ENHANCED KNOWLEDGE IN SCIENCES AND TECHNOLOGY
e-ISSN: 2773-6385

EKST

Vol. 5 No. 2 (2025) 210-220
https://publisher.uthm.edu.my/periodicals/index.php/ekst

Facial Expression Recognition Application using Convolutional Neural Network

Pang Kou Yi[1], Lee Siaw Chong[1]*

[1] *Department of Mathematics and Statistics, Faculty of Applied Sciences and Technology, UTHM Kampus Cawangan Pagoh, Hab Pendidikan Tinggi Pagoh, KM 1, Jalan Panchor, 84600 Pagoh, Muar, Johor, MALAYSIA.*

*Corresponding Author: sclee@uthm.edu.my
DOI: https://doi.org/10.30880/ekst.2025.05.02.024

Article Info

Received: 30 December 2024
Accepted: 17 January 2025
Available online: 19 December 2025

Keywords

Facial Expression Recognition, Deep Learning, Convolutional Neural Network

Abstract

Facial Expression Recognition (FER) is an important field of research in human-computer interaction, analysing facial expressions to understand human emotions. Traditional methods in FER suffer from low accuracy and rely on manual feature extraction, making them inefficient and less adaptable. To address these challenges, this study presents the development of a custom Convolutional Neural Network (CNN) model designed to classify seven facial expressions. CNNs are a deep learning technique widely used in image classification tasks due to their ability to efficiently extract relevant features from images. The research utilised the FER2013 dataset from Kaggle, which contains 35,887 labelled images, with 28,709 images for training and 7,178 images for testing. The custom CNN model achieved a maximum accuracy of 69% with a batch size of 32, a learning rate of 0.0001 and 400 training epochs. A confusion matrix analysis shows the model performs best with the "Happy" expression (88.3% true positive rate) and worst with the "Fear" expression (41.7% true positive rate). A tkinter-based application was created to allow users to leverage the CNN model to predict the facial expression of any given input image and display the breakdown of percentages for all seven predicted expressions.

1. Introduction

Facial expression recognition (FER) is a technology that analyses facial expressions captured in static images and videos to determine an individual's expressional state. Facial expressions refer to the changes in the faces that occur as a response to an individual's internal expressions, intentions, or social interactions [1]. Nowadays, automated facial expression recognition is widely applicable across diverse fields, including data-driven animation, neuromarketing, interactive gaming, sociable robotics and numerous other human-computer interaction systems [2].

In the field of Artificial Intelligence, machine learning (ML) has played a significant role in the development of facial expression recognition systems. ML has a broad range of applications across various domains, particularly in recommendation systems, classification, regression, clustering, and anomaly detection. For instance, in facial expression recognition, the algorithm is trained on a dataset where each facial image is labelled with the correct expression, such as happiness, sadness, anger, etc. From the labelled inputs, the algorithm can then learn and recognise patterns and features in the images associated with each expression. Some common ML algorithms for FER include decision trees, random forests, and neural networks.

Traditional machine learning algorithms, such as support vector machines (SVM) and random forests, have been used for facial expression recognition tasks but often face limitations in terms of accuracy and processing

speed. These algorithms struggled to capture the full complexity of human facial expressions. In contrast, Convolutional Neural Networks (CNNs), a deep learning technique, have emerged as a better solution due to their ability to accurately and efficiently recognise and classify facial expressions.

Deep learning, an advanced ML branch, uses artificial neural networks to learn from data. These neural networks are designed to resemble the structure and functions of the human brain, processing information through layers of interconnected nodes known as neurons. Each node in the network is responsible for learning specific features, allowing the network to handle complex tasks more effectively [3]. Artificial Neural Networks (ANNs) are a type of deep learning model that consists of three main layers, namely the input layer, hidden layer, and output layer. These layers work together to process and learn patterns, allowing the network to perform tasks such as classification. In recent years, deep learning has become a powerful tool for facial expression recognition, with algorithms like CNNs showing significant improvements over traditional machine learning techniques such as SVM, random forests and ANNs, especially in image processing tasks [4].

CNNs are better than ANNs in image classification tasks due to their ability to adaptively learn spatial hierarchies of features, from edges and shapes to complete objects. Unlike ANNs, CNNs use convolutional layers to extract and learn relevant patterns from images efficiently. Therefore, CNNs are preferred over ANNs for the development of FER systems.

The three core components of a CNN architecture are the convolutional layer, the pooling layer and the fully connected layer. In the convolutional layers, features are extracted from an input image using a series of learnable filters known as kernels. These filters capture the image's important patterns and details, such as edges or textures. Then, the feature maps' spatial size is reduced in the pooling layers. Finally, the fully connected layers transform and map the extracted features into the final output as classification scores. The basic architecture of CNN is illustrated in Fig. 1.

