has shaped the way I approach all aspects of work and increased my attention to detail.

I have been privileged to work under Dr. Ferdinand Katsriku as the lead supervisor. His excellent advice, comments, and guidance have led to the work's completion. I would also like to thank the team of beautiful co-supervisors (Dr. Wiafe Owusu-Banahene, Prof. Robert A. Sowah, and Dr.Godfrey A. Mills), whose advice and criticism contributed immensely to the completion of the work.

Last but not least, I would like to express my profound gratitude to my family for the love, patience, care, and understanding throughout the Ph.D. journey

University of Ghana http://ugspace.ug.edu.gh

Abstract

Advancement in technology has led to sequencing a vast number of genomic data. The genome sequencing process has gone through three main generations, and each generation improves tremendously on its predecessor. Quite recently, the human genome has also been successfully sequenced. Varied sequencing platforms have emanated through this technological advancement. Each platform uses a different sequencing procedure. This has led to the introduction of errors into sequenced data. These errors compromise the quality of the sequenced data and any inferences made about them. Many efforts and techniques have been used to try and correct sequencing errors. The issue remains prevalent primarily because the processes or methods used to correct genomic sequencing errors are overly intricate to implement, necessitating a significant amount of computational resources. The error correction process is identified as a pattern recognition problem and proposed a classification (machine learning) approach to solving the sequencing error correction problem. This involves using a deep convolutional neural network that is gentle on computational resources and very fast at correcting insertion and deletion errors. The system is tested using the genome data NA12878, which was obtained from the NBSI, and it achieved 99.5% classification accuracy

University of Ghana http://ugspace.ug.edu.gh

Abbreviations

ASF – Apache Software Foundation

ASIC - Application-Specific Integrated Circuits

BFC – Blocked Bloom Filter Error Correction

BLESS – Bloom filter–based Error Correction Solutions for high throughput Sequencing reads

BP – Base Pair

BWM – Burrows-Wheeler Matrix

BWT – Burrows-Wheeler Transform

CC – Correlation Coefficient

CNN – Convolutional Neural Network

Coral – Correction with alignments

CUDA – Computer Unified Device Architecture

cuDNN – CUDA Deep Neural Network

Cython – Combination of C and Python programming language

DAG – Directed acyclic graph

DataVec – Data Vector

DBG – De Bruijn Graphs

DiskBQcor – Disk-Based Index and Box Queries for Genome Sequencing Error Correction

University of Ghana http://ugspace.ug.edu.gh

DNA - Deoxyribonucleic Acid

DCNN – Deep Convolutional Neural Network

ECHO – Error Correction Algorithm

ETL – Extract Transform and Load

FIONA – A parallel and automatic strategy for read error correction

FM Index – Ferragina and Manzini index

FMLRC – FM-index Long Read Corrector

FN – False Negative

FP – False Positive

FPGA – Field Programmable Gate Arrays

HECIL – Hybrid Error Correction Algorithm for Long Reads with Iterative Learning

HiTEC - High Throughput Error Correction

HMM – Hidden Markov Model

GPU – Graphic Processing Unit

IP – Image Processing

JVM – Java Virtual Machine

Karect – Kaust Assembly Read Error Correction Tool

K-mer – Sequence Read of length K

University of Ghana http://ugspace.ug.edu.gh

KMC – K-Mer Counting

KLD - Kullback Leibler Divergence

LASSO – Least Absolute Shrinkage and Selection Operator

LIGHTER – Fast and Memory-Efficient Sequencing Error Correction Without Counting

LoRDEC – Long Read DBG Error Correction

LR – Long Read

LSTM – Long Short-Term Memory

MAE – Mean Absolute Error

Matplotlib – Python Plotting Library

MBE – Mean Bias Error

MCPU – Multiple Processors

MLC++ - Machine Learning with C++ Programming Language

MLF – Machine Learning Framework

MLP – Multiple Layer Perceptron

MSA – Multiple Sequence Alignment

MSE – Mean Square Error

MSLE – Mean Square Logarithmic Error

ONT – Oxford Nanopore Technology

University of Ghana http://ugspace.ug.edu.gh

ONNX – Open Neural Network Exchange

OpenCL – Open Computer Language

OpenCV – Open Computer Vision

NGS – Next Generation Sequencing

NLA – Natural Language Processing

Numpy – Numerical Python

ParLECH – Parallel Long-read Error Correction using Hybrid methodology

PRC – Polymerase Chain Reaction

RACER – Rapid and Accurate correction of Errors in Reads

RBM – Restricted Boltzmann Machine

RECKONER – Read Error Correction based on KMC

RNA – Ribonucleic Acid

SBS – Sequencing By Synthesis

SGS – Second Generation Sequencing

SHREC – Short Read Error Correction method

SR – Short Read

SVM – Support Vector Machine

SVR – Support Vector Machine Regression

University of Ghana http://ugspace.ug.edu.gh

Tanh – Hyperbolic Tangent Function

TGS – Third-Generation Sequencing

TP – True Positive

TN – True Negative

WEKA – Waikato Environment for Knowledge Analysis

University of Ghana http://ugspace.ug.edu.gh

Contents

Supervisors List ii

Declaration iii

ACKNOWLEDGEMENT iv

ABSTRACT v

Abbreviations vi

List of Figures xiv

List of Tables xvi

CHAPTER 1 Introduction 1

1.1 Deoxyribonucleic Acid 1

1.2 Sequencing Platform 5

1.2.1 First-Generation Sequencing 6

1.2.2 Second generation Sequencing – SGS – (Next Generation Sequences - NGS) 7

1.2.3 Third generation sequences (Long Read Sequencing) 14

1.2.4 Analysis of Genome Sequencing Errors 18

1.3 Read Alignment 19

1.4 Problem Statement 21

1.5 Objectives 24

1.6 Scope of Study 25

1.7 Significance of the Research 25

1.8 Structure of the Thesis 26

CHAPTER 2: Literature Review 28

2.01 De Bruijn graph (Multi-graph) 30

2.02 Bloom Filter 31

2.03 Suffix Arrays 33

2.04 Suffix tree 35

2.05 Burrows-Wheeler Transform (BWT) 38

2.06 FM – Index (Full-Text substring Index) 39

2.07 Hash Table 41

2.1 K – spectrum-based method 44

Limitation of K-Spectrum Error Correction Method 48

University of Ghana http://ugspace.ug.edu.gh

2.2 Multiple Sequence Alignment – MSA 50
Limitation of Multiple Sequence Alignment Error Correction Method 53
2.3 Suffix tree/array-base 53
Limitation of Suffix Tree / Array Error Correction Method 56
2.4 Hybrid Error Correction Method 57
Limitation of The Hybrid Error Correction Method 59
Error Correction Using Machine Learning 59
2.5 Theoretical Framework of Machine Learning 62
Neural Networks and Deep Learning 62
Multiple Layer Perceptron 64
Activation Functions 65
Sigmoid – Logistic Regression Activation Function 67
Tanh Function 68
Rectified Linear Unit – ReLU 69
Softmax 70
Feed Forward 71
Backpropagation Algorithm 72
Loss Function 74
Regression loss functions 75
The Mean Squared Error (L2 Loss) 75
Mean Absolute Error 77
Mean Squared Logarithmic Error (MSLE) 77
Huber Loss / Smooth Mean Absolute Error 78
Log-Cosh Loss 78
Classification Loss 78
Cross-Entropy (Log Loss) 79
Hinge Loss 80
Kullback Leibler Divergence Loss 80
Focal Loss 81
Exponential Loss 82
Avoiding overfitting 83
Convolutional Neural Network (CNN) 84

University of Ghana http://ugspace.ug.edu.gh

Machine Learning Frameworks 85
Hyperparameter Optimization 87
Gradient Descent Optimization 92
Bayesian Hyperparameter Optimization 96
Evaluation of Deep Learning Frameworks 97
Optimization of Distributed Deep Learning Frameworks on GPUs 99
Chapter 3. Methodology 105
Theoretical Analysis of Genome Sequencing Error Correction Using DCNN 105
Data Pre-processing and K-mer Generation 106
3.1 System Design 109
3.2 System Implementation 110
Chapter 4: Results and Discussion 119
4.1 Results 119
4.2 Result Analysis 124
4.3: Discussion 126
Chapter 5. Conclusion and Future Work 130
5.1 Introduction 130
5.2 Conclusions 131
5.3 Contribution to Knowledge 131
5.4 Observations 132
5.5 Recommendations 133
Appendix 1 – Dataset 135

University of Ghana http://ugspace.ug.edu.gh

List of Figures

2.1 De Bruijn graph ...31

2.2 Suffix tree generated by the kmer CAGTCAGG...37

2.3 Hash table for 3-mer...41

2.4 Appending of first 3-mer with offset 0 to the hash table...42

2.5 Addition of second 3-mer with offset 1 to the hash table...42

2.6 Hash table showing collision at bucket...43

3.1 Perceptron ...55

3.2: Network graph...58

3.3 Activation function of a neural network...58

3.4 None-linear, commonly used activation function...59

3.5 Diagram of the Leaky ReLU activation function...63

3.6 Feedforward is a fully connected layer...44

3.5 Convolutional Neural Network...65

3.7 Hinge loss function...72

3.8 Convolutional Neural Network...74

4.1 Convolutional Neural Network using one-hot-encoding scheme...77

4.2 Fully connected convolutional neural network with four hidden layers...77

INTEGRI PROCEDAMUS

University of Ghana http://ugspace.ug.edu.gh

4.3 Initial training and validation of the network showing divergence in the validation after epoch 5……………………………………………………………………………………….78

5.1 Improved network validation process after tweaking of hyperparameter…………..…………79

5.2 Network accuracy check diagram….…………………………………………………….79

5.3 Normalized confusion matrix……………………………………………………………….80

University of Ghana http://ugspace.ug.edu.gh

List of Tables

2.1 Kmer hashing table…………………………………………………………………....33

2.2 Kmer filtering table………………………………………………………………....33

2.3 Kmer position vector……………………………………………………………......34

2.4 Suffix table ……………………………………………………………………….........34

2.5 Suffix array……………………………………………………………………………….35

2.6 Index array………………………………………………………………………….......35

2.7 Genome data and index table …………………………………………………………….36

2.8 Fast reversion genome table ………………………………………………………...37

2.9 The Burrow–wheeler matrix of ACAACAGT$..38

2.10 The Burrow–wheeler transform of ACAACAGT$...38

2.11 The Burrow–wheeler transform converted to suffix array……………………….........39

2.12 FM index table extending the Burrow wheeler transform…………………………………40

University of Ghana http://ugspace.ug.edu.gh

CHAPTER 1 Introduction

1.1 Deoxyribonucleic Acid

The deoxyribonucleic acid (DNA) (also referred to as the genome) is the basic structure of every organism. The DNA is a double-stranded molecule [1], and each strand of the DNA will be treated mathematically as a string spanning the four alphabets $\sum=\{A, C, T, G\}$, where each $x \in \sum$ will be referred to as nucleotide consisting of basis that is represented by Adenine (A), Cytosine (C), Thymine (T) and Guanine (G). The bases are complementary where Adenine Complements Thymine and Cytosine Complements Guanine. This means that the complementing bases are found on the opposite ends of the nucleotide strand. The complementary bases are referred to as base pairs (bp). The length of a DNA sequence is a measure of the total number of nucleotides (when dealing with one strand) or the total number of base pairs (when dealing with the complementary strand) [2]. The task of finding the correct order or arrangement of the nucleotide bases in a DNA strand is referred to as DNA or genome sequencing.

The phi X174 consisting of 5,386 nucleotides, became the first organism whose DNA was sequenced and sequenced as a single-stranded genome [3]. At the time, the sequencing processes were not automated, producing only a few sequences at a time. Therefore, it was difficult for the sequencing procedure to be extended to other (larger) organisms. Genome sequencing became a dominant research area, and more research efforts to improve the genome sequencing process were set in motion. This led to the discovery of some semi-automated sequencing techniques, which was an improvement over its predecessor sequencing procedure but, these techniques could also produce only a few (hundreds) nucleotides at a time [4][5]. The use of capillary array electrophoresis alleviated the challenge of genome sequencing by enabling the development of

University of Ghana http://ugspace.ug.edu.gh

high throughput sequencing, which was commercialized and became known as the Sanger genome sequencing project [6].

The Sanger project was the first to introduce the concept of sequencing genomic data. The project is known to have several issues, such as spanning accuracy, performance, and time-lapse. Various strategies have been employed to resolve the problems emanating from the Sanger project. A new sequencing DNA method emerged in the quest to overcome the Sanger sequencing problems. It became known as the second generation of genome sequencing, where genome data is divided into several smaller fragments called short reads [7][8]. The reads were realigned after the fragmentation and analysis were done on them. There were challenges in the realignment process; aligning reads to their original position [9]. The division of the reads into smaller fragments leads to repetitions which are transferred into the dataset [10]. The shorter reads and the realignment process also introduced repetitions in some of the positions during the realignment process. Fragmentation of the datasets, significantly the larger datasets, amplified the realignment error due to the repetition of the alignment errors.[11]. The schematic representation of the Sanger sequencing process is shown in Figure 1.1. (a) represents the in vivo cloning or amplification (fragmentation) of the input data (bacterial vector), (b) is the primer extension of blocked and labeled nucleotide, and (c) represents the separation of electrophoreses and the reading out of labels.

University of Ghana http://ugspace.ug.edu.gh

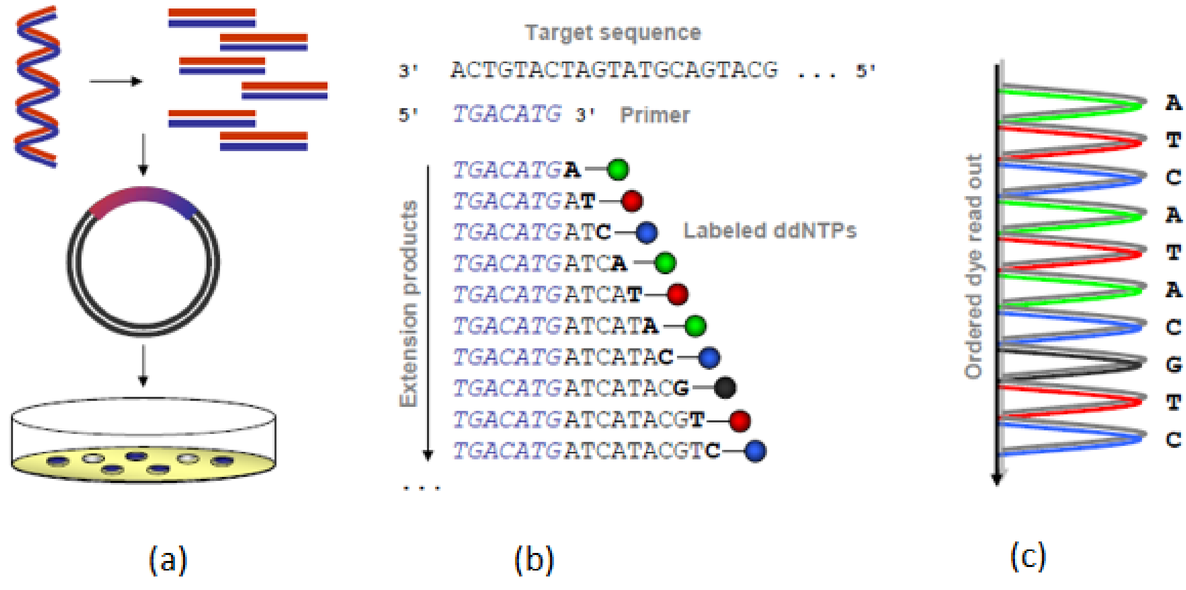


Figure 1.1 Sanger sequencing process. Source: X. Yang et al [12]

The error emanating from the repetitive sequencing data generated from the second-generation sequencing technique became a concern to both researchers and industries that use such information. This led to further research into improving the second-generation sequencing process, the third-generation (next-generation) sequencing process emerged as a better alternative to the second-generation sequence. The third-generation sequencing process uses read lengths that are longer than their predecessor (second-generation sequence)[7]. Many research domains (platforms) that use third-generation sequencing techniques emerged. Some of these platforms are – AB, Rocher/454 [12], Illumina [13], Ion Torrent [14], SOLID [11], Oxford NanoPore [15], and the PacBio[16]. Although third-generation sequencing is noted to correct the sequencing errors that emanate from second-generation sequencing, sequencing data from the third-generation sequencing platforms were also noted to contain some errors. The third-generation sequencing errors were smaller compared to their second-generation counterpart. The third-generation sequencing platform's major challenge is that the longer read length produced larger sequenced

University of Ghana http://ugspace.ug.edu.gh

datasets that were computationally difficult to handle. This means that more computer resources were needed to handle the third-generation sequencing data.[17]. However, further probes indicate that all the following sequencing platforms exhibited some errors and were more computationally intensive than the second-generation sequence [18][19]. Various sequencing platforms generated different error rates on a given dataset. Both researchers and industry have used second and third-generation sequencing techniques to generate genomic data and inferences drawn on the generated genomic data. This means that the inferences were drawn on the genomic dataset that contains some error level. The inferred error may vary depending on the sequencing platform used in generating the dataset. This calls for further probe into both the dataset and inferred conclusions [20]

Genome Sequencing Errors

Genome sequencing errors are inaccuracies or discrepancies that occur during the generation of generation of genome sequences. Various factors including sample preparation processes, limitations of genome sequencing technologies, or computational algorithms that are used in the data analysis processes contribute to the generation of genome sequencing errors. These are insertion error, deletion error, and substitution error. Understanding the different types of genome sequencing error types is important for developing effective error correction methods. Using Figure 1.1.1 The three common genome sequencing error types will be explored with the assistance of a reference genome at various positions. The reference genome is represented by the second row with a green background colour. It is made up of the nucleotide basis "GTCAAGTTCG" at positions 1,2,3,4,5,6,7,8,9,10. Read 1 is made up of nucleotide basis "GTCAAGTATCG" spanning 1 to 11 which is one base longer than the reference genome. It can be seen that position 8 in Read 1 has "A" instead of T as compared to the reference genome, but

University of Ghana http://ugspace.ug.edu.gh

shifting "TCG" at positions 8, 9, and 10 respectively to positions 9, 10, and 11. Making Read 1 one base longer than the reference genome. This means that Read 1 has an insertion error ("A") at position 8 which is indicated by the colour orange (Figure 1.1.1). Read 2 has a length of 9 instead of 10 when compared with the reference genome. "TTCG" at positions 7, 8, 9, and 10 in the reference genome has shifted to positions 6, 7, 8, and 9. Positions 1 to 5 in read 2 contain the same nucleotide basis as the reference genome. This means that "G" which should be at position 6 is missing. This is a deletion error and it is indicated by the colour yellow (Figure 1.1.1). For Read 3, except position 5, which is indicated by the colour blue (Figure 1.1.1), all the other nucleotide bases match with that of the reference genome. The length of Read 3 is 10 which is the same as the length of the reference genome. This makes "C" at position 5 of Read 3 a substitution error.

Position	1	2	3	4	5	6	7	8	9	10	. . .	
Reference Genome	G	T	C	A	A	G	T	T	C	G	. . .	
Read 1	G	T	C	A	A	G	T	A	T	C	G	. . .
Read 2	G	T	C	A	A	T	T	C	G	. .		
Read 3	G	T	C	A	C	G	T	T	C	G	. . .	
. . .											. . .	
Type of error		Substitution				Deletion				Insertion		

Figure 1.1.1: Structure of Genome Sequencing Error Types

1.2 Sequencing Platform

The advancement in technology such as computer vision, digital signal processing, and machine learning in general, whose application was further extended to genomics, led to the improvement of genomics research [21]. Although the initial direct application of the new technological

University of Ghana http://ugspace.ug.edu.gh

advancement in the area of genomics was challenging, many research establishments emerged, which further enhanced the genome sequencing processes. These genome sequencing establishments became known as genome sequencing platforms. These sequencing platforms extended the genome sequencing process to cover the genome of organisms that were hitherto difficult or near impossible to sequence. The sequencing processes carried out by the individual research institutions led to what became known as the generation of genome sequences.[6]. The generation of genome sequencing processes based on the approach or methodology used were labeled as the first, second, and third-generation sequencing processes, with each generation improving on either the time or resources required by its predecessor to sequence genomic data[22]. The first, second, and third generations of genome sequencing techniques are treated in sections 1.2.1 to 1.2.3.

1.2.1 First-Generation Sequencing

Using data that emanated from Rosalind Franklin and Maurice Wilkins, Watson, and Crick were the first to conceptualize the 3-dimensional structure of DNA in 1953 [23]; the extension of their pioneering work led to the improvement of genome sequencing by providing the ability to replicate genomic data and the possible encoding of protein into nucleic acid for the first time.[24]. Because the approach was new and needed further understanding and improvement to ease its application in genomics, it was initially focused on microbial ribosomal species, which are single-stranded bacterial species[25]. Although the approach (Sanger) had a challenge in producing very high output and accurate data at the time, its improvement led to the discovery of the genome of new species, which was difficult to do under the Sanger method

University of Ghana http://ugspace.ug.edu.gh

The successful sequencing of genomic data by Frederick Sanger, which became known as the first generation of sequences, used the chain termination process[26]. Under the Sanger project, targeted regions of DNA studied were transcribed using enzymes (a process known as DNA polymerase). These produced longer read, highly accurate lengths (up to 99.99%) but lower throughput, making them expensive to grow [27]. Under the chain-termination sequencing process, the DNA under investigation is first heated to separate the complementary strand, which is used as a template. Other primer sequences are brought in line with the template[27]. This allows primer sequences to bind with their complementary sequences in the generated template containing the DNA. The process is terminated by introducing dideoxynucleotide into the sequencing chamber. This is then followed by performing capillary gel electrophoresis. Under the capillary gel electrophoresis, smaller pieces of DNA move faster than longer ones. The various bases in the DNA emit light when they move. This enables the multiple components of the DNA under study to be detected. One of the major setbacks of the Sanger project is the need to generate capillary samples for every DNA that has to be sequenced and also produce clonal populations for easier reference. Although this may seem trivial, generating capillary samples for larger DNA samples will be quite challenging and time-consuming [21]. The procedure could only be used to create the genome sequence of bacteria whose DNA was smaller. This then called for a better and an efficient way to sequence genomic data [6]. The second-generation sequences came to solve the issues in the first generation of sequences.

1.2.2 Second generation Sequencing – SGS – (Next Generation Sequences - NGS)

INTEGRI PROCEDAMUS

Research that was aimed at correcting the problems found in the first-generation sequencing process led to the discovery of the second-generation sequencing process. Since the major problem in the first-generation sequencing process had to deal with longer read lengths and the preparation

University of Ghana http://ugspace.ug.edu.gh

of capillary or jell lanes for each DNA under study, it makes it extremely difficult to consider sequencing the DNA of larger organisms [28]. The second-generation sequencing process generated shorter read lengths compared to its first-generation counterpart and did not require capillary or jell lanes to sequence genomic data [29]. This method is computationally expensive which can be measured using Feldman and Crutchfield expense C_{LMC} estimator, which the product of Shannon's cross-entropy values H(s) and the probability that each of the nucleotides emanates from a uniform distribution referred to as disequilibrium D(s) and a lot of disk space (up to 250Gb using 168 processor core or up to 512GB using 48 processor core) is needed to sequence the dataset successfully [30][31]. Before the birth of the second-generation sequencing process, various intermediary research processes and transitions such as process parallelization (where multiple sequences were processed at the same time) and automated capillary electrophoresis (chemical analysis) were developed [32]. Although these sequencing processes were promising, the contributions by 454 Life Sciences (also referred to as Roche), Illumina, and Ion Torrent were better compared to all the others. They adopted multiple sequencing processes using multiple arrays. This increased the number of genomic data that can be sequenced at a go. [33].

Roche 454 became the first to commercialize the genome sequencing processes in the second generation of genome sequencing. This increases the number of organisms whose DNA was sequenced [34]. They continued the terminate chain amplification process; however, they brought on board a new sequencing technology known as pyrosequencing instead of dideoxynucleotides. On the addition of nucleotides, the pyrosequencing detected the release of pyrophosphate and the emission of light. A process that does not rely on the chain termination method under the Sanger project[35]. The DNA molecule is segmented into single strands and passes through beads, allowing the amplification of segmented strands on flat plates with multiple wells that represent

University of Ghana http://ugspace.ug.edu.gh

test tubes (picotiter plates) [36]. The segmented strands are then joined from opposing ends to add new nucleotides. The molecules then bind to the nucleotide sequence[6]. After which, amplification is done. This leads to the production of many complementary strands, which are further passed over the beads and loaded into the wells to generate the output[37]. The initial output of this sequencing process was 50 bases. However, the introduction of a new enzyme in the sequencing process increased the production to 700 bases per sequence. Figure 1.2.2a below shows 424 pyrosequencing sequencing processes were to determine a nucleotide from a group, one of the four nucleotides is sequentially washed over the group to cause polymerases that attract its complement nucleotide. This process terminates when the longest available nucleotide is synthesized.

In the first flow cycle, when the single nucleotides T, A, C, and G were washed over the target sequence, T, T, and C were respectively complemented. When the identical nucleotides (T, A, C, G) were washed over the targeted sequence in the second flow cycle, A, A, and C were complemented. In the third flow cycle, when the identical nucleotides (T, A, C, G) were washed over the targeted sequence, G, was complimented. Finally, in the fourth flow cycle when the nucleotides (T, A, C, G) washed over the targeted sequence, A, G, G, G, G, C were completed. This means that in the first flow, only A and G were complimented, with A being complemented twice. Likewise, only T and G were complemented in the second flow, with T being complemented twice. In the third flow, only C was complemented and in the fourth flow, T, C, and G were complemented with C being complemented four times. The sequence which is generated is highlighted in the yellow indented eclipse.

INTEGRI PROCEDAMUS

University of Ghana http://ugspace.ug.edu.gh

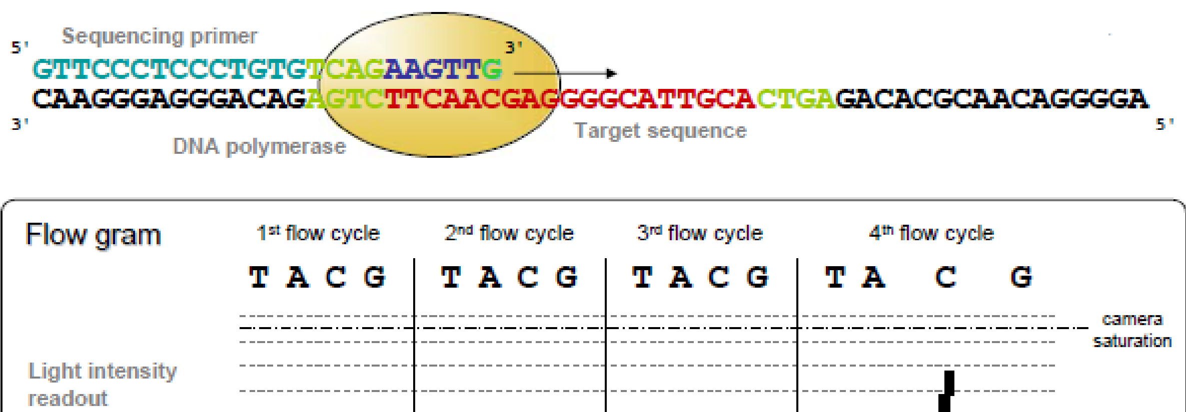


Figure 1.2.2a: A four-cycle pyrosequencing sequencing process where one of the nucleotides is sequentially washed over copies of the sequence to be determined. Source: C. Ledergerber et al [39]

The Ion Torrent genome sequencing platform used semiconductor chipsets similar to those found in cameras to produce a very scalable and faster sequencing process, which was cheaper than the other platforms, [38]. The essence of the digital camera chipset is to provide sensing layers to capture and transform light beams emitted by the DNA molecules into digital signals. The capturing and translation process is increased by the number of wells on the chipset[38]. The entire process of capturing and translating DNA information into digital information is automated. This speeds up the segmentation and attachment of beads as well as the capturing and translation process. The automation makes the entire sequencing process efficient and allows the extension of the sequencing procedure to cover larger organisms whose DNA was difficult or impossible to sequence before introducing the Ion torrent platform. Many samples of one of the four DNA nucleotides are introduced into the chip [39]. A hydrogen ion is released whenever a nucleotide is added to the single-stranded DNA. The hydrogen ion release is based on the presence of complementary DNA in the well. This means that the absence of complementary DNA in the well

University of Ghana http://ugspace.ug.edu.gh

will cause no hydrogen ions to be released, and since the release of hydrogen is associated with voltage change, no voltage change will be recorded when there is no complementary DNA in the well. When there are complementary DNA elements in the wells and two or more similar nucleotide bases are found closer, two or more nucleotide bases will be added. This increases the hydrogen ion released and simultaneously increases the voltage that will be recorded along with the wells[40].

Four major processes have to be considered to successfully use the Illumina sequencing technology to sequence genomic data [37]. The first step is what has become known as “sample preparation5”. Under this method, the DNA or RNA under consideration is changed into an Illumine adaptor library. The second step transforms the prepared samples into clusters that provide the opportunity to amplify the various clusters. The third step sequences the amplified clusters by synthesis. The fourth step is devoted to analyzing the sequenced data from the third phase.

Many library kits for sample preparation from either Illumina or third-party vendors can be purchased in the open market since Illumina is an available system[41].

The rationale behind library preparation is to have small DNA fragments with Illumina adapters hooked up at either end[42]. Transposons are simultaneously tagged onto the input DNA with adaptors in the fragmentation process. During a reduced cycle amplification, the adaptors are ligated, providing extra motifs, usually comprised of regions containing complementary flow cell oligo and indices of primary binding sites[43].

The individual fragments are amplified in the flow cell using a lined glass slide. The process then proceeds to the clustering stage, where two oligo-encoding schemes are applied to each lane [44]. The first oligo is used to hybridize the content in the cluster on the surface. The oligo then creates

University of Ghana http://ugspace.ug.edu.gh

a fragment that is complementarily hybridized [45]. When the hybridization process is completed, the original template is washed away after separating the double-stranded component. The next process is the clonal amplification of the strands using a bridge amplification process. This allows the individual strands to fold, thus allowing the adapter region to hybridize. The hybridized area becomes equal to the second oligo on the flow cell[46]. From the polymerase, complementary strands are created to form a double-stranded bridge. After which, the double-stranded bridge is separated. The bridge separation process is repeated, producing two single-stranded copies of the molecule bonded to the flow cell. After a while, all the fragments become clonally amplified [47]. After the bridge amplification, the forward strands are retained, but the reverse strands are then cleaned. To prevent unnecessary binding from other primers, the cleaned primers are blocked.

The genome sequencing process started by extending the first primer that emanated from the sequencing process. This produces the first sequencing read. The process is repeated, and each time, four nucleotides that have been fluorescently tagged are added to the growing chain[48]. The sequence of the template determines the base that is added. After adding the nucleotides, the clusters are passed through a light source, leading to the emission of characteristic fluorescent signals. Another name for this kind of sequencing is sequencing by synthesis. The length of the various reads determines the number of cycles required to sequence those reads. The base call of the sequencing process is dependent on the intensity of the generated signal strength and the emission wavelength. In any given cluster, similar strands are read simultaneously. A parallel sequencing process is used to sequence most of the clusters. The read product is washed away after the first read sequence [49]. This is followed by template hybridization after the introduction of the first index read primer. After completing the index read, the read product is no longer needed, so It is then discarded by washing it off and discontinuing the protection placed on the three

University of Ghana http://ugspace.ug.edu.gh

primers. This releases the template to connect to the second set of oligo in the flow cell. The process that led to the first read index is repeated for the subsequent read indices.

After this, a double-stranded bridge is formed when the polymerase extends the second flow oligo. The blocking of the three primers follows this after the linearization of the double-stranded DNA. Read one is ended by holding onto the reverse strand after separating and washing away the forward strand. This is followed by read two introducing the read two sequencing primer; which goes through a similar process to the first read, subsequent reads follow the same underlining process to obtain the desired read length**.** Reads representing the entire fragments are generated. The sample preparation generates unique indices to represent the various libraries that are used. using the index numbers, sequences that have very low index numbers are identified as coming from low libraries and are thus separated from the others. The base call structure is also used to cluster sequences. Sequences with similar base call structures are grouped into contiguous sequences. This allows the pairing of forward and reverse reads. [50]. To determine the success or otherwise of the sequencing process, the generated contiguous sequences are realigned to the reference genome. This enables the identification of any variances that might be introduced during the sequencing process. The alignment with the reference genome also provides the opportunity to correct any ambiguous alignments. This generates several short reads or k-mers. If a k-mer of size K is generated from a genome of size S, then the total number n of all possible k-mers derived from S would be given by the expression [(S – K) +1]. For example, given the nucleotide CTAGGATGACTACG (S=14) and 8-mer (K=8), then 7, 8-mers [14 – 8 +1 = 7], i.e. (CTAGGATG, TAGGATGA, AGGATGAC, GGATGACT, GATGACTA, ATGACTAC, TGACTACG) would be produced. It becomes computationally challenging when larger datasets

University of Ghana http://ugspace.ug.edu.gh

such as the human genome which contains about 3 billion base pairs (bp) when sequenced under the SGS method with k-mer length in the region of $20 \leq k \leq 100$ bp [51]

1.2.3 Third generation sequences (Long Read Sequencing)

The second-generation sequencing process simplified the way genomic data is sequenced by reducing the hitherto longer read length and transforming the dominant chain-termination processes under the first-generation sequences (Sanger) into a high throughput sequencing technique. This provided the opportunity to sequence the DNA of larger organisms. [52]. The second-generation sequencing process is based on the initial amplification of the DNA molecule, which is followed by sequencing by synthesis. The entire fluorescence signal that comes from synthesizing many amplified identical DNA strands helps provide inference for the various nucleotide types. With the continuous introduction of random errors during the amplification process, the DNA synthesis within the amplified interfaces increasingly becomes out of sync with the interface. Since the signal is dependent on the viability of the synthesizing process, its quality is affected. To rectify this anomaly, longer reads are identified and further segmented to produce shorter reads. This identification and segmentation process may also negatively impact the time taken to sequence DNA under the second-generation sequencing method. This may also impact on the accuracy of some results that may be obtained using the second-generation sequencing process.

To solve the large sequencing errors identified under the second generation of sequencing, the third generation of sequences was born. Unlike its second-generation counterpart, the third-generation sequencing process does not require breaking the genome into smaller reads to be followed by amplification and synthesis. The third generation of sequences produces longer reads compared to the second generation. Although they are an improvement over the second-generation

University of Ghana http://ugspace.ug.edu.gh

sequence methods, the third-generation sequencing method also produces read errors. Unlike the second-generation read errors that can be corrected through read alignments, the errors emanating from the third-generation sequences are difficult to correct. It was therefore not convenient to use them in de-novo genome assembly.

Most third-generation sequencing platforms use single-molecule sequencing. The work of platforms such as Helicos, Pacific BioScience, and the Oxford nanopore dominated the third generation of the sequencing era.

The Helicos platform does not use PCR amplification when sequencing genomic data. This means that the time of preparing the sequences is reduced, and, more importantly, sequencing error is reduced tremendously. The reason is that sequencing errors are also amplified during the PCR amplification process. During the sequencing process, DNA molecules are divided into smaller lengths of between 100 to 200. The tail of Adenine is attached to the three prime portions of every segment. This allows the attachment of fluorescently labeled nucleotides. From there, the inactive oligo-T- nucleotides that are complementary poly-A-primer on the surface of the flow cell are then hybridized with the components [53]. After this process, After labeling nucleotides with DNA polymerase and fluorescent, they are then fed into the flow cell. The nucleotide bases are sequentially added to the complement of the fragments on the templates. This approach is similar to the capturing and measuring of the emitted signal fluorescent in the Illumina flow cell process. Unlike the recording of signals in clusters in the Illumina sequencing platform.

Helicos records the signal as single fragments, which makes the sequencing fragments independent and thus reduces sequencing errors that may migrate from other fragments if they were sequenced in clusters. Helicos outperform other sequencing platforms due to their non-reliance on PCR amplification and sequential capturing sorting of nucleotide bases.

University of Ghana http://ugspace.ug.edu.gh

Pacific BioSceinces sequencing technology requires the attachment of fluorescence to the molecule at the initial stages; this creates phospholinked nucleotides with a different color for each of the nitrogenous bases[54]. The DNA polymerase is separated from the fluorescent label after attaching the fluorescent label to the terminal phosphate nucleotide when bases are added. The process triggers the emission of light, and a nano-photonic chamber captures and records the emitted light. The SMRT sequencing platform uses a specialized waveguide that consists of a metal film that is coated with silicon dioxide which makes it possible to detect single molecules. Simultaneously, the chains in the well are replicated by the DNA polymerase. This gives Pacific BioSceinces sequencing technology an edge over similar longer-read-length sequencing technologies.

In all the methods described above, the most challenging but important aspect is the ability to process every data or information that will be produced. Although several algorithms are used to process such data, the two commonly used algorithms are the primary local alignment tool (BLAST) and the cluster-weighted algorithms (CLUSTALW). Each employs a different strategy when sequencing genomic data. BLAST uses search heuristics which makes it faster while CLUSTALW uses multiple sequence alignment. The multiple alignment process requires a lot of time to realign genomic data. This makes CLUSTALW slower when compared to BLAST.

The Oxford nanopore sequencing platform as the name depicts, very small but hollow tube which is used to trap molecules as they pass through. The tubes are fused with a high electric-resistant synthetic membrane. If an electric potential is applied to the membrane while immersed in the solution, the ionic current is generated due to the resistance of the membrane, the generated current passes through the nanopore. Since the various nucleotide bases emit different light intensities when they come into contact with a source of potential energy, the introduction of either DNA or

University of Ghana http://ugspace.ug.edu.gh

RNA into the nanopore causes light of varied intensities to be emitted. The emitted light intensity is captured and the nucleotide of the emitted light is determined.

As the sequences are fed into the nanopore chamber, sequenced genomic data is produced for each cycle that the sequence goes through the chamber. Initially, the DNA or RNA dataset to be sequenced is treated to have a leader (Y) and a Chaplin (hp) adapter after which the ligation takes place. Enzymes are then introduced to the adapted sequences to augment their flow through the nanopore with ease.

For the flow through the nanopore to be successful, each DNA or RNA is associated with a tethering protein before enabling it to pass through the nanopore. When the tethering is completed, components of the DNA or RNA attach themselves briefly to the nanopore to start the sequencing process. For the sequence to be successful, an enzyme is introduced to prompt the 5' end of the Y adapter to split the double-stranded DNA to allow only a single strand of the DNA to go through the nanopore each time.

For a successful sequencing process under a nanopore, as the charged template goes through the nanopore on reaching the hp adapter, the complementary strand of the charged template also passes through the nanopore in reverse order. This enables both the segmented DNA and its complement to be sequenced at the same time. The process is repeated each time genomic data is to be sequenced under the Oxford nanopore platform. Under the nanopore platform, the entire sequence length of genomic data is read each time the data needs to be sequenced.

University of Ghana http://ugspace.ug.edu.gh

1.2.4 Analysis of Genome Sequencing Errors

The first-generation sequencing technologies, such as Sanger sequencing, generated relatively long sequencing reads with low error rates. Base calling accuracy and quality score assessments were the dominant genome sequencing error correction methods adopted. Quality filtering techniques and statistical models were usually used to identify and correct sequencing errors. This type of error correction generated high-quality consensus sequencing.

The dominant second-generation sequencing methodologies, including Illumina sequencing, 454 Pyrosequencing, Ion Torrent Sequencing, Roche 454 Sequencing, and SOLiD (Sequencing by Oligonucleotide Ligation and Detection) transformed the field by producing massive amounts of short sequencing reads at a much lower cost. However, these innovations introduced new and higher sequencing error correction challenges due to the generation of short sequencing reads. To correct such errors, alignment, and statistical models have to be employed to align the sequences with a reference genome to identify the errors using processes such as sequence context, quality scores and read depth. Consensus correction processes such as clustering and majority voting were used to generate consensus sequences from overlapping reads.

