BAREKENG: Journal of Mathematics and Its Application
September 2022 Volume 16 Issue 3 Page 927-938
P-ISSN: 1978-7227 E-ISSN: 2615-3017
https://doi.org/10.30598/barekengvol16iss3pp927-938

PERFORMANCE COMPARISON OF GRADIENT-BASED CONVOLUTIONAL NEURAL NETWORK OPTIMIZERS FOR FACIAL EXPRESSION RECOGNITION

Sri Nurdiati[1*], Mohamad Khoirun Najib[2], Fahren Bukhari[3], Refi Revina[4], Fitra Nuvus Salsabila[5]

[1,2,3,4,5]*Department of Mathematics, Faculty of Mathematics and Natural Sciences, IPB University*
Meranti Kampus St., Babakan, Dramaga, Bogor, 16680, Indonesia

Corresponding author's e-mail: [1] nurdiati@apps.ipb.ac.id*

Abstract. *A convolutional neural network (CNN) is one of the machine learning models that achieve excellent success in recognizing human facial expressions. Technological developments have given birth to many optimizers that can be used to train the CNN model. Therefore, this study focuses on implementing and comparing 14 gradient-based CNN optimizers to classify facial expressions in two datasets, namely the Advanced Computing Class 2022 (ACC22) and Extended Cohn-Kanade (CK+) datasets. The 14 optimizers are classical gradient descent, traditional momentum, Nesterov momentum, AdaGrad, AdaDelta, RMSProp, Adam, Radam, AdaMax, AMSGrad, Nadam, AdamW, OAdam, and AdaBelief. This study also provides a review of the mathematical formulas of each optimizer. Using the best default parameters of each optimizer, the CNN model is trained using the training data to minimize the cross-entropy value up to 100 epochs. The trained CNN model is measured for its accuracy performance using training and testing data. The results show that the Adam, Nadam, and AdamW optimizers provide the best performance in model training and testing in terms of minimizing cross-entropy and accuracy of the trained model. The three models produce a cross-entropy of around 0.1 at the 100th epoch with an accuracy of more than 90% on both training and testing data. Furthermore, the Adam optimizer provides the best accuracy on the testing data for the ACC22 and CK+ datasets, which are 100% and 98.64%, respectively. Therefore, the Adam optimizer is the most appropriate optimizer to be used to train the CNN model in the case of facial expression recognition.*

Keywords: *AlexNet architecture, confusion matrix, convolutional neural network, deep learning, facial expression recognition, gradient-based optimizer.*

Article info:

Submitted: 16th June 2022 *Accepted: 12th August 2022*

How to cite this article:
S. Nurdiati, M. K. Najib, F. Bukhari, F. N. Salsabila, and R. Refina, "PERFORMANCE COMPARISON OF GRADIENT-BASED CONVOLUTIONAL NEURAL NETWORK OPTIMIZERS FOR FACIAL EXPRESSION RECOGNITION", *BAREKENG: J. Math. & App.*, vol. 16, iss. 3, pp. 927-938, September, 2022.

1. INTRODUCTION

Facial expression recognition is part of the classification problem and is still a challenging area of research [1]. Not only needed in human-to-human communication, but facial expression recognition also plays an essential role in human-computer communication, including human-robot interaction [2]. In an automated system, the system can provide services according to the emotions of each customer. In other applications, facial expression recognition can also be used in virtual reality [3], augmented reality [4], and mental disease diagnosis [5].

Many methods have been developed to recognize facial expressions. In general, an automatic facial expression recognition system usually consists of four stages: data pre-processing, feature extraction, feature selection, and classification [6]. One method that gives good results is the convolutional neural network (CNN) [7]. The advantage of CNN is that the feature extraction and selection stages are carried out automatically at the feature learning stage using convolutional and sub-sampling layers. After that, the outcome of the feature learning is flattened and then used as input for the classification using fully-connected layers. Although it gives good results, the main problem of classification using CNN is the large number of parameters, so the training process requires a long computational time [8].

Various optimizers that can be used to train CNN models have been developed by many experts. The earliest and most common optimization method is gradient descent [9]. The gradient descent method is easy to implement, but convergence speed is slower as it approaches the optimal solution. The speed of this convergence can be accelerated by considering both traditional and Nesterov momentum. Subsequently, an optimizer with an adaptive learning rate was developed, known as AdaGrad. AdaGrad performs more significant updates for infrequent and minor updates for frequent parameters. AdaGrad is extended into two methods, namely AdaDelta and RMSprop. Adaptive Moment Estimation (Adam) is another method that computes adaptive learning rates for each parameter, designed to combine the advantages of the two methods, namely AdaGrad and RMSProp. Moreover, Adam has many variations, such as rectified Adam, AdaMax, AMSGrad, NADAM, ADAMW, OADAM, and AdaBelief.

With the development of many choices of optimizers that can be used to train CNN, this study aims to implement and compare 14 CNN optimizers for facial expression recognition problems. Facial expression data comes from Advance Computing Class 2022 (ACC22) and Extended Cohn-Kanade (CK+) datasets. The performance measured is the convergence speed of the optimizer to minimize the loss function used, namely cross-entropy. In addition, the accuracy of the CNN model on the training and testing data is also measured. The study results are expected to provide suggestions on an excellent optimizer that can be used to optimize CNN problems, especially in the case of facial expression recognition.

2. RESEARCH METHODS

2.1 Data Collection

This study uses two facial expression datasets, i.e., the dataset collected from applied mathematics master's degree students taking the 2022 advanced computing class (ACC22) and the dataset from the extended Cohn-Kanade (CK+) dataset. The details of each dataset are as follows.

2.1.1 Advance Computing Class 2022 (ACC22) Dataset

The first data was collected from IPB University applied mathematics master's students, who are taking advanced computing courses in 2022, called the Advance Computing Class 2022 (ACC22) Dataset. This data consists of seven students who express three types of expressions: neutral, smile, and grin [10]. The data has been augmented into 567 labelled data, then stored with grayscale and a pixel size of 48 so that in the WHCN (width, height, channel, batch size) order, the size of ACC22 data is $48 \times 48 \times 1 \times 567$.

