IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

MulAligner: A Multiple Sequence Alignment Error-Correction Tool Using Deep Learning Algorithm

Charles Saah, Ferdinand Katsriku, Robert A. Sowah, Wiafe Owusu – Banahene

Abstract

Advancement in technology has led to the sequencing of 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 problem persists mainly because either the action or approach is too complex to execute, thus requiring a high level of computational resources. We identified the error correction process 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. Our system is tested using the genome data NA12878, which was obtained from the NCBI, and it achieved 99.5% classification accuracy

Keywords: sequencing, genome, generation, errors, technological advancement, pattern recognition, machine learning

Introduction

Deoxyribonucleic acid (DNA) sequencing started with the chain termination process commonly known as the Sanger project or first-generation sequences (FGS) [1]. The FGS produced long sequencing reads (500 – 1000) and was successfully used to sequence the DNA of H. influenza [2] but the read length, cost, and complexity of producing sequencing reads restricted their use in research and thus lead to a new pyrosequencing technology by 454 Life Science whose sequencing reads were shorter than that of FGS and became known as the next generation sequencing (NGS) or the second generation sequencing (SGS) [3]. The approach sequenced by parallelism, where many sequencing reads are generated at a time [4] which lead to the sequencing of many organisms and gave birth to the whole genome sequencing (WGS) technique where the entire DNS of a larger organism was sequenced [5]. The parallelism introduced by the SGS duplicated sequencing reads and a lot of computing resources were needed to handle data from the SGS. Many novel sequencing algorithms and projects were undertaken to ease the challenges under the SGS sequencing processes, however, the issues persisted. Though SGS revolutionized the sequencing processes, its success was short-lived as it was detected that the sequencing processes introduced some levels of sequencing errors (insertion, deletion, and substitution) into the sequenced dataset [6][7]. In the quest to reduce or correct the errors introduced into the SGS sequencing datasets, a longer sequencing read process which became known as the third generation sequencing was introduced. Third-generation sequencing (TGS) processes which were mainly championed by platforms such as Pacific Biosciences [8], Illumina [9], Oxford Nanopore Technology (ONT) [10], Ion Torrent (Life Technologies) [11] revolutionized the genome sequencing processes by producing longer read lengths to correct the computational issues that arose from the use of second-generation sequences. The TGS processes did not alleviate the errors produced by its predecessor, but rather introduced higher error rates in its sequenced data. This calls for frantic efforts at reducing errors in generated datasets. Several genome sequencing error correction processes have been previously formulated through various research.

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 [12] 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.

The Bloom filtering algorithm is a specialized data structure that uses hash functions to guarantee that if a k-mer has been seen before during the hashing process, it reports the k-mer'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 k-mer as having seen it before, while in fact, the k-mer 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 there

Li Song et al [13] used the k-spectrum, a variant of the Debruin graph to correct errors in ribonucleic acid (RNA). For the successful use of the De Bruijn graph under the 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). 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 is 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. 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 [14]. The information is then used to build a filtering algorithm that helps to separate error k-mers from unique k-mers [14]. We will classify these error correction methods as traditional processes because they were the first call to genome sequencing error correction. The advancement in technology especially in machine learning has led to its application in various areas

Aside from the traditional genome sequencing error correction methods, machine learning approaches were adopted to correct sequencing errors. R. Poplin et al, [15] 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 possibility of the occurrence of a particular genotype. Using the NA12878 dataset, the network performance was measured as 98.98%. H. M. Al-Barhamtoshy [16] 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 network of generative adversaries to generate errors in their dataset and used their network to predict the error from the network of generative adversaries. Their network produced a 98.7% evaluation accuracy, thus missing out on 1.3% errors. Using several datasets, M. Krachunov et al [17] 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 below 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 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. L. Wang et al [18] 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. 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 dataset. They were able to reduce the errors in the Ecoli and the human NA12878 dataset 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. K. Kotlarz et al [19] used a combination of Keras and the Naïve algorithm as a machine learning tool that uses array-based genotype information to work on 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 possibility 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.

