Systems and Soft Computing 7 (2025) 200195

Contents lists available at ScienceDirect

Systems and Soft Computing

journal homepage: www.elsevier.com/locate/soft-computing-letters

Neural networks optimization via Gauss–Newton based QR factorization on SARS-CoV-2 variant classification

Mohammad Jamhuri [a,b], Mohammad Isa Irawan [a,*], Imam Mukhlash [a], Mohammad Iqbal [a], Ni Nyoman Tri Puspaningsih [c]

[a] *Department of Mathematics, Faculty of Science and Data Analytics, Institut Teknologi Sepuluh Nopember, Sukolilo, Surabaya, 60111, Jawa Timur, Indonesia*
[b] *Department of Mathematics, Faculty of Science and Technology, Universitas Islam Negeri Maulana Malik Ibrahim, Jl. Gajayana No. 50, Malang, 65144, Jawa Timur, Indonesia*
[c] *Department of Chemistry, Faculty of Science and Technology, Universitas Airlangga, Jl. Airlangga No. 4, Surabaya, 60115, Jawa Timur, Indonesia*

ARTICLE INFO

Keywords:
Second-order optimization
Neural networks
Genomic classification
SARS-CoV-2 variants

ABSTRACT

Studies on the COVID-19 pandemic continue due to the potential mutation creating new variants. One response to be aware of the situation is by classifying SARS-CoV-2 variants. Neural networks (NNs)-based classifiers showed good accuracies but are known very costly in the learning process. Second-order optimization approaches are alternatives for NNs to work faster instead of the first-order ones. Still, it needs a huge memory usage. Therefore, we propose a new second-order optimization method for NNs, called *QR-GN*, to efficiently classify SARS-CoV-2 variants. The proposed method is derived from NNs and Gauss–Newton with QR factorization. The goal of this study is to classify SARS-CoV-2 variants given their spike protein sequences efficiently with high accuracy. In this study, the proposed method was demonstrated on a public dataset for the protein SARS-CoV-2. In the demonstrations, the proposed method outperformed other optimization methods in terms of memory usage and run time. Moreover, the proposed method can significantly elevate the accuracy classification for various NNs, such as: single layer perceptron, multilayer perceptron, and convolutional neural networks.

1. Introduction

COVID-19, an acute respiratory disease caused by the SARS-CoV-2 virus, became a global pandemic threatening human populations worldwide [1]. One critical feature to diagnose the SARS-CoV-2 types is by observing its protein sequences, namely spike protein as a first part of SARS-CoV-2 attacking the antibody cells of the host. The attacking process is between the virus's surface and binds to the ACE2 receptors in human cells, allowing the virus to infect the host cell [2]. The spike protein is also the target of neutralizing antibodies the immune system produces in response to infection or vaccination. Generally, the virus can mutate and produce new variants that may evade the immunity of the generated vaccine. Some mutations change the spike protein, which makes the virus resistant to the vaccine [3].

To notify the evolution of SARS-CoV-2, we can monitor its new mutation by classifying its variants. We classify the virus's protein sequence and whether it belongs to the existing variants. The notification becomes essential as new variants may show diverse characteristics we should know as soon as possible, such as transmission rate, virulence, and response to treatments and vaccines. Further, classifying SARS-CoV-2 variants may help track the virus's emergence and prevalence. Up to mid-2022, the World Health Organization (WHO) has detected five variants of concern (VOC), i.e., Alpha, Beta, Delta, Gamma, and Omicron [4]. The accurate classification of these variants plays a critical role in monitoring the evolution of the virus and informing public health responses [5].

Machine learning methods, particularly neural networks, have been applied to genomic classification due to their ability to model complex patterns in biological data [6]. Models such as single layer perceptron (SLP), multilayer perceptron (MLP), convolutional neural networks (CNN), and long short term memory (LSTM) have shown promise [7–11]. However, training the neural networks involves complex architectures and significant computational resources and is prone to overfitting [12].

First-order optimization methods like SGD, AdaGrad, SGDM, RMSProp, and Adam are widely used to train neural networks [13,14].

* Corresponding author.
E-mail address: mii@its.ac.id (M.I. Irawan).

https://doi.org/10.1016/j.sasc.2025.200195
Received 21 December 2023; Received in revised form 18 November 2024; Accepted 6 February 2025
Available online 14 February 2025

These methods are known for their simplicity and efficiency in handling large datasets [15]. However, neural network training often requires deeper networks for high-dimensional, non-convex optimization problems to achieve satisfactory performance [16].

Second-order optimization methods, such as Quasi-Newton or Gauss–Newton algorithm, can provide a better solution to these challenges [17,18]. They consider the curvature of the loss function, which theoretically allows for more informed updates during training and could lead to better convergence properties [19]. However, second-order methods often consume big memory capacities due to the necessity of calculating and inverting Hessian matrices [20].

This study aims to classify SARS-CoV-2 variants with high accuracy, given their spike protein sequences efficiently in terms of memory usage and runtime. To do so, this study proposes an efficient second-order optimization, which helps to learn the spike protein better. It is proved by the superiority of the proposed method over the existing optimization techniques in various neural networks. We believe the proposed method offering a significant advancement in the artificial intelligence community.

The rest of this paper is structured as follows: Section 2 presents the novelty of this work. Section 3 covers relevant literature on SARS-CoV-2 variant classification and neural network optimization. Section 4 explains the proposed method in detail, including dataset information, neural network architectures, and the proposed optimization algorithms. The experimental results are discussed in Section 5, highlighting the effectiveness of the proposed method. Finally, Section 6 summarizes the outcomes of this study and discusses potential future directions for efficient neural network optimization methods with high predictive accuracy.

2. Novelty of the research

The key differences between this work and existing related studies are mainly listed below:

1. `Development of the SARS-CoV-2 classifier based on QR-GN optimization:` This study introduces a novel second-order optimization technique, QR-GN, which integrates the Gauss–Newton method with QR factorization.
2. `First application of QR-GN to SARS-CoV-2 variant classification:` The proposed QR-GN method is applied for the first time to classify SARS-CoV-2 variants using spike protein sequences, including newer variants.
3. `Significant reduction in memory usage and computational overhead:` By utilizing QR factorization within the Gauss–Newton framework, the proposed method significantly reduces memory requirements and computational overhead, enabling efficient handling of large datasets and training of more complex neural network models.
4. `Enhanced training efficiency and practicality for real-world applications:` The QR-GN method accelerates training times through vectorized and batch computations, addressing the prolonged durations associated with traditional methods. This improvement facilitates quicker model deployment and enhances the practicality of the approach for real-world applications.

The innovation of this work lies in proposing a novel, efficient second-order optimization method (QR-GN) for neural networks and applying it to the classification of SARS-CoV-2 variants using spike protein sequences. By addressing limitations of existing methods in terms of memory usage and computational efficiency, this study advances the theoretical framework of neural network optimization and provides a significant contribution to genomic data analysis and public health surveillance.

3. Related works

Some related studies on optimization methods and/or SARS-CoV-2 variant classification are discussed in this section. As both discussions are still limited, we review them separately in the following parts:

SARS-CoV-2 variant classification. Ali et al. [7] identified only five SARS-CoV-2 variants: `alpha, beta, gamma, delta, and epsilon` variants–no `omicron` and others using conventional machine learning. Muhammad et al. [21] predicted solely five SARS-CoV-2 variants with `omicron` rather than `epsilon` using machine learning models such as: XGBoost and LGBM. For better classifier using neural networks, Gunasekaran et al. [22] applied CNNs and various RNNs to classify a few variants of pneumonia diseases to extract DNA sequence features. They only focused on general COVID-19 instead of its specific variants, `delta, omicron`, etc. Similarly, Câmara et al. [9] implemented CNNs and the k-mer approach within image representation for distinguishing the sequence whether SARS-CoV-2 or not. Hossain et al. [23] used a pre-trained CNN model to extract features and employed optimization techniques for dimensionality reduction. Then, an ensemble technique was used to classify COVID-19 cases using CT scan images. Yektadoust et al. [24] developed a single-layer 1D-CNN model for classifying six SARS-CoV-2 variants with `eta` type and even detecting its mutations. Awe et al. [25] combined CNNs and LSTM to predict five variants—similar to this study for `omicron, beta, gamma, delta, alpha`. Jamhuri et al. [26] used CNNs for profiling the hydrophobicity of the SARS-CoV-2 spike protein sequence to help fit into the model and further classify its variants. In general, previous studies concentrated on specific SARS-CoV-2 variants without `others` variant as it can be considered as the default classification. Further, they utilized some existing optimization methods such as: stochastic gradient descent (SGD) or Adam. Those optimizer—first order methods may be less effective on the SARS-CoV-2 classification due to the complex nature of its spike protein sequences. Hence, this study wants to work with second-order optimization to achieve better SARS-CoV-2 variant classification with efficient learning steps.

Optimization method on neural networks. Wang et al. [27] developed a practical Newton method for optimizing CNNs on image classification data. In specific, they employed a Hessian-free approach to avoid explicit Hessian matrix computation. However, the method may lead to instability during the estimation process as it still returns negative definite on the Hessian matrix solution. Xu et al. [28] derived efficient Newton-type methods for non-convex optimization with inexact Hessian information. They discussed the approach theoretically without experimental evidence on data. Also, they did not design it for neural networks. Chen et al. [29] presented a fast algorithm to evaluate the Gauss–Newton Hessian matrix only on multilayer perceptron for image classification. Ozyildirim and Kiran [30] explored the Levenberg–Marquardt (LM) algorithm with hinge loss. Even if it is efficient, we argue that LM is slower than Gauss–Newton. Further, they ignored the computational efficiency in matrix inversion or memory usage. Jamhuri et al. [18] used the Gauss–Newton algorithm to optimize logistic regression models, which can improve the accuracy and fast convergence. Despite that, there is no memory usage issue as they worked on small data but it may occurs when large. Also, Hessian matrix computation may be intractable when the model is more complex or huge. In image classification, Mehmood et al. [31] modified the Adam optimizer to improve training and testing accuracy using VGG16, ResNet50, and DenseNet121. It remains expensive for tuning the hyperparameters carefully. Adeoye et al. [32] avoided Hessian matrix computation by applying generalized Gauss–Newton (GCN) method on multilayer perceptron (MLP) for image classification.

As we mentioned before, the above studies only focused either on SARS-CoV-2 classification with existing optimization method that may be costly or second-order optimization for different cases with Hessian matrix computation problem, but not both. Hence, this study derives efficient Gauss–Newton method from QR factorization concept to boost five SARS-CoV-2 variant classification and `others`.

