Received: 20 September 2024 | Accepted: 18 January 2025

DOI: 10.1049/tje2.70060

The Journal of Engineering

The Institution of Engineering and Technology WILEY

ORIGINAL RESEARCH

Exploring optimizer efficiency for facial expression recognition with convolutional neural networks

Syed Hamid Hussain Madni[1] | **Lokessh A/L Pathmanatan**[2] | **Muhammad Faheem**[3,4] | **Hafiz Muhammad Faisal Shahzad**[5] | **Sajid Shah**[2]

[1]School of Electronics and Computer Science, University of Southampton Malaysia, Johor, Malaysia

[2]Faculty of Computing, Universiti Teknologi Malaysia, Skudai, Johor, Malaysia

[3]School of Technology and Innovations, University of Vaasa, Vaasa, Finland

[4]VTT Technical Research Centre of Finland Ltd., Espoo, Finland

[5]Department of Computer Science, University of Sargodha, Sargodha, Pakistan

Correspondence
Muhammad Faheem, School of Technology and Innovations, University of Vaasa, Vaasa, Finland.
Email: muhammfa@uwasa.fi

Abstract

It's widely accepted that human expressions, considering for roughly sixty percent of all daily interactions, are among the most authentic forms of communication. Numerous studies are being conducted to explore the importance of facial expressions and the development of machine-assisted recognition techniques. Significant progress is being made in facial and expression recognition, largely due to the rapid growth of machine learning and computer vision. A variety of algorithmic approaches and methods exist for detecting and recognizing facial expressions and features. This study investigates various optimization algorithms used with convolutional neural networks for facial expression recognition. The primary focus is on Adam, RMSProp, stochastic gradient descent and AdaMax optimizers. A comprehensive comparison is being made, examining the key aspects of each optimizer, including its advantages and disadvantages. Furthermore, the study also incorporates findings from recent studies that used these optimizers in various applications, highlighting their performance in terms of training time and precision. The aim is to illuminate the process of selecting a suitable optimizer for specific applications, analysing the trade-offs between training speed and higher accuracy levels. Moreover, this study provides a deeper analysis of the role optimizers play in machine learning-based facial expression recognition models. The discussion of the technical challenges posed by these optimizers and future improvements for achieving much more optimal results concludes the study.

1 | INTRODUCTION

Facial recognition and detection technology is going through major iterations. Nowadays, our smartphones are also able to use our faces as an authentication method to unlock our devices. Faces like fingerprints are unique, with millions of unique traits that differentiate them from one another. Machines and other systems will analyse and evaluate all this data, which leads to a more accurate output. Unfortunately, just like other data-based models the facial detection system is still not 100% perfect. Nevertheless, it has almost reached a stage where it is commercially respectable in our daily lives Bui et al. [1]. Facial detection can be used for many things from unlocking phones, authorizing downloads, protecting data, approving transactions and many more. The increase in the amount of data being processed has had a huge effect on information technology development. From the earliest phases of computer vision development, researchers have been working on an alternative to solve the problem of recognizing a person based on face image analysis [2].

"Facial detection" refers to the process of recognizing and localizing human faces that are included in digital still photographs and moving movies. This technology can be utilized for a wide number of reasons, including entertainment, surveillance, social networking, and even safety [3]. There has been a significant amount of development in face detection over the course of the years, with several algorithms and methods being presented to handle the issues of detecting faces in contexts that are complex and variable. These advancements have allowed for a greater number of people to be able to identify faces. The application of face detection technology may be traced back to the 1970s, when the first automated facial recognition systems

were built. Since that time, facial detection technology has been used in a variety of settings [4]. Moreover, the commencement of the Internet can be attributed to this decade. These early systems depended on easy template matching methods, in which a photo was first supplied into the system, and then that image was compared to a database of face templates to identify a match. These early systems relied on template-matching methods. These algorithms have a limited degree of precision and were only able to differentiate between individuals' faces in environments that were extremely well-orchestrated. The decade of the 1990s witnessed the beginning of an expansion of the application of facial recognition technology outside of the realm of security and surveillance [5]. This development came about because of technological advancements. There has been a surge in demand for automated technologies that are able to detect and recognize faces in digital images. This need can be attributed to the widespread adoption of digital cameras and the internet.

The initial step in facial analysis is face detection, which then leads to all the other facial analysis methods such as face recognition, face verification, face modelling, face alignment, face tracking, and so on. Face detection is the first step [6]. Face detection is described as a categorization of patterns in which the input is an image, and the ultimate output of the class will be chosen based on the image. This means that the input and output of face detection are both determined by the image. To put it another way, the class takes an image as its input, and the image itself determines what the class produces. For a computer system to be able to detect and recognize whether the image being displayed is a face or something else, it needs to be trained to employ computer vision and machine learning. This training takes some time. Face recognition can also be broken down into four distinct subprocesses, which are known as appearance-based techniques, knowledge-based methods, template-based methods, and feature-invariant methods [7]. During the process of detection, the system considers several different facial variations and features. These include the shape of the face, the colour of the eyes, the colour of the skin, the shape of the lips, the size of the nose, as well as other characteristics of the face (Figure 1).

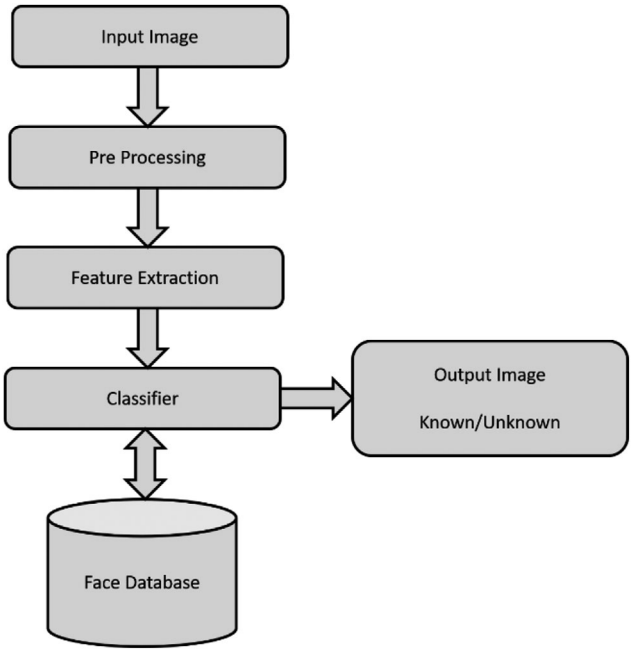

FIGURE 1 General facial detection method.

1.1 | Face detection

Face detection apps are able to identify human faces in photographs that also contain other components, such as landscapes, buildings, and other human body parts like feet or hands [8]. This is made possible by machine learning (ML) and algorithms used by face detection apps. Because human faces are easily distinguishable from other facial features, this is feasible. According to [9], one type of algorithm that can be utilized for the purpose of detecting faces is referred to as a convolutional neural network (CNN). In the process of analysing images, the use of CNNs is common [10]. CNNs are a type of artificial neural network. They are designed to be able to take data in the form of images as input and process that data through multiple layers, each of which extracts different features from the input data [11]. This capability is built into their architecture.

After that, the results of the very last layer are put to use for the purpose of making a prediction or carrying out a classification. During the training process, CNNs can be taught to recognize faces in the context of face detection by being shown a large dataset of images [11]. This allows CNNs to learn how to identify faces [12]. One component of the training process is the modification of the network's weights and biases. This is done with the intention of bringing the amount of variance that exists between the predicted output and the actual output down to a more manageable level. After it has been trained, the CNN can be used to recognize faces in new images by feeding the image into the network and making a prediction based on the output from the network's final layer [13].

When it comes to recognizing faces, CNNs offer many advantages. They are able to adapt to changes in lighting, pose, and expression [14], and they can learn to recognize faces even when they are obscured in some way [15]. These abilities have been demonstrated by research conducted by [14]. Additionally, their accuracy can be improved by training them on large datasets [16] and they can be fine-tuned to meet the requirements of a variety of applications [17]. Additionally, their accuracy can be improved by training them on large datasets.

Face detection techniques, such as CNNs, have advanced significantly due to improvements in machine learning algorithms and the availability of large datasets for training [10, 16]. By utilizing CNNs, face detection apps can effectively identify human faces in various contexts, such as photographs containing landscapes, buildings, and other body parts [18]. CNNs are highly adaptable to different lighting conditions, poses, expressions, and occlusions, making them a versatile and powerful tool for face detection applications [14]. As research in this field continues to progress, we can expect further enhancements

in the accuracy, efficiency, and applicability of face detection technologies.

1.2 | FER

The identification and analysis of human emotions through facial expressions is a component of computer vision that is referred to as facial expression detection [19]. The precise identification and comprehension of facial expressions is a crucial component of human interaction, with wide-ranging implications across diverse domains such as healthcare, education, entertainment, and marketing. The process of detecting facial expressions encompasses multiple stages, including the identification of facial features, the extraction of relevant features, and the classification of emotions [20]. Initially, a facial detection algorithm is utilized to recognize and determine the precise location of the face in an image or video. Subsequently, the facial characteristics encompassing the eyes, eyebrows, nose, and mouth are extracted and scrutinized to discern facial expressions. The expressions that have been identified are subsequently categorized into distinct emotional categories, including but not limited to happiness, sadness, anger, surprise, fear, and disgust [21].

There are two primary methods used in facial expression detection: geometric feature-based and appearance-based approaches [22]. Geometric feature-based methods for facial expression detection involve calculating the geometric relationships between key facial landmarks, such as the eyes, eyebrows, nose, and mouth [23]. These methods often start with detecting facial landmarks using techniques like the active shape model or active appearance model [24]. The geometric features, such as distances or angles between landmarks, are then extracted and compared with a predefined set of features corresponding to specific facial expressions. One advantage of geometric feature-based methods is their ability to handle variations in face orientation and scale [25]. However, these methods can be sensitive to errors in facial landmark detection, which may impact the overall accuracy of the expression recognition.

Appearance-based methods for facial expression detection focus on the texture and appearance of the face, rather than relying solely on geometric features [26]. Techniques like local binary patterns and Gabor filters can be used to capture the texture information of facial regions, which can then be analysed to identify facial expressions [27]. One advantage of appearance-based methods is their robustness to variations in lighting conditions and facial geometry [28]. However, these methods can be more sensitive to changes in face orientation and scale, which may require additional preprocessing steps for accurate recognition.

Significant strides in deep learning have led to considerable enhancements in the accuracy and efficiency of FER [29]. Deep learning models, especially CNNs and recurrent neural networks, are frequently utilized in this field. CNNs have shown their prowess in autonomously learning and identifying key facial features from raw image data [29]. These networks are composed of multiple layers, including convolutional, pooling, and fully connected layers, which facilitate the capture of both local and global information from facial images. CNNs have demonstrated superior performance in facial expression detection compared to traditional geometric feature-based and appearance-based methods [30]. Furthermore, the concept of multi-network fusion (MNF) based on CNNs has been introduced, which has been shown to be more effective than a single network in FER [31]. By training a deep learning model on a dataset that includes both types of features, the model can autonomously learn to extract and combine relevant information from both, leading to enhanced performance in facial expression detection.

Lately, there has been an increase in the need for real-time, accurate identity identification in many different fields, primarily in security and locating people. The setback of an automated person identification (recognition) system using the face image is far more sophisticated than basic face detection, and no algorithm has yet been developed that can determine a person's identity just by using a face image in real-world situations as well as a normal human can [32]. Year after year, new approaches to localization, processing and recognition of objects are being researched and developed, and yet these approaches do not have sufficient speed, accuracy or reliability in the real environment, which is often detected by the presence of noise in video sequences, and through various shooting conditions, such as the lighting conditions and the photo angles [33].

Based on the explanations above, the research is about determining which optimizer works best with the CNN and has a high accuracy result for FER.

The contributions of this research study are mentioned as follows:

- Improve the understanding of CNNs and their application in FER.
- Highlights the role of optimizer selection in model performance for FER.
- Identify areas that warrant further investigation by conducting a comprehensive literature review.
- Experiment design and execution will provide valuable insights into the performance of various optimizers such as Adam, stochastic gradient descent (SGD), RMSProp, and AdaMax by utilizing a well-established CNN architecture.
- This systematic evaluation will add to the existing body of knowledge and may help guide the selection of optimizers in similar tasks in the future.

The remaining sections of this research study are organized as follows: Section 2 describes the existing relevant reviews and survey studies for performance analysis of optimizers with CNN for FER are studied. Section 3 of the study focuses on the problem formulation that defines the loophole in existing FER systems, specifically addressing the challenge of optimizer selection for CNNs. This section highlights the lack of comprehensive comparative studies on optimizer performance in the context of FER, emphasizing the need for a systematic evaluation of popular optimizers to determine their impact on model accuracy, convergence speed, and overall efficiency.

TABLE 1 Comprehensive review of common optimization algorithms used in the machine learning domain.

References	Optimizers	Problems addressed	Improvement/achievements	Weakness/limitations
Kingma and Ba [34]	Adam	Adaptive learning rates, momentum, bias correction	Excels in rapid convergence and is well-suited for handling large datasets and sparse gradients	Hyperparameters can significantly influence its performance, and it may not always find the optimal solution
Liu et al. [35]	Adam	Adaptive learning rates, momentum, bias correction	Exhibits quick convergence and performs well with large datasets and sparse gradients	Sensitivity to hyperparameters is a concern, and it may settle on suboptimal solutions
Zou et al. [36]	RMSProp	Running average of squared gradient, Adaptive Learning Rates	Performs well with sparse gradients, is less hyperparameter-sensitive than Adam, and avoids rapid decrease of learning rate	In some scenarios, its convergence may be slower than desired
Yedida et al. [37]	RMSProp	Running average of squared gradient, adaptive learning rates	Effective for sparse gradients, exhibits less sensitivity to hyperparameters compared to Adam, and maintains a steady learning rate	Its convergence rate can be slow in certain cases
Ziyin et al. [38]	SGD	Gradient calculation, learning rate	Its simplicity and scalability are key strengths, and it can reach the global optimum with appropriate tuning	It may converge slowly, get trapped in saddle points, and is sensitive to learning rate
Ziyin et al. [38]	SGD	Gradient calculation, learning rate	Notable for its simplicity and scalability, and can reach the global optimum with the right tuning	Its convergence can be slow, it may get stuck in saddle points, and the learning rate can significantly affect its performance
He et al. [39]	SGD	Gradient calculation, learning rate	Achieves improved accuracy in deep learning tasks	As network depth increases, so does its complexity
Wang et al. [40]	SGD	Gradient calculation, learning rate	Enhances recognition performance and reduces time-cost	The learning rate needs to be carefully selected for optimal performance
Zeng et al. [41]	AdaMax	Adaptive learning rates, momentum, infinity norm	Less influenced by outliers, exhibits rapid convergence, and is ideal for large datasets and sparse gradients	Its performance can be affected by hyperparameters, and it may settle on suboptimal solutions

Furthermore, Section 4 focuses on methodology of preparing this research study. Section 5 of the research study demonstrates the development and implementation of an optimized CNN for facial emotion recognition (FER). This section covers the design of the CNN architecture, compares unoptimized and optimized CNN models, and outlines the complete processing pipeline from data collection to result interpretation and reporting. Section 6 represents the analysis and Computational Results of this study. Additionally, Section 7 sheds light on the performance analysis and Section 8 of the study demonstrates the suggestions for further works. Moreover Section 9 explain the discussion and lastly Section 10 is about conclusion.

2 | RELATED WORKS

In this section, the existing relevant reviews and survey studies for performance analysis of optimizers with CNNs for FER are studied. Table 1 shows the most common optimization algorithms used in the machine learning domain which are Adam, RMSProp, SGD and AdaMax. A comprehensive comparison of these four algorithms is provided in the table that has been provided. Each optimizer can be described in terms of its essential components, as well as its advantages and disadvantages.

The adaptive learning rates, momentum, and bias correction were all incorporated into Adam in 2014 when it was first introduced [42]. It converges quickly, is appropriate for use with large datasets, and is effective when applied to sparse gradients. These are its primary advantages. Nevertheless, Adam is susceptible to being sensitive to hyperparameters and may converge to solutions that are less than ideal. Adam's training can be completed in a relatively short amount of time, and it has a strong track record of accuracy in facial recognition tasks [43].

The RMSProp algorithm, which was first presented to the public in 2012, is predicated on a running average of squared gradients in conjunction with adaptive learning rates [44]. The capability of RMSProp to stop the rate of learning from falling too quickly is a significant benefit offered by this algorithm. On the other hand, there are circumstances in which it may converge at a slower rate than competing optimizers. Both the amount of time it takes to train RMSProp and the level of accuracy it demonstrates in facial detection tasks are considered to be moderate [44].

The SGD algorithm, which was first proposed in the 1950s, is an essential component of the machine learning field of optimization [45]. Calculation of gradients and a user-specified learning rate are the two pillars upon which it stands. The simplicity of SGD and how straightforward its implementation is are two of its primary selling points. SGD has the potential to converge to a global optimum with the application of the appropriate tuning. On the other hand, the algorithm has some flaws, such as a slow convergence, the possibility of becoming mired in saddle points, and a sensitivity to the learning rate selection. When compared to other optimizers, the training time for SGD

is typically longer, and its accuracy in facial detection tasks is typically lower. Moreover, the SGD algorithm has been shown to produce less accurate results [46].

