A Deep Learning-Based Data Minimization Algorithm for Fast and Secure Transfer of Big Genomic Datasets

Mohammed Aledhari, *Member, IEEE,* Marianne Di Pierro, Mohamed Hefeida, *Member, IEEE,* and Fahad Saeed, *Senior Member, IEEE*

***Abstract*—In the age of Big Genomics Data, institutions such as the National Human Genome Research Institute (NHGRI) are challenged in their efforts to share volumes of data between researchers, a process that has been plagued by unreliable transfers and slow speeds. These occur due to throughput bottlenecks of traditional transfer technologies. Two factors that affect the efficiency of data transmission are the channel bandwidth and the amount of data. Increasing the bandwidth is one way to transmit date efficiently, but might not always be possible due to resource limitations. Another way to maximize channel utilization is by decreasing the bits needed for transmission of a dataset. Traditionally, transmission of big genomic data between two geographical locations is done using general-purpose protocols, such as hypertext transfer protocol (HTTP) and file transfer protocol (FTP) secure. In this paper, we present a novel deep learning-based data minimization algorithm that 1) minimizes the datasets during transfer over the carrier channels; 2) protects the data from the man-in-the-middle (MITM) and other attacks by changing the binary representation (content-encoding) several times for the same dataset: we assign different codewords to the same character in different parts of the dataset. Our data minimization strategy exploits the alphabet limitation of DNA sequences and modifies the binary representation (codeword) of dataset characters using deep learning-based convolutional neural network (CNN) to ensure a minimum of code word uses to the high frequency characters at different time slots during the transfer time. This algorithm ensures transmission of big genomic DNA datasets with minimal bits and latency and yields an efficient and expedient process. Our tested heuristic model, simulation, and real implementation results indicate that the proposed data minimization algorithm is up to 99 times faster and more secure than the currently used content-encoding scheme used in HTTP of the HTTP content-encoding scheme and 96 times faster than FTP on tested datasets. The developed protocol in C# will be available to the wider genomics community and domain scientists.**

***Index Terms*—Machine Learning, Deep Learning, Convolutional Neural Networks, DNA, Big Genomic Data, Big Data, Content-Encoding, Transfer Protocols, HTTP, Wireless Communication, Variable-Length Binary Encoding.**

I. Introduction

DNA sequencing is needed in the most critical areas such as criminal investigations, genotyping and determination of disease-relevant genes or agents causing diseases, mutation analysis, screening of single nucleotide polymorphisms (SNPs), detection of chromosome abnormalities [1], and to identify disease- and/or drug-associated genetic variants to advance precision medicine [2] [3]. Also, the use of high-throughput DNA sequencing instruments, such as next-generation sequencing (NGS) technologies that include whole-genome sequencing (WGS) and whole-exome sequencing (WES), significantly decreases the sequencing costs and enables the genomic datasets to join the big data club. Those instruments became big data generators, not only for big biology centers, but also for small biology laboratories and researchers.

The current major big data generators are Astronomy, YouTube, and Twitter. For example, the Australian Square Kilometer Array Pathfinder (ASKAP), an astronomy project, currently generates about 7.5 terabytes/second of image data, while the expectation is to reach to 750 terabytes/second (~25 zettabytes per year) by 2025 [4] [5]. Also, YouTube uploads about 300 hours of videos per minute, while 1,700 hours of videos per minute are expected to be uploaded by 2025 (1 - 2 exabytes of video data per year). There are approximately 500 million tweets/day with an anticipation of about 3 kilobytes each, while the expectation is to reach 1.2 billion tweets per day: that is 1.36 petabytes/year by 2025. However, a big data generator, currently in progress, will exceed 35 petabases per year [6]. This capacity is surpassed only by a reduction in sequencing costs. For example, less than 10 years ago, the cost of sequencing genomes was approximately one million dollars and has spiraled down to several hundred dollars, while simultaneously scientists can map genomes at substantially increased rates. However, despite the advantages of enhanced speed and reduced costs, growing the genomic datasets brought challenges such as storing, handling, analyzing, visualizing, sharing, and transferring the genomic information generated by NGS technologies and that need to be addressed. For instance, sequencing a single whole genome generates more than 250 gigabytes of data since there are over 3 billion base pairs (sites) on a human genome as calculated:

1) Digitalizing a single cell would result in :
   - Single Unit of all DNA (A - C - G - T) - 1 bp.
   - Average Length of exon sequences for one protein-coding gene - (1100 bp).
   - Approximate total length of all exons of protein-

* Correspondence should be addressed to Mohammed Aledhari at mohammed.aledhari@wmich.edu

coding genes in the human genome (20000 genes - 22000000 bp).

- Approximate total length of all coding and non-coding DNA in haploid genome (3000000000 bp).
- Approximate length of diploid human genome. That is all DNA within a single nucleus (6000000000 bp).
- Approximate number of bp generated in a shotgun sequence using Next Generation Sequencing methods assuming 30x coverage (180000000000 bp).

2) Digitalizing of the world's population genomes (world's population on June 2016 [7]) would result in :
Approximate number of bp that would be generated by sequencing diploid genomes for all 7 billion people on earth today (1250000000000 bp).

Hint: 1kilo = 1024 used in these calculations, hence 1 Yotta = 270 KB.
In fact, the growth rate of DNA sequencing over the last 10 years has generated a massive amount of data that doubles approximately every 7 months. According to [8] to date as shown in Table I on page 3, there are more than 2,500 high-throughput sequencing instruments distributed over 55 countries placed in about 1,000 sequencing centers.

A. Research Problem

Although low-cost, high-throughput instruments and cloud-based services have solved big data generating and processing challenges to certain extents, they do not efficiently solve data transfer speed and security problems witnessed in , transferring big data between two or more places (e.g. between two biology laboratories or between lab-cloud-lab), which still results in a bottleneck due to the use of traditional transfer protocols such as HTTP [9] and FTP [10]. Recently, biologists ascertained that the bottleneck results in an inability to share and transfer large datasets in a timely manner. Therefore, some projects have begun to navigate potential solutions to access big data and share them with researchers worldwide. The possible solutions are data minimization during transfer over the networks [11] [12] or expansion of the network bandwidth [13]. For example, the Human Genome Project [14] and the HapMap project [15] facilitates sharing the sequence data and the more recent data-sharing structures for genome-wide association studies (GWAS) [16], such as dbGaP [17] and the European Genotyping Archive [18].

In addition, grantors and other funding agencies make data sharing a requirement of support for all projects, including all hypothesis-driven projects, whose primary purpose is to focus on a specific research question rather than to create data to be used by others. Implementing tools and techniques for accessing genomic datasets accelerates studies of the biological mechanisms of diseases and their treatment. Individual researchers can no longer download and analyze important datasets in their scientific fields on their own computers, a fact that inhibits access to critical information. This research posits a solution.

B. Purpose of the Study

We designed a data minimization algorithm to transfer big genomic datasets in an expedient, secure way to allow scientists to share their data and analyses. We used the HTTP as a baseline protocol [19] to compare and assess implementation results of transferring big genomic datasets. The goals of our data minimization algorithm are as follows: 1) reduce the size of data to be transferred between a server and a client [20]; 2) secure and protect the privacy of the data from unauthorized access due to attacks or data breach, such as MITM attack. Our heuristic model, simulation, and implementation results proved that our data minimization algorithm reduces significant amounts of data and makes more efficient use of network bandwidth, while also protecting the data by preventing unauthorized individuals from accessing them.

This paper represents an extension of our previous research papers in [21], [22], and [23], in which we changed the character codeword several times during transfer of a single dataset (file), a change that minimizes data transfers, thus shortening the overall transfer time of the genomic dataset and increasing data security. To the best of our knowledge, this is the first data minimization technique that reduces and secures datasets during data transfer via changing binary representations of data characters many times for the same file. Our algorithm provides improvements in response and transfer time of genomic datasets, as well as prevents unauthorized individuals from accessing the file contents in the event of a data breach because we assign different codewords to the same character of the dataset in different times and file parts based on data obtained in running the convolutional neural network (CNN) We also illustrate the added benefit of using the deep learning technique of random sampling to form a renewable content-encoding in different times and file parts, an outcome that yields optimal results in transfer data size, time, and security. Our approach is compatible with all existing browser implementations and specifications, such as Google Chrome [24], Safari [25], Internet Explorer [26], etc. The overall benefit of this work is to increase opportunities for data sharing among researchers to advance the dissemination of scientific knowledge.

C. Audience

The audience for this work is researchers (biologists), investigators, and clinicians as shown in Figure **??** on page **??**. Investigators use DNA sequencing to combat crimes by understanding finger printings and genetic clues in a crime scene: the use of gel electrophoresis to relate sperm DNA to potential suspects. Investigators utilize DNA sequencing to understand ethnicity and ancestry by identifying ethnic characteristics of a certain country via specific patterns or genes. DNA sequencing enables biologists and clinicians to investigate various diseases and genetic illnesses. Scientists can gain access to epidemiological data with multiple genomic candidates, and

via genomic sequencing (in clinical trials), provide critical information in the evolution of medical treatment.