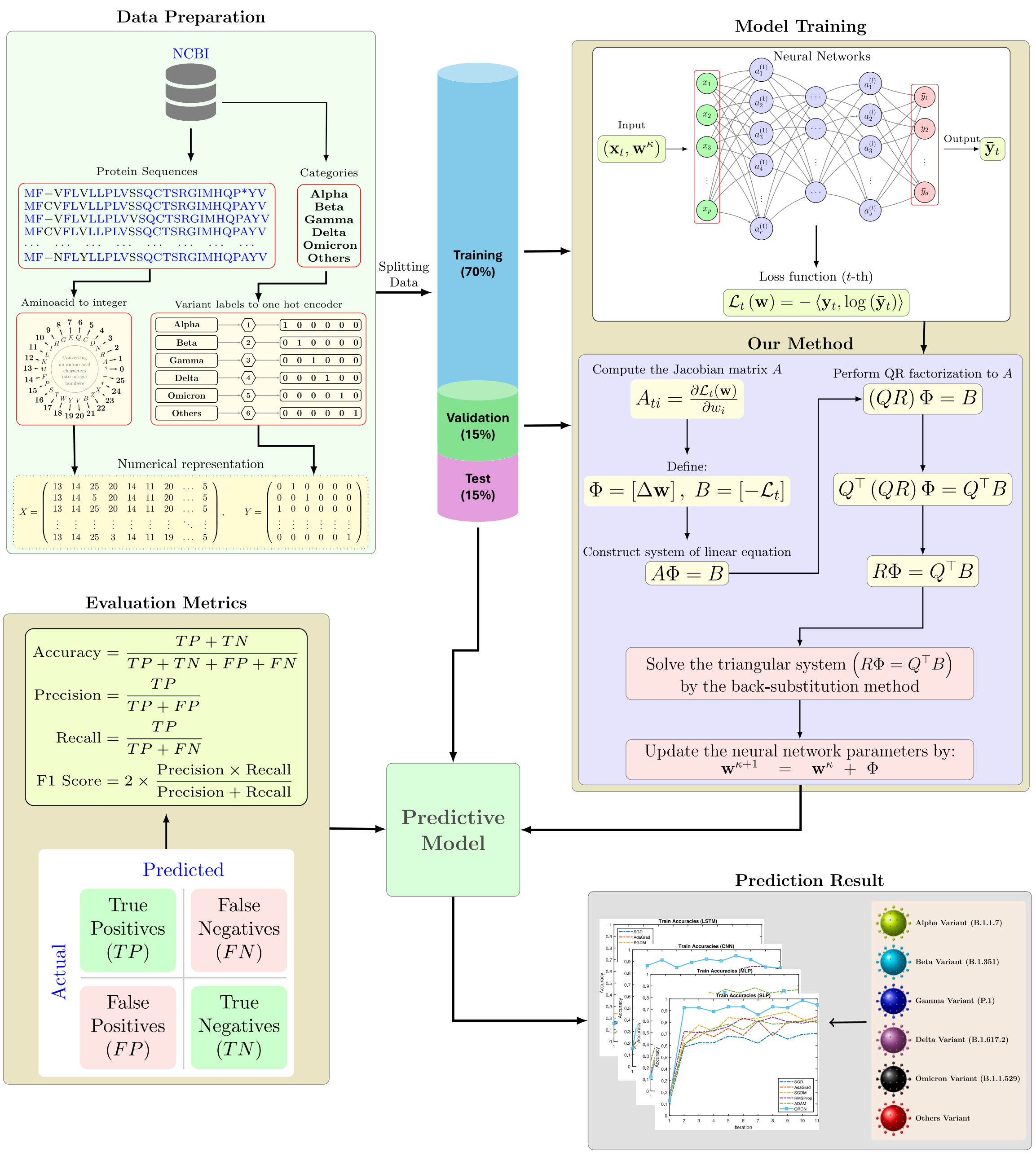


Fig. 1. An overview of the proposed method on SARS-CoV-2 variant classification case.

4. The proposed method

We propose a new second-order optimization method of neural networks (NNs) for classifying SARS-CoV-2 variants. The proposed method, called QR-GN, combines NNs and Gauss–Newton with QR decomposition as one of the speciality of this study. The proposed method contains three main phases, as follows:

- *Data preparation*: the first phase eventually encodes the SARS-CoV-2 spike protein sequences data into embedding features. There are three major steps: data cleansing, data transformation and splitting data. It returns two embedding features: train and test embedding features.
- *SARS-CoV-2 classifier construction*: the second phase learns the train embedding features based on neural networks via Gauss–Newton with QR factorization to have the model of SARS-CoV-2 classification.
- *SARS-CoV-2 variant prediction*: Based on the classifier, the final phase is to predict the test embedding feature class based on the classifier from the previous phase.

The overview of the proposed method in this study application is shown in Fig. 1.

Before going into the details of the proposed method, some necessary notations are discussed in the next section.

4.1. Preliminary

Let S be a set of SARS-CoV-2 spike protein sequences, and L be a set of the corresponding variant labels. To work with neural networks, a set S is encoded into an embedding matrix of input $X \in \mathbb{R}^{m\times p}$, where m is the total number of sequences and p is the embedding dimension obtained from amino acid to integer encoding [33]. Similarly, a set L is encoded into an embedding vector of output $Y \in \mathbb{N}^{m\times q}$ with

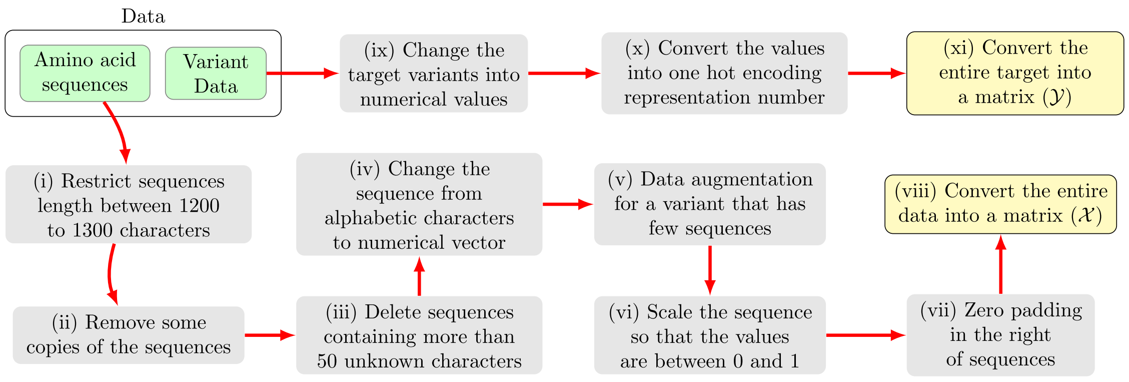

Data
Amino acid sequences
Variant Data
(ix) Change the target variants into numerical values
(x) Convert the values into one hot encoding representation number
(xi) Convert the entire target into a matrix ($\mathcal{Y}$)
(i) Restrict sequences length between 1200 to 1300 characters
(iv) Change the sequence from alphabetic characters to numerical vector
(v) Data augmentation for a variant that has few sequences
(viii) Convert the entire data into a matrix ($\mathcal{X}$)
(ii) Remove some copies of the sequences
(iii) Delete sequences containing more than 50 unknown characters
(vi) Scale the sequence so that the values are between 0 and 1
(vii) Zero padding in the right of sequences

Fig. 2. The details of data preparation for SARS-CoV-2 classification.

Table 1
The amount of SC2seq data from the original to each data preparation step, especially data cleansing and data augmentation.

SARS-CoV-2	#-sequence					
Variant	NCBI	LSR step	DSS step	UFR step	AD-over step	AD-under step
Alpha	191,070	190,926	34,789	20,341	20,341	4,000
Beta	4,097	4,083	1,753	584	*4,672	4,000
Gamma	17,102	17,037	4,117	2,363	*4,726	4,000
Delta	37,172	36,921	15,017	6,859	6,859	4000
Omicron	1,928	1,928	1,263	1,084	*4,336	4,000
Others	21,321	8,669	6,109	454	*7,264	4,000

q is the total number of existing variants, using a one-hot encoding scheme [34]. Following the general purpose of this study, we want to define a function (or classifier) f that maps each SARS-CoV-2 spike protein sequence x to a specific variant label y, i.e., $f : X \rightarrow Y$. From the main objective of this study, a function f is related to an optimization problem for minimizing the cross-entropy loss. Moreover, we solve the optimization problem using QR-GN to efficiently and effectively classify the SARS-CoV-2 spike protein sequences or SC2seq data. In order to learn the sequence data, we must prepare it to fit in neural networks.

4.2. Data preparation

This phase encodes the SC2seq data S into embedding matrices X along with a few common data preprocessing. It has three steps: data cleansing to remove some redundant sequences and missing characters, data transformation to encode the dataset, and splitting data for model construction and prediction purposes, which is shown in Fig. 2. The SC2seq data[1] in this study was fetched from the National Center for Biotechnology Information (NCBI) [35]. It has 272,690 sequences data and more than one thousand variants that were extracted from December, 2019 to July, 2022. In this study, we only observed six variants following five principal variants of concerns (VOCs): `Alpha`, `Beta`, `Gamma`, `Delta`, and `Omicron` [36], and one for other variants—which the class name is `Others`.

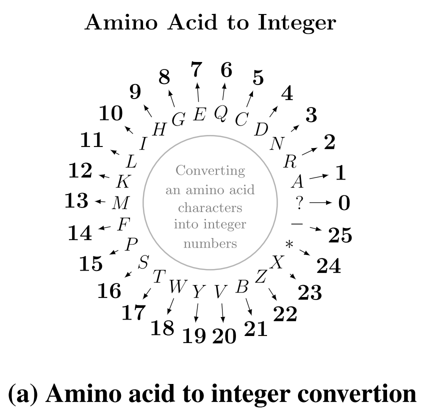
Data cleansing. As the data still has redundant and missing characters, we compiled three sequential cleansing approaches: (i) *long sequence removal* (LSR) is by selecting the sequence that has around 1,200–1,300 amino acid characters at maximum, (ii) *distinct sequence selection* (DSS) is by deleting redundant or similar sequences, and (iii) *uninformative feature reduction* (UFR) is by keeping sequences that only hold *unknown* characters with the length does not exceed 50. At the end of this phase, the data only contains 31,685 sequences. All detail information about the amount of the data after passing each data cleansing can be seen in Table 1.