AdaMax is a machine learning algorithm that was developed in 2015 and uses adaptive learning rates, momentum, and the infinity norm [41]. This optimizer is less sensitive to outliers and provides fast convergence, making it suitable for use with large datasets and sparse gradients. In a manner analogous to Adam, the AdaMax algorithm can be sensitive to the values of its hyperparameters and may converge on solutions that are less than ideal. AdaMax has a short period required for training, and it has exhibited a high degree of accuracy in facial detection tasks [47].

In a nutshell, the decision regarding which optimizer to use is determined by the nature of the dataset as well as the particular problem that is being tackled [47]. Although all four optimizers have their advantages and disadvantages, Adam and AdaMax are generally better suited for larger and more complex problems, while RMSProp and SGD are better suited for simpler problems. When selecting an optimizer for a specific application, researchers and practitioners should consider the trade-offs between the amount of time spent training and the accuracy of the results [48].

Although several studies have investigated the use of individual optimizers such as Adam, SGD, RMSProp, and AdaMax, there have been relatively few comprehensive comparative studies that have examined these optimizers side by side in the context of FER. In addition, the influence of characteristics of facial expression datasets on the performance of these optimizers is another area that has not been thoroughly investigated. This is one of the many areas that could be improved.

By carrying out a comprehensive analysis of these optimizers within the framework of CNN-based FER, this study intends to close these existing knowledge gaps. The findings from this research will provide valuable insights for future research as well as practical applications, and they will also contribute to the existing body of knowledge.

3 | PROBLEM STATEMENT

The ability to recognize facial expressions accurately and efficiently is crucial for the development of intelligent systems that can effectively interact with humans in various social contexts. CNNs have been successfully employed in recent years for FER tasks due to their remarkable performance in image and pattern recognition. However, the optimization of CNNs remains an open challenge, as selecting the appropriate optimizer for training can significantly impact the network's accuracy, convergence speed, and computational efficiency. Given the diverse range of optimizers available for training CNNs, determining the most suitable optimizer for FER tasks remains an unresolved issue.

This study aims to compare the performance of four widely used optimizers - Adam, SGD, RMSProp, and AdaMax - in the context of FER using CNNs, and to identify the most effective optimizer for enhancing the accuracy and efficiency of the model. The outcomes of this study provide valuable insights into the impact of optimizer selection on the performance of CNNs for FER tasks. By identifying the most effective optimizer for this specific application, the aim is to contribute to the development of more accurate and efficient models that can ultimately enhance the capabilities of intelligent systems in interpreting and responding to human emotions. Furthermore, the findings of this study may also have broader implications for the optimization of CNNs in various other image and pattern recognition tasks, as the effectiveness of different optimizers may vary depending on the specific problem domain.

The high dimension space of FER is the primary challenge that must be overcome, and any technique that may accomplish this goal should be utilized. The facial detection system relies on clear and easily detectable images to classify the expressions but is still affected by conditions such as poses and lighting and the overall clarity of the data.

4 | METHODOLOGY

4.1 | Data preparation

4.1.1 | Dataset

For this research, a publicly available dataset from Kaggle, known as FER 2013 [49] has been utilized. This dataset is widely recognized in the field of FER due to its rich collection of annotated facial expressions, which include Angry, Disgust, Fear, Happy, Sad, Surprise, and Neutral. However, upon preliminary analysis, it has been observed that the 'Disgust' class has significantly fewer data points compared to the other classes. Such a class imbalance can potentially skew the results and lead to a bias in the model's predictions, where it could favour classes with more data points. This is especially problematic in a multiclass classification problem like the one at hand, where the aim is to achieve a high level of accuracy across all classes.

To address this concern and to foster a more balanced dataset that promotes better learning, the decision has been made to exclude the 'Disgust' class from this study. This step ensures a more equitable distribution of data across the remaining classes, thereby mitigating any undue influence the 'Disgust' class could have had on the model's performance due to its disproportionately smaller size. This refined version of the FER 2013 dataset, now bereft of the 'Disgust' class, will serve as the foundation for the research. This strategic modification is expected to enhance the robustness of the learning process and improve the validity of the research outcomes.

4.1.2 | Data extraction

The expression dataset is first pre-processed to remove noises and artefacts from the images. This is to help the neural network to better process the photos and detect the expression of the faces. Since the images are already in greyscale, we can directly go through the segmentation process. Each of the segmented parts will then be used to train the neural network for the facial recognition process first. This process is primarily divided into three stages, which are training, testing and validation.

4.1.3 | Neural network

The deep learning method employed in this research study is the CNN algorithm, a powerful and versatile approach for tasks such as image classification and recognition. The Kaggle facial expression challenge dataset is used as the primary input for training the CNN model. The structure of the CNN consists of multiple layers, each contributing to the overall performance and accuracy of the model. The following elaborated sections provide an overview of how the CNN algorithm will be implemented in this research.

- **Input**

The CNN algorithm will utilize Kaggle's facial expression challenge dataset as its main input source. Given that the primary focus of the CNN model is facial expression detection and recognition, the input data is represented by a pixel matrix of the images. The dataset has been divided into six nodes, each representing a specific facial expression: 0 = Angry, 1 = Happy, 2 = Neutral, 3 = Fear, 4 = Sad and 5 = Surprise.

- **Convolutional layer**

The convolution process is a mathematical operation that combines two functions, x and y, to produce a new function, Z, representing the overlap between the two original functions [50]. In the context of CNN, x is a 2D array of pixels from the input image, and y is a 2D matrix, also known as a filter or kernel. The information in the filter matrices is used to determine the output image Z [51]. The model's filter and size parameters decide how these values are evaluated. An example of a convolutional layer is as follows:

```
model.add(Conv2D(32, (3, 3), activation = 'relu',
    input_shape = (48, 48, 1)))
```

In this model, there are 32 filters, each with a dimension of 3 × 3. These filters are stacked sequentially, leading to a total of 3 × 3 × 32, or 288 parameters learned by the model.

- **Max Pooling layer**

MaxPooling is a process that aims to down sample the input, typically the output of a Convolution2D layer, to reduce its dimensionality. This helps to decrease the computational load, memory usage, and the number of parameters, which effectively helps in reducing overfitting [52]. In the context of a CNN, MaxPooling2D works by sliding a 2D window across the input and taking the maximum value within that window as the output. The stride parameter defines how much the window shifts by in each step during the sliding process [53]. An example of a MaxPooling2D layer is as follows:

```
model.add(MaxPool2D(pool_size = (2, 2)))
```

In this model, the MaxPool2D layer is down sample the input feature map (or the output of the Conv2D layer that precedes it) by taking the maximum value within each 2 × 2 window. The result is a new feature map with reduced size, both in terms of height and width, which can help in improving the model's performance by focusing on the most important features and reducing the risk of overfitting.

- **Fully connected layer**

The dense layer, also known as the fully connected layer, is a type of layer where each neuron is connected to every neuron in the previous layer. It plays a critical role in learning the high-level patterns in the data, combining the features learned by the previous layers to make the final prediction [54]. In a dense layer, each neuron performs a weighted sum of all its inputs, adds a bias, and then passes the result through a nonlinear activation function. The number of neurons in the dense layer is a parameter that can be tuned, and it often depends on the specific requirements of the model [54]. An example of a dense layer is as follows:

```
model.add(Dense(128, activation = 'relu'))
```

In this model, the dense layer consists of 128 neurons. Each of these neurons takes as input a value from each neuron in the preceding layer, performs a weighted sum of these inputs, adds a bias term, and then passes the result through a 'relu' (rectified linear unit) activation function. The 'relu' activation function is a common choice in deep learning models as it introduces nonlinearity without requiring expensive computations.

- **Batch normalization**

Batch normalization is a technique used to standardize the inputs to a layer in a neural network, helping it learn more effectively. It achieves this by reducing the amount by which the distribution of the input changes during training, a phenomenon known as internal covariate shift [55]. By doing so, it allows each layer of a network to learn by itself a little more independently of other layers, leading to better performance and faster training. Batch normalization operates by normalizing the outputs of a previous activation layer by subtracting the batch mean and dividing by the batch standard deviation. As a result, the distributions of each layer's inputs remain more stable throughout training, which helps accelerate the training process and reduces the chance of getting stuck in local minima during optimization [56]. An example of a batch normalization layer in a Keras model would be as follows:

```
model.add(BatchNormalization())
```

In this model, the BatchNormalization layer is used to normalize the activations of the previous layer at each batch, i.e. it applies a transformation that maintains the mean activation close to 0 and the activation standard deviation close to 1. It is often used after a convolution or dense layer to stabilize the learning process and reduce the number of training epochs required to train deep networks.

- **Dropout layer**

Dropout is a regularization technique for reducing overfitting in neural networks by preventing complex co-adaptations on training data. The principle behind dropout is straightforward: during training, a proportion of the nodes in the layer are randomly "dropped out", meaning that they are excluded from the computation and no learning takes place on these units for the duration of that forward and backward propagation step [57]. This helps to avoid the over-reliance on any one neuron during the learning process and distributes the "knowledge" across a wider range of nodes, making the network more robust and preventing overfitting. The dropout layer in Keras takes a single parameter, the dropout rate, which is a float between 0 and 1. This dropout rate corresponds to the proportion of the neurons to be dropped out [58]. An example of how a dropout layer might look in a Keras model:

```
model.add(Dropout(0.25))
```

In this case, the dropout layer randomly disables 25% of the input units at each update during training time. This means that during each step in the training process, each node has a 25% chance of being temporarily dropped or deactivated. This randomness helps to make the model more robust, as it cannot rely too heavily on any single node and forces the learning to be distributed across all nodes.

4.1.4 | Reflection

The detailed explanation of each layer and component of the CNN model demonstrates the complexity and sophistication of the approach used in this research. By employing a well-structured CNN model and utilizing various techniques, such as batch normalization, dropout layers, and class weights, this research aims to create a robust and accurate FER system. The combination of these elements contributes to the model's adaptability to changes in input data and its ability to generalize well to new, previously unseen images.

Understanding the intricacies of the CNN model and the role of each layer in the process provides valuable insights into the challenges and opportunities associated with FER tasks. Furthermore, the implementation of different optimizers in the model enables a comprehensive evaluation of their impact on the performance and accuracy of the FER system. This research serves as a foundation for further advancements in the field of computer vision and machine learning, paving the way for more effective and efficient applications of CNNs in various domains.

4.2 | Performance metrics

In the context of FER, the application of optimizer performance metrics allows for the comparison and contrast of various optimization algorithms. These metrics are based on accuracy, loss, and recognition rate, and they encompass aspects such as the speed at which the model learns, the number of computational resources that are required, scalability, correctly classified instances, and error minimization during training and validation. This research study divides these performance metrics into three groups with regard to each objective, and then discusses the performance metrics in the subsequent order:

4.2.1 | Accuracy

Accuracy is the proportion of correctly classified instances relative to the total number of instances [59]. A high level of accuracy indicates that the CNN and optimizer have successfully collaborated to accurately identify the displayed facial expressions. By comparing the accuracy of various optimizers, it is possible to gain insight into how various optimizers affect the FER capability of the model.

$$f(x) = 100 \times \frac{(\mathrm{TP} + \mathrm{TN})}{N} \tag{1}$$

where TP denotes true positives, i.e. the number of instances where the model correctly identified a facial expression. TN represents true negatives, the instances where the model correctly identified the absence of a facial expression and N is the total number of instances [59].

4.2.2 | Loss

Loss serves as a crucial metric complementing accuracy. It indicates how effectively the model is able to reduce the number of mistakes made during FER [60]. The loss is the difference between the model's predictions and the actual values, including both training and validation loss. Lower values of loss indicate a more effective collaboration between the CNN and optimizer, resulting in more precise FER. By comparing the loss values associated with different optimizers, it is possible to gain a deeper understanding of their effects on the model's error minimization capability.

$$f(y) = -1/N \,\Sigma \left(Y\_i * \log(P\_i) + (1 - Y\_i) * \log(1 - P\_i)\right) \tag{2}$$

where Y_i denotes the true label for the nth instance and P_i signifies the predicted probability of the nth instance being the positive class. N represents the total number of instances [60].

4.2.3 | Recognition rate

The recognition rate is another important performance metric in this study. While accuracy provides a comprehensive measurement of performance across all classes, recognition rate can provide more specific insights [61]. In FER, it is essential that the model maintains high performance levels on average

while accurately identifying each individual facial expression. Consequently, the recognition rate measures the proportion of instances in which an expression was correctly identified.

$$f\left(\chi\right) = 100 \times \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \tag{3}$$

where TP denotes true positives, i.e. the number of instances where the model correctly identified a facial expression. FN stands for false negatives, the instances where the model incorrectly identified the absence of a facial expression [61] (Tables 2 and 3).

4.3 | Simulation setup requirements

4.3.1 | Hardware

TABLE 2 Hardware requirement.

Hardware	Uses
PC – Ryzen 7 5800x, 32GB 3200Mhz, NVIDIA GeForce RTX 3070 with 8GB VRAM, M.2 SSD.	• Used to research materials for the project • Used to store all necessary documentation and files of the project • Used to run the software and algorithms to analyse and produce results for the project

4.3.2 | Software

TABLE 3 Software requirement

Software	Uses
Microsoft Word	• Used to write the report. • Used to gather all research results and documentation purposes.
Visual Studio Code	• Used to run the algorithm and model for facial detection
OpenCV & Python	• Used to run deep learning model for learning and training
Nvidia CUDA	• Development toolkit for running deep learning models, debugging, optimization and runtime library
Convolutional neural network	• Neural network algorithm to process image data
Keras	• Python interface library for artificial neural networks

OpenCV

OpenCV is an acronym that refers to "open-source computer vision," which describes a library of programming functions with a primary focus on real-time computer vision. It provides access to over 2500 distinct algorithms that were developed specifically for completing tasks associated with the examination of images and videos [62]. These algorithms are capable of performing a wide variety of functions including face detection, object identification, and tracking the movement of the camera. Readability and ease of understanding are two qualities that contribute to Python's popularity as a high-level, general-purpose programming language. Python and OpenCV, when combined, produce a powerful tool that is especially helpful for putting deep learning models into practice and running them [63]. In the context of research on FER, OpenCV is responsible for managing image processing tasks, whereas Python is responsible for executing the learning and training phases of the model [63].

Nvidia CUDA

Nvidia's CUDA, which stands for "compute unified device architecture," is a platform and an application programming interface (API) model designed for parallel computing. In addition to graphical rendering, CUDA is capable of performing general computing tasks by harnessing the power of GPUs manufactured by Nvidia [64]. This results in a significant improvement to the computer's performance. Deep learning makes use of this power to run complex models in a more effective and rapid manner, thereby reducing the amount of time needed for training and enabling real-time predictions. The CUDA development toolkit is used to execute the deep learning models that were created for the purpose of this study. This toolkit offers support for debugging and optimizing, as well as a runtime library that enables model execution on the GPU [64].

CNN

CNNs, which are a subset of deep neural networks are developed specifically for the purpose of processing structured grid data like images. CNNs are advantageous for tasks involving image recognition and classification due to their capacity to learn spatial hierarchies of features automatically and adaptively [65]. These capabilities give CNNs an edge over traditional methods. By applying filters to the unprocessed pixel data of an image, CNNs are able to discover and learn higher-level image features. After that, the model will be able to perform recognition tasks using these features. In the research on recognizing facial expressions, CNNs are used to process image data in order to recognize facial expressions.

Keras

Keras is a Python-based open-source neural network library renowned for its accessibility, modularity, and extensibility. It is designed primarily to enable rapid experimentation with deep neural networks [66]. Keras, which operates on top of TensorFlow, provides a high-level, intuitive API, thereby hiding many lower-level complexities. This simplification permits the design and development process to focus more on strategic elements such as layer selection, activation functions, and optimizer selection [67]. In the context of this research, Keras plays a crucial role in the development and fine-tuning of CNN models for FER, collaborating effectively with the chosen optimizers.

- **NumPy**: This library is essential for high-level mathematical functions and operations on multi-dimensional arrays and

matrices. In this research, NumPy aids in the handling of linear algebra operations which are the backbone of many machine learning algorithms [68].

- **Pandas**: Pandas is a highly efficient library used for data manipulation and analysis. It provides data structures for efficiently storing large datasets and tools for data wrangling and analysis. In this context, pandas are used for processing data and reading CSV files, enabling seamless handling and manipulation of the dataset [69].
- **OS**: This module provides a way of using operating system dependent functionality, like reading, writing, or modifying files. In this research, the OS module is used for interacting with the file system to load the facial expression datasets [70].
- **Matplotlib and Seaborn**: These two libraries are used for data visualization. Matplotlib forms the foundation, providing a flexible and powerful framework for creating a wide range of plots and charts. Seaborn extends Matplotlib, introducing a number of more advanced visualization types and making existing Matplotlib plots more attractive [71]. In this research, these libraries assist in visualizing data distributions, model accuracy, loss curves, and more.
- **TensorFlow**: TensorFlow is a widely used library for high-performance numerical computation, particularly well-suited for large-scale machine learning (and deep learning). Its core is very flexible and can be used across many scientific domains [72].
- **Keras**: Used in conjunction with TensorFlow, Keras provides a high-level, user-friendly API for designing and training deep learning models. In this research, Keras aids in defining, training, and evaluating the CNN model [66].
- **ImageDataGenerator**: Part of TensorFlow's Keras API, the ImageDataGenerator class is used for real-time data augmentation. It configures random transformations and normalization operations to be done on the images when the model is training, which can improve model generalization [73].
- **CV2**: This is the OpenCV library, used for real-time computer vision. The library has more than 2500 optimized algorithms for image and video analysis [74].
- **Regularizes**: These are functions applied to the weights or activations of the neural network layers. Regularization techniques are used to prevent overfitting in the model [75].
- **Adam, RMSprop, SGD, AdaMax**: These are optimization algorithms used to adjust model parameters to minimize the model error. Each of them is used in this research to evaluate their impact on the model performance [76].
- **Sklearn.metrics**: This module implements several loss, score, and utility functions to measure classification performance, including confusion matrix and classification report. In this research, these functions are used to quantify the performance of the model in terms of its accuracy, precision, recall, and f-score [77].
- **Matplotlib.colors**: This module is part of the Matplotlib library, and it is used to conveniently change or manipulate colours in the plots [78].