D. Contribution

We have created a data minimization mechanism for big genomic datasets during real-time data transfer using a deep learning-based algorithm, as illustrated in section V on page 7. Creating data minimization mechanisms to be equipped to transfer protocols, such as HTTP and FTP, solves big data transmission challenges, in transfer time and data security [27]. Our proposed data minimization mechanism for the transfer protocols enables them to be smart protocols via using standard codewords for dataset headers, while using our data minimization mechanism for the dataset body. We test our data minimization algorithm by using three different transfer protocols: HTTP, FTP and BitTorrent [28] and by considering such variables as versatility, security and flexibility. These are commonly used protocols that transfer different data types in a variety of browsers, such as Google Chrome. These protocols have certain security features in the transport layer because they run on top of TCP [29] and are flexible because they are equipped with the ability to modify one or more components, such as content-encoding schemes, compression algorithms, and message headers. This paper is a continuation of our works published in [22] [23]. We extend this previous work by employing the convolutional neural network to update the encoding codewords periodically and to ensure the assignment of the minimum binary representation to the most repetitive characters in the file.

E. Motivation

The generation of big data sets led to other challenges such as data storage, process, and share. Many efforts have been made to solve the challenges of big data storage, and manipulation, including data analysis and visualization. These challenges still constitute a major challenge that must be addressed and resolved. In addition, issues of data minimization and transfer time, as well as accuracy, speed, and security still loomed on the research horizon. Big genomic datasets are part of the big data club that require special handling from generating and processing to transferring between two or more biology laboratories. Although many solutions have been developed to address the challenges of generating and analyzing big data, transmission challenges have not been addressed at the same level. As a result, scientists are motivated to navigate and discover new mechanisms to transfer big genomic datasets more efficiently in terms of transfer time and security. Current content-encoding algorithms for transfer protocols use the standard encoding scheme [30], which increases the size and the transfer time, and which are not suitable for use with big genomic datasets. By observing these limitations, we take advantage of the nature of the genomic dataset alphabet to reduce the size of data being transferred, as well as to reduce the transfer time, and enhance security. Access to these data, in the pursuit of scientific advances, is the end goal.

Table I: Four domains of Big Data in 2025

Data Lifecycle	Big Data Main Domains			
	Astronomy	Twitter	YouTube	Genomics
Acquisition	25 zetta bytes/year	0.5 -15 billion tweets/year	500 - 900 million hours/year	1 zetta bases/year
Storage	1 EB/year	1 - 17 PB/year	1 - 2 EB/year	2 - 40 EB/year
Analysis	In situ data reduction Real-time processing Massive volumes	Topic and sentiment mining Metadata analysis	Limited requirements	Heterogeneous data and analysis Variant calling, 2 trillion central processing unit (CPU) hours All pairs genome alignments 10,000 trillion CPU hours
Distribution	Dedicated lines from antennae to server (600 TB/s)	Small units of distribution	Major component of modern users bandwidth (10 MB/s)	Many small (10 MB/s) and fewer massive (10 TB/s) data movement

F. Paper Goals and Organization

The purpose of this paper is to develop and implement transfer protocols equipped with a novel data minimization algorithm for big genomic datasets that aim to share the data in less time and with more security. Moreover, these protocols will introduce a generic concept that can be modified to transfer securely minimum datasets that have limited symbols by using CNN-based algorithm content-encoding schemes. The implications of this paper are outlined as follows:

- Summarizes the standard and common use of content-encoding schemes that are currently employed in transfer protocols and their relevant standards to provide researchers with quick fundamentals, without having to search through the details presented in the standards' specifications.
- Provides an overview of some of the big genomic data challenges in terms of transmission and transfer time.
- Explores the relationship between big data and binary representation methods involving various binary encoding mechanisms.
- Presents the need for better transfer protocols equipped with data minimization algorithms to transfer datasets securely in shorter times, especially genomic datasets, and then to provide better services for big data demands.
- Implements, tests, and evaluates the proposed data minimization algorithm of transfer protocols in terms of transfer size, time, and security.

The remainder of this paper is organized as follows: Section II provides a summary of the related works that are used as a baseline for our implementations. Section III presents techniques used in this work. Section IV discusses the proposed method of data minimization algorithm, and heuristic model's description in section V. Section VI presents the experiments, results, and evaluations of the proposed data minimization mechanism. Finally, our conclusion is presented in Section VII.

II. Related Work

To the best of our knowledge, this work is the first network-based data minimization solution for big genomic datasets that utilizes data-encoding as a mechanism. GeneTorrent [31] is a file transfer protocol which uses the BitTorrent [32] technique to transfer genomic datasets, and which was originally designed to support distributed peer-to-peer (P2P) file transfer applications. In other words, GeneTorrent distributes the same file(s) on different machines settled in different locations and configures those machines to transfer certain part(s) of that file(s) to a requester. Although higher throughput can be

achieved by using multiple machines for transferring data, the underlying data are still transferred using general-purpose protocols. This protocol is no longer in use, and there is a need to create a data-aware network transfer protocol for the DNA genomic datasets that use minimum resources of the network to deliver data efficiently.

The possible network solutions to transfer big genomic datasets expediently are listed as follows:

1) Enhance bandwidth utilization by developing new solutions and techniques for:
   - *Flow Control* [33]: An operation that balances the rate at which bits are generated by the sender with the rate at which bits are received by the receiver. This matches the speed of a sender with the capabilities of a receiver.
   - *Congestion Control* [34]: An operation that regulates the rate at which senders generate traffic in order to avoid the over-utilization of the resources available within network. This prevents network congestion which if pronounced, could lead to a network collapse.
2) Maximize the bandwidth by expanding the network in terms of physical (hardware) resources as such internet2 project [35] that provides high speed internet connection.
3) Minimize the datasets that can be achieved by developing new techniques and solutions for:
   - Encoding schemes that deal with character codewords or binary representations, the focus and scope of this paper.
   - Compression techniques that are out of this work scope. However, we list briefly the three different lossless compression algorithms: compress [36], deflate [37], and GZIP that can be used for HTTP as a preprocessing operation that requires additional time and continued research. Also, it would be worthy to mention here that there are some efficient compression algorithms that provide higher compression ratios, such as BZIP2 [38] and MFCompress, and that are genomic-specific compared to those that can be used in HTTP as preprocessing functions. However, the higher compression ratio algorithm is not the best choice when considering the compression time and security aspects of browsers. This tradeoff between compression ratio and compression time requires additional attention when dealing with time-sensitive applications, as shown later in the results. Also, some compression algorithms might suffer from security issues, such as intermediate proxies of the Chromium browser, which corrupts the data when trying to use BZIP2. Therefore, we introduce a new content-encoding that works best for all browsers, without adding more time or affecting security, such as those attached to compression algorithms. This work utilizes GZIP and MFCompress as benchmarks to compare with our encoding scheme over HTTP.

We can summarize some differences between the two data minimization techniques: encoding and compression, and then discuss why we decided to utilize data-encoding as a core of this work. These explanations are as follows:

1) Compression techniques cannot be implemented on the network during transfer phase, work in static environments such as workstations, PCs, and any non-transferable environments, which need network solutions.
2) Compression algorithms provide better performances in terms of data minimization but require longer time due to computation costs, which we want to avoid.
3) Compression algorithms use the same codeword for the entire dataset characters rather than all datasets, while we are able to change characters' codewords several times for each dataset to add extra security level via a proposed data-encoding scheme.
4) Not all compression techniques supported by network browsers such as Firefox, Edge, Safari, and etc.
5) Finally, the data-encoding techniques can provide similar performances to compression techniques in shorter transfer times and in more secure ways.

We put forward a novel network-based data minimization solution using CNN to transfer big genomic datasets expediently and securely, thereby enabling more scientists to share and analyze datasets.

III. Techniques Used in the Study

Development and implementation of a new data minimization method, specified to the real-time big genomic datasets transfers, requires an understanding of CNN algorithm and data-encoding techniques. The next subsections provide a brief description about these techniques.

A. Data-Encoding Approaches

Data minimization can be divided into two main forms: data encoding and data compression. Data minimization using data encoding assigns the lowest possible bits to each alphabet's symbol using content-encoding schemes without complex computations, whereas the data minimization using data compression assigns the lowest possible bits to the entire dataset: this process involves complex computations and an extended period of time to compress and decompress operations. In general, binary representation can be divided into two categories: Fixed-Length Binary Encoding (FLBE) and Variable-Length Binary Encoding (VLBE). FLBE scheme, also called singular encoding, converts symbols into a fixed number of output bits, such as in an ASCII code which consists of an 8-bit long for each codeword [39]. Variable-length binary encoding (VLBE), also referred to as a uniquely decodable and non-singular code, converts symbols into variable-length codewords, such that $\lambda_i \neq \lambda_j$ for all *i and j* [40]. However, the scope of this paper is data minimization using data-encoding techniques that are divided into five main mechanisms as follows:

B. Naive Bit Encoding

This approach works by assigning fixed-length codeword/binary representation to each alphabet symbol in a way that represents more than a single symbol in a single byte, such as 2-bit length to genomic symbols [41] and [42], as shown in Figure 1-(a) on page 5.

C. Dictionary-based/Substitutional Encoding

This approach stores different patterns of the input symbols in a dictionary or a database, along with their codewords, and then replaces the new input parts with predefined portions [43] and [44], such as in 1977-78 Ziv and Lempel (LZ-77) [45] and [46], as shown in Figure 1-(b) on page 5. LZ-77 algorithm works by replacing the repeated occurrences of symbols with their references that indicate length and location of that string, which occurred before, and which can be presented in the tuple (offset, length, symbol).

D. Statistical/Entropy Encoding

This approach works by statistics, prediction, and a probabilistic model from the input [47] and [48], such as Huffman's coding [45] and [49], as shown in Figure 1-(c) on page 5. Huffman's coding, introduced in 1952, is a statistical method that assigns a fixed-length codeword/binary representation to alphabet symbols, such as 2-bit, 3-bit, 8-bit, etc. The codewords will have different lengths, and the lowest frequency symbols will be assigned with the longest codewords and vice versa. This research utilizes this type of encoding with CNN deep learning algorithm to ensure the assignment of the lowest possible codewords to the more frequent dataset characters, and to undertake this process several times during the data transfer phase.