Data transformation. The SC2seq data is on categorical values, thus it needs to be encoded to numerical ones to fit on neural networks. This step applies six transformation techniques: (i) *sequence-to-integer* (seq2int) is to encode all characters of amino acid in the spike protein sequences into distinct integer values adopting the common seq2int rule in MATLAB,[2] which is depicted on Fig. 3(a), (ii) *data augmentation* (AD) is to make the dataset into balance class as the number of each variant is imbalance, which will be discussed in the next paragraph, (iii) *data normalization* is to normalize the numeric values fall within range $[0, 1]$, (iv) *padding data* is to add some zero values on each sequence with the length is less than the maximum length p (i.e., 1,300 in this study), such that we have uniform length for all sequences or an embedding matrix $X \in \mathbb{R}^{m\times p}$, (v) *label-to-integer* (L2int) is to change categorical SARS-CoV-2 variants into certain integer values, and (vi) *one-hot encoding* is to encode the integer variants into unique bit array resulting in a matrix $Y \in \mathbb{N}^{m\times q}$. Finally, the output of data transformation step is the embedding matrix of SC2seq (X) and the encoded matrix of variant labels (Y). The illustration for fifth and sixth steps are shown in Fig. 3(b). Also, the details of the data transformation can be seen in Fig. 2.

In the data augmentation, over-sampling and under-sampling approaches are utilized. The over-sampling here employs shifted data augmentation techniques (SA) that commonly used for DNA sequences data as similar to [37,38]. Generally, the SA technique adds the number of the data by excluding the unrelated new data. The SA will generate an augmented sequence from shifting the existing sequence based on a specified base pair number as like sliding windows. In that case, the augmented sequence is almost similar to the origin one with a slightly difference depending on how large the base pair number is given. It makes the dataset richer enhancing the learning process to have good classifier f. There is about 48,198 sequences. Yet, it still imbalance issue on the variant numbers. To cope with the issue, under sampling is used by cutting randomly until 4,000 sequences for each variant. Overall, the dataset becomes 24,000 sequences with their corresponding variant labels which will be evaluated using the proposed method.

[1] The SC2seq can be downloaded on SARS-CoV-2SpikeData.

[2] The sequence-to-integer conversion is followed from the package 'aa2int' in MATLAB.


Amino Acid to Integer
Converting an amino acid characters into integer numbers
(a) Amino acid to integer convertion

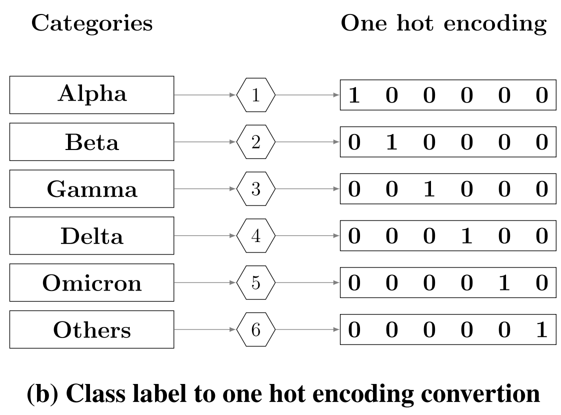


Fig. 3. Data transformation of SC2seq data from (a) seq2int conversion and (b) one-hot encoding.

Splitting data. Last step on data preparation is split the dataset into train, validation, and test sets. The ratio of each set in this study is 70% for train set, 15% for validation set, and 15% for test set. The experiments of this study is compiled from a standalone computer with an AMD RYZEN 7 processor, 8 CPU cores, an RTX 3060 with 6 GB VRAM GPU, and 32 GB of RAM. Moreover, this study used Matlab 2023b and the Deep Learning Toolbox for the software environment. In the next section, the SARS-CoV-2 classifier construction will be explained.

4.3. SARS-CoV-2 classifier construction

This phase attempts to construct the SARS-CoV-2 classifier. To have efficient and effective SARS-CoV-2 variant classification, this study introduces a new optimization algorithm for neural networks. Principally, the proposed algorithm is based on second order optimization. In this part, we will discuss three points for explaining the proposed algorithm, as follows:

- *The derivation of second order optimization for neural networks*: We first derive the second order optimization for neural networks through Gauss–Newton method. Basically, we optimize the loss function of neural networks using Gauss–Newton.
- *The proposed algorithm*: We employ QR factorization on the Gauss–Newton such that it can deliver an efficient optimization algorithm for neural networks. Not only that, we can then benefit the effectiveness of the proposed method on classifying the SARS-CoV-2 variants.
- *Neural networks integration*: In this part, we integrate a few neural networks on the proposed algorithm to see its efficacy.

Second order optimization for neural networks derivation. Suppose C be the cost function on neural networks, given by:

$$C(\mathbf{w}) = \sum_{t=1}^{m} \mathcal{L}_t(\mathbf{w}); \tag{1}$$

with m is the number of sample data, $\mathbf{w}$ represent a vector of learnable parameters, and $\mathcal{L}_t$ is the cross-entropy loss of the neural networks for tth sample which defined in Eq. (2) below:

$$\mathcal{L}_t = -\left\langle \mathbf{y}_t, \log\left(\hat{\mathbf{y}}_t\right)\right\rangle; \tag{2}$$

where $\mathbf{x}_t \in \mathbb{R}^p$ is the input vector, $\mathbf{y}_t \in \mathbb{N}^q$ is the encoded vector for the true label, and $\hat{\mathbf{y}}_t = f\left(\mathbf{x}_t; \mathbf{w}\right)$ is the predicted probability vector.

Generally, the optimization problem in neural networks involves minimizing the cost function defined in Eq. (1). This study approximates the problem in second order modes by using the Gauss–Newton method. To find the optimal solution, we need to consider the loss function at the tth instance, which is defined as a function of the parameter $\mathbf{w}$ at the $(\kappa+1)$th iteration in Eq. (3).

$$\mathcal{L}_t\left(\mathbf{w}^{\kappa+1}\right) = 0; \tag{3}$$

The goal of the approximation is to search for an update parameter $\Delta\mathbf{w}$ that can reduce the loss value of $\mathcal{L}_t$. Further, the approximation of $\mathcal{L}_t(\mathbf{w}^{\kappa+1})$ around the current parameters $\mathbf{w}^{\kappa}$ is based on the Taylor series expansion, where its truncation after the first derivative can be written, as follows:

$$\mathcal{L}_t\left(\mathbf{w}^{\kappa+1}\right) \approx \mathcal{L}_t\left(\mathbf{w}^{\kappa}\right) + \left\langle \nabla\mathcal{L}_t\left(\mathbf{w}^{\kappa}\right), \Delta\mathbf{w}\right\rangle; \tag{4}$$

with the inner product on the second term of Eq. (4) is defined as:

$$\left\langle \nabla\mathcal{L}_t\left(\mathbf{w}^{\kappa}\right), \Delta\mathbf{w}\right\rangle = \sum_{i=1}^{n} \frac{\partial\mathcal{L}_t}{\partial w_i}\Delta w_i, \qquad \Delta w_i = w_i^{\kappa+1} - w_i^{\kappa};$$

with n is the number of learnable parameters, $\mathbf{w}^{\kappa}$ and $\mathbf{w}^{\kappa+1}$ are the parameters at the κth and $(\kappa+1)$th iterations. $\Delta\mathbf{w}$ is the difference of the current and next parameters $\left(w_i^{\kappa+1} - w_i^{\kappa}\right)$. $\nabla\mathcal{L}_t\left(\mathbf{w}^{\kappa}\right)$ is the gradient of the loss function *w.r.t* the parameters, such that its element is coming from the partial derivative of $\partial\mathcal{L}_t/\partial w_i$. Subsequently, the dimensions of $\Delta\mathbf{w}$ and $\nabla\mathcal{L}_t\left(\mathbf{w}^{\kappa}\right)$ are n. Further, we shall substitute Eq. (4) into Eq. (3) to obtain the following equation:

$$\mathcal{L}_t\left(\mathbf{w}^{\kappa}\right) + \left\langle \nabla\mathcal{L}_t\left(\mathbf{w}^{\kappa}\right), \Delta\mathbf{w}\right\rangle = 0. \tag{5}$$

In the learning process, assume that there is m number or batch of data. Eq. (5) will forms into a system of linear equations, as follows:

$$\begin{aligned}
\mathcal{L}_1\left(\mathbf{w}^{\kappa}\right) + \left\langle \nabla\mathcal{L}_1\left(\mathbf{w}^{\kappa}\right), \Delta\mathbf{w}\right\rangle &= 0\\
\mathcal{L}_2\left(\mathbf{w}^{\kappa}\right) + \left\langle \nabla\mathcal{L}_2\left(\mathbf{w}^{\kappa}\right), \Delta\mathbf{w}\right\rangle &= 0\\
&\vdots\\
\mathcal{L}_m\left(\mathbf{w}^{\kappa}\right) + \left\langle \nabla\mathcal{L}_m\left(\mathbf{w}^{\kappa}\right), \Delta\mathbf{w}\right\rangle &= 0
\end{aligned}$$

The above system of linear equations can be rewritten to see their relations below.

$$\begin{aligned}
\left\langle \nabla\mathcal{L}_1\left(\mathbf{w}^{\kappa}\right), \Delta\mathbf{w}\right\rangle &= -\mathcal{L}_1\left(\mathbf{w}^{\kappa}\right)\\
\left\langle \nabla\mathcal{L}_2\left(\mathbf{w}^{\kappa}\right), \Delta\mathbf{w}\right\rangle &= -\mathcal{L}_2\left(\mathbf{w}^{\kappa}\right)\\
&\vdots\\
\left\langle \nabla\mathcal{L}_m\left(\mathbf{w}^{\kappa}\right), \Delta\mathbf{w}\right\rangle &= -\mathcal{L}_m\left(\mathbf{w}^{\kappa}\right)
\end{aligned}$$

For simplicity, the above equations are expressed into a matrix, as follows:

$$A\,\Phi = B; \tag{6}$$

with

$$A = \begin{bmatrix} \nabla\mathcal{L}_1\left(\mathbf{w}^{\kappa}\right)\\ \nabla\mathcal{L}_2\left(\mathbf{w}^{\kappa}\right)\\ \vdots\\ \nabla\mathcal{L}_m\left(\mathbf{w}^{\kappa}\right)\end{bmatrix}, \quad \Phi = [\Delta\mathbf{w}]^{\top}, \quad \text{and} \quad B = \begin{bmatrix} -\mathcal{L}_1\left(\mathbf{w}^{\kappa}\right)\\ -\mathcal{L}_2\left(\mathbf{w}^{\kappa}\right)\\ \vdots\\ -\mathcal{L}_m\left(\mathbf{w}^{\kappa}\right)\end{bmatrix} \tag{7}$$

In that case, the dimensions of A, Φ, and B in Eq. (7) are $m \times n$, $n \times 1$ and $m \times 1$. On the other hands, A is the Jacobian matrix, B is the residual vector, and Φ is the solution of Eq. (7) or the update neural networks parameter. As A is a non-square matrix, it is difficult to get the solution of Eq. (6) directly. An alternative way is based on the least-square technique (LS). Eq. (6) can be solved through LS into the following normal equation below.

$$\left[A^{\top} A\right] \Phi = \left[A^{\top} B\right]; \tag{8}$$

That is what this study wants to solve.