The third-generating sequencing methodologies such as PacBio and Oxford Nanopore sequences improved on the short reads from the second-generation and produced long reads. The long reads also generated higher error rates than their second-generation counterpart. The errors produced here are either random systematic or both, which made it difficult to correct them. Advanced error correction techniques that take advantage of machine learning processes such as deep neural networks as well as recurrent neural networks have to be adopted in correcting the error. The main reason is that these machine-learning techniques can capture complex dependencies and learn error patterns from raw data which leads to improved accuracy and corrected genomic sequences

University of Ghana http://ugspace.ug.edu.gh

In summary, genome sequencing error correction procedures have evolved together with sequencing technologies which address specific challenges that are associated with particular generations. Advancements in computational resources, algorithms, and machine learning techniques have enhanced more accurate error correction processes leading to the generation of high-quality genomic data for diverse genomic applications including predictive and personalized medicine

1.3 Read Alignment

This is the process of putting together a set of sequencing reads with a reference genome's help. This helps compare two different individuals of the same species and might show that they have a very similar genome. The process entails searching the entire reference genome to find where the sequencing read match is so close. This is done repeatedly for every single read. It is also essential to find the exact location in the reference genome where an exact match is found or the match's offset. The process will be very challenging if the reference genome is large, and the read lengths are longer. The naïve exact matching algorithm tries to remedy the situation by matching the reads throughout the entire genome and determining the match's offset; however, the algorithm requires extensive memory utilization and is not appropriate for use in situations where the read length is long and the reference genome extensive. The Boyer-Moore algorithm and many of its variants are intelligent algorithms that skip or ignore reads with pointless alignments. It does this by tying left-to-right order and trying character comparisons in right-to-left order. Upon mismatch, it skips alignments until a mismatch becomes a match or moves past the mismatch character. The approach is summarized as the bad character rule where there is a shift until a bad character becomes a good

University of Ghana http://ugspace.ug.edu.gh

one and the good suffix rule where the good suffix is reserved, after which the shift is done until there is a match.

The Boyer Moore preprocesses patterns and builds a loop-up table which it uses in the future if anonymous data are presented to it. It is initially time-consuming or costly to build the loop-up table, but the repetitive referential use of the loop-up table is amortized over many problems. Preprocessing algorithms are referred to as off-line algorithms, and those that do not preprocess data are referred to as the online algorithm. The choice of an online or offline algorithm for alignment is based on the type of data being used. If there are many variations in the dataset, then the online algorithm is the best choice for such a dataset. On the other hand, if there are no variations in the dataset, then the offline algorithm is best suited for such a dataset. In trying to remedy the huge memory requirement for the read alignment problem with long reads, the indexing approach was envisaged. The indexes contain a list of k-mers and their positions. This is a form of lookup table where k-mers and their locations are easily referenced instead of scanning through the entire genome for such information. This data structure approach allows for querying the dataset [55]. Although the indexing approaches reduce memory usage tremendously and produce faster querying time, there is also the potential of missing some of the k-mers that would hitherto not have been missed. This provides the need for approximate matching instead of exact matching techniques. In the approximate matching techniques, one or two positions in the k-mer might not produce the exact nucleotide, but that mismatch might be treated as an error. This may be considered an insertion, deletion, or substitution error. Another instance of this mismatch might be that the reference genome is not the same as the genome under study, which is a natural variation between the genomes. To help identify the nucleotide that is causing the mismatch, the distance factor was introduced. That is, the distance between them becomes a tool to ascertain which

University of Ghana http://ugspace.ug.edu.gh

nucleotide is causing the mismatch. The Hamming distance works on two strings (nucleotides) of the same length. The hinges on the minimum number of substitutions required to turn one of the nucleotides into the other. The Levenshtein (edit) distance has also been of prominence in trying to correct mismatches in reading alignments. The edit distance, unlike the Hamming counterpart, is the minimum number of edits (substitution, deletions, or insertions) required to turn one string into the other. The major difference between the two procedures is that, for edit distance, the nucleotide need not be of the same length as is the case in Hamming distance, and again in the edit distance, there is the option of using insertion, deletion, and substitution to correct the matching process, unlike the use of the only substitution in the Hamming distance.

1.4 Problem Statement

There have been three major transitions or phases under the genome sequencing process. Each of the phases uses major technological advancement to improve upon the strategies adopted by its predecessor. The Sanger project which is the first phase of the genome sequencing process, at the time was a major technological breakthrough come genome sequencing. This is because it was the first time the genome of an organism (though small) has been completely sequenced. It uses the chain termination approach to sequence genomic data.

These nucleotides have a hydrogen group instead of the typical hydroxyl functional group at the 3' end of the nucleotide. The sequencing process began with the DNA being broken with restriction enzymes creating fragments replicated through the polymerase chain reaction (PCR). The elements are placed in solutions and heat-denatured, while primers near DNA strands are fixed to a template. Either these primers or another nucleotide is labeled using fluorescence or a radioactive element.

University of Ghana http://ugspace.ug.edu.gh

This solution is then placed in four different tubes, each containing a different nitrogen base. Each tube contains the DNA polymerase of all four nucleotides and one of the four ddNTPs. The latter of which serves to randomly terminate the replication chains by preventing one nucleotide from bonding to them. Various lengths of DNA are created by this process, and they are all denatured again. Finally, the denatured DNA is run on a polyacrylamide gel tray, where electrostatic forces cause the fragments to travel across the gel.

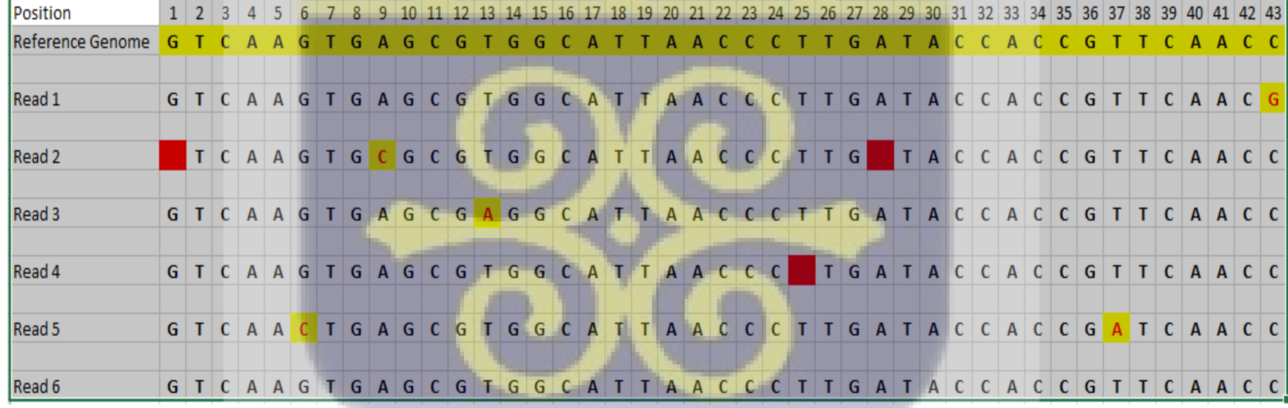
The second-generation sequencing methods, which rely on sequencing by ligation, or sequencing by synthesis, reduced the cost associated with the first generation of sequences by parallel sequencing. However, the continuous use of DNA fragments and amplification meant that shorter read lengths were used. The shorter read lengths restricted the type of experiments or datasets that can be sequenced by the NGS [56]

To sequence genomic data, Next Generation Sequencing (NGS) divides DNA molecules into many smaller fragments called k-mers. The actual sequencing technology determines the size of a kmer. Several million overlapping k-mers are generated for a given genomic data and later reassembled after the k-mer generating process using an overlapping graph to reproduce the underlining genome. The reassembling process is saddled with the generation of erroneous data [1] [2]. The incorrect data generated by some sequenced regions might have very low reads and an attribute of sequencing technology [3][4]. Therefore, the assembler is unable to generate the complete and continuous genome that was fed into it as input but instead ends up generating multiple – unordered sequences referred to as contigs [5]. Rearranging the contigs into their particular order and orientation is computationally intensive and a herculean task. Scaffolding methods have been employed to reduce the contigs, which will invariably reduce the error rate [6][7]. The sequencer introduces three types of errors, notably, substitution, insertion, and deletion (or indel) [8][9].

University of Ghana http://ugspace.ug.edu.gh

Illumina/Solexa are some of the NGS that are noted to produce substitution errors and have received much attention [7][10][11]. Roche's 454 and Ion Torrent are significant contributors to indels [12][13]. Shorter read lengths may not necessarily map up to or uniquely align with the reference genome, causing some repetitive regions to be unmappable. Areas of higher levels of diversities may also not align with the reference genome. The read repetitions also allow repetitions of errors that drastically reduce the sequencing process's accuracy [57]. Error application is a significant drawback of the second-generation sequencing process.

Read lengths emanating from third-generation sequences were more extended than those of the NGS. This further reduced the sequencing time and enhanced the detection of copy number variation [58]. Third-generation sequences are known to have lower read accuracy as compared with second-generation sequencing technologies [19] Our work is going to focus on correcting indels.



Position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43
Reference Genome	G	T	C	A	A	G	T	G	A	G	C	G	T	G	G	C	A	T	T	A	A	C	C	C	T	T	G	A	T	A	C	C	A	C	C	G	T	T	C	A	A	C	C
Read 1	G	T	C	A	A	G	T	G	A	G	C	G	T	G	G	C	A	T	T	A	A	C	C	C	T	T	G	A	T	A	C	C	A	C	C	G	T	T	C	A	A	C	G
Read 2		T	C	A	A	G	T	G	C	G	C	G	T	G	G	C	A	T	T	A	A	C	C	C	T	T	G		T	A	C	C	A	C	C	G	T	T	C	A	A	C	C
Read 3	G	T	C	A	A	G	T	G	A	G	C	G	A	G	G	C	A	T	T	A	A	C	C	C	T	T	G	A	T	A	C	C	A	C	C	G	T	T	C	A	A	C	C
Read 4	G	T	C	A	A	G	T	G	A	G	C	G	T	G	G	C	A	T	T	A	A	C	C	C		T	G	A	T	A	C	C	A	C	C	G	T	T	C	A	A	C	C
Read 5	G	T	C	A	A	C	T	G	A	G	C	G	T	G	G	C	A	T	T	A	A	C	C	C	T	T	G	A	T	A	C	C	A	C	C	G	A	T	C	A	A	C	C
Read 6	G	T	C	A	A	G	T	G	A	G	C	G	T	G	G	C	A	T	T	A	A	C	C	C	T	T	G	A	T	A	C	C	A	C	C	G	T	T	C	A	A	C	C

Figure 1.4.1 A 43-ker reference genome with 6 read alignments showing 8 insertions and 3 deletion errors.

Comparing the various reads with the reference genome in Figure 1.4.1 above, read 1 has G inserted at position 43 instead of C. Similarly read 2, has deletions at positions 1 and 28, it also has C inserted at position 9 instead of A. Read 3 has A inserted at position 13 instead of T. Read

University of Ghana http://ugspace.ug.edu.gh

4 has deletion at position 25. Read 5 has insertions of C at position 6 and A at position 37 instead of G and T respectively. Read 6 shows the deletion of nucleotide A from position 21.

The error correction problem has been approached as a classification problem using a DCNN. The DCNN will split the data into two classes (correct and erroneous dataset). The DCNN is trained to identify and correct the errors in the dataset with very high accuracy. The natural language processing approach has been employed, and it has:

a. Treated the reads (k-mers) as strings of words when correcting the error
b. A model has been constructed with performance superior to similar networks using the same dataset but with fewer computing resources (memory and processor speed)

Next, some literature in the chosen area will be reviewed, this will be followed by the methodology, then the result, discussion, and conclusion

1.5 Objectives

This thesis's general objective is to reduce genome sequencing errors (indels) by treating the genome sequencing process as a big data phenomenon and proposing an effective learning algorithm to train the deep convolutional neural network (DCNN) to sequence the genomic data. The particular objectives are:

1. To build an efficient DCNN model that consists of convolutional layers with correction filters and bounded by activation function to enable faster computation, better training stability, and improved performance.
2. Propose a novel natural language procession network that increases the detection of sequencing errors with higher accuracy.

University of Ghana http://ugspace.ug.edu.gh

3. The network is flattened after introducing non-linearity with the ReLU activation function to reduce the dimensionality of the DCNN, minimizing the utilization of computing resources like memory and processor cores. Then, the dropout function is introduced to reduce the loss between the expected output and the derived output value.

1.6 Scope of Study

The scope of the thesis will focus on designing a novel DCNN architecture to correct indel errors. The design process will incorporate various convolutional layers interspersed with pooling strategies, flatten, dropout, and activation functions to optimize the DCNN model. As a result, complicated sequencing error patterns will be efficiently captured. To enhance the model's performance against the correction of different sequencing error types, An investigation will be conducted into a novel data augmentation strategy that involves generating different read alignment lengths with the reference genome. A preprocessing technique is employed to improve the model's performance in the correction tasks. Extensive benchmarking experiments will be conducted on the DCNN-based error correction and compare its performance with existing methods to showcase the superiority of the model to other models based on efficiency and accuracy

1.7 Significance of the Research

Accurate genome sequencing is important for clinical diagnosis and the identification of genetic variations that may be associated with diseases. Our DCNN-based error correction approach can significantly improve the accuracy of genomic data thereby reducing false positives and false negatives in variant calling. The increased accuracy can lead to more reliable diagnoses and better-informed treatment decisions, eventually benefiting patients and healthcare providers. Our study concentrating on genome sequencing error correction contributes to more trustworthy genomic

University of Ghana http://ugspace.ug.edu.gh

data, which will enable improved detection of disease-causing mutations and prediction indicators. Since therapies in precision medicine are tailored to suit an individual's genetic makeup. A patient's treatment may be more efficient and individualized as a result of the genomic data's increased precision. Typically, genomic studies examine a substantial number of distinct populations. Our technique can speed up the error-correction process, making it possible to analyze huge amounts of genetic data more quickly. This would make it possible for researchers to carry out comprehensive genomic analyses with more computing capacity, which will yield important insights into the diversity of genetics and associated disorders.

Amidst sequencing errors, rare genetic diseases that may emanate from rare mutations may be challenging to detect. Our method can improve the accuracy of variant calling, thereby increasing the chances of identifying such rare disease-causing mutations. This will boost the understanding of the genetic bases for such disease-causing variants which will lead to the development of potential treatments. Drug discovery and development by pharmaceutical companies rely heavily on genomics. Our method can generate accurate genomic data which will enable researchers to have a better understanding of drug targets and patient responses. This has the potential to accelerate the drug development process and enhance the success rate of targeted therapies

1.8 Structure of the Thesis

The thesis is organized into seven chapters. Chapter 1 gives the general introduction of the thesis. It focuses on the general and academic need for correcting genome sequence errors.

Chapter 2 covers the literature review section, which describes the background theory and correction of genome sequencing error methods. This spells out the various genome sequencing error correction techniques that have so far been used.

University of Ghana http://ugspace.ug.edu.gh

Chapter 3 presents the methodology used to sequence the genome data and how the sequencing error was reduced. It explains the network architecture, the encoding function, and how the network was optimized to minimize the sequencing error.

Chapter 4 presents the Results and Discussion.

Chapter 5 presents the Conclusion

University of Ghana http://ugspace.ug.edu.gh

CHAPTER 2: Literature Review

In this chapter, we will take a look at the various methods that have been used in correcting genome sequencing errors. Genome sequencing error correction has attracted contributions from diverse research areas. The diverse research application areas can be grouped into four. These are the multiple sequence alignment (MSA) methods, the K-spectrum method, the Suffix tree or Suffix array method, and the hybrid method. Aside from the suffix tree and suffix array that use similar methods in handling genomic data, each of the other methods uses a different approach to handling and processing genomic data. [59].

The De Bruijn graph has been the basis for most of the error correction methods. An insight into the way the De Bruijn graph works will help us understand the error correction processes. We will therefore provide a detailed analysis of the De Bruijn graph before looking at the error correction methods. The K-spectrum, multiple sequence alignment (MSA), Suffix tree or Suffix array, and the hybrid methods will be respectively reviewed in chapters 2.1, 2.2, 2.3, and 2.4

Under the K-spectrum, the genomic data is first divided into smaller bits usually of varying lengths. These smaller bits are referred to as reads or K-mers (see Figure 2.0). This allows a filtering algorithm such as the Bloom filter (explained in section 2.02) to separate error reads or K-mers (Figure 2.0 – blue color) from true or distinctive reads. An attempt is then made to correct as many as possible the detected error reads by passing them through a hash table (table 2.3 – 2.6). The hash table uses a preset frequency to correct read errors. If the frequency of an initially classified error falls within the threshold, it is reclassified as distinctive and moved out of the error list.

University of Ghana http://ugspace.ug.edu.gh

K-mer position	K-mer length	Genome																			
		A	T	G	A	T	T	G	C	T	A	T	T	A	G	A	C	G	G	C	C
1	6	A	T	G	A	T	T														
2	7		T	G			T	G	C												
3	7			G	A	T	T	G	C	T											
4	7				A	T	T	G	C	T	A										
5	8					T	T	G	C	T	A	T	T								
6	9						T	G	T	C	C	T	T	A	G						
7	9							G	C	T	A	T	T	A	G	A					
8	10								A	A	G	T	T	A	G	A	C	G			
9	10									T	A	T	T	A	G			C	C		
10	11										A	T	T	A	G	A	C	G	G	C	C

Figure 2.0: K-spectrum k-mer table showing erroneous (colored blue) and unique data (yellow and white)

Both the multiple sequence alignment and the k-spectrum rely on sequencing reads. The only difference between them is that in the multiple sequence alignment, the generated reads or k-mers are realigned to form a matrix whose components are overlapping reads. The position of every nucleotide in the matrix is then determined. A column count of the number of occurrences of nucleotide is specified in the matrix. The nucleotide that occurs the most in the column count is then classified as distinctive.

However, this way of correcting the genomic error, that is, using the maximum frequency count in the matrix, may seem promising. The method is constrained if the frequency of two or more nucleotide bases is the same and is the highest or close to the highest predetermined number in the column.

The suffix tree or array sequencing error correction method also relies on reads. The reads have to be realigned with a reference genome. In the tree or array structure, the reference genome forms the root, and the reads form the stem and leaves. Using a predefined threshold, read counts from the leaves to the root are initiated. Nucleotides whose read count falls below the threshold are treated as error reads and are either discarded or a new but reduced threshold is reset to enable them to pass as distinctive reads.

INTEGRI PROCEDAMUS

University of Ghana http://ugspace.ug.edu.gh

The hybrid method blends both short and long reads error correction methods. It initially targets correcting long read errors. If the correction becomes challenging to effect because of the read length, the read is subdivided into smaller reads, and the short read error correction method is applied.

2.01 De Bruijn graph (Multi-graph)

The De Bruijn graph is an edge-directional graph that relies on reads or k-mers (read of size k). Here the size k will have to be initially determined. After which, the genomic data under study is divided into reads of size k. The total reads of size k become the input data for the graph, after which various graph operations are performed. The graph operation requires a smaller read size for larger input datasets.

Assuming our input dataset is AAABBBBA if a read of size three that is a 3-mer is required, in that regard, all the 3-mers for the input dataset will be AAA, AAB, ABB, BBB, BBB, and BBA. When this is successfully done, the left and right 1-mers from each of the 3-mer are then sorted. This will then produce AAA – (AA, AA), AAB – (AA, AB), ABB – (AB, BB), BBB – (BB, BB), BBB – (BB, BB), and BB – (BB, BA). Larger genomic data sets can be divided similarly. The only difference is that for larger datasets, we will require more than 2-mers and 3-mers.

The individual graph paths or directions are then added to generate the graph. AA emanating from (AA. AA) is the first and second left and right 2-mers respectively. A node with a direction from A to itself is placed on the graph (see Figure 2.1). After this, all the other 2-mers are sequentially placed on the graph following the steps described above being mindful of position and avoiding

University of Ghana http://ugspace.ug.edu.gh

repetitions. Figure 2.1 demonstrates a De Bruijn graph for the simple genome data AAABBBBA. It is important to note that the k-mer size choice depends on the input data size.

Depending on the data size. A unique k-mer size has to be chosen to avoid over-representation or generation of k-mers. When this is correctly done, an Eulerian walk through the graph along the edges should be able to reproduce the genomic data used in generating the De Bruijn graph.

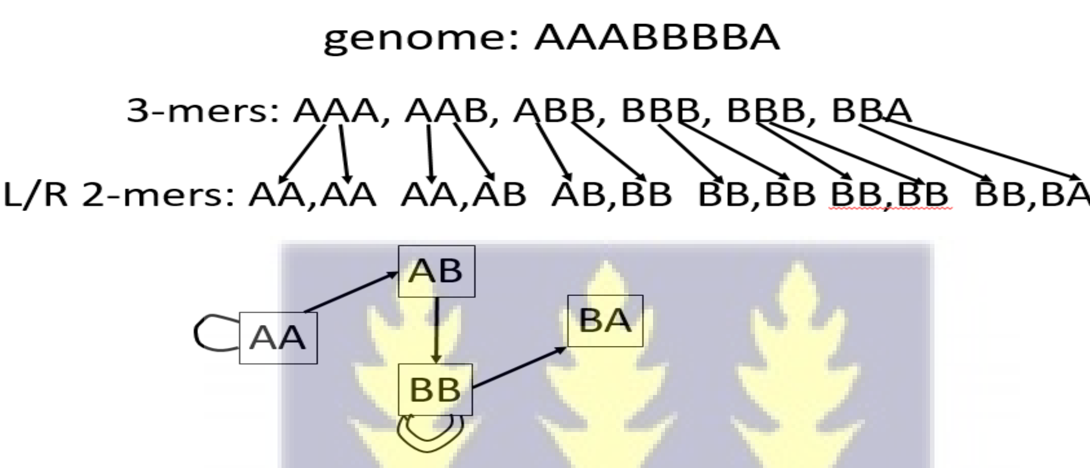


Figure 2.1: De Bruijn graph of a sample read.

2.02 Bloom Filter

The Bloom filter is a specialized data structure that is also referred to as an "approximate member query data structure." It uses hash functions to guarantee that if a kmer has been seen before during the hashing process, it reports the kmer's presence, which it classifies as true positive (TP). It correctly predicts the occurrence of all TP. However, it also exhibits the potential of falsely classifying a kmer as having seen it before, while in fact, the kmer is not present. This is considered a false positive (FP). The good side of this filtering algorithm is that it does not generate false negatives (FN). That is falsely classifying a kmer as not being present while it is actually there. This makes it an approximate data structure. Tables 2.1 and 2.2 below illustrate the workings of

University of Ghana http://ugspace.ug.edu.gh

the Bloom filter. In Table 2.1, two k-mers (K-mer 1 and K-mer 2) are passed through the independent hash functions, and each hash function produces a number as the output. The output from the various hash functions is then plotted in the filtering table. A plot of one (1) is recorded under a position if that position is found in the hashing table and zero (0) if the position is not found in the hashing table. All k-mers are then passed through the hashing table. Their output is compared with the data in the filtering table. If all the outputs are found in the filtering table, then it is classified as either TP or FP. For example, if we pass Kmer1 again through the hashing table, it will output [3, 14, 9, 5]. This can be found with plots [1, 1, 1, 1] in the plot table and thus classified as TP. However, if a Kmer5 is passed through the hash table and it generates [14, 13, 7, 10], its equivalent filtering table values will be [1, 1, 1, 1] indicating that it has seen it before but that is not the case. This is an FP condition. If another Kmer6, for instance, is passed through the hash table and generates the output [10, 6, 3,7], then its equivalent filtering table value will be [1,0,1,1]. Since the filtering values are not all ones, this will be a TN. From the analysis above, it is obvious to say that the Bloom filter cannot generate an FN. The independent hash functions produce different numbers for a given kmer. The probability that you generate FP is dependent on the number of hash functions and the size of the filtering table. This is because a larger filtering table will produce a larger structure, thus allowing the plotting of many values, thus increasing the occurrence of many plotted values. I must say that this is just for illustration purposes and that the hash function and filtering table will be quite large in a real situation.

Kmer	Hash function 1	Hash function 2	Hash function 3	Hash function 4
Kmer 1	3	14	9	5

University of Ghana http://ugspace.ug.edu.gh

Kmer 2	10	11	13	7

Table 2.1 Kmer hashing table

position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Plot	0	0	1	0	1	0	1	0	1	1	1	0	1	1	0

Table 2.2 Kmer filtering table

2.03 Suffix Arrays

This sorts out a given kmer into an array for easier referencing. This is possible because the sorted Kmer is transformed into indices. Referencing is then done via the index values instead of the kmer. For example, given the k-mer ACAGCATCGC of length 10, a position vector is generated (see Table 2.3. Using the starting position of the k-mer, suffixes are then generated from the vector table (table 2.4). A suffix array is then generated from the suffix table (table 2.5). This is done by rearranging the k-mers alphabetically, taking cognizance of the index of the position vector (Table 2.5). An array of indices is generated (table 2.6), which is used to determine whether a new kmer is a true kmer or not. For instance given index 4, the value of the kmer is [C, A, T, C, G, C] table 2.6. this means that there is no need to store the k-mers, but rather the array. Using the complementary or reverse process given a kmer, we should be able to tell which index will the given kmer be found. For example, given the kmer ATCG, using the index array (table 2.6), it is likely to come from index 0, 2, or 5. However, the best array index that would hold ATCG is the array index 5.

The suffix array can be mathematically defined as let $\Sigma = \{A, C, G, T\}$be the string of nucleotide alphabets and \$ be lexicographically smaller than each of the set of alphabets in Σ. For any

University of Ghana http://ugspace.ug.edu.gh

nucleotide sequence $X = x_1, x_2, \ldots x_{n-1}$, with $x_{n-1} = \$$. If $X_{[i]}=x_i$ is the ith symbol, then $X_{[i,j]}=x_i \ldots x_j$ is a sequence of X, and $X_i=X_{[i,n-1]}$ is a suffix of X [60]. Then the suffix array A of X is a permutation of integers 0, 1, . . . n-1 such that $A_{(i)} = j$ if and only if X_j is the ith lexicographically smallest suffix.

0	1	2	3	4	5	6	7	8	9
A	C	A	G	C	A	T	C	G	C

Table 2.3 kmer position vector.

0	A	C	A	G	C	A	T	C	G	C
1	C	A	G	C	A	T	C	G	C	
2	A	G	C	A	T	C	G	C		
3	G	C	A	T	C	G	C			
4	C	A	T	C	G	C				
5	A	T	C	G	C					
6	T	C	G	C						
7	C	G	C							
8	G	C								
9	c									

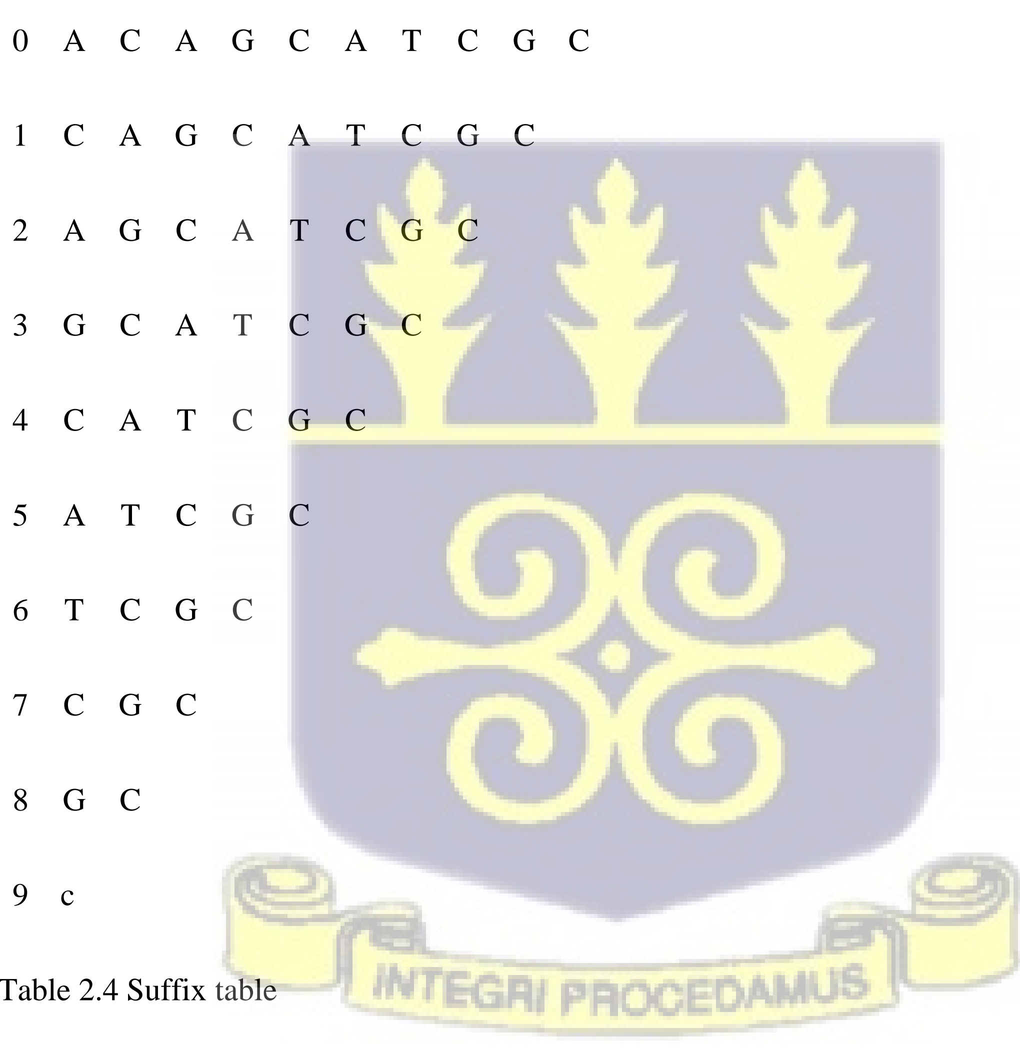


Table 2.4 Suffix table

0	A	C	A	G	C	A	T	C	G	C

University of Ghana http://ugspace.ug.edu.gh

2	A	G	C	A	T	C	G	C	
5	A	T	C	G	C				
9	c								
1	C	A	G	C	A	T	C	G	C
4	C	A	T	C	G	C			
7	C	G	C						
8	G	C							
3	G	C	A	T	C	G	C		
6	T	C	G	C					

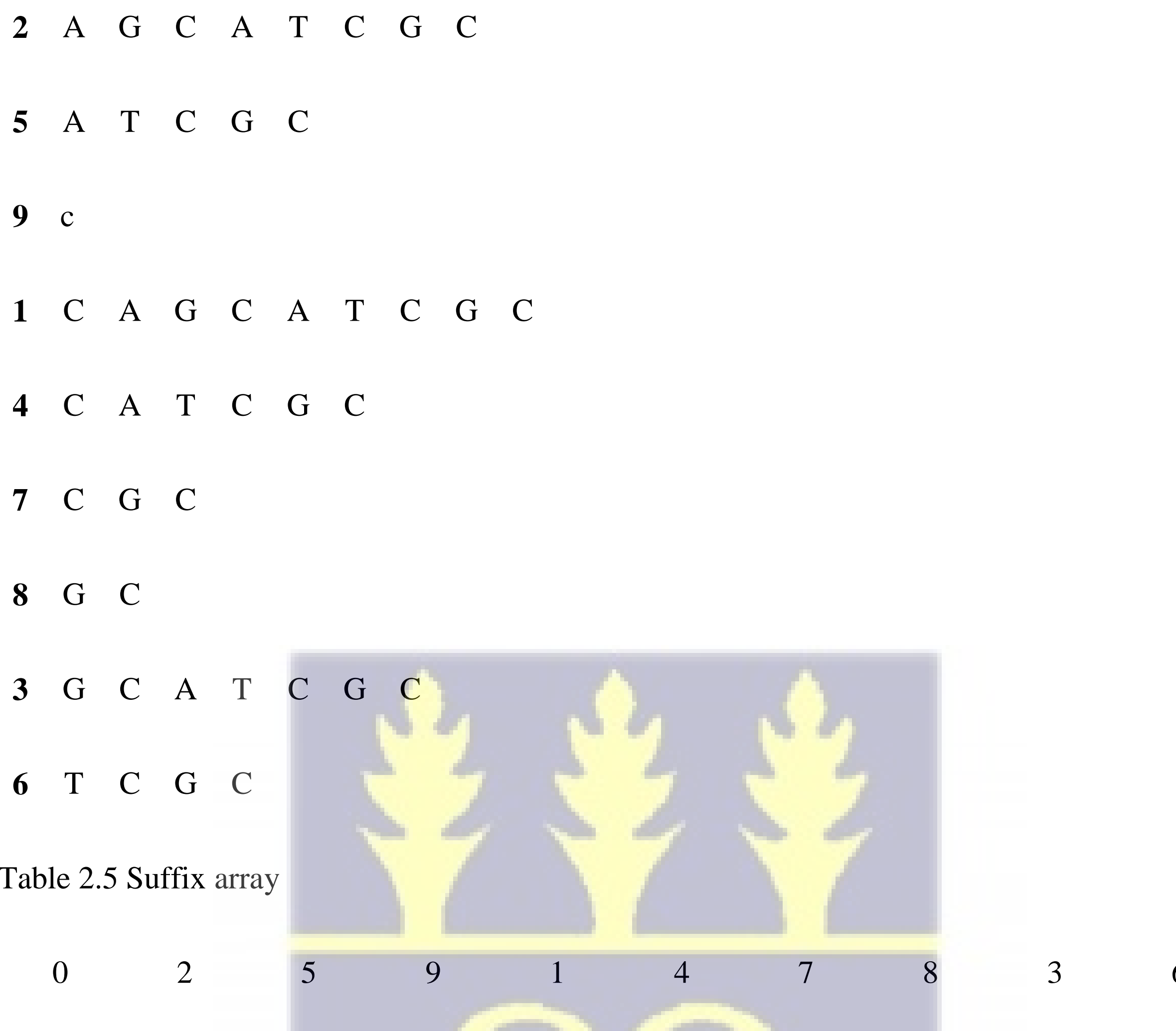


Table 2.5 Suffix array

0	2	5	9	1	4	7	8	3	6

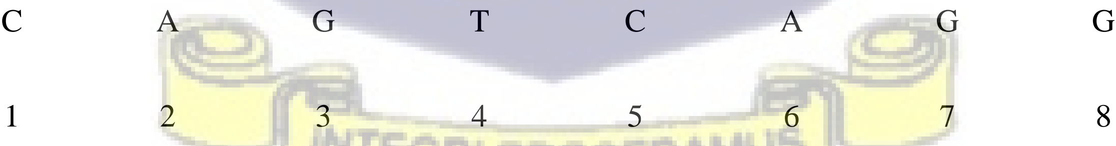
Table 2.6 Index Array

2.04 Suffix tree

The suffix tree concept is similar to the suffix array; the only difference between the two is that a tree structure is used in the former instead of an array structure depicted in the latter. Because there are four traces of nucleotides, there are going to be at most four branches or leaves in the tree structure. For example, given the genome data CAGTCAGG and their corresponding indices in Table 2.7, the indices start from one (1) because the root of the tree must have an index of zero (0). We were going to generate the suffix tree Figure 2.2 from its fast reversion genome table (table 2.8). We first create the root node and attach the rightmost read ($G_{\$8}$ with index 8) to form the first

University of Ghana http://ugspace.ug.edu.gh

leave. The next read ($G_{\$7}$**G)** is attached to the first leave because it takes the first ‘G’ as its source. To do that successfully, the first index, ‘8’, is shifted to the left to enable the attachment. The next read, $A_{\$6}$GG (with index 6), is attached to the root because it starts with a new nucleotide ‘A’ that has not been previously attached. Next, $C_{\$5}$AGG (with index 5) is attached. Again the attachment is done by creating a new leave since the starting nucleotide ‘C’ has not been seen yet. Next $T_{\$4}$CAGG (with index 4) is attached as a new leave because the nucleotide ‘T’ has not yet been seen. Next $G_{\$3}$TCAGG (with index 3) is attached. However, since ‘G’ has been attached before, but in the order of ‘GG’ but we have ‘GTCAGG’ to deal with, the rightmost ‘G’ is branched to the left (see Figure 2.2) to enable the kmer GRCAGG to be created. Next, $A_{\$2}$GTCAGG has to be attached to the tree. However, AGG has already been attached, but we are to attach AGTCAGG, so the rightmost nucleotide ‘G’ in AGG is branched off to the left to pave the way for the attachment of TCAGG to AG to create AGTCAGG. Finally, $C_{\$1}$AGTCAGG (with index 1) is to be attached. Since the start nucleotide, C has been attached before; we travel along that path; however, we have CAGG, not CAGT, so the rightmost nucleotide G is again branched off, leaving CAG and TCAGG is attached as a new leave, thus generating the suffix tree (Figure 2.2). Given a particular index, travel from that index to the root thus generates the expected kmer D. Savel et al [61]. For instance, index 7, 8, 1, 5, and 2 from the tree generates GG, G, CAGTCAGG, CAGG, and AGTCAGG, respectively.



