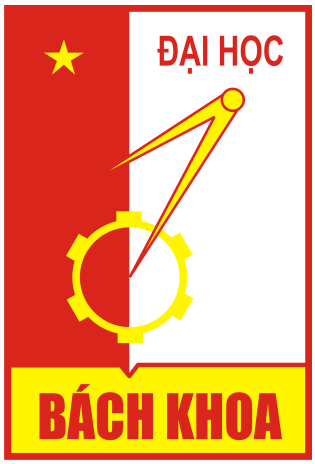
HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY



MACHINE LEARNING AND DATA MINING

Detection of oncogene mutations from cancer gene sequencing data of patients sample

HOANG HUY CHIEN
chien.hh200084@sis.hust.edu.vn

NGUYEN MINH HUYEN
huyen.nm206652@sis.hust.edu.vn

Major: Information Technology
Specialization: Global ICT

Supervisor: Nguyen Nhat Quang _____

Signature

Department: Computer Science

School: School of Information and Communications Technology

HANOI, 06/2023

ABSTRACT

In this project, we tackled the critical challenge of accurately detecting somatic mutations from gene sequencing data, a task crucial in the diagnosis and treatment of cancer. Recognizing the limitations of existing probabilistic and statistical methods, we opted for a deep learning approach, building on the deep convolutional neural networks (CNN) model used by NeuSomatic. Our project centered on enhancing the mutation detection accuracy through the model's training and testing with comprehensive datasets provided by the National Center for Biotechnology Information (NCBI).

The model input comprised candidate somatic mutations represented as a 3-dimensional mutation matrices. Our final model output included a detailed list of identified somatic mutations, with genomic positions, mutation types, and corresponding confidence scores. The use of this deep learning approach significantly improved the precision of somatic mutation detection. We found this method to be more reliable, leading to potentially better diagnosis and personalized treatment outcomes for cancer patients. Further study and applications of this model in real-world conditions hold promise for advancing precision medicine in oncology.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1. INTRODUCTION

1.1 Introduction

Cancer, a leading cause of death globally, owes its incidence and progression largely to somatic mutations - genetic alterations in somatic cells' DNA. These mutations often precipitate uncontrolled cellular growth, resulting in tumor formation. Therefore, the accurate identification of somatic mutations is crucial for effective cancer diagnosis and treatment.

The detection of somatic mutations from cancer gene sequences remains a significant challenge in biomedicine, despite considerable advancements. Most current methods are dependent on probabilistic and statistical formulas, and their effectiveness is restricted to specific datasets. This limited applicability emphasizes the need for more robust and universally applicable methods capable of detecting somatic mutations from a wide range of gene sequencing data.

Deep learning techniques have demonstrated promising potential in various bioinformatics applications in recent years, particularly in genomics for variant detection. Models such as Convolutional Neural Networks (CNNs) can decipher intricate patterns from high-dimensional data, making them potentially well-suited for detecting somatic mutations from gene sequencing data.

This report documents our exploration of deep CNNs' capabilities in the context of somatic mutation detection. Our project builds upon the research of the NeuSomatic model, a deep learning algorithm used for variant detection, by training and testing a reconstructed model using comprehensive datasets provided by the National Center for Biotechnology Information (NCBI). Our goal is to enhance the accuracy of somatic mutation detection, which could subsequently contribute to more effective diagnoses and personalized treatment plans for cancer patients.

The report unfolds as follows: The methodology, inclusive of data collection and preprocessing, model design, training, and evaluation, is detailed in the next section. The subsequent section presents and discusses the results. Finally, we conclude the report and offer directions for future research.

1.2 Problem Description

The overarching problem this project addresses is the accurate and reliable detection of somatic mutations from cancer gene sequencing data. Somatic mutations, which are non-inherited genetic changes in the DNA of somatic cells, are key contributors to cancer development and progression. They can cause normal

cells to turn into cancer cells, leading to uncontrolled cell growth and tumor formation. Therefore, the precise identification and characterization of these mutations are of paramount importance in the diagnosis, prognosis, and treatment of various cancers.

The challenge lies in detecting these somatic mutations effectively from high-dimensional and complex gene sequencing data. The complexity is exacerbated due to several factors:

1. Error in Data: Sequencing data often contain errors, either due to the sequencing process itself or subsequent data processing stages. These errors can lead to false positives (detecting a mutation where there isn't one) or false negatives (failing to detect an existing mutation).
2. Inter-Laboratory Variability: Variability in sequencing depth, coverage, and data processing protocols among different laboratories can lead to inconsistent results and make it challenging to compare results across studies.
3. Diverse Mutational Events: Somatic mutations encompass a range of mutational events, including single nucleotide variants (SNVs), insertions and deletions (INDELs), and more complex structural variants. Each type requires different computational strategies for accurate detection.

Existing mutation detection tools mainly employ probabilistic and statistical methods. While these methods have made substantial progress in variant detection, they have significant limitations when dealing with diverse and complex datasets. Furthermore, these tools often struggle with the generalization of the learned patterns to new, unseen data.

In light of these challenges, this project aims to utilize deep learning techniques, specifically Convolutional Neural Networks (CNNs), to develop a more robust and reliable model for detecting somatic mutations. By leveraging deep learning's ability to learn complex patterns from high-dimensional data, we seek to achieve improved accuracy in mutation detection, ultimately contributing to more effective cancer diagnosis and personalized treatment plans.

1.3 Organization

This report is organized into five main chapters, which collectively present the process of our research, our findings, and their implications.

Chapter 2: Methodology - This chapter describes the comprehensive process used in our study. It begins with Data Collection, where we discuss the origins of our data and how we acquired it. The Data Preparation and Preprocessing section

details the measures taken to clean and organize the data, preparing it for effective use in the model. In the Model Architecture and Implementation section, we delve into the specifics of our model, from the structure to the implementation process. Lastly, we outline the methods we used to assess our model's performance in the Model Evaluation section. This includes the evaluation methods, model training, validation, and testing procedures, as well as the workflow of these steps.

Chapter 3: Results - This chapter presents the findings of our study, detailing the performance of our model when applied to the datasets. The results are depicted in terms of model accuracy, precision, recall, and other evaluation metrics that were employed.

Chapter 4: Discussion - In this chapter, we reflect upon our findings, discussing their implications and comparing the performance of our model to other similar models used in the field of genetic variant detection. We highlight the strengths and weaknesses of our approach and model, as well as potential improvements and future directions for our research.

Chapter 5: Summary - The final chapter serves as a concise wrap-up of the entire report. It provides a succinct overview of our research process, main findings, their significance, and potential implications. It brings together the central points from the previous chapters and concludes with insights for future research and improvements.

The systematic organization of this report ensures clarity and aids in understanding the entire process of our study - from methodology to results and discussions.

CHAPTER 2. METHODOLOGY

2.1 Methodology

The methodology used for this project follows several key steps, including data preprocessing, model design and implementation, and model evaluation.

2.1.1 Dataset

a, Data Collection

This project employs two primary BAM (Binary Alignment/Map) datasets, both possessing unique characteristics vital for the robust evaluation and training of our deep learning model focused on somatic mutation detection.

The first dataset encompasses Whole Exome Sequencing (WES) data derived from the gene sequencing of patient HCC1395. Significantly, this data originates from six distinct laboratories, including Illumina (IL), National Cancer Institute (NC), Novartis (NV), European Infrastructure for Translational Medicine (EA), Fudan University (FD), and Loma Linda University (LL). Each laboratory's data is characterized by varying levels of depth, coverage, and accuracy relative to standard label files, reflecting the practical differences and inconsistencies that can occur in real-world genetic sequencing. These variations present an opportunity to build a model that can accommodate and learn from the inherent complexities and discrepancies of multi-source data.

The secondary dataset is a synthetic compilation of data from all six laboratories mentioned earlier. This meticulously engineered dataset houses known somatic mutations and spans a spectrum of allele frequencies, sequencing depths, and tumor purity levels. Serving as a reliable ground truth for our project, this dataset facilitates a confident selection of high-reliability features during the model training and validation stages.