The proposed algorithm. To obtain the solution of Eq. (8), this study proposes an algorithm namely QR-GN_Opt based on QR factorization. The proposed algorithm includes two stages: QR factorization, and update neural network weights, which will be discussed on the following paragraphs.

QR_fact. The QR factorization procedures are depicted on Algorithm 1. Note that direct methods can solve Eq. (8) only on small datasets or light systems. Otherwise, they become intractable, especially when more than thousands of parameters and the size of $\left[A^{\top} A\right]$ turns out huge, and the variance may be very high. To avoid high biases and heavy computations, this study introduces the QR factorization on Eq. (8) as it is known as a robust approach for solving linear equations system [39]. In Algorithm 1, the matrix A is factorized into the product of an orthogonal matrix Q and an upper triangular matrix R, i.e., $A = QR$, to facilitate the identification of the solutions of Eq. (8) from the Back substitution technique. Specifically, the factorization is obtained by an iterative algorithm under the Gram–Schmidt process and Householder reflections, as in [40]. Based on the QR factorization, we can update the weights on the neural networks until the stopping conditions are met.

Algorithm 1: QR_fact

```
Input: A matrix $A \in \mathbb{R}^{m \times n}$ with linearly independent column
Output: Orthogonal matrix $Q$, and upper triangular $R$
for $j = 1$ to $n$ do
    $y \leftarrow A_j$
    $R_{j,j} \leftarrow \|y\|$
    $Q_j \leftarrow y / R_{j,j}$
    for $i = 1$ to $j - 1$ do
        $R_{i,j} \leftarrow Q_i^{\top} A_j$
        $y \leftarrow y - R_{i,j} Q_i$
    end
end
return $Q$, $R$
```

QR-GN_Opt. The procedures of update neural network weights are described on Algorithm 2. The idea of Algorithm 2 is to cope with the inconsistency issue on Eq. (6) (which may occurs) as follows:

1. Recall Algorithm 1 to obtain the QR factorization of A or $A = Q\ R$.
2. Transpose the orthogonal matrix Q, i.e., $Q^{\top}$ and multiply it to both sides of Eq. (6) such that $Q^{\top}\ A\ \Phi = Q^{\top}\ B$ or $R\ \Phi = Q^{\top}\ B$.
3. Based on the back substitution method, solve the upper triangular system $R\ \Phi\ =\ Q^{\top} B$ from the previous step to obtain Φ.

In the first step, we randomly initialize the weight parameter $\mathbf{w}^0$. Then, we set the iteration κ. During the learning process, there are two major stages: classifier construction and updating the weight parameters. The classifier construction is based on neural networks, which will be discussed in the next section. Yet, it returns the output predictions $\hat{\mathbf{y}}$ and the loss functions $\mathcal{L}$. The updated weight parameters are basically

Algorithm 2: QR-GN_Opt

```
Input: Training set $\{(\mathbf{x}_t, \mathbf{y}_t)\}_{t=1}^{m}$, initial parameters $\mathbf{w}^0$,
       convergence threshold $\epsilon$, and maximum iteration $\mathcal{N}$
Output: Optimized parameters $\mathbf{w}^*$
$\mathbf{w}^{\kappa} \leftarrow \mathbf{w}^{\mathbf{0}}$; $\kappa \leftarrow 0$;
while $\kappa < \mathcal{N}$ do
    for $t \leftarrow 1$ to $m$ do
        $\hat{\mathbf{y}}_t = f(\mathbf{x}_t; \mathbf{w}^{\kappa}) \leftarrow$ neural network output with input $\mathbf{x}_t$
        and parameters $\mathbf{w}^{\kappa}$; $\mathcal{L}_t \leftarrow \left\langle \mathbf{y}_t, \log\left(\hat{\mathbf{y}}_t\right)\right\rangle$
    end
    $A \leftarrow$ Compute the Jacobian matrix of $\mathcal{L}_t$ w.r.t. parameters
     $\mathbf{w}$;
    $B \leftarrow \left(-\mathcal{L}_1, -\mathcal{L}_2, \ldots, -\mathcal{L}_m\right)^{\top}$;
    $Q, R \leftarrow QR$_fact$(A)$;
    $\Phi \leftarrow$ Solve $R\Phi = Q^T B$ using the Back substitution;
    if $\|\Phi\| < \epsilon$ then
        break;
    end
    $\mathbf{w}^{\kappa+1} \leftarrow \mathbf{w}^{\kappa} + \Phi$;
    $\kappa \leftarrow \kappa + 1$;
end
return $\mathbf{w}^* \leftarrow \mathbf{w}^{\kappa}$
```

described as follows:

$$\mathbf{w}^{\kappa+1} = \mathbf{w}^{\kappa} + \Delta\mathbf{w}; \tag{9}$$

Based on the Gauss–Newton with QR factorization, $\Delta\mathbf{w}$ is estimated from Φ. To obtain Φ, we take the Jacobian matrix A of the loss functions *w.r.t.* $\mathbf{w}$. Following the above idea, the next step is to recall QR_fact(A) to get the orthogonal matrix Q and the upper triangular matrix R. From both matrices, the Φ can be obtained from $R\Phi = Q^T B$ using the Back substitution. Finally, we update the weight parameters for the next iteration, as follows:

$$\mathbf{w}^{\kappa+1} = \mathbf{w}^{\kappa} + \Phi; \tag{10}$$

This learning process will be terminated when the iteration κ exceeds $\mathcal{N}$ or the norms of Φ are less than the convergence threshold ϵ. For the second terminal condition, the assumption is that only a very small change happens if the norm of the local optima solution Φ is below ϵ. In other words, it is close enough to the optimal set of parameters $\mathbf{w}^*$.

In the next section, the neural networks that were implemented in this study will be explained.

Neural networks integration. From Algorithm 2, a few steps are basically the main course for constructing the classifier for SCseq data using neural networks f. In this study, a few neural networks are observed to see the robustness of the proposed algorithm. Four well-known and basic neural networks will be compiled with different domain natures: single-layer perceptron (SLP), multilayer perceptron (MLP), convolutional neural networks (CNNs), and long short terms memory (LSTM). In general, the architecture of each neural network can be seen in Fig. 4.

The details of neural network architectures, such as input and output sizes, number of layers, and the total parameters, are listed on Table 2. Indeed, each model has the same input and output sizes: 1,280 sequences and 6 SARS-CoV-2 variants for this case. Moreover, the architecture of each neural network follows `Deep Learning Toolbox` in MATLAB. Those default settings achieved good performance in this study's applications. Moreover, we can describe the number of parameters in each layer of neural networks and the total at the end. SLP encompasses fully connected layers with 7,686 learnable parameters, including $6 \times 1,280$ weight and 6×1 bias parameters. MLP composes two fully connected layers with 64,356 learnable parameters, including weight and bias parameters for each layer. CNNs include one-dimensional convolution layers along with a fully connected layer with

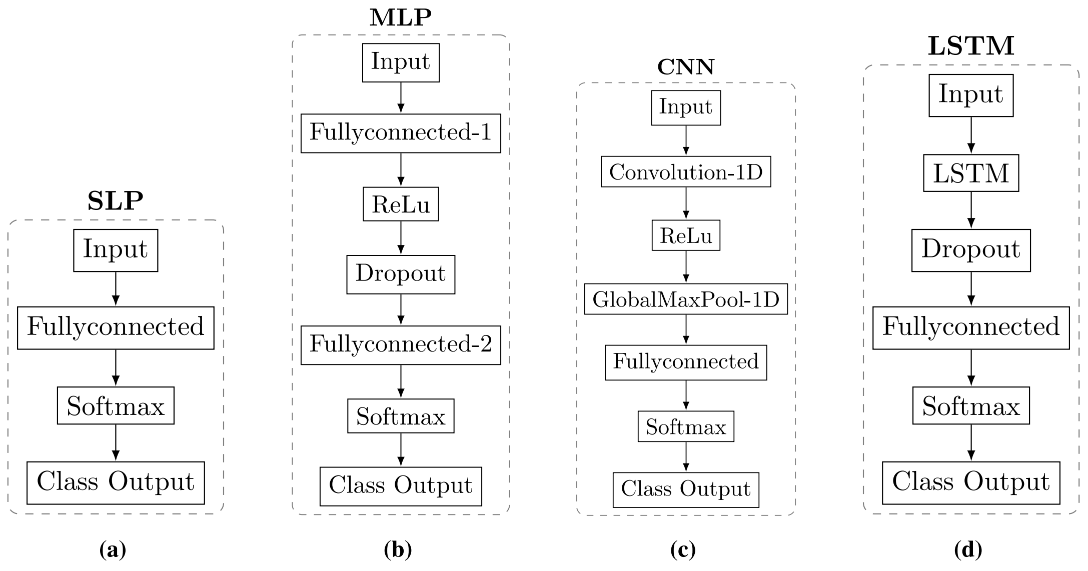


Fig. 4. Neural network architectures that are used: (a) SLP, (b) MLP, (c) CNNs, and (d) LSTM. The arrows in the diagram describe the data flow, starting at the input layer until the output layer.

Table 2
Number of learnable parameters of the models.

Model	Input size	Output size	Layer	Type	Size	Total parameters
SLP	1,280	6	Fc	Weights	$6 \times 1,280$	7,686
				Bias	6×1	
MLP	1,280	6	Fc-1	Weights	$50 \times 1,280$	64,356
				Bias	50×1	
			Fc-2	Weights	6×50	
				Bias	6×1	
CNN	1,280	6	Conv1d	Weights	$11 \times 1 \times 96$	1,734
				Bias	1×96	
			Fc	Weights	6×96	
				Bias	6×1	
LSTM	1,280	6	lstm	Input weights	$50 \times 4 \times 1,280$	266,506
				Recurrent weight	$50 \times 4 \times 50$	
				Bias	50×4	
			Fc	Weights	6×50	
				Bias	6×1	

1,734 learnable parameters in total. LSTM consists of fully connected layers and four gate mechanisms with a total learnable parameters of 266,506. The early assumption is that LSTM will have the heaviest computation among other neural networks in this study as it has the largest number of parameters. During the training process, we set the batch size equal to 250 and activated early stopping to prevent overfitting issues.

4.4. SARS-CoV-2 variant prediction

The previous phase constructs the SARS-CoV-2 classifier or model. Then, the test data, the spike protein sequences, is fed to the classifier to predict their variants. In this study, the test data contains 3,600 SARS-CoV-2 spike protein sequences to be predicted. The prediction results were evaluated based on four metrics: accuracy, precision, recall, and F1-score, as follows:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

This study examined not only the above metrics but also the proposed method based on computational time and memory usage. The computational time analysis is based on run-time comparisons, #-iterations, and #-epochs. The memory usage analysis is based on the total memory allocations in the systems and matrix computations. Further details for the evaluations are discussed in the next section.