C	A	G	T	C	A	G	G
1	2	3	4	5	6	7	8

Table 2.7 Genome data and index table

University of Ghana http://ugspace.ug.edu.gh

G$8								
G$7	G							
A$6	G	G						
C$5	A	G	G					
T$4	C	A	G	G				
G$3	T	C	A	G	G			
A$2	G	T	C	A	G	G		
C$1	A	G	T	C	A	G	G	

Table 2.8 Fast Reversion Genome Table

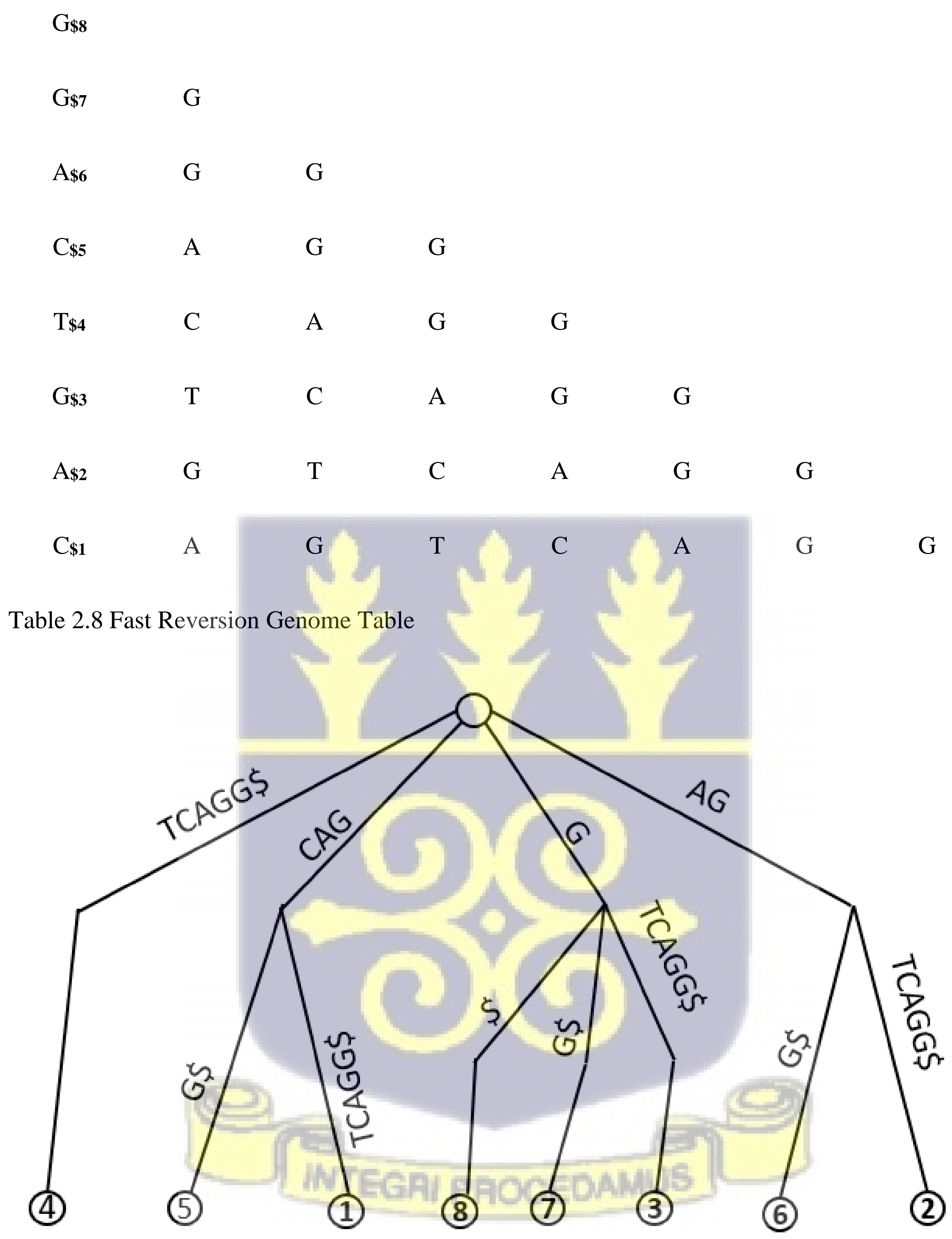


Figure 2.2 suffix tree generated from the kmer CAGTCAGG

University of Ghana http://ugspace.ug.edu.gh

2.05 Burrows-Wheeler Transform (BWT)

BWT uses a distinct reversible permutation (or rotations) of the characters of a string (kmer) [62]. Given a kmer (ACAACAGT$), where $ is lexicographically smaller than any of the characters, the Burrows–wheeler matrix (BWM), which is a rotation of the kmer at various positions, is first produced in Table 2.9. After the rotation, the rows are ranked and rearranged in ascending order, as shown in Table 2.10. The final column (TC$ACAAAG) of Table 2.10 is the Burrows-Wheeler transform (BWT) of the kmer (ACAACAGT$). This means that the BWT sorts the given kmer into a more compressible format, which is another form of data structure. It can be transformed into a suffix array see Table 2.11. In that regard, the indices can be used to access a particular kmer instead of the BWT. Given the mathematical definition of suffix array in section 2.03, the mathematical definition of BWT becomes a permutation of X, where B[i] = $ if A[i] = 0 and $B_{[i]}$ = $X_{[A(i)-1]}$ otherwise

$	A	C	A	A	C	A	G	T
T	$	A	C	A	A	C	A	G
G	T	$	A	C	A	A	C	A
A	G	T	$	A	C	A	A	C
C	A	G	T	$	A	C	A	A
A	C	A	G	T	$	A	C	A
A	A	C	A	G	T	$	A	C
C	A	A	C	A	G	T	$	A
A	C	A	A	C	A	G	T	$
Table 2.9: The Burrow–wheeler	A	C	A	A	C	A	G	**T**

University of Ghana http://ugspace.ug.edu.gh

matrix of ACAACAGT$.$								
A	A	C	A	G	T	$	A	**C**
A	C	A	A	C	A	G	T	**$**
A	C	A	G	T	$	A	C	**A**
A	G	T	$	A	C	A	A	**C**
C	A	A	C	A	G	T	$	**A**
C	A	G	T	$	A	C	A	**A**
G	T	$	A	C	A	A	C	**A**
T	$	A	C	A	A	C	A	**G**

Table 2.10 The Burrow–wheeler transform of ACAACAGT$.

$	A	C	A	A	C	A	G	**T**
A	A	C	A	G	T	$	A	**C**
A	C	A	A	C	A	G	T	**$**
A	C	A	G	T	$	A	C	**A**
A	G	T	$	A	C	A	A	**C**
C	A	A	C	A	G	T	$	**A**
C	A	G	T	$	A	C	A	**A**
G	T	$	A	C	A	A	C	**A**
T	$	A	C	A	A	C	A	**G**

BWT of ACAACAGT$

8	$								
2	A	A	C	A	G	T	$		
0	A	C	A	A	C	A	G	T	**$**
3	A	C	A	G	T	$			
5	A	G	T	$					
1	C	A	A	C	A	G	T	$	
4	C	A	G	T	$				
6	G	T	$						
7	T	$							

Transformation of BWT of ACAACAGT$ to suffix array

Table 2.11: BWT converted to the suffix array

2.06 FM – Index (Full-Text substring Index)

The FM index extends the functionalities of the BWT by providing extra rows and columns (table 2.12), which easily enables the detection of substrings (kmer), the position of substring through the introduction of an offset variable, and the number of occurrences of strings (kmer) using the count's parameter. Values for the extra columns are obtained by going through the BWT vertically

University of Ghana http://ugspace.ug.edu.gh

and counting the number of times a particular variable occurs. The count is done for each row of the underlying column, and repeated occurrences are noted. Values of the count are obtained by going through the first column of the suffix array table that generated the BWT and counting the number of times a nucleotide occurs. Since $ is lexicographically smallest, its position is ignored in the count for the nucleotide A. The offset values are obtained by going through the index of the suffix array table of the BWT and recording the first occurrence of a particular nucleotide. To access a particular string (kmer), the offset value, which indicates the position of the leftmost variable together with its count counterpart, which indicates the number of times the first position variable occurs, is used. This improves the performance of searching through the suffix array.

Index	Suffix Array								BWT	$	A	C	T	G
0	$	A	C	A	A	C	A	G	T	0	0	0	0	0
1	A	A	C	A	G	T	$	A	C	0	0	0	1	0
2	A	C	A	A	C	A	G	T	$	0	0	1	1	0
3	A	C	A	G	T	$	A	C	A	1	0	1	1	0
4	A	G	T	$	A	C	A	A	C	1	1	1	1	0
5	C	A	A	C	A	G	T	$	A	1	1	2	1	0
6	C	A	G	T	$	A	C	A	A	1	2	2	1	0
7	G	T	$	A	C	A	A	C	A	1	3	2	1	0
8	T	$	A	C	A	A	C	A	G	1	4	2	1	0
	Counts									1	5	2	1	1
	Offset									0	1	5	8	7

Table 2.12: FM index table extending the BWT with five extra columns and two rows.

University of Ghana http://ugspace.ug.edu.gh

2.07 Hash Table

The hash table is an extremely versatile and efficient multi-map data structure that consists of a set of an array of empty buckets that are used to represent sets and maps. The empty arrays initially represent null references or null pointers (Figure 2.3). As items are added to the hash table, the empty buckets become lists. Also associated with the hash table is the hash function *h*, which maps each distinct key or 3-mer (GTG, TGC, GCG, CGT, GTG, TGT, GTG, TGG, GGG, GGG, GGG) from the kmer T (GTGCGTGTGGGGG) onto one of these buckets—using the hash function to assign all the 3-mers to the hash table. Figure 2.4 shows the first 3-mer GTG added to bucket three by the hash function showing the offset of zero [63]. Assuming the has function adds the second 3-mer TGC to the top bucket with offset 1, The process continues till the last 3-mer is added. A careful look at the 3-mers shows that there are situations of a collision where a particular 3-mer occurs more than once (GTG 3 times, GGG 3 times). These 3-mers may not necessarily be assigned by the hash function to the same buckets, while different 3-mers may be assigned to the same bucket. The entire hash table can be seen in Figure 2.6. This means that querying the hash table for distinct 3-mer might produce variables from either the same or different buckets. This slows down or delays the querying process.

University of Ghana http://ugspace.ug.edu.gh

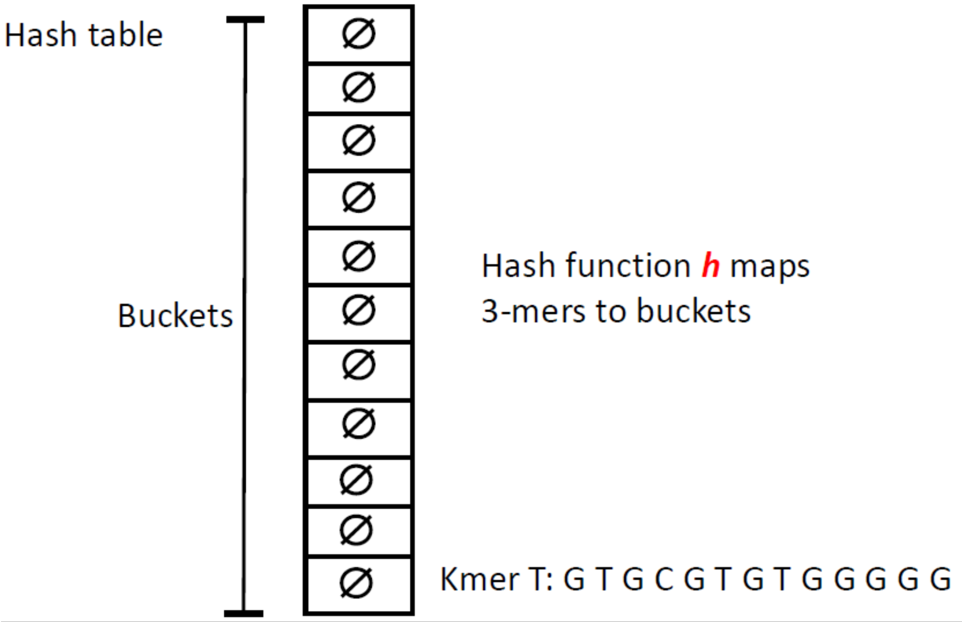


Figure 2.3: Hash table showing has function h that maps 3-mer from the kmer T: to the buckets

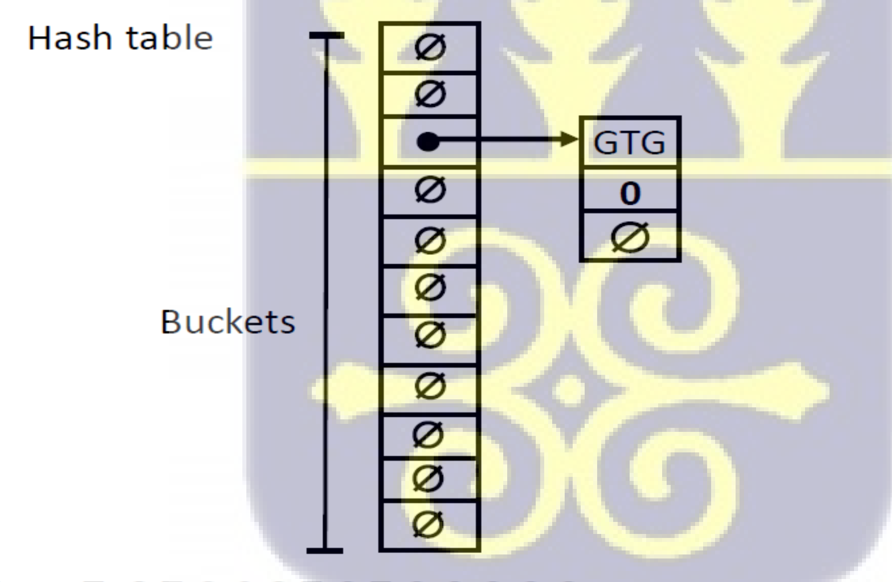


Figure 2.4: The appending of the first 3-mer with offset 0 to the hash table by the hash function

INTEGRI PROCEDAMUS

University of Ghana http://ugspace.ug.edu.gh

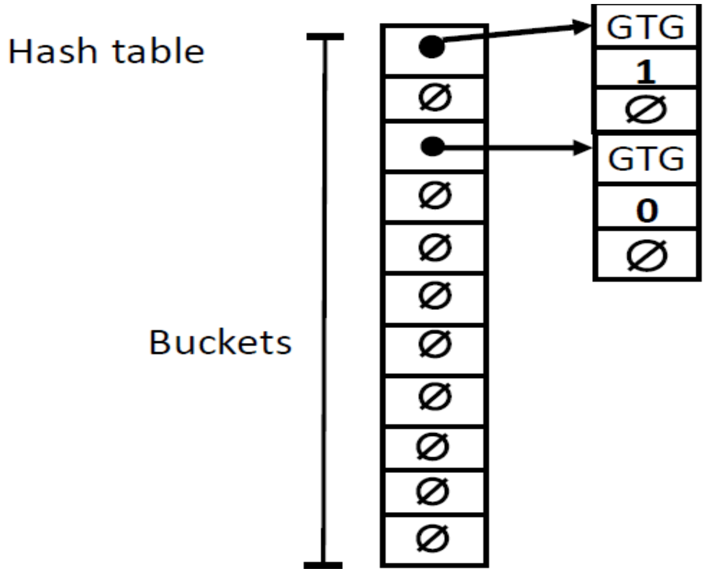


Figure 2.5: Addition of the second 3-mer with offset 1 to the hash table by the hash function

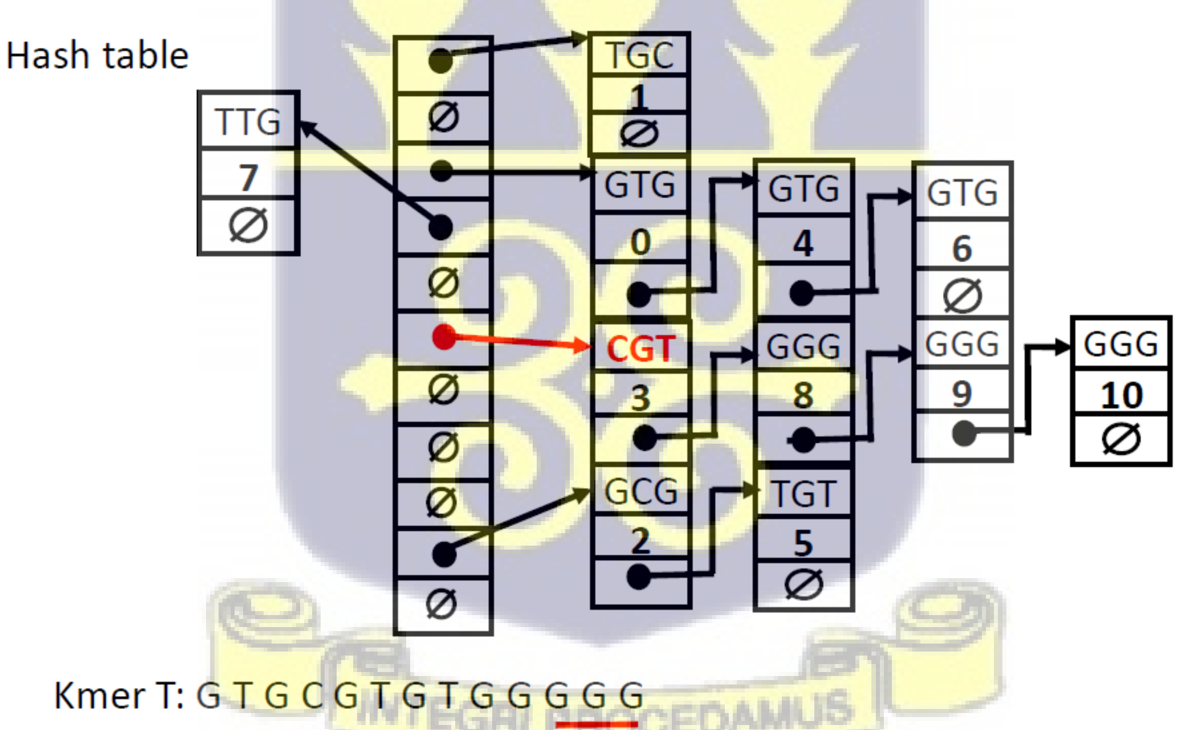


Figure 2.6: A hash table showing collision at the bucket where CGT is stored.

University of Ghana http://ugspace.ug.edu.gh

2.1 K – spectrum-based method

The k-spectrum genome sequencing error correction theory underpinning comes from the De Bruijn spectral alignment graph [64]. For the successful use of the De Bruijn graph under the K – K-spectrum error correction methodology, an integer value k usually based on the size of the dataset is set. After which, the dataset is subdivided into reads each of size k (k-mer reads sizes) [65]. To tag k – mers either as unique or error, a threshold T is set after which all K-mers are counted, and the frequency is compared with ***T***. K-mers whose frequency of occurrence is either equal to or greater than ***T*** are considered to be unique (also known as solid) and k-mers whose frequencies are not up to T are regarded as error or erroneous k-mers [66].

The frequency counts of the individual k-mers are used to assign varied weighted values to the k-mers. The weighted k-mer values and the k-mer frequency count are used to build a hash [67]. The information is then used to build a filtering algorithm that helps to separate error k-mers from unique k-mers [65].

The threshold value is reset, and a new filtering algorithm is designed; error k-mers are filtered again but this time using the new filtering algorithm. The aim is to reduce the number of k-mers that were classified as errors [68].

[69] D. Kelly et al, used the k-spectrum approach to enhance the performance of their algorithm, they provided another variable called protocol-specific miscall rate which computed the individual weights of all the k-mers (unique and error) and denoted their weighted sum in the dataset as ∏_(i=0)^(n-1)*pi, where pi signifies the correct sequencing of the i-th base in a quality score. Using the Gaussian and Zeta distributions and a cut-off point, the error k-mer was modeled into a Gamma distribution, and those that met the condition of the cut-off point were reclassified as

University of Ghana http://ugspace.ug.edu.gh

unique. Based on the size of the data set and the size of errors detected, the new cut-off point can be reset and the process of reclassification repeated, particularly if a very large error size is detected. They then proposed that there is quality-based scores were proportional to the sequencing error in datasets. However, Juliane C. et al. [70] showed that that is not usually the case.

[71] X, Yang, et al, blended quality-based scores, k-spectrum, and k-mer context to improve the sensitivity and specificity of reads. Using tiles, which are overlapping substrings of k-mers, they corrected sequencing errors. This approach prevented or reduced the k-mer misallocation during k-mer realignments. Using dataset D, assuming the observed frequency of a particular tile in the dataset are f and t, respectively. If the threshold frequency is set to f, then all tiles with frequencies y>f will be reclassified as unique, thus reducing the number of error k-mers in the dataset [72]. The process is used to build a hamming graph; in the hamming graph, nodes are represented by k-mers, and the hamming distance becomes the edges between nodes. Using a predetermined value d, any nodes with a hamming distance less than d are treated as candidates for alignment. The approach adopted by Reptile requires a lot of processes and requires a lot of computational resources to achieve an output. This makes it difficult to use it in sequencing large datasets.

Y. Heo et al [73] relied on Bloom's filtering algorithm to determine whether a k-mer should be regarded as either unique or an error. Using all k-mers and their reverse complement, which they called canonical k-mers, they plotted a graph of k-mers and determined the k-mer with the highest frequency (threshold). In this approach, odd k-mers emanated from the nucleotide value that appeared in the middle of the k-mer. If the value is either C or A, then the k-mer length is odd. The canonical value is used. However, when the middle value of the k-mer is either T or G, the reverse complement of the k-mer is used. Using the hash function in Bloom's filtering algorithm, all k-mers that pass the threshold value are considered unique [74]

University of Ghana http://ugspace.ug.edu.gh

L. Ilie et al [75] relied on a 2–bit scheme to change all k-mers into a 64-bit integer which helps in replacing the positions of all new k-mers. The method requires the upload of the whole dataset to determine the threshold value and also the k-mer size. Just like BLESS and the other k-mer correction schemes, they used a threshold value to classifier k-mers either as an error or unique. If a k-mer frequency is either equal to or greater than the threshold value, then that k-mer is classified as unique. If this condition is not met, then such a k-mer will be considered an error k-mer. The major advantage of their method over other sequencing platforms that use similar error correction methods is that the threshold is automatically determined and does not require readjustment to improve the number of unique k-mers if their size is small. The readjustment schemes employed by the other methods grant the possibility of reclassifying error k-mers as unique.

[76] L Song et al's approach to genome sequence error correction is similar to [73]. This is because it also relies on Bloom's filtering algorithm. It first segregates the entire genome dataset into k-mers and then stores them in Bloom's filtering algorithm. The filtering algorithm separates unique k-mers from the error k-mers. It then generates a probabilistic hash function of unique and error k-mers using a greedy algorithm. A sampling fraction that is inversely proportional to the sequencing depth and reads the overlapping position of k-mers is used to correct the sequencing errors. If r is the read overlapping position and $\boldsymbol{k}$ is the k-mer length, $\boldsymbol{r}$ should be chosen such that $1 \leq \boldsymbol{r} \leq k$, unlike the harming distance in Reptile, which relies on distances between edges, Lighter uses the position vector of the k-mers.

[77] H. Li treats the sequencing error correction process as a 4-tuple consisting of i, W, C, and p. Where i is the position of the preceding k-mer. The last (k-1)-mer is represented by W, and C is a k-mer that was previously corrected and held in place. p is the penalty value set such that the entire i, W, C, p. is iterated. With the new condition of a being the read base and W[o a] being a trusted

University of Ghana http://ugspace.ug.edu.gh

k-mer, a new tuple (i+1, W[1, k -2] ɵ a, C’, p’) is selected from the iteration. The entire process is further repeated by adjusting a, i, and p, to accommodate error sequencing reads whose tuple was close to the initial tuple selection value [78]. Unlike most sequencing error correction algorithms that are used in correcting insertion and deletion errors, BFC is capable of correcting substitution errors.

E. Marinier et al [79] style of error correction is based on removing both leading and trailing N’s and a method referred to as preprocessing. Since all Ns are unknown genomes, they are replaced with any of the nucleotides A, T, G, or C. The N replacement in this manner makes the preprocessing method behave as treating all the Ns as substitution errors. After the N replacement, all k-mers and their complements are used to build a hash table. With the aid of a threshold value, k-mers are treated as either unique or error. To reduce the size of the dataset, all unique k-mers are removed from the hash table, thus freeing memory space to increase the performance of the underlying algorithm and reduce execution time. The error k-mers are then used to build an array consisting of r – k + 1 k-mers. Where r and k are the read and k-mer lengths, respectively, this technique, unlike the preprocessing stage, is used to correct both indels and substitution errors.

M. Dlugosz et al [80] target errors that are found in the eukaryotic dataset. It relies on the BLESS and KMC tools to correct sequencing errors. The KMC tools focus on segregating genomic datasets into k-mers and counting the k-mers. After that, the BLESS tool is used to determine whether a k-mer is unique or erroneous based on a predetermined frequency threshold value. Quality indicators such as frequency values or error k-mers that are close to the threshold value are reclassified as unique k-mers.

Y. Liu et al [81] use the k-spectrum sequencing error correction method. It was initially focused on correcting short-read errors that were generated by the Illumina dataset. Using Bloom’s filtering

University of Ghana http://ugspace.ug.edu.gh

algorithm, k-mer segregation and parallel k-mer counting are embarked upon. The output is the generation of errors and unique k-mers. The error k-mers are placed in a hash table for reclassification. K-mer frequency counts are used to prevent unique k-mers from being mistakenly added to the hash table. A new variable known as k-mer accuracy, which is based on the closeness of the frequency of occurrence of an error k-mer to the threshold value, is set. This allows error k-mers to be reclassified and either discarded or treated as unique.

Since the theory underpinning the [81] is the k-spectrum, which is designed for correcting substitution errors, further laborious adjustments have to be made for it to be used in correcting both indels and substitution errors. The adjustment limits its use in platforms such as Illumina and the 454 sequencing platforms. The primary reason is that these platforms generate very large genomic datasets, and the adjustments required and computational resources needed to handle such datasets may make it very difficult, if not impossible, to be used on such platforms. Threshold value settings in [81] are based on guesswork; this means that at the time of guessing the threshold value, a wrong threshold value may either misclassify error k-mers as unique or unique k-mers as an error.

Limitation of K-Spectrum Error Correction Method

The K-Spectrum spectrum approach for genome sequencing error correction is based on the estimate of the likelihood of error occurring based on the frequencies of k-mers. Though it has been successfully used in the citations under Chapter 2.1, there are notable gaps that need to be addressed.

One of the primary limitations of K-Spectrum methods is their limited capability to handle complex insertion and deletion errors, A. D. Smith et al [82] This limitation hampers the extensive

University of Ghana http://ugspace.ug.edu.gh

correction of genome sequencing errors and limits the accuracy that may emanate from the genome sequencing error correction. One other restriction in the K-Spectrum approach is the choice of the k-mer length, longer k-mer lengths can lead to the identification and correction of more errors however, they also increase computational complexities and will require more memory to handle. On the other hand, short k-mer lengths may miss important sequencing errors which will lead to the introduction of more false positive identification. This means that the selection of an optimal k-mer length remains a challenge. This requires more research into the use of suitable k-mer lengths for different sequencing technologies and error types.

Scalability and computational efficiency especially when dealing with large-scale datasets is another gap when examining the performance of K-Spectrum-based error correction. This is because each time a new dataset is introduced, there should be manual adjustments of parameters to handle such datasets.

The lack of standardized evaluation metrics is a significant gap when assessing the performance of K-Spectrum-based error correction methods. This is because there is a lack of consensus and coordination in the appropriate evaluation and comparison criteria to use. Without standardization, it becomes challenging to compare the various approaches used and this may reduce the identification of the strengths and weaknesses of the approach used. This has the potential to slow the advancement of genome sequencing error correction techniques under the K-Spectrum approach.

INTEGRI PROCEDAMUS

University of Ghana http://ugspace.ug.edu.gh

2.2 Multiple Sequence Alignment – MSA

The multiple sequencing alignment methods use algorithms to generate k-mers from the genomic dataset to be studied. All the generated reads or k-mer are then realigned with the help of the sequencing alignment algorithm. In the alignment process, all reads or k-mers are aligned with a reference genome. The alignment of the k-mers with the reference genome produces an overlapping matrix of k-mers. Using a priority voting scheme that measures the number of occurrences of a k-mer in the dataset, reads with the highest probability of occurrence (counting) are classified as unique, while k-mers that could not meet such values are classified as error k-mers. The error reads or k-mers with a voting value close to the adjusted voting scheme value are reclassified as unique reads.

[83] M. Schulz et al, first segment the target dataset into k-mers, after which it employs the suffix–tree error correction methods to the reads generated from the segmentation. With the help of a uniform sampling method, the mean expected coverage, the log-odd ratio, and the naïve Bayesian classifier, the reads are separated into unique and error reads. The use of the naïve Bayes classifier enables the method to be used in correcting both substitution and indel errors. A voting system is used to determine whether an error is a substitution or an indel error. If there is a tie in the voting scheme for k-mer counts, then that type is considered a substitution error; otherwise, the error is regarded as an indel. The read lengths were varied under [83], thus allowing the error correction method to be used in both the Illumina and the 454 sequencing error correction platforms. To avoid the system misclassifying reads as false positives during the correction process, the reads were allowed to overlap. Also, the use of the naïve Bayes classifier enables [83] to be used in correcting sequencing errors coming from both small and large datasets.

University of Ghana http://ugspace.ug.edu.gh

[12] L. Salmela relies on sequencing alignment in correcting sequencing errors. It generates sequencing reads from the dataset and indexes the reads and their reverse complement. To successfully do this, it uses a reverse sequencing process to generate the complementary reads. Even read lengths are discarded since the orientation of their reverse complement cannot be ascertained. Also, reads that contain the unknown or indeterminate characters ***N*** are also discarded. The accepted reads are used to build a k-mer indexing table. Reads from the k-mer indexing table are used to create a multiple sequence alignment scheme. This allows for the computation of the k-mer neighborhood variable, which is fed into the Needleman – Wunsch algorithm to build consensus on the aligned reads. The consensus allows for the read quality score to be computed. Under this scheme, reads with low-quality scores are classified as error reads. An attempt is then made to reduce the error reads by creating a threshold value. In this regard, all error reads that make the threshold value is then reclassified as unique. This method of reclassifying corrected genome read errors may be either restrictive or lenient if the threshold value is set either too high or too low. This calls for a more scientific way of generating the threshold value.

[84] A. Allam et al used the Kaust Assembly sequencing error correction scheme. Under the scheme, a partial order graph is generated using information produced from performing multiple sequence alignments on reads produced from the underlying genome dataset. A set of heuristics (rules) are then used to identify indel errors that occur in the read alignment. If a k-mer length is k, for instance, each k-mer is divided into three equal parts (k/3). Using a reference read r, a read is treated as an error if it does not match the reference read. A graph method is then adopted to correct the error reads. In the correction scheme, the distance between the error read, and the reference genome along the path on the graph must be minimal. It must also have a low heuristic value in the indel error identification process. A better way of achieving a minimal graph path

University of Ghana http://ugspace.ug.edu.gh

distance between reads and the reference read is by assigning weights to the edges and normalizing the weights.

[85] W. Kao et al did not rely on referencing the genome in correcting sequencing read errors. Rather it uses a read-overlapping scheme to correct sequencing errors. An automated maximum acceptable error value and minimum overlapping lengths are set. These two values are not disclosed at the beginning of the error correction process to avoid false adjustments in the dataset. Read clusters are formed, and reads are then aligned within clusters. Using the automated maximum acceptable error correction value and the minimum overlapping length, a near-neighbor filtering algorithm is developed, and the reads are fed into the algorithm. Outputs from the filtering algorithm are used to generate a hash table for reads in the various clusters. An assumption is then made that the reads are uniformly distributed, thus allowing the Poisson distribution to determine a threshold value. The threshold value is then used to classify reads as either unique or error. Depending on the size of the dataset and the number of k-mer errors generated, the error reads may be sent back for reclassification by adjusting the threshold value.

They use multiple sequencing alignment as a scheme for determining whether a k-mer is either unique or error requires the pairwise alignment of reads and a deterministic minimum overlap value that needs to be set. If this is not properly done, the error correction scheme will be overwhelmed, and misclassification can occur. The Poisson distribution is used to remedy this situation on the assumption that the reads are uniformly distributed. The major setback of this approach is that large datasets are normally distributed instead of Poisson; hence the assumption that they are uniformly distributed and thus setting a threshold and minimum overlapping values may be misleading and thus affecting the inferences drawn from it.

University of Ghana http://ugspace.ug.edu.gh

Limitation of Multiple Sequence Alignment Error Correction Method

Multiple sequence alignment method of detecting sequence error is by piling up k-mer to identify variation and conserved regions. A major limitation of MSA-based error correction methods is their efficiency and scalability when handling large-scale datasets. When the size of genomic data increases, the computational resources required to perform MSA on this type of dataset become a challenge. Time and memory resource required to efficiently perform the alignment processes on the large-scale dataset also becomes a challenge. Developing scalable and efficient algorithms to handle this challenge is a limitation for MSA.

MSA may also struggle to handle low occlusion or repetitive regions which will lead to misalignments that will not help in integrating variant calling. In this regard, benchmarking and accurate evaluation becomes a challenge.

This means that although MSA has shown promise in genome sequencing error correction, some gaps, and limitations include handling large-scale datasets efficiently and establishing standardized benchmark datasets and evaluation metrics which lead to accurate genomic analyses and interpretations.

2.3 Suffix tree/array-base

This sequencing error correction method is similar to the processes used in either an array-based or tree-based data structure. It also uses threshold values in classifying a read either as an error or unique. The major difference between the tree-based and array-based methods is how the threshold variable M is selected. The suffix-based data structure has been detected to manage disc space better than its tree-based counterpart, X. Yang [86].

University of Ghana http://ugspace.ug.edu.gh

J. Schoeder [87] uses a more liberal form of the weighted suffix tree method. In their approach, a subtree sn is born by traversing from the root node through the edges to a leaf in a given dataset. This means that the total number of subtrees will depend on the total traversal along different paths to all the leaves in the data structure. The Bernoulli distribution is used to analyze the information in the subtrees. The root of the subtree is classified as an error if the sum of the subtree is less than the variance [88], to correct a subtree Sn that is labeled as an error subtree at root n, a subtree Sm which is declared as error-free, is then merged with Sn at n. For multiple error correction methods, this approach is repeatedly carried out. Substitution errors are corrected under this error correction scheme.[89]. A variant of [87] called Hybrid-SHREC is used in correcting insertion and deletion errors. To extend the [87] error correction method to capture and correct insertion and deletion errors, Hybrid-SHREC, a variant of [87], is used, L. Salmela [90]. To enable the correction of an insertion error being the last root of the subtree, that subtree is instead compared with its tree sibling of the parent tree. However, if it is a deletion error, then the comparison is made with either its children or its siblings.

L. Ilie [91] is considered a high-throughput sequencing error correction method. All are read together with their reverse complement first determined using a suffix array algorithm. Reads are then classified as either error or unique based on a predetermined threshold variable. Using a cluster of support-weighted variables similar to that adopted by [87], error reads are then corrected. Computed mean and variance values from the binomial distribution are then used to authenticate the status of the state of the unique and error reads. The uniqueness of the algorithm allows it to be used to handle both small and large datasets. However, it is important to point out that the performance of the algorithm is inversely proportional to the size of the dataset. This means that increasing datasets generate lower performance.

University of Ghana http://ugspace.ug.edu.gh

D. Savel et al [61] is another suffix-tree error correction method. Unlike other suffix-tree methods, Pluribus simultaneously handles multiple suffixes when correcting sequencing errors. It uses variable read or k-mer lengths, and the average read length together with an average number of reads in the dataset under investigation is first determined before the correction process starts. Trees are formed using edges (paths) from siblings. The number of nodes from the root node to the leave in a substring is also determined. To differentiate between various nodes, the reverse complement of the reads is matched with the reads. A voting scheme that relies on the frequency of occurrence of siblings is then determined and used to classify reads as either unique or erroneous. To extend the sequencing error correction process, the entire process is looped through multiple sequence alignments. It is noted to require minimal computation resources despite its looping process to sequence large datasets.

Y. Gu et al [88] correct genome sequencing errors using box query and disk-based indexing methods. In such methods, large datasets that require more computational resources are benchmarked into suffix tree or array platforms. The benchmarking is done by loading reads and their corresponding complementary metadata onto a Bond-tree disk. All computations involving the sequencing reads and determination of reads as either unique or erroneous are done on the Bond-tree disk. The reads are first treated as non-ordered multidimensional vector space. The dataset is then treated as a finite set of strings comprising characters in the domain (A, C, T, G). The Cartesian product of the reads is then transformed onto a Bond-tree disk. To correctly detect sequencing error, a voting scheme that schedules the maximum likelihood of being classified as either error or unique on the Bond-tree disk is applied. A hierarchical tree indexing structure is maintained on each box for easy identification of read positions.

University of Ghana http://ugspace.ug.edu.gh

The classification of reads as either error or erroneous is done on the leaves or children on the tree at the same level. Since there is a proportionality between tree size and leave size, as the size of the dataset increases, so do the tree and leave sizes. This means that there will be a corresponding increment in the size of children on the tree. Scheduling and indexing strategies could be employed to sequence genomic data on the platform. Larger datasets will be represented by very large trees, and their sequencing, aside from the indexing and scheduling strategies, will require efficient and specialized algorithms that will manage the large computational resources such as memory and disk space.

Limitation of Suffix Tree / Array Error Correction Method

Suffix trees or arrays use data structure methods to store sorted suffixes of a given genomic sequence which allows efficient substring searches that aid in the error correction processes. One major limitation of suffix-tree or suffix-array-based error correction is their efficiency and scalability when handling large-scale datasets. This is because as the size or volume of genomic data increases, the computational resources required for constructing and manipulating either suffix-tree or suffix-array-based error correction method becomes a challenge. Also, the time and memory required for query operation can be hampered. This makes it prudent to develop scalable and efficient algorithms to avert these limitations. They may also struggle to identify errors introduced during PCR amplification which calls for the incorporation of additional processing strategies to handle such errors.

Another gap identified under suffix tree or array error correction methods is their inability to handle repetitive or low occlusion regions within the genome. This can lead to ambiguous matches which will hamper the identification and correction of sequencing errors. In this regard,

University of Ghana http://ugspace.ug.edu.gh

complementary error correction processes will be required to enhance the accuracy of the error correction processes in these regions.

Suffix tree or array error correction also lack standardized benchmark datasets that can be used across all platform applications thus making it difficult to evaluate and compare results that were obtained from different suffix approaches.

2.4 Hybrid Error Correction Method

The hybrid method is a blend of sequencing read methods employed for correcting sequencing errors found in short and long sequencing reads. It is focused on correcting errors found in third-generation sequencing platforms.

A. Das et al [92] rely on the De Bruijn graph to correct sequencing errors found in short reads. It first determines which error is emanating from weak or vital regions in the dataset. Since weak regions have a high likelihood of generating errors due to noise, errors from such regions are first compared with the counterparts from vital areas in the dataset. A voting scheme is employed together with the widest path algorithm on the De Bruijn graph. This approach is used to correct insertion and deletion errors. To employ the same method to correct substitution errors, the longer read length is first reduced to short reads. A threshold value is then computed from Pearson's skew coefficient. Low and high score coefficients are classified as unique and erroneous, respectively. To reclassify error reads in the dataset, difference threshold values are set and the correction process is repeated.

INTEGRI PROCEDAMUS

L. Salmela [93] is used to correct both long and short sequencing error reads. Short reads and their reverse complements are plotted on the De Bruijn graph. Depending on the size of the overlapping read, the read may be treated as coming from either a short or long read. Bloom's filtering

University of Ghana http://ugspace.ug.edu.gh

algorithm is then used to classify a read as unique or an error. To correct error reads, the shortest path between two unique reads boarding the error read is used to determine the read to be used in correcting the error read. If no such path exists, the error read is discarded. If only one unique read borders such an error read, it is referred to as either a head or tail region based on the position of the read that borders it. The entire reads in the dataset are looped with the head and tail regions, and error reads are either corrected or discarded based on an automatic threshold value that is set by the system.

J. Wang et al [94] blends the FM-index with the multiple Burrows-Wheeler Transform (BWT) [95], and the De Bruijn graph is populated with the help of the suffix tree algorithm [96]. The use of the FM-index algorithm allows the platform to work with short read lengths. This means that variable read lengths are generated based on the arbitrary sizes of the reads that are fed into the system. The frequencies of the various short reads are used in building the De Bruijn graph. This is used as a benchmark to transform a region that is populated with long reads to short reads. [97].

The automatic adjustment of long read lengths into short reads leads to an automatic proportional adjustment of threshold frequencies and a dynamic adjustment of reads that are classified as either unique or error. The dynamic adjustment produces varied k-mer indices. This allows the admission of varied data sizes into the platform for sequencing.

O. Choudhury et al [98] is another hybrid sequencing error correction method that uses iterative learning by applying optimization techniques to the weight that is assigned to short sequencing reads. The short reads are initially assigned with long reads and various weights are assigned based on the specific alignment. The generated weights are then normalized to produce a table of normalized short-read weights [99]. The short reads that were used to build the normalized weighted table are used to correct errors in the long reads. This means the short sequencing reads

University of Ghana http://ugspace.ug.edu.gh

whose weighted sum does not meet the qualifications to be in the normalization of weights does not play a role in correcting long sequencing error. This is a downside of the platform because all unique reads should contribute in one way or the other in the long read error correction process. Using consensus based on k-mer frequencies from the normalized weighted short reads, long read errors are corrected.

Long reads are noted to contribute minimally to sequencing errors; the size of such reads is reduced in sequencing large datasets since only a few algorithms and systems have the resources to handle such datasets [100][101] may be required.

Limitation of The Hybrid Error Correction Method

The hybrid method combines multiple techniques such as graph-based, alignment-based, or statistical methods in the error correction process. This means both long and short k-mers can be incorporated to significantly harness their error correction properties. However, one significant gap in this method of error correction is in the limited integration of multiple data types into the various underlying processes. Also, the various methods that were integrated to form the hybrid may have their limitations, this means that when they are put together, their limitations combine to increase the limitations of the hybrid method.