5 | DEVELOPMENT AND IMPLEMENTATION OF AN OPTIMIZED CNN FOR FER

FER is a complex task, requiring the accurate identification of emotional states from facial expressions. It is a critical function in a variety of applications, ranging from human-computer interaction to mental health monitoring (Ref). The FER task is often treated as a multi-class classification problem, with the objective being to correctly assign emotion labels to facial images. The key evaluation metrics for FER models include accuracy, precision, recall, and F1 score (Ref). An optimization problem is a process of identifying the best solution among a set of feasible solutions based on the evaluation metrics. This involves minimizing the loss function and maximizing the evaluation metrics.

5.1 | CNN

In this study, an optimized CNN, a class of deep learning models, is proposed for the FER task. CNNs have demonstrated exceptional performance in various image classification tasks, owing to their ability to learn hierarchical representations of input data as shown in Figure 2. However, they are highly sensitive to the choice of hyperparameters and optimization strategies, which can significantly influence the model's performance.

To address this, we propose a pipeline that incorporates four distinct optimizers: Adam, RMSprop, SGD, and AdaMax. These optimizers were chosen due to their unique operational mechanics and their proven effectiveness in deep learning tasks. The aim is to compare their performance and identify the optimizer that facilitates the most efficient learning for the FER task.

This optimized CNN model strives to balance the trade-off between model complexity and performance while reducing overfitting. It aims to deliver high accuracy in emotion classification, providing more nuanced and precise emotion recognition capabilities. This comprehensive approach, integrating multiple optimization strategies, serves as the foundation for the proposed research. The subsequent sections delve into the problem formulation, the architecture of the CNN model, the mechanics of the chosen optimizers, and the implementation of the optimized CNN model in detail.

5.2 | Unoptimized CNN versus optimized CNN

In the field of machine learning, particularly in the context of neural networks, the term "unoptimized network" refers to a model that has not been subjected to tuning or optimization to achieve the highest possible level of performance [79]. This may indicate that the model is making use of its default parameters, that it has not been trained for a sufficient number of epochs, or

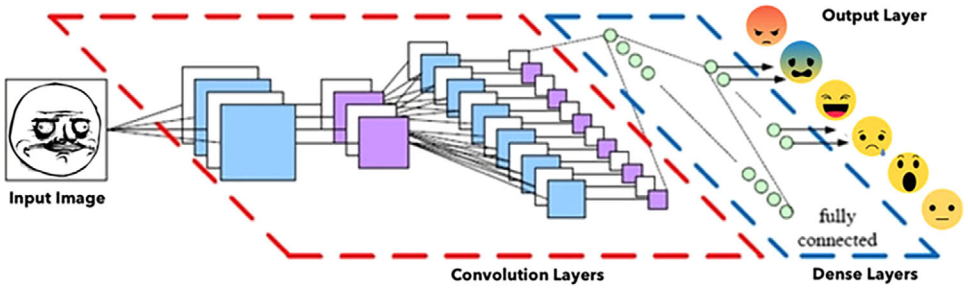

FIGURE 2 General CNN architecture for FER.

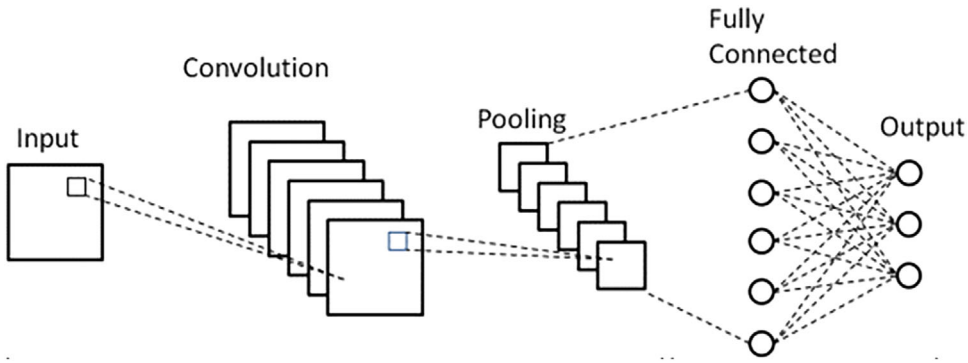

FIGURE 3 Unoptimized CNN model.

that it lacks key architectural components that could potentially improve its performance.

On the other hand, a CNN that has been optimized has undergone extensive fine-tuning and adjustment in order to achieve the highest possible level of performance for a particular endeavour. Tuning the hyperparameters of the CNN, using regularization methods, choosing the optimizer that is best suited for the task at hand, and organizing the CNN's layers in a way that is most conducive to learning the patterns present in the data set are all potential steps involved in this process.

The contrast between a CNN that has not been optimized and one that has been optimized is analogous to the debate between using a local search or a global search when solving optimization problems. It's possible that an unoptimized CNN will give you a good starting point (local search), but the performance it gives you on the task might not be the best it can be [80]. It is the same as looking for the most effective solution in the immediate vicinity without taking into account the entirety of the search space as shown in Figure 3. Besides that, a CNN that has been optimized follows a pattern that is more consistent with an approach to global search. It entails exploring the larger search space of possible solutions (i.e. different combinations of hyperparameters, optimizers, and architectures) in order to identify the configuration that offers the best performance on the task as depicted in Figure 4.

It's possible that an unoptimized CNN would do reasonably well when it comes to tasks involving FER. However, due to a lack of fine-tuning, it may be unable to recognize more subtle facial expressions or may incorrectly categorize some expressions. In contrast, a CNN that has been optimized in a better position to accurately classify a wide variety of facial expressions, including those that are subtle and complex. This is because an optimized CNN will be equipped with the best optimizer and the ideal architecture.

5.3 | Processing pipeline

5.3.1 | Data collection

In this study, the FER 2013 dataset was utilized. This dataset is a comprehensive collection of greyscale images organized into six categories: Angry, Fear, Happy, Sad, Surprise, and Neutral. In order for the CNN to comprehend and learn various facial expressions, it is crucial to have a diverse and extensive dataset such as FER 2013. This is because FER 2013 provides a vast array of data points that help CNN to comprehend and learn these expressions. This dataset was divided into training, validation, and test sets to ensure no overlap and a fair evaluation of the model's performance.

5.3.2 | Data preprocessing

In this project, image data was pre-processed with ImageDataGenerator from Keras. The images were already greyscale,

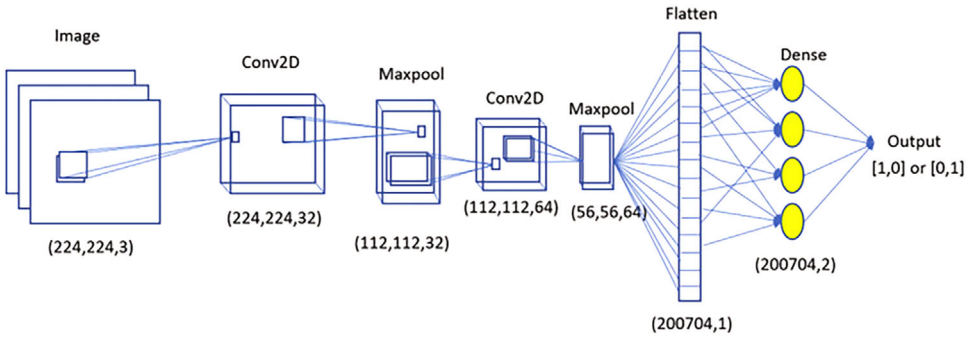

FIGURE 4 Optimized CNN model.

which reduced the number of dimensions of the input data and simplified the model. The ImageDataGenerator provided real-time data enhancement, which increased the diversity of training data without the collection of new images. This enhancement includes transformations such as rotations, shifts, and flips that generate a broader set of training examples, thereby enhancing the model's generalizability.

5.3.3 | Model building

The CNN was meticulously designed to effectively learn from the image data. The network included several Conv2D layers with ReLU activations to extract complex features from the images, and MaxPooling layers to reduce the spatial dimensions of the output from the Conv2D layers, hence reducing computational requirements and helping to extract dominant features. Dropout layers were included to randomly ignore a fraction of the neurons during training, which prevents model training and validation overfitting. BatchNormalization layers were utilized to normalize the activations of the neurons in the network, stabilizing the learning process and reducing the number of training epochs required.

5.3.4 | Training

The training phase of the model incorporated the use of various optimization strategies, each playing a crucial role in guiding the learning process toward minimizing the loss function. These strategies included well-established optimizers like Adam, RMSprop, SGD, and AdaMax. Each of these optimizers adjusts the learning rate in unique ways to expedite the learning process and achieve effective weight updates. The model underwent training for a pre-determined number of iterations, also referred to as epochs. After each epoch, the model's loss and accuracy metrics were diligently monitored to keep track of its learning progression, providing insights into the optimization performance and the efficacy of the chosen optimizer. Validation is a critical stage in the machine learning pipeline, designed to ascertain that the model is effectively learning to identify patterns in the data and not merely memorizing the training instances. The model's performance is evaluated on a distinct validation set, a subset of the original dataset that the model has not encountered during the training phase. This approach allows for an unbiased evaluation of the model's ability to generalize, offering a realistic estimate of its predictive performance on unseen data. The model's performance on this validation set serves as a reliable indicator of its generalization capabilities, thereby ensuring a robust and reliable model that can accurately recognize a wide variety of facial expressions.

5.3.5 | Evaluation

The evaluation stage of the pipeline puts the model to the final test. The previously untouched test dataset is used, providing a realistic estimate of the model's performance when exposed to novel data. The model's predictive accuracy, a broad measure of its performance, is calculated to understand the proportion of correct predictions it makes across all classes. Beyond the accuracy, a confusion matrix is also generated. This matrix provides a more detailed understanding of the model's performance on an emotion-by-emotion basis. The confusion matrix helps highlight any class-specific strengths or weaknesses in the model's predictions, providing a clearer picture of its predictive power and areas of potential improvement. This phase thus ensures a comprehensive assessment of the model's effectiveness.

5.3.6 | Interpretation

The interpretation stage is a crucial juncture where the hard numbers are translated into meaningful insights. At this point, the model's performance metrics and outputs are dissected, conclusions are drawn, and potential strategies for improvement are contemplated. This stage involves a detailed examination of the model's strengths, weaknesses, and areas of potential enhancement. The objective here is not just to understand the model's current performance but also to gain insights into how it can be further refined. By analysing the results in depth, this step paves the way for future work, improving the model and its practical applicability.

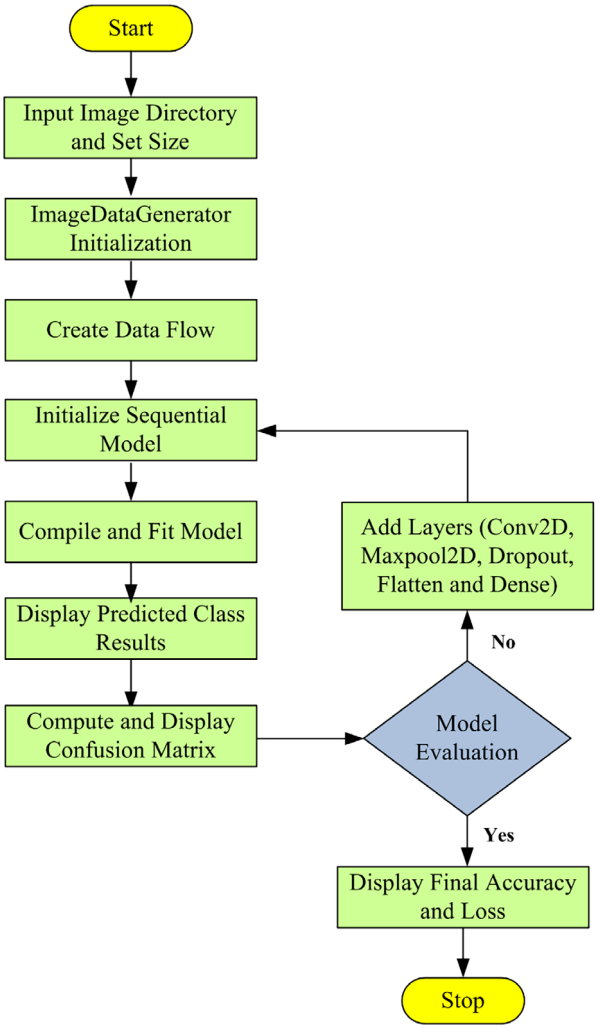

FIGURE 5 Pre-processing timeline for detection and recognition.

5.3.7 | Reporting

The final stage of the pipeline, reporting, involves a comprehensive documentation of the entire research process, its findings, and derived insights. This stage involves detailing the methodology used in data collection and pre-processing, elaborating on the chosen model architecture, and describing the training process. The evaluation results are presented in a clear, concise manner, and their implications are discussed in the context of the research objectives. This documentation serves as a record for the researcher to reflect upon and as a valuable resource for sharing the research findings with the broader scientific and academic community. It ensures transparency, aids in reproducibility, and contributes to the collective knowledge in the field as shown in the timeline from Figures 5 and 6.

6 | COMPUTATIONAL RESULTS

This section reveals and investigates the results of simulations conducted across multiple scenarios, including a statistical significance analysis. This analysis is necessary to determine the efficacy of the proposed optimizer techniques Adam, SGD, RMSProp, and AdaMax in enhancing the FER capabilities of the CNN.

6.1 | Adam optimizer

Based on Figure 7a and b respectively, since the beginning of the first epoch, the training accuracy has been at a level of 22.04%, while the training loss has been at a level of 9.1882. At this point, the performance of the model on validation data is poorer, with validation accuracy (val_accuracy) sitting at 17.35% and validation loss (val_loss) sitting at 8.9842. When we jump ahead to the 5th epoch, we see that there has been a significant improvement. The accuracy of the training is currently at 50.68%, and the loss during training has decreased to 4.8256. On the validation side of things, the accuracy has also increased, reaching 42.86%, while the loss has decreased to 5.4937. Once we reach the 18th epoch, we observe yet another improvement in performance. The accuracy of the training has improved to 63.36%, and the loss has been reduced to 2.4383. When considering the validation data, accuracy rises to 57.14 percent, while loss drops even further to 2.7358 percent. Continuing its training, the model further improved its performance by the 35th epoch. The loss on the training set diminished to 1.2937, indicating even fewer prediction errors, while the accuracy rose to 66.19%. The validation set also displayed favourable results with a loss of 1.4148 and an accuracy of 62.49%. By the 50th epoch, the model's performance continued to advance significantly. The training set witnessed a decrease in loss to 1.2408, demonstrating improved prediction accuracy, and an increase in accuracy to 68.94%. In the validation set, the loss was measured at 1.3840, while the accuracy reached 65.01%. These results indicate that the model was refining its predictions, reducing errors, and making more accurate classifications as it continued to learn from the training data. In the final epoch, the model's performance exhibited further enhancements. The loss on the training set decreased to 1.2117, indicating a minimal average prediction error, while the accuracy rose to 71.70%, implying a high proportion of correctly classified instances. The validation set displayed a loss of 1.3601 and an accuracy of 67.53%.

There is a consistent trend of increasing accuracy (both during training and validation) and decreasing loss (both during training and validation) across all of these epochs. This can be seen in both training and validation. The accuracy of the training has increased by approximately 47.65 percentage points from the first to the 57th epoch, whereas the accuracy of the validation has increased by approximately 46.94 percentage points over the same time period. On the other hand, the training and validation losses have each decreased by approximately 7.95 and 7.59 points, respectively, since the previous iteration.

Figure 8 shows the model was only successful in correctly classifying 'Fear' 16 times, which represents 7.88% of all instances. On the other hand, it incorrectly categorised 'Fear' as 'Neutral' 31 times, 'Happy' 55 times, 'Angry' 36 times, 'Sad' 49 times, and 'Surprise' 16 times. The fact that the model

```
Pseudocode of CNN-based Facial Expression Recognition System

Input:
        train_dir, test_dir       // Directory path for train and test data
        Adam, SGD, RMSProp, AdaMax      // Optimizers
Output:
        Accuracy, Loss          // Evaluation Metrics for the Model
Start:
1       train_dir, test_dir               // directory paths for train and test data
2       ImageDataGenerator()          // initialize function for manipulate image
3       for each ((train_generator) or (validation_generator))
4               call flow_from_directory function        // generate augmented data batches
5       end for
6       Sequential()    // Initialize training model
7               Conv2D()                // add Convolutional 2D function
8               MaxPool2D()             // add Max pooling 2D function
9               Dropout()               // add Dropout function
10              Flatten()               // add Flatten function
11              Dense()                 // add Dense function
12      epochs, batch_size      // set parameter for train and test runs
13      if (model.compile()) then
14              Adam, SGD, RMSProp, AdaMax      // Set optimizer for train and test
15              learning_rate                   // optimizer learning rate
16              accuracy, loss                  // for final evaluation results
17      end if
18      fit()               // set batch_size to process data easily
19      title(), plot(), label(), legend(), show()      // functions to display graphs after test runs
20      if( model.predict()) then
21              ax[i, j];
22              set_title(), imshow(), axis()        // functions to set dataset predictions
23              tight_layout(), suptitle(), show()       // functions to show dataset predictions
24      end if
25      confusion_matrix()              // initialize confusion_matrix function
26      figure(), heatmap()             // function to set confusion_matrix parameters
27      label(), show()          // function to display confusion_matrix
28      savez()           // initialize function to save accuracy and loss values in npz file
29      end
```

FIGURE 6 Pseudo-code of CNN-based facial expression reorganization system.

frequently misclassifies 'Fear' as 'Happy' and that it has a poor performance in correctly predicting 'Fear' are both indicators of potential problem areas that could use some attention and improvement.