E. Referential/Reference-based Encoding

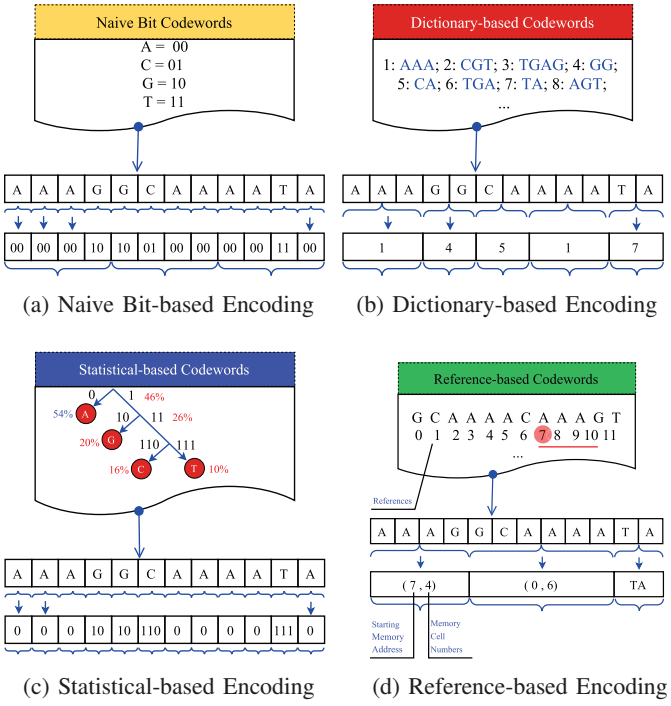
This approach is similar to a dictionary-based technique, except that it uses the pointer to the internal and external references, as shown in Figure 1-(d) on page 5.

F. Hybrid Encoding

This approach works by combining two or more encoding methods. For example, The Burrows-Wheeler transform (BWT) [50] [45] and [51], is one of the hybrid encoding methods, especially popular in bioinformatics, used for data minimization. The BWT method works by permuting the input sequence in a way that symbols are grouped by their neighborhood. Our proposed data minimization algorithm can be classified as a hybrid encoding method by incorporating elements of the standard (8-bit) and the statistical encoding methods (variation of 1 - 3 bits.

IV. Proposed Method

In this section, we illustrate our two implementation versions of the content-encoding schemes: standard (FLBE) and proposed (dynamic VLBE/DVLBE). Also, we discuss model formulations in this section for different possible scenarios of symbol repetitions that verify how our proposed encoding scheme compares to the current transfer protocols i.e. HTTP and FTP content-encoding scheme. The encoding model description and formulation are presented in the following subsections:



(a) Naive Bit-based Encoding (b) Dictionary-based Encoding

(c) Statistical-based Encoding (d) Reference-based Encoding

Figure 1: Data-encoding methods

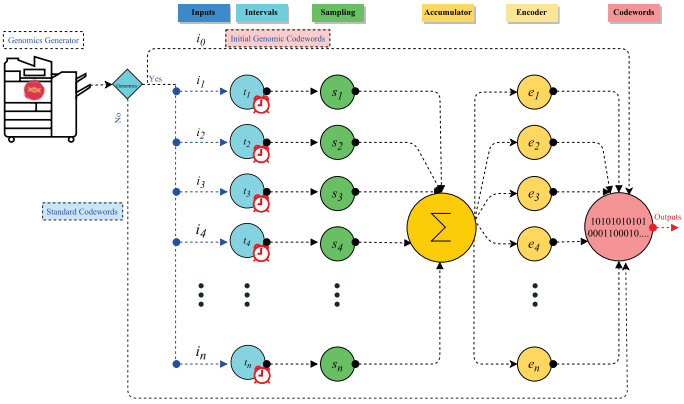


Figure 2: The Proposed Encoding System Framework

A. Convolutional Neural Network

Convolutional Neural Network (CNN) [52] [53] is an example of deep learning (DL) [54] techniques that refer to both deep neural networks and other branches of machine learning, such as deep reinforcement learning. Neural networks are defined as a set of algorithms, modeled loosely after the human brain, and that are designed to recognize patterns. They interpret sensory data through a kind of machine perception, labeling or by clustering raw input. The patterns they recognize are numerical, contained in vectors, into which all real-world data, be it images, sound, text or time series, must be translated. Thus, CNN is defined as an end-to-end system, in which the input is raw data, while the output is a prediction through the distinctive features extracted via intermediate layers. CNN divides into four main layers: convolution, pooling, normalization, and fully connected, as illustrated in Figure 3 on page 6 as follows:

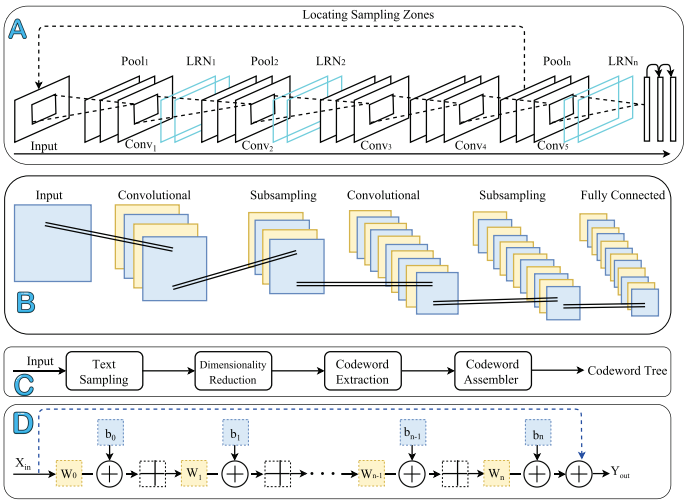


Figure 3: Convolutional Neural Network Conceptual Model.

1) Convolution Layers: Convolution layers are used to convolve previous layer's feature maps with multiple filter masks to find the most common patterns of tested data. This layer is responsible for running two important jobs: connection and sharing. Connection applies each single convolve filter only to a local region of the input volume and thus, decreases the network weight parameters. Thus, the spatial extent of the local connectivity is sometimes referred to as the receptive field. Sharing the network parameters between layers means it is necessary to use the same filter to convolve the entire feature map at each layer. To formulate the convolution layers, we denote Con_i^l as the i^{th} input feature map of l layer; Ker_{ij}^l refers to connecting the kernels of the feature map of the output layer j^{th} to the feature map of the input layer i^{th} and $bias_j^l$, as an additive bias.; and then the following:

$$Con_j^l = f(\sum_{i=1}^{n} Con_i^{l-1} Ker_{ij}^l + bias_j^l), \tag{1}$$

where f denotes the activation function that is usually a rectified linear function.

2) Pooling Layers: Pooling layers represent the sampling layers that work on combining the outputs of the convolution layers and the related classical spatial pyramid [55]. These layers reduce the spatial size of the feature maps, thus decreasing computation costs in the network. To formulate the pooling layers, we denote as:

$$Con_j^l = next(Con_j^{l-1}), \tag{2}$$

where next(Con_j) is a subsampling function of the next layer.

3) Normalization Layers: Normalization layers are responsible for assembling the output of different layers and provide the best pattern recognition i.e. the most repetition characters. Denoting by a_i the single value of i^{th} feature map, the normalize activity $bias_i$ is given by the expression

$$bias_i = \frac{a_i}{(Ker + \alpha \sum_{j=max(0,1-n/2)}^{min(N-1,i+n/2)} a_j^2)^\beta}. \tag{3}$$

The constants Ker, n, α, and β are hyperparameters, and we use $Ker = 2$, $n = 5$, $\alpha = 10^{-4}$, and $\beta = 0.75$ in our experiments.

4) Fully Connected Layers: The final cycles of normalization produce several fully connected layers that draw the final forms of available data patterns (classifiers). The final layer consists of a combination of the outputs of fully connected layers which generate the new character-codeword tree. The output of character-codeword tree represents the probabilities of the character repetitions, in which the highest one corresponds to the predicted codeword. Then, we can formulate the fully-connected layer,

$$P(y = 1|Con; w) = \frac{1}{1 + \exp(-w^T Con)}, \tag{4}$$

where y is label, $x \in R^{(D+1)1}$ represents the D dimensional feature vector, $w \in R^{(D+1)1}$ represents the weight vector, T refers to training data. Considering a classification problem where the response variable y can take any one of N values, we can generalize the binary classification (genomic or non-genomic characters). Thus,

$$P(y = c|Con; W) = \frac{\exp(w_i^T Con)}{\sum_{i=1}^{n} \exp(w_i^T Con)}, \tag{5}$$

where $W = [w_1, w_2, ..., w_n] \in R^{(D+1)N}$, each w represents the corresponding category weight parameters.