The combination of both the real-world and synthetic BAM datasets ensures a comprehensive representation of data scenarios, thereby assisting in the creation and validation of an effective and robust model for the accurate detection of somatic mutations.

b, Data Preparation

Our dataset is meticulously partitioned into a training set and a test set. The training set primarily consists of all candidate Single Nucleotide Variants (SNVs) and Insertion and Deletion (INDEL) instances identified within the Variant Call Format (VCF) file. However, any instances found within a select group of 50

genes are specifically excluded, as visually depicted in the corresponding table. To ensure a balanced training dataset, we incorporate non-somatic instances — data points that are neither INDEL nor SNV variants, randomly selected from the larger pool of available instances. For each candidate SNV or INDEL, a single non-somatic instance is chosen through an undersampling strategy, which allows us to manage the high prevalence of non-somatic data effectively.

The test set is constructed solely around the data corresponding to these 50 genes. For each candidate SNV or INDEL in this subset, 2 non-somatic instances are randomly chosen and labelled as non-somatic data. Again, these non-somatic instances are selected through undersampling, in an effort to mimic, to a certain extent, the composition of real-world data within the test set. While it's true that the number of non-somatic variant candidates often greatly surpasses the count of somatic variants in reality, our strategic partitioning and careful undersampling methodology assure that our model maintains robustness and balance. In this way, we have effectively addressed the challenge posed by the preponderance of non-somatic data, ensuring our model can manage diverse mutation scenarios and retain data balance.

ABL1	AKT1	ALK	APC	ATM	BRAF	CDH1	CDKN2A	CSF1R	CTNNB1
EGFR	ERBB2	ERBB4	EZH2	FBXW7	FGFR1	FGFR2	FGFR3	FLT3	GNA11
GNAQ	GNAS	HNF1A	HRAS	IDH1	IDH2	JAK2	JAK3	KDR	KIT
KRAS	MET	MLH1	MPL	NOTCH1	NPM1	NRAS	PDGFRA	PIK3CA	PTEN
PTPN11	RB1	RET	SMAD4	SMARCB1	SMO	SRC	STK11	TP53	VHL

Table 2.1: List of 50 genes

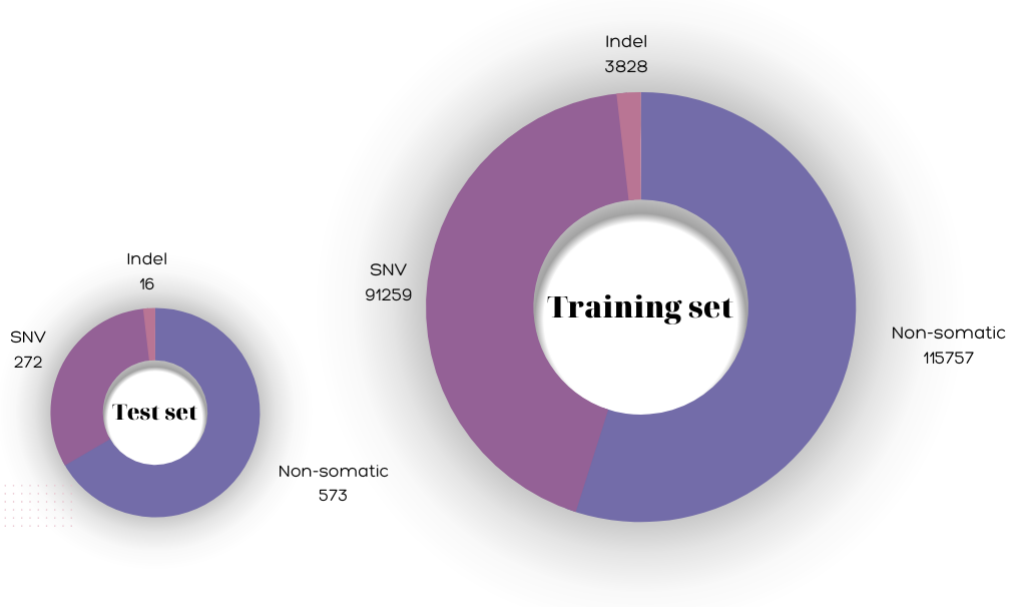
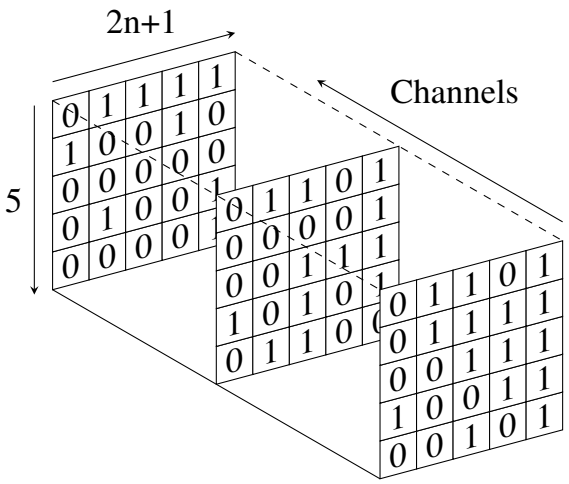


Figure 2.1: The number of samples in training set and test set

c, Data Preprocessing

Data preprocessing is a critical stage in this study, ensuring that the high-quality sequencing data from different laboratories are consistently prepared for the Convolutional Neural Networks (CNN) model. The preprocessing was conducted as follows:

1. **Data Cleaning and Read Extraction:** The preprocessing commenced with the data cleaning and read extraction stage. The raw sequencing data was carefully evaluated to identify and correct any inconsistencies or errors. This process involved scanning the Sequence Alignment/Map (SAM) files to extract the aligned reads and the associated information, including the genomic position, the cigar string, the sequence of the read, and the base quality scores. Reads with potential sequencing errors or not in exome region were excluded, ensuring that any low-quality reads based on the mapping quality scores were appropriately managed.
2. **Data Integration and Mutation Identification:** This phase involved the harmonization of the gene sequencing data from different laboratories, followed by mutation identification. To ensure the datasets were consistent, sequences were aligned to a common reference genome, and discrepancies between datasets were resolved. The cigar strings and the alignments were scanned to identify candidate somatic mutations, focusing on Single Nucleotide Variants (SNVs) and Insertions/Deletions (INDELs), for further analysis.
3. **Data Transformation and Matrix Construction:** Following data integration, the data was transformed into a suitable format for the CNN model. For each identified candidate mutation, a 3-dimensional mutation matrix of size k×5×(2n+1) was constructed, which represented the local genomic context around the mutation. The five rows corresponded to the four nucleotides (A, T, G, C) and ”-” for an empty base. The columns represented a window of nucleotides around the mutation, with ’n’ nucleotides on either side of the candidate mutation, thus having 2n+1 columns in total. ’k’ channels included the reference channel, tumor frequency channels, and quality channels, among others.



- The reference channel counts the number of bases on the reference segment after alignment.
- The tumor block frequency channel counts the frequency of bases present on the reading segments at specific positions.
- The tumor block quality channel calculates the total quality of the bases present on the reading segments at a certain position.

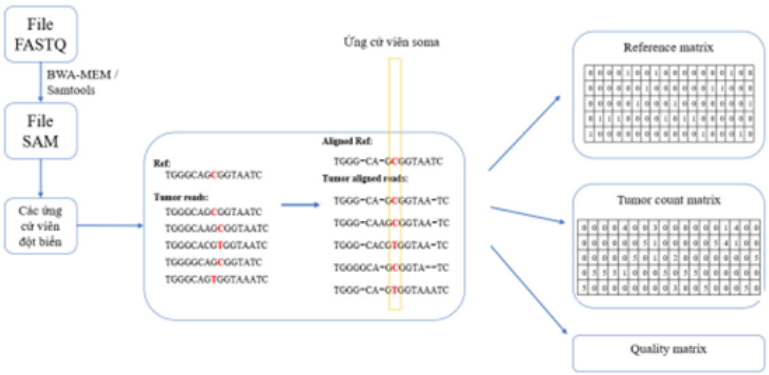


Figure 2.2: Alignment and Construction input matrix for model

Furthermore, before feeding the data into the model, normalization of the matrix values was performed to ensure that all input features were on the same scale, significantly influencing the CNN model's performance. This was typically achieved by scaling the values between 0 and 1.