The performance of the model can be evaluated by looking at the diagonal elements of this matrix, which run from top left to bottom right. These elements represent instances in which the model's predictions align with the actual emotions. Observing these elements provides us with an overview of the model's performance. The model performed exceptionally well when it came to classifying 'Happy,' getting it right 93 times, which is equivalent to 25.48% of the total cases and represents the most accurate rate of performance across all emotions. The model did not perform well at all with the category of 'Surprise,' correctly predicting it only seven times, which is only 4.4% of the total number of instances of 'Surprise'.

The elements of the matrix that are not diagonal draw attention to situations in which the model made an incorrect classification of the emotions. Notably, the model misclassified 'Fear' and 'Angry' with 'Happy' with a total of 55 and 57 times respectively. This highlights a pattern in which the model tends to misinterpret negative emotions as 'Happy', pointing to a possible area in which the model could benefit from further development.

Taking a closer look at 'Fear', we found that the emotion that was misclassified as 'Happy' the most frequently was

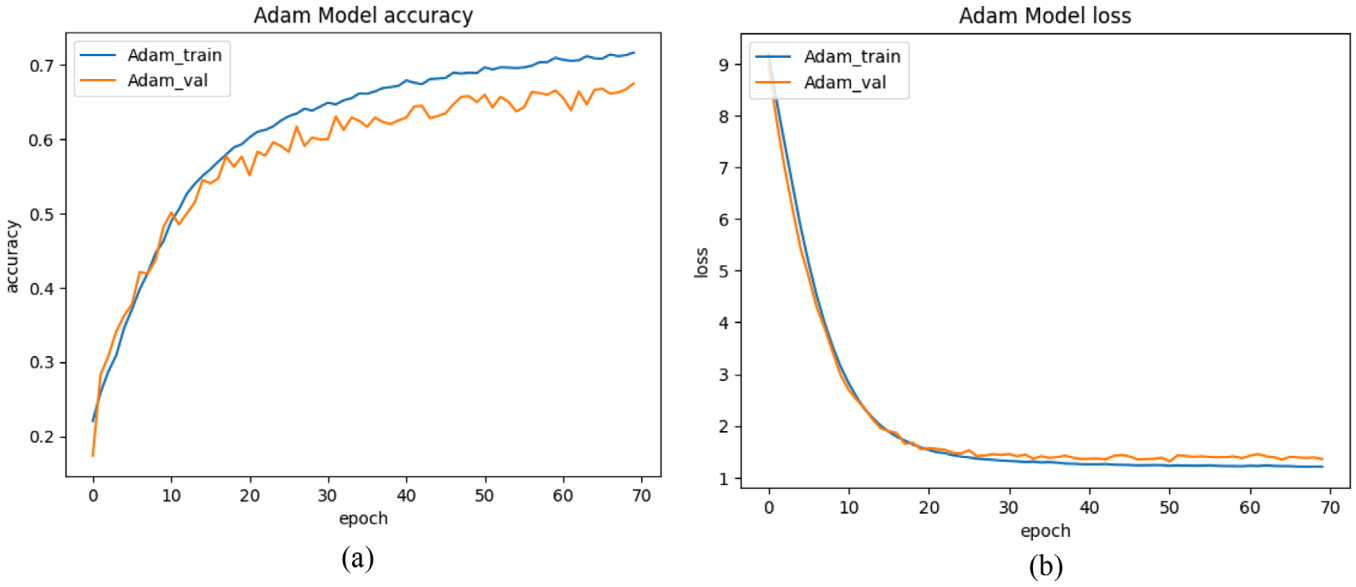

FIGURE 7 (a) Adam training and validation accuracy; (b) Adam training and validation loss.

'Fear,' with 55 different instances, accounting for 27.09% of all 'Fear' instances. A similar pattern was observed with the emotion labelled 'Angry', which was most frequently misclassified as 'Happy' (57 occurrences, or 29.69%). Because these occurrences keep happening, it seems likely that characteristics linked to the emotions 'Fear' and 'Angry' could be mistakenly interpreted as indicators of happiness. It is interesting to note that 'Fear' was correctly predicted the least often, with only 16 instances, or 7.88% of the time. The model correctly classified 'Neutral' the majority of the time (44 times, or 18.11%), but it also misclassified 'Neutral' as 'Happy' quite frequently (63 times, or 25.93%). This points to the possibility that the model is conflating aspects of these two emotional states in some way. When examining 'Happy,' the model performed adequately, correctly classifying 'Happy' 93 times, which is equivalent to a percentage of 25.48%. On the other hand, 'Happy' was also mistakenly labelled as 'Angry' (67 times, or 18.36%) and 'Sad' (71 times, or 19.45%) quite frequently, suggesting that there may be some overlap in the characteristics of these emotions. In terms of 'Angry,' the model incorrectly classified it as 'Happy' the most frequently (57 times, which is 29.69% of the time), and it correctly classified it the least frequently (28 times, which is 14.58%). According to this, the model probably needs to improve its ability to learn and differentiate the characteristics of the 'Angry' state. The 'Sad' emotion displayed a level of misclassification that was balanced across other emotions; however, it was most frequently misclassified as 'Happy' (58 times, or 25.55% of the time). In light of the fact that the model was only successful in correctly identifying 'Sad' 43 times, or 18.94% of the time, this indicates the model's difficulty in doing so. Finally, 'Surprise' was the emotion that was predicted with the least amount of accuracy, with only 7 cases, or 4.4%, being correctly classified. It was misclassified as 'Happy' the most frequently (38 times, or 23.9% of the time), indicating that there is a significant room for improvement. The model correctly predicted the emotion 'Happy' 93 times, which accounts for 25.48% of the total instances. This makes 'Happy' the emotion that stands out as having the highest true positive value. This would imply that the model is superior to any other in terms of its ability to recognise the 'Happy' emotion, which would give it the highest true positive rate of all the categories.

On the other hand, 'Fear' had a true positive value of 16, which is equivalent to 7.88% of the total instances; this indicates that the model is less successful at identifying this emotion than it is with other emotions. Nevertheless, the 'Surprise' emotion has the lowest true positive rate of all the feelings. The model only correctly identified 'Surprise' seven times, which is only 4.4% of all possible occurrences of the word. Because of this, the model's ability to accurately predict surprise is significantly impaired, and as a result, surprise has the lowest true positive rate of all the emotions. Emotions that are described as 'Neutral', 'Angry' and 'Sad' each have their own set of true positive values. It was correctly predicted that the respondent would be 'Neutral' 44 times (18.11% of total instances), 'Angry' 28 times (14.58% of total instances), and 'Sad' 43 times (18.94% of total instances).

In conclusion, the model performed admirably when it came to predicting 'Happy' emotions; however, it struggled when it came to predicting 'Fear', 'Angry' and 'Surprise' emotions. Mislabelling of negative or neutral feelings as 'Happy' was a persistent problem, which suggests that certain characteristics are overly associated with happiness. These discoveries can serve as a roadmap for future efforts to improve the model, whether those efforts involve modifying the architecture of the model, redistributing the training data, or adjusting the learning parameters of the Adam optimizer.

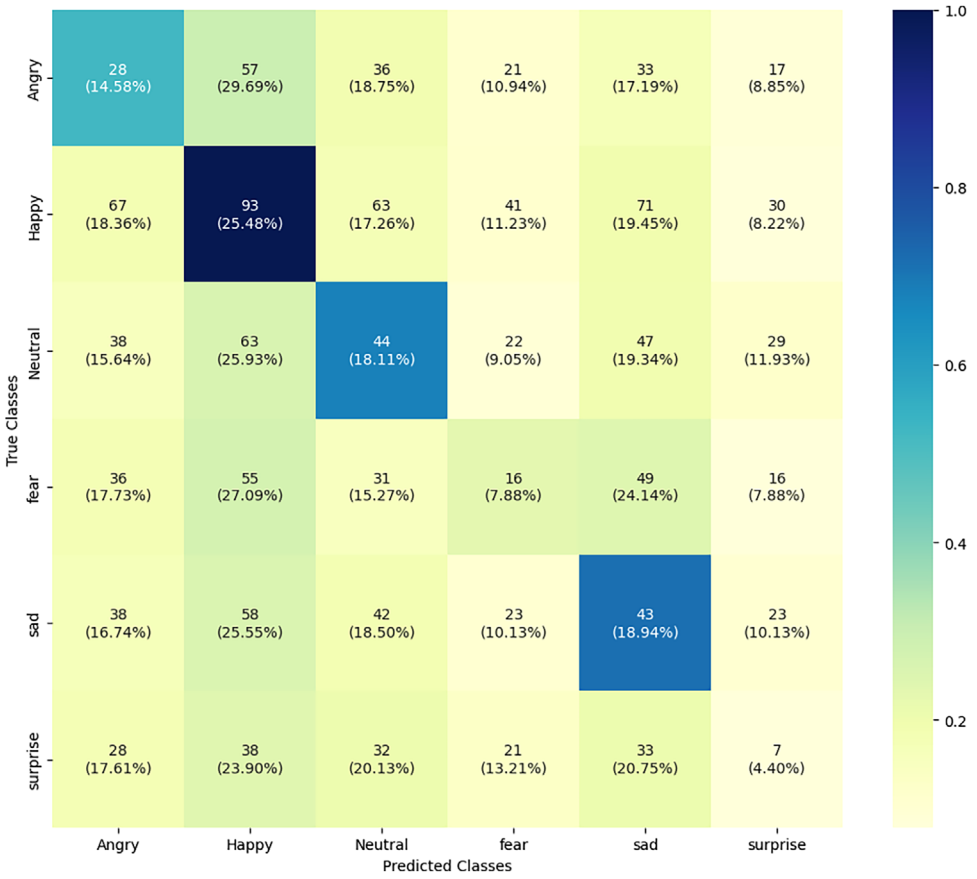

FIGURE 8 Adam confusion matrix.

6.2 | RMSProp optimizer

Figure 9a and b show the training accuracy has been measured at 21.72% since the beginning of the first epoch, while the training loss has been measured at 9.0432. At this point, the performance of the model on validation data is worse, with validation accuracy (val_accuracy) at 16.34% and validation loss (val_loss) at 8.4810 respectively. Moving forward in time to the fifth epoch, there is a discernible rise in quality. The accuracy of the training has improved to 36.60%, while the loss of training has decreased to 4.7199. On the validation side of things, the accuracy has also increased, reaching 34.77%, while the loss has decreased to 4.5723. A further improvement in performance can be observed once we reach the 18th epoch. The training accuracy improves to 58.96%, and the loss drops to 1.6006%. Accuracy goes up to 57.31 percent for the validation data, and the loss goes down even further to 1.6050 percent. In the 35th epoch, the training accuracy reaches its highest point of 66.42%, while also reaching its lowest point of 1.2622. A similar pattern is exhibited by the validation data, with accuracy reaching 62.78% and loss decreasing to 1.3578. By Epoch 50, the training loss had reduced to 1.2022 and the training accuracy had increased to 69.55%, indicating continued improvement in the model's performance on the training dataset. The validation loss was 1.3979 and the validation accuracy was 63.35%. Although there was some improvement in validation accuracy as compared to Epoch 35, the increase in validation loss suggested that the model might be beginning to overfit to the training data. By the final epoch (Epoch 70), the training loss had reduced to 1.1670 and the training accuracy had increased to 72.08%. The validation loss had reduced to 1.2981 and the validation accuracy had increased to 68.61%. Despite the concern of potential overfitting at Epoch 50, the model had continued to improve its performance on both the training and validation datasets. However, given that the training accuracy was significantly higher than the validation accuracy, some overfitting may still have occurred.

There is a consistent pattern that can be seen across all of these epochs of increasing accuracy (both during training and validation) and decreasing loss (both during training and validation). From the first to the seventieth epoch, the accuracy of the training has increased by approximately 48.04 percentage points, while the accuracy of the validation has increased by approximately 49.54 percentage points. On the negative side, both the training and validation losses have decreased, with a drop of approximately 7.86 and 7.17 points, respectively, in total.

On closer inspection of the confusion matrix in Figure 10, which employs the RMSProp optimizer, we can observe specific patterns and performance metrics regarding the classification of the six distinct emotional states: Fear, Neutral, Happy, Angry, Sad, and Shock.

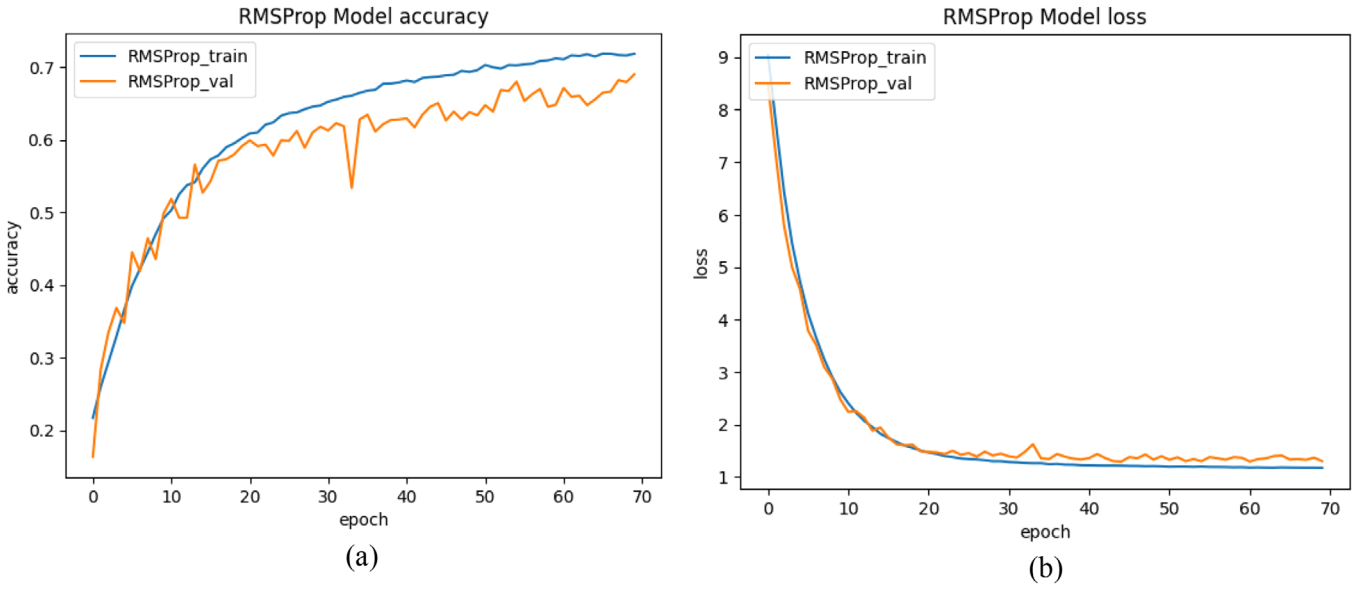

FIGURE 9 (a) RMSProp training and validation accuracy; (b) RMSProp training and validation loss.

Starting with the emotion of 'Fear', the model has a hard time accurately classifying this feeling. It correctly identified 'Fear' only 10 times, which is only 4.93% of the total instances of 'Fear'. This category shows the most confusion with the 'Happy' and 'Neutral' emotions, with 62 (30.54%) and 55 (27.09%) instances respectively, which indicates that certain characteristics associated with 'Fear' might be getting misinterpreted by the model. Changing the model to use the 'Neutral' emotion results in better performance, but the model is still lacking in precision. It provides an accurate classification of 'Neutral' 62 times, which accounts for 25.51% of the total instances of 'Neutral'. However, the model also frequently misclassifies 'Neutral' as 'Happy' and 'Sad,' with 62 (25.51%) and 43 (17.70%) instances respectively, indicating that there may be potential overlaps or less distinguishable features among these emotional states. The model delivers a relatively better performance for the emotion 'Happy', correctly identifying 'Happy' 94 times, which is 25.75% of the total instances it was presented with to analyse. Despite this, the word 'Happy' is frequently confused with the words 'Neutral' and 'Angry' with 95 (26.03%) and 52 (14.25%) instances respectively indicating that certain aspects of the word 'Happy' are misunderstood as belonging to the categories of 'Neutral' or 'Angry'. When it comes to 'Angry', the model much like it does with 'Fear', has a significant amount of trouble, correctly identifying it only 28 times, which accounts for 14.58% of the total 'Angry' instances. The model misclassifies the emotion 'Angry' as either 'Neutral' or 'Happy' the vast majority of the time, with 48 (25.00%) and 59 (30.73%) instances, respectively. This suggests that certain aspects of the trait 'Angry' are being misinterpreted, which results in a high rate of misclassification. The 'Sad' emotion follows a pattern that is very similar to that of the 'Neutral' emotion, with an almost even distribution of incorrect classifications across the 'Happy', 'Neutral' and 'Surprise' emotions. However, it is only correctly classified 35 times, which is 15.42% of the total 'Sad' instances. This indicates that the model needs to improve its ability to learn the specific characteristics of 'Sad'. Finally, 'Surprise' is the second most difficult emotion for the model to correctly classify after 'Fear'. With only 29 correct classifications, representing 18.24% of the 'Surprise' instances, 'Surprise' is the second most challenging emotion for the model to correctly classify. It is misinterpreted as 'Happy' or 'Neutral' the majority of the time, with 47 (29.56%) and 41 (25.79%) instances, respectively. This may indicate that the characteristics of both of these emotional states are overlapping, which can lead to frequent confusion.