Error Correction Using Machine Learning

Genome sequencing error correction is the process of detecting an error, also known erroneous dataset, at a position in a genomic dataset. Depending on the way the genome dataset is produced, they are not without errors. These errors can either be substitution, insertion, or deletion errors. These errors depending on the size of the dataset can render some inferences that may be drawn on the dataset after the dataset has undergone various transformations inaccurate or nonapplicable.

University of Ghana http://ugspace.ug.edu.gh

[18] used a deep variant calling approach for the error correction method. This was done by creating reads from both the reference genome and the genome under study. They adopted a three-tier approach by initially aligning reads with the reference genome and treating the aligned reads as pileup images. In the second tier, the pileup images were used to train a CNN with stochastic gradient descent to optimize the network. In the third tier, the pileup images were evaluated. This was done by realigning the pileup images with the reference genome. Each image block was then encoded using a red (R), green (G), blue (B), i.e., RGB pixel encoder. The encoded data was analyzed using a CNN. The network outputs the likelihood of occurrence of a particular genotype. Using the NA12878 dataset, the network performance was measured as 98.98%.

Barthamtoshy H. M. et al. [102] used TensorFlow and Google Nucleus (a framework for manipulating genomic data) to predict and correct errors in genomic data. They used both DNA and RNA datasets for the error correction process. They also used a generative adversarial network to generate errors in their dataset and used their network to predict the error from the generative adversarial network. Their network produced a 98.7% evaluation accuracy, thus missing out on 1.3% errors.

Using several input datasets, Krachunov M. et al. [103] used random forest together with Repeated Incremental Pruning methods to reduce genome sequencing errors. Their strategy involves using the frequencies of selected bases of varying lengths in a weighted set of bases. With a preset threshold value, all bases whose frequencies are less than the threshold value are treated as erroneous or errors. The random forest machine learning approach is employed in the filtering of the dataset. The random forest machine learning algorithm speeds up the filtering processes that lead to the detection of errors in the dataset. The detected errors are further filtered by assigning

INTEGRI PROCEDAMUS

University of Ghana http://ugspace.ug.edu.gh

new threshold values, and those nucleotides whose count falls below the new threshold are now considered an error and then corrected based on the nucleotide with the highest probability.

Qu L. W. et al. [104] focused on correcting errors emanating from the Oxford Nanopore sequencing process. The Oxford nanopore is a third-generation sequencing process whose reads are longer than those of the second-generation sequencing process. The longer reads of the third-generation sequencing process produce quite a large number of sequencing errors. Qu. L. W. et al.'s work combined two machine learning approaches, the convolutional neural network and the bidirectional long short-term memory approach. Their error reduction approach primarily uses consensus building to correct errors on the Ecoli and the human NA12878 datasets. They were able to reduce the errors in the Ecoli and the human NA12878 datasets by 7% and 17.07%, respectively. This means that after their correction process, the dataset still contains 2.03% out of the 20% error exhibited by the dataset.

Kotlarz K. et al. [105] used a combination of Keras and the Naïve algorithm as a machine learning tool that uses array-based genotype information to work second-generation single-nucleotide polymerase datasets. Their algorithm was able to detect errors in the dataset by computing a loss metric. The loss metric comprised the sequence alignment of the dataset with a reference genome. This provided the opportunity to detect errors in the dataset. If an erroneous dataset is determined and the probability of occurrence of the error is very small, the various weights that were used in classifying them as errors are modified to transform some of the erroneous datasets into valid genome variables. Their algorithm produced a performance of 97.94% on the dataset through single nucleotide polymerase.

University of Ghana http://ugspace.ug.edu.gh

2.5 Theoretical Framework of Machine Learning

Neural Networks and Deep Learning

This section introduces artificial neural networks (ANN). An artificial neural network will be classified as a mathematical model that is made up of neurons (interconnected processes) also referred to as mathematical processing units that are capable of imitating the functionality of the human brain [106][107]. The artificial neurons can be modeled as perceptron and fed with various input data types and output binary data for ease of inferring the output data. The network consists of various layers that are capable of transforming the input data. The output of the input layer becomes an input of the next layer in sequence. This process goes on between the inner layers (also known as hidden layers) till an output is produced [84][108].

In some simple cases, a single neuron (also known as perceptron) which consists of single linear classifiers is used to generate the binary output data [109][110]

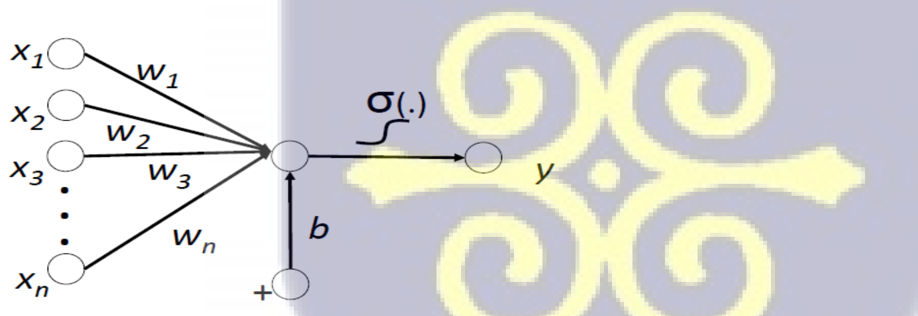


Figure 2.5.1: Perceptron

For more complex problems, a nonlinear model known as multiple layer perceptron (MLP) is generated by assigning weights and biases to the input data to enhance the network performance. Such weights and biases may be automatically generated based on the underlying network architecture. Figure 2.5.1 is the diagrammatic representation of a neuron that takes input vector

University of Ghana http://ugspace.ug.edu.gh

data x and assigns a nonlinear activation function ***f*** that will transform the input data together with their weights and biases to output data. Using an n-dimensional input vector dataset x, which is represented by $x \in R^n$ the output will be given by:

$$a = f(w^T x + b), \tag{2.3.1}$$

w^T represents the transpose of the weighted vector w, and $w^T x$ is the matrix multiplication of the input dataset and the transpose of the weighted matrix. The activation function works on various combinations of summing the biases with the requisite matrix multiplication to generate possible output variables. This allows the model to learn various characteristics of the input dataset. The learning process is also known as training the network. Based on the input dataset and the type of training employed, artificial neural networks may be classified as either supervised [111][112][113] or unsupervised [114][1][115].

Though Figure 2.5.1 looks simple, the training process consists of a combination of factors. The network will have to embark on weight initialization, that is generating random samples of appropriate weights and biases to be assigned to each input dataset. [116], This is followed by forward propagation (also known as feedforward), that is using the activation function σ to generate the output dataset from the combination of weighted and biased input dataset.[117], backpropagation (determining the forward propagation error by learning in reverse order, this time from the generating output data to the input dataset. [118] and perform weight adjustments as it moves from the generated output dataset to the input datasets (backpropagation) where the input dataset is likely to be missed. [119]. For very good network performance, the input dataset has to be prepared to meet the objectives for which it is to be used. The objective is to bring the attributes of the dataset to scale. This is done through the normalization of the dataset.

University of Ghana http://ugspace.ug.edu.gh

Multiple Layer Perceptron

Multiple layer perceptron emanates from a combination of a single perceptron. Here the activation of individual neurons is handled by the vector product of the weight and data followed by the addition of the bias i.e. ai = f(Wixi + bi). Wi and bi are the weight and bias attached to the ith data i.e. xi. The multiple-layer perceptron can also be viewed as a stack consisting of the matrix representation of various neurons as expressed in equations 2.3.2, 2.3.3, and 2.3.4 which has a weighted vector $W_i \in R^n$

$$Z = Wx + b \quad (2.3.2)$$

$$a = f(Z), \quad (2.3.3)$$

Here $w \in R^{mxn}$, $b \in R^m$ and f applied step-by-step.:

$$f(z) = f([z_1, z_2, \ldots, z_m]) = [f(z_1), f(z_2), \ldots, f(z_m)] \quad (2.3.4)$$

The vector transformation of the various combinations of the input data together with the assigned weights and biases generates the output dataset f(z). The layers between the input and output datasets form the hidden layers. Neural networks whose architecture consists of at least three hidden layers are referred to as deep neural networks [120][121][122]. For better analysis, a parameterized matrix representation of the multiple-layer perceptron in a stack format is generated from equations 2.3.5a to 2.3.5b

$$Z^{(2)} = W^{(1)} x + b^{(1)} \quad (2.3.5a)$$

$$a^{(2)} = f(z^{(2)})$$

$$Z^{(3)} = W^{(2)} a^{(2)} + b^{(2)}$$

University of Ghana http://ugspace.ug.edu.gh

$a^{(3)} = f(z^{(3)})$

. . .

$Z^{(l+1)} = W^{(l)} a^{(l)} + b^{(l)}$

$a^{(l+1)}=f(z^{(l+1)})$ (2.3.5b)

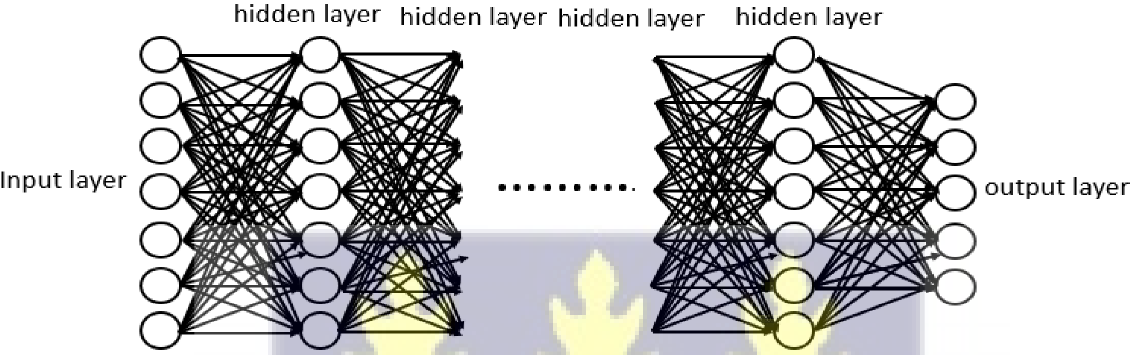


Figure 2.3.2 Network graph of a *(T+1)* layer perceptron *K* input layers, *T* hidden layers, and *M* output layers. Source: J. Lee et al. [124]

Activation Functions

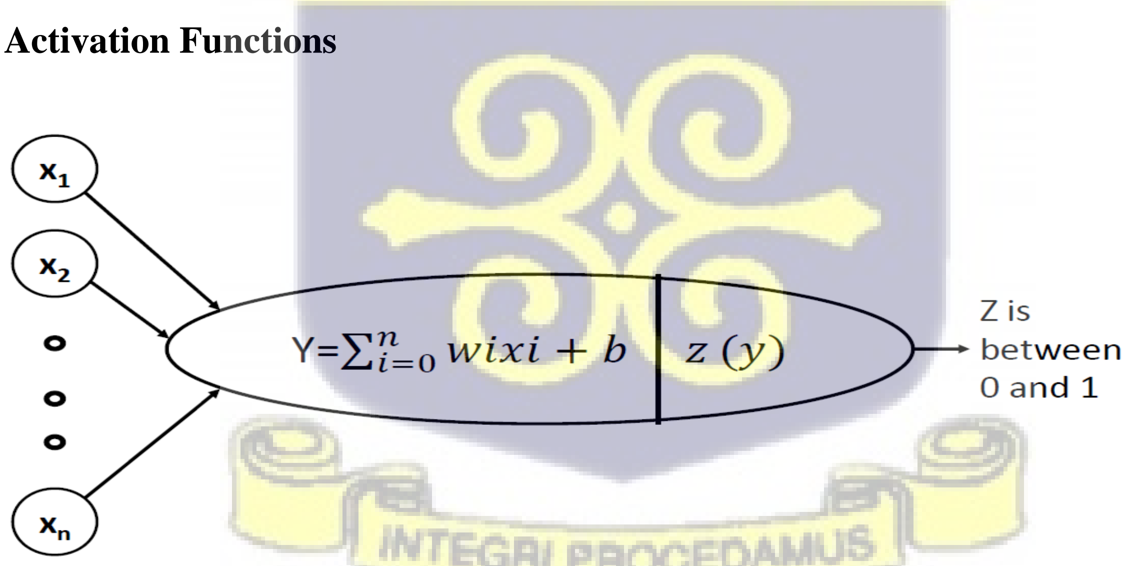


Figure 2.3.3: Activation function of a neural network with n input lines and a probabilistic output data

University of Ghana http://ugspace.ug.edu.gh

The activation function is a mathematical or statistical model that is fed with input data to generate or produce output data based on defined parameters. Depending on the type of experiment and dataset in hand, various activation functions such as the Tanh, Sigmoid, Rectified Linear Unit (ReLU) and Softmax may be chosen to perform that task [123][124][125]. Since the training process involves both feedforward, backpropagation, and nonlinear architecture, the activation function should possess the capacity to be differentiated to support both the backpropagation and nonlinear processes [126]. Large nonlinear input datasets may require longer training time and may also affect the network's performance. This makes the choice of activation function in most machine-learning tasks a crucial event. [127].

The sigmoid (sometimes referred to as logistic) activation function has an s-shaped graph that is differentiable and produces non-negative outputs [128]. The hyperbolic tangent activation function tanh(x) which is also differentiable, linearly transforms the sigmoid activation function onto the interval [-1, 1]. This makes it a better choice than the sigmoid function if negative values are expected to be generated during a learning process. [129]. The major problem of sigmoid and tanh activations is that they show the vanishing gradient effect or saturate when used in machine learning architectures that contain many layers [124][127]. The Rectified Linear Unit (ReLU) is another nonlinear activation function that does not suffer a severe vanishing gradient effect as compared to the sigmoid and tanh activation functions [124][130]. The ReLU activation function is faster when compared with the sigmoid and tanh activation functions because its computational processes do not include the computation of exponentials. This speeds up its performance when used as an activation function. The softmax activation function is generally used to display the output of machine learning processes as a probabilistic distribution [115][131]

University of Ghana http://ugspace.ug.edu.gh

Name	Activation Function	Derivative	Diagram
Sigmoid	$\sigma(x) = \frac{1}{1+e^{-x}}$	$f^{l}(x)=f(x)(1-f(x))^{2}$	
Tanh	$\sigma(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$	$f^{l}(x)=1-f(x)^{2}$	
ReLU	$\sigma(x) = \max\{0, x\}$	$f'(x) = \begin{cases} 0 & \text{for } x \leq 0 \\ 1 & \text{for } x > 0 \end{cases}$	
Softmax	$\sigma(x) = \frac{e^{x}}{\sum_{1}^{j} e^{x}}$	$f'(x) = \frac{e^{x}}{\sum_{1}^{j} e^{x}} - \frac{(e^{x})^{2}}{(\sum_{1}^{j} e^{x})^{2}}$	

Figure 2.3.4: Non–linear, commonly used activation function. The ReLU and softmax functions will be used in this thesis. Source: [134]

Sigmoid – Logistic Regression Activation Function

The sigmoid is a nonlinear activation function whose output ranges between zero and one (Figure 2.3.4). It is used mostly in machine learning classifications. The gradient or slope of the sigmoid function becomes zero when input values are largely positive or largely negative. This is because, for largely positive or negative input values, the change in y is zero; thus, zero divided by any change in x is zero. This means that when used in neural networks, the neurons may not be able to recover. This phenomenon is known as the vanishing gradient effect [132]. During backpropagation, the derivative of the sigmoid function equation 2.3.6 when given values will be between zero and zero point two five $(0 \leq \frac{\partial y}{\partial x} \leq 0.25)$ Mathematically it is expressed as

University of Ghana http://ugspace.ug.edu.gh

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{2.3.6}$$

The derivative of equation 2.3.6 is

$$\sigma^{1}(x) = \frac{\partial}{\partial x}(1+e^{-x})^{-1}$$

$$= e^{-x}(1+e^{-x})^{-2}$$

$$= \frac{e^{-x}}{(1+e^{-x})^{-2}}$$

$$= \frac{1+e^{-x}-1}{(1+e^{-x})^{-2}}$$

$$= \frac{1+e^{-x}}{(1+e^{-x})^{-2}} \frac{-1}{(1+e^{-x})^{-2}}$$

$$= \sigma(x) - \sigma^{2}(x)$$

$$= \sigma(x)(1-\sigma(x)) \tag{2.3.7}$$

Tanh Function

The tanh activation function is similar to the sigmoid or logistic regression function. The only difference is that instead of giving a range between zero and one, it gives an output range between a negative one and a positive one. The major difference between the two is their gradient or slope. That is how much output changes for a given change in input. The tanh function also suffers from the vanishing gradient effect as the gradient of both negatively or positively large input values would be zero[133]. Mathematically the tanh function can be expressed as

University of Ghana http://ugspace.ug.edu.gh

$$\sigma(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.3.8}$$

The derivative or slope of 2.3.8 is given below.

$$\sigma^{1}(x) = \frac{\partial}{\partial x}\frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

$$= \frac{(e^{x} + e^{-x})(e^{x} + e^{-x}) - (e^{x} - e^{-x})(e^{x} - e^{-x})}{(e^{x} + e^{-x})^{2}}$$

$$= \frac{(e^{x} + e^{-x})^{2} - (e^{x} - e^{-x})^{2}}{(e^{x} + e^{-x})^{2}}$$

$$= \frac{(e^{x} + e^{-x} + e^{x} - e^{-x})(e^{x} + e^{-x} - e^{x} + e^{-x})}{(e^{x} + e^{-x})^{2}}$$

$$= \frac{(2e^{x})(2e^{-x})}{(e^{x} + e^{-x})^{2}}$$

$$= \frac{4}{(e^{x} + e^{-x})^{2}}$$

$$= \frac{2}{(e^{x} + e^{-x})^{2}} \frac{2}{(e^{x} + e^{-x})^{2}}$$

$$= \frac{1}{(\cosh)^{2}}$$

$$= (\sec h)^{2} \tag{2.3.9}$$

Rectified Linear Unit – ReLU

University of Ghana http://ugspace.ug.edu.gh

The ReLU activation function, (Figure 2.3.4), is computationally efficient. This is because during backpropagation, its value is either zero or one (derivative is 0 for negative values or 1 for positive values (Equation 2.3.10). This solves the vanishing gradient problem. However, because its derivation is either zero or one, it may create dead neurons when one of the derivatives is zero during backpropagation, where the derivatives are multiplied. This problem is corrected by the introduction of the leaky ReLU (equation 2.3.11). In the leaky ReLU, instead of the output being zero for negative input data, it becomes a small fraction of the input value and thus prevents it from being a zero, which is a constant.

$$f^{1}(x) = \begin{cases} 0 & \text{for } x \leq 0 \\ 1 & \text{for } x > 0 \end{cases} \tag{2.3.10}$$

$$f(x) = \begin{cases} 0.01x & \text{if } \quad x < 0 \\ x & \text{otherwise} \end{cases} \tag{2.3.11}$$

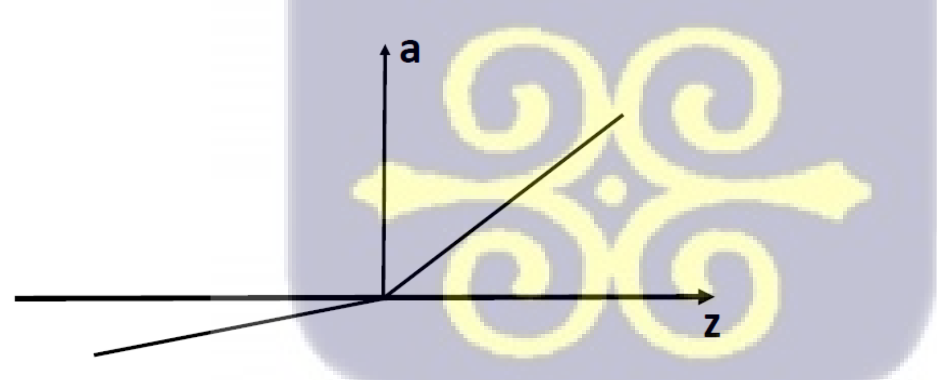


Figure 2.3.5: Diagram of the Leaky ReLU

Softmax

University of Ghana http://ugspace.ug.edu.gh

The softmax activation function from its definition in Figure 2.3.4 will always convert input data into a normalized probability distribution. It can also be said that it forces the output of a neural network to sum to one. It is, therefore, important to place it at the very end of your classification process so that your neural network output becomes probability distribution.

Feed Forward

This is the process of randomly multiplying every input data with an associated weight and adding a bias; this is then followed by the summation of all products [134]. The network is then passed onto the nest layer for the same process to be performed. This goes on for a predefined number of times, generating what has become known as the hidden layers [135]. A fully connected network is established when every output of a given layer becomes connected to every input of the next layer [136]. The network is then passed through an activation function to display the output. layer[137]. There are several activation functions, figure 3.2; however, the choice of an activation function usually depends on the objectives of the task being undertaken [138]. Some activation functions output only a positive value (ReLU) irrespective of the value or size of the input data [139][140][141][142] while others output either positive or negative values (sigmoid and tanh) [143][132][140] (Figure 2.3.4). In Figure 2.3.6, if data is fed into a fully connected dense layer from t to ***T+1***. This means that the fully connected layers architecture has a ***T*** number of layers and ***n*** neurons. Then forward feed between the various layers can be expressed as

$$z_1^l = w_{11}^l a_1^{l-1} + w_{12}^l a_2^{l-1} + \ldots + w_{1j}^{l-1} a_1^{l-1} + \ldots + b_j^l \qquad (2.3.12)$$

$$z_2^l = w_{21}^l a_1^{l-1} + w_{22}^l a_2^{l-1} + \ldots + w_{2j}^{l-1} a_2^{l-1} + \ldots + b_j^l$$

$$\cdots$$

University of Ghana http://ugspace.ug.edu.gh

$$z_i^l = w_{i1}^l a_1^{l-1} + w_{i2}^l a_2^{l-1} + \ldots + w_{ij}^{l-1} a_i^{l-1} + \ldots + b_j^l$$

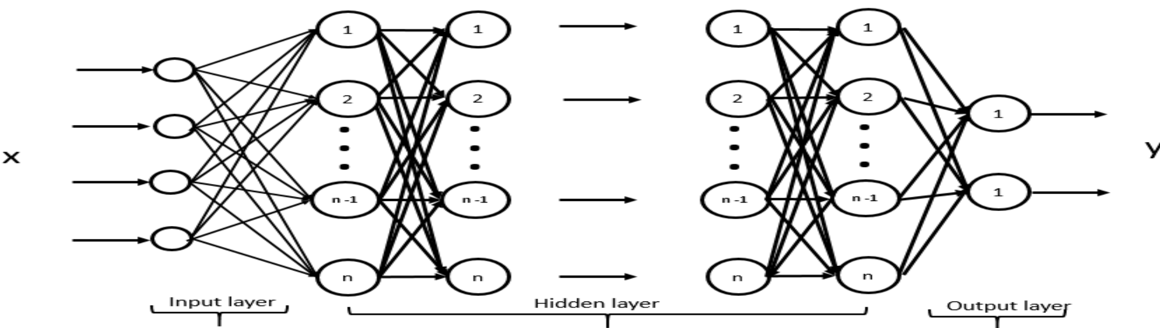


Figure 2.3.6: Feedforward is a fully connected layer. Source: P. Murugan [144]

For any neuron (k) at layer l in a fully connected feedforward network, as shown in Figure 2.3.6

A pooling layer reduces the various input dimensions into a single long vector[145][146][147].

Backpropagation Algorithm

The backpropagation algorithm[148][149] [150][151] learns its way back to the input layer from the predicted output layer by automatically adjusting the various weights and biases. This is done by applying the chain rule for the derivative computation of the composite function. As an illustration, the backpropagation algorithm of equation 2.3.13 will be derived as follows from the top on the assumption for all 1 – s(x) + s(xc) > 0. The derivative of the vector U will be

$$\frac{\partial s}{\partial U} = \frac{\partial}{\partial U} U^T a$$

$$\frac{\partial s}{\partial U} = a, \qquad (2.3.13)$$

University of Ghana http://ugspace.ug.edu.gh

Here $\boldsymbol{a}$ is as defined in equation (2.3.13)

The derivative of the weight W that transforms the layer before the output layer is:

$$\frac{\partial s}{\partial W} = \frac{\partial}{\partial W} U^T a = \frac{\partial}{\partial W} U^T f(x) = \frac{\partial}{\partial W} U^T f(Wx+b)$$

$$\frac{\partial s}{\partial W} = \frac{\partial}{\partial W} U^T a = \frac{\partial}{\partial W} U^T f(x) = \frac{\partial}{\partial W} U^T f(Wx+b) \qquad (2.3.14)$$

For a single weight, $\boldsymbol{W_{ij}}$ found in $\boldsymbol{a_i}$, that is the $\boldsymbol{i^{th}}$ element of $\boldsymbol{a}$, and then have

$$\frac{\partial}{\partial W_{ij}} U^T a \rightarrow \frac{\partial}{\partial W_{ij}} U_i a_i \qquad (2.3.15)$$

The chain rule is used to transform the partial derivative:

$$U_i \frac{\partial}{\partial W_{ij}} a_i = U_i \frac{\partial a_i}{\partial z_i} \frac{\partial z_i}{\partial W_{ij}}$$

$$= U_i \frac{\partial (\mathrm{a}_i)}{\partial z_i} \frac{\partial z_i}{\partial W_{ij}}$$

$$= U_i f^l(z_i) \frac{\partial z_i}{\partial W_{ij}}$$

$$= U_i f^l(z_i) \frac{\partial W_i x + b_i}{\partial W_{ij}}$$

$$= U_i f^l(z_i) \frac{\partial}{\partial W_{ij}} \sum_k W_{ik} x_k$$

$$= U_i f^l(z_i) x_j$$

$$= \partial_i x_j \qquad (2.3.16)$$

University of Ghana http://ugspace.ug.edu.gh

Here ∂_i x_j are the local error and input signals, respectively. For all, ***i*** and ***j*** where ***i = 1, 2,. . ., h*** and j = 1, 2,. . ., n, the cross products between the vectors ∂ $_x$ represent the gradient of ***W***. Although the backpropagation simply demonstrates the taking of partial derivatives using the chain rule, Partial derivatives emanating from higher layers can be used to derive the partial derivative of lower layers[152][153]. This is what the network uses for the training process

Loss Function

The loss function helps map one or many variables onto a real number with an associated cost[154]. The loss function is used to measure the network performance[155], which depicts the inconsistencies between the actual value Y_i and the predicted value $\overline{Y}_i$ [156]. As the loss function reduces, the performance of the model increases[138]. Its architecture includes a one-convolutional layer, a pooling layer, and several dense layers[144]. A variety of loss functions have been developed to suit the underlying classification or clustering process [157]. The major problem of using DNN is to develop an efficient algorithm that will learn the patterns in the data and improve the network performance [158]. To improve results or training processes, several approaches, including batch regularization and normalization [159], proper initialization, activation function, and dropout [160], just to mention a few, have been employed. The proper choice of activation function must not lead to what is known as the vanishing gradient effect[161]. The loss function can be grouped into two main categories (Regression loss and Classification loss) [162]. It is worth noting that there is no one-fit-all loss function approach. A selection of well-publicized loss functions and the platforms where they are commonly used will be discussed in the next section.

University of Ghana http://ugspace.ug.edu.gh

Regression loss functions

Regression loss functions are models used to optimize the performance of regression models such as multiple linear regression, polynomial regression, robust regression, decision tree, random forest, Gaussian process regression, and support vector regression, just to mention a few. These models are used to find the relationship between a dependent (also known as the target) variable and an independent (also known as the predictor) variable. In its simplest form (linear), it is expressed as equation 2.3.16 below

$$\sum_{i=1}^{k} y_i = \sum_{i=1}^{k}\left(a_0 + \sum_{i=1}^{k} b_i x_i\right) \tag{2.3.16}$$

Here y is the dependent variable and x_i is the independent variable. a_0 and b_i are respectively a constant and the gradient (or coefficient) of the model. If the model has one input variable, then i = 1 is referred to as a linear model. If the value of k is equal to p say, then the model is a polynomial with p input variables. All the other models are variants of equation 2.3.16. For instance, multiple representations of equation 2.3.16 will lead to the matrix notation in equation 2.3.17

$$\sum_{i=1}^{k} y_i = \sum_{i=1}^{k}\left(a_0 + \sum_{i=1}^{k} b_i x_i\right) \tag{2.3.17}$$

For large values of the input data, equation 2.3.17 attains robustness; that is when some of the assumptions that establish the regression model are not fulfilled or met, it does not lead to the breakdown of the model.

The Mean Squared Error (L2 Loss)

University of Ghana http://ugspace.ug.edu.gh

The Mean Squared Error (MSE) is mostly used to measure the performance of linear regression models[163][164]. It gradually reduces the gradient as the network converges efficiently to the minimum [165]. It is also sensitive to outliers and may produce higher losses due to variations in the outlier values [166]. A smaller MSE value will, therefore, depict a consistent and efficient algorithm. Suppose $\bar{Y}_i$ the predicted output value after some $\boldsymbol{k}$ training process and $\boldsymbol{Y_i}$ the corresponding actual value, then the MSE can be represented by equation 2.3.18a. MSE is also expressed as the average of the error squared (equation 2.3.18b). This makes the estimator with the smallest MSE more efficient. The MSE becomes zero for a large sample size. This makes the MSE consistent. That is, a large sample will give a more accurate result, that is, a result with a smaller spread around the true value. Equation 2.3.18c, which is a mathematical transformation of equation 2.3.18b, shows that the MSE can be expressed as the sum of variance and the bias squared. (equation 2.3.18c)

$$MAE = \frac{1}{k}\sum_{i=1}^{k}(Y_i - \bar{Y})^2 \quad (2.3.18a)$$

$$MSE(\bar{\theta}) = E\left[\left(\bar{\theta} - \theta\right)^2\right] \quad (2.3.18b)$$

$$= E\left[\left(\bar{\theta} - E(\bar{\theta}) + E(\bar{\theta}) - \theta\right)^2\right]$$

$$= E\left[\left((\bar{\theta} - E(\bar{\theta}))^2 + \left((\bar{\theta} - E(\bar{\theta}))(E(\overline{\theta}) - \bar{\theta}\right) + (E(\bar{\theta}) - \theta)^2\right)\right]$$

$$= E\left[(\bar{\theta} - E(\bar{\theta}))^2\right] + 2E\left[(\bar{\theta} - E(\bar{\theta}))(E(\theta) - \bar{\theta})\right] + E\left[(E(\overline{\theta}) - \bar{\theta})^2\right]$$

$$= E\left[(\bar{\theta} - E(\bar{\theta}))^2\right] + E\left[(E(\overline{\theta}) - \bar{\theta})^2\right]$$

University of Ghana http://ugspace.ug.edu.gh

Since $2E\left[(\overline{\theta}-E(\overline{\theta}))(E(\theta)-\overline{\theta})\right]=0$

$$=Var(\overline{\theta})+Bias(\overline{\theta},\theta)^2 \tag{2.3.18c}$$

Mean Absolute Error

This is also used to measure the performance of a regression function. However, unlike the MSE, it averages the sum of the absolute difference between the predicted variable and the true variable. Its mathematical composition supposes that it does not require the use of squaring the errors[167]. This makes it a better choice of error correction than the MSE as it is very efficient on outliers. Suppose $\overline{Y}_i$ is the predicted output value after some $\boldsymbol{k}$ training process and $\boldsymbol{Y_i}$ the corresponding actual value, then the MAE can be represented by:

$$MAE=\frac{1}{k}\sum_{i=1}^{k}|Y_i-\overline{Y}_i| \tag{2.3.19}$$

Mean Squared Logarithmic Error (MSLE)

The MSLE is similar to the MSE, it is usually adopted when the input variables are normally distributed, and a fair penalty value is imposed irrespective of the error size [168]. Suppose $\overline{Y}_i$ is the predicted output value after some $\boldsymbol{k}$ training process and $\boldsymbol{Y_i}$ the corresponding actual value, then the MAE can be represented by:

$$MSLE=\frac{1}{K}\sum_{i=1}^{K}(\log(Y+1)-\log(\overline{Y}+1))$$

$$MSLE=\frac{1}{K}\sum_{i=1}^{K}(\log(Y+1)-\log(\overline{Y}+1)) \tag{2.3.20}$$

University of Ghana http://ugspace.ug.edu.gh

Huber Loss / Smooth Mean Absolute Error

The Huber loss or smooth means absolute error as defined by equation 4.6.1.4 shows less sensitivity toward outliers in data than the mean square error [169]. This means that the Huber loss is the choice of a loss function in regression modeling when there are outliers (data lying outside or at quite a distance from the regression line compared to other data that are plotted on the regression diagram) in the data set. Again the Huber loss is preferred in the training environment when the gradient is very large. The reason is that the mean square error may miss the minimum at such conditions.

$$|x| - \log(2) \quad \begin{matrix} if\ |y - y_p| \leq \delta \\ otherwise \end{matrix} \tag{2.3.21}$$

Log-Cosh Loss

The log-cosh uses the logarithm of the hyperbolic cosine for predicting the error in a model. Given the actual value of the dependent variable y and the expected value of the dependent variable y_p, the log-cosh loss is defined as shown in equation 2.3.22 below:

$$L(y, y_p) = \sum_{i=1}^{n} \left| \log(\cosh(y^i - y_p^i)) \right| \tag{2.3.22}$$

The log-cosh loss function (log(cosh(x))) approximates to $\frac{x^2}{2}$ and $|x| - \log(2)$ for small and large values of x respectively. Comparing equation 2.3.19 to equation 2.3.22, it can be realized that the mean absolute error is similar to the log-cosh loss except for the outliers

Classification Loss

University of Ghana http://ugspace.ug.edu.gh

Before delving into classification loss, classification will be treated as a map that assigns a class label $y \in \{-1,1\}$ to a feature vector $x \in X$ where X is a feature space. Classification of loss functions is used to optimize classification models that work on labeled data. This means that classification loss and regression loss though both are loss functions used to improve the performance of the underlying models, are different. The major difference between regression and classification loss is that regression loss is used to predict quantities in regression models, while classification loss is used to predict labels in classification models. Below are the classification loss functions used to optimize the performance of classification models

Cross-Entropy (Log Loss)

Given two probability distributions, p, and q on the same underlying dataset, the cross-entropy measures the disorder between the dataset[170]. It does so by comparing each predicted probability to the actual and assigning a probabilistic value between 0 and 1. In classification problems, the MSE is not convex and does not penalize variation in classes. The cross-entropy function is always convex [171]. Mathematically, the cross-entropy loss for discrete and continuous distributions are represented by equations 2.3.23a and 2.3.23b, respectively. The negative sign is because the log Q(x) will always produce a negative value.

$$H(p,q) = -\sum_{x \in X} p(x) \log q(x) \quad (2.3.23a)$$

$$H(p,q) = -\int_{x} P(x) \log Q(x) dr(x) \quad (2.3.23b)$$

University of Ghana http://ugspace.ug.edu.gh

Hinge Loss

The hinge loss (Figure 2.3.7) is another loss function used in training mostly support vector machine (SVM) classifiers [172]. For any given instance, the x-axis represents the distance from the boundary, while the y-axis is the loss size or penalty the function will suffer depending on the length. The number 1 on the x-axis is represented by the dotted line. This means that a positive distance from the right-hand side of the dotted line will incur very minimal or no loss. At the same time, a negative length from the left-hand side of the dotted line will incur loss ranging from small to high depending on its distance from the dotted line. Chen-Yu et al. [173] and Yichuan T [174], in their separate works, demonstrated that a well-fitted squared hinge loss (a variant of hing loss) outperformed log loss in typical classification tasks

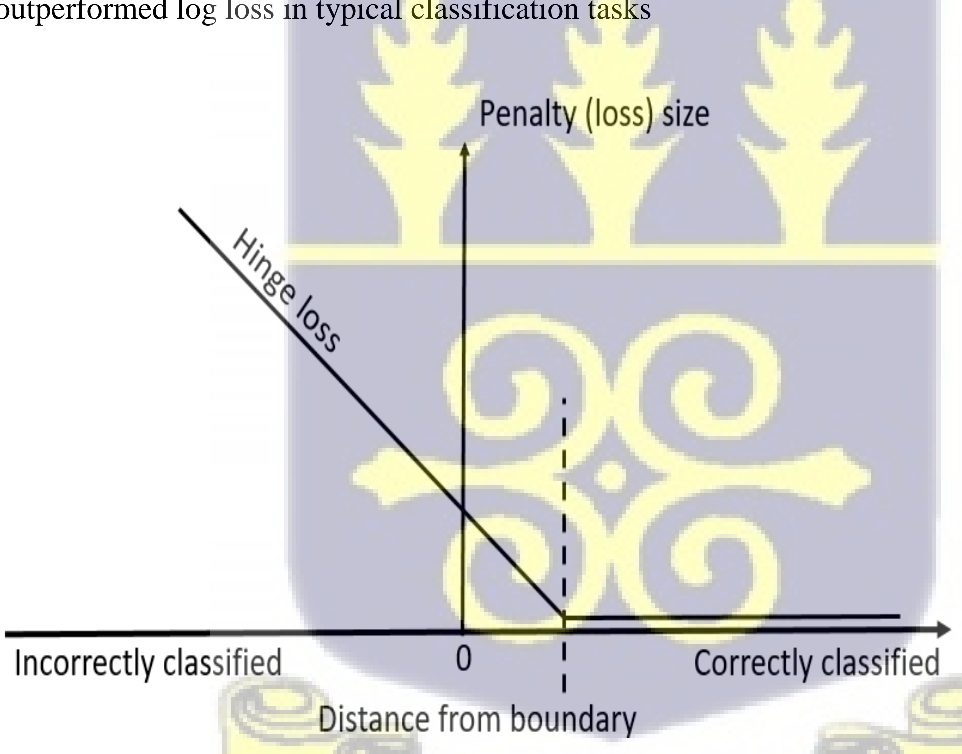


Figure 2.3.7: The hinge loss function shows the distance from the boundary of an instance.

Kullback Leibler Divergence Loss

University of Ghana http://ugspace.ug.edu.gh

The Kullback Leibler Divergence (KLD), also known as relative entropy, is used to measure variations between two probability distributions [175]. It is similar to the cross-entropy loss function. The major difference is that an approximate distribution is added, and the log value of the difference between them is sorted. Mathematically DLD is expressed in equation 2.3.24. This shows the expectation of the log difference between the data in the origin and approximate distribution, respectively.

$$D_{KL}(p \| q) = \sum_{i=1}^{N} p(x_i).(\log p(x_i) - \log q(x_i)) \tag{2.3.24}$$

Focal Loss

In class imbalance problems where a classifier is applied to sparse object locations, the cross-entropy loss is noted to perform poorly [176]. Usually, in object detection, the two-stage approach where initially, sparse sets of object location are generated by the classifier and later classifies each of the sparse locations as either a foreground or background through convolutional neural network has been noted to produce the highest accuracy, outperforming the one-stage approach where the classifier is applied over the sampling of dense and regular candidate location without segregation. This means that the one-stage approach deals with larger object locations when sampled across the dataset. Meanwhile, only a few of the detected candidate locations contain objects that can be inferred. Although the one-state approach is simpler compared to the two-stage counterpart, it trails the two-stage approach in performance. This drew the attention of researchers to research ways that may lead to improving the performance of the one-stage approach. This led to the development of the Focal loss function. The focal loss is used to solve the challenges that the one-

University of Ghana http://ugspace.ug.edu.gh

stage object detection approach faces when there is the utmost imbalance disparity between the foreground and background of the training dataset. The focal loss is expressed mathematically as equation 2.3.25. includes a tunable focus factor to alter the cross-entropy equation 2.3.23.a

$$FL(pt) = -(1 - pt)^{\gamma} \log(pt) \tag{2.3.25}$$

Here $(1 - pt)^{\gamma}$ is a modulating factor added to the cross-entropy equation 2.3.23a and $\gamma \geq 0$ is the tunable focusing factor?