4. **Data Reduction:** Owing to the substantial volume of high-throughput sequencing data, it was necessary to implement a data reduction step. This involved a targeted selection process, focusing specifically on the pertinent information related to each mutation's position as identified in our ground truth data, within a comparatively confined analytical window. This streamlined approach to data reduction not only condensed the model's input data

but also ensured the preservation of crucial positional context information. This is critical as it influences the mutation's effect and is thus, central to our analytical process.

This comprehensive data preprocessing protocol assured that the raw sequencing data was accurately cleaned, integrated, transformed, and reduced, ready for input into the deep learning model. The result was high-quality, consistent data, enhancing the reliability and effectiveness of the subsequent mutation detection process.

2.1.2 Model Architecture and Implementation

The next step involves the design and implementation of our deep learning model. In this research, our focus was on the application of the NeuSomatic model, a prominent multi-layer deep learning model proposed in the scientific paper "Deep convolutional neural networks for accurate somatic mutation detection". This model has been trained on large datasets to learn how to classify variants. With the implementation of deep learning in tackling the problem of detecting somatic variants, the NeuSomatic model has demonstrated remarkable efficiency compared to traditional methods. This opens up numerous opportunities for applications in the field of genetics and molecular biology. However, the application of these tools and algorithms must be executed with caution to ensure accurate and reliable analysis results.

Our model architecture is lighter compared to the original NeuSomatic network and focuses solely on variant type classification.

a, Structure

The input matrix is first fed into the initial convolutional layer with three output channels, a sliding window size of 1x3, and the ReLu activation function. This is followed by a normalization layer and a max pooling layer. The output from this layer is then passed on to a set of four blocks that use uniform connections (Residual Block) similar to the ResNet architecture.

Each of these blocks includes a convolutional layer with a sliding window size of 3x3, followed by a normalization layer and another convolutional layer with a sliding window size of 5x5. Between these blocks are normalization layers and max pooling layers.

The output of the final block is passed through a Flatten layer to straighten it out, and then through a Fully Connected layer. This is considered the feature vector of the input data, which will be used for the purpose of classifying variant types.

Specifically, this feature vector is sent to a Softmax classifier with the aim of classifying three types of variants: Single Nucleotide Variant (SNV), Insertion/Deletion (INDEL), and Non-Somatic.

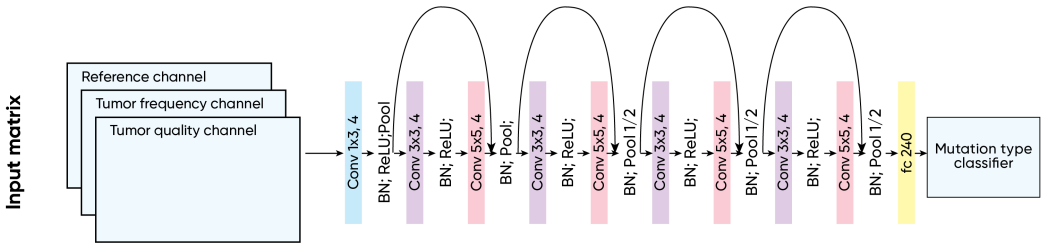


Figure 2.3: CNN model's architecture

b, Functionality

- **Initial Convolutional Layer:** This is the first step where the input matrix is processed. This layer uses a sliding window size of 1x3 and applies a convolution operation on the input. It has four output channels, meaning it will learn and produce four different feature maps. This increases the depth of the data while keeping its spatial dimensions the same. The purpose of this layer is to learn local features in the input matrix, like edges or small patterns, which helps in variant detection.
- **ReLu Activation Function:** This layer introduces non-linearity into the model, enabling it to learn complex patterns. The Rectified Linear Unit (ReLu) is applied element-wise and replaces all negative pixel values in the feature map with zero, thus accelerating the training process.
- **Normalization Layer:** This layer performs normalization which helps to reduce overfitting, and ensures that the weights are on a similar scale. This aids in improving the stability of the neural network, and accelerates its training.
- **Max Pooling Layer:** This is a down-sampling operation that reduces the dimensionality of the input while retaining important information. It allows for assumptions to be made about features contained in the sub-regions binned.
- **Residual Blocks:** The heart of the model consists of four blocks that follow the ResNet architecture design, utilizing skip connections or shortcuts to jump over some layers. Each block has two convolutional layers - one with a sliding

window size of 3x3, and the other with a 5x5 window. These layers aim to learn more complex and abstract features from the input data. Interspersed with normalization layers, these blocks aim to avoid the vanishing gradient problem and to make deeper networks easier to optimize.

2.1.3 Model Evaluation

a, Training and Validation

The model was trained and validated on the Genome Wide Exome Sequencing datasets with various window sizes. Training of the model was monitored using both training and validation losses, accuracy, and F1-scores. The balanced weights were calculated to handle class imbalance and were used in the cross-entropy loss function during the training of the model. The learning rate was initially set to 0.001 and was decreased after every epoch using a learning rate scheduler to help the model converge to a better solution. Early stopping was also implemented to prevent overfitting, where the training process was halted if there was no significant decrease in the validation loss over five consecutive epochs. The model was trained for a maximum of 100 epochs.

The performance of the model on the training and validation set was logged into the Weights and Biases (wandb) platform, allowing real-time monitoring of the training process and simplifying hyperparameter tuning. The model achieving the highest F1-score on the validation set was saved for testing.

b, Testing

The performance of the model was evaluated on a separate testing set, unseen during the training and validation phase. The testing phase was carried out on two checkpoints - the one corresponding to the best validation F1-score and the final model at the last epoch.

Accuracy was used as the primary metric for the evaluation of the model's performance. We calculated the count of true predictions for each class and the total count of each class in the test dataset, which enabled us to compute the overall accuracy and the accuracy for each class individually.

Additionally, a confusion matrix was used to provide a visualization of the model's performance. The confusion matrix displays the counts of true positive, false positive, true negative, and false negative predictions for each class in the test dataset. The matrix allowed us to understand the model's strengths and weaknesses in predicting different classes.

Finally, a detailed classification report was generated using Scikit-learn's classi-

fication_report function, providing a comprehensive summary of the model's performance. This report included precision, recall, F1-score, and support for each class and also provided overall averages for these metrics.

Overall, the model evaluation was robust and meticulous, using various measures to ensure the model's performance is accurately assessed and understood. This rigorous testing procedure helped identify the model's performance not only on the whole dataset but also in predicting individual classes, thus enabling targeted improvements in future iterations.

2.2 Workflow

2.2.1 Evaluation Method

A convolutional neural network (CNN) was used in this study, specifically the NeuSomaticNet model, for mutation detection in genomic data. The data was divided into three sets: training, validation, and testing. Data splitting was performed using the "train_test_split" function of the sklearn library, where the training dataset was further divided into a training set and a validation set.

2.2.2 Model Training

Before training the model, data was loaded and preprocessed. The data consisted of labeled genomic sequences derived from various chromosomes. These sequences were then normalized to ensure that no single feature would dominate others during training. Class imbalance was also addressed by calculating class weights which were then used in our loss function. These weights were inversely proportional to the class frequencies. The higher the class frequency, the lower the weight and vice versa.

Our model underwent training via the Adam optimization algorithm, starting with an initial learning rate of 0.001. To address potential class imbalance in our training dataset, we calculated class weights according to the class distribution and incorporated these into the CrossEntropyLoss function. Training took place across a maximum of 100 epochs, during which each batch of training data was used to adjust the model parameters. By calculating the loss with the Cross-Entropy Loss function and applying backpropagation, we continually updated the model parameters to minimize this loss, resulting in progressive enhancement of the model's performance.