2.1.2 Extended Cohn-Kanade (CK+) Dataset

The CK+ dataset is an extension of the CK dataset. This data was collected from a total of 123 subjects aged 18 to 50 years [11]. This study extracted the last three frames from each sequence in the CK+ dataset, resulting in 981 data labelled with seven different types of expressions, i.e., angry, disgust, fear, happy, sad, surprise, and neutral. The CK+ data is also extracted at grayscale and a pixel size of 48 so that in the WHCN order, the size of CK+ data is 48×48×1×981.

2.2 Convolutional Neural Network (CNN)

A convolutional neural network (CNN) is a feedforward neural network that can extract features from image data automatically using convolution structures. CNN has many advantages compared to general artificial neural networks: 1) Local connection. Each neuron in the new layer is only connected to a small number of neurons in the previous layer, effectively reducing parameters and accelerating convergence; 2) Weight sharing. A group of connections can share the same weight, which reduces parameters even further. 3) Sub-sampling dimensionality reduction [12]. In general, there are two stages in the CNN model, i.e., feature learning and classification. Feature learning usually consists of convolution and pooling layers, while the classification layer consists of fully connected layers.

Convolution is essential for feature learning, producing outputs called feature maps. Four components must be defined in the convolution layer: padding, stride, kernel, and activation function. When setting the convolution kernel to a specific size, the information on the border will be lost. Therefore, padding was introduced to enlarge the input by zero on each edge, which can adjust the size indirectly. The number of additional borders depends on the needs and the size of the specified kernel. Once the padding is set, the convolution operation is performed according to the fixed kernel. In a two-dimensional CNN, the kernel is a matrix of size $n \times n$ containing a number of parameter values. Each sub-matrix on the input is multiplied point by point by the kernel then summed and operated with an activation function. The sub-matrix is shifted right and down so that each value in the input matrix is convoluted. The length of the shift step is called the stride. Stride is employed to control the density of convolving. The larger the stride, the lower the density. After convolution, feature maps consist of a large number of features that is prone to causing overfitting problem. As a result, sub-sampling (max-pooling and average pooling) is proposed to avoid redundancy. After all the convolution and sub-sampling processes have been carried out, the last feature maps are flattened and used as input to the fully-connected layer for the classification stage. Figure 1 illustrates the procedure of convolution layer with max-pooling on two-dimensional CNN.

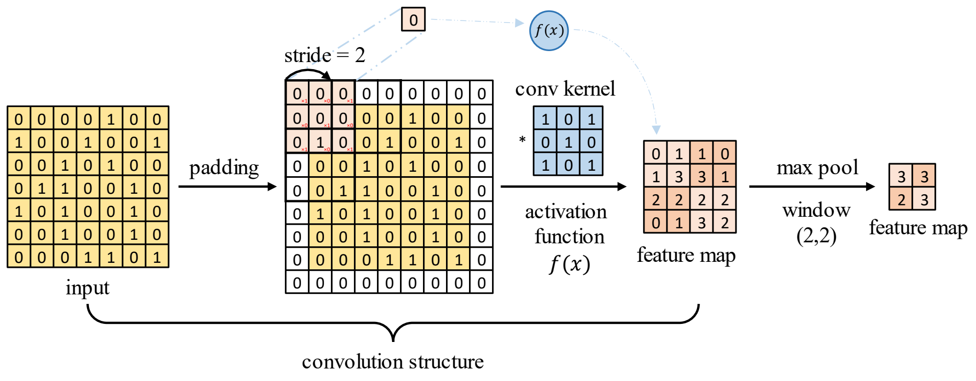

Figure 1. Procedure of convolution layer with max-pooling on two-dimensional CNN

In the development of CNN, many architectures have been designed to solve classification problems. The first architecture that became the forerunner of CNN was LeNet-5, published in 1992 [13]. This was followed by several variations of LeNet-5, such as AlexNet and VGG, published in 2012 and 2014 [14]. This study uses an AlexNet-like architecture to classify facial expressions from both datasets, with a design as shown in Figure 2.

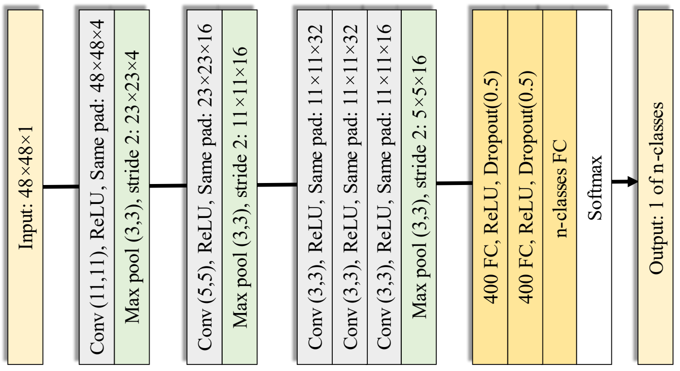
Input: 48×48×1
Conv (11,11), ReLU, Same pad: 48×48×4
Max pool (3,3), stride 2: 23×23×4
Conv (5,5), ReLU, Same pad: 23×23×16
Max pool (3,3), stride 2: 11×11×16
Conv (3,3), ReLU, Same pad: 11×11×32
Conv (3,3), ReLU, Same pad: 11×11×32
Conv (3,3), ReLU, Same pad: 11×11×16
Max pool (3,3), stride 2: 5×5×16
400 FC, ReLU, Dropout(0.5)
400 FC, ReLU, Dropout(0.5)
n-classes FC
Softmax
Output: 1 of n-classes

Figure 2. AlexNet-like architecture for n-classes facial expression recognition

AlexNet comprises eight processing layers, including five convolution layers and three fully connected layers. The hallmark of AlexNet is three successive convolution layers, after two convolution and sub-sampling layers. The sub-sampling uses max pooling, and the activation function used is rectified linear unit (ReLU), given by $f(x) = x^+ = \max(0, x)$. In deep learning, if the estimation of parameters uses limited data, it can lead to high variance and overfitting. Therefore, dropout is employed on fully-connected layers to prevent overfitting and improve the generalization ability of the network [15].