This means that under misclassification, if pt is small, the modulating factor ($(1 - pt)^{\gamma}$) is approximately equal to 1, thus unaffecting the loss. It can be deduced that as $pt \rightarrow 1$ (approaches 1), the modulating factor becomes 0, thus downweighting the loss for well-classified situations. From equation 2.3.24, it can be noted that as the tunable focusing factor $\gamma = 0$, the focal loss equation 2.3.24 becomes equal to the cross-entropy equation 2.3.23a under the probability distribution $P(x)$

Exponential Loss

The exponential loss function is another classification loss function used to improve the performance of classification algorithms. The exponential loss function, which is defined as equation 2.3.25a, is an upper bound function on 0-1 equation 2.3.25b. This means that the exponential function $l_{\exp}(yh(x)) : \mathbb{R} \rightarrow \mathbb{R}$ is a positive and convex function [177]. This means that if all the training datasets are correctly classified, then the exponential loss is zero.

$$l_{\exp}(h(x),(x, y)) = e^{-yh(x)} \tag{2.3.25a}$$

University of Ghana http://ugspace.ug.edu.gh

$$l_{\exp}(x, y) \geq l_{0-1}(x, y) \qquad (2.3.25b)$$

However, the exponential loss function is noted to exhibit deficient performance when sampled over the entire dataset, and also when it reaches zero training error, it is expected that any addition of new weak should lead to overfitting. However, the opposite occurs. At zero training error, the addition of new but weak classifiers improves the performance of the exponential loss function [178]. This problem is solved by introducing early stopping, where the data set is divided into two, one for training and the other for testing. The training dataset is used to train the network; however, after each training process, the binary loss is computed before the next training session. A comparison of the various loss values shows improvement in the binary loss values after each training session. When the training, computation, recording, and comparing of the loss values are continuous, at a point, the loss value begins to deteriorate. At that point, the training process is stopped, and this is referred to as the early stopping stage. Other variants such as adaptive boosting (Adaboost) of the exponential loss have been introduced to improve the performance of the exponential loss function.

Avoiding overfitting

When a machine learning algorithm (network) memorizes the dataset during the validation process instead of generalizing it, then the network is said to be overfitting the dataset. Overfitting networks produce poor network performance during the inference stage. According to S. Santurkar [179] Batch normalization and dropout are used to avoid overfitting your system. The batch normalization targets a particular batch and controls that batch's first and second moments [180]. In batch normalization, values of randomly selected neurons are reduced to zero, a phenomenon known as a dropout. This enables the creation of new learning pathways to reduce the network's ability to memorize the training data during the validation process. Two other alternatives to

University of Ghana http://ugspace.ug.edu.gh

solving the network overfitting problem are by randomly adjusting the weights assigned to the dataset to zero (drop connect) or increasing the size of the input data. This forces the network to relearn the paths traversed by the network during the training process.

Convolutional Neural Network (CNN)

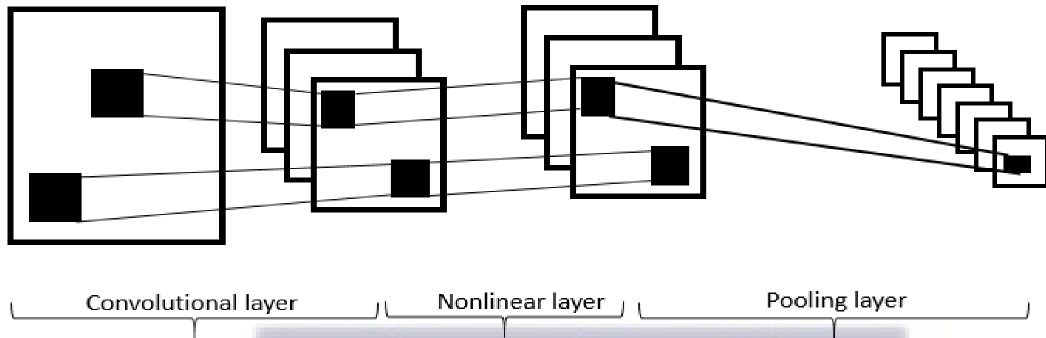


Figure 2.3.8: Convolutional neural network

CNNs were initially designed to process multiple datasets, which were usually in the form of two-dimensional images (multidimensional arrays), and they mimic the visual cortex of the human brain[181]. A hierarchy of two basic cell types referred to as simple and complex cells can be found in the visual cortex.[182]. The single cell works on basic patterns while the complex cell analyzes the information passed out from the single cell to identify more complex patterns. CNNs have been successful as a machine learning architecture and have been applied to solving various problems due to the way it is structured. According to G. Rostami [183], the structure consists of three main sections: local connectivity, invariance to its location, and invariance to local transitions. Another major success of CNN is its ability to filter the input data before it is used to train the neural network. The filters are a combination of multipliers that identify specific attributes of the dataset. The filtering allows any features in the dataset to be visualized. The features are then used in the feature identification processes.

University of Ghana http://ugspace.ug.edu.gh

The identified features are used to break the input dataset into enhanced subsets of the dataset using a process known as pooling. All of the features of the input dataset are transferred to the enhanced subsets (or pooled data). This is successfully done by passing randomly initialized filters over the input dataset. The network then matches the results from the pulled data and then passes them on to the next layer for further learning. With time, the filtering process through feature extraction learns all characteristics of the input dataset without restraining itself to basic patterns of the input data and outputs data that has the best matches of all characteristics of the input data. This has allowed CNN to be successfully applied to areas such as voice recognition, feature extraction [184], natural language processing [185], genome sequencing [186], image recognition, and computer vision [187][188]

Machine Learning Frameworks

The introduction of machine learning in the not-too-distant past initially for performing simple sorting and statistical computations has evolved into a very dynamic discipline being applied to various disciplines with an ever-increasing application domain where new application frontiers are emerging now and then. It has currently been successfully applied to natural language processing, text-to-speech synthesis, speech-to-text synthesis, computer vision, video surveillance, computer vision, product recommendations, and genomics just to mention a few. Various theoretical underpinnings have been formulated, developed, and tested to produce reliable outputs or predictions in many application domains. This has led to the characterization of machine learning application strategies as supervised, unsupervised, and reinforcement learning [189], [190]. Various protocols have been outlined to enable a particular machine learning process to qualify to be characterized as supervised, unsupervised, or reinforcement learning. Supervised learning, also referred to as classification, deals with labeled data to do the learning. The labeled data comes with

University of Ghana http://ugspace.ug.edu.gh

tagging the data (labeling) with specific identifiable properties or characteristics that can be used for the learning (classification). The data used for unsupervised learning is unlabeled; that is, there are no specialized properties or characterizations associated with the data. For reinforcement learning (associated with gaming), the model or intelligent agent learns from its environment to take action that will yield some maximum reward [191]. Algorithms are needed to support and optimize the activities of the various machine learning processes. This opened a wide network of research and sometimes collaboration between academia and research firms, all in the quest of generating and supporting algorithms to either the general machine learning process or targeting specific areas in the machine learning domain. Another area in machine learning (semi-supervised learning) that is attracting much attention is putting unlabeled and labeled data together and then learning performed on it.

Based on the above descriptions of the various machine learning processes, it suffices to say that machine learning (ML) relies heavily on the generation of algorithms to perform identified tasks. The algorithms may be complicated and sometimes difficult to comprehend. Not only do the algorithms perform a specific task, but they are also expected to have significantly high performance and be efficient in dealing with the task they are employed to do. This means that only a few persons who have the desired skills and ability to put these algorithms together will be privileged to design ML programs. To help alleviate this challenge and make ML accessible to many, machine learning frameworks (MLF) were generated. MLFs are libraries or interfaces that help build and deploy machine learning models easily and faster. Some MLFs may require specialized hardware (multiple processors MCPU) support; some may require specialized graphics procession unit (GPU), and others require MapReduce (a programming paradigm for implementing extensive data-parallel processes on the Hadoop framework). Most of these

University of Ghana http://ugspace.ug.edu.gh

frameworks allow the efficient optimization of mathematical equations over multidimensional arrays [192]. Some of the frameworks are very good at training a network, and some are good at deployment, while others are good at linking the training models to the deployment models (referred to as intermediary models) [193]. These frameworks have greatly influenced and enhanced the development and implementation of most successful applications in domains such as computer vision, speech recognition, and natural language processing [194].

Hyperparameter Optimization

Machine learning models are mathematical functions formulated to transform different types of data with the optimum aim of learning from the data such that when it encounters similar but different data, it can draw inferences or conclusions on the new data based on the learning experience from the old data. For example, for a simple linear regression model $y = w^T x + b$, where x is a vector representing features of the data, w^T is the transpose of the weighted matrix, which normally is the gradient or slope of the linear model, and b the bias (a constant). The model will be used to predict the targeted scalar y after learning from the data to present the line of best fit. This means that various lines may be generated to represent the model but, the optimum line (also referred to as the line of best fit) for the model is what is desired. This means that a combination of various weights and biases will have to be tried to arrive at the line of best fit. The task here is determining what values of w^T and b will bind with the data to generate the line of best fit. For more complex models such as convolutional neural networks (CNN) or recurrent neural networks (RNN), the optimization task may not be as trivial as may seem in the linear model presented above. The reason is that various functional mathematical transformations comprising matrices of various dimensions and activation functions may be involved during the feed-forward process. Another complex gradient calculation, either intuitively or dynamically, may be handled during

University of Ghana http://ugspace.ug.edu.gh

the backpropagation process may be involved. It may also be parameters that may control the model's complexity. These may include parameters that affect the learning procedure initialization conditions, step sizes, and learning decay. Fine-tuning a model (forward feed and backpropagation process) may be a herculean task. However, selecting a combination of the model's characteristics that will help produce the best (optimum) output is referred to as hyperparameter optimization. David Maclaurin et al. [195] have it that these parameters should be chosen to optimize validation loss after the model's parameters have been thoroughly trained. They also demonstrated that the automatic tuning of hyperparameters increases performance tremendously. However, building the model should be of utmost concern before hyperparameter optimization; the reason is that without the model, there will be no hyperparameter optimization or tuning. In light of this, two main types of parameter tuning may be considered. One can be done based on input data (such as the weight of neurons) referred to as the model parameter, and the other cannot be directly estimated from the data learning process and has to be declared before the learning process begins. For example, in support vector machines (SVM), it is referred to as the penalty rate P or learning rate in neural networks. This means that there are different hyperparameter considerations based on the type of model used (categorical, discrete, and continuous)[196]. Most of these hyperparameter tuning have been manually done, a process that can be tedious and frustrating to embark on, especially when the model consists of many hidden layers and may have non-differentiable or non-convex gradient problems. Automatic hyperparameter optimization is the current optimization under consideration[195].

INTEGRI PROCEDAMUS

Mathematically, optimization can be treated or modeled as finding the extremum (best solution) from a set of possible solutions that either minimizes (constrained) or maximizes (unconstrained) an object function. To be able to associate a constraint to a machine learning problem, the various

University of Ghana http://ugspace.ug.edu.gh

categorization methods (supervised, semi-supervised, unsupervised, and reinforced learning) must be considered. For supervised learning, you may be looking at either classification (image classification) or sentence (natural language processing) problems. In the case of unsupervised learning, you may be considering either clustering or dimension reduction problems, while reinforcement learning may be considering the Markov decision process [197].

In supervised learning, where labeled data is used, the purpose of optimization is to derive a mapping function (optimal) that will minimize the loss of the training data. This can be represented by equation 2.3.26 below.

$$\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} L(y^i, f(x^i, \theta)) \qquad (2.3.26)$$

Here N represents the number of training samples, Θ is a parameter of the mapping function, x^i is the feature vector of the ith sample, y^i is the corresponding label, and L is the loss function. Depending on the algorithm and the task on hand, a variety of classification loss functions spanning cross-entropy, hinge loss, the square of Euclidean distance, contrast loss, information gain, etc. In regression models, for example, minimizing the square of errors (square of the Euclidean space) on the training samples. For support vector machine (SVM), a supervised machine learning algorithm, for instance, a regularization tool, has to be added to minimize overfitting during the validation process equation 2.3.27.

$$\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} L(y^i, f(x^i, \theta) + \lambda \|\theta\|_2^2 \qquad (2.3.27)$$

Here $\lambda \|\theta\|_2^2$ is a regularization tool that is added to check overfitting during the validation process.

University of Ghana http://ugspace.ug.edu.gh

Optimization is needed in the semi-supervised learning frameworks, where a mixture of labeled and unlabeled data is used for training purposes. This means different types of training and validation processes such as classification [198], dimensionality reduction[199], regression [200], and clustering tasks [201]. Because of this, specialized algorithms (graph-based, multi-learning, semi-supervised support vector machines) that can handle various scenarios are used in this domain. To formulate the optimization process, Both the labeled, and unlabeled data types must be factored into the training and validation process. Suppose $D^l = \{(x^1, y^1), (x^2, y^2), \ldots, (x^l, y^l)\}$ represent the labelled data and $D^U = \{x^{l+1}, x^{l+2}, \ldots, x^N\}$ be the unlabeled data where $N = l + u$. To optimize both the labeled and unlabeled data, the support vector machine is modified by adding the slack variable ζ^i constraint to it [201]. The constraint consists of the misclassification error $\in^i$ of the unlabeled instance, given that the true value of the unlabeled instance is z^i. The slack variable constraint is the mean value of the misclassification error ($\in^i$) and the true value (z^i) that is $\sum_{j=l+1}^{N} \min(\in^i, z^j)$ the whole idea here is to reduce the misclassification error to the barest minimum. Therefore, the optimization of the semi-supervised learning will be formulated by equation 2.3.28.

$$\min \|\omega\| + c\left[\sum_{i=1}^{l} \zeta^i + \sum_{j=l+1}^{N} \min(\in^i, z^j)\right] \tag{2.3.28}$$

Subject to: $y^i(w.x^i + b) + \zeta^i \geq 1, \zeta \geq 0, i = 1,\ldots,1, \quad w.x^j + b + \in^i \geq 1, \in \geq 0, j = l+1, l+2, \ldots, N$ and $-(w.x^j + b) + z^j \geq 1, z^j \geq 0$. Where C is the penalty coefficient

For optimization in unsupervised learning algorithms, first, an analysis of how the algorithm works is conducted, followed by an examination of the optimization process. The algorithm divides the

University of Ghana http://ugspace.ug.edu.gh

sample data into a group of clusters, minimizing differences within clusters but ensuring that the differences between individual clusters are as large as possible (maximizing the differences between clusters)[202]. If the algorithm generates K clusters, for instance, then equation 2.3.29 will be used to optimize the clustering algorithm.

$$\min_{S}\sum_{k=1}^{K}\sum_{x\in S}\|x-\mu_k\|_2^2 \qquad (2.3.29)$$

Where x represents the feature vector of the samples, μ_k represents the center of cluster K and S_k represents the set of samples for cluster K. Looking at Equation 2.3.29 critically shows that the sum of variance between them is being minimized. It is also possible not only to minimize the variance between the clusters but also to minimize the dimensionality between the clusters to ensure that data information within clusters is maintained. The optimization process that takes care of the dimensionality is formulated in equation 2.3.30 below.

$$\min\sum_{i=1}^{N}\left\|\overline{x}^{i}-x^{i}\right\|_2^2 \qquad \text{where } \overline{x}^{i}=\sum_{j=1}^{D'} z_j^i e_{j,} \text{ and } D \gg D' \qquad (2.3.30)$$

Here N is the total number of samples, x_i is a D-dimensional vector $\overline{x}^{i}$, and is a reconstruction of $x^i.z^i=\{z_1^i, z_2^i,..., z_{D^i}^i\}$ that is a projection of x^i in the D'-dimensional coordinates. At the same time, $e_{j,}$ it represents the standard orthogonal basis under D'-dimensional coordinates. A probabilistic optimization approach has been considered if the underlying algorithm for clustering is Bayesian. In this approach, the optimum value of the probability density function p(x) that will maximize the logarithmic likelihood function of the training samples is used as the optimizer equation 2.3.31 below.

University of Ghana http://ugspace.ug.edu.gh

$$\max \sum_{i=1}^{N} \ln p(x^i;\theta) \tag{2.3.31}$$

Here Θ is a Bayesian prior distribution that minimizes or reduces overfitting, optimization for reinforcement learning is different from that of supervised, semi-supervised, and unsupervised. The general reason is that in reinforcement learning, the goal is for the model to come out with the best prediction based on its interaction with its environment. This best prediction may be determined by an objective function whose output may either be minimum or maximum based on the parameters of the model [203]. These can be considered deterministic and in-deterministic (or uncertain) strategies. Whether the process is deterministic or uncertain, the optimization process is determined by the actions of a policy function $a = \pi(s)$. Equation 2.3.32 formulates the optimization of reinforcement learning based on the policy function.

$$\max_{\pi} V_{\pi}(s) \quad \text{where} \quad V_{\pi}(s) = E\left[\sum_{k=0}^{\infty} \gamma^k r_{i+k} \,|S_t = s\right] \tag{2.3.32}$$

Here $V_{\pi}(s)$ represents the value of the function of state S under the policy π while r is the reward and the policy γ

Gradient Descent Optimization

Machine learning frameworks learn through feed-forward and backpropagation processes. The backpropagation process is how the algorithm learns in reverse order to get to the input layer by adjusting the weight and biases to improve the algorithm's performance. This backpropagation process is actually a way of finding derivatives from the output layer back to the input layer. The optimization of the gradient components (gradient descent) will minimize the cost function to the barest minimum. This can be treated as a way of minimizing an objective function $f(\theta)$ that is

University of Ghana http://ugspace.ug.edu.gh

parameterized by the model's parameters $\theta \in \mathbb{R}^d$. The parameterization is done by updating the parameters, which are direction-wise, opposite to the gradient of the objective function $\nabla_\theta f(\theta)$ concerning the parameters [204]. In this regard, the number of steps that will be taken to reach a local minimum is determined by the learning rate ρ. It can also be interpreted as descending to the bottom (minimum) by following the path along the surface as created by the objective function. Due to the quality and robustness of the gradient descent processes, different variants (batch gradient descent, stochastic gradient descent, and mini-batch gradient descent) have been developed to augment its use in different machine learning processes.

Since each update of the gradient involves performing the gradient concerning the objective function with the underlying parameters, batch gradient descent breaks the processes into the mini process intending to speed up the gradient computation process equation 2.3.33.

$$\theta = \theta - \rho.\nabla_\theta f(\theta) \tag{2.3.33}$$

Here θ is the model's parameter, ρ is the learning rate, $f(\theta)$ is the objective function, and $\nabla_\theta f(\theta)$ is the change in the objective function concerning the parameter? From equation 2.3.33, the best performance is achieved when $\nabla_\theta f(\theta)$ it gets closer to zero(0). Although the batching process leads to the convergence to either local and global minimum for convex and non-convex surfaces, for a large dataset, a lot of memory is required to perform such computations making it unfriendly to use in devices that have limited memory resources and slower for online learning [205]. The large memory requirement is a result of the redundant computation. To alleviate this challenge, stochastic gradient descent was introduced.

University of Ghana http://ugspace.ug.edu.gh

The stochastic gradient descent, on the other hand, updates its parameters after each training process. This leads to the elimination of the redundancy that occurs in the batch gradient descent processes. The stochastic gradient descent can be formulated as shown in equation 2.3.34 below.

$$\theta = \theta - \rho.\nabla_{\theta} f(\theta; x^{i}; \mathrm{y}^{i}) \quad (2.3.34)$$

Here, θ is the model's parameter, ρ is the learning rate, $f(\theta)$ is the objective function, and $\nabla_{\theta} f(\theta)$ is the change in the objective function concerning the parameter, x^{i} and y^{i} are the ith and jth input and output variables, respectively. However, when the learning rates ρ slowly reduced, the convergence of stochastic gradient descent and batch gradient descent behaved the same way by converging respectively to a global and local minimum for convex and non-convex surfaces.

Mini-batch gradient descent equation 2.3.35 is used to parallelize stochastic gradient descent in both synchronous and asynchronous settings by reducing communication costs.

$$\theta = \theta - \rho.\nabla_{\theta} f(\theta; x^{i:i+n}; \mathrm{y}^{i:i+n}) \quad (2.3.35)$$

Here, θ is the model's parameter, ρ is the learning rate, $f(\theta)$ is the objective function and $\nabla_{\theta} f(\theta)$ is the change in the objective function concerning the parameter, $x^{i:i+n}$ and $\mathrm{y}^{i:i+n}$ are the input and output batches, respectively.

However, if the mini-batch size increases, it decreases the rate of convergence of the gradient [206]. This is because, in parallelized or distributed settings, the machines involved would have to communicate with each other on shared variables or parameters to efficiently synchronize the shared resources. This means that for large training and validation data sets, scarce resources such

University of Ghana http://ugspace.ug.edu.gh

as memory will be a problem as its size is fixed, but the communication between the computing devices increases with an increase in data size. To alleviate the increase in communication concerning increasing input data size, Sarit Khirirat et al. [207] introduce two optimizations (unconstrained and regularized optimization) methods. In the unconstrained method, they treated the problem as a way of minimizing a convex function (equation 2.3.36) with L_i-Lipschitz continuous gradient

$$\min_{x \in \mathbb{R}^n} f(x) \triangleq \sum_{i=1}^{m} f_i(x) \qquad (2.3.36)$$

Here $f_i : \mathbb{R}^n \mapsto \mathbb{R}$ is convex with L_i-Lipschitz continuous gradient μ, for each i and f, μ is convex. In the regularized optimization method, they added a smooth convex function to the mini-batch function (equation 2.3.37) and minimized the sum using the Li-Lipschitz, which approximated.

minimize $f(x) + h(x)$ (2.3.37)

Here $f(x) = \frac{1}{m} \sum_{i=1}^{m} f_i(x) \mu$ will be represented as a strongly convex function μ, and each $f_i : \mathbb{R}^n \mapsto \mathbb{R}$ has a continuous Li-Lipschitz gradient and $h : \mathbb{R}^n \mapsto \mathbb{R}$ is a convex function but not smooth. They then went further to show mathematically (initially) and later with numerical experimentation by minimizing equation 2.3.37. D. Randall Wilson et al. [205] have it that the size of the learning rate plays a critical role in the efficiency of an algorithm. When the learning rate is too large, it delays the learning process and steps over the valley in the gradient descent computation process, thereby mission curves that may contain errors, creating the general impression that the gradient descent process is performing well. Under such situations, generalizing the accuracy becomes worse, and takes too long to train the network. This is mainly

University of Ghana http://ugspace.ug.edu.gh

because it tends to unlearn what has already been learned, thus increasing the computational time unnecessarily. On the other hand, very small learning rates waste computational time by taking many steps to move between weights, a process that could have been accomplished with fewer steps but for a lower learning rate.

Bayesian Hyperparameter Optimization

The Bayesian hyperparameter optimization technique models the performance (or accuracy) of an evaluated metric y and a hyperparameter λ based on conditional probability $p(\mathrm{y}|\lambda)$. Under this Bayesian process, three main models (Sequential model-based algorithm configuration, Tree-Structure Parzen Estimator, and the Spearmint) were used to optimize such algorithms [208][209]. These three optimization approaches on the Bayesian model all employ different strategies in the optimization process. In modeling $p(\mathrm{y}|\lambda)$, the Sequential model-based algorithm uses a random forest as a Gaussian distribution approach, the Tree-Structure Parzen Estimator uses the tree-structure Parzen density estimator, and the Spearmint also uses a Gaussian model by slicing samples over Gaussian processes. In all these cases, the major task is the ease of querying and updating the Bayesian model either as a non-parameterized or parameterized model. For nonparametric Bayesian models, the weights are initially restricted in Bayesian linear regression after which a kernel process is used to build a Bayesian nonparametric regression model. To be able to do this successfully, an assumption of a fixed observation variance σ^2 is made after which a zero-mean Gaussian priori on the regression coefficient $p(w|V_0) = N(0,V_0)$ is determined thus, paving the way to analytically integrate the weights without losing it Gaussian attributes equation 2.3.38

University of Ghana http://ugspace.ug.edu.gh

$$p(\mathrm{y}|X,\sigma^2) = \int p(\mathrm{y}|X,w,\sigma^2)p(w|0,V_0)dw$$
$$= \int N(y|Xw,\sigma^2 I)N(w|0,V_0)dw \qquad (2.3.37)$$
$$= N(y|0, XV_0 X^T + \sigma^2 I)$$

Here $p(\mathrm{y}|X,\sigma^2)$ is the probability of y given that X, and σ^2 exist

Evaluation of Deep Learning Frameworks

Deep learning frameworks have been widely used in both academia and industry, sometimes using the same dataset and metrics to conclude a concerning situation of interest such as medical condition, plant genomics, and image processing, just to mention a few. However, depending on the framework used, the performance, which relies on the use of computational resources, time to finish the desired computation, or viability of the output (inference), varies between the frameworks. Most consumer devices such as mobile or handheld devices do not have the hardware capacity (disk space, memory) to execute these frameworks on them. Although through the use of the internet and some specialized data centers, mobile devices can transfer the computational part seamlessly using virtual resources to be done at these data centers or specialized platforms with the capability of performing such computation virtually on behalf of the mobile device and transfer the output when done back to the mobile device. It is, therefore, important to assess the performance of the most used frameworks [210] given the same data and conditions to operate on and, more importantly, whether they support the virtualization of the processes to enhance the use of such frameworks on mobile devices. Using data from the modified national institute of Standards and Technologies (MNIST) database and various processors spanning one to six cores, twelve GPU threads, GPUs enhanced with CUDA deep neural networks (cuDNN), and various training scenarios where the input data and architecture varied (incorporating deep neural network), Zhaobin Wang et al. [211] compared the performance of, Keras, Theano, CAFFE2,

University of Ghana http://ugspace.ug.edu.gh

TensorFlow, PyTorch, Deep Learning 4J (DL4J), CNTK and discovered that when the dataset was small and a simple architecture was used, Keras outperformed all the other framework in terms of the time to complete both the forward feed and backpropagation processes. However, the performance of Keras dwindled as the dataset size increased and architecture became complex; Theano became the best framework in their standard experiment. When the testing process was increased to include parallel processing on GPU, DL4J became the dominant framework and outperformed all the other frameworks. When other wrappers such as SciKit-learn and last short-term memory (LSTM) were incorporated, CNTK outperformed all the other networks in terms of both forward feed and backpropagation time.

Shauhuai Shi et al. [212] also compared the performance of CAFFE, CNTK, TensorFlow, Theano, and Torch given the same dataset. Two different networks (simple – CPU only and complex - GPU) were built for each framework. The Torch had a better performance with just four threads than the other frameworks running on more threads. With helper libraries like AlexNet on one CPU with four threads, CAFFE had the best performance. When the CPU threads were increased to 16 on the same AlexNet architecture, TensorFlow had the best performance. When the frameworks were made to run recurrent neural network (RNN) architecture, CNTK outperformed the other frameworks but, the performance of TensorFlow became better than the other frameworks in a thirty-two CPU thread server environment. Initial performance of the various frameworks on GPUs showed that CNTK had the best performance with both the feedforward and backpropagation processes; however, as larger networks such as ResNet-50 were introduced, CNTK could not handle the large computation mini-batch processes and crashed. TensorFlow performed best in this circumstance, demonstrating that TensorFlow can manage large processes on GPUs but performs poorly on single CPU cores.

University of Ghana http://ugspace.ug.edu.gh

In the case of multiple GPUs, specialized convolutional neural networks capable of handling larger layers of the dataset without vanishing gradient effect such as datAlexNet, GoogleNet, and ResNet were used on CAFFE, CNTK, MXNet, and TensorFlow. It was discovered that CAFFE, MXNet, and TensorFlow achieved linear scaling from one GPU to two GPUs. On the whole, CNTK underperformed when it comes to the use of multiple GPUs because it does not parallelize its gradient computation and aggregation.

Optimization of Distributed Deep Learning Frameworks on GPUs

Given a large dataset, a deep learning framework can learn the feature representation of the dataset well and draw good inferences on a similar but unknown dataset. However, as the size of the data set increases, more computational resources such as larger hard disc space and large memories are required to support the smooth manipulation (computation) of data. These hardware resources are usually unavailable in most computing environments, thus hampering the smooth execution of the task at hand. GPUs offer an alternative to this hardware resource challenge by providing high processing speed and space to accommodate the computational requirements of these frameworks. Also, it is possible to enable parallelization of distributed large computational processes on GPUs, thus enabling high throughput with the assistance of high-performance deep neural network frameworks such as NVIDIA's Cuda Deep Neural Network (cuDNN) [213]. Depending on the type of framework being used, scalability across multiple platforms and GPUs is readily available; therefore, optimization of parallel and distributed cross-platform (GPU) utilization for computational processes is necessary [214]. Since stochastic gradient descent (SGD) is widely used in optimization processes on a single machine, to effectively extend its capabilities to distributed networks or GPUs, a little modification (granting it the opportunity to perform mini-batch iteration) of the SGD is required. This involves the process of reading mini-batch data into

University of Ghana http://ugspace.ug.edu.gh

memory, transferring the data read into memory to GPU, and launching the GPU kernel to perform layer-by-layer feedforward and back propagations within the GPU. The symmetrical (also referred to as synchronous) gradient descent (S-SGD) is the extension of the stochastic gradient descent that handles gradient descent processes on GPU. In this regard, the backpropagation (first-order) is done for the weight assigned to the data in the feedforward stage. A successful first-order backpropagation process which is derived from the chain rule, then leads to the update of the model's gradient. A consideration of the total time of an iteration can be computed as shown in equation 2.3.39

$$t_{iteration} = t_{io} + t_{h2d} + t_f + t_b + t_u \quad = t_{io} + t_{h2d} + \sum_{i=1}^{K} t_f^k + \sum_{i=K}^{1} t_n^l + t_u \qquad (2.3.39)$$

Here $t_{iteration}$ is the total time taken to complete an iteration, t_{io} the time taken to input data and output data in each iteration, t_{h2d} the time to transfer data from the CPU memory to GPU memory in each iteration, t_f is the time of the forward phase of each iteration, t_b the time taken for the backward phase of each iteration and t_u is the time taken to update the model in each iteration? This means that for a K-layer mini-batch training using symmetric stochastic gradient descent on GPU, during each iteration, each member of the k-layers will simultaneously load training data (mini-batch), transfer the training data onto the GPU and perform the feed-forward operation from the first layer to the last layer (K) on the GPU, this is followed by back layer propagation from last layer (K) to the first layer, after which the model is updated. That is, the GPU performs in parallel, feedforward, and backpropagation with different data sets. Various optimization processes on GPUs have been implemented.

University of Ghana http://ugspace.ug.edu.gh

Hao Shang et al. [215] introduced a hybrid communication process to improve the performance of communication and computations in GPUs. Looking at the feedforward and backpropagation communication and computation processes which are repeated severally, introducing a hybrid method of optimizing, especially the wait-free backpropagation process. That is, they considered the backpropagation and feedforward processes as independent, and one need not wait for the other in parallel or distributed processes. Considering the forward feed and backpropagation through layer L of a network as f_t^L and b_t^L respectively, which follow that a computational step which occurred at iteration t is considered as $C_t = f_t^1, f_t^2, ..., f_t^L, b_t^L, b_t^{L-1}, ..., b_t^1$. Then the wait-free backpropagation kicks in as soon as the feedforward is completed for the particular layer since the two processes are independent. It was realized that wait-free backpropagation was successfully implemented to improve computational time. It, however, discovered that it works best with fully connected layers when their parameters are concentrated at the upper layer but, when the parameters are concentrated at the lower layers, such as convolution networks and tree-like data structures, they perform poorly. It is for this reason that hybrid communication and computation in GPU were introduced. It combines the best parameter server (allocation of parameters) and sufficient factor broadcasting (parallel cluster computation) by structuring computing clusters to allow synchronization between various clusters. This means that the synchronization processes are independent of each other and thus have different computations and different communication methods.

The directed acyclic graph (DAG), optimization of distributed (parallel) computation in GPU where nodes represent the computational tasks, and the edges represent the communication pipeline between the nodes is done by not allowing the start of any new iteration before the complete execution and final update of an already started iteration [216]. DAG uses synchronous

University of Ghana http://ugspace.ug.edu.gh

(parallelized) input data reading and backpropagation tasks. In that respect, equation 4.11.1 is transformed into equation 2.3.40, which maximizes the feedforward and backpropagation process.

$$t_{iteration} = \max(t_{io} + t_{h2d} + t_f + t_b + t_g) \tag{2.3.40}$$

Here t_g is the time taken to complete the gradient aggregation in an iteration.

The input data parallelization under DAG is done such that if any of the data input, execution, and update completes in one segment before the other segments, the completed segment does not wait for the others to complete their execution and update but loads a new input and goes through a feed-forward, backpropagation, and update. This allows overlapping of input and output computation by a particular section of the segment, thus reducing the iterative computation time for that segment. This means that if computations in several segments overlap as described above, although some individual iterations might be slower than others, there will be a reduction in the total iteration time. The second optimization option under DAG is the overlapping of gradient communication with computation. This is referred to as the wait-free backpropagation process [215]. Under this parallel process, a communication task (for example, T_1) is parallelized with a computation task (for example, $T_9 - T_{15}$) so that time is not wasted for either computation task or communication task to complete before another computation or communication process starts. This is so because the communication task of layer t – 1, for example, depends on the gradient task of layer t – 1.

Chu-Hsing Lin et al. [217] used the fast Fourier transform (FFT) to optimize the GPU parallel process with an emphasis on optimizing memory allocation. This was done by dividing the input data into even, and odd parts; the N-point DFT is divided into two or more DFTs with each size smaller than N. Since the fast Fourier transform can be considered a one-dimensional Discrete

University of Ghana http://ugspace.ug.edu.gh

Fourier transform (DFT), The various divisions of the DFT are joined together to obtain the FFT. That is using N as the size of data, if a sequence S[t], t = 0, 1, . . ., N-1 is divided into two sequences each of length N/2 to perform a radix-2 fast Fourier transform, the complexity of computing in the DFT is reduced from $\boldsymbol{O}(N^2)$ to $\boldsymbol{O}(N\log N)$ thus allowing fast processing of data in the GPU's

Avadhesh P. Singh et al. [218], using CUDA, modified Dijkastra's algorithm, which is usually used to find the shortest path between nodes in a graph into the implementation of finding shortest paths between parallel graph execution that is the K – shortest path algorithm in GPU. While disallowing the repetition of vertices, the K- shortest path algorithm finds the shortest path between two pairs of vertices of a graph consisting of N nodes and V vertices. The algorithm determines the K-shortest path without a loop. That is if G = (V, E) is a directed graph, with V vertices (number of nodes) and E edges (number of paths) where V > E. If each edge is assigned a positive weight of w, then the algorithm finds the K-shortest paths as the weight increases. The time complexity of the K-shortest path without repetition is O (m + n*logn + k), and each additional path will attract a time complexity of O(n). Where k is the number of shortest paths, m is the number of nodes (vertices), and n is the number of paths (edges). The linear time complexity of non-repeating vertices is better than that of repeating vertices $O\ (kn\ (m+nlogn))$.

You Zhou et al. [219] used Standard particle swarm optimization (SPSO) on the compute unified device architecture (CUDA) to improve the processing time (performance). SPSO takes its root from particle swarm optimization (PSO)[220], which mimics the behavior of birds flocking. The swam optimization process employs stochastic global optimization techniques to improve the processing time of large computation processes. It requires a lot of computing resources to handle problems with large dimensions, which will necessitate a large swam population. This makes the algorithm unsuitable to be implemented on CPUs. However, GPUs provide enough resources to

University of Ghana http://ugspace.ug.edu.gh

allow the ease of implementing PSO on it. SPSO set aside the use of global best variable in PSO which is used in computational processes. This meant that much time is wasted in searching for global variables when large datasets are involved. major factor in delaying the PSO algorithm in various computations. In SPSO, every particle is treated as local best so the time spent in searching for global best variable is eliminated. With the implementation of the algorithm with large datasets via CUDA, there is a significant reduction in implementation

University of Ghana http://ugspace.ug.edu.gh

Chapter 3. Methodology

Theoretical Analysis of Genome Sequencing Error Correction Using DCNN

Genome sequencing is the basic process in modern biology and genomics, it provides valuable insights into the genetic makeup of organisms. However, the data generated by genome sequencing technologies usually contains errors such as calling inaccuracies and noise which can be attributed to various factors. These errors can affect downstream analysis and affect the accurate interpretation of the underlying dataset. To overcome these challenges, the exploration of the potential use and benefits of applying DCNN for genome sequencing error correction is undertaken.

DCNNs are artificial neural networks that have demonstrated immense success at learning and extracting features from complex datasets such as audio, images, and sequences. DCNNs are made up of multiple convolutional layers, activation functions, pooling layers, flattened layers, and fully connected layers. The convolutional layers consist of filters that are used to capture local patterns or features in the input data which is used to generate a feature map. The activation function converts the feature maps into a non-linear dataset which enables the DCNN to learn better. The pooling layer down samples the feature map by reducing the special dimension and at the same time maintaining the important learned features. The fully connected layers combined the learned feature to enhance the prediction or classification.

Employing DCNN in genome sequencing error correction involves training the network with a large dataset consisting of a reference genome and its corresponding erroneous counterparts. The DCNN learns the patterns or features that distinguish between correct and incorrect genome sequences. During the training process, the network automatically adjusts its internal parameters

University of Ghana http://ugspace.ug.edu.gh

such as assigned weights and biases, to minimize the difference between the predicted corrections and the actual value.

The genome sequencing error correction, identified as a classification (pattern recognition) problem, falls in the basic tasks performed by DCNN. This is because it has demonstrated good performance in many pattern recognition tasks by learning features from large and noisy datasets. They can also learn dependencies between neighbouring bases and use this gained insight to correct errors in the dataset which will undoubtedly improve the overall quality of the generated sequence.

DCNN can handle both small and large-scale datasets efficiently. When it is trained, it can process new sequences easily. This makes them suitable for high-throughput and real-time sequencing applications.

DCNNs therefore provide an optimistic approach for correcting genome sequencing errors by leveraging their ability to learn both complex patterns and dependencies. Their high accuracy and scalability make them important tools for improving the quality and reliability of genome sequencing data. Further research is necessary to optimize DCNNs to enhance the interpretability of some outputs.

Data Pre-processing and K-mer Generation

The data (NA12878), was downloaded from the National Centre for Biotechnology Information (NCBI) website. The genomic data was preprocessed by removing all unknown nucleotide residue (N) from the data. This was crucial to enhance data quality, improve alignment accuracy, and ensure standardization and compatibility. Next, the Guanine Cytosine (GC) content analysis was conducted. This produced a read coverage value of 58.96% which indicates that the data is good

University of Ghana http://ugspace.ug.edu.gh

and can be used for the analysis. GC content values of less than 50% or more than 60% are an indication that the dataset is not viable for analysis purposes. The data contains 134,061,687 nucleotide bases, with 134,061,548 k-mers generated (Table 3.1). Multiple sequence alignment of the k-mers was then performed, based on the read information content with the reference genome.