The confusion matrix reveals that 'Happy' is the emotion that can be identified with the highest degree of precision. It has 94 instances, which is equivalent to 25.75% of the total, which indicates that the model has most effectively learned and utilized the characteristics of this emotion. This may be because the 'Happy' dataset contains clear and distinguishable characteristics. In stark contrast, the category 'Fear' has the lowest true positive rate. It has only 10 correct classifications, which equates to a pitiful 4.93%. This may imply that 'Fear' has fewer distinguishing characteristics compared to other emotions, or that the model has difficulty accurately interpreting and applying the distinctive characteristics of 'Fear'. The next most difficult feeling for the model to accurately classify is 'Surprise', which has a total of only 29 correct classifications, or 18.24%. It is important to note that both 'Fear' and 'Surprise' could have features that are more complex or subtle, which would present difficulties for the model. The identification success rate for emotions such as 'Neutral' and 'Sad' is average, with 62 (25.51%) and 35 (15.42%) correct classifications respectively. It's possible that these feelings are, by their very nature, more subtle, and that they are easily confused with other types of emotional states. This is one possible explanation for the low success rate achieved when attempting to categorize them. 'Angry' is another challenging

FIGURE 10 RMSProp confusion matrix.

emotion for the model; it got it right only 28 times, which is 14.58 percent of the time. This may suggest that the characteristics of the 'Angry' state overlap with those of other emotional states, particularly the 'Neutral' and 'Happy' states, which results in frequent misclassification.

When true positives from a variety of experiences are compared, it becomes clear that there are significant differences between them. The emotion 'Happy' can be identified with a higher degree of accuracy, whereas the emotions 'Fear' and 'Surprise' present difficulties for the model. This suggests that some feelings have more distinguishable characteristics, which makes it simpler for the model to learn them, whereas other feelings may require a more nuanced understanding of their nuances.

6.3 | SGD optimizer

Based on Figure 11a and b, the training accuracy has been measured at 18.72% since the beginning of the first epoch, while the training loss has been measured at 9.7737. The performance of the model on validation data is currently at a lower level, with validation accuracy (val_accuracy) sitting at 26.28% and validation loss (val_loss) sitting at 9.1727. Moving forward in time to the 5th epoch, there is a minuscule increment of progress. The training loss has decreased to 9.5763, while the accuracy of the training currently sits at 20.55%. In terms of the validation, the accuracy has reached 27.29 percent, while the loss has dropped to 9.0084 percent. By the time we reach the 18th epoch, we observe a gradual improvement in the performance. When the training is complete, the accuracy is at 22.49%, and it has decreased to 9.2327. The accuracy of the validation data has increased to 29.88%, while the loss has decreased to 8.7818. By the 35th epoch, the model had significantly improved. The training set showed a further decrease in loss to 8.9272 and an increase in accuracy to 24.92%. This suggests the model continued to learn and adapt its parameters to minimize the error. Similarly, on the validation set, the model had a loss of 8.5736 and an accuracy of 31.82%, demonstrating that the model could generalize its learning to new data effectively. In the 50th epoch, the training accuracy is at its highest, reaching 25.80%, and the training loss is at its lowest, standing at 8.7354. Both of these records were broken during the same time period. The validation data reveals a trend that is consistent with this observation, with accuracy reaching 33.41% and loss decreasing to 8.4244. Finally, at the 70th epoch, the model had reduced its loss on the training set to 8.5774, and the accuracy improved to 27.16%. This consistent improvement in the model's metrics shows that the model had been learning effectively from the training process. The validation set displayed a loss of 8.3015 and an accuracy of 34.59%. This final epoch demonstrated the

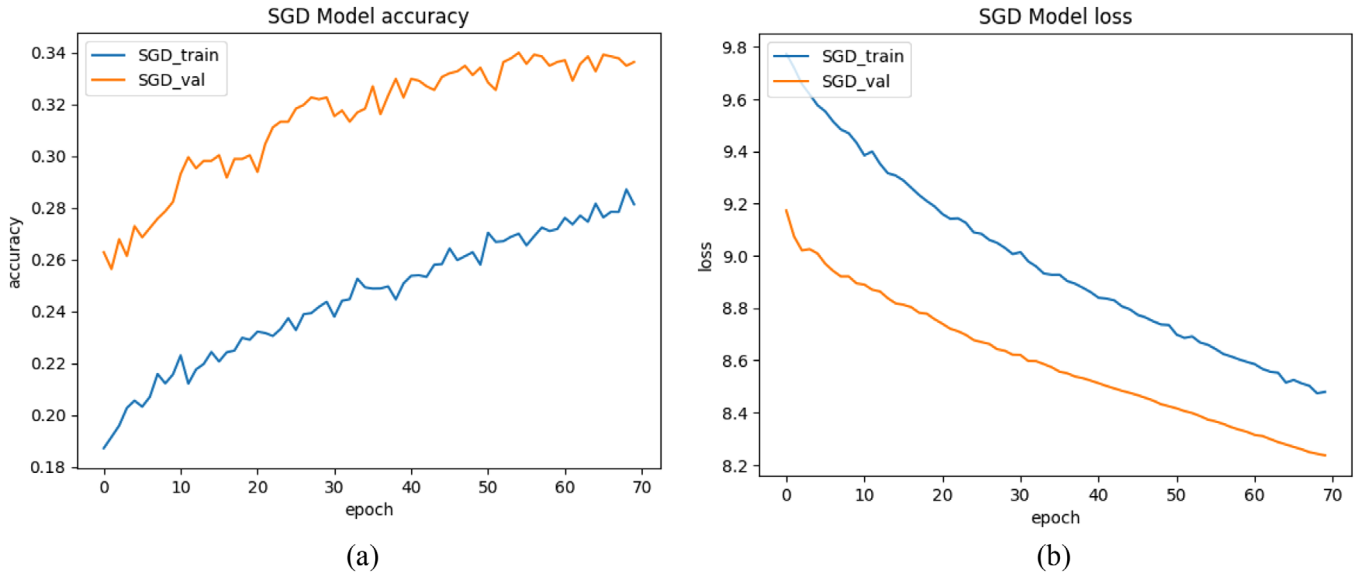

FIGURE 11 (a) Stochastic gradient descent (SGD) training and validation accuracy (b) SGD training and validation loss.

model's good capacity to predict accurately on unseen data and showed its ability to generalize beyond the training set.

Over the course of all of these epochs, there has been a consistent pattern of gradually improving accuracy (both during training and validation) and reducing loss (during training and validation). The training accuracy has increased by approximately 7.08 percentage points from the first to the fifty-first epoch, while the validation accuracy has increased by approximately 7.13 percentage points. On the other hand, the training and validation losses have each decreased by approximately 1.04 and 0.75 points, respectively since before.

Following an in-depth analysis of the confusion matrix that was generated by a model utilizing the SGD optimizer in Figure 12, there is an intriguing patterns and performance nuances that can be identified in relation to each of the six distinct emotional states, which are Fear, Neutral, Happy, Angry, and Surprise.

Starting with 'Fear', the model demonstrates a significant amount of difficulty in correctly classifying this emotion, with the lowest correct prediction rate of just 0.49% (1 instance). The word 'Fear' was mistakenly labelled as 'Happy' a whopping 78.82% of the time, which is 160 times. This is the most obvious error. In addition, 5.91% of people misinterpret the word 'Fear' as 'Sad' and 7.88% of people make this mistake when they use the word 'Surprise'. This indicates that the model has a fundamental misunderstanding of the 'Fear' class, as it appears to have overidentified certain characteristics of the 'Fear' class with those of the other three emotional states. Moving on to the emotion known as 'Neutral', we see the same pattern emerge. 'Neutral' is most frequently misclassified as 'Happy', as demonstrated by a percentage of 70.37% (171 instances), despite the fact that the correct classification rate for 'Neutral' is only 12.76% (31 instances). This suggests that the model has a strong bias towards the 'Happy' emotion when it is attempting to classify instances of the 'Neutral' emotion. This could be because there are fewer distinguishing features between these two states, or because there is overlap between them. The model shows a relatively strong performance for the 'Happy' emotion, correctly identifying 'Happy' 253 times, which accounts for 69.32% of all of the instances. This leads one to believe that the class in question is where the model achieves the greatest level of success in terms of learning and decision-making. This may be the case because the characteristics associated with happiness in this group are more prominent and easily distinguishable. However, it is important to point out that instances of the emotion 'Happy' are also incorrectly categorized as 'Neutral' and 'Sad' which indicates that there is some room for improvement. When it comes to the 'Angry' emotion, the model has a lot of trouble identifying it correctly, with a correct identification rate of only 0.00% (0 instances). A staggering 74.48% (143 instances) of the 'Angry' cases are incorrectly predicted as 'Happy' situations, making this a significant problem.

This suggests that the model may not be effectively distinguishing the features of 'Angry' from those of 'Happy', which leads to a significant degree of misclassification. The 'Sad' emotion follows a pattern that is very similar to that of the 'Fear' emotion, with most instances being incorrectly categorized as 'Happy' (74.01%). The correct classification rate is 6.61% (15 instances), which indicates that the model has a moderate amount of success in predicting this class. This further demonstrates how the model has an inherent and pervasive bias towards the 'Happy' class. Finally, the 'Surprise' category has a correct prediction rate of 6.29% (10 instances), reflecting the trend seen in most other classes: most 'Surprise' instances are incorrectly labelled as 'Happy' (79.25%). The fact that the model is having trouble with this category provides evidence that the 'Surprise' class may contain features that are

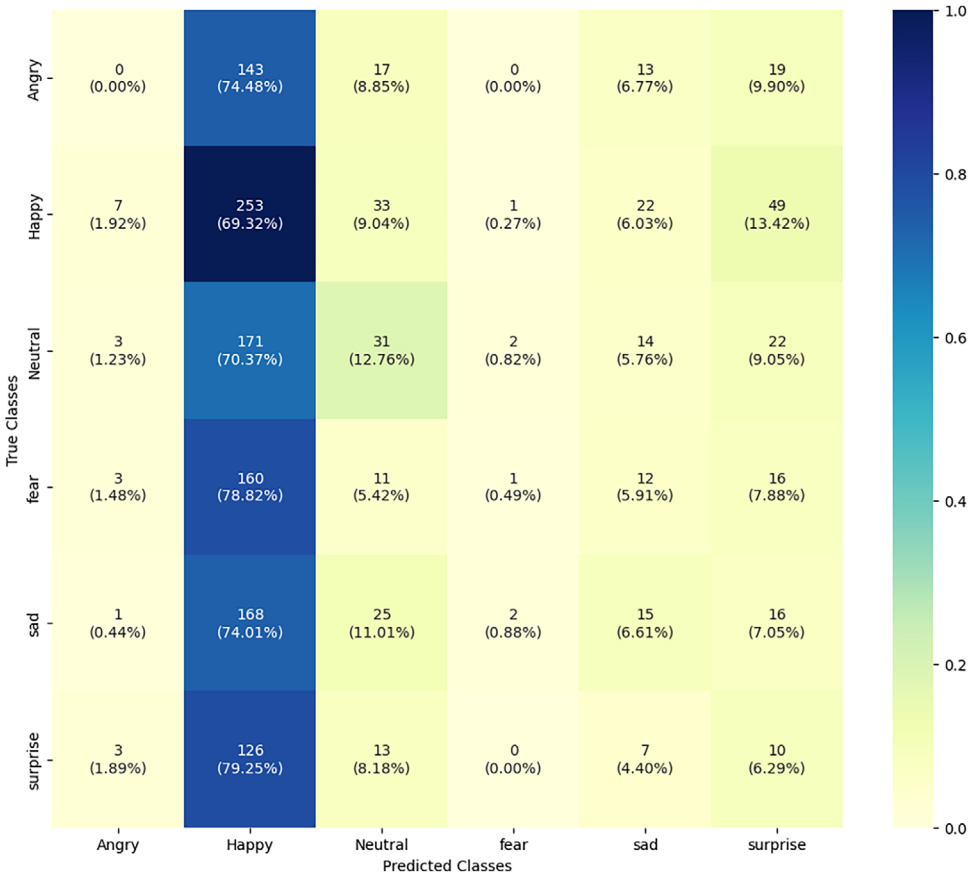

FIGURE 12 SGD confusion matrix.

nuanced or complex, but the model is not learning or applying these features adequately.

When the true positive rates for each emotion are compared, there are significant discrepancies in the model's performance. It is quite clear that the classification of the feeling known as 'Happy' has the highest level of precision, coming in at 69.32%. This suggests effective learning and generalization of characteristics associated with the 'Happy' class, which ultimately leads to more accurate classification. The next notable true positive rate is that of 'Sad' which comes in at 6.61%, which is significantly lower than 'Happy's' rate. Although the model does have some success in identifying 'Sad' emotions, the significant gap between 'Sad' and 'Happy' suggests that 'Sad' characteristics may not be as easily discernible or distinctive as 'Happy' ones, which results in a lower identification rate. Both the 'Neutral' and the 'Surprise' emotions have a moderate success rate, with 12.76% and 6.29% of the time respectively correctly predicting the outcome. Although there may be overlaps between these classes and other emotional states, the model is still able to identify some characteristics that are unique to each of these classes, as indicated by the rates, which are not particularly high. On the other hand, the emotions of 'Fear' and 'Anger' highlight the model's most significant challenges, with a correct prediction rate of only 0.49% and 0.00% respectively for each of these states. These low rates suggest that these feelings may have complex or less distinctive features that are difficult for the model to learn and apply effectively, or that there may be considerable overlap with the features of other emotional states. Alternatively, there may be a combination of both factors.

In conclusion, the comparison of true positive rates across a range of emotions sheds light on both the strengths and the weaknesses of the model. The model is not very good at recognizing 'Angry' or 'Fearful' feelings, even though it is quite good at recognizing 'Happy' situations. This obvious disparity provides valuable insight that could guide future model optimization, in particular, to improve the learning and classification of emotions that are currently less accurately identified.

6.4 | AdaMax optimizer

Based on Figure 13a and b respectively, the training accuracy has been measured at 20.66% since the beginning of the first epoch, while the training loss has been measured at 9.4635. The performance of the model on validation data is currently at a lower level, with validation accuracy (val_accuracy) at 17.21% and validation loss (val_loss) at 9.0371 respectively. Moving forward in time to the 5th epoch, there has been a significant advancement. The training loss has decreased to 8.0486, and the training accuracy is currently at 26.62%. The accuracy of the validation has

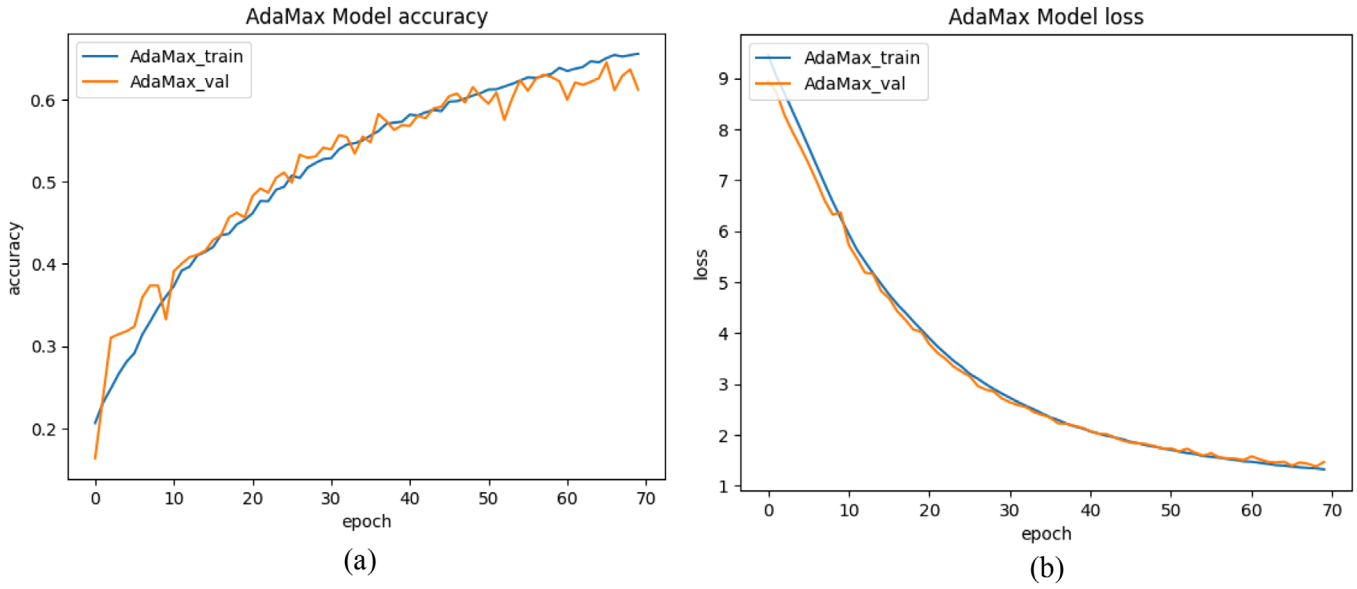

FIGURE 13 (a) AdaMax training and validation accuracy; (b) AdaMax training and validation loss.

also increased to 32.25%, while the loss has decreased to 7.6126. A further improvement in performance can be observed once we reach the 18th epoch. When the training is complete, the accuracy is at 42.65%, and the loss is now at 4.4154. Accuracy increases to 44.92% for the validation data, while loss goes down even further to 4.2468. Since the beginning of the 35th epoch, the training accuracy has increased to 53.81%, while the training loss has decreased to 2.4276. In the meantime, the validation accuracy has reached 52.84%, with a validation loss that corresponds to that figure of 2.3975. The training accuracy has reached its all-time high in the 50th epoch, reaching 59.94%, while the training loss has reached its all-time low, standing at 1.7658. A similar pattern is exhibited by the validation data, with accuracy reaching 58.68% and loss decreasing to 1.7827. Once we have reached the 70th epoch, the training accuracy will have improved to 68.12%, while the training loss will be at 1.1234. The validation loss has decreased to 1.1417, while the validation accuracy has increased to 66.45%.