2.3 Optimizer

Before the CNN model is ready to be used, the model must be trained using training data to minimize the loss function $L(\theta)$, which is a measure of the proximity of the model output to the actual label. This study uses cross-entropy as a loss function given by

$$L(\theta) = -\frac{1}{k}\sum_{i=1}^{k} y_i \cdot \log f_\theta(x_i) \tag{1}$$

where k is the number of training samples, $f_\theta(x)$ is the CNN model to be learned, and θ is the parameter to be optimized [16]. This study uses 14 optimizers to train the CNN model, with the details of each optimizer as follows.

2.3.1 Gradient Descent

Gradient descent is one of the most popular algorithms for optimizing neural networks [17]. Gradient descent computes the gradient of the loss function w.r.t. the parameters for the training dataset, and the parameters are updated in the negative gradient direction to minimize the loss function given by

$$\theta^* = \theta - \eta \cdot \frac{\partial L(\theta)}{\partial(\theta)} \tag{2}$$

where η is the learning rate, which determines the step size in each iteration and thus influences the number of iterations to reach the optimal value [18].

2.3.2 Momentum Gradient Descent

In classic gradient descent, the update speed is fixed and does not take into account updates in previous iterations (epochs), i.e.

$$v = -\eta \cdot \frac{\partial L(\theta)}{\partial(\theta)} \tag{3}$$

By utilizing the information from the update in the previous epoch, the convergence speed of the gradient descent method can be accelerated. This method, called momentum, helps accelerate gradient descent in the relevant direction and dampens oscillations [19][20]. This method adds a fraction γ of the update vector in the previous epoch to the current update vector, i.e.

$$\begin{aligned} v_t &= \gamma \cdot v_{t-1} - \eta \cdot \frac{\partial L(\theta)}{\partial(\theta)}, \\ \theta^* &= \theta + v_t \end{aligned} \tag{4}$$

where γ is called the momentum factor and is usually set to 0.9 or similar.

2.3.3 Nesterov Accelerated Gradient Descent

The Nesterov Accelerated Gradient Descent (NAG) is an improvement over the traditional momentum method [21]. In Nesterov momentum, the momentum $\gamma \cdot v_{t-1}$ is also added to θ, when calculating the gradient of the loss function. Therefore, the updating formula of Nesterov momentum is given by

$$\begin{aligned} v_t &= \gamma \cdot v_{t-1} - \eta \cdot \frac{\partial L(\theta + \gamma \cdot v_{t-1})}{\partial(\theta)}, \\ \theta^* &= \theta + v_t \end{aligned} \tag{5}$$

The improvement of Nesterov momentum is reflected in updating the gradient of the future position instead of the current position.

2.3.4 AdaGrad

Adaptive gradient descent (AdaGrad) is a refinement of the gradient descent method that adjusts the learning rate dynamically based on the historical gradient in previous iterations [22]. The updating formula of AdaGrad is given by

$$\begin{aligned} g_t &= \frac{\partial L(\theta_t)}{\partial(\theta)}, \\ \theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{\sum_{i=1}^{t}(g_i)^2 + \epsilon}} \cdot g_t \end{aligned} \tag{6}$$

where θ_t is the parameter of θ at iteration t, and g_t is the gradient of parameter θ_t. Since the learning rate changes with each iteration based on the accumulative gradient of the previous iterations, manual learning rate tuning are no longer necessary.

2.3.5 AdaDelta

The main problem with AdaGrad is that the learning rate will go to zero as iterations increase, causing parameter updates to become ineffective. AdaDelta only focuses on the gradients in a window over a period and uses the exponential moving average to calculate the second-order cumulative momentum [23]. The updating formula of AdaDelta is given by

$$\begin{aligned} E[g^2]_t &= \beta \cdot E[g^2]_{t-1} + (1-\beta) \cdot g_t^2, \\ E[\Delta\theta^2]_{t-1} &= \beta \cdot E[\Delta\theta^2]_{t-2} + (1-\beta) \cdot \Delta\theta_{t-1}^2, \\ \Delta\theta_t &= -\frac{\sqrt{E[\Delta\theta^2]_{t-1} + \epsilon}}{\sqrt{E[g^2]_t + \epsilon}} \cdot g_t \\ \theta_{t+1} &= \theta_t + \Delta\theta_t \end{aligned} \tag{7}$$

where β is the exponential decay parameter. With AdaDelta, there is no need to set the learning rate η, as it has been removed from the update rule.

2.3.6 RMSProp

RMSProp and AdaDelta have been developed independently around about the same time. Both were designed to cope with the learning rate of AdaGrad, which is radically going to zero. With a learning procedure that is almost similar to AdaDelta, the updating formula of RMSProp is given by

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} \cdot g_t \tag{8}$$

where β is the exponential decay parameter and is usually set to 0.9 at a learning rate of 0.001 [24]. The difference between RMSProp and AdaDelta is that RMSProp still requires settings for the learning rate.

2.3.7 Adam

Adaptive moment estimation (Adam) is another advanced gradient descent method which combines the adaptive learning rate and momentum methods [25]. In addition to storing an exponentially decaying average of past squared gradients, like AdaDelta and RMSProp, Adam also keeps an exponentially decaying average of past gradients, similar to the momentum method. The updating formula of Adam is given by

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t, \qquad \hat{m}_t = \frac{m_t}{(1-\beta_1^t)}$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2, \qquad \hat{v}_t = \frac{v_t}{(1-\beta_2^t)} \tag{9}$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \cdot \hat{m}_t$$

where β_1 and β_2 are exponential decay rates. The default values of β_1, β_2, and ϵ are suggested to be set to 0.9, 0.999, and 10^{-8}, respectively.