This subsection presents our implementation of network data minimization solution that relies on the statistical encoding and CNN algorithm via modifying HTTP content-encoding to transfer big genomic datasets expediently and securely. This work assures better performance and bandwidth utilization for the transfer of big genomic datasets via the minimization of data size and time. This model assigns the shortest possible codeword for more symbol occurrences via utilizing a CNN algorithm, as illustrated in section V on page 7, a codeword that divides a dataset into parts and that reads a short string randomly to specify symbol repetitions. Two encoding schemes are used in this model: standard (8-bit) and modified (1,2, and 3-bit). The standard one uses the title (header) of a dataset and other symbols out of the genomic scope i.e. N in the body. The modified encoding scheme uses a convolutional neural network technique for the body of datasets. The proposed encoding scheme starts with initial codewords such those shown in Table II on page 7 and then periodically updates codewords via a CNN deep learning algorithm, as illustrated in section V on page 7, while using the former codeword table. After each real-time update, the server send the proposed encoding scheme to the client (receiver) prior to applying it on datasets at the server side. Therefore, this model encodes the contents using two main codeword tables: fixed (static) and dynamic, as shown in Figure 2 on page 5.

The fact that the genomic DNA alphabet consists of only four symbols inspired us to build an adaptive encoding scheme to speed up the transfer time. This implementation combines a VLBE scheme and dynamic behavior via applying a CNN deep learning algorithm to guarantee producing the best updatable VLBE scheme for each single genomic dataset at each transfer session. Producing an updatable VLBE scheme assures getting the minimum possible data that facilitates the minimum transfer time. The genomic alphabet (A) is

Table II: The standard and proposed codewords

Symbol	Frequency	Proposed	Standard
A	0.85	0	01000001
T	0.05	11	01010100
G	0.05	100	01000111
C	0.05	101	01000011
Total bits for 20 symbols	100%	25	160

comprised of four symbols: {A, T, G, C}, which can be presented in less than an 8-bit codewords length as used in the current implementation. We can encode the genomic dataset symbols in four unique decipherable codewords i.e. [0, 11, 100, 101] or simply [0, 3, 4, 5], as shown in Table II on page 7.

We utilize a binary tree as a structure to represent our proposed encoding scheme because it is faster to search, avoids duplicate values, and facilitates decoding at the receiver side. Also, the use of a binary tree as a structure gives the programmer special flexibility because it offers one of the two paths to follow and cuts the search time to half, thereby increasing process throughput. A simple example is listed below to theoretically assess our binary encoding, in contrast to the current use of HTTP binary encoding for a 20 genomic symbols string based on Table II on page 7. Therefore, encoding 20 symbols in 25 bits yields an average of 1.25 bits/symbol in the proposed example, whereas 160 bits in the current HTTP encoding yields an average of 8 bits/symbol. We designed variable codes in Table II on page 7 in a way that facilitates decoding, using a prefix property (unambiguously). The prefix property assigns a unique specific bit pattern for each alphabet symbol to ease the decoding operation on the client side. Variable binary encoding is not a new idea; however, it is more common in single or static applications. The use of the proposed encoding scheme requires a prior knowledge to assign short codes for high probabilities; otherwise, short codes would be assigned for rare occurrences (negative results). The current use of HTTP content-encoding is fixed (standard), which means assigning the same weight for each symbol i.e. 8-bits. Therefore, we design a proposed encoding scheme for HTTP in a way that always enables the server to assign short codes for letters that appear more frequently. Although this scheme produces minimum possible bits, it consumes extra time. Our implementation relies on reading parts of the file via the CNN algorithm to estimate symbol frequencies and to set codes. Consequently, we can get minimum possible codes in less time via applying the CNN algorithm to the proposed scheme. Time complexity is *O(n)*. This encoding approach works well with high symbol repetition occurrences, so that assigning a 1-bit length codeword for the highest occurrence symbol, a 2-bit length codeword for less repetition, and a 3-bit codeword length for the remaining 2 symbols is an optimal approach. The pseudocode for our protocol is highlighted in algorithm 1 on page 7. In addition, we implemented a variable-length binary encoding (arbitrary or bind) to compare with our proposed and standard.

Algorithm 1 Dynamic Variable-length binary encoding

```
1:  procedure ENCODING
2:      DVLBE.doSamplingAndBuildFreqArray(inputStream , filePartitions,SamplingRatioPerPartition)
3:      if inputStream.hasGenomeFileheader then
4:          outputStream.write(GenomeFileheader)
5:      end if
6:      DVLBE.writeGenomeSymbolsEncoder(outputStream)
7:      while !inputStream.EOF do
8:          genomeChar ← inputStream.GetChar().
9:          code ← DVLBE.encode(genomeChar).
10:         oneByteStore.store(code).
11:         if oneByteStore.ISFull() then
12:             outputStream.write(oneByteStore).
13:             oneByteStore.empty().
14:         end if
15:     end while
16:     if !oneByteStore.ISEmpty() then
17:         outputStream.write(oneByteStore).
18:         outputStream.write(NumOfExtraBits).
19:     end if
20: end procedure
1:  function DOSAMPLINGANDBUILDFREQARRAY(INPUTSTREAM , FILEPARTITIONS,SAMPLINGRATIOPERPARTITION)
2:      PartitionSize ← FileSize / filePartitions().
3:      SamplesPerPartition ← PartitionSize * SamplingRatioPerPartition.
4:      SymbolsFrequencyArray ← Interger Array with Four elements filled with zeros.
5:      while filePartitions >0 do
6:          Offest ← Random.GetDouble * PartitionSize . ▷ Random.GetDouble generates random numbers between 0 and 1
7:          inputStream.Seek(offest).
8:          inputStream.Read(SamplesPerPartition, dataBuffer).
9:          UpdateFrequency(SymbolsFrequencyArray, ← dataBuffer).
10:         filePartitions ← filePartitions - 1.
11:     end while
12:     FreqArray.Build(SymbolsFrequencyArray). return FreqTable
13: end function
```

V. HEURISTIC MODEL

In this paper, a novel data minimization method is proposed that significantly reduces data transfer size and time, as illustrated in Section VI. Two data-encoding schemes are used in this work: standard and proposed. The proposed data-encoding scheme is created via codeword generators $\mathscr{E}$ that run a deep learning CNN algorithm during the data transfer process. The codeword (encoding) generator $\mathscr{E}(\mathscr{E}_l, m, f_i, a_c, a_l, n)$ consists of two main processes: sampling $\mathscr{E}$ and encoding improvement function f_i, where $\mathscr{E}_l$ is the last symbol in the encoding codewords. The input symbols m represents the last generated array of symbol codewords. The current array of symbol codewords is represented by a_l, a_c and output bits n.

Table III: Datasets used in our experiments

IDs	Source	Size(KB)	Renamed
pataa	National Center for Biotechnology Information	563,318	1
refGeneexonNuc	University of California Santa Cruz	639,183	2
envnr	National Center for Biotechnology Information	1,952,531	3
hg38	University of California Santa Cruz	11,135,899	4
patnt	National Center for Biotechnology Information	14,807,918	5
gss	National Center for Biotechnology Information	30,526,525	6
estothers	National Center for Biotechnology Information	43,632,488	7
humangenomic	National Center for Biotechnology Information	45,323,884	8
othergenomic	National Center for Biotechnology Information	346,387,292	9

The following two theorems illustrate the proposed scheme of data-encoding: the codeword generator discussed in theorem 5.1 and sampling improvement function discussed in theorem 5.2. Proofs of 5.1 and 5.2 theorems can be found on the supplementary pages.

Theorem 5.1

The proposed symbols encoding E_{S_n} generates minimum possible variable-length codewords for alphabet symbols via running series of sampling S_n over encoding generator $\mathscr{E}(\mathscr{E}_l, m, f_i, a_c, a_l, n)$. For $S_n \geq 1$.

Theorem 5.2

The sampling improvement function f_i for symbols that are randomly picked by the encoding generator $\mathscr{E}(\mathscr{E}_l, m, f_i, a_c, a_l, n)$ is always assigns minimum possible codewords for symbols as can be formatted by

VI. Experiments and Results

In this section, we discuss the performance of FTP and HTTP protocols in terms of transfer time and size using both data-encoding schemes: standard and proposed with/out GZIP and MFCompress compression algorithms for a variety of genomic datasets. The genomic datasets examined through standard and proposed encoding schemes are in FASTA format [56]. FASTA file is a single sequence described by a title line followed by one or more data lines. The title line begins with a right-angle bracket followed by a label. The label ends with the first white space character. Everything after that on the first line is considered a comment. The data lines begin right after the title line and contain the sequence characters in order. Each data line, except the last, should be exactly 60 letters long, although many programs allow some flexibility on that score. The examined genomic datasets were divided into two groups: actual and simulated datasets. Actual datasets were downloaded through two sources: National Center for Biotechnology Information (NCBI) [57] and the University of California Santa Cruz (UCSC) [58] websites, as shown in Table III on page 8. Simulated datasets were generated via our genome generator that controls symbol repetitions to assess our encoding scheme, as can been seen in Table III on page 8.

Table IV: Experimental setup

Specifications	Details
Processor	2.4 GHz Intel Core i7
Memory	8 GB 1600 MHz DDR3
Graphics	Intel HD Graphics 4000 1024 MB
Operating System	Windows 8.1 Pro
Download	87 Mb/s
Upload	40 Mb/s
Programming Language	C# .Net
Protocols	FTP, HTTP, and BitTorrent
Dataset Sizes	550MB (1) - 340GB (9)

A. Experimental setup

This paper compares the proposed CNN algorithm-based content-encoding to the standard content-encoding of network transfer protocols, such as HTTP, FTP, and BitTorrent. Several datasets of sizes up to 430GB of FASTA files have been fed into these implementations to validate our content-encoding. The experiments were performed on machines that have specifications shown in Table IV on page 8.