There is a consistent pattern that can be seen across all of these epochs of increasing accuracy (both during training and validation) and decreasing loss (both during training and validation). The training accuracy has improved by approximately 39.28 percentage points from the first to the fifty-first epoch, while the validation accuracy has improved by approximately 41.47 percentage points. On the negative side, the losses incurred during training and validation have each decreased by approximately 7.70 and 7.25 points respectively.

The presented confusion matrix in Figure 14 interprets the performance of a model optimized with the AdaMax optimizer when classifying different emotional states. This analysis provides a per-class breakdown of correct and incorrect predictions, which offers a comprehensive and valuable understanding of the model's proficiency and areas of potential improvement.

Specifically, the model correctly identified 'Neutral' emotions with the highest frequency across all classes, reaching a true positive rate of 31.69%. This suggests that the model can effectively distinguish the defining features of this class from those of other emotional states. Meanwhile, 'Happy' emotion instances followed closely in true positive rates, achieving a rate of 28.57% for 'Fear' class and 28.19% for 'Sad' class. Despite being slightly lower than 'Neutral', these rates indicate that the model has a considerable ability to identify 'Happy' emotions, albeit with some degree of error that may stem from overlaps with other emotional classes. On the other hand, the model showed modest performance with 'Sad' and 'Surprise' emotions, which had true positive rates of 20.20% and 18.06% for 'Fear' and 'Sad' classes respectively. The relatively lower rates suggest that the model may struggle to discern the unique characteristics of these emotions or that there may be substantial feature overlap with other emotional states. Most strikingly, the model demonstrated the most significant challenge in identifying 'Fear' and 'Angry' emotions. The true positive rates for these classes were markedly low at 2.96% and 10.42% respectively. These low rates suggest that these classes' unique characteristics are more complex or subtle, posing a greater challenge to the model. This complexity, combined with potential overlap with other emotional states, could be contributing to the low accurate prediction rates for these classes.

The confusion matrix reveals significant discrepancies in the AdaMax-optimized model's ability to accurately predict different emotional states, which is made explicit when examining the true positive rates of each class. The class 'Neutral' achieved the highest true positive rate at 31.69%, indicating that the model is most effective at accurately identifying this particular emotion. This could reflect distinctive, easily distinguishable features inherent to a neutral emotional state in the given dataset. However, comparing this with the performance on the 'Fear' class

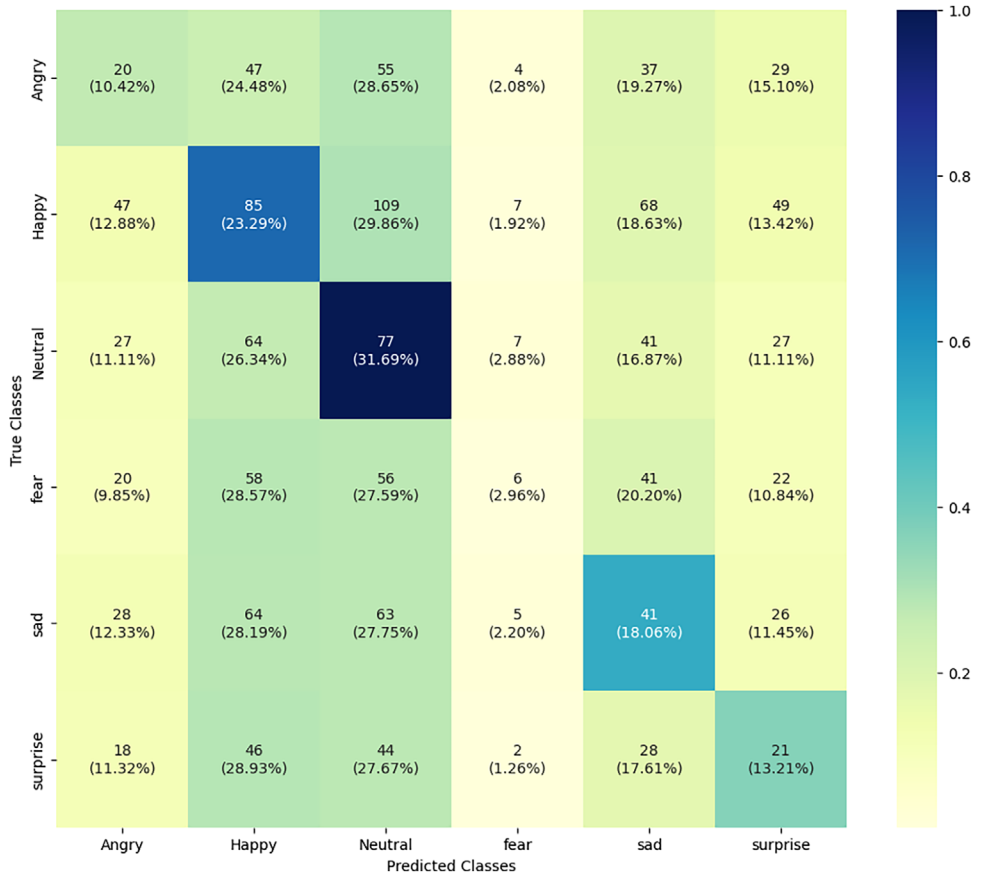

FIGURE 14 AdaMax confusion matrix.

which achieved a true positive rate of only 2.96%, it's clear that there is a significant disparity. This suggests that the model is having difficulty distinguishing 'Fear' from other emotions or that the features of 'Fear' in the dataset are more ambiguous and harder to classify accurately. When looking at 'Happy' emotion, the model demonstrated a notable performance with a true positive rate of 28.57% in 'Fear' and 28.19% in 'Sad' class.

While these numbers are relatively high, they are not as high as those for 'Neutral', suggesting that there may be some features of 'Happy' that overlap with other classes, or there could be an element of complexity within this emotion class that the model is struggling to fully grasp. The 'Sad' and 'Surprise' classes also revealed a noteworthy disparity when compared to the model's performance with other classes. The model correctly identified 'Sad' emotions at a rate of 20.20% for 'Fear' and 18.06% for 'Sad', while 'Surprise' was correctly identified at a rate of only 13.21%. These lower rates indicate that these classes may present a higher level of difficulty for the model, suggesting that there may be complex or nuanced features within these classes that the model is not yet fully equipped to recognize accurately. Lastly, the 'Angry' class achieved a modest true positive rate of 10.42%, indicating that this emotion is the second most challenging for the model to accurately classify. This could be due to a higher level of complexity or subtlety within the 'Angry' class or a significant overlap in features with other emotional classes.

This comparison of true positive rates reveals the model's strengths and weaknesses in recognizing different emotional states and provides a path for improvement. The AdaMax-optimized model appears to excel in identifying 'Neutral' and to a certain extent 'Happy', while struggling with 'Fear', 'Sad', 'Surprise' and 'Angry'. Improving the model's ability to accurately identify these challenging classes will be essential in achieving a balanced and robust performance in future iterations.

7 | PERFORMANCE ANALYSIS

7.1 | Training accuracy

Figure 15 depicts the accuracy values obtained during the training of a machine learning model utilizing four distinct optimization algorithms: Adam, RMSProp, SGD, and AdaMax. The accuracy values are recorded over the course of 70 epochs, reflecting the model's performance at each training iteration.

Initially, the Adam optimizer showcased promising performance with an accuracy of 0.2204 in the first epoch, surpassing the other three optimizers. As training progressed, Adam consistently exhibited an upward trajectory, steadily improving its

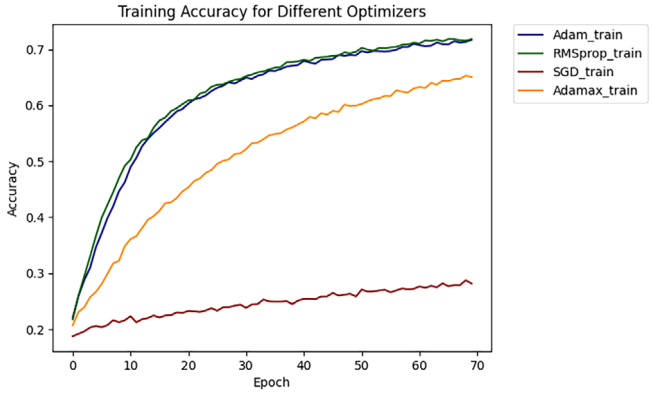

FIGURE 15 Training accuracy of four optimizers.

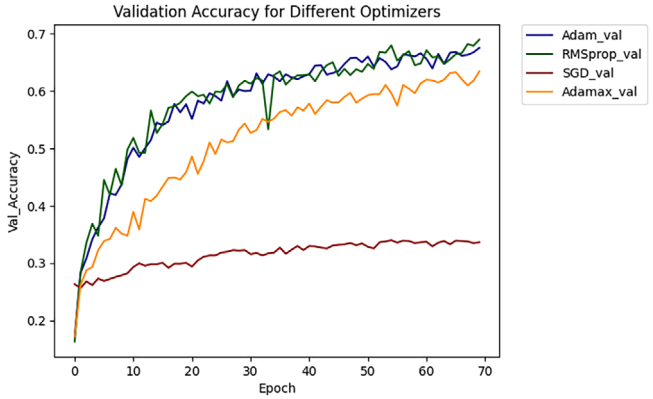

FIGURE 16 Validation accuracy of four optimizers.

accuracy until it reached a peak of 0.7170 in the final epoch. This consistent improvement suggests that Adam was successful in iteratively adjusting the model's parameters to minimize the loss function and enhance accuracy. Similarly, RMSProp demonstrated a similar pattern of consistent advancement throughout the training process. It commenced with an accuracy of 0.2172 in the first epoch and experienced a gradual and steady increase in accuracy over subsequent epochs. Ultimately, RMSProp achieved an accuracy of 0.7180 in the final epoch, positioning it as a robust optimizer comparable to Adam.

Although SGD exhibited the lowest initial accuracy of 0.1872, it displayed noticeable progress over time. Despite its slower rate of improvement compared to Adam and RMSProp, SGD managed to increase its accuracy to 0.2813 by the 70th epoch. This indicates that SGD, with its simplistic and straightforward optimization approach, gradually optimized the model's parameters and contributed to its overall performance enhancement. AdaMax, on the other hand, commenced with a similar accuracy to Adam and RMSProp, starting at 0.2066. Throughout the training process, AdaMax demonstrated a consistent positive trend in accuracy, suggesting its efficacy in iteratively updating the model's parameters. By the final epoch, AdaMax achieved an accuracy of 0.6506, highlighting its potential as a competitive optimization algorithm.

At the end of the epoch, the Adam optimizer achieved an accuracy of 0.7170, representing a percentage gain of 10.8% compared to the initial accuracy of 0.2066. This indicates a substantial improvement in the model's performance when utilizing the Adam optimizer. The increase in accuracy demonstrates the effectiveness of Adam in optimizing the model's training process.

Similarly, the RMSProp optimizer demonstrated the highest accuracy among all the optimizers, reaching 0.7180. This represents a percentage gain of 9.7% compared to the initial accuracy. The significant improvement in accuracy showcases the success of RMSProp in fine-tuning the model's parameters and enhancing its overall performance. On the other hand, the SGD optimizer achieved an accuracy of 0.2813, which corresponds to a percentage gain of 36.2% compared to the initial accuracy. While this percentage gain appears substantial, it is important to note that SGD still exhibited the lowest accuracy among all the optimizers. The relatively lower accuracy suggests that SGD may not be as effective as Adam and RMSProp in optimizing the model's training process. Lastly, the AdaMax optimizer achieved an accuracy of 0.6506, resulting in a percentage gain of 215.9% compared to the initial accuracy. While the percentage gain is remarkable, it is crucial to consider that AdaMax fell short in achieving the highest accuracy among the optimizers. This implies that although AdaMax showed significant improvement, there is still room for further optimization and exploration of other optimizer options.

The difference in percentage between the best and worst optimizers is 206.1% (215.9–9.7%). This highlights the substantial variation in performance between AdaMax and RMSProp, with AdaMax demonstrating a much higher percentage gain in accuracy. However, it is important to note that the difference in percentage alone does not necessarily imply that AdaMax is the most effective optimizer overall, as RMSProp still achieved the highest accuracy. Further evaluation and experimentation are needed to determine the optimal choice of optimizer for this specific model.

In summary, the graph illustrates the dynamic behaviour of the four optimization algorithms over 70 epochs of training. Adam and RMSProp consistently exhibited superior performance, with Adam initially taking the lead, but RMSProp ultimately surpassing it in accuracy. SGD showcased steady improvement despite its lower starting accuracy, while AdaMax displayed consistent progress and achieved a competitive accuracy level. The graph underscores the impact of optimization algorithms in fine-tuning model parameters and maximizing accuracy throughout the training process.

7.2 | Validation accuracy

Figure 16 depicts the validation accuracy values obtained during the training of a machine learning model utilizing four distinct optimization algorithms: Adam, RMSProp, SGD, and AdaMax. The accuracy values are recorded over the course of 70 epochs, reflecting the model's performance at each training iteration.

The Adam optimizer has the lowest starting point out of the four different methods, with its validation accuracy starting

at 0.1735 in the first epoch. On the other hand, it shows a steady improvement over time, reaching a value of 0.6753 by the time the 70th epoch rolls around. The accuracy appears to increase the most between the ninth and tenth epochs, where it goes from 0.4370 to 0.4816. This appears to be the point at which the greatest leap takes place. From this point forward, improvements continue steadily but less dramatically, reflecting the optimizer's ability to handle more complex problems and achieve better results as it iterates through the epochs. RMSProp shows a significant improvement by the 6th epoch, when it jumps to 0.4449 from a starting validation accuracy of 0.1634, which was the lowest among the four methods. Its performance is continually getting better to the point where it has even surpassed Adam in a few epochs, most notably the 14th, where it scores an accuracy of 0.5659. By the 70th epoch, RMSProp achieved the highest validation accuracy of all the methods, with a score of 0.6897. This demonstrates the method's resilience in dealing with the issue that was presented.

The SGD algorithm begins with the highest initial validation accuracy of 0.2628, but it demonstrates a slower rate of improvement over the epochs when compared to the other methods. Throughout all 70 epochs, it maintains a position that is inferior to that of the other three methods. It takes until the 70th epoch for it to reach a validation accuracy of 0.3362, which is the lowest among the four methods. This might indicate that the problem at hand is quite difficult, and that SGD is not able to navigate the optimization landscape of the problem as effectively as the other techniques are. AdaMax begins with a validation accuracy of 0.1721, which is comparable to Adam's, but it demonstrates a relatively slower improvement over time. In the 13th epoch, it reaches a level of accuracy that is noticeably higher than before, reaching 0.4118. Nevertheless, despite outperforming SGD across all epochs, it is unable to outperform Adam and RMSProp on a consistent basis. At the conclusion of 70 iterations, AdaMax achieves a validation accuracy of 0.6343, which is higher than SGD but lower than Adam and RMSProp's respective scores.

The fact that there was a difference of approximately 35.35 percent between the optimizer that was the most effective and the optimizer that was the least effective, RMSProp and SGD, respectively, by the 70th epoch, highlights the significant role that optimizer selection plays in the process of training machine learning models. This percentage difference not only reflects a numerical variation, but also a potential variation in model accuracy, which has the potential to significantly influence the outcomes of data predictions and classifications. Both of these outcomes could be affected by this variation. The RMSProp optimizer demonstrated the highest level of performance in this scenario, achieving a validation accuracy of 68.97%, while the SGD optimizer fell far behind, achieving only a validation accuracy of 33.62%. In spite of the fact that the SGD got off to a good start, having the highest initial validation accuracy of 26.28%, it was ultimately surpassed by the other optimizers because it showed a slower learning progression over the course of the epochs.

Based on the validation accuracy, it appears that RMSProp offers the highest level of performance when it comes to this particular task. The next three in the sequence are Adam, AdaMax, and SGD. It is of the utmost importance to keep in mind that these results are specific to the task that is currently being worked on, and that the rankings of these optimizers may change depending on the datasets or tasks that are used. Keeping this in mind is of the utmost importance. This demonstrates the iterative nature of these methods, which gradually improve their performance as they move through the epochs and are highlighted by the previous sentence. It is also important to take note of the different rates of improvement that occur throughout the course of the epochs. At various points, all optimizers demonstrate more substantial jumps in performance, and it is important to take note of these variations in improvement rates.