Methodology

We treated the error correction problem 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. In the architecture, multiple neurons are heaped together to form a layer, and multiple layers are also heaped together to form the architecture of the deep neural network. We then designed a deep convolutional neural network which is a deep neural network that uses convolutional functioning as defined mathematically in Equation 01 below. Deep convolutional neural networks use a parameter-sharing strategy to reduce the total number of parameters that would have been used if a different architectural design had been chosen. Through subsampling techniques (called pooling) of the input data, the deep convolutional neural network further reduces the number of parameters used, thus making them faster compared to similar network architecture. A sliding window originating from learned filters is used in the deep convolutional neural network architecture to automatically recognize patterns at multiple places [19].

$$\begin{cases} Y_i = X & i = 1 \\ Y_i = f_i\ (Z_i) & i > 1 \\ Z_i = g_i\ (Y_{i-\ 1;} W_i) & \end{cases} \qquad (01)$$

Where i represent 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, and $f_i(.)$ 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 previous layer $(Y_{i-1}\ )$. Equation 02 below shows the three weighted operations (convolution, pooling, and fully connectedness) that are performed under the architecture

$$\begin{cases} Z_i = W_i * Y_{i-1} & \text{if } i^{th} \text{ layer is convolutional} \\ Z_i = Pool(Y_{i-1}) & \text{if } i^{th} \text{ layer is pooling} \\ Z_i = W_i . Y_{i-1} & \text{if } i^{th} \text{ layer is fully-connected} \end{cases} \quad (02)$$

Where * represents a convolutional operation, represents either a maximum or average pooling, and (.) is used for matrix multiplication in a fully connected layer [20]. Figures 01 and 02, respectively, show the convolutional and dense layer activities that go on in the deep convolutional neural network. Figure 01 is the block diagram of the network

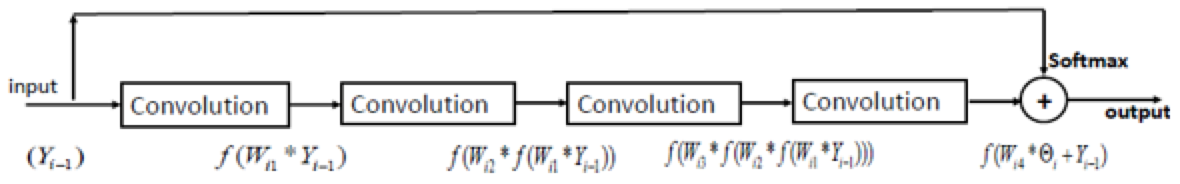
input
Convolution
Convolution
Convolution
Convolution
Softmax
+
output
(Y_{i-1})
$f(W_{i1} * Y_{i-1})$
$f(W_{i2} * f(W_{i1} * Y_{i-1}))$
$f(W_{i3} * f(W_{i2} * f(W_{i1} * Y_{i-1})))$
$f(W_{i4} * \Theta_i + Y_{i-1})$

Figure 01 block diagram of the deep convolutional neural network

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

$\Theta_i = f(W_{i3} * f(W_{i2} * f(W_{i1} * Y_{i-1})))$

Figure 02 below is the workflow of the architecture.

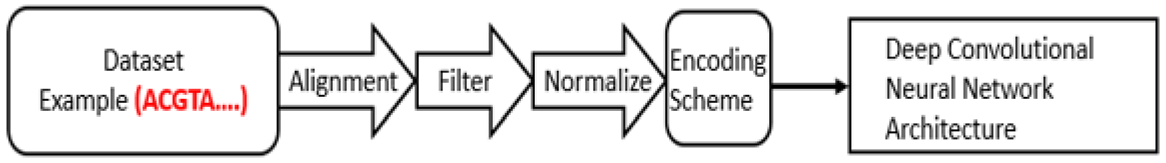
Dataset Example (ACGTA....)
Alignment
Filter
Normalize
Encoding Scheme
Deep Convolutional Neural Network Architecture

Figure 02: Schematic workflow of the framework

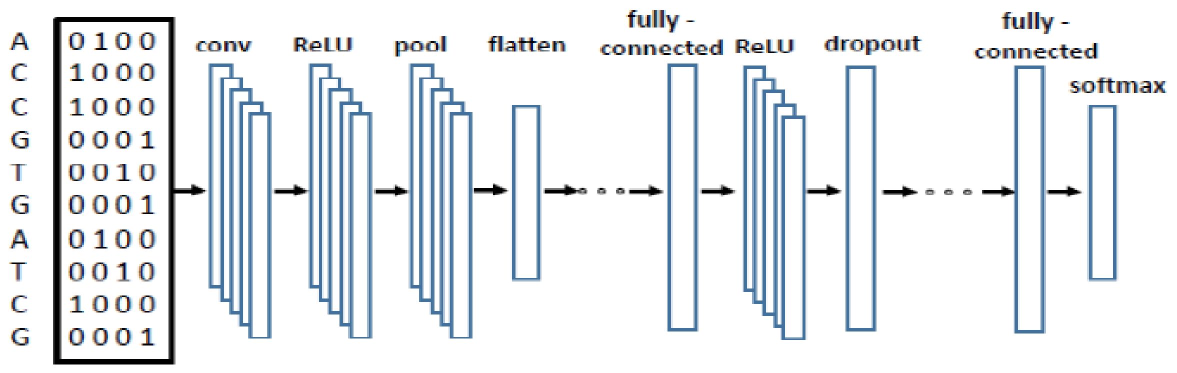

Figure 03. Convolutional Neural Network using the one-hot encoded scheme as input data for training the network and the softmax function to predict the output.