Parameter	Value
Nucleotype	134,061,687bp
K-mer length	2x140
Number of K-mers	134,061,548bp
Error rate	12%

Table 3.1: Variables generated from the Dataset (NA12878)

University of Ghana http://ugspace.ug.edu.gh

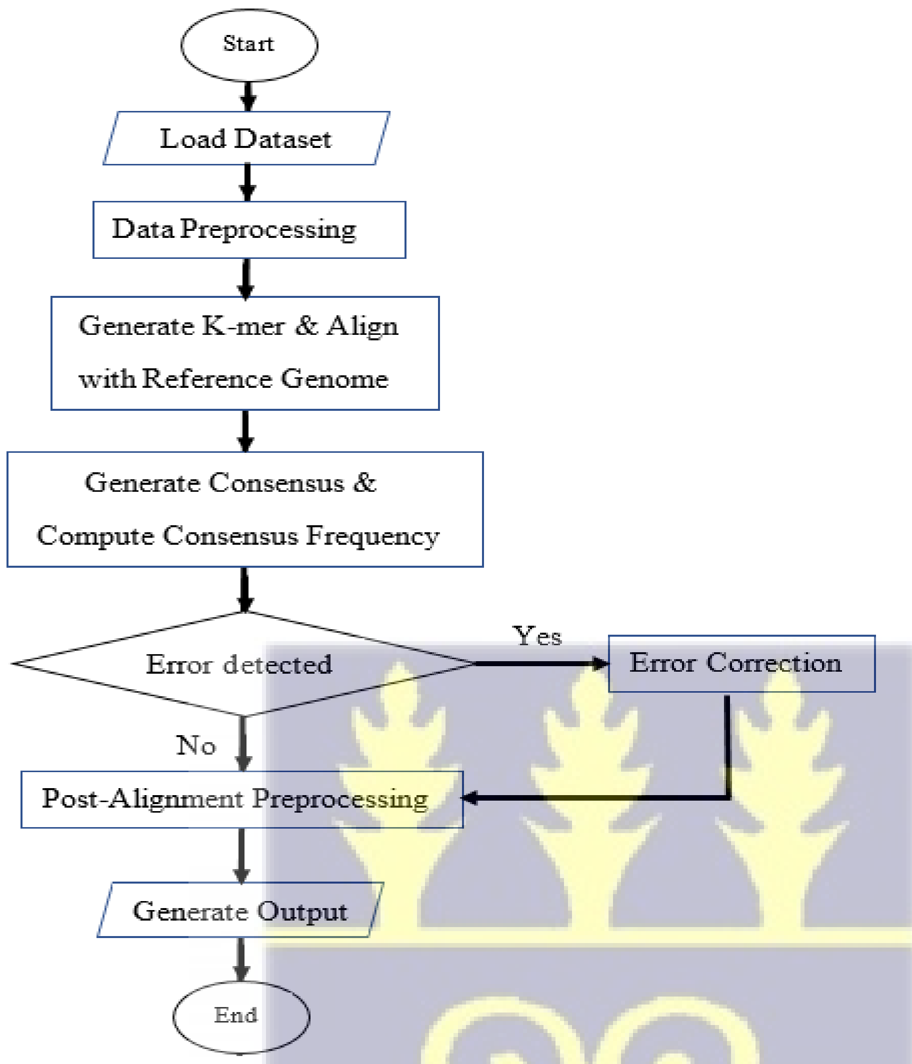


Figure 3.1: Flowchart for the genome sequencing error correction

Under this thesis, the error correction problem has been treated as a classification problem. This was done by first creating labeled data through multiple sequence alignment by piling up k-mer reads with the reference genome. The k-mers, were not converted into image data (Red - R, Bleu -B, Green - G,) as has been done by other researchers [104][122][18]. When converted to an image, it increases in size which slows the network when learning and validating the dataset. Rather, the k-mers were treated as words, and then the classification was natural language processing (NLP) instead of image classification. In the architecture, multiple neurons are heaped together to form a

University of Ghana http://ugspace.ug.edu.gh

layer, and multiple layers are also heaped together to form the architecture of the deep neural network.

The one-hot encoding scheme (a binary vector) was used to change the categorical data (nucleotide) into a numeric dataset to allow the DCNN to work on them.

3.1 System Design

The design phase was initiated by making key architectural considerations that will impact the network's performance and efficiency. The network depth, which is the number of layers was critically chosen as four because each added layer from two to four helps the network learn more complex features. After four, any additional layer added contributes negligibly towards the performance and learning capability of the network while increasing the vanishing gradient effect on the network.

The convolutional layers performed the local field operations by applying sixteen filters to extract features from the input dataset and form a feature map. They contain a kernel size of five, and a kernel regularization of size twelve to help detect complex patterns in the dataset. Because of the size of the dataset, to prevent the dimensionality of the network from growing out of proportion, the pooling (max pooling) layer is introduced to reduce the dimensionality of the feature map which will translate into reducing the computational requirements needed to execute the DCNN.

To increase the learnability of the DCNN, The ReLU (Rectified Linear Unit) activation function was applied to introduce non-linearity to the network. The ReLU activation function is defined as max(0, x). This makes it simple and computationally efficient. That is when the input of ReLU is positive ($x>0$), the output is equal to the input. This makes it preserve only positive gradients, thus ensuring that during backpropagation, there is the free flow of gradients through the ReLU without

University of Ghana http://ugspace.ug.edu.gh

being attenuated or diminished. Thus ReLU does not suffer from the saturation problem that causes gradients to become small or vanish. The non-linearity introduced by ReLU also helps the network to model more complex relationships between the input features and the target outputs.

The model was pre-trained with the Homo Sapiens isolate NA12878 chromosome 21 (in the FASTA file format), a smaller version of the data under investigation. This provided the opportunity to apply transfer learning from the pre-trained dataset to the dataset under investigation.

Activation visualization was employed to identify the regions or positions in the dataset that contribute most to the decisions made by the network. This was done by visualizing the activation maps of the filters in convolutional layers. The saliency map which is a computation of the gradients of the output concerning input, provided an insight of the region with high-gradient values which is an indication of where most of the errors can be found and thus focus most in that region during the error correction process.

3.2 System Implementation

Next, a deep convolutional neural network was designed, which is a deep neural network utilizing convolutional operations as defined mathematically in Equation 3.1 below. The network initially consists of a convolutional layer that contains sixteen filters which is used to learn the local feature of the input dataset and forms a feature map. The network automatically adds weights and biases to the input dataset. This is then passed through the ReLU activation function to introduce nonlinearity into the feature map which helps the DCNN learn better. This is pooled and flattened to reduce the dimensionality of the feature map. The new feature map is then transferred into the next layer without any unwarranted attachments as fully connected. This process is repeated

University of Ghana http://ugspace.ug.edu.gh

throughout all the convolutions. The deep convolutional neural network architecture also uses a sliding window emanating from learned filters to detect patterns at various locations [142] automatically.

$$\begin{cases} Y_i = X & i = 1 \\ Y_i = f_i\ (Z_i) & i > 1 \\ Z_i = g_i\ (Y_{i-1;} W_i) \end{cases} \tag{3.1}$$

Here i constitutes a layer number, X is an input tensor of the whole deep convolutional neural network, Y_i is the output tensor for the layer i, $f_i(.)$ and constitutes the activation function that is used in the layer i. Z_i Represents the weighted output of the deep neural network operation. $g_i(.)$ represents the output of the weighted operation between the current (W_i) and the previous layer (Y_{i-1}). Equation 3.2 below shows the three weighted operations (convolution, pooling, and full connectedness) that are performed under the architecture.

$$\begin{cases} Z_i = W_i * Y_{i-1} & if\ i^{th}\ layer\ is\ convolutional \\ Z_i = Pool(Y_{i-1}) & if\ i^{th}\ layer\ is\ pooling \\ Z_{i.} = W_i . Y_{i-1.} & if\ i^{th}\ layer\ is\ fully-connected \end{cases} \tag{3.2}$$

Here * represents a convolutional operation, $Pool$ represents either a maximum or average pooling, and (.) is used for matrix multiplication in a fully connected layer [221]. The block diagram of the network (Figure 3.2) shows the convolutional and dense layer activities that go on in the deep convolutional neural network.

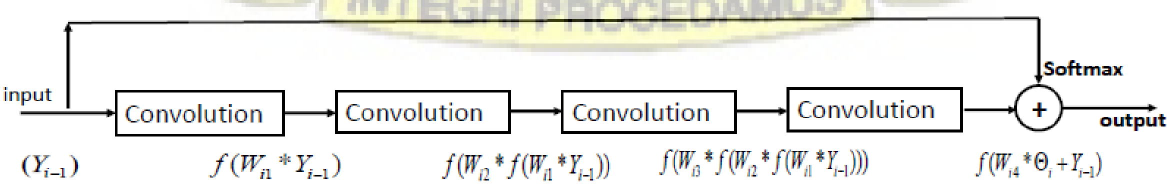


Figure 3.2 Block diagram of the deep convolutional neural network

University of Ghana http://ugspace.ug.edu.gh

Here Y_i is the ith input dataset, W_{ij} is the weight assigned to the dataset i in layer j, and

$\Theta_i = f(W_{i3} * f(W_{i2} * f(W_{i1} * Y_{i-1})))$. Figure 3.3 below is the workflow of the architecture.

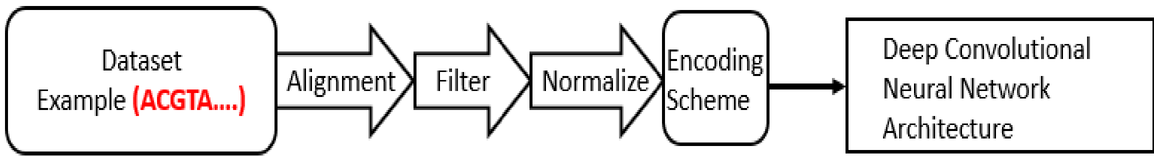


Figure 3.3: Schematic workflow of the framework

The model (Figure 3.4) consists of four hidden layers; each hidden layer consists of a convolutional network, the ReLU activation function which introduces nonlinearity into the model to help the model learn better, and the maxpool layer, which reduces the input volume for the next layer. We used a flattened layer to convert the maxpool featured map into a column vector to serve as input data for the fully connected layer. Next, a dropout layer was introduced to fine-tune the network, preventing it from either underfitting or overfitting. The dropout layer's output is then passed through another fully connected layer before passing it through the softmax probability function to predict the output data.

The data NA12878 (appendix 1), taken from the National Center for Biotechnology Information (NBCI), was initially divided into training, validation, and testing, respectively, using the 60%, 20%, and 20% ratios. A one-hot encoding scheme, where the nucleotide bases A, C, T, and G were represented as follows encoded as [0 1 0 0], [1 0 0 0], [0 0 1 0], and [0 0 0 1] was utilized. This is because the nucleotide bases A, C, T, and G form categorical data, which has to be transformed into numeric values for the machine learning (ML) algorithm to work on it efficiently. The one-hot encoding scheme takes the attributes of a nucleotide and encodes it as a binary array of

University of Ghana http://ugspace.ug.edu.gh

dimension four (4). The DCNN was trained with the training dataset, and the validation dataset was introduced after the training. A divergence between the training and validation processes was observed after epoch 8, and this divergence persisted even after epoch 100 (Figure 3.9).

We corrected the divergence of the validation from the training data set by increasing the size of the training dataset from 60% to 80% and reducing the size of the validation and testing dataset to 10% each. The learning rate was reduced from 0.008 to 0.004 to decelerate the training process and facilitate network recovery. Additionally, batch normalization was introduced to normalize the activation of the mini-batch during training, promoting stability, enhancing the model's capacity to learn meaningful representations, and improving the generalization of the DCNN while reducing the risk of overfitting. Subsequently, the network was retrained and validated using these parameters, resulting in the correction of overfitting (Figure 3.10).

To test the effect of other activation functions on the network, The sigmoid and hyperbolic tangent (tanh) functions were utilized at the output layer of the network to assess their impact on network training, validation, and performance in comparison to the softmax function. The network with the sigmoid function at the output layer (Figure 4.2) was quite noisy during the validation process between epochs 5 to 40. The network with the tanh function at the output layer (Figure 4.3) produced a high spike of about 1.3 loss during the validation process. The validation process for both networks improved considerably after epoch 20. However, the network with the softmax activation function at the output layer (Figure 3.10) produced a better validation process than that of the sigmoid and tanh functions at the output layer.

INTEGRI PROCEDAMUS

University of Ghana http://ugspace.ug.edu.gh

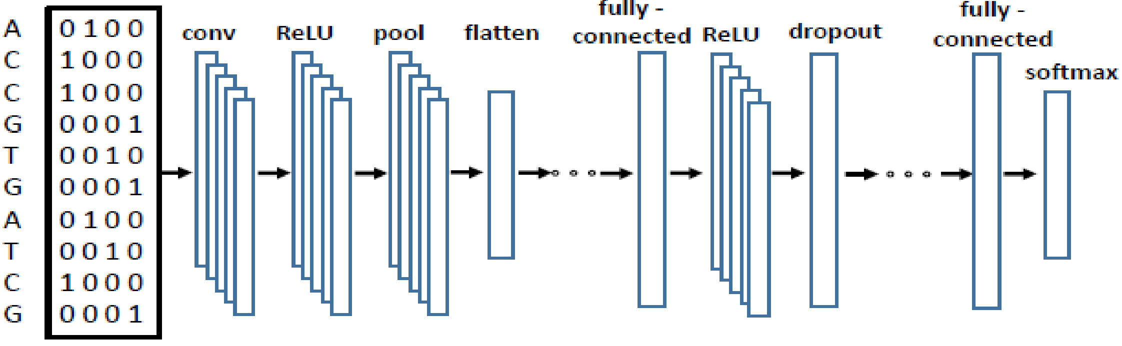


Figure 3.4. Convolutional Neural Network using the one-hot encoded scheme as input data for training the network and the softmax function to predict the output.

The network accuracy (performance) diagrams for the softmax (Figure 4.4), sigmoid (Figure 4.5), and tanh (Figure 4.6) show that the softmax function performed better than the sigmoid and tanh functions. The network with the softmax function at the output layer was very accurate in performance after epoch 10. Very high performance of the network with the sigmoid function at the output layer was achieved after epoch 30. The network with the tanh function at the output layer was the least accurate when compared with the softmax and sigmoid functions. From both the validation and performance values of the three functions, the softmax function outperformed the sigmoid and tanh functions.

The network architecture is shown in Figure 3.4. Instead of correcting errors in single reads, A consensus-based approach was employed to create a consensus from multiple reads and generate the underlying DNA. The algorithm used for the error correction is shown in Figure 3.5 below.

University of Ghana http://ugspace.ug.edu.gh

Input: An extracted read set $R = \{R^1, R^2, \ldots, R^m\}$, their corresponding alignment position is $\{u_i\}_{i=1}^{m}$ and threshold value λ

Output: A corrected read set S and an updated nucleotide sequence A

Function: Error_Correction (R, $\{u_i\}_{i=1}^{m}$, λ)

begin:

q ← max(u_i)

A ← initially empty //stores the updated nucleotide sequence

for $j = 1$ to $(q + n - 1)$ do

(c, d) ← (0, 0)

for $i = 1$ to m do

if $u_i \leq j \leq (u_i + n - 1)$ then

$f(j, r^i{}_{(j - u_i + 1)}) \leftarrow f(j, r^i{}_{(j - u_i + 1)}) + 1$

$c \leftarrow c+1$

$A(j) \leftarrow \underset{x \in \{A, G, C, T\}}{\text{argmax}} (f(j, x))$

foreach $x \in \{A, G, C, T\}$ do

if $f(j, x) > 0$ then

$d \leftarrow d+1$

foreach $x \in \{A, G, C, T\}$ do

if $f(j, x) \leftarrow f(j, x)/c + d * 0.1$ then

$d \leftarrow d+1$

$S \leftarrow \theta$ *//store the corrected reads for I = 1 to m*

for $j = 1$ to n do

$t(j) \leftarrow f(u_i + j - 1, r^i{}_j)$

[B, I] ← sort(t(1…n)) if B(3) > λ then

$r^i{}_{I(1)} \leftarrow A(I(1) + u_i - 1)$

$r^i{}_{I(2)} \leftarrow A(I(2) + u_i - 1)$ Append R^i to S

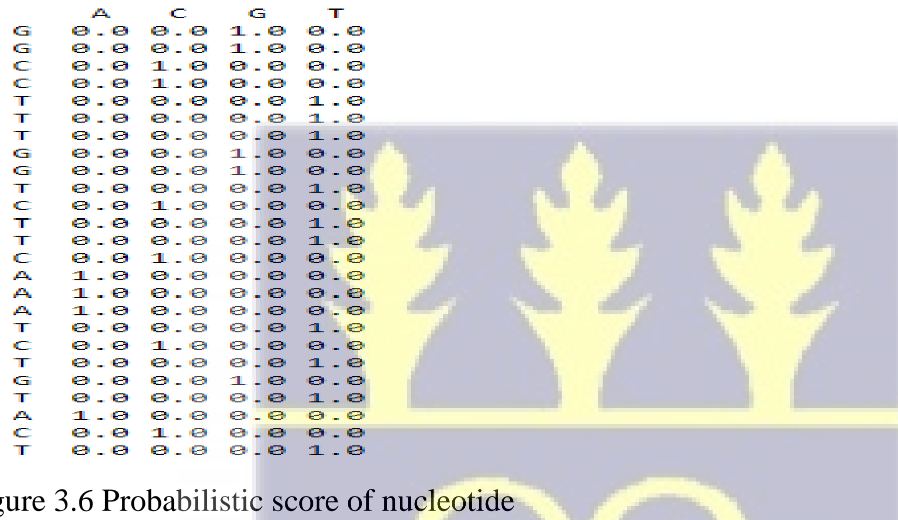
return S, A

University of Ghana http://ugspace.ug.edu.gh

Figure 3.5: Algorithm for error correction

The position-specific matrix, representing the probabilistic score of the nucleotide at different positions, was generated (Figure 3.5) by inserting the code below into the consensus-building algorithm

```
pssm = summary_align.pos_specific_score_matrix(second_seq, chars_to_ignore=["N"])
```



	A	C	G	T
G	0.0	0.0	1.0	0.0
G	0.0	0.0	1.0	0.0
C	0.0	1.0	0.0	0.0
C	0.0	1.0	0.0	0.0
T	0.0	0.0	0.0	1.0
T	0.0	0.0	0.0	1.0
T	0.0	0.0	0.0	1.0
G	0.0	0.0	1.0	0.0
G	0.0	0.0	1.0	0.0
T	0.0	0.0	0.0	1.0
C	0.0	1.0	0.0	0.0
T	0.0	0.0	0.0	1.0
T	0.0	0.0	0.0	1.0
C	0.0	1.0	0.0	0.0
A	1.0	0.0	0.0	0.0
A	1.0	0.0	0.0	0.0
A	1.0	0.0	0.0	0.0
T	0.0	0.0	0.0	1.0
C	0.0	1.0	0.0	0.0
T	0.0	0.0	0.0	1.0
G	0.0	0.0	1.0	0.0
T	0.0	0.0	0.0	1.0
A	1.0	0.0	0.0	0.0
C	0.0	1.0	0.0	0.0
T	0.0	0.0	0.0	1.0

Figure 3.6 Probabilistic score of nucleotide

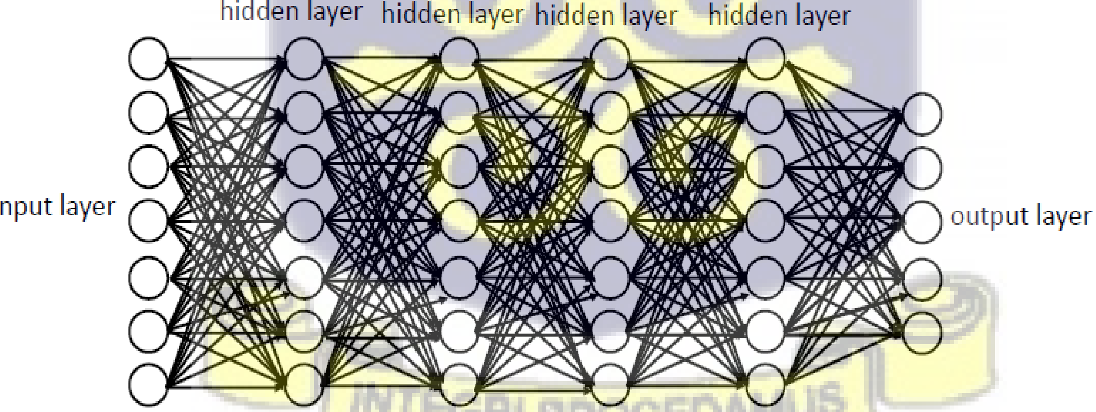


Figure 3.7. Fully connected convolutional neural network with four hidden layers. Source A. Ghaffari [222]

University of Ghana http://ugspace.ug.edu.gh

```
[[0. 0. 1. 0. 0.]
 [0. 0. 1. 0. 0.]
 [0. 0. 0. 1. 0.]
 ...
 [0. 0. 0. 1. 0.]
 [0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0.]]
```

Figure 3.8: Encoded dataset

The network was subsequently trained, and validated; please refer to Figure 3.9 below. The validation process between epochs 0 and 5 seemed good but took a divergent tangent after epoch five and did not recover even after epoch 50.

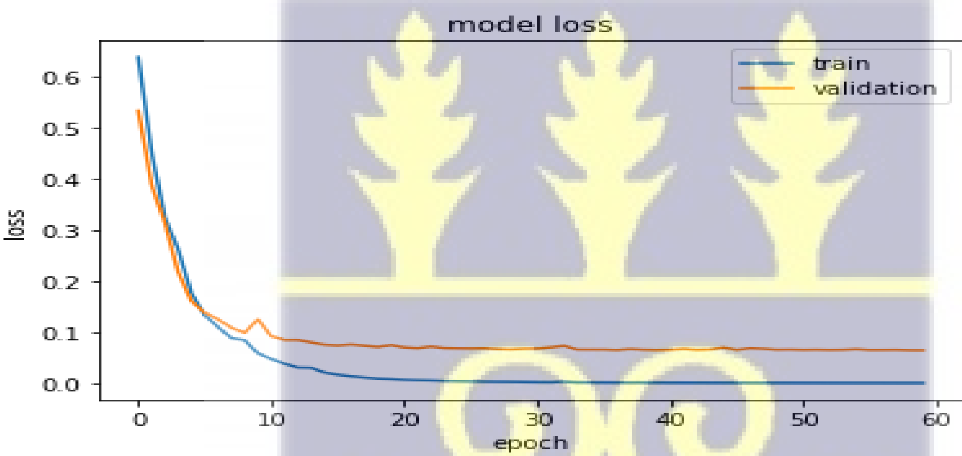


Figure 3.9. Initial training and validation of the network show divergence in the validation after epoch 5.

University of Ghana http://ugspace.ug.edu.gh

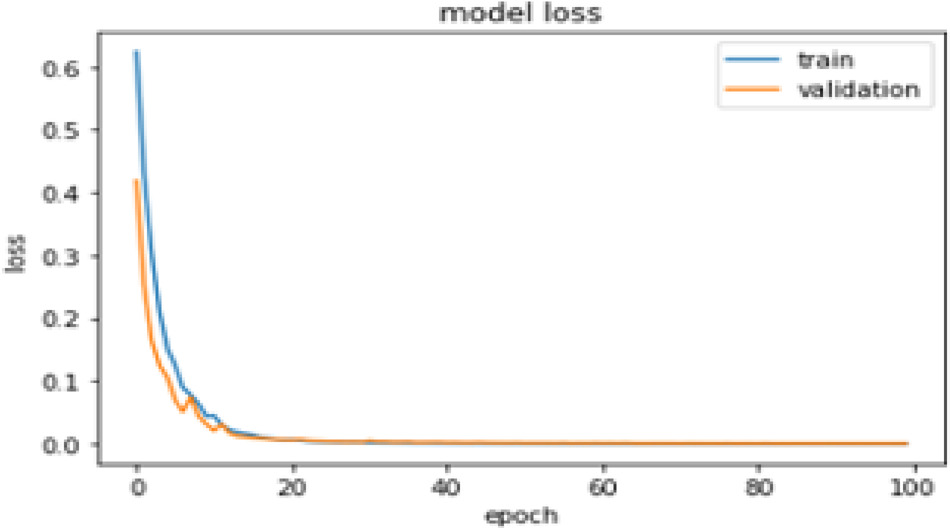
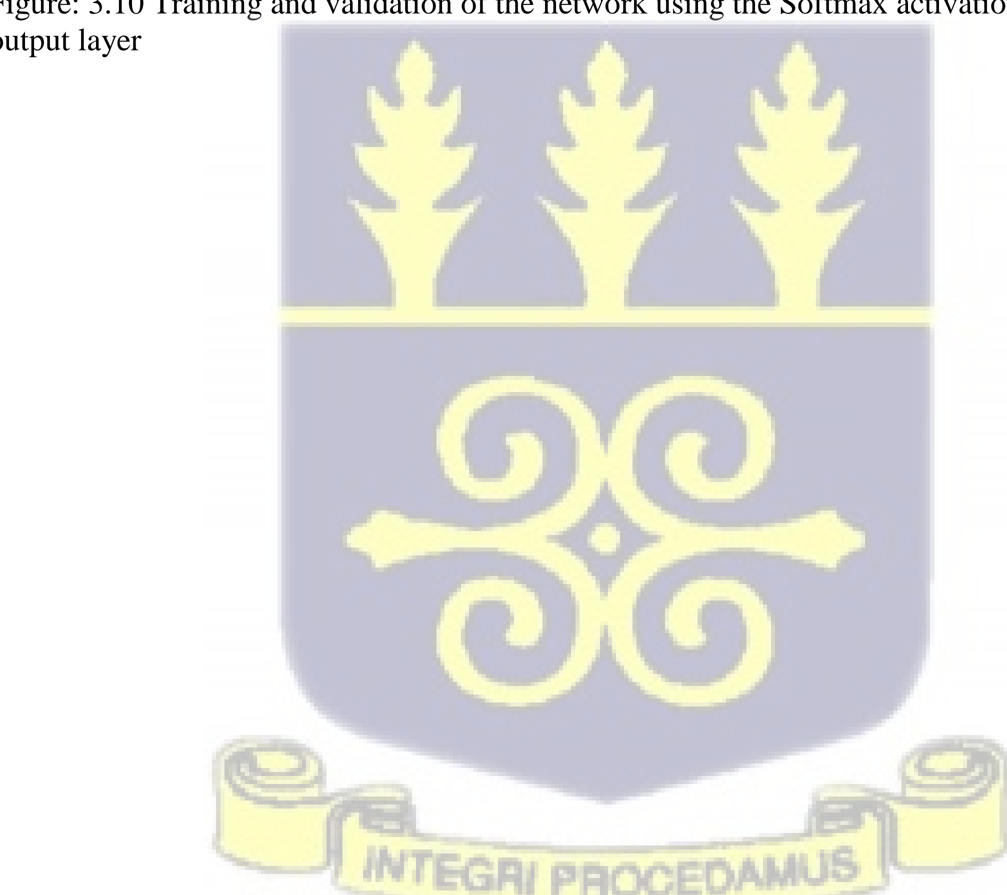


Figure: 3.10 Training and validation of the network using the Softmax activation function at the output layer



University of Ghana http://ugspace.ug.edu.gh

Chapter 4: Results and Discussion

4.1 Results

The network was constructed using the Python programming language within a Jupyter Notebook environment. The entire experiment was conducted on a Hewlett-Packard (HP) pavilion i5-4210U laptop with 12GB RAM @ 2.40GHz and a 1 TB Hard Disk using Windows 10 OS. The data was obtained from the National Center for Biotechnology Information (NCBI) using the link https://www.ncbi.nlm.nih.gov. Table 4.1 below shows the number of k-mers, the k-mer length, and the error rate contained in the data. The precision, accuracy, misclassification rate, sensitivity, and F1 score are all computed using the True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN) values.

Parameter	**Value**
No of k-mers	134,061,548bp
K-mer length	2x140bp (av)
Error rate before correction	12%
Error rate after correction	0.94%
TP	99%
TN	1%
FP	1
FN	0%
Precision	99%
Accuracy	99.59 %
Misclassification Rate	0.5%
Sensitivity	1
F1	99.50%

Table 4.1: Results generated by the DCNN model.

University of Ghana http://ugspace.ug.edu.gh

The network's performance (accuracy) was verified using a series of classifications, using the encoded dataset as shown in Figure 3.8. The network's performance was evaluated in terms of accuracy and efficiency, and compared with other machine-learning models that operated on the same dataset (Table 4.2).

Error Correction Methodology	RAM (GB)	Processor (No GPU/TPU)	Platform (OS)	F1 Score	Performance (Accuracy %)
Deep Variant	24	2x14-core (2.6GH)	MacBook	99.75	98.98
Our Method	12	Core i5 (2.4 GHz)	Windows	99.50	99.95
Nanopore Error Correction	18	Core i7 (2.2 GHz)	MacBook	86.94	97.7

Table 4.2: Comparison of the performance of the deep variant and Nanopore error correction method with our method

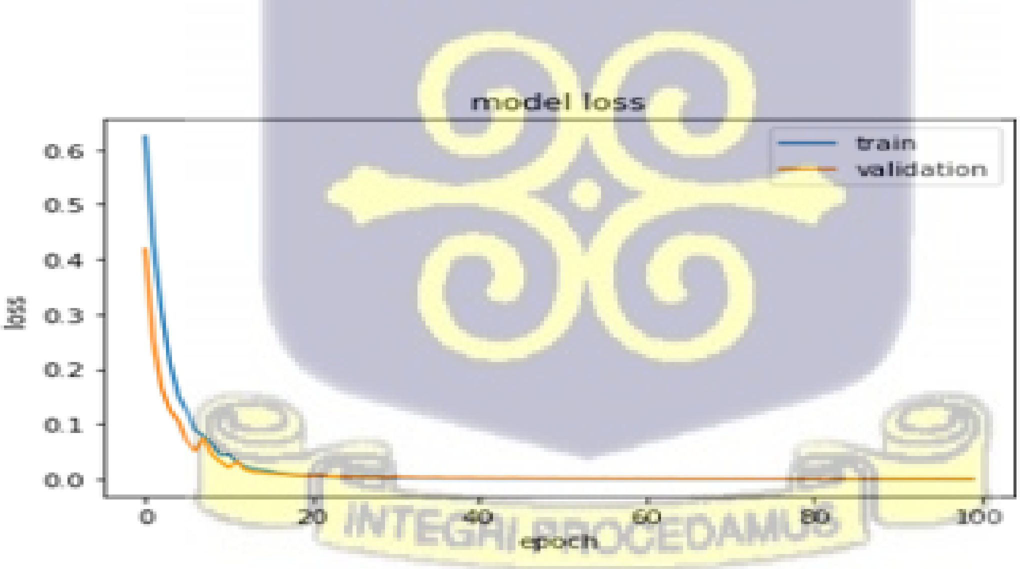


Figure 4.1: Training and validation of the network using the Softmax activation function at the output layer

University of Ghana http://ugspace.ug.edu.gh

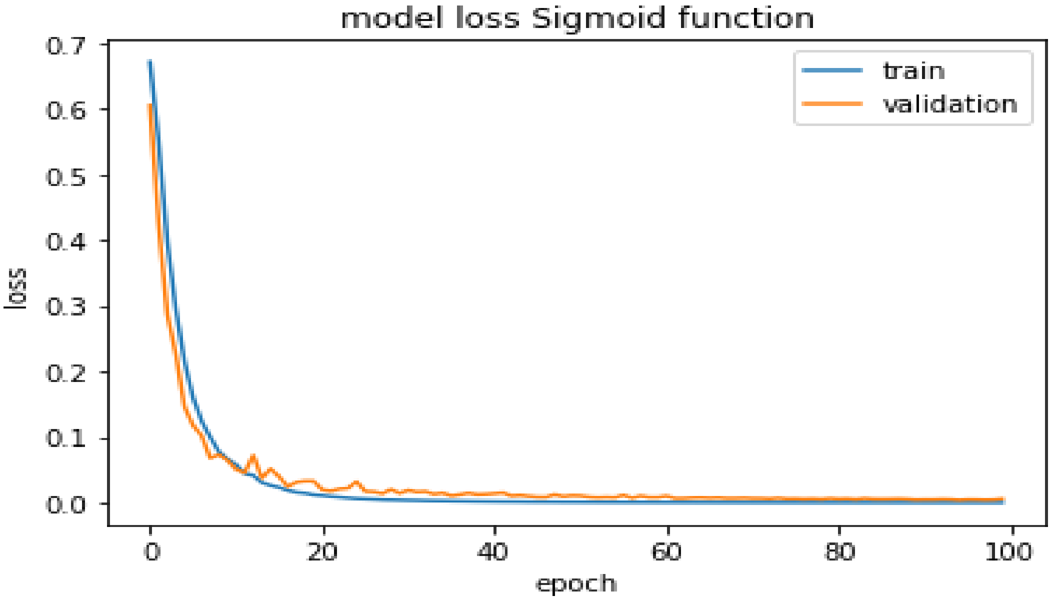


Figure 4.2: Training and validation of the network using the Sigmoid activation function at the output layer

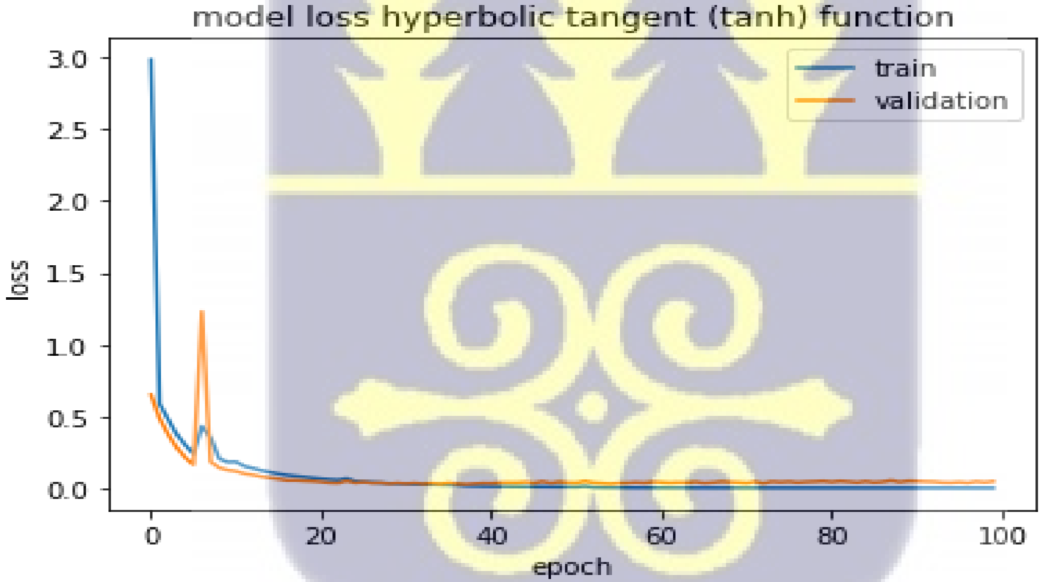


Figure 4.3: Training and validation of the network using the Tangent (tanh) activation function at the output layer

INTEGRI PROCEDAMUS

University of Ghana http://ugspace.ug.edu.gh

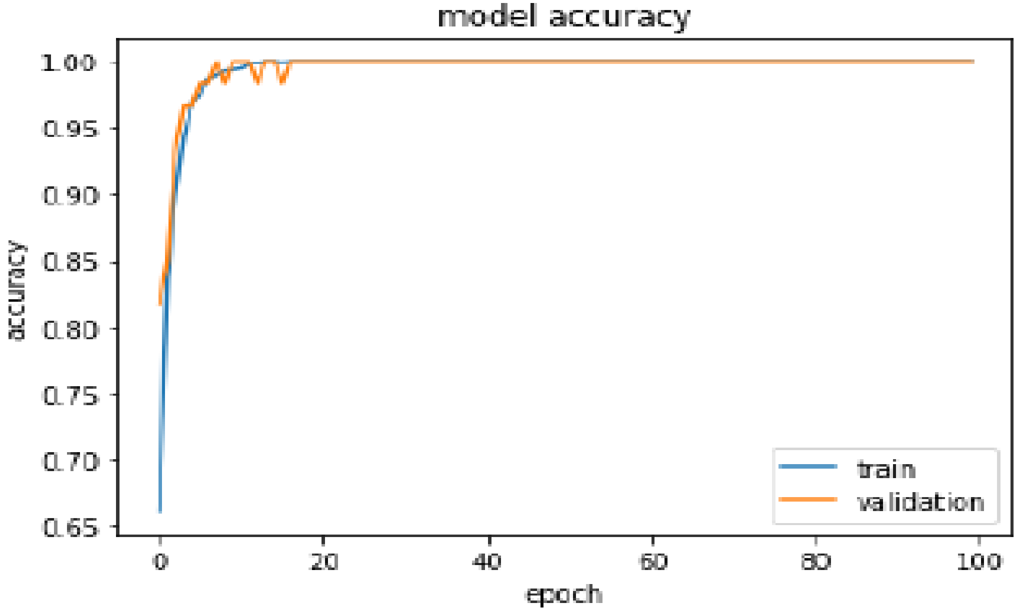


Figure 4.4: Network performance (accuracy) diagram using the Softmax Activation function

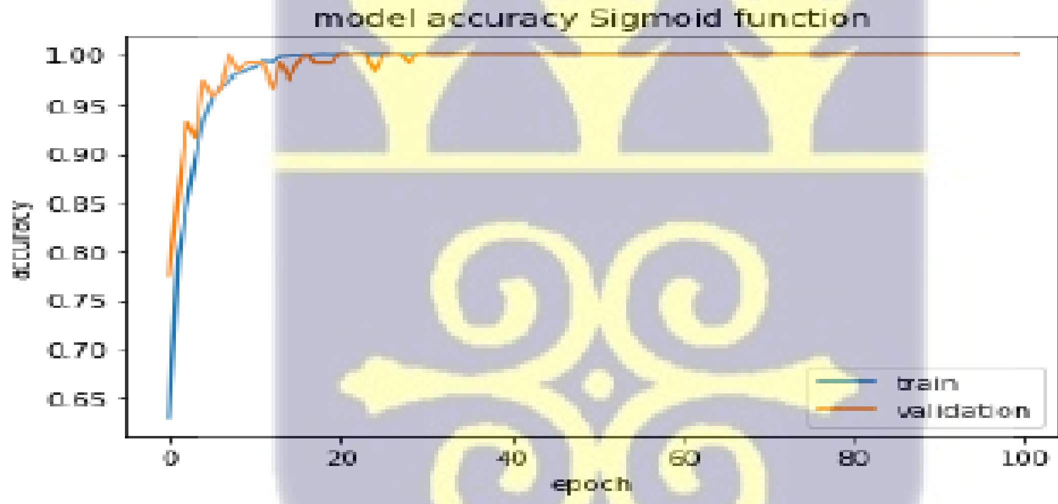


Figure 4.5: Network performance (accuracy) diagram using the Sigmoid activation function

University of Ghana http://ugspace.ug.edu.gh

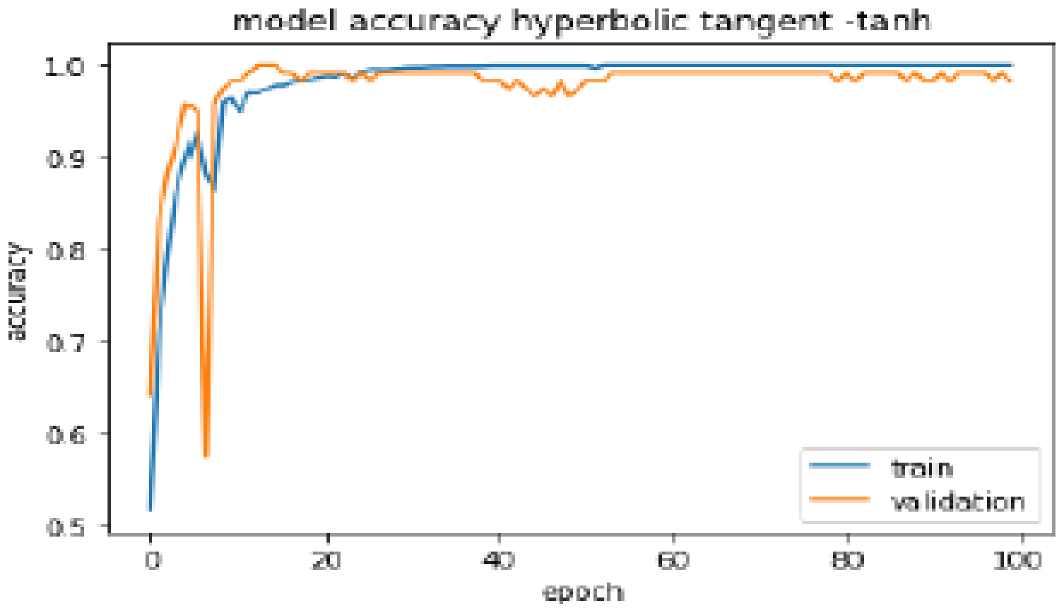


Figure 4.6: Network performance (accuracy) diagram using the Tanh activation function

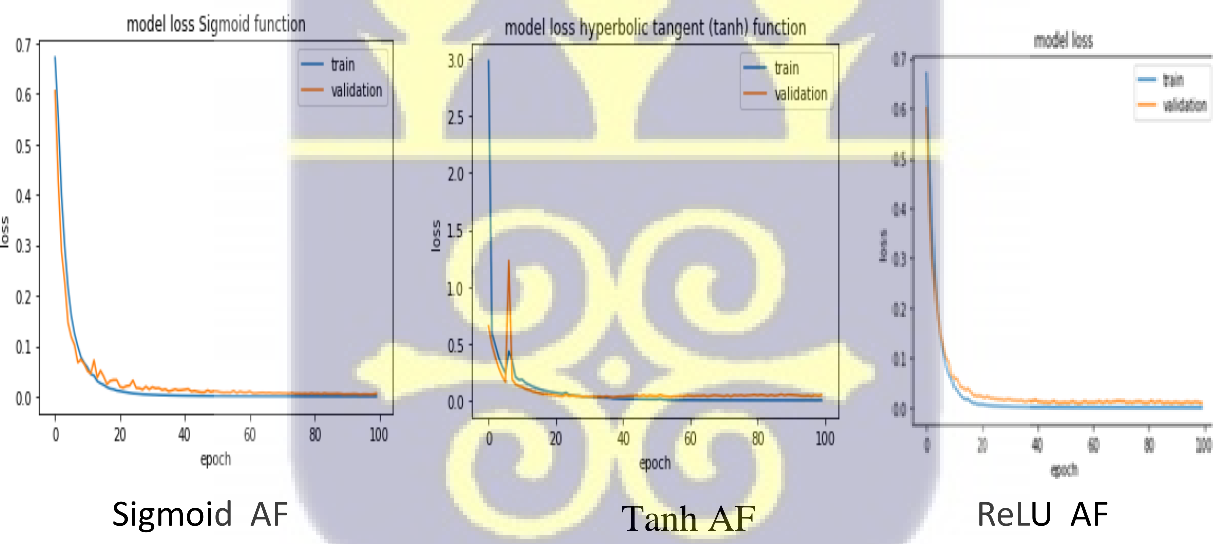


Figure 4.7: Classification comparison of various activation functions (Sigmoid, Tanh, ReLU)

University of Ghana http://ugspace.ug.edu.gh

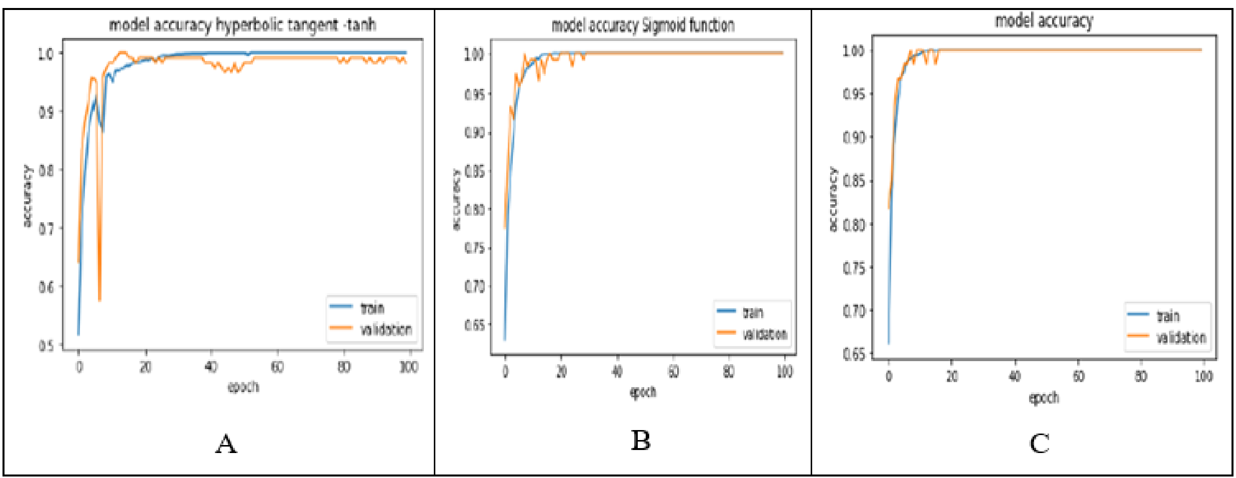


Figure 4.8: Performance comparison of various activation functions (Sigmoid, Tanh, ReLU)

4.2 Result Analysis

To assess the effectiveness of our DCNN for genome sequencing error correction, Several standard performance metrics were utilized, including:

a. Accuracy: The percentage of correctly corrected bases in the data set.

b. Precision: The ratio of the true positive to the sum of the true positive and false negative.

c. Misclassification (error) rate: The ratio of the sum of false positive and false negative to the sum of true positive, true negative, false positive, and false negative.

d. F1 score: An evaluation metric that provides a balanced measure of the model's performance using the precision and sensitivity values

The primary metric used to evaluate the performance of the DCNN model is accuracy, this measures the proportion of the correctly corrected errors. The DCNN achieved an impressive accuracy of 99.59% on the test dataset. This indicates the model's ability to accurately identify and correct sequencing errors (indels) in genome sequences which enhances its reliability in downstream analysis.