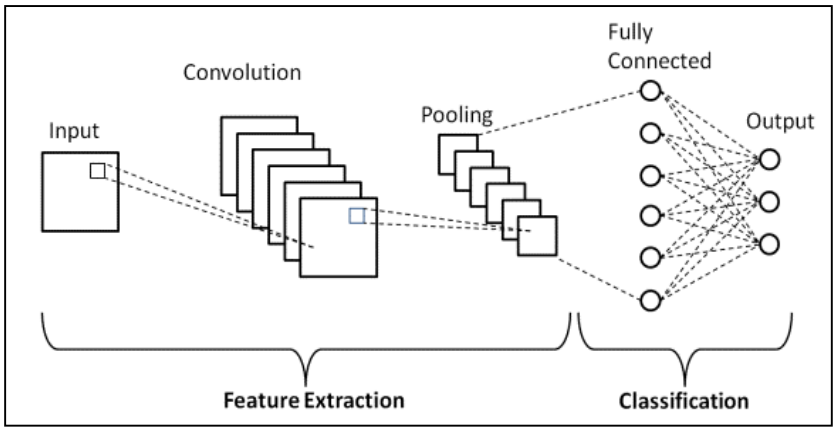


Fig. 1 *Basic CNN architecture [5]*

Some popular libraries for working with CNNs in Python include TensorFlow, Keras, PyTorch, and Caffe. TensorFlow, an open-source deep learning framework, is renowned for its computational graph foundation, which enables efficient computation and optimisation. It stands out because it supports automatic differentiation, distributed computing, and specialised hardware such as Graphics Processing Units (GPUs). TensorFlow also offers comprehensive visualisation tools, making it valuable in machine learning. Unlike deep learning libraries like Theano or Torch, TensorFlow introduces innovative features and improvements, further advancing its capabilities [6]. Due to its robustness and flexibility, TensorFlow is the primary library used in this research to construct CNN models.

In facial expression recognition, the Facial Expression Recognition 2013 Dataset (FER2013) is a widely used benchmark. FER2013 is a Kaggle dataset introduced by Pierre-Luc Carrier and Aaron Courvill at the 2013 International Conference on Machine Learning (ICML) [7]. It consists of 35,887 grayscale images with a resolution of 48 × 48 pixels. These images are classified into seven expressions: Angry, Disgust, Fear, Happy, Neutral, Sad, and Surprise. Examples from each category in the FER2013 dataset are presented in Fig. 2.

Fig. 2 Examples of facial expression categories in the FER2013 dataset [8]

In this research, a custom CNN model was developed for facial expression recognition. The architecture includes convolutional, max-pooling, dropout, and fully connected layers. Convolutional layers extract features by detecting spatial patterns, while max pooling reduces the feature map size, improving computational efficiency. Dropout layers help prevent overfitting by randomly deactivating neurons during training. The Rectified Linear Unit (ReLU) activation function introduces non-linearity, which improves the model's ability to learn. Finally, the fully connected layers aggregate features, with the softmax activation in the output layer providing probabilities for each expression class.

The custom CNN model was initially trained for 50 epochs using the Adam optimiser with a learning rate of 0.0001, a batch size of 64, and categorical cross-entropy loss. Additionally, the model's performance was tested with different hyperparameter values for batch size and epochs to measure both training time and accuracy in predicting the FER2013 dataset. The performance of the CNN model was evaluated using two common evaluation metrics: a confusion matrix and accuracy. A confusion matrix visually represents the performance of a classification algorithm by showing the counts of true negatives, true positives, false negatives, and false positives in the form of an $N \times N$ matrix where N is the number of target classes. Accuracy measures the proportion of correct predictions out of all predictions made.

This research aims to develop a CNN model for facial expression recognition using Keras and TensorFlow in Python, assess its performance through evaluation metrics such as accuracy and confusion matrix, and develop a user-friendly graphical user interface (GUI) for the system using tkinter in Python.

2. Methodology

In the development of a custom convolutional neural network (CNN) model for facial expression recognition (FER) in Python, the model is trained on the Facial Expression Recognition 2013 Dataset (FER2013). The dataset is preprocessed using normalisation and augmentation techniques. Using 13 layers, including 4 convolutional layers, 3 max-pooling layers, 3 dropout layers, 1 flatten layer and 2 dense layers, the architecture of the custom CNN model is created. The model is then trained on the preprocessed dataset using the Adam optimiser with a learning rate of 0.0001 and a categorical cross-entropy loss function. The performance of the models is then evaluated using metrics such as accuracy and a confusion matrix. An overview of the methodology is illustrated in Fig. 3.