2.3.8 Radam

Due to the limited number of samples in the early stage of model training, the adaptive learning rate has an undesirably large variance. It can cause the model to converge to suspicious or bad local optima. Rectified Adam (Radam) is a variation of Adam that overcomes this issue [26]. Radam considers the existence of high variance by estimating the degrees of freedom in each iteration, i.e.,

$$\rho_\infty = \frac{2}{1-\beta_2} - 1, \qquad \rho_t = \rho_\infty - \frac{2t\beta_2^t}{1-\beta_2^t} \tag{10}$$

If the variance is tractable, i.e., $\rho_t > t$, then the updating formula of Radam is given by

$$l_t = \sqrt{\frac{1-\beta_2^t}{v_t}}, \qquad r_t = \sqrt{\frac{(\rho_t - 4)(\rho_t - 2)\rho_\infty}{(\rho_\infty - 4)(\rho_\infty - 2)\rho_t}}$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \cdot \hat{m}_t \cdot r_t \cdot l_t \tag{11}$$

Otherwise, Radam will calculate the regular Adam optimizer.

2.3.9 AdaMax

The v_t factor in Adam scales the gradient inversely proportionally to the l_2 norm of the past gradients and the current gradient. AdaMax is a variant of Adam which considers l_∞ rather than l_2 [25]. In AdaMax, the value of v_t is replaced with u_t, so that the updating formula of AdaMax is given by

$$u_t = \beta_2^\infty v_{t-1} + (1-\beta_2^\infty)|g_t|^\infty = \max(\beta_2 v_{t-1}, |g_t|)$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{u_t} \cdot \hat{m}_t \tag{12}$$

Bias correction for u_t is not necessary because u_t depends on the max operation, and bias towards zero is not as recommended as m_t and v_t in Adam.

2.3.10 AMSGrad

AMSGrad slightly modifies the Adam optimizer to provide the algorithm with long-term memory of past gradients [27][28]. AMSGrad slightly changes the bias correction of v_t, denoted by $\tilde{v}_t$ to avoid confusion with Adam. Thus, the updating formula of AMSGrad is given by

$$\tilde{v}_t = \max(\tilde{v}_{t-1}, v_t), \qquad \tilde{V}_t = \mathrm{diag}(\tilde{v}_t)$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\tilde{v}_t}} \cdot m_t \tag{13}$$

Note that AMSGrad does not use the correction bias of m_t, but uses m_t itself.

2.3.11 Nadam

Nesterov-accelerated adaptive moment estimation (Nadam) is a combination optimizer of the Nesterov momentum and Adam. The updating formula of Nadam is given by

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \cdot \left(\beta_1 \cdot \hat{m}_t + \frac{1-\beta_1}{1-\beta_1^t} \cdot g_t\right) \tag{14}$$

The formula derivation from Nadam can be studied further in [29][18].

2.3.12 AdamW

Adam with decoupled weight decay (AdamW) is a variant of ADAM fixing (as in repairing) its weight decay regularization. The updating formula of AdamW is given by

$$\theta_{t+1} = \theta_t - \zeta_t \left(\frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \cdot \hat{m}_t + \lambda \cdot \theta_t \right) \tag{15}$$

where λ is the weight decay value. Note that AdamW uses the l_2 regularization to calculate the gradient of the parameter θ_t, i.e., $g_t = \partial L(\theta_t)/\partial(\theta) + \lambda\theta_t$. To account for possible scheduling of η and λ, AdamW introduce a scaling factor ζ_t delivered by a user-defined procedure *SetScheduleMultiplier(t)* [30]. Within the i-th run, the value of ζ_t decays according to a cosine annealing learning rate for each batch [31].

2.3.13 OAdam

Optimistic Adam (OAdam) is a variant of Adam adding an "optimistic" term suitable for adversarial training. The updating formula of OAdam is given by

$$\theta_{t+1} = \theta_t - \frac{2\eta}{\sqrt{\hat{v}_t} + \epsilon} \cdot \hat{m}_t + \frac{\eta}{\sqrt{\hat{v}_{t-1}} + \epsilon} \cdot \hat{m}_{t-1} \tag{16}$$

Optimistic Adam is claimed to be able to achieve high numbers of inception scores after very few epochs of training [32].

2.3.14 AdaBelief

The AdaBelief optimizer is another variant of the well-known ADAM optimizer. No extra parameters are introduced in AdaBelief. Specifically, in Adam, the update direction is $m_t/\sqrt{v_t}$, where v_t is the EMA of g_t^2; in AdaBelief, the update direction is $m_t/\sqrt{s_t}$, where s_t is the EMA of $(g_t - m_t)^2$ [33]. Thus, the updating formula of AdaBelief is given by

$$\begin{gathered} s_t = \beta_2 s_{t-1} + (1-\beta_2)(g_t - m_t)^2 + \epsilon, \qquad \hat{s}_t = \frac{s_t}{(1-\beta_2^t)} \\ \theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{s}_t} + \epsilon} \cdot \hat{m}_t \end{gathered} \tag{17}$$

AdaBelief adaptively scales the step size by the difference between the predicted and observed gradients. AdaBelief is the first optimizer to achieve three goals simultaneously: fast convergence as in adaptive methods, good generalization as in SGD, and training stability in complex settings such as GANs.

The CNN model in Figure 2 was trained using the 14 optimizers mentioned above to classify facial expressions in the ACC22 and CK+ datasets. The training process uses 80% of the data, and the other 20% is used for testing. The hyperparameter values of each optimizer use their respective best default settings, as in Table 1. Moreover, the value of ϵ is set to 10^{-8} for the optimizer that requires it.