To avoid overfitting and enhance generalization capabilities, we used a batch size of 32. This setting allows the model to estimate the error gradient more accurately compared to using the entire dataset. From the 31st epoch, we implemented

a learning rate decay strategy, decreasing the learning rate by a factor of 0.9 with every subsequent epoch. This balance between exploration and exploitation in the model's parameter space aids in finding a more optimal solution during the training process. To ensure the model was exposed to a diverse set of samples each epoch, the training data was shuffled after every epoch. These settings were encapsulated in a dictionary named params.

2.2.3 Model Validation and Early Stopping

Model validation was conducted after each training epoch. During training, an early stopping condition was implemented, where the training stopped if the absolute difference between the validation loss and the previous loss was less than or equal to a defined epsilon, it was assumed that the model was no longer learning, and a counter was incremented. . If this situation occurred for a specified number of consecutive epochs (early_stop), the training was terminated prematurely.

At each epoch, the performance metrics - training accuracy, validation accuracy, and F1-score - were calculated and logged using the weights and biases tool (wandb). The model with the smallest validation loss was then chosen as the best model.

2.2.4 Model Evaluation

Model performance was evaluated using the F1-score. This metric, being the harmonic mean of precision and recall, offers a more balanced measure of a model's performance than accuracy, particularly in scenarios with imbalanced datasets. After each epoch, the F1-score was computed on the validation dataset and logged.

2.2.5 Testing

Upon completion of training, the model was evaluated on the test set, which had not been seen by the model during the training process. Two checkpoints were evaluated - the model at the final epoch and the model that achieved the best F1-score during validation. For both checkpoints, the performance metrics such as accuracy and F1-score were calculated, and a detailed classification report was generated using sklearn's metrics module

3.1 Results

3.1.1 Window size

a, Training model with data from all labs

Our experimental process involved training and testing our model across a variety of lab datasets, namely WES_EA_T_1, WES_FD_T_3, WES_IL_T_2, WES_LL_T_2, WES_NC_T_3, and WES_NV_T_1. For these datasets, we set the learning rate at 0.001. Using the slice window width to dictate the size of the input matrix, we conducted experiments on two window values: 10, 16 and , of which 16 is the window size chosen in the NeuSomatic study.

Figures below displays the results of the training and evaluation process for the model with window size 10 on the dataset.

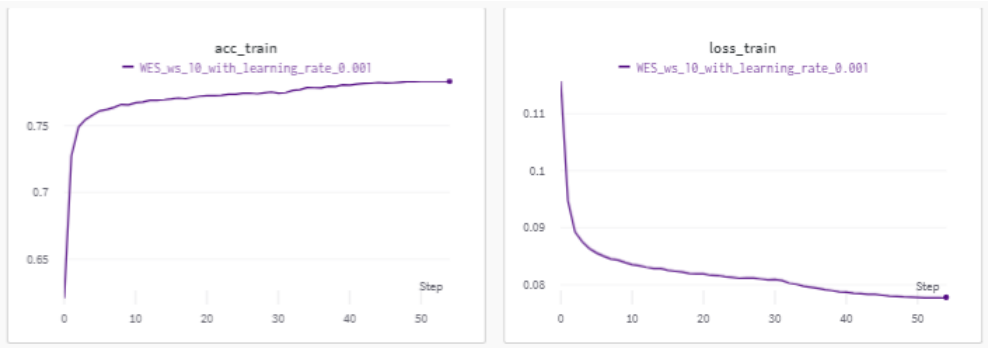


Figure 3.1: The training process with window size 10

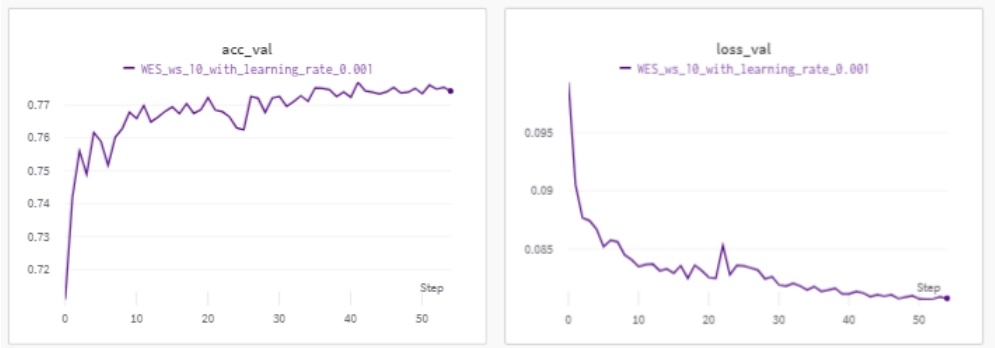


Figure 3.2: The evaluation process with window size 10

The result of testing model with test dataset

```
****** Testing model with best checkpoint******
[502. 237.  11.] [632. 293.  16.] [0.7943038  0.80887372 0.6875    ] 0.7970244420828906
              precision    recall  f1-score   support

           0       0.89      0.79      0.84       632
           1       0.67      0.81      0.73       293
           2       0.48      0.69      0.56        16

    accuracy                           0.80       941
   macro avg       0.68      0.76      0.71       941
weighted avg       0.81      0.80      0.80       941
```

Figure 3.3: The result of test dataset with window size 10

Figures below displays the results of the training and evaluation process for the model with window size 16 on the dataset.

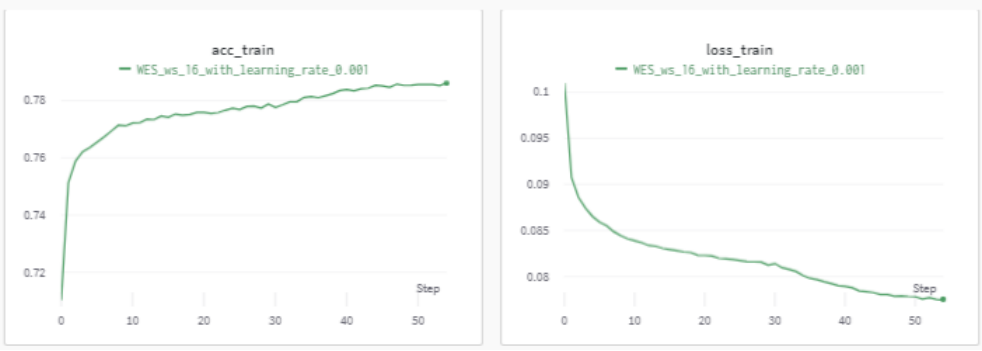


Figure 3.4: The training process with window size 16

Figure 3.5: The evaluation process with window size 16

The result of testing model with test dataset

```
****** Testing model with best checkpoint******
[478. 237.  11.] [597. 283.  16.] [0.80067002 0.83745583 0.6875    ] 0.8102678571428571
              precision    recall  f1-score   support

           0       0.90      0.80      0.85       597
           1       0.68      0.84      0.75       283
           2       0.65      0.69      0.67        16

    accuracy                           0.81       896
   macro avg       0.74      0.78      0.75       896
weighted avg       0.83      0.81      0.81       896
```

Figure 3.6: The result of test dataset with window size 16

Figures below displays the results of the training and evaluation process for the model with window size 20 on the dataset.