B. Actual Datasets Results

This section discusses and evaluates the performance and results of the proposed data minimization scheme, compared to the standard encoding. Our experimental results of real datasets are obtained through the transfer of the datasets using FTP, HTTP, and BitTorrent protocols using standard and proposed data-encoding schemes with/out two compression algorithms:GZIP and MFCompress as shown in Figure 4 on page 10, 5 on page 10, and 5 on page 10 and Tables V on page 9, VI on page 9, VII on page 9, VIII on page 9 , IX on page 9, X on page 11.

In order to assess the effectiveness of this work, we compared the genomic data size and transfer time of each dataset, using both data-encoding schemes: standard and proposed. The results show that the proposed method decreases data sizes that need to be transferred quickly, and also illustrate a corresponding decrease in the transfer time, as shown in Figure 4 on page 10 and Tables V on page 9, VI on page 9, VII on page 9, VIII on page 9 , IX on page 9, and X on page 11. For example, 1.20e+05 millisecond (ms) are required to transfer a compressed 550MB dataset using the traditional HTTP content-encoding with GZIP, 3.83e+04 ms via the FTP, whereas 2.02e+03 ms are required to transfer the same file via the DVLBE and 6.36e+05 ms over HTTP with MFCompress algorithm. This rate of transfer is approximately 98 times faster than HTTP standard-based, and about 95 times faster than FTP, and 99-fold faster than HTTP-standard with MFCompress. Also, the 30GB dataset was transferred in 7.20e+06 ms, using the HTTP standard and GZIP-based, 6.312e+06 ms by FTP standard and GZIP-based, 9.60e+07 HTTP FLBE-MFCompress-based, whereas it took only 3.26e+05 ms to transfer the file using HTTP proposed and GZIP-based. This is about 95-fold faster than the standard content-encoding of HTTP and FTP, without using compression algorithms, compared to 99-fold faster than the standard content-encoding of

Table V: Time acceleration comparisons of actual datasets in (ms) without compression

Dataset	HTTP	FTP	HTTP
	standard-based	standard-based	proposed-based
1	1.44e+05	4.28e+04	2.24e+03
2	1.68e+05	7.12e+04	7.31e+03
3	6.10e+05	1.66e+05	1.11e+04
4	3.33e+06	1.24e+06	8.08e+04
5	5.11e+06	3.22e+06	1.98e+05
6	8.86e+06	7.58e+06	4.90e+05
7	1.44e+07	1.06e+07	7.77e+05
8	2.23e+07	1.30e+07	8.01e+05
9	1.03e+08	4.18e+07	8.22e+06

HTTP when using the compression algorithm of MFCompress. Our results illustrate the performance of examined genomic datasets that have sizes up to 340GB. The size reduction, when using the proposed method, reaches to 96%, compared to the standard content-encoding of HTTP and FTP when utilizing compression algorithm of GZIP as shown in Figure 4 on page 10. Tables V on page 9, VI on page 9, VII on page 9, VIII on page 9 , IX on page 9, and X on page 11 that provide more results about time acceleration and size reduction of actual datasets over HTTP and FTP protocols using standard and proposed content-encoding schemes with/out compression algorithms (GZIP and MFCompress).

Moreover, we implemented a BitTorrent protocol that uses the standard content-encoding scheme to compare the results in terms of transfer time and size with HTTP that uses our proposed encoding scheme. The results are shown in Figures 5 on page 10 and 6 on page 11 for a 1GB genomic dataset of a FASTA format that was downloaded from the NCBI website (Homo_sapiens.GRCH38.dna_sm_toplevel). The results show that transfer time, when using a single computer (server) equipped by HTTP that uses the proposed encoding method, is almost similar to the transfer time when using 10 computers in parallel, equipped by a BitTorrent protocol that utilizes the standard content-encoding scheme. As expected, with the increasing number of machines, the time to transfer decreases sharply over BitTorrent. It can also be observed that 1 machine using HTTP utilizes the proposed scheme and requires the same time to transfer 1GB of the genomic dataset using 10 parallel machines over BitTorrent. This occurrence is due to the massive reduction in size that can be attributed to our encoding scheme. Also, note that employing HTTP equipped by the proposed encoding mechanism illuminates the need to use multiple machines in parallel and therefore reduces overall costs.

C. Simulated Dataset Results

In this section, we discuss the performance of the transfer protocols HTTP and FTP in terms of transfer time and size when using standard and proposed encoding methods during the transmission of simulated genomic datasets. We implemented a genomic *DNA* generator to generate simulated *FASTA* format genomic datasets.

Table VI: Time acceleration comparisons of actual datasets in (ms) with compression

Dataset	HTTP	FTP	HTTP	HTTP
	standard-GZIP	standard-GZIP	proposed-GZIP	standard-MFCompress
1	1.20e+05	3.83e+04	2.02e+03	6.36e+05
2	1.20e+05	5.89e+04	4.57e+03	6.82e+05
3	4.21e+05	1.42e+05	5.85e+03	3.73e+06
4	2.52e+06	1.15e+06	4.25e+04	N/A
5	3.60e+06	2.78e+06	1.52e+05	3.75e+05
6	7.20e+06	6.32e+06	3.26e+05	9.60e+07
7	1.08e+07	8.62e+06	4.57e+05	2.24e+08
8	1.80e+07	1.05e+07	5.72e+05	2.46e+07
9	7.98e+07	3.24e+07	5.27e+06	1.74e+08

Table VII: Size reduction comparisons of actual datasets in (KB) without compression

Dataset	HTTP	FTP	HTTP
	standard-based	standard-based	proposed-based
1	5.63e+05	5.63e+05	6.76e+04
2	6.39e+05	6.39e+05	9.59e+04
3	1.95e+06	1.95e+06	3.32e+05
4	1.11e+07	1.11e+07	1.45e+06
5	1.48e+07	1.48e+07	2.81e+06
6	3.05e+07	3.05e+07	4.58e+06
7	4.36e+07	4.36e+07	5.67e+06
8	4.53e+07	4.53e+07	7.25e+06
9	3.46e+08	3.46e+08	4.85e+07

Table VIII: Size reduction comparisons of actual datasets in (KB) with compression (P refers to the proposed while S refers to the standard encoding scheme)

Dataset	HTTP	FTP	HTTP	HTTP
	S-GZIP	S-GZIP	P-GZIP	S-MFCompress
1	3.94e+05	3.15e+05	1.69e+04	1.88e+05
2	3.96e+05	3.13e+05	6.64e+04	9.31e+04
3	1.35e+06	8.79e+05	8.78e+04	6.67e+05
4	6.24e+06	5.79e+06	7.96e+05	N/A
5	8.74e+06	6.22e+06	2.97e+06	2.34e+06
6	1.56e+07	1.34e+07	6.42e+06	5.39e+06
7	2.49e+07	1.79e+07	8.75e+06	6.47e+06
8	2.81e+07	2.31e+07	1.11e+07	9.08e+06
9	1.91e+08	1.63e+08	8.30e+07	6.96e+07

Table IX: Size reduction rates of actual datasets with compression (P refers to the proposed while S refers to the standard encoding scheme)

Dataset	HTTP	HTTP	HTTP vs FTP
	P-GZIP vs S-GZIP	P-GZIP vs S-MFCompress	P-GZIP vs S-GZIP
1	96%	91%	95%
2	83%	29%	79%
3	93%	87%	90%
4	87%	N/A%	86%
5	66%	127%	52%
6	59%	119%	52%
7	65%	135%	51%
8	61%	122%	52%
9	56%	119%	49%