5. Result and discussion

This section will discuss the results of the experiment and their analysis. Generally, the discussions are categorized into four parts: (i) baseline optimization method comparison, (ii) convergence tests, (iii) memory requirement analysis, (iv) real case prediction, and (v) further discussion to provide clear explanations of why the proposed method is important.

5.1. Baseline optimization method comparisons

As this study focuses on optimizing the neural networks, the first discussion is to compare the proposed method with the baseline ones. The baseline optimization methods are stochastic gradient descent

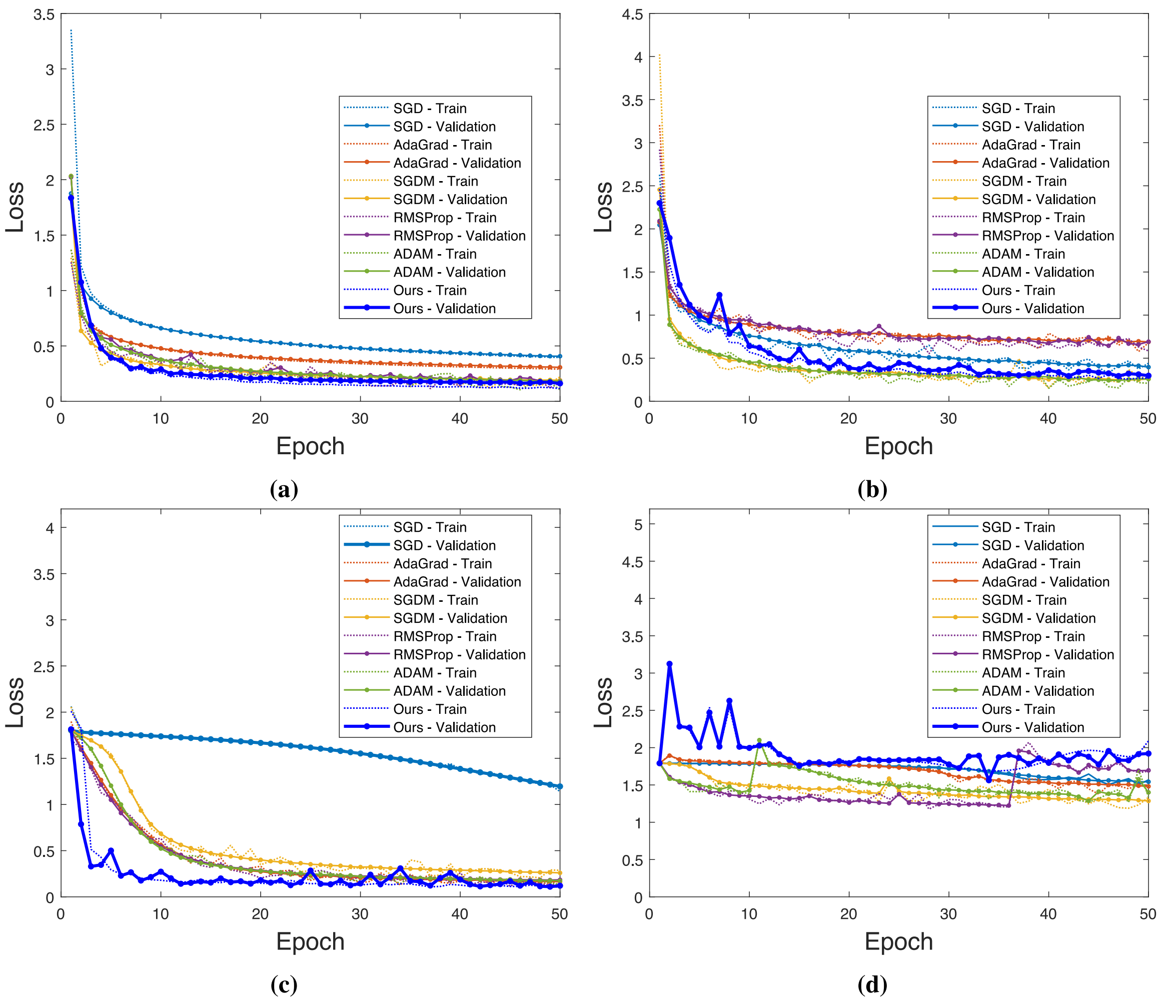


Fig. 5. Learning curve comparisons on SARS-CoV-2 Classification using (a) SLP, (b) MLP, (c) CNN, and (d) LSTM from various optimization methods.

(SGD) [41], AdaGrad [42], stochastic gradient descent with momentum (SGDM) [43], root mean square propagation (RMSProp) [44], and Adam [45] to be compared. To be specific, the comparisons concentrate on learning curves to see the effectiveness of the proposed method and computational time to know its efficiency. In this study, we observed the learning processes until 50 epochs. The learning processes of all NNs here with different optimization methods can be seen in Fig. 5. Moreover, the computational times for all NNs with different optimization methods are listed on Table 3. The discussions are delivered for each NN in the following paragraphs.

Single layer perceptron. As the simplest architecture, SLP turns out to demonstrate the most stable learner over different optimization methods with the minimum error among other NNs. Especially, the proposed method can alleviate SLP to achieve such exceptional performance on SARS-CoV-2 classification. On the other hand, it is clear that the proposed method showed the longest execution time compared to others since it is the one and only second-order optimization method type in this study. Yet, the proposed method provided the most accurate as expected.

Multilayer perceptron. The proposed method indicated competitive performances in MLP by taking third place after Adam and SGDM, consecutively with only a small gap of around 0.05. The main reason is that MLP has larger parameters than SLP, which makes the Jacobian matrix calculation of the proposed method more complex. Accordingly, the computation time of the proposed method here is twice slower than in SLP, while other methods are only slightly longer. However, we seek the simplest model to classify the SARS-CoV-2 variants accurately, which still comes from the proposed method in SLP architecture with the smallest loss value.

Convolutional neural networks. The proposed method can harmonize on CNNs very well by exhibiting the most excellent learning process even at the earliest stage than others. The proposed method provides better kernel matrix on the feature filtering (or selection) of CNNs. Also, it requires more expensive computation than others. Despite that, the proposed method in the CNNs has the most accurate classification compared to other NNs and optimization methods.

Long short term memory. Similar to MLP, LSTM has the largest parameters and is known with the four gates mechanism which makes the computation time is the longest ones. Also, the learning process from the proposed method tends to be difficult to perform better than others.

In general, the proposed method can be very effective on simple yet robust models instead of complex ones. Overall, we recommend to apply CNNs with the proposed method for accurate SARS-CoV-2 variant classification. Further, if seeking little bit faster and edible classification, the proposed method with SLP can be the best alternative approach.

5.2. Convergence tests

This section analyzes the convergence behavior of the proposed QR-GN optimization method compared to state-of-the-art ones: SGD, AdaGrad, SGDM, RMSProp, and Adam. The evaluation focuses on the following aspects:

Table 3
The loss value and computational time of each NN with different optimization method at 50 epoch.

NNs	Optimizer	Loss		Total runtime (s)
		Training	Validation	
SLP	SGD	0.3955	0.4071	821.18
	AdaGrad	0.2910	0.3067	929.75
	SGDM	0.1194	0.1896	867.32
	RMSProp	0.1139	0.1771	919.50
	Adam	0.1042	0.1807	919.54
	Ours	**0.1159**	**0.1617**	**6,252.79**
MLP	SGD	0.3262	0.3964	894.71
	AdaGrad	0.5769	0.6885	858.60
	SGDM	0.1547	0.2476	950.32
	RMSProp	0.5517	0.6444	828.93
	Adam	0.1499	0.2416	930.34
	Ours	**0.2587**	**0.2932**	**12,797.45**
CNN	SGD	1.1405	1.1958	6,373.95
	AdaGrad	0.1273	0.1645	6,152.45
	SGDM	0.2055	0.2594	6,948.20
	RMSProp	0.1150	0.1816	6,335.01
	Adam	0.1357	0.1738	6,067.15
	Ours	**0.1073**	**0.1099**	**16,671.61**
LSTM	SGD	1.5779	1.6192	9,394.23
	AdaGrad	1.7430	1.7581	10,891.37
	SGDM	1.2030	1.2919	11,212.57
	RMSProp	1.3110	1.3641	11,899.25
	Adam	1.2608	1.3274	9,629.12
	Ours	**1.6877**	**1.5625**	**48,975.29**

Table 4
Summary of convergence performance for different neural network models and optimizers.

Model type	Optimizer	Accuracy		Epochs	Total runtime (s)
		Training	Validation		
SLP	SGD	0.9800	0.9356	197	3,235.48
	AdaGrad	0.9850	0.9458	307	5,708.66
	SGDM	0.9800	0.9511	48	832.63
	RMSProp	0.9750	0.9561	48	882.72
	Adam	0.9750	0.9539	43	790.81
	Ours	**0.9821**	**0.9708**	**22**	**2,907.85**
MLP	SGD	0.9500	0.9256	57	1,019.98
	AdaGrad	0.9300	0.8869	20	343.44
	SGDM	0.9650	0.9386	21	399.14
	RMSProp	0.9200	0.8719	22	364.73
	Adam	0.9700	0.9481	25	465.17
	Ours	**0.9560**	**0.9456**	**18**	**4,104.66**
CNN	SGD	0.9800	0.9411	829	105,680.14
	AdaGrad	0.9650	0.9303	40	4,921.96
	SGDM	0.9300	0.9142	31	4,307.89
	RMSProp	0.9450	0.9167	28	3,547.61
	Adam	0.9750	0.9372	49	5,945.81
	Ours	**0.9679**	**0.9617**	**12**	**2,059.94**
LSTM	SGD	0.5900	0.5544	51	9,582.12
	AdaGrad	0.6500	0.5517	21	4,533.48
	SGDM	0.6450	0.5639	20	4,485.03
	RMSProp	1.0000	0.5628	9	2,141.87
	Adam	0.6000	0.5497	10	1,925.83
	Ours	**0.3000**	**0.3038**	**6**	**5,877.04**

1. *Training and validation accuracy improvement:* This part observed the improvement of each neural network with different optimization methods during the learning process according to the accuracy metric. It simply shows how effective the proposed method in maximizing the accuracy is.
2. *Number of epochs:* This part investigates how fast the proposed method to converge is. It compared the total number of epochs during the training phase between the proposed method and others.
3. *Computation time:* This part computes the runtime of the proposed method on different NNs to see how fast it is to achieve convergence. It is also compared to the other optimization methods.

Before discussing the above three aspects, the experiments followed early stopping condition as in [46] to avoid the overfitting phenomena. The first aspect is shown in Fig. 6. Meanwhile, the second and third aspects are listed on Table 4. The discussion on those aspects will be delivered on each neural network architecture in the next following paragraphs.