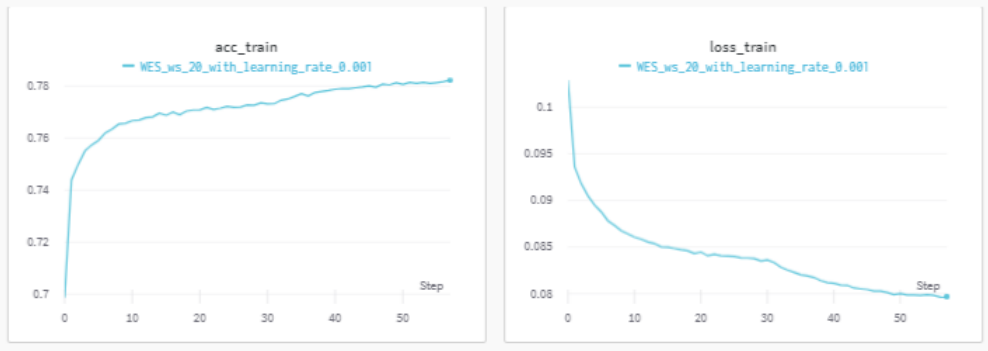


Figure 3.7: The training process with window size 20

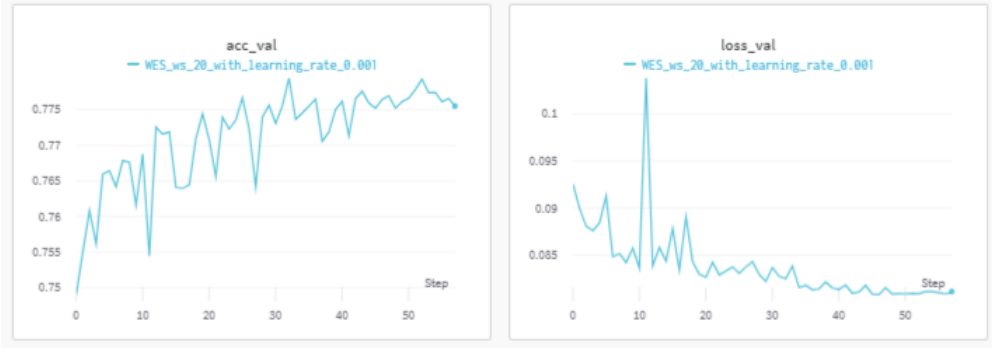


Figure 3.8: The evaluation process with window size 20

The result of testing model with test dataset

```
****** Testing model with best checkpoint******
[463. 227.  13.] [573. 272.  16.] [0.80802792 0.83455882 0.8125    ] 0.8164924506387921
              precision    recall  f1-score   support

           0       0.91      0.81      0.85       573
           1       0.69      0.83      0.75       272
           2       0.65      0.81      0.72        16

    accuracy                           0.82       861
   macro avg       0.75      0.82      0.78       861
weighted avg       0.83      0.82      0.82       861
```

Figure 3.9: The result of test dataset with window size 20

Figures below displays the comparation of results of the training and evaluation process for the model

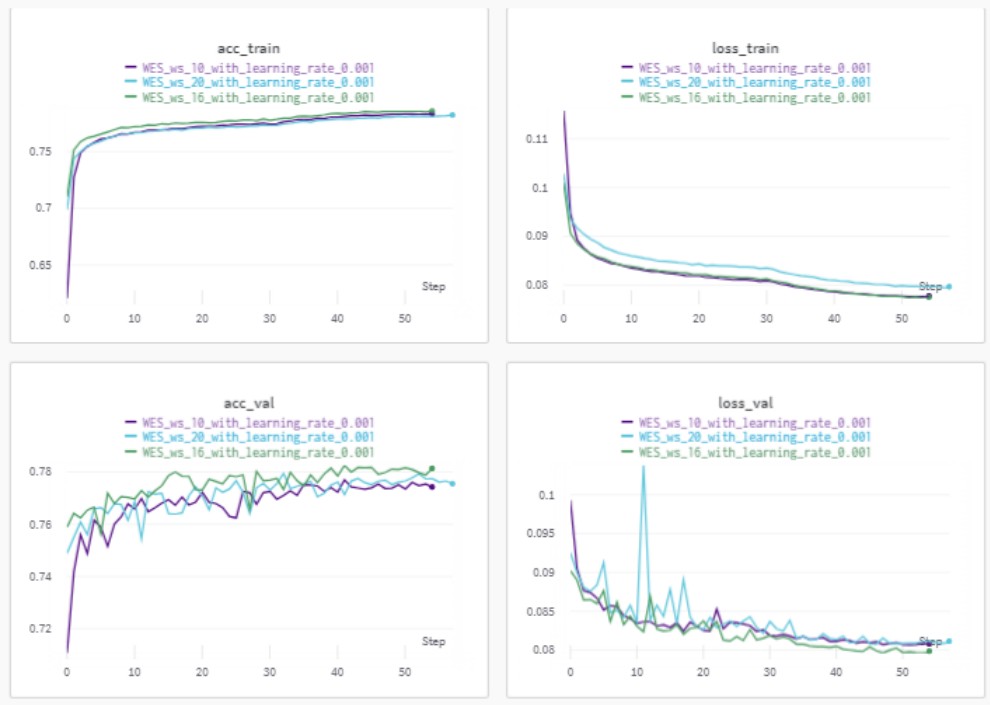


Figure 3.10: The training process and evaluation process comparation

In our experiment, we demonstrated the impact of varying the window size on the performance of our deep learning model in genomic variant classification. We trained and tested our model using three different window sizes: 10, 16, and 20. The experiment results showed a noteworthy trend. The model that was trained with a window size of 16 demonstrated slightly improved performance in comparison to the models trained with window sizes of 10 and 20.

This observation suggests that the choice of window size, an integral aspect of data preparation, can have a meaningful effect on the performance of genomic classification models. Particularly, a mid-sized window of 16 seems to offer a balanced context to effectively capture the pertinent features for genomic variant classification, as opposed to smaller or larger windows.

b, Training model with data from WES_IL_T_2

Following this, we focused on the WES_IL_T_2 dataset to conduct additional training and testing. We employed three distinct window sizes for this dataset: 10, 16, and 20, to examine how these variations affected the model's performance.

In our quest to optimize the model, we also experimented with tuning the learning rate. We were interested in identifying the learning rate that would yield the best results for the WES_IL_T_2 dataset. After thorough exploration, we established that a learning rate of 0.0003 seemed to produce the most favorable outcome for this dataset. Through this process, we have developed a flexible model that can adjust to different datasets and parameters to ensure optimal performance.

Figures below displays the results of the training and evaluation process for the model with window size 10 on the dataset.

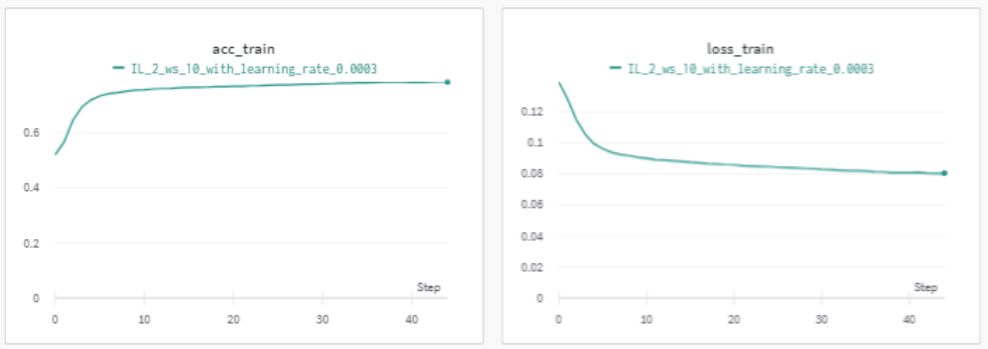


Figure 3.11: The training process with window size 10

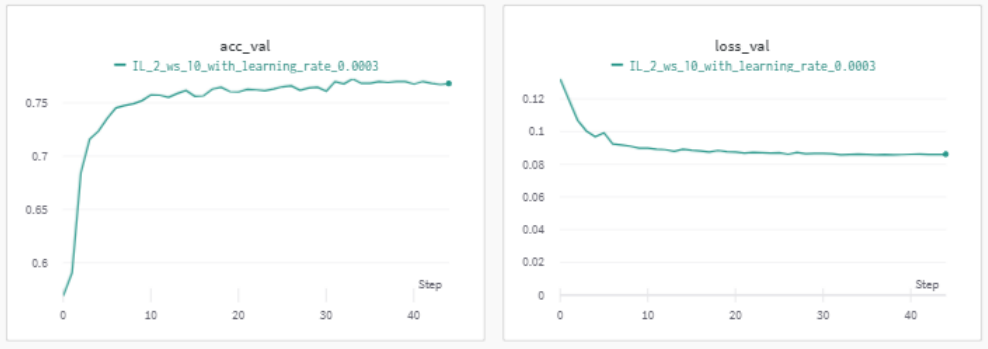


Figure 3.12: The evaluation process with window size 10

The result of testing model with test dataset

```
****** Testing model with best checkpoint******
[104.  59.   4.] [141.  67.   5.] [0.73758865 0.88059701 0.8       ] 0.784037558685446
              precision    recall  f1-score   support

           0       0.92      0.74      0.82       141
           1       0.63      0.88      0.74        67
           2       0.57      0.80      0.67         5

    accuracy                           0.78       213
   macro avg       0.71      0.81      0.74       213
weighted avg       0.82      0.78      0.79       213
```

Figure 3.13: The result of test dataset with window size 10

Figures below displays the results of the training and evaluation process for the model with window size 16 on the dataset.