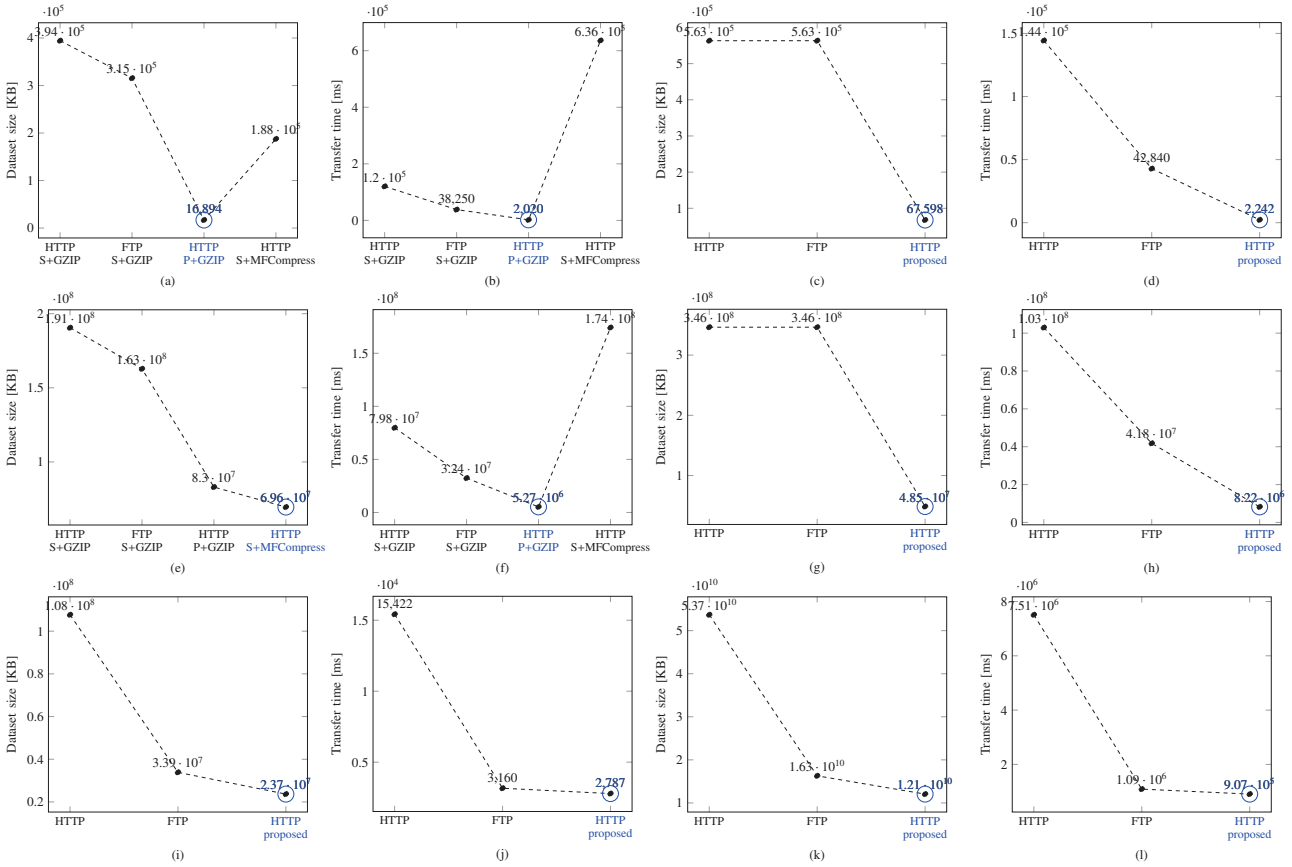
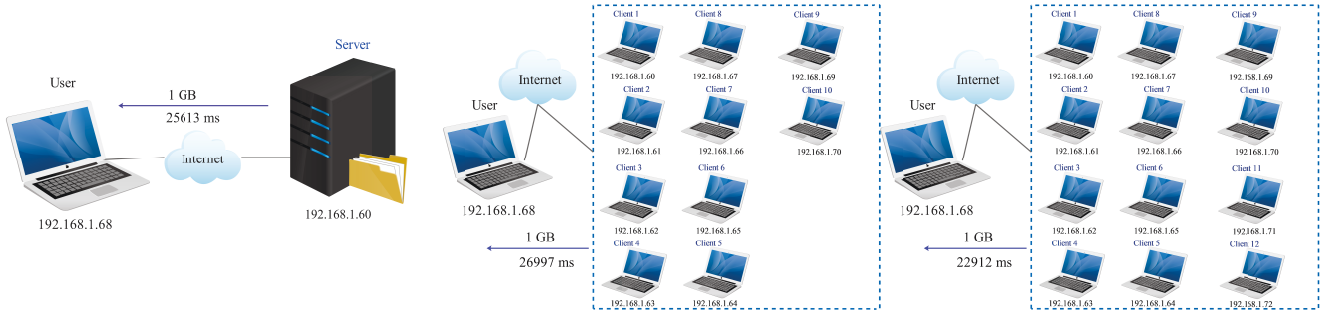


Figure 4: Transfer size and time of actual datasets 550MB and 340GB as defined in Table III on page 8 and generated datasets 100MB and 50GB as defined in Table XI on page 11 over multiple transfer protocols: HTTP and FTP, standard and proposed encoding schemes, with/out involving compression algorithms: GZIP and MFCompress. (a) Transfer size of 550MB with compression (b) Transfer time of 550MB with compression (c) Transfer size of 550MB without compression (d) Transfer time of 550MB without compression. (e) Transfer size of 340GB with compression (f) Transfer time of 340GB with compression (g) Transfer size of 340GB without compression (h) Transfer time of 340GB without compression. (i) Transfer size of 100MB without compression (j) Transfer time of 100MB without compression (k) Transfer size of 50GB without compression (l) Transfer time of 50GB without compression



(a) Transfer time of 1GB genomic dataset using 1 machine that utilizes the proposed data minimization algorithm

(b) Transfer time of 1GB genomic dataset using 10 machines that utilizes the standard content-encoding algorithm

(c) Transfer time of 1GB genomic dataset using 12 machines that utilizes the standard content-encoding algorithm

Figure 5: Transfer time in millisecond of 1GB genomic dataset using a proposed and a standard content-encoding schemes using multiple (1 - 12) machines in parallel.

We generated 18 FASTA datasets in different sizes and symbol frequencies to assess the results of the proposed encoding scheme, which works efficiently for more occurrence symbols, as shown in Table XI on page 11. The genomic generator (GG) runs in two different modes: *auto* and *manually*.

In the auto mode, the GG takes one input parameter

Table X: Time acceleration rates of actual datasets with compression (P refers to the proposed while S refers to the standard encoding scheme)

Dataset	HTTP	HTTP	HTTP vs FTP
	P-GZIP vs S-GZIP	P-GZIP vs S-MFCompress	P-GZIP vs S-GZIP
1	98x	99x	95x
2	96x	99x	92x
3	99x	99x	96x
4	98x	N/A	96x
5	96x	99x	95x
6	95x	99x	95x
7	96x	99x	95x
8	97x	98x	95x
9	93x	97x	84x

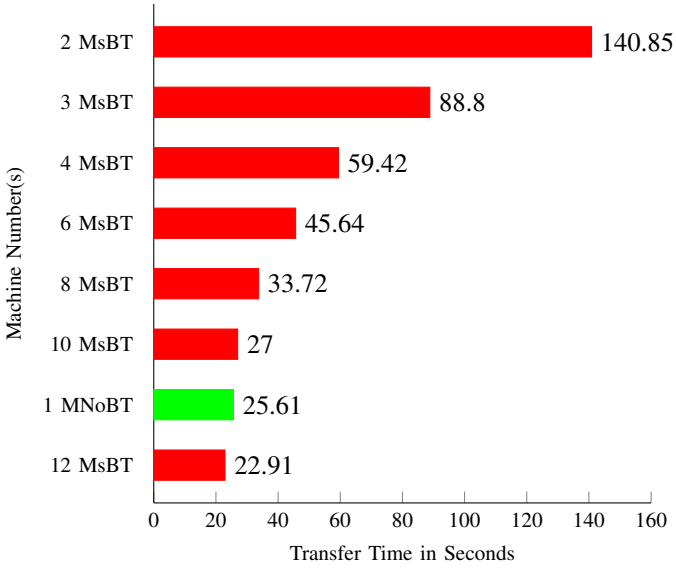


Figure 6: Transfer time in millisecond of 1GB genomic dataset using the proposed data minimization algorithm using a single machine and a standard content-encoding algorithm using multiple (2 - 12) machines in parallel.

that represents the size of the dataset to be generated, and then produces a genomic dataset with almost equal symbol repetitions i.e. 25% each. In the manual mode, the GG takes three input parameters: needed *dataset size*, *symbol* and needed *symbol repetition* for this symbol and generates a genomic dataset with required repetitions. The manual GG controls 1 symbol repetition only, and the rest are generated randomly at different rates. To simplify our experiments, we generated 18 different genomic datasets at different symbol frequencies and sizes up to 50GB. For example, we generated a 10GB genomic dataset in 3 different frequency rates that are 25%, 35% and 50% for the symbol (A). Specifying symbol repetitions verifies the performance of the proposed encoding method of network transfer protocols for big genomic datasets, from size and time perspectives. The first scenario assumes 25% occurrence rate for each symbol in a dataset. The second scenario assumes a 35% repetition rate for one symbol of a dataset, and 65% occurrences for the 3 other symbols. The third scenario assumes a 50% repetition for one symbol, while

Table XI: Generated FASTA files using a genomic generator

Datasets	Number of DNA Symbols in each Dataset							
	A	C	G	T	A%	C%	G%	T%
100M[25]	26843545	26843645	31808042	21878898	0.25%	0.25%	0.30%	0.20%
100M[35]	37580963	30868915	13425306	25498926	0.35%	0.28%	0.13%	0.24%
100M[50]	53687091	14786579	25675457	13224952	0.50%	0.14%	0.24%	0.12%
500M[25]	134217728	149129757	149112975	104410196	0.25%	0.28%	0.28%	0.19%
500M[35]	187904819	153550318	116324344	79091072	0.35%	0.29%	0.21%	0.15%
500M[50]	268435456	93792797	100263349	74378798	0.50%	0.17%	0.19%	0.14%
1G[25]	268435456	219637924	292809252	292858680	0.25%	0.21%	0.27%	0.27%
1G[35]	375809638	221793199	254370797	221767473	0.35%	0.20%	0.24%	0.21%
1G[50]	536870912	203878326	101922932	231068630	0.50%	0.19%	0.09%	0.22%
5G[25]	1342177280	1283854849	1517198501	1225475930	0.25%	0.24%	0.28%	0.23%
5G[35]	1879048192	1143192988	1203320398	1143143958	0.35%	0.21%	0.22%	0.22%
5G[50]	2684354560	723874757	1176284699	784189984	0.50%	0.13%	0.22%	0.15%
10G[25]	2684354560	2467876125	2338051024	3247131411	0.25%	0.23%	0.22%	0.30%
10G[35]	3758096384	2594087000	2532325440	1852902248	0.35%	0.24%	0.24%	0.17%
10G[50]	5368709120	894831061	2013252324	2460615495	0.50%	0.08%	0.19%	0.23%
50G[25]	13421772800	13421823505	14380428192	12463041103	0.25%	0.25%	0.27%	0.23%
50G[35]	18790481920	10550108277	9738567871	14607897292	0.35%	0.20%	0.18%	0.27%
50G[50]	26843545600	10949303940	7064179103	8830011357	0.50%	0.20%	0.14%	0.16%

the 3 other symbols have different repetitions within 50% of the dataset. The goals of the GG implementation are to customize file sizes and occurrence rates. Also, the GG avoids privacy violations to validate our content-encoding method, in contrast to other possible approaches during big genomic transferring. Note: A file of 25% of symbol (A) does not necessarily mean the rest of the symbols have the same symbol repetitions i.e. 25% for (C), 25% for (G), and 25% for (T), they may vary.