Table 1. Best default setting of each optimizer's hyperparameter value

Optimizer	Hyper-parameter(s)	Optimizer	Hyper-parameter(s)
Gradient descent	$\eta = 0.1$	Radam	$\eta = 0.001, \beta_1 = 0.9, \beta_2 = 0.999$
Momentum	$\eta = 0.01, \gamma = 0.9$	AdaMax	$\eta = 0.001, \beta_1 = 0.9, \beta_2 = 0.999$
Nesterov	$\eta = 0.001, \gamma = 0.9$	AMSGrad	$\eta = 0.001, \beta_1 = 0.9, \beta_2 = 0.999$
AdaGrad	$\eta = 0.1$	Nadam	$\eta = 0.001, \beta_1 = 0.9, \beta_2 = 0.999$
AdaDelta	$\beta = 0.9$	AdamW	$\eta = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \lambda = 0$
RMSProp	$\eta = 0.001, \beta = 0.9$	OAdam	$\eta = 0.0001, \beta_1 = 0.5, \beta_2 = 0.9$
Adam	$\eta = 0.001, \beta_1 = 0.9, \beta_2 = 0.999$	AdaBelief	$\eta = 0.001, \beta_1 = 0.9, \beta_2 = 0.999$

2.4 Performance Metrics

After the model is trained, the model's performance is assessed using a confusion matrix based on training and testing data. For the case of n-class classification, the confusion matrix obtained can be seen in Figure 3. The value of $C_{1,1}$ indicates the amount of data that is actually class-1 predicted by the model in class-1 as well, while $C_{1,2}$ shows the actual data is class-1 but is predicted to be class-2 by the model. The same interpretation applies to values $C_{1,3}$, $C_{2,1}$, to $C_{3,3}$. The prediction model's accuracy can be calculated using the resulting confusion matrix. For the case of classification with n classes, the accuracy is determined by

$$accuracy = \frac{\sum_{i=1}^{n} C_{i,i}}{\sum_{i=1}^{n} \sum_{i=1}^{n} C_{i,j}} \tag{18}$$

The accuracy value is the ratio between the number of main diagonals (trace) and the number of all elements in the confusion matrix.

		Prediction				
		Class-1	Class-2	Class-3	$\cdots$	Class-n
Actual	Class-1	$C_{1,1}$	$C_{1,2}$	$C_{1,3}$	$\cdots$	$C_{1,n}$
	Class-2	$C_{2,1}$	$C_{2,2}$	$C_{2,3}$	$\cdots$	$C_{2,n}$
	Class-3	$C_{3,1}$	$C_{3,2}$	$C_{3,3}$	$\cdots$	$C_{3,n}$
	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
	Class-n	$C_{n,1}$	$C_{n,2}$	$C_{n,3}$	$\cdots$	$C_{n,n}$

Figure 3. Confusion matrix for n-classes classification problems.

2.5 Device Requirement

The entire computing process in this study was carried out using a Lenovo Ideapad 330-14AST with an AMD A4-9125 Radeon R3 processor, 4 Core (2C+2G) 2.30 GHz and 8 GB RAM. This study uses the Julia programming language version 1.6.5, which has fast performance, dynamic language, and is open source. The pre-processing stage was carried out using the *Images.jl* package. Meanwhile, the CNN model construction and training process used the *Flux.jl* package available on Julia.

3. RESULTS AND DISCUSSION

3.1 Pre-processing

The datasets are partitioned into two parts for training and testing, randomly by setting the seed of the random generator to 1. With a ratio of 80:20%, the training and testing data obtained are 454 and 113 images for ACC22, while as many as 785 and 196 images for CK+, respectively.

3.2 Performance Loss Function

Using the training data, each optimizer is used to train the CNN model to minimize the cross-entropy value. The training process is carried out up to 100 epochs. The same procedure was run 10 times to avoid computational bias. The following are the results of the cross-entropy (loss function) propagation and the computational time required for the ACC22 and CK+ datasets.

Figure 4 compares all optimizers that minimize the cross-entropy value in the first and second running processes. Both figures (4A and 4B) show that Adam, AdamW, and Nadam are the three fastest optimizers in training the CNN model for facial expression recognition on the ACC22 dataset. In only 100 epochs, the three optimizers obtained a cross-entropy of less than 0.1, although the fastest optimizer differed in the first and second runs. Adam is the fastest optimizer on the first run, but Adam is slower than Nadam and AdamW on the second run. Furthermore, several optimizers can train models at moderate convergence speeds, including AMSGrad, AdaMax, RMSProp, OAdam, Radam, and AdaDelta. There are behavioural differences in using the AdaBelief optimizer, which provides low convergence speed on the first run and moderate speed on the second run. Classic optimizers such as gradient descent, momentum, Nesterov, and AdaGrad provide very slow convergence speeds. Even worse, AdaGrad could not train the CNN model, which was indicated by an increase in the cross-entropy and stuck around the value of 10.

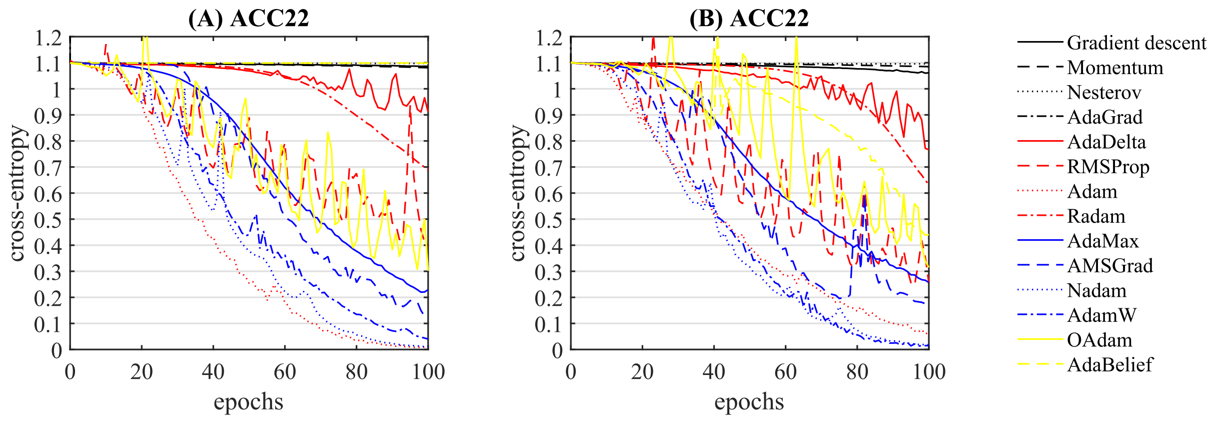

Figure 4. Comparison of cross-entropy propagation for each optimizer for facial expression recognition on the ACC22 dataset: (A) the first and (B) the second running processes.