Single layer perceptron. QR-GN achieved a validation accuracy of 0.9708 at epoch 22, which is higher than most other optimizers. However, RMSProp and Adam achieved slightly lower validation accuracies of 0.9561 and 0.9539, respectively, but with fewer epochs. Despite QR-GN's quick convergence, its significantly higher total runtime of 2,907.85 s compared to other optimizers made it less efficient in terms of computational cost.

Multilayer perceptron. QR-GN achieved a validation accuracy of 0.9456 at epoch 18, demonstrating rapid convergence and competitive accuracy. Adam achieved a slightly higher validation accuracy of 0.9481 at epoch 25. QR-GN's quick convergence highlights its strength in achieving high accuracy rapidly, but its high computational cost of 4,104.66 s indicates substantial resource requirements.

Convolutional neural networks. QR-GN achieved a validation accuracy of 0.9617 at epoch 12, showing the fastest convergence among the optimizers and achieving the highest accuracy. Adam and SGD had competitive validation accuracies of 0.9372 and 0.9411, respectively. QR-GN's exceptional convergence speed and high accuracy at epoch 12 are noteworthy.

The QR-GN for the LSTM model shows the lowest validation accuracy at 0.3038, indicating it is less effective at maximizing accuracy compared to other optimizers like RMSProp, which achieves a higher validation accuracy of 0.5628. Although QR-GN converges quickly in just 6 epochs, like Adam, its total runtime is significantly higher at 5,877.04 s due to the computational complexity of the method. This suggests that while QR-GN offers rapid convergence in terms of epochs, its lower accuracy values and extended computational time make it less suitable for LSTM models without further optimization.

Afterall, QR-GN demonstrates strong performance in terms of rapid convergence and high accuracy in CNN models, achieving competitive validation accuracies and offsetting its higher computational cost with its efficiency. In SLP and MLP models, QR-GN achieves high accuracy but is outperformed by Adam and RMSProp in terms of computational cost, which limits its practicality.

5.3. Memory requirement analysis

The memory requirements for an optimization algorithm are a vital consideration, especially for large-scale problems. In this section, we address the memory requirements for updating trainable parameters in neural networks using the Gauss–Newton method. We originally faced the challenge of solving an inconsistent system of linear Eq. (6), which led to the normal Eq. (8). Solving the normal equation requires significant memory to store the Hessian matrix $\left[A^\top A\right]$, which can be large depending on the number of trainable parameters in the neural network's architecture. We will discuss the memory requirements for solving this system using the least squares method, serving as the baseline approach, and compare them with the proposed approach.

Least squares method. Solving the normal Eq. (8) requires memory to store the Jacobian matrix A, the Hessian matrix $\left[A^\top A\right]$, and the gradient vector Φ. Specifically, the Jacobian matrix requires $m \times n \times \Omega$ bytes of memory, where Ω represents the memory size allocated for each data type, typically 8 bytes. In comparison, the Hessian matrix and the gradient vector, respectively, require $n \times n \times \Omega$ and $n \times \Omega$ bytes of

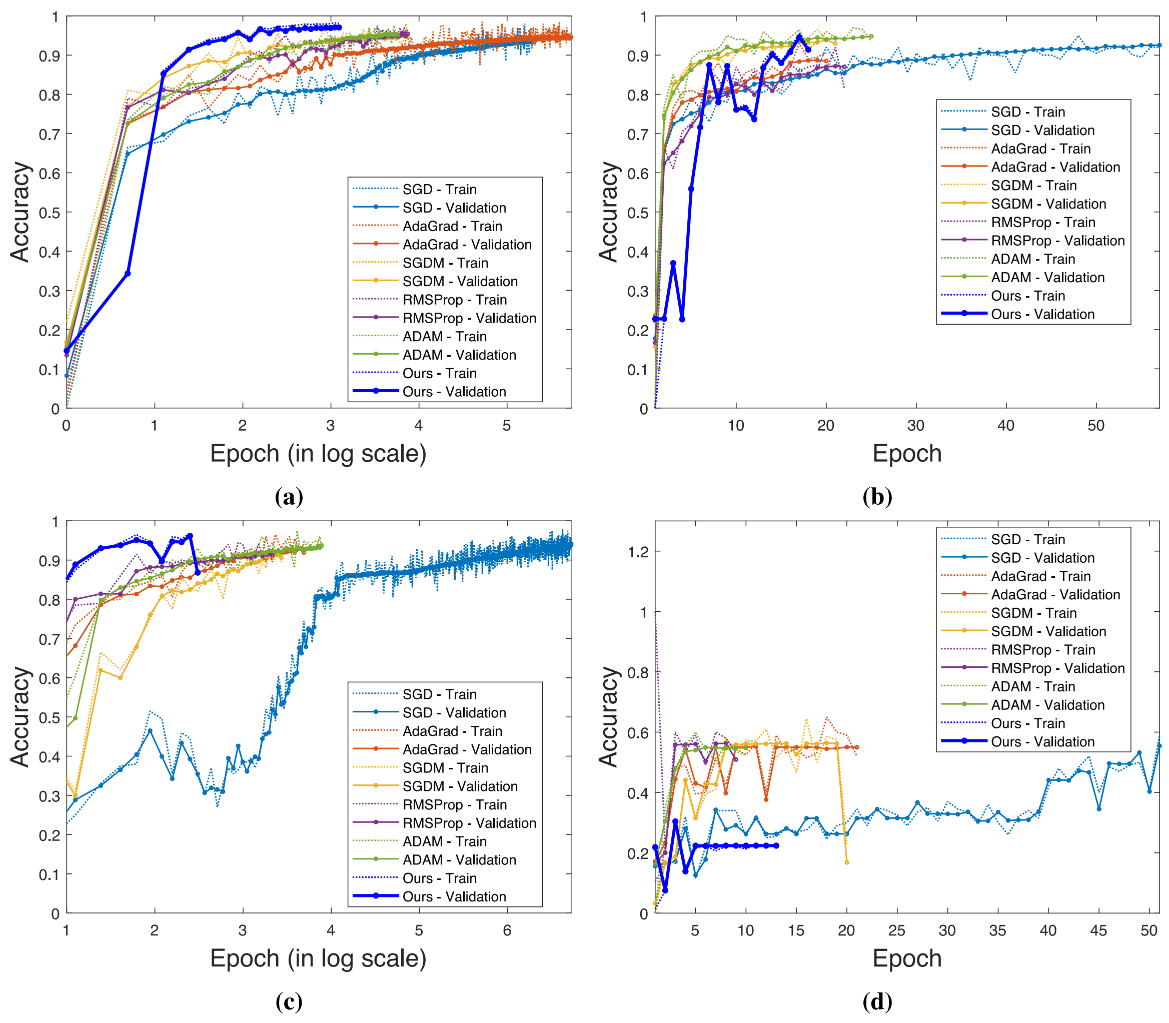


Fig. 6. Convergence curve comparisons on SARS-CoV-2 Classification using (a) SLP, (b) MLP, (c) CNN, and (d) LSTM from various optimization methods.

Table 5
Memory requirement of the Gauss–Newton by least squares.

Type of neural network	Number of data (m)	Number of parameters (n)	Memory for matrices (in bytes) Jacobian (A) ($m \times n \times 8$)	Hessian ($A^\top A$) ($n \times n \times 8$)	Gradient (Φ) ($n \times 1 \times 8$)	Total memory (in Gigabytes)
SLP	250	7,686	15,372,000	472,447,392	61,488	0.46 Gb
MLP	250	64,356	128,712,000	33,173,291,136	514,848	31.81 Gb
CNN	250	1,734	3,468,000	24,032,328	13,872	0.03 Gb
LSTM	250	266,506	532,012,000	567,346,407,616	2,132,048	528.99 Gb

allocated memory. We can summarize the total memory requirement (M_{total}) using the formula:

$$M_{\text{total}} = \left(mn + n^2 + n\right) \times \Omega$$

Further details are provided in Table 5.

QR factorization. Solving Eq. (6) via QR Factorization primarily requires allocated memory for the Jacobian matrix A, the residual vector B, and the QR decomposition of A. This decomposition yields an orthogonal matrix Q, an upper triangular matrix R, a gradient vector Φ, and an updated parameter vector. The allocated memory requirements are as follows: the Jacobian matrix requires $m \times n \times \Omega$, the residual vector needs $m \times \Omega$, and Q and R from the QR decomposition require $m \times m \times \Omega$ and $m \times n \times \Omega$, respectively. The gradient vector Φ and the updated parameter vector collectively require $2 \times n \times \Omega$. Thus, the total pre-allocated memory requirement is:

$$M_{\text{total}} = \left(m^2 + m + 2mn + 2n\right) \times \Omega$$

The memory requirements of the Gauss–Newton method with QR factorization, as applied to the models (SLP, MLP, CNN, and LSTM), are detailed in Table 6.

The Gauss–Newton method effectively addresses the problem by utilizing QR factorization to solve linear Eq. (6), thus circumventing the need to calculate the Hessian matrix.

5.4. Real case results

In this section, we delve into the performance metrics of the proposed method compared to other baseline optimizers such as SGD, AdaGrad, SGDM, RMSProp, and ADAM across various neural network models. The metrics considered for this analysis include accuracy, precision, recall, and F1-score, which comprehensively evaluate the

Table 6
Memory requirement of the Gauss–Newton by QR factorization.

Type of neural Network	Number of data (m)	Number of parameters (n)	Memory for matrices (in bytes) Jacobian (A) ($m \times n \times 8$)	Residual (B) ($m \times 1 \times 8$)	2 × Gradient (Φ) ($2 \times n \times 1 \times 8$)	Orthogonal (Q) ($m \times m \times 8$)	Up-Trian (R) ($m \times n \times 8$)	Total memory (in MB)
SLP	250	7,686	15,372,000	2,000	123,072	500,000	15,372,000	29.38
MLP	250	64,356	128,712,000	2,000	1,029,688	500,000	128,712,000	246.94
CNN	250	1,734	3,468,000	2,000	27,744	500,000	3,468,000	6.63
LSTM	250	266,506	533,012,000	2,000	4,264,096	500,000	533,012,000	1,021.39

Table 7
Comparative performance metrics of neural network models.