D. Evaluation of Simulated Dataset Results

For more accuracy and further validation of the proposed encoding scheme, we performed experiments on generated genomic datasets, using our genome generator, as shown in Table XI on page 11. In this section, we compare and discuss the performance of HTTP and FTP using both encoding schemes: standard and proposed, with/out involving compression algorithms (GZIP and MFCompress). Performance of the proposed encoding method indicates better results than the standard scheme, as shown in Tables XII on page 12 and XIII on page 12.

Both compression algorithms: GZIP and MFCompress were involved, along with our proposed encoding scheme over HTTP protocol, while only GZIP was utilized for FTP protocol implementation to validate our proposed scheme performance. The results showed a significant improvement in terms of the data size reduction and acceleration of transfer time when using our encoding scheme by significantly minimizing dataset sizes during the data transfer phase, as shown in Figure 4 on page 10 and Tables XII on page 12 and XIII on page 12 compared to the standard encoding method. As expected, the transfer time for genomic datasets is always much shorter when using the proposed encoding scheme than the standard one that is currently used to encode contents using HTTP and FTP protocols. HTTP protocol shows a significant enhancement of genomic data transfer time when using the proposed encoding in contrast to the standard scheme, due to symbol repetitions and continuous updates of the encoding tree via utilizing the CNN algorithm. The proposed encoding method works efficiently for datasets that include more symbol repetitions. However, utilizing the CNN algorithm, as illustrated in section V on page 7, needs more string reads

Table XII: Size reduction comparisons of simulated datasets in (kB)

Dataset	HTTP	FTP	HTTP	HTTP vs HTTP	HTTP vs FTP
	Stand. + GZIP	Stand. + GZIP	Propo. + GZIP	Propo. vs Stand.	Propo. vs Stand.
100MB[25]	1.078e+08	1.078e+08	2.906e+07	73%	73%
100MB[35]	1.075e+08	1.075e+08	2.703e+07	75%	75%
100MB[50]	1.077e+08	1.077e+08	2.367e+07	78%	78%
500MB[25]	5.369e+08	5.369e+08	1.473e+08	73%	73%
500MB[35]	5.372e+08	5.372e+08	1.352e+08	75%	75%
500MB[50]	5.369e+08	5.369e+08	1.217e+08	77%	77%
1GB[25]	1.074e+09	1.074e+09	2.928e+08	73%	73%
1GB[35]	1.074e+09	1.074e+09	2.769e+08	74%	74%
1GB[50]	1.074e+09	1.074e+09	2.430e+08	77%	77%
5GB[25]	5.369e+09	5.369e+09	1.466e+09	73%	73%
5GB[35]	5.369e+09	5.369e+09	1.393e+09	74%	74%
5GB[50]	5.369e+09	5.369e+09	1.195e+09	77%	77%
10GB[25]	1.074e+10	1.074e+10	2.922e+09	73%	73%
10GB[35]	1.074e+10	1.074e+10	2.763e+09	74%	74%
10GB[50]	1.074e+10	1.074e+10	2.433e+09	77%	77%
50GB[25]	5.369e+10	5.369e+10	1.486e+10	72%	72%
50GB[35]	5.369e+10	5.369e+10	1.412e+10	74%	74%
50GB[50]	5.369e+10	5.369e+10	1.205e+10	78%	78%

Table XIII: Time acceleration comparisons of simulated datasets in (ms)

Dataset	HTTP	FTP	HTTP	HTTP vs HTTP	HTTP vs FTP
	Stand. + GZIP	Stand. + GZIP	Propo. + GZIP	Propo. vs Stand.	Propo. vs Stand.
100MB[25]	2.277e+04	1.265e+04	3.249e+03	7.01x	3.89x
100MB[35]	1.997e+04	8.681e+03	2.710e+03	7.37x	3.20x
100MB[50]	1.542e+04	9.639e+03	2.787e+03	5.53x	3.46x
500MB[25]	9.534e+04	5.417e+04	1.133e+04	8.41x	4.78x
500MB[35]	8.502e+04	4.383e+04	1.049e+04	8.11x	4.18x
500MB[50]	7.595e+04	5.238e+04	9.205e+03	8.25x	5.69x
1GB[25]	1.881e+05	8.956e+04	1.837e+04	10.24x	4.87x
1GB[35]	1.665e+05	6.938e+04	1.858e+04	8.96x	3.73x
1GB[50]	1.289e+05	6.894e+04	1.731e+04	7.45x	3.98x
5GB[25]	9.830e+05	5.922e+05	1.152e+05	8.53x	5.14x
5GB[35]	8.448e+05	3.840e+05	9.681e+04	8.73x	3.97x
5GB[50]	6.662e+05	2.221e+05	7.688e+04	8.67x	2.89x
10GB[25]	1.881e+06	6.967e+05	1.716e+05	10.96x	4.06x
10GB[35]	1.670e+06	8.433e+05	1.902e+05	8.78x	4.43x
10GB[50]	1.367e+06	4.712e+05	1.787e+05	7.65x	2.64x
50GB[25]	1.005e+07	4.189e+06	1.149e+06	8.75x	3.64x
50GB[35]	8.793e+06	4.677e+06	1.073e+06	8.19x	4.36x
50GB[50]	7.511e+06	3.266e+06	9.066e+05	8.29x	3.60x

as samples, as well as additional time to update the encoding tree to ensure the assignment of the shortest codewords to the most repetition letters. The proposed encoding scheme is designed in a way that encodes datasets in the minimum possible binary codewords. Therefore, the proposed method minimizes datasets significantly during the transfer process due to including more symbols *repetitions* after applying a CNN mechanism, as illustrated in section V on page 7. For example, the first experiment spent 2.28e+04 ms to transfer the 100M dataset with 25% frequency for the symbol A using standard encoding over HTTP, 1.27e+04 ms via the FTP, compared to 3.25e+03 ms to transfer the same file via the proposed scheme with 7-fold faster than standard encoding.

The proposed encoding is impacted by two factors: the number and repetitions of the symbols of genomic datasets. Therefore, utilizing our encoding scheme that relies on the CNN algorithm provides a better performance for big genomic datasets. Also, the proposed content-encoding method presents an ideal solution to transfer all genomic datasets that contain different symbol repetitions with an average acceleration of 99-fold, compared to the traditional HTTP. The time acceleration and size reduction for all FASTA files in Table XI on page 11 using all mentioned encoding schemes are summarized in Figure 4 on page 10 and Tables XII on page 12 and XIII on page 12. This encoding scheme, when compared to a standard scheme, features improved reliability, scalability, performance and security. The reliability factor is evidenced in its ability to use both encoding (genomics and universal) schemes when assigning a codeword for each symbol. The scalability feature is present by enabling the protocol to update the binary representation of each alphabet symbol according to the deep learning mechanisms, as illustrated in section V on page 7. This encoding scheme indicates better performance via reducing the dataset during the data transfer phase, which, in turn, reduces the network traffic. The security of this scheme stems from changing the binary representations to each single character of the alphabet several times for the same dataset in a single transfer session.

VII. Conclusion

In this paper, we implemented a novel deep learning-based data minimization algorithm to integrate with transfer protocols to reduce the size of big genomic datasets during the transfer phase, and then to transfer the data securely in less time. The implementation results illustrate that the proposed data minimization algorithm is capable of reducing the transfer time 99-fold, compared to the standard content-encoding of HTTP, and 96-fold compared to FTP on tested datasets. We used GZIP and MFCompress algorithms as optional compression algorithms, in addition to our data minimization algorithm to assess how the transfer protocol behaves in terms of transfer time and size. Also, we showed that our data minimization algorithm provides the best size reduction, reduces transfer time, and securely transfers big genomic datasets. Our proposed data minimization mechanism relies on a deep learning-based method, while encoding the data during data transfer, and then transfers the data securely in a shortened time, as illustrated in section V on page 7. We demonstrated that the data size can be significantly reduced by our adaptive encoding, compared to a standard content-encoding scheme, with and without compression algorithms, as well. Also, we implemented a genomic dataset generator of a FASTA file format to verify the performance of our current data minimization scheme and then compared it to the standard, as well as our previous content-encoding schemes. Our proposed genome generator allowed us to control the repetition in the data, which was instrumental in assessing the performance of our data minimization algorithm. The tested encoding schemes, standard and proposed, were implemented over HTTP, FTP and BitTorrent protocols, with/out involving compression algorithms. Our experiments indicated that utilizing a CNN-based content-encoding scheme performs much better than the current and common use of the transfer protocol content-encoding scheme by assigning short codewords for the dataset characters. We conclude that the proposed data minimization algorithm provides the best performance among current content-encoding approaches for big genomic datasets.

Acknowledgment

The authors thank Dr. Todd Barkman, a professor in the Department of Biological Sciences at Western Michigan University for his valued feedback and comments. This work was supported in part by grants US National Science Foundation CRII CCF-1464268 and US National Science Foundation CAREER ACI-1651724. The authors would like to thank the members of the Parallel Computing and Data Science (PCDS) lab at Western Michigan University for their valuable input while in the process of developing this paper.