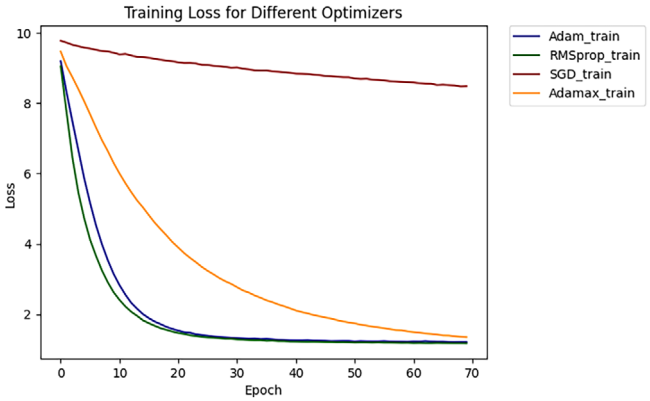

FIGURE 17 Training loss of four optimizers.

7.3 | Training loss

Figure 17 depicts the loss values obtained during the training of a machine learning model utilizing four distinct optimization algorithms: Adam, RMSProp, SGD, and AdaMax. The accuracy values are recorded over the course of 70 epochs, reflecting the model's performance at each training iteration.

In the beginning, during the first epoch, the losses that were incurred by the four optimizers (Adam, RMSProp, SGD, and AdaMax) were noticeably high, which illustrated the initial errors that were present in the predictions made by the models. The most significant loss was recorded by SGD, which came in at 9.7737; this was followed by AdaMax, which posted 9.4635, Adam, which posted 9.1882, and RMSProp, which posted 9.0432. All optimizers showed a consistent decline in training loss across epochs, albeit at varying rates and efficiencies, as the models learned from the data and adjusted their parameters. This was observed despite the fact that the rates at which the models learned and the efficiencies at which they adjusted their parameters varied. The RMSProp optimizer displayed the most reliable performance, showing the greatest reduction in the amount of training loss. Within the end of the second epoch, RMSProp's loss had already been cut down to 7.7244, indicating an early advantage. This trend carried on, with the loss eventually coming down to an impressive 1.1771 by the time epoch 70 arrived. In terms of the percentage reduction in loss from epoch

1 to epoch 70, RMSProp demonstrated an approximately 87% decrease in loss, which was the highest among all of the optimizers. Adam was able to keep a consistent and effective decrease in loss over the course of multiple epochs, even though he began with a slightly higher initial loss than RMSProp. By the time we reached the 70th and final epoch, Adam's loss had been reduced to 1.2117. This represents a reduction of approximately 86.8% from its starting point, closely aligning with the performance of RMSProp.

The initial loss that was incurred by AdaMax was greater than that which was incurred by either RMSProp or Adam. However, it also demonstrated a steady improvement in loss over the course of time, ultimately culminating in a loss of 1.3486 at epoch 70. This results in an approximate 85.8% reduction in loss over the 70 epochs, which is a commendable performance despite trailing behind RMSProp and Adam in terms of overall efficiency. Within the parameters of this analysis, SGD was the optimizer that produced the poorest results. At epoch 70, the value of SGD's loss was still fairly high, coming in at 8.4793. This amounts to a loss reduction of only 13.3%, which is significantly lower than the reductions achieved by the other optimizers. There is a significant gap in terms of performance between the optimizer with the best results, RMSProp, and the optimizer with the worst results, SGD. RMSProp demonstrated a reduction in loss of 87 percent, whereas SGD was only able to achieve a reduction of 13.3 percent. This stark contrast, which amounts to about 73.7% of the difference, highlights the significant influence that the optimizer choice has on the model's performance.

In conclusion, the varying rates of decrease in training loss are a reflection of the various efficiencies and convergence speeds of the optimizers. The performance that RMSProp and Adam turned in was exceptional, and they were successful in achieving a significant and prompt reduction in loss. In spite of the fact that it was slower, AdaMax displayed consistent learning throughout the entirety of the game. In contrast to this, SGD demonstrated a significantly slower convergence speed, which resulted in a loss reduction that occurred at a more measured pace. This was achieved by reducing the number of lost packets.

7.4 | Validation loss

Figure 18 depicts the validation loss values obtained during the training of a machine learning model utilizing four distinct optimization algorithms: Adam, RMSProp, SGD, and AdaMax. The accuracy values are recorded over the course of 70 epochs, reflecting the model's performance at each training iteration.

At the beginning of the first epoch, all optimizers displayed high validation loss values, with Adam displaying 8.9842, RMSProp displaying 8.4810, SGD displaying 9.1727, and AdaMax displaying 9.0371. The errors that were made in the models' predictions based on the validation dataset are represented by these high initial validation losses. The degree to which an optimizer was successful in fine-tuning the model parameters in response to the loss function was reflected in the rate at which the optimizer's loss began to decrease over the

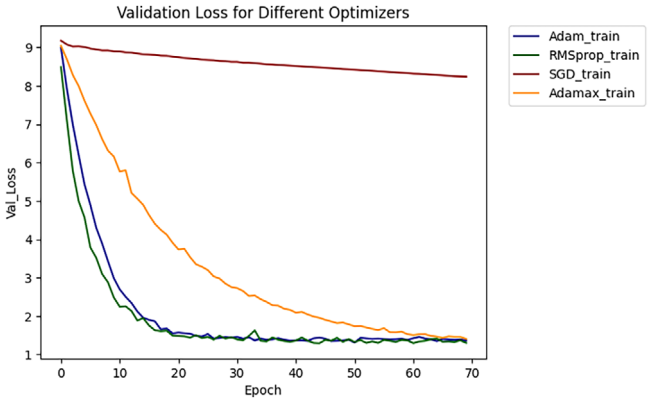

FIGURE 18 Validation loss of four optimizers.

course of subsequent epochs. When we looked at RMSProp, we saw that it demonstrated a steady and dramatic decrease in validation loss. From epoch 1 to epoch 5, RMSProp went from 8.4810 to 4.5723 and then back up again. After that, it maintained its downward trend and reached its lowest point of 1.3023 by the end of epoch 70. This translates to a remarkable decrease of approximately 84.6% in total. It is clear that the RMSProp optimizer has a high level of expertise in adjusting the model's parameters, as evidenced by the steady and dramatic decrease in validation loss.

On the other hand, Adam demonstrated a decrease in validation loss as well, albeit at a rate that was marginally less rapid than RMSProp's. As an illustration, by epoch 5, the validation loss for Adam had decreased from 8.9842 to 5.4313, while the validation loss for RMSProp had decreased to 4.5723. However, it continued to decrease in a steady fashion, reaching a validation loss of 1.3601 by epoch 70. This loss is equivalent to a reduction of approximately 84.9% of the original value. Adam's progression over the epochs demonstrates its efficiency, despite the fact that it got off to a slightly slower start when compared to RMSProp. AdaMax, which began with an initial validation loss of 9.0371, demonstrated a rate of decrease that was slower than that of both Adam and RMSProp. At the end of epoch 5, it had fallen to 7.6126, which was still higher than Adam and RMSProp at the same epoch. However, it continued to demonstrate consistent learning and loss reduction, reaching a loss of 1.4013 by epoch 70. This loss is slightly higher than the other two optimizers at the same epoch, but it is still significantly lower than the initial loss. This reduction represents a drop of approximately 84.5% of the total amount.

Even though this is a significant improvement, it is important to point out that AdaMax learned at a slower pace compared to RMSProp and Adam. This can be seen by the more gradual decline in validation loss. In contrast, the SGD exhibited a very different trend throughout the day. It began with the highest initial validation loss, which was 9.1727, and the decrease in its validation loss over the course of epochs was significantly slower than the decrease in the validation loss of the other optimizers. For instance, the validation loss for SGD was quite high at the epoch 5 point, coming in at 9.0084. This pattern of extremely high validation loss persisted throughout all of the epochs, with

the validation loss at epoch 70 remaining at 8.2363, which is significantly higher than all of the other optimizers. When compared to the other optimizers, this minuscule reduction only represents a decrease of 10.2%, which is a significant departure from the norm.

When looking at the effectiveness of various optimizers in relation to validation loss at the 70th epoch, it is clear that there is a significant gap between the performance of the optimizer with the most advantageous performance, RMSProp, and the performance of the optimizer with the least advantageous performance, SGD. Considerable 6.934 is what the absolute difference between these validation losses comes out to when calculated. This value offers a definitive and quantitative measurement of the gap in performance that exists between the two poles, RMSProp and SGD. It is estimated that RMSProp and SGD have a difference of approximately 84.2% in terms of the percentage of their validation loss. This percentage reaffirms the disparity that has been observed while also providing a form that is easier to understand. It is clear from this that the validation loss for the optimizer that performed the best, RMSProp, is approximately 84.2% higher than the validation loss for the optimizer that performed the worst, SGD.

The trajectories of validation loss for each of these optimizers highlight the differences in their respective efficiencies in learning from the validation dataset and lowering the prediction error. RMSProp and Adam appear to have the best performance, displaying steep and steady declines in validation loss. This indicates that learning and tuning of model parameters are occurring effectively. AdaMax demonstrates a consistent learning curve, despite the fact that it is more slowly developing than the previous two. On the other hand, SGD demonstrates slow learning and lower efficiency due to its high validation loss rate.

7.5 | Test runs average

Table 4 provides a comparative analysis of four different optimizers: Adam, RMSprop, SGD, and AdaMax, based on the results of 30 experimental runs. It outlines their respective performance in terms of train and test accuracy, represented as percentages. The data covers mean, median, mode, robustness, interquartile range (IQR), standard deviation, and variance for both train and test accuracy for each optimizer. The mean train accuracy ranges from 35.32% (SGD) to 77.16% (RMSprop), and the mean test accuracy varies from 33.41% (SGD) to 68.11% (RMSprop).

8 | DISCUSSION

The in-depth analysis of optimization algorithms used in this research reveals important insights into the functionality of these algorithms within the setting of our machine learning model. After conducting an exhaustive investigation into our findings, it has become abundantly clear that the RMSProp optimizer is the best option available. This optimizer provides superior performance in terms of accuracy and robustness while

TABLE 4 Comparative analysis of optimizers after 30 runs.

Results/optimizers	Train accuracy (%)							Test accuracy (%)							Computational time (s)
	Mean	Median	Mode	Robustness	IQR	Standard deviation	Variance	Mean	Median	Mode	Robustness	IQR	Standard deviation	Variance	
Adam	75.96	75.96	76.23	1.23	0.51	0.34	0.11	67.24	67.28	67.51	1.3	0.7	0.41	0.16	678
RMSprop	77.16	77.18	76.87	1.32	0.68	0.41	0.17	68.11	68.08	68.12	1.25	0.59	0.38	0.18	612
SGD	35.32	35.34	35.00	1.26	0.76	0.39	0.15	32.41	33.44	33.70	0.9	0.49	0.28	0.08	645
AdaMax	70.41	70.48	70.36	1.2	0.67	0.38	0.14	61.13	61.10	60.68	1.16	0.61	0.35	0.12	627

still allowing for additional optimization and improvement. This conclusion is reached as a result of a number of significant factors, each of which has a significant impact on the performance and utility of our CNN model.

When comparing RMSProp and Adam's performance, it is essential to look at their mean accuracies in both training and testing. This will give you the most accurate picture of their abilities. The RMSProp optimizer produces results that are commendably high in terms of both the mean training accuracy (77.16%) and the mean test accuracy (68.11%). On the other hand, the Adam optimizer, despite delivering a respectable performance, falls just a little bit short, with a mean training accuracy of 75.96% and a mean test accuracy of 67.35%. The superior performance of RMSProp is further highlighted when contrasted with the SGD and AdaMax optimizers, which both show significantly lower mean accuracies. This highlights the fact that RMSProp is the superior optimizer. Specifically, the SGD optimizer has a poor performance, with a training accuracy of only 35.32% and a test accuracy of 33.41%. AdaMax also falls short of expectations, with a training accuracy of 70.41% and a test accuracy of 61.13%.

Another crucial factor in terms of performance is robustness, which can be defined as the ability to produce consistent results despite the presence of data that is both diverse and challenging. With Interquartile Range (IQR) values of 0.68 for training and 0.39 for testing, RMSProp demonstrates outstanding robustness, indicating that it maintains consistency even with outliers or unexpected data variations. Although Adam exhibits a level of robustness that is acceptable, with IQR values of 0.51 for training and 0.7 for testing, this level is not higher than the one shown by RMSProp.

In addition, the standard deviations shed light on yet another aspect of the contrast between the two performances. The fact that RMSProp has a lower standard deviation than Adam does suggest that the results obtained with RMSProp are less likely to be affected by outside factors and are, as a result, more accurate. The slightly higher standard deviation in test accuracy exhibited by Adam may suggest a wider spread in its results, which in turn may indicate an opportunity for additional optimization and refinement.

In contrast to the results of the preliminary research, the RMSProp algorithm outperforms Adam not only in terms of accuracy but also in terms of robustness, which has caused us to rethink our initial preference. To get to the heart of the matter, it is essential to emphasize the part that robustness and the IQR play in this analysis. A lower robustness and IQR are generally desirable because they indicate less variability in accuracy and, as a result, a more consistent performance across multiple runs of the model. In this scenario, RMSProp demonstrated comparable levels of robustness at 1.32, whereas the rest of the optimizers had lowers robustness which indicates a marginally higher degree of variability. This conclusion, however, does not lessen the potential value of Adam, which may still be an appropriate choice depending on the specifics of the situation or the data sets being considered. In spite of this, the evidence from our tests indicates that, in this particular scenario, RMSProp provides superior performance [81–88].

In conclusion, while the 'best' optimizer can be determined by a number of different factors, such as the particular characteristics of the dataset, the architecture of the model, and the various nuances in the implementation of the optimizers, our research indicates that RMSProp is the leading contender in this scenario. However, as is the case with all applications of machine learning, these findings should be evaluated in the context of the particular model and the data requirements.

9 | SUGGESTIONS FOR FUTURE WORKS

The numerous breakthroughs that have been achieved in the field of FER by making use of CNNs have shed light on the vast scope of opportunities that exist for additional research and technological advancement in this area of the scientific discipline. Even though we have demonstrated that machine learning is effective in the analysis of facial expressions, we have barely even begun to scratch the surface of what is possible in the field of emotion-aware artificial intelligence systems. Considering the significant progress that has been made up to this point, we are able to pinpoint a number of areas in which further research has the potential to make significant contributions to this developing field.

(i) **Integration with voice analysis**: The combination of facial expression recognition and voice sentiment analysis has the potential to serve as an exciting new path for the continuation of research soon. This multimodal fusion has the potential to bring about an interpretation of human emotional states that is significantly more comprehensive and precise, thereby elevating the concept of machine empathy to new heights. Imagine a scenario in which an artificial intelligence system can accurately interpret both visual and auditory cues to gauge a user's emotional state. This would result in an interaction between humans and machines that was more natural and empathic. This cutting-edge area of research may have significant repercussions for a variety of industries, including those dealing with customer service and healthcare, as well as those dealing with entertainment and gaming.

(ii) **Real-time FER**: The development of models that are capable of real-time FER is another potential direction that research could go in. The ramifications of such technological advances are extremely broad, ranging from interactive gaming and the personalization of user experiences to diagnostics for mental health and the development of advanced security systems. Real-time FER systems could see significant improvements in speed, efficiency, and accuracy if subsequent research investigated methods to reduce the amount of time spent processing data, also known as latency.

(iii) **Personalized models**: Since individual, cultural, and environmental factors all have an impact on a person's facial expressions, potential future research might centre on the development of personalized models. These models could be programmed to learn and adjust to the specific

emotional expression patterns of individual users, which would result in a higher degree of personalization and accuracy in the emotion recognition process. This emphasis on personalization has the potential to further bridge the gap between AI systems and the human users of those systems, thereby facilitating interactions that are more genuine and nuanced.

(iv) **Energy-efficient models**: As the world continues to place a growing emphasis on sustainability and responsible energy usage, the development of energy-efficient FER models is an area that is ripe for exploration and should be pursued. This kind of research might involve the development of algorithms that provide high levels of performance despite using only a small number of computational resources. This dual-focus approach has the potential to significantly reduce the energy footprints of these systems, which would be in line with the goals of global sustainability initiatives and the growing demand for green AI solutions.

(v) **Ethical and privacy concerns**: As FER technologies become more widespread, addressing ethical and privacy concerns becomes an issue of the utmost importance. In subsequent work, we might investigate the possibility of developing comprehensive privacy protection measures, consent management systems, and transparent usage policies to guarantee the ethical deployment and utilization of these technologies.

(vi) **Expanding dataset diversity**: Future works could also focus on expanding the diversity of datasets used in FER. Leveraging datasets beyond the FER 2013 dataset or creating new, more diverse datasets could enhance the model's inclusivity and universal applicability. This would enable the model to recognize a broader range of facial expressions across different demographic groups, thus enhancing its reliability and generalizability.

10 | CONCLUSION

The capacity of automated systems to accurately recognise facial expressions is a key enabler for a variety of applications, including emotion-sensitive artificial intelligence as well as enhanced human-computer interactions. These applications range in scope from emotional intelligence to enhanced human-computer interactions. These kinds of applications necessitate the use of reliable machine learning models that are able to efficiently learn from extensive datasets and to generalize their findings to data they have not previously encountered. The success of these models is heavily dependent on the architecture that was selected, the quality of the training data, the efficiency of the pre-processing pipeline, and the optimizer that was selected for the learning process. This study delved into these complexities and presented a pipeline that trained a CNN model for the task of FER using the FER 2013 dataset. This dataset was used to identify a variety of facial expressions. The study used the ImageDataGenerator class for real-time data augmentation, leveraged the simplicity and effectiveness of the sequential model architecture from Keras, and investigated four distinct optimizers for the learning process: Adam, RMSprop, SGD and AdaMax. The research included a thoughtful pre-processing stage using the class.