Model Type	Optimizer	Accuracy	Precision	Recall	F1-Score
SLP	SGD	0.9547	0.9548	0.9547	0.9547
	AdaGrad	0.9508	0.9509	0.9508	0.9508
	SGDM	0.9669	0.9671	0.9669	0.9670
	RMSProp	0.9458	0.9460	0.9458	0.9459
	ADAM	0.9633	0.9634	0.9633	0.9634
	Ours	**0.9789**	**0.9789**	**0.9789**	**0.9789**
MLP	SGD	0.9186	0.9197	0.9186	0.9189
	AdaGrad	0.8858	0.8893	0.8858	0.8867
	SGDM	0.9575	0.9575	0.9575	0.9574
	RMSProp	0.9278	0.9280	0.9278	0.9278
	ADAM	0.9553	0.9558	0.9553	0.9553
	Ours	**0.9456**	**0.9499**	**0.9456**	**0.9466**
CNN	SGD	0.9372	0.9368	0.9372	0.9369
	AdaGrad	0.9581	0.9579	0.9581	0.9579
	SGDM	0.9208	0.9204	0.9208	0.9204
	RMSProp	0.9556	0.9554	0.9556	0.9554
	ADAM	0.9603	0.9601	0.9603	0.9601
	Ours	**0.9633**	**0.9640**	**0.9633**	**0.9631**
LSTM	SGD	0.5558	NaN	0.5558	NaN
	AdaGrad	0.5619	0.4754	0.5619	NaN
	SGDM	0.5608	NaN	0.5608	NaN
	RMSProp	0.5600	NaN	0.5600	NaN
	ADAM	0.5583	NaN	0.5583	NaN
	Ours	**0.3042**	**NaN**	**0.3042**	**NaN**

models' performance. The results are summarized in Table 7. This study focuses on highlighting how the proposed method outperforms the others, with a detailed discussion of each performance metric for the different neural network models tested: SLP, MLP, CNN, and LSTM.

Accuracy comparison. The proposed method with the Gauss–Newton custom optimizer demonstrated superior accuracy across all neural network models evaluated. As presented in Table 7, the accuracy metrics reveal that our method consistently achieves the highest accuracy rates. For instance, in the SLP model, our method achieved an accuracy of 0.9789, significantly outperforming Adam (0.9633) and SGDM (0.9669). This trend is also observed across other models, with the CNN model reaching an accuracy of 0.9633 with the proposed method, compared to 0.9603 with Adam and 0.9372 with SGD.

Precision analysis. Precision is crucial for evaluating model performance, particularly in classification tasks where false positives can be costly. The proposed method showed notable improvements in precision. For instance, the proposed method in MLP model achieved a precision of 0.9499, whereas Adam and SGDM achieved 0.9558 and 0.9575, respectively. This demonstrates that the proposed method effectively reduces the occurrence of false positives, leading to more reliable model predictions.

Recall analysis. Recall, which measures the ability of the model to identify all relevant instances, also saw significant enhancement with the proposed method. In the CNN model, the proposed method achieved a recall of 0.9633, compared to Adam's 0.9603 and SGDM's 0.9208. This improvement is particularly beneficial for applications where missing a positive instance can have severe consequences, highlighting the robustness of the proposed method.

F1-score analysis. The F1-score, which balances precision and recall, further highlights the efficiency of the proposed method. Across all models, the F1-scores were consistently higher with the proposed method. For example, the SLP model achieved an F1-score of 0.9789 with the proposed method, surpassing Adam's 0.9634 and SGDM's 0.9670. This indicates that the proposed method not only enhances the detection of positive instances but also maintains a balance between precision and recall, making it highly effective for classification tasks.

The superior performance of the proposed method can be attributed to its effective handling of the optimization landscape, allowing for more precise adjustments in the weights of the neural networks. This results in faster convergence and lower generalization error. The consistent improvements across different metrics and neural network models underscore the robustness and versatility of the proposed method. Moreover, the enhanced accuracy, precision, recall, and F1-scores validate the efficacy of the proposed method in producing reliable and accurate classification results, making it a valuable tool for tasks that require high performance.

However, it is important to note that the proposed method's performance in the LSTM model was notably lower, with an accuracy of 0.3042. This discrepancy suggests that the Gauss–Newton optimizer may need further refinement or adjustment when applied to recurrent neural networks, particularly for complex sequential data. This area warrants further investigation to enhance the optimizer's applicability across a broader range of neural network architectures.

5.5. Further discussion

By far, the proposed models have shown good performances on SLP, MLP, and CNN models. Evaluating the performance of each optimizer across all neural network models reveals that QRGN achieves the best performance in terms of accuracy, precision, recall, and f1-score, particularly on the SLP model. However, we also recognize that the proposed model requires high computational runtime to train in every model, including SLP.

The significant computational time in the second order optimization is caused by calculating the Jacobian matrix at each iteration for weight updates. The Jacobian matrix calculation needs to evaluate the gradient of the loss function with respect to all parameters in the batch size. Even by using the automatic differentiation method, the proposed model finds it hard to compute the entries of the Jacobian matrix simultaneously (very heavy computation).

To address the above issues, we re-evaluated the gradient derivation on the SLP by implementing the Jacobian matrix construction from the proposed optimization method. To know how efficient the proposed method, we analyze the computational complexity. Given an SLP with input $\mathbf{x}$, weights $\mathbf{w}$, and bias b, the output $\hat{\mathbf{y}}$ is computed as:

$$\hat{\mathbf{y}} = \sigma(\mathbf{x}\mathbf{w} + b)$$

with σ is the activation function. Typically, a softmax function is used as it fits for multi-class classification tasks. The loss function $\mathcal{L}$, such as: cross-entropy, is defined as:

Table 8
The computational time required by autodiff and gradient formulas to train the SLP.

Optimization method	Accuracy		Epoch	Computational time (s)	
	Train	Test		By Autodiff	By Eq. (11)
SGD	0.9800	0.9547	197	3,235.48	367.49
AdaGrad	0.9850	0.9508	307	5,708.66	356.30
SGDM	0.9800	0.9669	48	832.63	116.11
RMSProp	0.9750	0.9458	48	882.72	122.23
Adam	0.9750	0.9633	43	790.81	126.58
Ours	**0.9821**	**0.9789**	**22**	**2,907.85**	**10.92**

$$\mathcal{L}(\hat{\mathbf{y}}, \mathbf{y}) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K} y_{ik}\log(\hat{y}_{ik})$$

To optimize the weights $\mathbf{w}$, the gradient of the loss function w.r.t. $\mathbf{w}$ is required:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \frac{\partial \mathcal{L}}{\partial \hat{\mathbf{y}}} \cdot \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{z}} \cdot \frac{\partial \mathbf{z}}{\partial \mathbf{w}}$$

with $\mathbf{z} = \mathbf{x}\cdot\mathbf{w} + b$. For the Jacobian matrix A, each element is the partial derivative of the loss function w.r.t. each parameter:

$$A_{ij} = \frac{\partial \mathcal{L}_i}{\partial w_j}$$

.

Intuitively, the computation is very costly as it needs to calculate the element-wise on the Jacobian matrix or element-by-element. Hence, this study leverages factorization matrix. For instance, the gradient of the chain rule is transformed into a vectorized form, as follows:

$$\nabla_{\mathbf{w}}\mathcal{L} = \mathbf{x}^{\top}(\hat{\mathbf{y}} - \mathbf{y}) \tag{11}$$

Further step modifies the calculation of Eq. (11) by considering the entire batch instead of individual instance. Accordingly, Eq. (11) is solved using the vectorized gradient approximation. For a batch size m, the Jacobian matrix A for each sample can be combined to form a batch Jacobian matrix. At the final step, the optimized Jacobian matrix is substituted in the QRGN optimization loops to ensure the matrix operations are compatible with the existing QRGN implementation. As the outcomes, the computational time can be reduced significantly. Also, it maintains the high predictions. Therefore, the proposed method is more practical in real-world applications due to its efficiency.

Table 8 compares the computational time required by the automatic differentiation method and the optimized gradient formula method for training the SLP model. Results in Table 8 clearly show that the optimized gradient formulas significantly reduce the computational time required to train the SLP model. This optimization brings the QRGN method's computational time closer to that of baseline optimizers, making it a more feasible option for practical applications while maintaining superior performance metrics.

Furthermore, employing auto-differentiation to compute the gradient took significantly longer than using the gradient formula across all methods. We recommend utilizing the gradient formula, as it is more efficient for gradient computation in the context of second-order optimization methods. From the test results, the proposed method outperforms first-order ones in achieving the desired accuracy, even though the number of iterations varied among methods. On the other hand, it is confirmed that the computational time of our proposed method is 34, 33, 10, 11, and 11 times faster than SGD, AdaGrad, SGDM, RMSProp, and Adam, respectively. These results underscore the significant efficiency of using the gradient formula for gradient computation over auto-differentiation, particularly for the second-order method.

6. Conclusion

In this study, we propose an efficient second-order optimization method by integrating Gauss–Newton and QR factorization into neural networks for classifying SARS-CoV-2 variants given its spike protein sequences. The proposed method is called QR-GN. The proposed method was evaluated from public datasets for SARS-CoV-2 protein sequences. The evaluations are compiled in three series based on accuracy, run time, and memory usage within different NNs, as well as optimization techniques.

The first series shows that the proposed method adapts very well to various NNs by predicting the SARS-CoV-2 variants more accurately than the baseline optimization methods. In the second series, the proposed method significantly boosts the learning process by having the fastest convergence and computational time for most NNs. The third series is mainly focused on second-order optimization method comparisons, where the proposed method can dramatically reduce memory usage. Further investigations on recurrent neural network types are needed as they showed unstable performances in a few moments. Yet, the proposed model successfully worked on complex biological sequences efficiently with high accuracy.

Future directions of this study can be explored on further derivation for more diverse neural networks especially for sequence-based models, computational efficiency methods on Gauss–Newton optimization, and hybrid optimization approaches. Moreover, the proposed model can also be applicable to wider applications such as mutation prediction, activity recognition, and smart manufacturing.

CRediT authorship contribution statement

Mohammad Jamhuri: Writing – original draft, Visualization, Data curation. **Mohammad Isa Irawan:** Supervision, Funding acquisition, Conceptualization. **Imam Mukhlash:** Supervision, Methodology, Formal analysis. **Mohammad Iqbal:** Writing – review & editing, Validation, Investigation. **Ni Nyoman Tri Puspaningsih:** Writing – review & editing, Supervision.

Declaration of Generative AI and AI-assisted technologies in the writing process

During the preparation of this study, the authors used Grammarly to improve language and readability. After using this tool/service, the authors reviewed and edited the content as needed and took full responsibility for the publication's content.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Mohammad Isa Irawan reports administrative support and writing assistance were provided by Ministry of Research, Technology, and Higher Education, Republic of Indonesia. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Funding and acknowledgments

We gratefully acknowledge the financial support provided by the Ministry of Research, Technology, and Higher Education, Republic of Indonesia, under contract number 1928/PKS/ITS/2023, which has made this research possible.

Data availability

Data will be made available on request.

References

[1] J. Zheng, SARS-CoV-2: an emerging coronavirus that causes a global threat, Int. J. Biol. Sci. 16 (10) (2020) 1678.

[2] L. Duan, Q. Zheng, H. Zhang, Y. Niu, Y. Lou, H. Wang, The SARS-CoV-2 spike glycoprotein biosynthesis, structure, function, and antigenicity: Implications for the design of spike-based vaccine immunogens, Front. Immunol. 11 (October) (2020) 1–12, http://dx.doi.org/10.3389/fimmu.2020.576622.