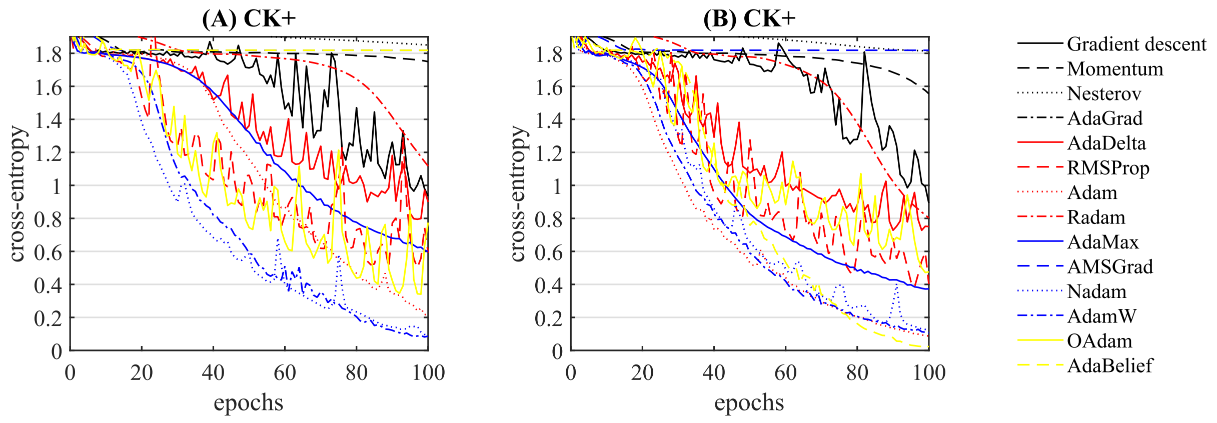

Figure 5. Comparison of cross-entropy propagation for each optimizer for facial expression recognition on the CK+ dataset: (A) the first and (B) the second running processes.

The same pattern is also seen in Figure 5, which shows the epoch-to-epoch cross-entropy propagation in the CK+ dataset. Adam, Nadam, and AdamW optimizers still hold the fastest optimizer. The behaviour of the AdaBelief optimizer also looks unstable in this dataset. In the first run, the AdaBelief optimizer belongs to the slow optimizer group, but it has a fast convergence speed in the second run. Even better, the cross-entropy value obtained at the 100th epoch is smaller than the Adam, Nadam, and AdamW optimizers.

Furthermore, differences are also seen in the AMSGrad optimizer. On the ACC22 dataset, AMSGrad can train the CNN model at a moderate speed, but AMSGrad is very slow on the CK+ dataset, where the cross-entropy value is stuck at 1.8. To further generalize the results obtained, the cross-entropy value at the 100th epoch of each optimizer for 10 runs is visualized in a boxplot, as shown in Figure 6.

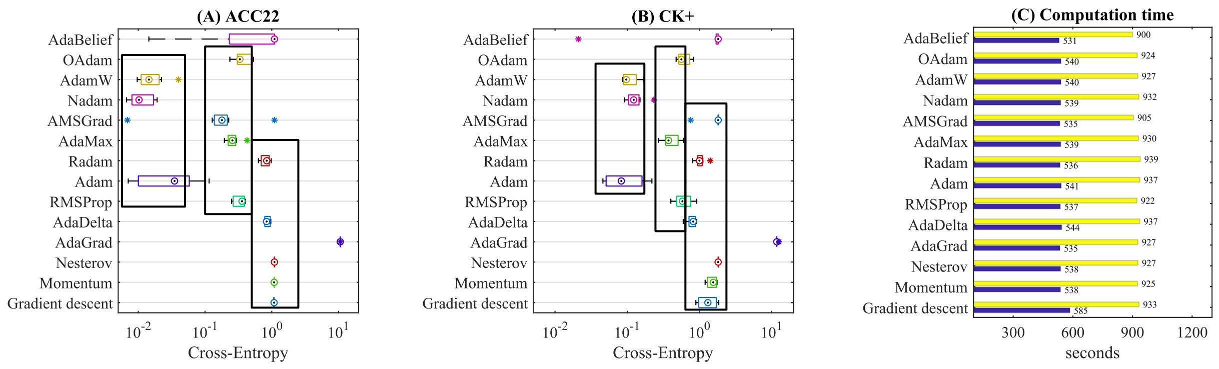

Figure 6. Boxplot of cross-entropy values after the 100th epoch for each optimizer of 10 runs to classify (A) ACC22 and (B) CK+ datasets; and (C) the minimum computation time required for each optimizer on the ACC22 (blue) and CK+ (yellow) datasets.

Figures 6A and 6B re-emphasize that the most consistently fast optimizers for training CNN models in facial expression recognition are the Adam, Nadam, and AdamW optimizers. Although it once had good results, the AdaBelief optimizer worked slowly. The same results were also shown for AMSGrad, where AMSGrad had a moderate convergence speed for training models on the ACC22 dataset but very slowly on the CK+ dataset. Moreover, Figure 6C shows the minimum computational time required for each optimizer to train the CNN model. The minimum (fastest) time is taken rather than the entire time because the slow computation time can be caused by other factors, such as interference on the device. Based on Figure 6c, the computation time of each optimizer to reach the 100th epoch is not significantly different. In the ACC22 dataset, the minimum computation time is around 500s, while the CK+ dataset requires a minimum computational time of around 900s.

3.3 Performance Accuracy

After comparing the resulting loss function after 100 epochs, the accuracy of each optimizer is calculated using Equation 18. The accuracy results from 10 runs are shown in the boxplot in Figure 7 for both training and testing data.