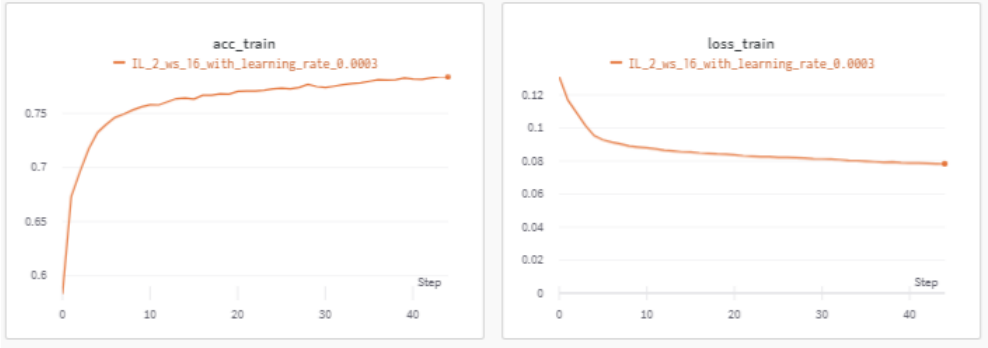


Figure 3.14: The training process with window size 16

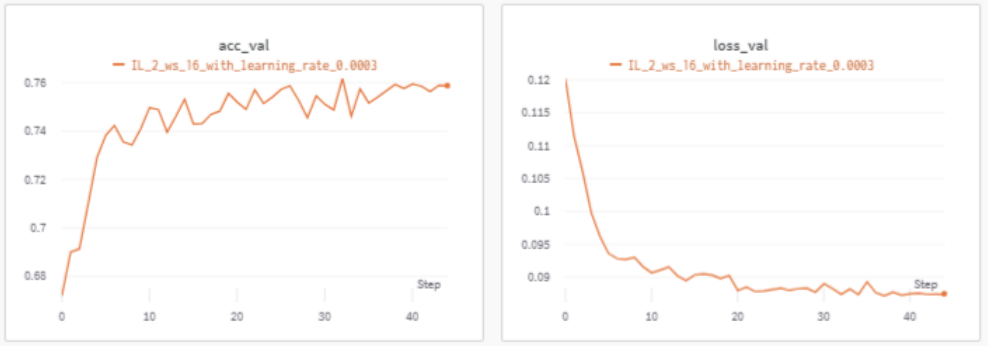


Figure 3.15: The evaluation process with window size 16

The result of testing model with test dataset

```
****** Testing model with best checkpoint******
[110.  53.   1.] [137.  67.   5.] [0.80291971 0.79104478 0.2       ] 0.784688995215311
              precision    recall  f1-score   support

           0       0.86      0.80      0.83       137
           1       0.67      0.79      0.73        67
           2       0.50      0.20      0.29         5

    accuracy                           0.78       209
   macro avg       0.68      0.60      0.61       209
weighted avg       0.79      0.78      0.78       209
```

Figure 3.16: The result of test dataset with window size 16

Figures below displays the results of the training and evaluation process for the model with window size 20 on the dataset.

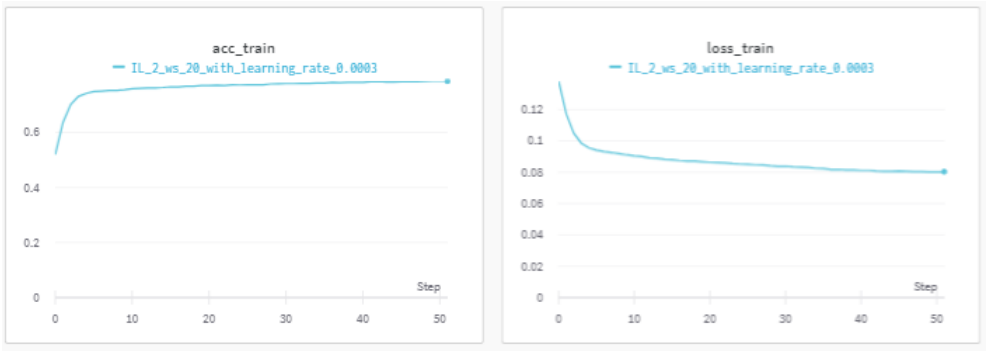


Figure 3.17: The training process with window size 20

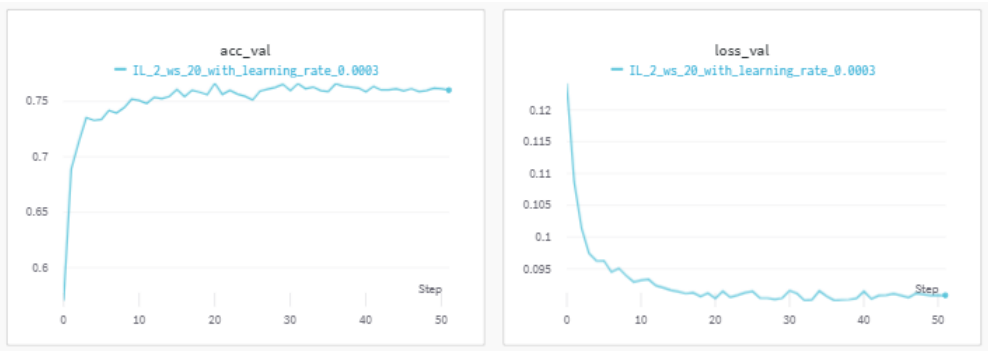


Figure 3.18: The evaluation process with window size 20

The result of testing model with test dataset

```
****** Testing model with best checkpoint******
[104.  54.   4.] [131.  65.   5.] [0.79389313 0.83076923 0.8       ] 0.8059701492537313
              precision    recall  f1-score   support

           0       0.90      0.79      0.84       131
           1       0.67      0.83      0.74        65
           2       1.00      0.80      0.89         5

    accuracy                           0.81       201
   macro avg       0.85      0.81      0.82       201
weighted avg       0.82      0.81      0.81       201
```

Figure 3.19: The result of test dataset with window size 20

Figures below displays the comparation of results of the training and evaluation process for the model