[3] F.X. Heinz, K. Stiasny, Distinguishing features of current COVID 19 vaccines knowns and unknowns of antigen presentation and modes of action, 2021.

[4] S. Zahmatkesh, M. Sillanpaa, Y. Rezakhani, C. Wang, Review of concerned SARS-CoV-2 variants like Alpha (B. 1.1. 7), Beta (B. 1.351), Gamma (P. 1), Delta (B. 1.617. 2), and Omicron (B. 1.1. 529), as well as novel methods for reducing and inactivating SARS-CoV-2 mutants in wastewater treatment facilities, J. Hazard. Mater. Adv. (2022) 100140.

[5] K. Tao, P.L. Tzou, J. Nouhin, R.K. Gupta, T. de Oliveira, S.L. Kosakovsky Pond, D. Fera, R.W. Shafer, The biological and clinical significance of emerging SARS-CoV-2 variants, Nature Rev. Genet. 22 (12) (2021) 757–773, URL https://doi.org/10.1038/s41576-021-00408-x.

[6] O.P. Singh, M. Vallejo, I.M. El-Badawy, A. Aysha, J. Madhanagopal, A.A. Mohd Faudzi, Classification of SARS-CoV-2 and non-SARS-CoV-2 using machine learning algorithms, Comput. Biol. Med. 136 (2021) 104650, http://dx.doi.org/10.1016/j.compbiomed.2021.104650, URL https://www.sciencedirect.com/science/article/pii/S0010482521004443.

[7] S. Ali, B. Sahoo, N. Ullah, A. Zelikovskiy, M. Patterson, I. Khan, A k-mer based approach for SARS-CoV-2 variant identification, in: Bioinformatics Research and Applications: 17th International Symposium, ISBRA 2021, Shenzhen, China, November 26–28, 2021, Proceedings 17, Springer, 2021, pp. 153–164.

[8] M. Togrul, H. Arslan, Detection of SARS-CoV-2 main variants of concerns using deep learning, in: 2022 Innovations in Intelligent Systems and Applications Conference, ASYU, 2022, pp. 1–5, http://dx.doi.org/10.1109/ASYU56188.2022.9925559.

[9] G.B. Câmara, M.G. Coutinho, L.M.d. Silva, W.V.d.N. Gadelha, M.F. Torquato, R.d.M. Barbosa, M.A. Fernandes, Convolutional neural network applied to SARS-CoV-2 sequence classification, Sensors 22 (15) (2022) 5730.

[10] W. Ullah, A. Ullah, K.M. Malik, A.K.J. Saudagar, M.B. Khan, M.H.A. Hasanat, A. AlTameem, M. AlKhathami, Multi-stage temporal convolution network for COVID-19 variant classification, Diagnostics 12 (11) (2022) 2736.

[11] A. Whata, C. Chimedza, Deep learning for SARS COV-2 genome sequences, IEEE Access 9 (2021) 59597–59611, http://dx.doi.org/10.1109/ACCESS.2021.3073728.

[12] M.M. Taye, Understanding of machine learning with deep learning: architectures, workflow, applications and future directions, Computers 12 (5) (2023) http://dx.doi.org/10.3390/computers12050091, URL https://www.mdpi.com/2073-431X/12/5/91.

[13] R.-Y. Sun, Optimization for deep learning: An overview, J. Oper. Res. Soc. China 8 (2) (2020) 249–294.

[14] Z. Wu, M. Xiao, C. Fang, Z. Lin, Designing universally-approximating deep neural networks: A first-order optimization approach, IEEE Trans. Pattern Anal. Mach. Intell. (2024).

[15] R. Dwivedi, V.K. Srivastava, 12 - fundamental optimization methods for machine learning, in: T. Goswami, G. Sinha (Eds.), Statistical Modeling in Machine Learning, Academic Press, 2023, pp. 227–247, http://dx.doi.org/10.1016/B978-0-323-91776-6.00005-1, URL https://www.sciencedirect.com/science/article/pii/B9780323917766000051.

[16] P. Dvurechensky, S. Shtern, M. Staudigl, First-order methods for convex optimization, EURO J. Comput. Optim. 9 (2021) 100015, http://dx.doi.org/10.1016/j.ejco.2021.100015, URL https://www.sciencedirect.com/science/article/pii/S2192440621001428.

[17] A.S. Berahas, M. Jahani, P. Richtárik, M. Takáč, Quasi-Newton methods for machine learning: forget the past, just sample, Optim. Methods Softw. 37 (5) (2022) 1668–1704.

[18] M. Jamhuri, I. Mukhlash, M.I. Irawan, Performance improvement of logistic regression for binary classification by Gauss-Newton method, in: Proceedings of the 2022 5th International Conference on Mathematics and Statistics, 2022, pp. 12–16.

[19] H. Li, Z. Xu, G. Taylor, C. Studer, T. Goldstein, Visualizing the loss landscape of neural nets, Adv. Neural Inf. Process. Syst. 31 (2018).

[20] R. Anil, V. Gupta, T. Koren, K. Regan, Y. Singer, Scalable second order optimization for deep learning, 2020, arXiv preprint arXiv:2002.09018.

[21] I. Muhammad, I. Mukhlash, M. Jamhuri, M. Iqbal, M.I. Irawan, Classification of covid-19 variants using boosting algorithm, in: 2022 9th International Conference on Electrical Engineering, Computer Science and Informatics, EECSI, IEEE, 2022, pp. 29–34.

[22] H. Gunasekaran, K. Ramalakshmi, A. Rex Macedo Arokiaraj, S. Deepa Kanmani, C. Venkatesan, C. Suresh Gnana Dhas, Analysis of DNA sequence classification using CNN and hybrid models, Comput. Math. Methods Med. 2021 (1) (2021) 1835056.

[23] M.M. Hossain, M.A.A. Walid, S.S. Galib, M.M. Azad, W. Rahman, A. Shafi, M.M. Rahman, Covid-19 detection from chest ct images using optimized deep features and ensemble classification, Syst. Soft Comput. 6 (2024) 200077.

[24] E. Yektadoust, A. Janghorbani, A.F. Talebi, XCNN-SC: Explainable CNN for SARS-CoV-2 variants classification and mutation detection, Comput. Biol. Med. 167 (2023) 107606.

[25] O.I. Awe, h.o. obura, M.J. Mwanga, M. Evans, Enhanced deep convolutional neural network for SARS-CoV-2 variants classification, 2023, BioRxiv 2023-2008.

[26] M. Jamhuri, M.I. Irawan, I. Mukhlash, N.N.T. Puspaningsih, CNN-based detection of SARS-CoV-2 variants using spike protein hydrophobicity, in: 2023 1st International Conference on Advanced Engineering and Technologies, ICONNIC, IEEE, 2023, pp. 207–212.

[27] C.-C. Wang, K.L. Tan, C.-J. Lin, Newton methods for convolutional neural networks, ACM Trans. Intell. Syst. Technol. (TIST) 11 (2) (2020) 1–30.

[28] P. Xu, F. Roosta, M.W. Mahoney, Newton-type methods for non-convex optimization under inexact Hessian information, Math. Program. 184 (1) (2020) 35–70.

[29] C. Chen, S. Reiz, C.D. Yu, H.-J. Bungartz, G. Biros, Fast approximation of the Gauss-Newton Hessian matrix for the multilayer perceptron, SIAM J. Matrix Anal. Appl. 42 (1) (2021) 165–184.

[30] B.M. Ozyildirim, M. Kiran, Levenberg-marquardt multi-classification using hinge loss function, Neural Netw. 143 (2021) 564–571.

[31] F. Mehmood, S. Ahmad, T.K. Whangbo, An efficient optimization technique for training deep neural networks, Mathematics 11 (6) (2023) 1360.

[32] A.D. Adeoye, P.C. Petersen, A. Bemporad, Regularized Gauss-Newton for optimizing overparameterized neural networks, 2024, arXiv preprint arXiv:2404.14875.

[33] J.F. Yan, A.K. Yan, B.C. Yan, Prime numbers and the amino acid code: analogy in coding properties, J. Theoret. Biol. 151 (3) (1991) 333–341.

[34] N. Kosaraju, S.R. Sankepally, K. Mallikharjuna Rao, Categorical data: need, encoding, selection of encoding method and its emergence in machine learning models—a practical review study on heart disease prediction dataset using Pearson correlation, in: Proceedings of International Conference on Data Science and Applications: ICDSA 2022, Volume 1, Springer, 2023, pp. 369–382.

[35] E.W. Sayers, J. Beck, E.E. Bolton, D. Bourexis, J.R. Brister, K. Canese, D.C. Comeau, K. Funk, S. Kim, W. Klimke, et al., Database resources of the national center for biotechnology information, Nucleic Acids Res. 49 (D1) (2021) D10.

[36] M. Salehi-Vaziri, M. Fazlalipour, S.M. Seyed Khorrami, K. Azadmanesh, M.H. Pouriayevali, T. Jalali, Z. Shoja, A. Maleki, The ins and outs of SARS-CoV-2 variants of concern (VOCs), Arch. Virol. 167 (2) (2022) 327–344.

[37] P. Chaudhari, H. Agarwal, V. Bhateja, Data augmentation for cancer classification in oncogenomics: an improved KNN based approach, Evol. Intell. 14 (2) (2021) 489–498, URL https://doi.org/10.1007/s12065-019-00283-w.

[38] T.-H. Zhang, M. Flores, Y. Huang, ES-ARCNN: Predicting enhancer strength by using data augmentation and residual convolutional neural network, Anal. Biochem. 618 (2021) 114120, http://dx.doi.org/10.1016/j.ab.2021.114120, URL https://www.sciencedirect.com/science/article/pii/S000326972100021X.

[39] S. Zhang, E. Baharlouei, P. Wu, High accuracy matrix computations on neural engines: a study of qr factorization and its applications, in: Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing, 2020, pp. 17–28.

[40] T. Sauer, Numerical Analysis, Addison-Wesley Publishing Company, 2011, pp. 220–234.

[41] H. Robbins, S. Monro, A stochastic approximation method, Ann. Math. Stat. (1951) 400–407.

[42] J. Duchi, E. Hazan, Y. Singer, Adaptive subgradient methods for online learning and stochastic optimization., J. Mach. Learn. Res. 12 (7) (2011).

[43] B.T. Polyak, Some methods of speeding up the convergence of iteration methods, Ussr Comput. Math. Math. Phys. 4 (5) (1964) 1–17.

[44] T. Tieleman, G. Hinton, et al., Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude, COURSERA: Neural Netw. Mach. Learn. 4 (2) (2012) 26–31.

[45] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, 2014, arXiv preprint arXiv:1412.6980.

[46] L. Prechelt, Early stopping-but when? in: Neural Networks: Tricks of the Trade, Springer, 2002, pp. 55–69.