Using an 80%, 10%, and 10% ratio, the data NA12878, obtained from the National Center for Biotechnology Information (NCBI), was separated into training, validation, and testing, respectively. To encode the nucleotide bases, we employed an encoding technique. A, C, T, and G into [0 1 0 0], [1 0 0 0], [0 0 1 0], and [0 0 0 1]. This is because the nucleotide bases A, C, T, and G form categorical data, which has to be transformed into numeric data for the machine learning (ML) algorithm to work on it efficiently. The encoding scheme takes the attributes of a nucleotide and encodes it as a binary array of dimension 4. The network was subsequently trained and validated; please see figure 04 below. The validation process appeared to be working well between epochs 0 and 5, but it diverged after that point and did not recover.

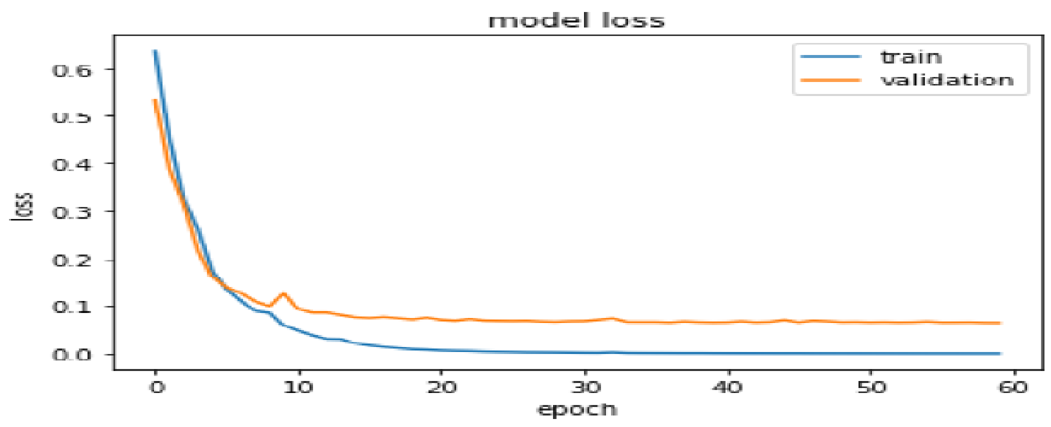

Figure 04. Initial training and validation of the network showing divergence in the validation after epoch 5.

Results and Discussion

Python programming language in the Jupiter notebook environment was used to create the architecture. The entire experiment was carried out using a Windows 10 laptop with a Hewlett-Packard (HP) Pavilion i5-4210U processor, 12GB of RAM running at 2.40GHz, and a 1TB hard drive. The performance of our network was verified using a series of classifications and the encoded dataset. We evaluated the network performance in terms of its accuracy and efficiency and compared it with similar work (table 01). Our network architecture performed better than the other two networks using the same dataset

Data set	Error correction methodology	Network performance (Accuracy %)
NA12878	Deep variant calling	98.98
NA12878	MulAligner	99.5
NA12878	Nanopore Error correction tool	97.7

Table 01: Comparison of the performance of the deep variant and Nanopore error correction method with our method

Figure 4 shows that after epoch 5, there was some divergence between the validation and training datasets suggesting that the network was overfitting the validation dataset. To correct this anomaly, we tweaked the network (figure 05) to resolve the overfitting.

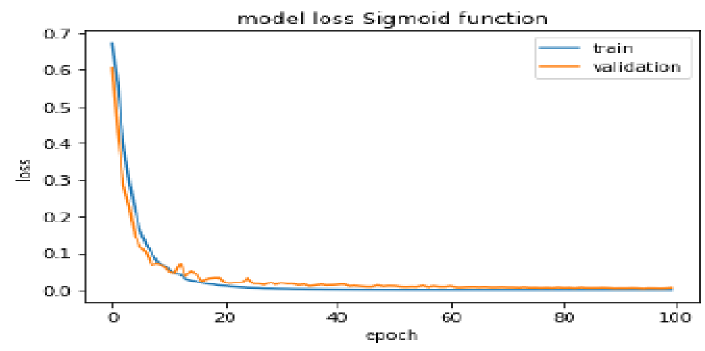

Figure: 05 Training and validation of the network using the sigmoid function at the output layer

Figure 05 above shows a near 100% validation of the trained data after epoch 20. This leads to very high network performance with about 99.5% accuracy, confirmed by the normalized confusion matrix (figure 06). To explain the confusion matrix (figure 06), we first translated it to table 2. The true positive (TP) value of 0.99 means that our network correctly predicted the underlying DNA with about 99% accuracy, and the false positive (FP) value of 0.01 or 1% means it wrongly predicted a correct nucleotide as an error. A false negative (FN) value of 0.00 or 0% means that the network did not predict an error as a correct nucleotide. Finally, a true negative (TN) value of 1.00 or 100% means that the network correctly predicted all nucleotide errors as errors. Using equations 03, 04, and 05 respectively, we calculated the classification accuracy (99.5%), error rate (0.5%), and precision value of 99%.