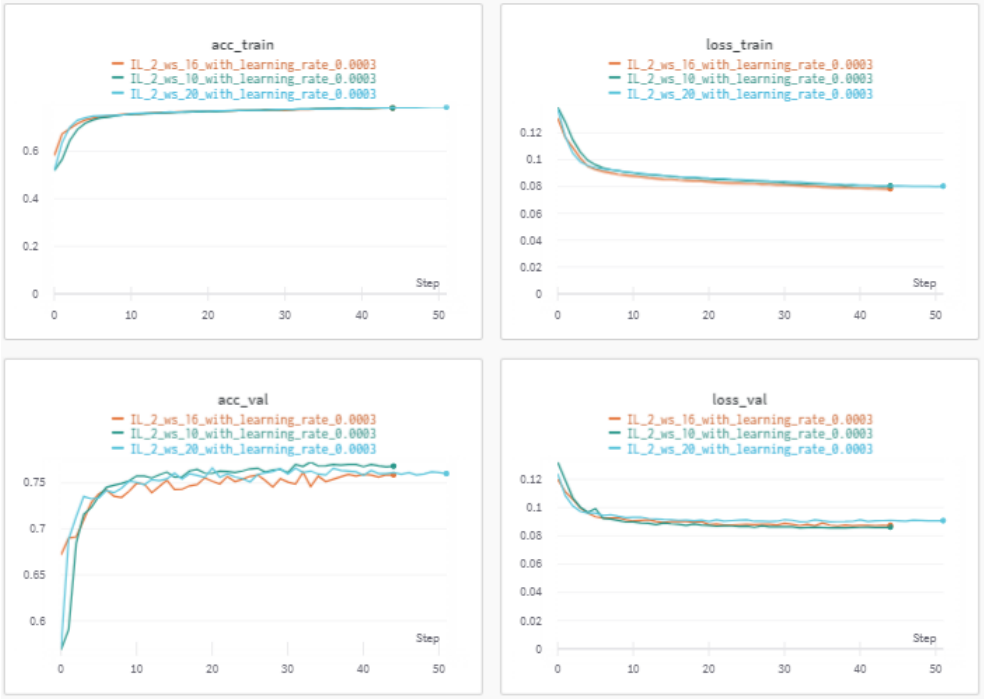


Figure 3.20: The training process and evaluation process comparation

Following our comprehensive model training and testing across varying parameters, it was observed that a window size of 10 yielded the most superior results. This outcome underscores the significance of optimal parameter selection in machine learning models and presents the window size of 10 as an effective choice for this specific task. It's an intriguing finding that directs future research and applications of our model, and further emphasizes the model's adaptability to different conditions.

3.1.2 Learning rate

Upon establishing that the optimal window size for our genomic classification model with data from all laboratories was 16, we proceeded to test the effects of varying learning rates on the model's performance. We experimented with different initial learning rates including 0.0001, 0.0005, 0.001, and 0.0015, observing

how each setting impacted the ability of our model to classify genomic variants accurately.

In deep learning, the learning rate is a critical hyperparameter that controls the adjustments made to a model's weights with the aim of reducing the loss. A high learning rate can lead to volatile training and the possibility of missing the optimal solution, while a low learning rate, although ensuring more steady progress, may take longer to converge or get trapped in a local minimum.

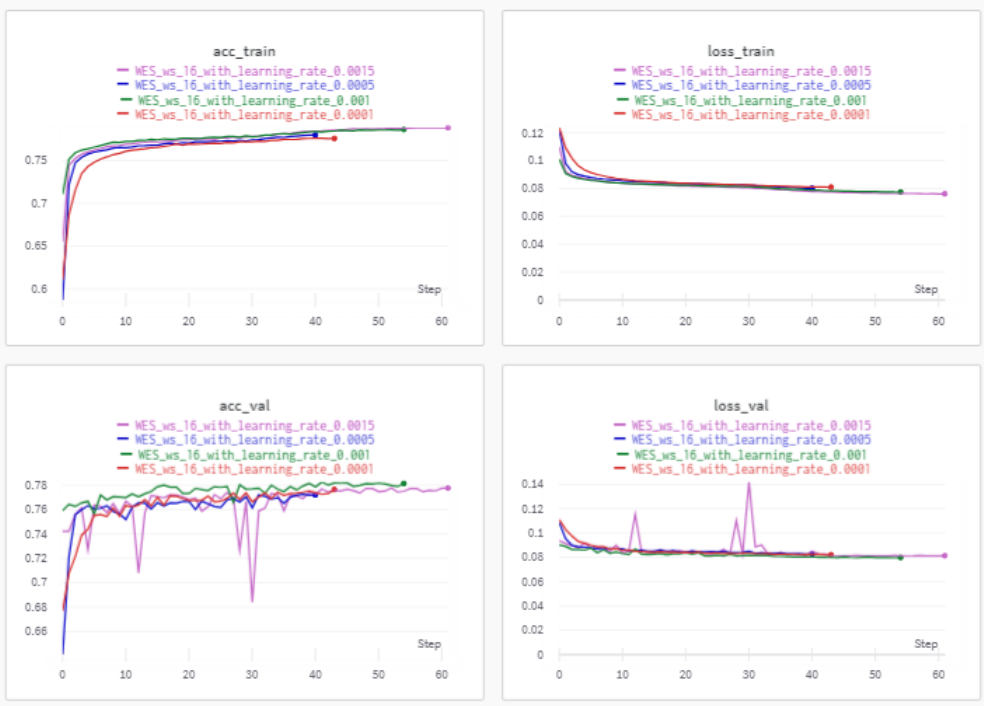


Figure 3.21: The comparation result with different learning rate

Our experiment indicated that a learning rate of 0.001 delivered the best performance among the ones we tested. It provided a balance between the speed of convergence and the stability of learning, ensuring that our model learned effectively from the data without overshooting the optimal weights. This demonstrates the importance of judiciously selecting the learning rate, which is instrumental in achieving optimal model performance.

These results underscore the importance of careful hyperparameter tuning in machine learning model development. Although further experiments with more finely grained learning rate values or other optimization strategies could potentially yield even better results, the learning rate of 0.001 proved to be a good starting point for our specific task. This valuable insight will guide our future endeavors in developing robust and efficient models for genomic variant classification.

3.1.3 ”Dying ReLU” problem

ReLU (Rectified Linear Unit) activation function is widely used in Convolutional Neural Networks (CNNs) and other types of neural networks due to several reasons:

1. **Non-linearity:** ReLU introduces non-linearity into the network, which is critical because it allows the network to learn complex patterns. Without non-linearity, no matter how many layers the network has, it would still behave like a single-layer network.
2. **Computational Efficiency:** ReLU is computationally efficient compared to other activation functions like sigmoid or tanh. This is because ReLU simply applies a threshold at zero, and any input value below zero is made zero, whereas other activation functions involve more complex mathematical operations.
3. **Sparsity:** ReLU produces sparse activations; for any given layer, some neurons output zero and others output positive numbers. This can make the network more efficient and easier to train because the resulting sparse matrix of weights is easier to update and store.
4. **Avoidance of Vanishing Gradient Problem:** Unlike sigmoid or tanh activation functions, the ReLU function does not saturate in the positive direction and the gradient is constant (equal to 1), which helps to mitigate the vanishing gradient problem that can occur during backpropagation, particularly in deep networks.

However, ReLU is not without its drawbacks, such as the ”dying ReLU” problem where neurons can become stuck and only output zero values, no matter the input.

```
****** Testing model with best checkpoint******
[112.  51.   0.] [137.  67.   5.] [0.81751825 0.76119403 0.        ] 0.7799043062200957
              precision    recall  f1-score   support

           0       0.84      0.82      0.83       137
           1       0.68      0.76      0.72        67
           2       0.00      0.00      0.00         5

    accuracy                           0.78       209
   macro avg       0.51      0.53      0.52       209
weighted avg       0.77      0.78      0.77       209
```

Figure 3.22: The best-checkpoint model’s result of WES_IL_T_2 test dataset with window size 16 and learning rate 0.0003

```
****** Testing model with last checkpoint******
[111.  51.   0.] [137.  67.   5.] [0.81021898 0.76119403 0.        ] 0.7751196172248804
              precision    recall  f1-score   support

           0       0.84      0.81      0.83       137
           1       0.67      0.76      0.71        67
           2       0.00      0.00      0.00         5

    accuracy                           0.78       209
   macro avg       0.50      0.52      0.51       209
weighted avg       0.77      0.78      0.77       209
```

Figure 3.23: The last-checkpoint model's result of WES_IL_T_2 test dataset with window size 16 and learning rate 0.0003

As we embarked on training our model using different window sizes and learning rates, we encountered an issue that became evident in the model's performance, as depicted in the preceding figures. The consistent prediction of zero for INDEL values led us to suspect the cause to be the phenomenon known as "dying ReLU."

ReLU, our model's activation function, can encounter a problem where neurons effectively "die," consistently outputting zero irrespective of the input. This could be a likely explanation for our model's unexpected behavior, where it seemed unable to capture and learn from the nuanced patterns associated with non-zero INDEL values.