University of Ghana http://ugspace.ug.edu.gh

The high precision value of 99% indicates that the DCNN model is accurate in its error correction predictions while minimizing the introduction of false error corrections. This indicates the model's ability to make precise and reliable corrections, thus enhancing the quality and accuracy of the genome sequence. On the other hand, a low precision value will suggest a higher proportion of false positives which will be an indication that the model may be introducing incorrect corrections or misclassifying non-error regions as errors.

The misclassification rate is the ratio of the classification errors to the total errors in the genome sequence. This provides a quantitative measure of the model's accuracy in identifying and correcting errors. The low misclassification rate of 0.5% indicates that the DCNN model has a high accuracy in identifying and correcting errors in the genome sequence. It also indicates that the model is effectively minimizing both false positives and false negatives, making the DCNN a reliable error correction model. On the other hand, a high misclassification rate will indicate a higher proportion of errors that are incorrectly classified or corrected by the model which will compromise the accuracy and quality of the genome data.

The F1 score provides a balanced measure of the model's performance by taking both the precision and sensitivity values which are key indicators in the computation of the accuracy into account. This shows a comprehensive evaluation of the model's accuracy and error correction process at the same time. A low F1 score will be an indication of an imbalance between precision and sensitivity. This means that the model may be biased towards either minimizing false positives or minimizing false negatives at the expense of the other variables. A high F1 score indicates a good balance between precision and sensitivity, thus suggesting that the DCNN model has achieved both accurate and comprehensive error corrections. Optimizing the F1 score will enable a balance

INTEGRI PROCEDAMUS

University of Ghana http://ugspace.ug.edu.gh

to the error correction method by ensuring the accurate and comprehensive identification and correction of errors in the genome sequence.

Figure 4.1 shows the model loss for the training and validation of the network after tweaking the network using the softmax activation. The effect of the sigmoid (Figure 4.2) and tanh (Figure 4.3) activation functions on the DCNN was examined after the model was adjusted. Subsequently, their impact on performance was assessed, Figure 4.3 and Figure 4.6 respectively show the performances of the sigmoid and tanh activation functions

Figure 4.7 is a comparison between the three activation functions, it shows that the Softmas activation function outperformed the sigmoid and Tanh activation functions when validating the dataset. Figure 4.8 compares the performance of the three activation functions and again the Sigmoid activation function outperformed both the Sigmoid and Tanh activation functions. This makes the softmax activation the best for the task

4.3: Discussion

The traditional error correction methods rely on heuristics, data structure, or statistical models such as the k-spectrum, De Bruijn graph, FM indexing, Suffix Array/Tree, or BWT to correct genome sequencing errors. This means that they require significant computational resources such as memory and processing power when analyzing large genomic datasets. Because of the way they are structured, they can be used to correct limited error types and may require substantial modification when it is to be used to correct different error types. They may also struggle to correct complex error types. They also have scalability issues because as the size of the dataset increases, the demand for computational resources also increases. This makes them less scalable for large or

University of Ghana http://ugspace.ug.edu.gh

new datasets. The k-spectrum method is sensitive to parameters such as k-mer value sizes. This means that a lot of tuning may be required when it comes to the selection of optimal parameters which may involve k-mer sizes. Another challenge suffered by the traditional methods is the issue of handling repetitive regions. The traditional error correction methods produce ambiguous results when used to sequence repetitive regions thus, making the correction of errors in such regions challenging.

Depending on the size and complexity of the dataset, the traditional methods may also have long runtime which may slow further analysis downstream if a pipeline architecture is being followed. In cases where sequencing errors are not randomly but systematically distributed, traditional error correction methods may provide an inaccurate error correction process.

Errors created by structural variations in the genomic data may also be missed by the traditional error correction process. Since genome sequencing is a dynamic process, the traditional error correction methods may become out of date and cannot handle new types of errors as they emerge. It can also be complex if not impossible to integrate the traditional error correction method into a comprehensive genome analysis pipeline.

Though traditional genome sequencing error correction methods have been valuable in genomic research, they are facing challenges as sequencing technologies continue to advance. New error correction techniques such as DCNN are needed to address these challenges and offer improved scalability, performance, and ease of using it for the correction of genome sequencing errors in large and heterogeneous datasets.

INTEGRI PROCEDAMUS

The DCNN which is made up of four convolutional neural networks with each convolution increasing the learning capabilities of the entire network has a better performance and total learning

University of Ghana http://ugspace.ug.edu.gh

capability than the traditional error correction processes. Once trained, it does not require any fine-tuning for the new dataset. This makes it scalable and grants it the ability to be used in noisy or low-occlusion datasets.

Treating the error correction process as a natural language process without converting the k-mers into images as was done by R. Luo et al [122], R Oplin et al [18] L. Wang et al [104] provided a lighter dataset for our DCNN. This increased the performance of the network and it outperformed these similar architectures thus making it more efficient than these similar machine-learning processes using the same dataset

This indicates that the experimental results demonstrated the superior performance (accuracy) of our DCNN-based approach compared to conventional error correction methods, all achieved with minimal computational resources. Our DCNN used processor core i5 with a speed of 2.4 GHz and 12GB memory capacity, Nanopore Error Correction used core i7 with a speed of 2.2GHz and 18GB memory capacity while Deep Variant used a 2x24 processor core with a processor speed of 2.6GHz and 24GB memory capacity. Looking at the three methods, our method has the least processor core (i5) while Deep Variant and Nanopore Error Correction processes used 2x24 core and core i7 respectively. Apart from the processor core, our network also has the least memory 12GB while Deep Variant and Nanopoer Error Correction used 24GB and 18GB respectively

Our DCNN demonstrated remarkable robustness to noise and variability which is usually encountered in genome sequencing data. Our model was able to adapt effectively to various error types as well as noisy levels, demonstrating its ability to handle different error profiles that emanate from different genome sequencing technologies. It also demonstrated the ability to learn from large-scale genomic datasets such as the dataset under investigation, this provided the opportunity to generalize well to unseen genome sequence datasets and also exhibit resilience to noisy data.

University of Ghana http://ugspace.ug.edu.gh

Using DCNN for genome sequencing error correction has exhibited good efficiency and scalability. The reason is that once the model is trained, it will correct errors in genomic sequences quickly, thereby making it suitable for large-scale applications. This efficiency can be attributed to the parallel processing capabilities of the DCNNs and the ability to use graphical processing units (GPUs) for accelerated computation. Additionally, the scalability of DCNNs allows them to be easily integrated into other frameworks and pipelines to be used for object detection.

This demonstrates that the network predicts the underlying nucleotide with very high accuracy.

To evaluate our network, the network performance was compared with that of "Deep Variant Calling" (Table 4.2) [91] and [13], and our system performed better by demonstrating a higher accuracy. Equations 4.1, 4.2, and 4.3 were respectively used to compute the classification accuracy, Misclassification rate, and the precision

a. Classification accuracy = (TP + TN) / (TP + TN + FP + FN). (4.1)

   = (0.99+1.0) / (0.99 + 1 + 0.01 + 0.00) = 1.99/ 2 = 0.995 or 99.5%

b. Misclassification rate (error rate) = (FP + FN) / (TP + TN + FN + FP) (4.2)

   =(0.01+0.00) / (0.99 + 1 + 0.01 + 0.00) = 0.01 / 2 = 0.005 or 0.5%

c. Precision = TP/(TP +FP) (4.3)

   = 0. 99 / (0.99 + 0.01) = 0.99 or 99%

University of Ghana http://ugspace.ug.edu.gh

Chapter 5. Conclusion and Future Work

5.1 Introduction

In this final chapter, the findings, implications, and limitations of employing a Deep Convolutional Neural Network (DCNN) for genome error correction have been summarized. Throughout this work, the potential of the DCNN model in addressing challenges and complexities linked to erroneous sequencing data has been explored. This exploration harnessed the capabilities of deep learning and convolutional neural networks to improve the accuracy and efficiency of genome error correction processes.

The first chapter of this thesis introduces fundamental concepts related to genome sequencing and emphasizes the significance of error correction for ensuring the reliability of genome data analysis. It also addresses the limitations of traditional error correction methods and underscores the necessity for an innovative approach capable of adapting to the diverse error patterns present in genome sequencing.

An exploration was conducted into the potential applications of DCNN models for genomic error correction, drawing inspiration from their success in various other domains. The architecture and key components of DCNN models were described, emphasizing their ability to discern intricate patterns and uncover hierarchical representations from unprocessed sequencing data. This makes it a desirable choice for genomic error correction tasks, DCNN also has the potential benefits of autonomous feature extraction and end-to-end learning.

INTEGRI PROCEDAMUS

University of Ghana http://ugspace.ug.edu.gh

5.2 Conclusions

In conclusion, this research has successfully demonstrated the potential of DCNN for correcting errors in genome sequences. The results obtained from the extensive experiments conducted on various genomic datasets showcased the ability of DCNN models to accurately identify and correct errors, leading to improved genome sequence quality. The key contributions of this study include:

Development of an efficient DCNN Model: The DCNN architecture was developed and evaluated for genome sequence error correction. The model was trained on diverse datasets, representing different organisms and sequencing technologies. The results consistently revealed the effectiveness of DCNN in accurately identifying and rectifying errors, surpassing existing error correction methods in terms of accuracy and efficiency.

Robustness to Varying Error Types and Rates: The DCNN model exhibited robustness to different types and rates of errors commonly encountered in genome sequences, including insertion, and deletion errors. The ability of the models to handle diverse error patterns is promising for real-world applications, where error types can vary significantly.

Improved Genome Sequence Quality: The application of DCNN for error correction showed a significant improvement in genome sequence quality, leading to more reliable downstream genetic analysis. The corrected sequences exhibited higher accuracy and reduced false-positive rates, enhancing the accuracy of variant calling, phylogenetic analysis, and other genetic investigations.

5.3 Contribution to Knowledge

The novelty of this work lies in the successful creation of an efficient DCNN that outperforms traditional error correction tools and similar machine-learning approaches The error correction

University of Ghana http://ugspace.ug.edu.gh

process was effectively treated as a classification problem using natural language processing, as opposed to being treated as an image classification process.

Two peer-reviewed papers have been published with the following titles:

1. Deep Neural Network: An Efficient and Optimized Machine Learning Paradigm for Reducing Genome Sequencing Error (International Journal of Engineering Trends and Technology)
2. MulAligner: A Multiple Sequence Alignment Error-Correction Tool Using Deep Learning Algorithm

5.4 Observations

Though our DCNN has shown tremendous success in genome sequencing error correction, some limitations need to be considered. Some of the key challenges or limitations of our DCNN in effectively correcting indels in genome sequencing error corrections are:

a. Diverse Genomic Datasets:

   DCNN models usually rely on learning patterns and features from the training dataset to make predictions. However, in diverse datasets with a wide range of error patterns, the DCNN model may struggle to generalize well to unseen or unfamiliar error types. This means that the model's performance may be limited to the patterns it encountered in the training stage. This will lead to a suboptimal error correction for new or rare error patterns.

INTEGRI PROCEDAMUS

b. Transfer Learning:

   Transfer learning involves leveraging pre-trained models on different but related tasks. However, the DCNN trained on non-genomic data may not directly align with the unique

University of Ghana http://ugspace.ug.edu.gh

characteristics, distinct patterns, or structural properties that may differ from other data types. Also since the training process leads to optimization to fine-tune the model, the optimization process may not necessarily conform with the new dataset that will be introduced to the model for testing.

c. Integrating DCNN-based Error Correction with other Error Correction Methods:
   Integrating the DCNN-based error correction model with other methods may involve complex model architectures or pipelines. This will require careful design and coordinating techniques to ensure effective integration and avoid redundancies or conflicts. This will increase the computational demands which will make the implementation and maintenance more difficult.

5.5 Recommendations

Based on the findings and limitations of this research, the following recommendations are proposed for future work in the field of DCNN for correcting genome sequence errors:

Further research should focus on training DCNN models on more diverse genomic datasets to ensure generalization across different organisms, sequencing technologies, and error profiles. This will enhance the application of DCNN models in real-world scenarios.

Investigate the potential of transfer learning techniques and fine-tuning strategies to leverage pre-trained DCNN models for error correction tasks. This approach can reduce the need for extensive training on large datasets and accelerate the deployment of error correction solutions in different genomic contexts.

University of Ghana http://ugspace.ug.edu.gh

Investigate the integration of DCNN-based error correction methods with existing genome analysis tools and pipelines. This will ensure compatibility and will encourage the adoption of these methods by the genomics community.

University of Ghana http://ugspace.ug.edu.gh

Appendix 1 – Dataset

Homo sapiens isolate NA12878 (Due to the size of the dataset only about three pages are shown here. The entire dataset is attached in word and txt format)

AATCCTTTCAATTCCATTTGATTATGATTCCATTTTATTCATTCAGTGATGATTCCATT

CGTGTCCGTTAGATGAATCCATTGATTCCATTCGATGATGATTGATTTATTTCTTTCA

ATGATGATTCCATTCGGTCCATTAGATGATTCCATTCAATTCAATTTGATGATGATTC

CATTTGTGTCCATTCGATGATTCCATTCTACTCCATTTGATGATGCTTCAATTCAAGT

CCATTCAATGATTCTATATGAGTCCAATCAGTAACTGCTTTCGATTCATTCGATATGA

TTCCATTCGATTCCATTTGATGATGATTCCATTCGGTACCGTTCTATGATTCCATTCG

ATTCCATTCAATGTTGATTCCGTTCGATCCCATTCGATGATTCCCTTTGATTCCATTC

GATGATCATTCCATTTGATTCAATTCGGTGATTCCATTCGATTGTATTCAATGATGAT

TCCATTTGACTCCATTCGATGATGATTCCATTCGATTCCATTCGATTATGATTTCATT

CAATTCCATTCGATGATGATTCCATTCGATTCCATTAGATGATTCCATTCGATTCCAT

TCGATGATGATTCCACTCAAGTCCATTTGATGATTTCTTTTGAGTCCATTTGATGATT

CCATTAGATTCCATTCGATGATGATTCCATTCGATTCCATTCAACAATGATTCCATTC

ATGTCCATTCAATGATTCCATTTGATTCCATTTCATTCCATTTGATGATGATTCCATTC

GAGCCCATTCGGTGATTCCACTCAATTACATTCGATGATGATTCCCTTCTAATCCATT

TGATGATTCCATTCGATTCCATTCGATGGTGATTCCATTCGGGTCCATTCGATGATTC

CATTCGATTCCATTCGATGATGCTTCCACTCAAGTCCATTTGATGATTTCTTTTGAGT

CCATTTGATGATTCCATTAGATTCCATTTGATGATGATTCCATTCGATTCCATTCAAC

AATGATTCCATTCGTGTCCATTCAATGATTCCATTTGACTCCATTTCTTTCCATTTGAT

GATGATTCCATTCGAGCCCATTCGGTGATTCCACTCAATTACATTCAATGATGATTCC

CTTCTATTCCATTTGATGAGTCCATTCGATTCCATTCGATGATGACTGCATTCGGTTC

University of Ghana http://ugspace.ug.edu.gh

ATTCCATTCTTTTCCATTTGATGATGATTGCATTCTATTCCATTCGATGATGATTCCAT
TCGATTCCATTCGATGGTGATTCCATTCGGGTCCATTCGATGATTCCATTCGATTCCA
TTCGATGATGATTCCATTCGAGTACATTCAATGATTCCATTTAAGTCCATTCGAAGAT
TACTTTCATTTCCATTTGATGATTCCATTTGAGTCCATTCGATGATTCCGTTCAAGTC
CATTCAATGTTTTCTTTTGATTCCAATCAATATTGATTCCATTTGAGTCCATTCGATG
ATTCCATTCGAGTGCATTCCATGATGATTCCATTCGAGTCCATTTGATGATTCCATTT
GATTTCATTTGGTGATGATTCCATTTGATTCCATTCGATGATTCCATTCAAGTCCATT
CATTGATTATATTCGAGTCCATTAAATGATTTCATTCGTTTCCATTCGATGATGACTC
CATTCGAGTCCATTCAAGGATGATTCCAGAAGATTCCATTCGATGATAACGTTGGAT
TCCATTCTTTGTTTCATTTCGATACATTTTGGTGATGATTCCGTTTGATTTCATTTGAT
GATCCCATTCGATTCTATTTGATGATGATTCCATTCGATTCCATTGGATGAAAATTCC
ATTCATTTCCATTCATTGATGATTCCATTCTATTCTATTCGATGCTGATTCTATTCAAT
TCCATTCAACGATGATTCCATTCGATTTCATTCGATGATGATTCCTTTCGAGACCATT
CGATGATTCCATTCACTTCATTCGATGATGATTCCATTCAATTCCATTAGATGATTCC
ATTAGAATCCATTTGATGACGATTCCATTCGATTCCATTCGATGATGATTCCATACGA
TTCCATTGGATGATGACTCCTTTCATTTCCATTCGATGATTATTCCTTTCGACTCCATT
TGATGTTGATTCTTTCAATTCCGTTTGATGATGTTTCCATTTGATTCCATTCGATGATG
ATTCCATTCCATTCCATTCGATGATTCCATTCAATTCCATTAGATGATGATTCCATTA
GAGTCCATTCAATGATTCCATTCGATTCCATTCTATGATAATTCCACTGGAGTCCATT
CGATGATTCTATTTGATTCTATTCAATAATGATTCCATTCGAGTCCATTCGATGATTG
CTTTTGAGTCTATTCGAAGATTCCATTCAATACCATTCAATGATGTTTCCATTCCAGT
CCATTCAATAATTCCATCCGATTCCATTCGATGATGTCTGCACTCAGTTCCATTTGAT

University of Ghana http://ugspace.ug.edu.gh

ATTCGATTCCATTCGATTATGATTACATTCAATTCCATTCGATGATGATTTCATTGGA
TTCCATTCCATGATGATTCCATTCGATTCCATTCAATGATGATTCCATTCAATTCCAT
TTGATGAAAATTCCATTCAATTCCATTCGATGATGATTCCATTCGATTCTATTCGATG
CTGACTCTATTCGATTCATTTGGATGATGATTCCATTCGATTCCATTTGATGATTCCA
TTCGATTCCATTCAATTATGATTCCATTCGAGACCGTTCGATGATTCCATTCAATTCT
ATTCAATGGTGATTCCATTCGAGTCCATTCGATGATTCCATTCAAGTCCATTCTATGA
TTCCATCTCTTTCCATTCAATGATGATTCCATTCGATGATGATTCCATTCGATTCCATT
CAATGATGATTCCATTGGATTCCATTCGATGATTCCATTCGATTCCATTTGATGGTGA
TTCCATTCGGGTTCATTCGATGATTCCATTCGATTCCATTCGATGATGATTCCATTTG
AGTACATTCAATGATTCCATTTAAGTCCATTCTAAGATTACTTTCAATTCCATTTGAT
GATTCCATTTGAGTCCATTCGATGATTCCATTCAAGTCCATTCGATGTTTTCTTTCGA
TTCCACTCGATATTGATTCCATCTGAGTCCGTTCGATGATTCCATTCAAGTGCATTCC
ATGATTTCATTCGACTCCATTCGATGATCATTCCATTCGAGTCCATTCGATGATTCCA
TTTGATTTCATTCGGTGATGATTCCATTCGATTCCTTTCGATGCTTCCATTCAAGTCC
ATTCATTGATTACATTCGAGTCCATTAAATGATTCCATTCGTTTCCATTCGATGATGA
CTCCATTCGAGTCCATTCAATGATGATTCCAGTCGATTCCATTCGATGATAACGTTGG
ATTCCATTCTTTGTTTTTTTTCGATTCATTTGGTGATGATTCCGTTTGATTTCATTTG
ATGATCCCATTCGATTCTATTTGATGATGATTCCATTCGATTGCATTGGATGAAAATT
CCATTCGTTTCCATTCATTGATGATTCCATTCGATTTTATTCGATGCTGATTCTATTCA
ATTCCATTCGACGATGATTCCATTCGATTTCATTCGATGATGATTGCTTTCGAGACCA
TTCGATGATTCCATTCACTTCATTCGATGATGATTCTATTCAATTCCATTAGATGATT
CCATTAGAATGCATTTGGTGATGATTCCATTCGATTCCATTTGATGATGATTCCATAC

University of Ghana http://ugspace.ug.edu.gh

Reference

[1] K. C. Wong, Y. Li, and Z. Zhang, "Unsupervised learning in genome informatics," in *Unsupervised Learning Algorithms*, 2016, pp. 405–448. doi: 10.1007/978-3-319-24211-8_15.

[2] D. Nachmanson *et al.*, "Targeted genome fragmentation with CRISPR / Cas9 enables fast and efficient enrichment of small genomic regions and ultra-accurate sequencing with low DNA input (CRISPR-DS)," pp. 1589–1599, 2018, doi: 10.1101/gr.235291.118.

[3] B. NL and S. M, "DNA sequence of a region of the phi X174 genome coding for a ribosome binding site," *Nature*, vol. 265, no. 5596, pp. 695–698, 1977, doi: 10.1038/265695A0.

[4] S. H and G. R, "Capillary gel electrophoresis for rapid, high resolution DNA sequencing," *Nucleic Acids Res.*, vol. 18, no. 6, pp. 1415–1419, Mar. 1990, doi: 10.1093/NAR/18.6.1415.

[5] N. J. Dovichi, "DNA sequencing by capillary electrophoresis," *Electrophoresis*, vol. 18, no. 12–13, pp. 2393–2399, Nov. 1997.

[6] J. M. Heather and B. Chain, "The Sequence of Sequencers : The History of SC," *Genomics*, 2015, doi: 10.1016/j.ygeno.2015.11.003.

[7] A. K. Das, K. Lee, and S. Park, "ParLECH : Parallel Long-Read Error Correction with Hadoop," *2018 IEEE Int. Conf. Bioinforma. Biomed.*, pp. 341–348, 2018.

[8] J. Schröder, H. Schröder, S. J. Puglisi, R. Sinha, and B. Schmidt, "SHREC: A short-read error correction method," *Bioinformatics*, vol. 25, no. 17, pp. 2157–2163, 2009, doi:

University of Ghana http://ugspace.ug.edu.gh

10.1093/bioinformatics/btp379.

[9] E. Borrayo, E. G. Mendizabal-Ruiz, H. Velez-Perez, R. Romo-Vazquez, A. P. Mendizabal, and J. Alejandro Morales, "Genomic signal processing methods for computation of alignment-free distances from dna sequences," *PLoS One*, vol. 9, no. 11, pp. 1–13, 2014, doi: 10.1371/journal.pone.0110954.

[10] P. A. Pevzner, H. Tang, and M. S. Waterman, "An Eulerian path approach to DNA fragment assembly," *Proc. Natl. Acad. Sci. U. S. A.*, vol. 98, no. 17, pp. 9748–9753, 2001, doi: 10.1073/pnas.171285098.

[11] D. Laehnemann, A. Borkhardt, and A. Carolyn McHardy Corresponding author Alice McHardy, "Denoising DNA deep sequencing data-high-throughput sequencing errors and their correction," *Brief. Bioinform.*, vol. 17, no. 1, pp. 154–179, 2016, doi: 10.1093/bib/bbv029.

[12] L. Salmela and J. Schröder, "Correcting errors in short reads by multiple alignments," *Bioinformatics*, vol. 27, no. 11, pp. 1455–1461, 2011, doi: 10.1093/bioinformatics/btr170.

[13] K. Sameith, J. G. Roscito, and M. Hiller, "Iterative error correction of long sequencing reads maximizes accuracy and improves contig assembly," *Brief. Bioinform.*, vol. 18, no. 1, pp. 1–8, 2017, doi: 10.1093/bib/bbw003.

[14] I. Akogwu, N. Wang, and C. Zhang, "Factorial Analysis of Error Correction Performance Using Simulated Next-Generation Sequencing Data," pp. 1164–1169, 2016.

[15] Q. Lou, "BRAWL: A Spintronics-based Portable Basecalling-In-Memory Architecture for Nanopore Genome Sequencing," *IEEE Comput. Archit. Lett.*, vol. PP, no. c, p. 1, 2018,

University of Ghana http://ugspace.ug.edu.gh

doi: 10.1109/LCA.2018.2882384.

[16] M. Qin *et al.*, "LRScaf : improving draft genomes using long noisy reads," pp. 1–12, 2019.

[17] I. Abnizova, R. te Boekhorst, and Y. L. Orlov, "Computational Errors and Biases in Short Read Next Generation Sequencing," *J. Proteomics Bioinform.*, vol. 10, no. 1, pp. 1–17, 2017, doi: 10.4172/jpb.1000420.

[18] Rp. Oplin *et al.*, "Creating a universal SNP and small indel variant caller with deep neural networks," p. 092890, 2016, doi: 10.1101/092890.

[19] F. J. Rang, W. P. Kloosterman, and J. De Ridder, "From squiggle to basepair : computational approaches for improving nanopore sequencing read accuracy," pp. 1–11, 2018.

[20] S. Suzuki, N. Ono, C. Furusawa, B. W. Ying, and T. Yomo, "Comparison of sequence reads obtained from three next-generation sequencing platforms," *PLoS One*, vol. 6, no. 5, pp. 1–6, 2011, doi: 10.1371/journal.pone.0019534.

[21] R. K. Varshney, S. N. Nayak, G. D. May, and S. A. Jackson, "Next-generation sequencing technologies and their implications for crop genetics and breeding," *Trends Biotechnol.*, vol. 27, no. 9, pp. 522–530, 2009, doi: 10.1016/j.tibtech.2009.05.006.

[22] M. Imelfort and D. Edwards, "De novo sequencing of plant genomes using second-generation technologies," vol. 10, no. 6, 2009, doi: 10.1093/bib/bbp039.

[23] A. Stützer *et al.*, "Analysis of protein-DNA interactions in chromatin by UV induced cross-linking and mass spectrometry," *Nat. Commun.*, pp. 1–12, 2020, doi:

University of Ghana http://ugspace.ug.edu.gh

10.1038/s41467-020-19047-7.

[24] J. M. Heather and B. Chain, “Genomics The sequence of sequencers : The history of sequencing DNA,” *Genomics*, vol. 107, no. 1, pp. 1–8, 2016, doi: 10.1016/j.ygeno.2015.11.003.

[25] J. D. Dinman and R. B. Wickner, “Ribosomal frameshifting efficiency and gag/gag-pol ratio are critical for yeast M1 double-stranded RNA virus propagation.,” *J. Virol.*, vol. 66, no. 6, pp. 3669–3676, 1992, doi: 10.1128/jvi.66.6.3669-3676.1992.

[26] F. Sander, A. R. Goulson, and H. Road, “A Rapid Method for Determining Sequences in DNA by Primed Synthesis with DNA Polymerase,” pp. 441–448, 1976.

[27] M. Margulies *et al.*, “Genome sequencing in microfabricated high-density picolitre reactors,” vol. 437, no. September, pp. 376–381, 2005, doi: 10.1038/nature03959.

[28] R. E. Lenski, P. Sniegowski, and P. Gerrish, “Evolution of high mutation rates in experimental populations of E. coli,” *Nature*, vol. 387, no. June, pp. 703–705, 1997.

[29] L. Liu *et al.*, “Comparison of next-generation sequencing systems,” *J. Biomed. Biotechnol.*, vol. 2012, 2012, doi: 10.1155/2012/251364.

[30] J. Henson, G. Tischler, and Z. Ning, “Europe PMC Funders Group Next-generation sequencing and large genome assemblies,” vol. 13, no. 8, pp. 901–915, 2014, doi: 10.2217/pgs.12.72.Next-generation.

[31] M. C. Schatz, A. L. Delcher, and S. L. Salzberg, “Assembly of large genomes using second-generation sequencing,” *Genome Res.*, vol. 20, no. 9, pp. 1165–1173, 2010, doi: 10.1101/gr.101360.109.

University of Ghana http://ugspace.ug.edu.gh

[32] S. C. Schuster, “Next-generation sequencing transforms today ’ s biology,” vol. 5, no. 1, pp. 16–18, 2008, doi: 10.1038/NMETH1156.

[33] H. Ellegren, “Sequencing goes 454 and takes large-scale genomics into the wild,” *Mol. Ecol.*, vol. 17, no. 7, pp. 1629–1631, 2008, doi: 10.1111/j.1365-294X.2008.03699.x.

[34] M. A. Quail *et al.*, “A tale of three NGS sequencing platforms,” *BMC Genomics*, vol. 13, no. 341, p. 13, 2012.

[35] C. Bernet, M. Garret, B. De Barbeyrac, C. Bebear, and J. Bonnet, “Detection of mycoplasma pneumoniae by using the polymerase chain reaction,” *J. Clin. Microbiol.*, vol. 27, no. 11, pp. 2492–2496, 1989, doi: 10.1128/jcm.27.11.2492-2496.1989.

[36] M. T. Gansauge *et al.*, “Single-stranded DNA library preparation from highly degraded DNA using T4 DNA ligase,” *Nucleic Acids Res.*, vol. 45, no. 10, 2017, doi: 10.1093/nar/gkx033.

[37] A. Alderborn, A. Kristofferson, and U. Hammerling, “Determination of single-nucleotide polymorphisms by real-time pyrophosphate DNA sequencing,” *Genome Res.*, vol. 10, no. 8, pp. 1249–1258, 2000, doi: 10.1101/gr.10.8.1249.

[38] B. Merriman, I. Torrent, and J. M. Rothberg, “Progress in Ion Torrent semiconductor chip based sequencing,” *Electrophoresis*, vol. 33, no. 23, pp. 3397–3417, 2012, doi: 10.1002/elps.201200424.

[39] N. J. Loman *et al.*, “Performance comparison of benchtop high-throughput sequencing platforms,” *Nat. Biotechnol.*, vol. 30, no. 5, pp. 434–439, 2012, doi: 10.1038/nbt.2198.

[40] M. A. Weinstock, “Epidemiology and UV exposure.,” *J. Invest. Dermatol.*, vol. 133, no.

University of Ghana http://ugspace.ug.edu.gh

E1, pp. e11-4, 2013, doi: 10.1038/jid.2013.248.

[41] M. Kircher, S. Sawyer, and M. Meyer, "Double indexing overcomes inaccuracies in multiplex sequencing on the Illumina platform," *Nucleic Acids Res.*, vol. 40, no. 1, pp. 1–8, 2012, doi: 10.1093/nar/gkr771.

[42] I. Birol *et al.*, "Assembling the 20 Gb white spruce (Picea glauca) genome from whole-genome shotgun sequencing data," *Bioinformatics*, vol. 29, no. 12, pp. 1492–1497, 2013, doi: 10.1093/bioinformatics/btt178.

[43] M. Kircher, *Chapter 23 Analysis of High-Throughput Ancient DNA Sequencing Data*, no. January. 2016. doi: 10.1007/978-1-61779-516-9.

[44] M. Meyerson, S. Gabriel, and G. Getz, "Advances in understanding cancer genomes through second-generation sequencing," *Nat. Rev. Genet.*, vol. 11, no. 10, pp. 685–696, 2010, doi: 10.1038/nrg2841.

[45] K. V. Voelkerding, S. A. Dames, and J. D. Durtschi, "Next-generation sequencing:from basic research to diagnostics," *Clin. Chem.*, vol. 55, no. 4, pp. 641–658, 2009, doi: 10.1373/clinchem.2008.112789.

[46] D. Lipson *et al.*, "Quantification of the yeast transcriptome by single-molecule sequencing," *Nat. Biotechnol.*, vol. 27, no. 7, pp. 652–658, 2009, doi: 10.1038/nbt.1551.

[47] J. Shendure and H. Ji, "Next-generation DNA sequencing," *Nat. Biotechnol.*, vol. 26, no. 10, pp. 1135–1145, 2008, doi: 10.1038/nbt1486.

[48] W. J. Ansorge, "Next-generation DNA sequencing techniques," *N. Biotechnol.*, vol. 25, no. 4, pp. 195–203, 2009, doi: 10.1016/j.nbt.2008.12.009.

University of Ghana http://ugspace.ug.edu.gh

[49] T. S. Seo, X. Bai, H. Ruparel, Z. Li, N. J. Turro, and J. Ju, “Photocleavable fluorescent nucleotides for DNA sequencing on a chip constructed by site-specific coupling chemistry,” *Proc. Natl. Acad. Sci. U. S. A.*, vol. 101, no. 15, pp. 5488–5493, 2004, doi: 10.1073/pnas.0401138101.

[50] M. El Khoury, “The Effects of Heat and Explosions on Forensic DNA Analyses by The Effects of Heat and Explosions on Forensic DNA Analyses,” no. March, 2019.

[51] J. C. Venter *et al.*, “Celera_genoma,” *Science (80-.).*, vol. 291, no. February, pp. 1–49, 2001, [Online]. Available: sftp://cerca@192.168.2.5/home/cerca/Desktop/data/laptop_files/info/biologia/homo_sapiens/human_genome/Celera_genoma.pdf%5Cnpapers2://publication/uuid/21C9A6AC-3A9B-4931-BB7D-CE922633B16B

[52] S. Pabinger *et al.*, “A survey of tools for variant analysis of next-generation genome sequencing data,” *Brief. Bioinform.*, vol. 15, no. 2, pp. 256–278, 2014, doi: 10.1093/bib/bbs086.

[53] M. M. Dias, Vera Junn, Eunsung Mouradian, “基因的改变NIH Public Access,” *Bone*, vol. 23, no. 1, pp. 1–7, 2008, doi: 10.1002/0471142727.mb0710s92.Single.

[54] E. Webb, “Pacific Biosciences’ Develops Third Generation DNA Sequencing Technology,” pp. 1–14, 2008, [Online]. Available: file:///Users/davidlee/Dropbox/Papers2/2008/Webb/Webb - 2008 ,1-14 - Pacific Biosciences’ Develops Third Generation.pdf%5Cnpapers2://publication/uuid/710939FF-385A-493A-808A-A6365FD24140

University of Ghana http://ugspace.ug.edu.gh

[55] H. M. Manske and D. P. Kwiatkowski, "SNP-o-matic," *Bioinformatics*, vol. 25, no. 18, pp. 2434–2435, 2009, doi: 10.1093/bioinformatics/btp403.