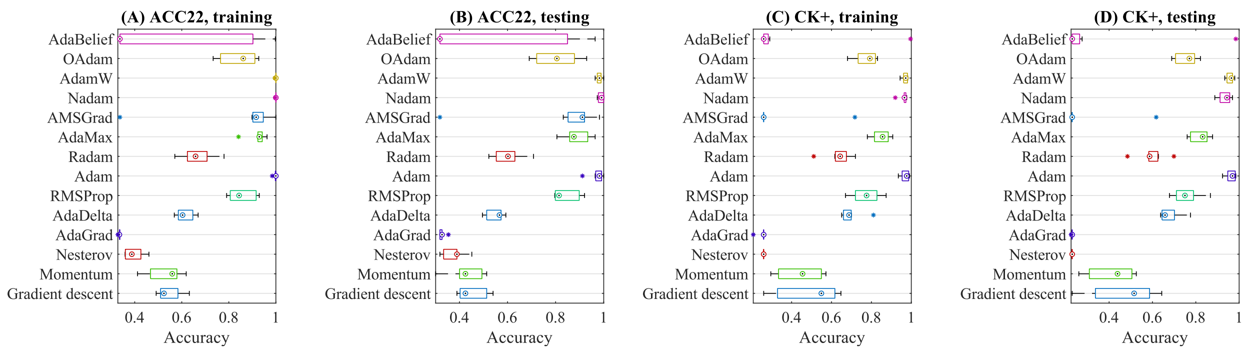

Figure 7. Boxplot of accuracy after 100th epoch for each optimizer of 10 runs to classify ACC22 dataset for (A) training and (B) testing data; and CK+ dataset for (C) training and (D) testing data.

The accuracy value strongly correlates with the loss function (cross-entropy), which is used as a target in model training. The lower the cross-entropy value, the better the accuracy of the model. The best accuracy value after 100 epochs is obtained using the Adam, Nadam, and AdamW optimizers. These optimizers provide accuracy values above 90% for both training and testing data for the ACC22 and CK+ datasets. Furthermore, optimizers with moderate convergence speed yield accuracy of around 80%, such as RMSProp, AdaMax, and Radam; and an additional AMSGrad optimizer for the ACC22 dataset. The best accuracy of each optimizer on the testing data can be seen in Table 2, both for the ACC22 and CK+ datasets.

Table 2. Best accuracy of each optimizer on the ACC22 and CK+ datasets testing data.

Optimizer	**ACC22 Dataset**		**CK+ Dataset**	
	Accuracy (%)	***Run***	***Accuracy (%)***	***Run***
Gradient descent	53.98	3	64.28	2
Momentum	51.32	1	52.55	5
Nesterov	45.13	4	22.95	1
AdaGrad	35.39	5	22.95	1
AdaDelta	59.29	1	77.55	4
RMSProp	92.03	2	86.73	2
Adam	**100**	**3**	**98.46**	**3**
Radam	70.79	2	69.89	2
AdaMax	96.46	5	87.75	3
AMSGrad	98.23	7	61.73	4
Nadam	**100**	**2**	96.93	1
AdamW	**100**	**4**	97.95	1
OAdam	92.92	3	82.14	2
AdaBelief	96.46	4	**98.46**	**2**

Referring to Table 2, the three optimizers, Adam, Nadam, and AdamW, were able to classify facial expressions in the ACC22 dataset with 100% accuracy for the testing data on the 3rd, 2nd and 4th run, respectively. Meanwhile, the Adam and AdaBelief optimizers provide the highest accuracy on the CK+ dataset, 98.46%, on the 3rd and 2nd run, respectively. However, even though they give good results, AdaBelief does not always consistently provide them. Thus, the Adam optimizer is more reliable for training CNN models. In the ACC22 dataset, the best optimizer can produce an accuracy of 100%, so the confusion matrix is a diagonal matrix, indicating that each facial expression can be predicted accurately. Meanwhile, the confusion matrix of the best results in the CK+ dataset is shown in Table 3, which uses the Adam optimizer. According to Table 3, misclassification occurs when predicting angry facial expressions. Three of the 25 test data expressed anger, but the CNN model predicted it would be sad.

Table 3. Confusion matrix of the best CNN model for the CK+ dataset.

		Prediction						
		angry	*disgust*	*fear*	*happy*	*sad*	*surprise*	*neutral*
Actual	*angry*	22	0	0	0	3	0	0
	disgust	0	34	0	0	0	0	0
	fear	0	0	21	0	0	0	0
	happy	0	0	0	44	0	0	0
	sad	0	0	0	0	18	0	0
	surprise	0	0	0	0	0	45	0
	neutral	0	0	0	0	0	0	9

4. CONCLUSIONS

This study focuses on implementing and comparing the performance of gradient-based CNN model optimizers for facial expression recognition. Using the ACC22 and CK+ datasets, the Adam, Nadam, and AdamW optimizers provide the best performance in model training and testing in terms of minimizing cross-entropy and accuracy of the trained model. The three models produce a cross-entropy of around 0.1 at the 100th epoch with an accuracy of more than 90% on both training and testing data. Furthermore, the Adam optimizer provides the best accuracy on the testing data for the ACC22 and CK+ datasets, which are 100% and 98.64%, respectively. Therefore, the Adam optimizer is the most appropriate optimizer to be used to train the CNN model in the case of facial expression recognition.

ACKNOWLEDGEMENT

The authors would like to thank Ade Irawan, Evi Ardiyani, Muhammad Reza Ardhana, Salsabilla Rahmah, and Trianty Putri Blante, who have been willing to contribute to data collection. The authors also thank the Department of Mathematics, IPB University, for all the support that has been given.