AUTHOR CONTRIBUTIONS

Syed Hamid Hussain Madni: Conceptualization; formal analysis; investigation; methodology; project administration; supervision; validation; visualization; writing—review & editing. **Lokessh A/L Pathmanatan**: Conceptualization; formal analysis; investigation; methodology; validation; writing—original draft. **Muhammad Faheem**: Investigation; validation; writing—review & editing. **Hafiz Muhammad Faisal Shahzad**: Conceptualization; investigation; software; writing—review & editing. **Sajid Shah**: Formal analysis; investigation; validation; visualization; writing—review & editing.

ACKNOWLEDGEMENT

The research work of Muhammad Faheem is supported by the VTT Technical Research Center of Finland.

CONFLICT OF INTEREST STATEMENT

The authors declare no conflicts of interest.

DATA AVAILABILITY STATEMENT

Data sharing is not applicable to this article as no new data were created or analysed in this study.

ORCID

Syed Hamid Hussain Madni https://orcid.org/0000-0002-3816-1382
Muhammad Faheem https://orcid.org/0009-0000-2274-6821
Sajid Shah https://orcid.org/0009-0006-9980-101X

REFERENCES

1. Bui, T., Phan, N., Spitsyn, V., Bolotova, Y.A., Savitsky, Y.V.: Development of algorithms for face and character recognition based on wavelet transforms, PCA and neural networks. In: 2015 International Siberian Conference on Control and Communications (SIBCON), pp. 1–6. IEEE, Piscataway, NJ (2015)
2. Melinte, D.O., Vladareanu, L.: Facial expressions recognition for human–robot interaction using deep convolutional neural networks with rectified adam optimizer. Sensors 20(8), 2393 (2020)
3. Abuhussein, M.F.A., Darwish, A., Hassanien, A.E.: Exploring thermography technology: a comprehensive facial dataset for face detection, recognition, and emotion. arXiv:2407.09494 (2024).
4. Rostami, M., Farajollahi, A., Parvin, H.: Deep learning-based face detection and recognition on drones. J. Ambient Intell. Hum. Comput. 15(1), 373–387 (2024)
5. Lu, Z., Zhou, C., Wang, H.: A face detection and recognition method built on the improved MobileFaceNet. Int. J. Sens. Networks 45(3), 166–176 (2024)
6. Winarno, E., Hadikurniawati, W., Nirwanto, A.A., Abdullah, D.: Multi-view faces detection using Viola-Jones method. J. Phys. Conf. Ser. 1114, 012068 (2018)
7. Zakaria, Z., Suandi, S.A., Mohamad-Saleh, J.: Hierarchical skin-adaboost-neural network (h-skann) for multi-face detection. Appl. Soft Comput. 68, 172–190 (2018)
8. Baqeel, H., Saeed, S.: Face detection authentication on smartphones: end users usability assessment experiences. In: 2019 International

Conference on Computer and Information Sciences (ICCIS), pp. 1–6. IEEE, Piscataway, NJ (2019)
9. Krichen, M.: Convolutional neural networks: a survey. Computers 12(8), 151 (2023)
10. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature 521(7553), 436–444 (2015)
11. Yamashita, R., Nishio, M., Do, R.K.G., Togashi, K.: Convolutional neural networks: an overview and application in radiology. Insights Imaging 9, 611–629 (2018)
12. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press, Cambridge, MA (2016)
13. Schroff, F., Kalenichenko, D., Philbin, J.: FaceNet: a unified embedding for face recognition and clustering. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 815–823. IEEE, Piscataway, NJ (2015)
14. Parkhi, O.: Vedaldi, A., Zisserman, A.: Deep face recognition. In: BMVC 2015-Proceedings of the British Machine Vision Conference 2015, pp. 1–12. BMVA, Durham (2015)
15. Zhang, K., Zhang, Z., Wang, H., Li, Z., Qiao, Y., Liu, W.: Detecting faces using inside cascaded contextual CNN. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 3171–3179. IEEE, Piscataway, NJ (2017)
16. Sun, Y., Chen, Y., Wang, X., Tang, X.: Deep learning face representation by joint identification-verification. In: Advances in Neural Information Processing Systems, pp. 1988–1996. ACM, New York, NY (2014)
17. Adjabi, I., Ouahabi, A., Benzaoui, A., Taleb-Ahmed, A.: Past, present, and future of face recognition: a review. Electronics 9(8), 1188 (2020)
18. Wanyonyi, D., Celik, T.: Open-source face recognition frameworks: a review of the landscape. IEEE Access 10, 50601–50623 (2022)
19. Almeida, J., Vilaça, L., Teixeira, I.N., Viana, P.: Emotion identification in movies through facial expression recognition. Appl. Sci. 11(15), 6827 (2021)
20. Canal, F.Z., et al.: A survey on facial emotion recognition techniques: a state-of-the-art literature review. Inf. Sci. 582, 593–617 (2022)
21. Saarimäki, H., et al.: Classification of emotion categories based on functional connectivity patterns of the human brain. NeuroImage 247, 118800 (2022)
22. Shanthi, P., Nickolas, S.: Facial landmark detection and geometric feature-based emotion recognition. Int. J. Biom. 14(2), 138–154 (2022)
23. Liu, C., Hirota, K., Ma, J., Jia, Z., Dai, Y.: Facial expression recognition using hybrid features of pixel and geometry. IEEE Access 9, 18876–18889 (2021)
24. Van Nam, N., Quyen, N.T.N.: Flash: Facial landmark detection using active shape model and heatmap regression. In: International Conference on Industrial Networks and Intelligent Systems, pp. 170–182. Springer, Cham (2023)
25. Paul, S.K., Bouakaz, S., Rahman, C.M., Uddin, M.S.: Component-based face recognition using statistical pattern matching analysis. Pattern Anal. Appl. 24, 299–319 (2021)
26. Yaddaden, Y.: An efficient facial expression recognition system with appearance-based fused descriptors. Intell. Syst. Appl. 17, 200166 (2023)
27. Niu, B., Gao, Z., Guo, B.: Facial expression recognition with LBP and ORB features. Comput. Intell. Neurosci. 2021(1), 8828245 (2021)
28. Cheng, Y., Wang, H., Bao, Y., Lu, F.: Appearance-based gaze estimation with deep learning: a review and benchmark. IEEE Trans. Pattern Anal. Mach. Intell. 46, 7509–7528 (2024)
29. Kim, T., Yu, C., Lee, S.: Facial expression recognition using feature additive pooling and progressive fine-tuning of CNN. Electron. Lett. 54(23), 1326–1328 (2018)
30. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. Commun. ACM 60(6), 84–90 (2017)
31. Li, C., Ma, N., Deng, Y.: Multi-network fusion based on CNN for facial expression recognition. In: 2018 International Conference on Computer Science, Electronics and Communication Engineering (CSECE2018), pp. 166–169. Atlantis Press, Amsterdam (2018)
32. Cardaioli, M., Conti, M., Orazi, G., Tricomi, P.P., Tsudik, G.: BLUFADER: blurred face detection & recognition for privacy-friendly continuous authentication. Pervasive Mob. Comput. 92, 101801 (2023)
33. Venkatachalam, K., Siuly, S., Kumar, M.V., Lalwani, P., Mishra, M., Kabir, E.: A hybrid approach for COVID-19 detection using biogeography-based optimization and deep learning. Comput. Mater. Contin. 70(2), 3717–3732 (2021)
34. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. arXiv:1412.6980 (2014)
35. Liu, L., et al.: On the variance of the adaptive learning rate and beyond. arXiv:1908.03265 (2019)
36. Zou, F., Shen, L., Jie, Z., Zhang, W., Liu, W.: A sufficient condition for convergences of adam and rmsprop. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 11127–11135. IEEE, Piscataway, NJ (2019)
37. Yedida, R., Saha, S., Prashanth, T.: Lipschitzlr: Using theoretically computed adaptive learning rates for fast convergence. Appl. Intell. 51, 1460–1478 (2021)
38. Ziyin, L., Li, B., Simon, J.B., Ueda, M.: SGD with a constant large learning rate can converge to local maxima. arXiv:2107.11774 (2021)
39. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778. IEEE, Piscataway, NJ (2016)
40. Wang, L., Yang, Y., Min, R., Chakradhar, S.: Accelerating deep neural network training with inconsistent stochastic gradient descent. Neural Networks 93, 219–229 (2017)
41. Zeng, X., Zhang, Z., Wang, D.: AdaMax online training for speech recognition, Beijing, China, 1–5 (2016)
42. Dozat, T.: Incorporating nesterov momentum into Adam, United Kingdom, 1–4 (2016)
43. Rouast, P.V., Adam, M.T., Chiong, R.: Deep learning for human affect recognition: insights and new developments. IEEE Trans. Affective Comput. 12(2), 524–543 (2019)
44. Kartowisastro, I.H., Latupapua, J.: A comparison of adaptive moment estimation (Adam) and RMSProp optimisation techniques for wildlife animal classification using convolutional neural networks. Rev. Intell. Artif. 37(4), 1123–1130 (2023)
45. Sun, R.-Y.: Optimization for deep learning: An overview. J. Oper. Res. Soc. China 8(2), 249–294 (2020)
46. Bottou, L., Curtis, F.E., Nocedal, J.: Optimization methods for large-scale machine learning. SIAM Rev. 60(2), 223–311 (2018)
47. Soydaner, D.: A comparison of optimization algorithms for deep learning. Int. J. Pattern Recognit. Artif. Intell. 34(13), 2052013 (2020)
48. Chaudhury, S., Yamasaki, T.: Robustness of adaptive neural network optimization under training noise. IEEE Access 9, 37039–37053 (2021)
49. FER 2013 (Kaggle open database) https://www.kaggle.com/datasets/msambare/fer2013. Accessed 24 March 2013
50. Pontryagin, L.S.: Mathematical Theory of Optimal Processes. Routledge, Oxfordshire (2018)
51. Liu, Y., Pu, H., Sun, D.-W.: Efficient extraction of deep image features using convolutional neural network (CNN) for applications in detecting and analysing complex food matrices. Trends Food Sci. Technol. 113, 193–204 (2021).
52. Guo, Y., Wang, B., Li, W., Yang, B.: Protein secondary structure prediction improved by recurrent neural networks integrated with two-dimensional convolutional neural networks. J. Bioinf. Comput. Biol. 16(05), 1850021 (2018)
53. Naqvi, S.F., et al.: Real-time stress assessment using sliding window based convolutional neural network. Sensors 20(16), 4400 (2020)
54. Ram, A., Reyes-Aldasoro, C.C.: The relationship between fully connected layers and number of classes for the analysis of retinal images. arXiv:2004.03624 (2020)
55. Laurent, C., Pereyra, G., Brakel, P., Zhang, Y., Bengio, Y.: Batch normalized recurrent neural networks. In: 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 2657–2661. IEEE, Piscataway, NJ (2016)
56. Schilling, F.: The effect of batch normalization on deep convolutional neural networks. Degree Thesis, KTH Royal Institute Of Technology (2016)
57. Phaisangittisagul, E.: An analysis of the regularization between L2 and dropout in single hidden layer neural network. In: 2016 7th International

Conference on Intelligent Systems, Modelling and Simulation (ISMS), pp. 174–179. IEEE, Piscataway, NJ (2016)
58. Gulli, A., Pal, S.: Deep Learning with Keras. Packt Publishing, Birmingham (2017)
59. Sharma, N., Jain, V., Mishra, A.: An analysis of convolutional neural networks for image classification. Procedia Comput. Sci. 132, 377–384 (2018)
60. Sensoy, M., Kaplan, L., Kandemir, M.: Evidential deep learning to quantify classification uncertainty. In: Advances in Neural Information Processing Systems, pp. 3183–3193. ACM, New York, NY (2018)
61. Tharwat, A.: Classification assessment methods. Appl. Comput. Inf. 17(1), 168–192 (2021).
62. Gupta, R., Singh, A.: Hand gesture recognition using OpenCV. In: 2023 10th International Conference on Computing for Sustainable Global Development (INDIACom), pp. 145–148. IEEE, Piscataway, NJ (2023)
63. Yang, J.: Real time object tracking using OpenCV. In: 2023 IEEE 3rd International Conference on Data Science and Computer Application (ICDSCA), pp. 1472–1475. IEEE, Piscataway, NJ (2023)
64. Hamid, N.A.W.A., Singh, B.: High-performance computing based operating systems, software dependencies and IoT integration. In: High Performance Computing in Biomimetics: Modeling, Architecture and Applications, pp. 175–204. Springer, Cham (2024)
65. Taye, M.M.: Theoretical understanding of convolutional neural network: Concepts, architectures, applications, future directions. Computation 11(3), 52 (2023)
66. Sarkar, D., Bali, R., Ghosh, T.: Hands-On Transfer Learning with Python: Implement Advanced Deep Learning and Neural Network Models Using TensorFlow and Keras. Packt Publishing, Birmingham (2018)
67. Bettray, T.: Machine learning based on KERAS for single top physics at ATLAS. Degree Thesis, University of Bonn (2019)
68. Harris, C.R., et al.: Array programming with NumPy. Nature 585(7825), 357–362 (2020)
69. Molin, S.: Hands-On Data Analysis with Pandas: Efficiently Perform Data Collection, Wrangling, Analysis, and Visualization Using Python. Packt Publishing, Birmingham (2019)
70. Laštovička, M., Husák, M., Velan, P., Jirsík, T., Čeleda, P.: Passive operating system fingerprinting revisited: evaluation and current challenges. Comput. Networks 229, 109782 (2023)
71. Adebanjo, S., Banchani, E.: Application of different python libraries for visualisation of female genital mutilation. Int. J. Data Sci. 4(2), 67–83 (2023)
72. Antunes, B., Hill, D.R.: Reproducibility, energy efficiency and performance of pseudorandom number generators in machine learning: a comparative study of python, numpy, tensorflow, and pytorch implementations. arXiv:2401.17345 (2024)
73. Luke, O.A.: Enhancing Sign Language Recognition and Hand Gesture Detection Using Convolutional Neural Networks and Data Augmentation Techniques. Nova Southeastern University, Fort Lauderdale, FL (2024)
74. Suresh Kumar, B., Viswanadha Raju, S., Uma Maheswari, V.: OpenCV libraries for computer vision. In: Computer Vision: Applications of Visual AI and Image Processing, pp. 1–22. De Gruyter, Berlin (2023)
75. Salehin, I., Kang, D.-K.: A review on dropout regularization approaches for deep neural networks within the scholarly domain. Electronics 12(14), 3106 (2023)
76. Albayati, A.Q., Altaie, S.A.J., Al-Obaydy, W.N.I., Alkhalid, F.F.: Performance analysis of optimization algorithms for convolutional neural network-based handwritten digit recognition. IAES Int. J. Artif. Intell. 13(1), 563–571 (2024)
77. Gupta, P., Sehgal, N.K., Acken, J.M.: Machine learning algorithms. In: Introduction to Machine Learning with Security: Theory and Practice Using Python in the Cloud, pp. 45–176. Springer, Cham (2024)
78. Zollanvari, A.: Three fundamental python packages. In: Machine Learning with Python: Theory and Implementation, pp. 63–109 Springer, Cham (2023)
79. Pathan, S., Siddalingaswamy, P., Kumar, P., MM, M.P., Ali, T., Acharya, U.R.: Novel ensemble of optimized CNN and dynamic selection techniques for accurate COVID-19 screening using chest CT images. Comput. Biol. Med. 137, 104835 (2021)
80. Salehi, A.W., et al.: A study of CNN and transfer learning in medical imaging: advantages, challenges, future scope. Sustainability 15(7), 5930 (2023)
81. Faheem, M., Al-Khasawneh, M. A., Khan, A. A., Madni, S. H. H.: Cyberattack patterns in blockchain-based communication networks for distributed renewable energy systems: A study on big datasets. Data in Brief 53(5), 110212 (2024a)
82. Raza, B., Kumar, Y. J., Malik, A. K., Anjum, A., Faheem, M.: Performance prediction and adaptation for database management systemworkload using case-based reasoning approach. Inf. Syst. 76(5), 46–58 (2018)
83. Faheem, M., Mahmoud Ahmad, A.-K.: Multilayer cyberattacksidentification and classification using machine learning in internetof blockchain (IoBC)-based energy networks. Data in Brief. 54(5), 110461 (2024)
84. Butt, R. A., et al.: A survey of dynamic bandwidth assignment schemes for TDM-based passive optical network. J. Opt. Commun. 41(3), 279–293 (2020)
85. Muhammad, F., et al.: A multiobjective, lion mating optimization inspired routing protocol for wireless body area sensor network based healthcare applications. Sensors 19(23), 5072 (2019)
86. Al-Khasawneh, M. A. S., Faheem, M., Aldhahri, E. A., Alzahrani, A., Alarood, A. A.: A MapReduce based approach for secure batch satellite image encryption. IEEE Access. 11, 62865–62878 (2023)
87. Raza, B., Aslam, A., Sher, A., Malik, A. K., Faheem, M.: Autonomic performance prediction framework for data warehouse queries usinglazy learning approach. Appl. Therm. Eng. 91, 106216 (2020)
88. Faheem, M., Raza, B., Bhutta, M. S., Madni, S. H. H.: A blockchain-based resilient and secure framework for events mon-itoring and control in distributed renewable energy systems. IETBlockchain 1–15 (2024b)

How to cite this article: Madni, S.H.H., A/L Pathmanatan, L., Faheem, M., Shahzad, H.M.F., Shah, S.: Exploring optimizer efficiency for facial expression recognition with convolutional neural networks. J. Eng. 2025, e70060 (2025).
https://doi.org/10.1049/tje2.70060