[56] J. M. Rizzo and M. J. Buck, "Key principles and clinical applications of 'next-generation' DNA sequencing," *Cancer Prev. Res.*, vol. 5, no. 7, pp. 887–900, 2012, doi: 10.1158/1940-6207.CAPR-11-0432.

[57] L. Mamanova *et al.*, "Target-enrichment strategies for next-generation sequencing," *Nat. Methods*, vol. 7, no. 2, pp. 111–118, 2010, doi: 10.1038/nmeth.1419.

[58] J. Weaver, "Promise and Pitfalls of Third-Generation Sequencing," *Biotechniques*, pp. 25–28, 2012.

[59] Y. Liu, C. Lan, M. Blumenstein, and J. Li, "Bi-level error correction for PacBio long reads," vol. 5963, no. NOVEMBER, 2017, doi: 10.1109/TCBB.2017.2780832.

[60] H. Li and R. Durbin, "Fast and accurate long-read alignment with Burrows-Wheeler transform," *Bioinformatics*, vol. 26, no. 5, pp. 589–595, 2010, doi: 10.1093/bioinformatics/btp698.

[61] D. Savel, T. Laframboise, A. Grama, and M. Koyuturk, "Pluribus — Exploring the Limits of Error Correction Using a Suffix Tree," *IEEE/ACM Trans. Comput. Biol. Bioinforma.*, vol. 14, no. 6, pp. 1378–1388, 2017, doi: 10.1109/TCBB.2016.2586060.

[62] M. A. Field *et al.*, "Recurrent miscalling of missense variation from short-read genome sequence data," *BMC Genomics*, vol. 20, no. Suppl 8, pp. 1–9, 2019, doi: 10.1186/s12864-019-5863-2.

[63] M. Kchouk, "An Error Correction Algorithm for NGS Data," pp. 84–87, 2017, doi:

University of Ghana http://ugspace.ug.edu.gh

10.1109/DEXA.2017.33.

[64] I. Akogwu, N. Wang, C. Zhang, and P. Gong, “A comparative study of k -spectrum-based error correction methods for next- generation sequencing data analysis,” *Hum. Genomics*, vol. 10, no. Suppl 2, 2016, doi: 10.1186/s40246-016-0068-0.

[65] S. Pal and S. Aluru, “In search of perfect reads,” *BMC Bioinformatics*, vol. 16, no. Suppl 17, p. S7, 2015, doi: 10.1186/1471-2105-16-S17-S7.

[66] H. Article, “The Spectrum of Replication Errors in the Absence of Error Correction Assayed Across the Whole Genome of Escherichia coli,” vol. 209, no. August, pp. 1043–1054, 2018.

[67] D. Mapleson, G. G. Accinelli, G. Kettleborough, J. Wright, and B. J. Clavijo, “KAT: A K-mer analysis toolkit to quality control NGS datasets and genome assemblies,” *Bioinformatics*, vol. 33, no. 4, pp. 574–576, 2017, doi: 10.1093/bioinformatics/btw663.

[68] M. Heydari, G. Miclotte, P. Demeester, Y. Van De Peer, and J. Fostier, “Evaluation of the impact of Illumina error correction tools on de novo genome assembly,” pp. 1–13, 2017, doi: 10.1186/s12859-017-1784-8.

[69] D. R. Kelley, M. C. Schatz, and S. L. Salzberg, “Quake: Quality-aware detection and correction of sequencing errors,” *Genome Biol.*, vol. 11, no. 11, p. R116, Nov. 2010, doi: 10.1186/gb-2010-11-11-r116.

[70] J. C. Dohm, C. Lottaz, T. Borodina, and H. Himmelbauer, “Substantial biases in ultra-short read data sets from high-throughput DNA sequencing,” *Nucleic Acids Res.*, vol. 36, no. 16, 2008, doi: 10.1093/nar/gkn425.

University of Ghana http://ugspace.ug.edu.gh

[71] X. Yang, K. S. Dorman, and S. Aluru, "Reptile : representative tiling for short read error correction," vol. 26, no. 20, pp. 2526–2533, 2010, doi: 10.1093/bioinformatics/btq468.

[72] X. Yang, K. S. Dorman, and S. Aluru, "Sequence analysis Reptile: representative tiling for short read error correction," vol. 26, pp. 2526–2533, 2010, doi: 10.1093/bioinformatics/btq468.

[73] Y. Heo, X. L. Wu, D. Chen, J. Ma, and W. M. Hwu, "BLESS: Bloom filter-based error correction solution for high-throughput sequencing reads," *Bioinformatics*, vol. 30, no. 10, pp. 1354–1362, 2014, doi: 10.1093/bioinformatics/btu030.

[74] P. Greenfield, K. Duesing, A. Papanicolaou, and D. C. Bauer, "Blue : correcting sequencing errors using consensus and context," no. Yang 2013, pp. 1–8, 2014.

[75] L. Ilie and M. Molnar, "RACER : Rapid and accurate correction of errors in reads," vol. 29, no. 19, pp. 2490–2493, 2013, doi: 10.1093/bioinformatics/btt407.

[76] L. Song, L. Florea, and B. Langmead, "Lighter: fast and memory-efficient sequencing error correction without counting," *Genome Biol.*, vol. 15, no. 11, p. 509, 2014, doi: 10.1186/s13059-014-0509-9.

[77] H. Li, "BFC: Correcting Illumina sequencing errors," *Bioinformatics*, vol. 31, no. 17, pp. 2885–2887, 2015, doi: 10.1093/bioinformatics/btv290.

[78] D. Laehnemann, A. Borkhardt, and A. C. Mchardy, "Denoising DNA deep sequencing data — high-throughput sequencing errors and their correction," vol. 17, no. April 2015, pp. 154–179, 2016, doi: 10.1093/bib/bbv029.

[79] E. Marinier, D. G. Brown, and B. J. McConkey, "Pollux: Platform independent error

University of Ghana http://ugspace.ug.edu.gh

correction of single and mixed genomes," *BMC Bioinformatics*, vol. 16, no. 1, pp. 1–12, 2015, doi: 10.1186/s12859-014-0435-6.

[80] M. Długosz and S. Deorowicz, "RECKONER: read error corrector based on KMC", doi: 10.1093/bioinformatics/btw746.

[81] Y. Liu, J. Schröder, and B. Schmidt, "Musket: A multistage k-mer spectrum-based error corrector for Illumina sequence data," *Bioinformatics*, vol. 29, no. 3, pp. 308–315, 2013, doi: 10.1093/bioinformatics/bts690.

[82] A. D. Smith, Z. Xuan, and M. Q. Zhang, "Using quality scores and longer reads improves accuracy of Solexa read mapping," *BMC Bioinformatics*, vol. 9, pp. 1–8, 2008, doi: 10.1186/1471-2105-9-128.

[83] M. H. Schulz *et al.*, "Fiona: A parallel and automatic strategy for read error correction," *Bioinformatics*, vol. 30, no. 17, pp. 356–363, 2014, doi: 10.1093/bioinformatics/btu440.

[84] A. Allam, P. Kalnis, and V. Solovyev, "Karect: Accurate correction of substitution, insertion and deletion errors for next-generation sequencing data," *Bioinformatics*, vol. 31, no. 21, pp. 3421–3428, 2015, doi: 10.1093/bioinformatics/btv415.

[85] W. Kao, A. H. Chan, and Y. S. Song, "ECHO : A reference-free short-read error correction algorithm," pp. 1181–1192, 2011, doi: 10.1101/gr.111351.110.21.

[86] X. Yang, S. P. Chockalingam, and S. Aluru, "A survey of error-correction methods for next-generation sequencing," *Brief. Bioinform.*, vol. 14, no. 1, pp. 56–66, 2013, doi: 10.1093/bib/bbs015.

[87] J. Schröder, H. Schröder, S. J. Puglisi, R. Sinha, and B. Schmidt, "SHREC: A short-read

University of Ghana http://ugspace.ug.edu.gh

error correction method," *Bioinformatics*, vol. 25, no. 17, pp. 2157–2163, Sep. 2009, doi: 10.1093/bioinformatics/btp379.

[88] Y. Gu, Q. Zhu, X. Liu, Y. Dong, C. T. Brown, and S. Pramanik, "Using disk based index and box queries for genome sequencing error correction," *Proc. 8th Int. Conf. Bioinforma. Comput. Biol. BICOB 2016*, no. April, pp. 69–76, 2016.

[89] M. Z. Molnar and S. Lucian Ilie, "Error Correction and de novo Genome Assembly of DNA Sequencing Data," no. December, 2017, [Online]. Available: http://ir.lib.uwo.ca/etd%0Ahttp://ir.lib.uwo.ca/etd/5050

[90] L. Salmela and A. Bateman, "Correction of sequencing errors in a mixed set of reads," vol. 26, no. 10, pp. 1284–1290, 2010, doi: 10.1093/bioinformatics/btq151.

[91] L. Ilie, F. Fazayeli, and S. Ilie, "HiTEC: Accurate error correction in high-throughput sequencing data," *Bioinformatics*, vol. 27, no. 3, pp. 295–302, 2011, doi: 10.1093/bioinformatics/btq653.

[92] A. K. Das, S. Goswami, K. Lee, and S. Park, "Open Access A hybrid and scalable error correction algorithm for indel and substitution errors of long reads," *BMC Genomics*, vol. 20, no. Suppl 11, pp. 1–15, 2019, doi: 10.1186/s12864-019-6286-9.

[93] L. Salmela and E. Rivals, "Sequence analysis LoRDEC : accurate and efficient long read error correction," vol. 30, no. 24, pp. 3506–3514, 2014, doi: 10.1093/bioinformatics/btu538.

[94] J. R. Wang, J. Holt, L. Mcmillan, and C. D. Jones, "Open Access FMLRC : Hybrid long read error correction using an FM-index," pp. 1–11, 2018.

University of Ghana http://ugspace.ug.edu.gh

[95] T. Gagie, G. Manzini, and J. Sirén, “Wheeler graphs: A framework for BWT-based data structures,” *Theor. Comput. Sci.*, vol. 698, pp. 67–78, 2017, doi: 10.1016/j.tcs.2017.06.016.

[96] M. Heydari, G. Miclotte, Y. Van De Peer, and J. Fostier, “Illumina error correction near highly repetitive DNA regions improves de novo genome assembly,” *BMC Bioinformatics*, vol. 20, no. 1, pp. 1–13, 2019, doi: 10.1186/s12859-019-2906-2.

[97] N. Du, J. Chen, and Y. Sun, “Improving the sensitivity of long read overlap detection using grouped short k -mer matches,” vol. 20, no. Suppl 2, 2019, doi: 10.1186/s12864-019-5475-x.

[98] O. Choudhury, A. Chakrabarty, and S. J. Emrich, “OPEN HECIL : A Hybrid Error Correction Algorithm for Long Reads with Iterative Learning,” *Sci. Rep.*, no. January, pp. 1–9, 2018, doi: 10.1038/s41598-018-28364-3.

[99] G. Marçais and C. Kingsford, “A fast, lock-free approach for efficient parallel counting of occurrences of k-mers,” *Bioinformatics*, vol. 27, no. 6, pp. 764–770, 2011, doi: 10.1093/bioinformatics/btr011.

[100] M. Vyverman, B. De Baets, V. Fack, and P. Dawyndt, “SURVEY AND SUMMARY: Prospects and limitations of full-text index structures in genome analysis,” *Nucleic Acids Res.*, vol. 40, no. 15, pp. 6993–7015, 2012, doi: 10.1093/nar/gks408.

[101] M. Tahir, M. Sardaraz, A. A. Ikram, and H. Bajwa, “Review of Genome Sequence Short Read Error Correction Algorithms,” *Am. J. Bioinforma. Res.*, vol. 3, no. 1, pp. 1–9, 2013, doi: 10.5923/j.bioinformatics.20130301.01.

University of Ghana http://ugspace.ug.edu.gh

[102] H. M. Al-Barhamtoshy and R. A. Younis, “DNA sequence error corrections based on tensorflow,” *Proc. - 2020 21st Int. Arab Conf. Inf. Technol. ACIT 2020*, 2020, doi: 10.1109/ACIT50332.2020.9300094.

[103] G. Datasets, M. Krachunov, and M. Nisheva, “Application of Machine Learning Models in Error and Variant Detection in High-Variation,” 2017, doi: 10.3390/computers6040029.

[104] L. Wang, L. Qu, L. Yang, Y. Wang, and H. Zhu, “NanoReviser : An Error-Correction Tool for Nanopore Sequencing Based on a Deep Learning Algorithm,” vol. 11, no. August, pp. 1–12, 2020, doi: 10.3389/fgene.2020.00900.

[105] K. Kotlarz, M. Mielczarek, T. Suchocki, B. Czech, B. Guldbrandtsen, and J. Szyda, “The application of deep learning for the classification of correct and incorrect SNP genotypes from whole-genome DNA sequencing pipelines,” pp. 607–616, 2020.

[106] C. Angermueller, T. Pärnamaa, L. Parts, and O. Stegle, “Deep learning for computational biology,” *Mol. Syst. Biol.*, vol. 12, no. 7, p. 878, 2016, doi: 10.15252/msb.20156651.

[107] M. K. K. Leung, A. Delong, B. Alipanahi, and B. J. Frey, “Machine learning in genomic medicine: A review of computational problems and data sets,” *Proc. IEEE*, vol. 104, no. 1, pp. 176–197, 2016, doi: 10.1109/JPROC.2015.2494198.

[108] G. Zhang, B. Eddy Patuwo, and M. Y. Hu, “Forecasting with artificial neural networks: The state of the art,” *Int. J. Forecast.*, vol. 14, no. 1, pp. 35–62, 1998, doi: 10.1016/S0169-2070(97)00044-7.

[109] S. K. Pal and S. Mitra, “Multilayer perceptron, fuzzy sets, classifiaction,” *IEEE Transactions on Neural Networks*, vol. 3, no. 5. pp. 683–697, 1992.

University of Ghana http://ugspace.ug.edu.gh

[110] R. Nason, P. Lloyd, and I. Ginns, "Format-free databases and the construction of knowledge in primary school science projects," *Res. Sci. Educ.*, vol. 26, no. 3, pp. 353–373, 1996, doi: 10.1007/BF02356945.

[111] J. Fauconnier and B. Rothenburger, "A Supervised Machine Learning Approach for Taxonomic Relation Recognition through Non-linear Enumerative Structures Categories and Subject Descriptors," pp. 423–425.

[112] J. Erman, A. Mahanti, and M. Arlitt, "QRP05-4: Internet Traffic Identification using Machine Learning," in *IEEE Globecom 2006*, IEEE, Nov. 2006, pp. 1–6. doi: 10.1109/GLOCOM.2006.443.

[113] B. Li, M. H. Gunes, G. Bebis, and J. Springer, "A supervised machine learning approach to classify host roles on line using sFlow," *Proc. first Ed. Work. High Perform. Program. Netw. - HPPN '13*, p. 53, 2013, doi: 10.1145/2465839.2465847.

[114] S. Amershi and C. Conati, "Unsupervised and Supervised Machine Learning in User Modeling for Intelligent Learning Environments," pp. 72–81, 2007.

[115] J. Koushik, "Understanding Convolutional Neural Networks," 2016, [Online]. Available: http://arxiv.org/abs/1605.09081

[116] N. Japkowicz, "Supervised Versus Unsupervised Binary-Learning by Feedforward Neural Networks," vol. 5, pp. 97–122, 2001.

[117] X. Han, D. Zhou, S. Wang, and S. Kimura, "CNN-MERP: An FPGA-based memory-efficient reconfigurable processor for forward and backward propagation of convolutional neural networks," *Proc. 34th IEEE Int. Conf. Comput. Des. ICCD 2016*, pp. 320–327,

University of Ghana http://ugspace.ug.edu.gh

2016, doi: 10.1109/ICCD.2016.7753296.

[118] M. Naumov, “Parallel Complexity of Forward and Backward Propagation,” vol. 2, no. 3, pp. 1–18, 2017, [Online]. Available: http://arxiv.org/abs/1712.06577

[119] L. Bottou, “Stochastic Gradient Descent Tricks,” *Neural Networks: Tricks of the Trade*, vol. 1, no. 1, pp. 421–436, 2012, doi: 10.1007/978-3-642-35289-8_25.

[120] M. D. Zeiler and R. Fergus, “Stochastic pooling for regularization of deep convolutional neural networks,” *1st Int. Conf. Learn. Represent. ICLR 2013 - Conf. Track Proc.*, pp. 1–9, 2013.

[121] R. Wu, S. Yan, Y. Shan, Q. Dang, and G. Sun, “Deep Image: Scaling up Image Recognition,” 2015, [Online]. Available: http://arxiv.org/abs/1501.02876

[122] R. Luo, F. J. Sedlazeck, T. W. Lam, and M. C. Schatz, “A multi-task convolutional deep neural network for variant calling in single molecule sequencing,” *Nat. Commun.*, vol. 10, no. 1, 2019, doi: 10.1038/s41467-019-09025-z.

[123] J. Dysart, “Deep Learning of Representations for Unsupervised and Transfer Learning Yoshua,” *Can. Nurse*, vol. 104, no. 4, p. 4, 2008, doi: 10.1109/IJCNN.2011.6033302.

[124] H. Li, J. Wang, M. Tang, and X. Li, “Polarization-dependent effects of an Airy beam due to the spin-orbit coupling,” *J. Opt. Soc. Am. A Opt. Image Sci. Vis.*, vol. 34, no. 7, pp. 1114–1118, 2017, doi: 10.1002/ecs2.1832.

[125] H. Salman, J. Grover, and T. Shankar, “Hierarchical Reinforcement Learning for Sequencing Behaviors,” vol. 2733, pp. 2709–2733, 2018, doi: 10.1162/NECO.

[126] P. Ghamisi, Y. Chen, and X. X. Zhu, “A Self-Improving Convolution Neural Network for

University of Ghana http://ugspace.ug.edu.gh

the Classification of Hyperspectral Data," *IEEE Geosci. Remote Sens. Lett.*, vol. 13, no. 10, pp. 1537–1541, 2016, doi: 10.1109/LGRS.2016.2595108.

[127] N. Tavakoli, "Modeling Genome Data Using Bidirectional LSTM," *2019 IEEE 43rd Annu. Comput. Softw. Appl. Conf.*, vol. 2, pp. 183–188, 2019, doi: 10.1109/COMPSAC.2019.10204.

[128] K. Hornik, M. Stinchcombe, and H. White, "Kornick et. al.," *Neural Networks*, vol. 2, pp. 359–366, 1989.

[129] R. O. Duda, P. E. Hart, and D. G. Stork, "PATTERN Second Edition," 2001.

[130] V. Sangeetha and K. J. R. Prasad, "Syntheses of novel derivatives of 2-acetylfuro[2,3-a]carbazoles, benzo[1,2-b]-1,4-thiazepino[2,3-a]carbazoles and 1-acetyloxycarbazole-2-carbaldehydes," *Indian J. Chem. - Sect. B Org. Med. Chem.*, vol. 45, no. 8, pp. 1951–1954, 2006, doi: 10.1002/chin.200650130.

[131] J. Zuallaert, M. Kim, Y. Saeys, and W. De Neve, "Interpretable convolutional neural networks for effective translation initiation site prediction," *Proc. - 2017 IEEE Int. Conf. Bioinforma. Biomed. BIBM 2017*, vol. 2017-Janua, no. 2007, pp. 1233–1237, 2017, doi: 10.1109/BIBM.2017.8217833.

[132] J. Feng and S. Lu, "Performance Analysis of Various Activation Functions in Artificial Neural Networks," *J. Phys. Conf. Ser.*, vol. 1237, no. 2, pp. 111–122, 2019, doi: 10.1088/1742-6596/1237/2/022030.

[133] S. C. Kwasny, "Why Tanh :," pp. 578–581, 1992.

[134] P. Iswarya, "Speech and Text Query based Tamil – English Cross Language Information

INTEGRI PROCEDAMUS

University of Ghana http://ugspace.ug.edu.gh

Retrieval System," *2014 Int. Conf. Comput. Commun. Informatics*, pp. 1–4, 2014, doi: 10.1109/ICCCI.2014.6921752.

[135] C. Chen, O. Li, C. Tao, A. J. Barnett, J. Su, and C. Rudin, "This Looks Like That: Deep Learning for Interpretable Image Recognition," no. NeurIPS, pp. 1–12, 2018, [Online]. Available: http://arxiv.org/abs/1806.10574

[136] Y. Shen *et al.*, "Deep learning with coherent nanophotonic circuits," *Nat. Photonics*, vol. 11, no. 7, pp. 441–446, 2017, doi: 10.1038/nphoton.2017.93.

[137] G. Koch and G. Koch, "Siamese Thesis," *Cs.Toronto.Edu*, vol. 2, 2015, [Online]. Available: http://www.cs.toronto.edu/~gkoch/files/msc-thesis.pdf

[138] B. Wu *et al.*, "An end-to-end deep learning approach to simultaneous speech dereverberation and acoustic modeling for robust speech recognition," *IEEE J. Sel. Top. Signal Process.*, vol. 11, no. 8, pp. 1289–1300, 2017, doi: 10.1109/JSTSP.2017.2756439.

[139] D. Yarotsky, "Error bounds for approximations with deep ReLU networks," *Neural Networks*, vol. 94, pp. 103–114, 2017, doi: 10.1016/j.neunet.2017.07.002.

[140] B. Zoph and Q. V Le, "Searching for activation functions," *6th Int. Conf. Learn. Represent. ICLR 2018 - Work. Track Proc.*, pp. 1–13, 2018.

[141] A. F. Agarap, "Deep Learning using Rectified Linear Units (ReLU)," no. 1, pp. 2–8, 2018, [Online]. Available: http://arxiv.org/abs/1803.08375

[142] Y. Li and Y. Yuan, "Convergence analysis of two-layer neural networks with RELU activation," *Adv. Neural Inf. Process. Syst.*, vol. 2017-Decem, no. Nips, pp. 598–608, 2017.

University of Ghana http://ugspace.ug.edu.gh

[143] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, "Activation Functions: Comparison of trends in Practice and Research for Deep Learning," pp. 1–20, 2018, [Online]. Available: http://arxiv.org/abs/1811.03378

[144] P. Murugan, "Feed Forward and Backward Run in Deep Convolution Neural Network," pp. 1–20, 2017, [Online]. Available: http://arxiv.org/abs/1711.03278

[145] Y. Sun, D. Liang, X. Wang, and X. Tang, "DeepID3: Face Recognition with Very Deep Neural Networks," pp. 2–6, 2015, [Online]. Available: http://arxiv.org/abs/1502.00873

[146] H. Shi, M. Xu, and R. Li, "Deep Learning for Household Load Forecasting-A Novel Pooling Deep RNN," *IEEE Trans. Smart Grid*, vol. 9, no. 5, pp. 5271–5280, 2018, doi: 10.1109/TSG.2017.2686012.

[147] B. Drayer and T. Brox, "Training deformable object models for human detection based on alignment and clustering," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 8693 LNCS, no. PART 5, pp. 406–420, 2014, doi: 10.1007/978-3-319-10602-1_27.

[148] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986, doi: 10.1038/323533a0.

[149] P. Kadebu and V. Thada, "Natural Language Processing and Deep Learning Towards Security Requirements Classification," *2018 3rd Int. Conf. Contemp. Comput. Informatics*, pp. 135–140, 2018.

[150] "D. h. m. m.," vol. 284, pp. 255–284, 1971.

University of Ghana http://ugspace.ug.edu.gh

[151] S. Thesis, P. D. Appl, and M. Harvard, "Beyond Regression : New Tools for Prediction and Analysis in the Behavioral," no. January 1974, 2018.

[152] J. C. Rajapakse and L. S. Ho, "Markov Encoding for Detecting Signals in Genomic Sequences," vol. 2, no. 2, pp. 131–142, 2005.

[153] S. Ben-david, *Understanding Machine Learning : From Theory to Algorithms*. 2014.

[154] S. Dietz, "Big Data Impacts on Stochastic Forecast Models : Evidence from FX Time Series," no. 3, pp. 277–291, 2013.

[155] F. Shaheen, B. Verma, and M. Asafuddoula, "Impact of Automatic Feature Extraction in Deep Learning Architecture," *2016 Int. Conf. Digit. Image Comput. Tech. Appl. DICTA 2016*, 2016, doi: 10.1109/DICTA.2016.7797053.

[156] T. Hinz, N. Navarro-Guerrero, S. Magg, and S. Wermter, "Speeding up the Hyperparameter Optimization of Deep Convolutional Neural Networks," *Int. J. Comput. Intell. Appl.*, vol. 17, no. 2, pp. 1–15, 2018, doi: 10.1142/S1469026818500086.

[157] D. Bera and P. V. Tharrmashastha, "Error reduction of quantum algorithms," *Phys. Rev. A*, vol. 100, no. 1, pp. 1–6, 2019, doi: 10.1103/PhysRevA.100.012331.

[158] J. Turian, J. Bergstra, and Y. Bengio, "Quadratic features and deep architectures for chunking," no. June, p. 245, 2009, doi: 10.3115/1620853.1620921.

[159] Y. Lecun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015, doi: 10.1038/nature14539.

[160] A. Cicuttin *et al.*, "A programmable System-on-Chip based digital pulse processing for high resolution X-ray spectroscopy," *2016 Int. Conf. Adv. Electr. Electron. Syst. Eng.*

University of Ghana http://ugspace.ug.edu.gh

ICAEES 2016, vol. 15, pp. 520–525, 2016, doi: 10.1109/ICAEES.2016.7888100.

[161] Y. Bengio, P. Simard, and P. Frasconi, "Learning Long-Term Dependencies with Gradient Descent is Difficult," *IEEE Trans. Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994, doi: 10.1109/72.279181.

[162] C. H. Sudre, W. Li, T. Vercauteren, S. Ourselin, and M. Jorge Cardoso, "Generalised dice overlap as a deep learning loss function for highly unbalanced segmentations," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 10553 LNCS, pp. 240–248, 2017, doi: 10.1007/978-3-319-67558-9_28.

[163] L. Bian and N. Gebraeel, "Stochastic Methodology for Prognostics under Continuously Varying," 2012, doi: 10.1002/sam.

[164] Y. Xu, J. Du, L. Dai, and C. Lee, "A Regression Approach to Speech Enhancement Based on Deep Neural Networks," vol. 23, no. 1, pp. 7–19, 2015.

[165] D. Guo, S. Shamai, and S. Verdú, "Mutual information and minimum mean-square error in Gaussian channels," *IEEE Trans. Inf. Theory*, vol. 51, no. 4, pp. 1261–1282, 2005, doi: 10.1109/TIT.2005.844072.

[166] T. Chai and R. R. Draxler, "Root mean square error (RMSE) or mean absolute error (MAE)? -Arguments against avoiding RMSE in the literature," *Geosci. Model Dev.*, vol. 7, no. 3, pp. 1247–1250, 2014, doi: 10.5194/gmd-7-1247-2014.

[167] E. J. Coyle and J. H. Lin, "Stack Filters and the Mean Absolute Error Criterion," *IEEE Trans. Acoust.*, vol. 36, no. 8, pp. 1244–1254, 1988, doi: 10.1109/29.1653.

[168] M. A. Blommer and G. H. Wakefield, "Pole-zero approximations for head-related transfer

University of Ghana http://ugspace.ug.edu.gh

functions using a logarithmic error criterion," *IEEE Trans. Speech Audio Process.*, vol. 5, no. 3, pp. 278–287, 1997, doi: 10.1109/89.568734.

[169] A. Guitton and W. W. Symes, "Robust inversion of seismic data using the Huber norm," *Geophysics*, vol. 68, no. 4, pp. 1310–1319, 2003, doi: 10.1190/1.1598124.

[170] K. Nar, O. Ocal, S. Shankar Sastry, and K. Ramchandran, "Cross-entropy loss and low-rank features have responsibility for adversarial examples," *arXiv*, 2019.

[171] J. S. Cardoso and R. Sousa, "Measuring the performance of ordinal classification," *Int. J. Pattern Recognit. Artif. Intell.*, vol. 25, no. 8, pp. 1173–1195, 2011, doi: 10.1142/S0218001411009093.

[172] Y. Wu and Y. Liu, "Robust truncated hinge loss support vector machines," *J. Am. Stat. Assoc.*, vol. 102, no. 479, pp. 974–983, 2007, doi: 10.1198/016214507000000617.

[173] C. Y. Lee, S. Xie, P. W. Gallagher, Z. Zhang, and Z. Tu, "Deeply-supervised nets," *J. Mach. Learn. Res.*, vol. 38, pp. 562–570, 2015.

[174] Y. Tang, "Deep Learning using Linear Support Vector Machines," 2013, [Online]. Available: http://arxiv.org/abs/1306.0239

[175] S. Eguchi and J. Copas, "Interpreting Kullback-Leibler divergence with the Neyman-Pearson lemma," *J. Multivar. Anal.*, vol. 97, no. 9, pp. 2034–2040, 2006, doi: 10.1016/j.jmva.2006.03.007.

[176] Atta-ur-Rahman and V. U. Ahmad, "Retinane," *13C-NMR Nat. Prod.*, pp. 30–33, 1992, doi: 10.1007/978-1-4615-3288-0_5.

[177] S. Tripathi and N. Hemachandra, "Scalable linear classifiers based on exponential loss

University of Ghana http://ugspace.ug.edu.gh

function," *ACM Int. Conf. Proceeding Ser.*, vol. 3152521, no. 1, pp. 190–200, 2018, doi: 10.1145/3152494.3152521.

[178] D. J. Fleet and M. Brubaker, "AdaBoost," pp. 123–129, 2015.

[179] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry, "How does batch normalization help optimization?," *Adv. Neural Inf. Process. Syst.*, vol. 2018-Decem, no. NeurIPS, pp. 2483–2493, 2018.

[180] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *32nd Int. Conf. Mach. Learn. ICML 2015*, vol. 1, pp. 448–456, 2015.

[181] J. G. Lee *et al.*, "Deep learning in medical imaging: General overview," *Korean J. Radiol.*, vol. 18, no. 4, pp. 570–584, 2017, doi: 10.3348/kjr.2017.18.4.570.

[182] D. L. K. Yamins, H. Hong, C. F. Cadieu, E. A. Solomon, D. Seibert, and J. J. DiCarlo, "Performance-optimized hierarchical models predict neural responses in higher visual cortex," *Proc. Natl. Acad. Sci. U. S. A.*, vol. 111, no. 23, pp. 8619–8624, 2014, doi: 10.1073/pnas.1403112111.

[183] G. Rostami, M. Hamid, and H. Jalaeikhoo, "Impact of the BCR-ABL1 fusion transcripts on different responses to Imatinib and disease recurrence in Iranian patients with Chronic Myeloid Leukemia," *Gene*, vol. 627, no. 1, pp. 202–206, 2017, doi: 10.1016/j.gene.2017.06.018.

[184] X. Xu, J. Zhou, H. Zhang, and X. Fu, "Chinese characters recognition from screen-rendered images using inception deep learning architecture," *Lect. Notes Comput. Sci.*

INTEGRI PROCEDAMUS

University of Ghana http://ugspace.ug.edu.gh

(including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 10735 LNCS, pp. 722–732, 2018, doi: 10.1007/978-3-319-77380-3_69.

[185] W. Yin, K. Kann, M. Yu, and H. Schütze, “Comparative Study of CNN and RNN for Natural Language Processing,” 2017, [Online]. Available: http://arxiv.org/abs/1702.01923

[186] J. A. Hawkins, S. K. Jones, I. J. Finkelstein, and W. H. Press, “Indel-correcting DNA barcodes for high- throughput sequencing,” 2018, doi: 10.1073/pnas.1802640115.

[187] F. Milletari, N. Navab, and S. A. Ahmadi, “V-Net: Fully convolutional neural networks for volumetric medical image segmentation,” *Proc. - 2016 4th Int. Conf. 3D Vision, 3DV 2016*, pp. 565–571, 2016, doi: 10.1109/3DV.2016.79.

[188] J. Bobo, C. Hudley, and C. Michel, “The Black studies reader,” *Black Stud. Read.*, pp. 1–488, 2004, doi: 10.4324/9780203491348.

[189] K. Zhou, Y. Qiao, and T. Xiang, “Deep Reinforcement Learning for Unsupervised Video Summarization With Diversity-Representativeness Reward,” *Proc. AAAI Conf. Artif. Intell.*, vol. 32, no. 1, Apr. 2018, Accessed: Aug. 27, 2021. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/12255

[190] E. S. Han and A. goleman, daniel; boyatzis, Richard; Mckee, “International Journal of Advanced Research in Artificial Intelligence,” *J. Chem. Inf. Model.*, vol. 53, no. 9, pp. 1689–1699, 2019.

[191] C. Szepesvári, “Algorithms for Reinforcement Learning,” *Synth. Lect. Artif. Intell. Mach. Learn.*, vol. 9, pp. 1–89, Jul. 2010, doi: 10.2200/S00268ED1V01Y201005AIM009.

[192] R. Al-rfou *et al.*, “Theano: A Python framework for fast computation of mathematical

University of Ghana http://ugspace.ug.edu.gh

expressions," pp. 1–19.

[193] S. Bahrampour, N. Ramakrishnan, L. Schott, and M. Shah, "Comparative Study of Deep Learning Software Frameworks".

[194] T. F. Gonzalez, "Handbook of approximation algorithms and metaheuristics," *Handb. Approx. Algorithms Metaheuristics*, pp. 1–1432, 2007, doi: 10.1201/9781420010749.

[195] D. Maclaurin, D. Duvenaud, and R. P. Adams, "Gradient-based Hyperparameter Optimization through Reversible Learning," vol. 37, 2015, [Online]. Available: http://arxiv.org/abs/1502.03492

[196] L. Yang and A. Shami, "On hyperparameter optimization of machine learning algorithms: Theory and practice," *Neurocomputing*, vol. 415, pp. 295–316, 2020, doi: 10.1016/j.neucom.2020.07.061.

[197] S. Sun, Z. Cao, H. Zhu, and J. Zhao, "A Survey of Optimization Methods from a Machine Learning Perspective," *IEEE Trans. Cybern.*, vol. 50, no. 8, pp. 3668–3681, 2020, doi: 10.1109/TCYB.2019.2950779.

[198] D. P. Kingma, D. J. Rezende, S. Mohamed, and M. Welling, "Semi-supervised learning with deep generative models," *Adv. Neural Inf. Process. Syst.*, vol. 4, no. January, pp. 3581–3589, 2014.

[199] X. Zhou and M. Belkin, *Academic Press Library in Signal Processing: Volume 1 - Signal Processing Theory and Machine Learning*, vol. 1. Elsevier Masson SAS, 2014. doi: 10.1016/B978-0-12-396502-8.00022-X.

[200] Z. Zhou and M. Li, "经典半监督co-training：COREG," *IJCAI Proc. 19th Int. Jt. Conf.*

University of Ghana http://ugspace.ug.edu.gh

Artif. Intell., pp. 908–913, 2005.

[201] J. E. van Engelen and H. H. Hoos, "A survey on semi-supervised learning," *Mach. Learn.*, vol. 109, no. 2, pp. 373–440, 2020, doi: 10.1007/s10994-019-05855-6.

[202] L. Rokach and O. Maimon, "Distance Measures for Numeric Attributes," *Data Min. Knowl. Discov. Handb.*, pp. 322–352, 2005.

[203] "Reinforcement learning: An introduction, 2nd ed. - PsycNET." https://psycnet.apa.org/record/2019-19679-000 (accessed Sep. 02, 2021).

[204] S. Ruder, "An overview of gradient descent optimization," pp. 1–14, 2016.

[205] D. R. Wilson and T. R. Martinez, "The general inefficiency of batch training for gradient descent learning," vol. 16, pp. 1429–1451, 2003, doi: 10.1016/S0893-6080(03)00138-2.

[206] M. Li, T. Zhang, Y. Chen, and A. J. Smola, "Efficient mini-batch training for stochastic optimization," *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Min.*, pp. 661–670, 2014, doi: 10.1145/2623330.2623612.

[207] S. Khirirat, H. R. Feyzmahdavian, and M. Johansson, "Mini-batch gradient descent: Faster convergence under data sparsity," *2017 IEEE 56th Annu. Conf. Decis. Control. CDC 2017*, vol. 2018-Janua, no. Cdc, pp. 2880–2887, 2018, doi: 10.1109/CDC.2017.8264077.

[208] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 6683 LNCS, pp. 507–523, 2011, doi: 10.1007/978-3-642-25566-3_40.

[209] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. De Freitas, "Taking the human

University of Ghana http://ugspace.ug.edu.gh

out of the loop: A review of Bayesian optimization," *Proc. IEEE*, vol. 104, no. 1, pp. 148–175, 2016, doi: 10.1109/JPROC.2015.2494218.

[210] M. Li, Y. I. Liu, X. Liu, Q. Sun, and X. I. N. You, "The Deep Learning Compiler : A Comprehensive Survey," vol. 1, no. 1, 2020.

[211] Z. Wang, K. Liu, J. Li, Y. Zhu, and Y. Zhang, "Various Frameworks and Libraries of Machine Learning and Deep Learning: A Survey," *Arch. Comput. Methods Eng.*, no. 0123456789, 2019, doi: 10.1007/s11831-018-09312-w.

[212] S. Shi, Q. Wang, P. Xu, and X. Chu, "Benchmarking State-of-the-Art Deep Learning Software Tools," pp. 111–116, 2016, doi: 10.1109/CCBD.2016.33.

[213] S. Chetlur *et al.*, "cuDNN: Efficient Primitives for Deep Learning," pp. 1–9, 2014, [Online]. Available: http://arxiv.org/abs/1410.0759

[214] F. Florencio, T. Valença, E. D. Moreno, and M. Colaço Junior, "Performance analysis of deep learning libraries: Tensor flow and PyTorch," *J. Comput. Sci.*, vol. 15, no. 6, pp. 785–799, 2019, doi: 10.3844/jcssp.2019.785.799.

[215] H. Zhang *et al.*, "Poseidon: An efficient communication architecture for distributed deep learning on GPU clusters," *Proc. 2017 USENIX Annu. Tech. Conf. USENIX ATC 2017*, pp. 181–193, 2019.

[216] S. Shi, Q. Wang, X. Chu, and B. Li, "A DAG Model of Synchronous Stochastic Gradient Descent in Distributed Deep Learning," *Proc. Int. Conf. Parallel Distrib. Syst. - ICPADS*, vol. 2018-Decem, pp. 425–432, 2019, doi: 10.1109/PADSW.2018.8644932.

[217] C. Lin, "Performance Enhancement of GPU Parallel Computing Using Memory

University of Ghana http://ugspace.ug.edu.gh

Allocation Optimization," pp. 1–5, 2020.

[218] A. P. Singh and D. P. Singh, "Implementation of K-shortest path algorithm in GPU using CUDA," *Procedia - Procedia Comput. Sci.*, vol. 48, no. Iccc, pp. 5–13, 2015, doi: 10.1016/j.procs.2015.04.103.

[219] Y. Zhou, Y. Tan, and S. Member, "GPU-based Parallel Particle Swarm Optimization," no. 1, pp. 1493–1500, 2009.

[220] M. O. Okwu and L. K. Tartibu, "Particle Swarm Optimisation," *Stud. Comput. Intell.*, vol. 927, pp. 5–13, 2021, doi: 10.1007/978-3-030-61111-8_2.

[221] F. E. Fernandes and G. G. Yen, "Pruning Deep Convolutional Neural Networks Architectures with Evolution Strategy," *Inf. Sci. (Ny).*, vol. 552, pp. 29–47, 2021, doi: 10.1016/j.ins.2020.11.009.

[222] A. Ghaffari and Y. Savaria, "CNN2Gate: Toward Designing a General Framework for Implementation of Convolutional Neural Networks on FPGA," pp. 1–15, 2020, [Online]. Available: http://arxiv.org/abs/2004.04641