References

[1] C. Mora, D. P. Tittensor, S. Adl, A. G. Simpson, and B. Worm, "How many species are there on earth and in the ocean?" *PLoS biology*, vol. 9, no. 8, p. e1001127, 2011.
[2] F. S. Collins and H. Varmus, "A new initiative on precision medicine," *New England Journal of Medicine*, vol. 372, no. 9, pp. 793–795, 2015.
[3] T. C. Carter and M. M. He, "Challenges of identifying clinically actionable genetic variants for precision medicine," *Journal of healthcare engineering*, vol. 2016, 2016.
[4] N. Drake *et al.*, "Cloud computing beckons scientists." *Nature*, vol. 509, no. 7502, pp. 543–544, 2014.
[5] R. Spencer, "The square kilometre array: The ultimate challenge for processing big data," in *Data Analytics 2013: Deriving Intelligence and Value from Big Data, IET Seminar on*. IET, 2013, pp. 1–26.
[6] M. Schatz, "The next 20 years of genome research," *bioRxiv*, p. 020289, 2015.
[7] Census, "World population."
[8] J. Hadfield and N. Loman, "Next generation genomics: world map of high-throughput sequencers," 2014.
[9] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext transfer protocol–http/1.1," Internet Engineering Task Force (IETF), Tech. Rep., 1999.
[10] J. Postel and J. Reynolds, "File transfer protocol," *The Internet Engineering Task Force*, 1985.
[11] S. Deorowicz and S. Grabowski, "Compression of dna sequence reads in fastq format," *Bioinformatics*, vol. 27, no. 6, pp. 860–862, 2011.
[12] A. J. Cox, M. J. Bauer, T. Jakobi, and G. Rosone, "Large-scale compression of genomic sequence databases with the burrows–wheeler transform," *Bioinformatics*, vol. 28, no. 11, pp. 1415–1419, 2012.
[13] S. W. Hodson, S. W. Poole, T. M. Ruwart, and B. W. Settlemyer, "Moving large data sets over high-performance long distance networks," Citeseer, Tech. Rep., 2011.
[14] N. I. of Health *et al.*, "An overview of the human genome project," 2005.
[15] R. A. Gibbs, J. W. Belmont, P. Hardenbol, T. D. Willis, F. Yu, H. Yang, L.-Y. Ch'ang, W. Huang, B. Liu, Y. Shen *et al.*, "The international hapmap project," 2003.
[16] W. S. Bush and J. H. Moore, "Genome-wide association studies," *PLoS computational biology*, vol. 8, no. 12, p. e1002822, 2012.
[17] M. D. Mailman, M. Feolo, Y. Jin, M. Kimura, K. Tryka, R. Bagoutdinov, L. Hao, A. Kiang, J. Paschall, L. Phan *et al.*, "The ncbi dbgap database of genotypes and phenotypes," *Nature genetics*, vol. 39, no. 10, pp. 1181–1186, 2007.
[18] J. Kaye, C. Heeney, N. Hawkins, J. De Vries, and P. Boddington, "Data sharing in genomics–re-shaping scientific practice," *Nature reviews. Genetics*, vol. 10, no. 5, p. 331, 2009.
[19] K. Holtman and A. Mutz, "Transparent content negotiation in http," Internet Engineering Task Force (IETF), Tech. Rep., 1998.
[20] J. C. Mogul, F. Douglis, A. Feldmann, and B. Krishnamurthy, "Potential benefits of delta encoding and data compression for http," in *ACM SIGCOMM Computer Communication Review*, vol. 27, no. 4. ACM, 1997, pp. 181–194.
[21] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of things: A survey on enabling technologies, protocols, and applications," *IEEE Communications Surveys Tutorials*, vol. 17, no. 4, pp. 2347–2376, Fourthquarter 2015.
[22] M. Aledhari and F. Saeed, "Design and implementation of network transfer protocol for big genomic data," *IEEE 4th International Congress on Big Data (BigData Congress 2015)*, June 2015.
[23] M. Aledhari, M. Hefeida, and F. Saeed, *Wired/Wireless Internet Communications: 14th International Conference, WWIC 2016, Thessaloniki, Greece, May 25-27, 2016, Revised Selected Papers*. Springer International Publishing, 2016, ch. A Variable-Length Network Encoding Protocol for Big Genomic Data.
[24] S. Pichai and L. Upson, "Introducing the google chrome os," *The Official Google Blog*, vol. 7, 2009.
[25] Apple. Safari 3.1: Product overview. Apple.
[26] S. J. Davis and K. M. Murphy, "A competitive perspective on internet explorer," *American Economic Review*, pp. 184–187, 2000.
[27] M. Diaz, G. Juan, O. Lucas, and A. Ryuga, "Big data on the internet of things," in *2012 Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, 2012, pp. 978–0.
[28] B. Cohen, "Incentives build robustness in bittorrent," in *Workshop on Economics of Peer-to-Peer systems*, vol. 6, 2003, pp. 68–72.
[29] J. Postel, "Transmission control protocol," *The Internet Engineering Task Force*, 1981.
[30] A. Ng, P. Greenfield, and S. Chen, "A study of the impact of compression and binary encoding on soap performance," in *Proceedings of the Sixth Australasian Workshop on Software and System Architectures (AWSA2005)*. Citeseer, 2005, pp. 46–56.
[31] C. Wilks, M. S. Cline, E. Weiler, M. Diehkans, B. Craft, C. Martin, D. Murphy, H. Pierce, J. Black, D. Nelson *et al.*, "The cancer genomics hub (cghub): overcoming cancer through the power of torrential data," *Database: The Journal of Biological Databases and Curation.*, vol. 2014, no. bau093, 2014.
[32] B. Cohen, "Bittorrent-a new p2p app," *Yahoo eGroups*, 2001.
[33] A. K. Parekh and R. G. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: the single-node case," *IEEE/ACM transactions on networking*, vol. 1, no. 3, pp. 344–357, 1993.
[34] Y. Ren, J. Li, S. Shi, L. Li, G. Wang, and B. Zhang, "Congestion control in named data networking–a survey," *Computer Communications*, vol. 86, pp. 1–11, 2016.
[35] (2017). [Online]. Available: https://www.internet2.edu/about-us/
[36] Y. Rathore, M. K. Ahirwar, and R. Pandey, "A brief study of data compression algorithms," *International Journal of Computer Science and Information Security*, vol. 11, no. 10, p. 86, 2013.
[37] L. P. Deutsch, "Deflate compressed data format specification version 1.3," *The Internet Engineering Task Force*, 1996.
[38] J. Seward, "Bzip2 and libbzip2: a program and library for data compression," *htpp://sources. redhat. com/bzip2*, 1998.
[39] L. J. Krakauer and L. Baxter, "Method of fixed-length binary encoding and decoding and apparatus for same," Apr. 4 1989, uS Patent 4,818,969.
[40] E. N. Gilbert and E. F. Moore, "Variable-length binary encodings," *Bell System Technical Journal*, vol. 38, no. 4, pp. 933–967, 1959.
[41] S. Grumbach and F. Tahi, "A new challenge for compression algorithms: genetic sequences," *Information Processing & Management*, vol. 30, no. 6, pp. 875–886, 1994.
[42] L. Chen, S. Lu, and J. Ram, "Compressed pattern matching in dna sequences," in *Computational Systems Bioinformatics Conference, 2004. CSB 2004. Proceedings. 2004 IEEE*. IEEE, 2004, pp. 62–68.
[43] N. J. Larsson and A. Moffat, "Off-line dictionary-based compression," *Proceedings of the IEEE*, vol. 88, no. 11, pp. 1722–1732, 2000.
[44] Y. Shibata, T. Matsumoto, M. Takeda, A. Shinohara, and S. Arikawa, "A boyer-moore type algorithm for compressed pattern matching," in *CPM*. Springer, 2000, pp. 181–194.
[45] D. Salomon and G. Motta, *Handbook of data compression*. Springer Science & Business Media, 2010.
[46] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Transactions on information theory*, vol. 23, no. 3, pp. 337–343, 1977.
[47] J. Cleary and I. Witten, "Data compression using adaptive coding and partial string matching," *IEEE transactions on Communications*, vol. 32, no. 4, pp. 396–402, 1984.
[48] M. D. Cao, T. I. Dix, L. Allison, and C. Mears, "A simple statistical algorithm for biological sequence compression," in *Data Compression Conference, 2007. DCC'07*. IEEE, 2007, pp. 43–52.
[49] D. A. Huffman *et al.*, "A method for the construction of minimum redundancy codes," *Proceedings of the IRE*, vol. 40, no. 9, pp. 1098–1101, 1952.
[50] M. Effros, K. Visweswariah, S. R. Kulkarni, and S. Verdú, "Universal lossless source coding with the burrows wheeler transform," *IEEE Transactions on Information Theory*, vol. 48, no. 5, pp. 1061–1081, 2002.
[51] M. Burrows and D. J. Wheeler, "A block-sorting lossless data compression algorithm," 1994.

[52] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
[53] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.
[54] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.
[55] S. Lazebnik, C. Schmid, and J. Ponce, "Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories," in *Computer vision and pattern recognition, 2006 IEEE computer society conference on*, vol. 2. IEEE, 2006, pp. 2169–2178.
[56] R. Chenna, H. Sugawara, T. Koike, R. Lopez, T. J. Gibson, D. G. Higgins, and J. D. Thompson, "Multiple sequence alignment with the clustal series of programs," *Nucleic acids research*, vol. 31, no. 13, pp. 3497–3500, 2003.
[57] (2015, 12). [Online]. Available: http://www.ncbi.nlm.nih.gov
[58] (2015, 12). [Online]. Available: http://www.ucsc.edu