Recognizing the need to address this potential shortcoming, we considered the implementation of ReLU variants designed to alleviate the dying ReLU problem. These variants, such as Leaky ReLU or Parametric ReLU, introduce minor modifications to the ReLU function to keep neurons activated and avoid their premature "death."

Ultimately, in our specific case, we chose to employ LeakyReLU as our activation function. This choice aims to mitigate the suspected issue and enhance the learning and predictive capability of our model by preventing neurons from falling into the "dying" state, thereby better equipping our model to handle diverse genomic patterns.

```
****** Testing model with best checkpoint******
[110.  53.   1.] [137.  67.   5.] [0.80291971 0.79104478 0.2       ] 0.784688995215311
              precision    recall  f1-score   support

           0       0.86      0.80      0.83       137
           1       0.67      0.79      0.73        67
           2       0.50      0.20      0.29         5

    accuracy                           0.78       209
   macro avg       0.68      0.60      0.61       209
weighted avg       0.79      0.78      0.78       209
```

Figure 3.24: The best-checkpoint model's result with LeakyReLU activation function

```
****** Testing model with last checkpoint******
[110.  54.   1.] [137.  67.   5.] [0.80291971 0.80597015 0.2       ] 0.7894736842105263
              precision    recall  f1-score   support

           0       0.87      0.80      0.83       137
           1       0.68      0.81      0.73        67
           2       0.50      0.20      0.29         5

    accuracy                           0.79       209
   macro avg       0.68      0.60      0.62       209
weighted avg       0.80      0.79      0.79       209
```

Figure 3.25: The last-checkpoint model's result with LeakyReLU activation function

3.1.4 Run tool for unseen data

Building upon the success of our trained model, we've extended our work to include a comprehensive tool designed to handle data without any pre-existing labels. This feature is especially crucial in real-world scenarios where we often encounter unseen data sets for which no ground truth information is available. Rather than being a limitation, this is precisely where our model shines. By leveraging the knowledge it has gained through training, our model can make predictions about this unlabeled data, making it a powerful tool for generating insights from raw, unprocessed genomic sequences. This not only widens the applicability of our model but also provides an opportunity for future exploration and continued learning in the complex and ever-evolving field of genomics.

The tool that we have developed has the capability to handle raw genomic data in the form of a BAM file. It is specifically designed to intake this type of file format, which is commonly used in genomics for storing sequence data. Upon receiving a BAM file, the tool proceeds by scanning for any identifiable mutations within the data. Each detected mutation is then converted into a unique input matrix that is compatible with the requirements of our pre-trained model.

Once these input matrices are generated, they are systematically fed into our model. This is where the complex computations occur, with our model processing the matrices and making predictions based on its prior learning.

Finally, the model's predictions are neatly organized and exported to a CSV file. This format was chosen for its widespread use and compatibility with a broad range of applications. In this way, we not only provide a practical tool for interpreting raw data but also ensure that the results are presented in a manner that is both user-friendly and suitable for further analysis.

Our run tool has two modes. The first one is calling all the variants that are found in the file along with their types and the second mode only specifies high confidence variants (statistic that the variant belong to the predicted type is above

0.8). Example results of a BAM file is shown as follow.

output > sample_081_200_out.csv

```
candidate_position,type_somatic,statiscal
136503391,indel,0.52033746
130874997,snv,0.5416458
10142129,indel,0.7750317
114709634,indel,0.8275241
116699580,snv,0.8222616
87961014,indel,0.9312813
28034218,snv,0.538855
179198997,indel,0.53068256
179199003,snv,0.51935565
112840243,snv,0.54767627
129211763,snv,0.5699517
37025814,snv,0.5775079
112840083,snv,0.5407839
116700197,snv,0.5088383
10149774,snv,0.6728111
28034134,snv,0.665174
121515246,snv,0.7818173
7676632,snv,0.49535027
7674948,indel,0.39076978
55191888,indel,0.92918396
121498497,indel,0.4652055
```

Figure 3.26: Run tool's result of all variants found in file

output > high_confidence_sample_081_200_out.csv

```
candidate_position,type_somatic,statiscal
48459720,indel,0.9500277
39724133,indel,0.93374187
116699580,snv,0.8222616
55087635,indel,0.8753817
114709634,indel,0.8275241
87961014,indel,0.9312813
55191887,indel,0.9738695
1806167,snv,0.9157517
55143368,indel,0.8013169
108301794,snv,0.8455364
55181434,snv,0.9428293
136504979,indel,0.83996314
55191888,indel,0.92918396
43120212,snv,0.87381786
116699628,indel,0.979626
39891641,indel,0.8970894
43120213,snv,0.8761445
55087638,indel,0.99524385
129206584,indel,0.9154474
55191889,indel,0.84909564
```

Figure 3.27: Run tool's result of high confidence variants found in file

CHAPTER 4. DISCUSSION

In the course of our research, we have developed a Deep Neural Network model for detecting genetic variants in cancer patients, based on the NeuSomatic model architecture as proposed in the scientific paper "Deep convolutional neural networks for accurate somatic mutation detection". Notably, a majority of the relevant research on variant detection either do not disclose their source code or they involve preprocessing data using uncommon libraries and less popular languages in the field of machine learning, such as C/C++. This poses a significant challenge to modifications and improvements of their models.

In our current model, we've implemented a more accessible approach to aligning read segments, leveraging Python - a robust language for machine learning - to facilitate easy adjustments and enhancements in the future. Moreover, we have experimented with training the model on a synthetic dataset before proceeding with the Whole Exome Sequencing (WES) dataset, which showed some potential in improving the model's accuracy. Our research and model development were aimed at building a foundation for future improvements, where we successfully established a pipeline for data processing, model training, and evaluation.

However, our current model is not without limitations. We recognized several challenges, both internal and external, such as:

1. Inadequate utilization of features for model training. While the NeuSomatic research paper incorporates nearly 100 different features (as shown by the number of channels in the NeuSomatic input matrix), we currently employ just three main features including reference channel, tumor base frequency, and base quality. Despite the mention of these features in the related research papers, a detailed explanation and implementation would be time-consuming.
2. There is no filter for candidate selection. When searching for variant candidates in the SAM file, the number of non-variant candidates is overwhelmingly large. Additionally, their feature channels are quite similar to actual variant candidates, making it difficult for the model to differentiate variant types. Therefore, the dataset used for the project or related research must have a specific tumor purity ratio.

Therefore, the future holds immense potential for improvements and adjustments. With a larger and more diverse data pool, we expect to improve the model's robustness. Furthermore, integrating a filtering mechanism for candidate selection

can help handle the challenge of data imbalance. Additional features should also be explored and incorporated to enhance the model's distinguishing capabilities between different types of variants.

CHAPTER 5. SUMMARY

In this study, we focused on developing a Deep Neural Network for the detection of genetic variations in cancer patients, based on the NeuSomatic model proposed in the scientific article, "Deep Convolutional Neural Networks for Accurate Somatic Mutation Detection". Notably, the current research landscape lacks openness in code sharing or uses non-standard libraries and languages that are not extensively developed in the machine learning field, making the alteration and improvement of existing studies challenging.

Our model, however, leverages an accessible alignment approach, implemented in Python - a potent programming language in machine learning. This allows for more straightforward adjustments and improvements in the future. We conducted preliminary training of the model on synthetic data before progressing to Whole Exome Sequencing (WES) datasets, which showed promising potential in enhancing model accuracy.

We also recognized several limitations in our research and model implementation. These included the underutilization of available features for model training and a limited and non-diverse dataset for training and testing. Furthermore, we currently lack a filter for candidate selection, which impacts the model's ability to distinguish between actual variant candidates and non-variants.

As we move forward, we anticipate that addressing these issues will significantly enhance the model's performance. This includes introducing a filtering process for non-variant candidates to manage data imbalance, and expanding the number of feature channels for the model's input matrix by incorporating additional features. These improvements will provide the model with a broader and more nuanced understanding of genetic variations, thereby enhancing its predictive accuracy.