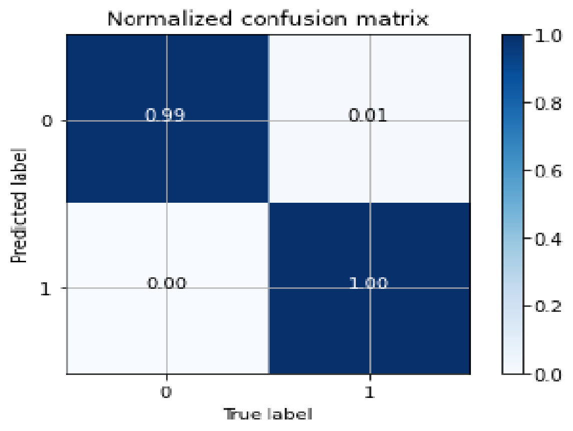

Figure 06: Normalized Confusion Matrix

True positive (TP) (0.99)	False positive (FP) (0.01)
False negative (FN) (0.00)	True negative (TN) (1.00)

Table 2: translated confusion matrix

Classification accuracy = (TP + TN) / (TP + TN + FP + FN)...……………equation 03

Misclassification rate (error rate) = (FP + FN) / (TP + TN + FN + FP)……equation .04

Precision = TP/(TP +FP) ……………………….,,equation 05

Conclusion

We treated the error correction problem as a big data analytic problem. This was done by dividing the dataset into k-mers and aligned with the reference genome. This created a large and variable dataset, thus qualifying as big data classification. We then built a deep convolutional neural network (DCNN), a specialized machine learning approach that is generally used in the area of natural language processing (NLP), computer vision (CV), and image recognition (IR) to reduce insertion and deletion errors. Our approach is highly successful because the error correction relies on k-mer alignments and analysis, which are very similar to NLP, where words of varying sizes are analyzed. Here k-mers, which are a set of strings or characters of variable lengths, are analyzed.

Reference

[1] F. Sanger and S. Nicklen, "DNA sequencing with chain-terminating," vol. 74, no. 12, pp. 5463–5467, 1977.

[2] 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.

[3] J. M. Heather and B. Chain, "Genomics The sequence of sequencers : The history of sequencing DNA," *Genomics*, vol. 107, no. 1, pp. 1–8, 2015, doi: 10.1016/j.ygeno.2015.11.003.

[4] W. Parson *et al.*, "Evaluation of next generation mtGenome sequencing using the Ion Torrent Personal Genome Machine (PGM)," *Forensic Sci. Int. Genet.*, vol. 7, no. 5, pp. 543–549, 2013, doi: 10.1016/j.fsigen.2013.06.003.

[5] R. Pinard *et al.*, "Assessment of whole genome amplification-induced bias through high-throughput,

massively parallel whole genome sequencing," *BMC Genomics*, vol. 7, 2006, doi: 10.1186/1471-2164-7-216.

[6] 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: 10.1093/bioinformatics/btp379.

[7] 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.

[8] E. Webb, "Pacific Biosciences' Develops Third Generation DNA Sequencing Technology," pp. 1–14, 2008.

[9] M. Meyer and M. Kircher, "Illumina sequencing library preparation for highly multiplexed target capture and sequencing," *Cold Spring Harb. Protoc.*, vol. 5, no. 6, 2010, doi: 10.1101/pdb.prot5448.

[10] 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.

[11] E. E. Schadt, S. Turner, A. Kasarskis, P. Biosciences, W. Road, and M. Park, "A window into third-generation sequencing," vol. 19, no. 2, pp. 227–240, 2010, doi: 10.1093/hmg/ddq416.

[12] F. S. Annexstein, "Generating De Bruijn sequences: An efficient implementation," *IEEE Trans. Comput.*, vol. 46, no. 2, pp. 198–200, 1997, doi: 10.1109/12.565596.

[13] L. Song and L. Florea, "Rcorrector: Efficient and accurate error correction for Illumina RNA-seq reads," *Gigascience*, vol. 4, no. 1, pp. 1–8, 2015, doi: 10.1186/s13742-015-0089-y.

[14] 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.

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

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.