REFERENCES

[1] B. Niu, Z. Gao, and B. Guo, “Facial Expression Recognition with LBP and ORB Features,” *Comput. Intell. Neurosci.*, 2021, doi: 10.1155/2021/8828245.
[2] D. O. Melinte and L. Vladareanu, “Facial expressions recognition for human–robot interaction using deep convolutional neural networks with rectified adam optimizer,” *Sensors (Switzerland)*, vol. 20, no. 8, 2020, doi: 10.3390/s20082393.
[3] C. N. W. Geraets *et al.*, “Virtual reality facial emotion recognition in social environments: An eye-tracking study,” *Internet Interv.*, vol. 25, 2021, doi: 10.1016/j.invent.2021.100432.
[4] C. H. Chen, I. J. Lee, and L. Y. Lin, “Augmented reality-based self-facial modeling to promote the emotional expression and social skills of adolescents with autism spectrum disorders,” *Res. Dev. Disabil.*, vol. 36, pp. 396–403, 2015, doi: 10.1016/j.ridd.2014.10.015.
[5] N. Samadiani *et al.*, “A review on automatic facial expression recognition systems assisted by multimodal sensor data,” *Sensors (Switzerland)*, vol. 19, no. 8, 2019, doi: 10.3390/s19081863.
[6] A. Mahmood, S. Hussain, K. Iqbal, and W. S. Elkilani, “Recognition of Facial Expressions under Varying Conditions Using Dual-Feature Fusion,” *Math. Probl. Eng.*, vol. 2019, 2019, doi: 10.1155/2019/9185481.
[7] S. Li and W. Deng, “Deep facial expression recognition: a survey,” *J. Image Graph.*, vol. 25, no. 11, pp. 2306–2320, 2020, doi: 10.11834/jig.200233.
[8] I. H. Sarker, “Deep Learning: A Comprehensive Overview on Techniques, Taxonomy, Applications and Research Directions,” *SN Comput. Sci.*, vol. 2, no. 6, 2021, doi: 10.1007/s42979-021-00815-1.
[9] S. Sun, Z. Cao, H. Zhu, and J. Zhao, “A Survey of Optimization Methods from a Machine Learning Perspective,” *IEEE Trans. Cybern.*, vol. 50, no. 8, pp. 3668–3681, 2020, doi: 10.1109/TCYB.2019.2950779.
[10] S. Nurdiati, M. K. Najib, F. Bukhari, M. R. Ardhana, S. Rahmah, and T. P. Blante, “Perbandingan AlexNet dan VGG untuk Pengenalan Ekspresi Wajah pada Dataset Kelas Komputasi Lanjut.” 2022.

[11] P. Lucey, J. F. Cohn, T. Kanade, J. Saragih, Z. Ambadar, and I. Matthews, "The extended Cohn-Kanade dataset (CK+): A complete dataset for action unit and emotion-specified expression," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Workshops, CVPRW 2010*, 2010, pp. 94–101. doi: 10.1109/CVPRW.2010.5543262.
[12] Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou, "A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects," in *IEEE Transactions on Neural Networks and Learning Systems*, 2021, pp. 1–21. doi: 10.1109/tnnls.2021.3084827.
[13] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," in *Proceedings of the IEEE*, 1998, vol. 86, no. 11, pp. 2278–2323. doi: 10.1109/5.726791.
[14] M. Z. Alom *et al.*, "A state-of-the-art survey on deep learning theory and architectures," *Electron.*, vol. 8, no. 3, 2019, doi: 10.3390/electronics8030292.
[15] S. Park and N. Kwak, "Analysis on the dropout effect in convolutional neural networks," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 10112 LNCS. pp. 189–204, 2017. doi: 10.1007/978-3-319-54184-6_12.
[16] Z. I. Botev, D. P. Kroese, R. Y. Rubinstein, and P. L'Ecuyer, "The cross-entropy method for optimization," *Handb. Stat.*, vol. 31, pp. 35–59, 2013, doi: 10.1016/B978-0-444-53859-8.00003-5.
[17] J. Lu, "Gradient Descent, Stochastic Optimization, and Other Tales," *arXiv Prepr.*, no. arXiv:2205.00832, 2022.
[18] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv Prepr.*, no. arXiv:1609.04747, 2016.
[19] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Internal Representations by Error Propagation," in *Parallel distributed processing*, vol. 1, Cambridge, MA: MIT Press, 1986, pp. 318–362.
[20] N. Qian, "On the momentum term in gradient descent learning algorithms," *Neural Networks*, vol. 12, no. 1, pp. 145–151, 1999, doi: 10.1016/S0893-6080(98)00116-6.
[21] Y. Nesterov, "A method for unconstrained convex minimization problem with the rate of convergence O(1/k^2)," *Dokl. AN USSR*, vol. 269, pp. 543–547, 1983.
[22] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *J. Mach. Learn. Res.*, vol. 12, pp. 2121–2159, 2011.
[23] M. D. Zeiler, "ADADELTA: An Adaptive Learning Rate Method," *arXiv Prepr.*, no. arXiv:1212.5701, 2012.
[24] T. Tieleman and G. E. Hinton, "Coursera: Neural networks for machine learning," *University of Toronto, Technical Report*, 2012. https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf
[25] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," *arXiv Prepr.*, no. arXiv:1412.6980, pp. 1–15, 2017.
[26] L. Liu *et al.*, "On the Variance of the Adaptive Learning Rate and Beyond," *arXiv Prepr.*, no. arXiv:1908.03265v4, 2021.
[27] S. J. Reddi, S. Kale, and S. Kumar, "On the convergence of Adam and beyond," in *6th International Conference on Learning Representations, ICLR 2018*, 2018, pp. 1–23.
[28] T. T. Phuong and L. T. Phong, "On the Convergence Proof of AMSGrad and a New Version," *IEEE Access*, vol. 7, pp. 61706–61716, 2019, doi: 10.1109/ACCESS.2019.2916341.
[29] T. Dozat, "Incorporating Nesterov Momentum into Adam," in *ICLR Workshop*, 2016, no. 1, pp. 2013–2016.
[30] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in *7th International Conference on Learning Representations, ICLR 2019*, 2019, pp. 1–19.
[31] I. Loshchilov and F. Hutter, "SGDR: Stochastic gradient descent with warm restarts," in *5th International Conference on Learning Representations, ICLR 2017*, 2017, pp. 1–16.
[32] C. Daskalakis, A. Ilyas, V. Syrgkanis, and H. Zeng, "Training GaNs with optimism," in *6th International Conference on Learning Representations, ICLR 2018*, 2018, pp. 1–30.
[33] J. Zhuang *et al.*, "AdaBelief optimizer: Adapting stepsizes by the belief in observed gradients," *Adv. Neural Inf. Process. Syst.*, vol. 33, pp. 18795–18806, 2020.