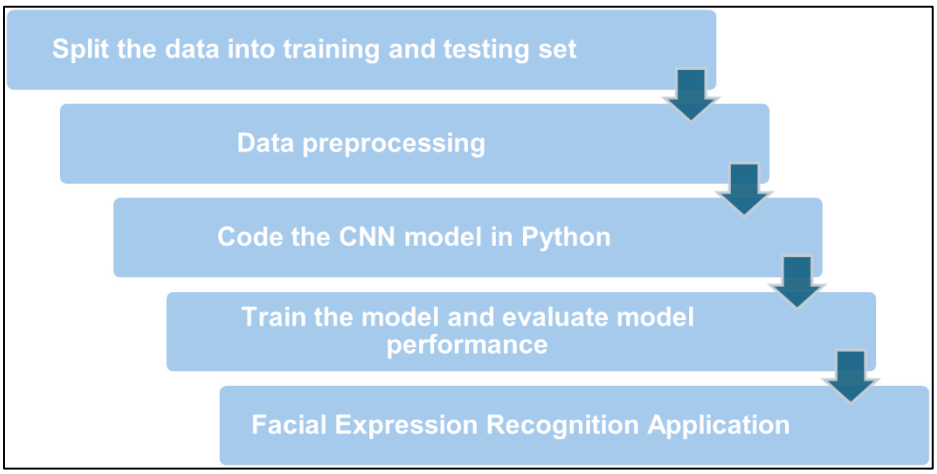


Fig. 3 Overview of the Methodology for Facial Expression Recognition

2.1 Concepts Behind Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are widely used for image recognition and feature extraction due to their ability to process visual data effectively. The primary components of a CNN architecture are the convolutional layers, pooling layers, activation functions, and fully connected layers. Each component plays a specific role in contributing to the overall functionality of the neural network.

2.1.1 Convolutional Layer

The convolutional layer uses a kernel (also called a filter) to extract features from an input tensor by performing element-wise multiplication and summing the results to produce a feature map. The stride specifies the amount by which the kernel shifts during this operation; it is usually set to 1 for detailed feature extraction, although larger strides can be used to down sample the data. Fig. 4 shows an example of a convolution operation.

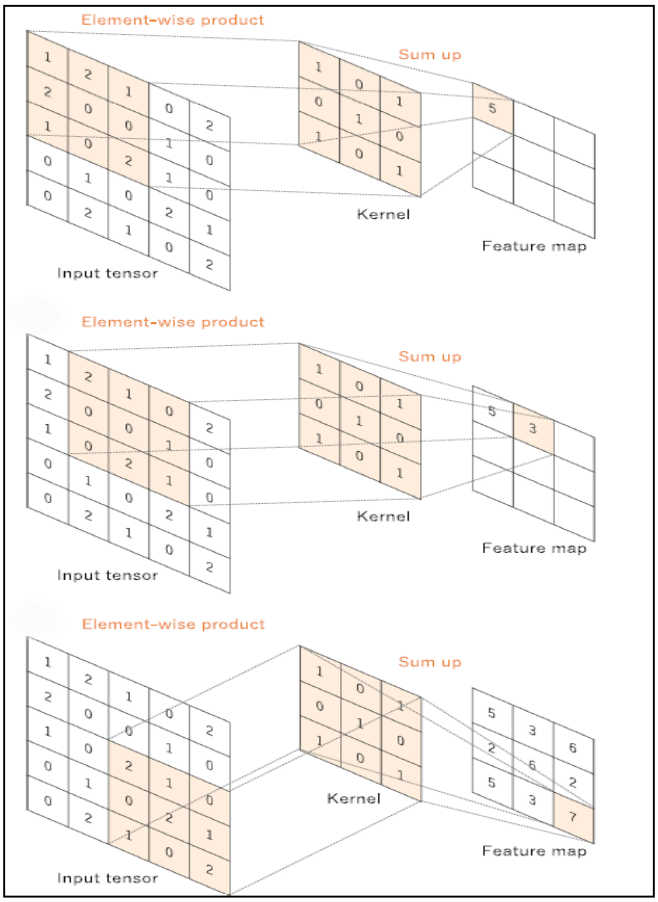


Fig. 4 Convolution operation with a kernel size of 3 × 3 [9]

According to Fig. 4, the convolution operation starts by applying a 3 × 3 kernel matrix, as follows:

$$K = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}.$$

A 3 × 3 kernel matrix is commonly used because it provides a good balance between capturing local patterns and computational efficiency. This specific kernel is designed to emphasise specific patterns in the input image. The central 1 focuses on the central pixel of the local region, while the 1s on the diagonals and anti-diagonals highlight diagonal patterns. The 0s in the kernel have minimal effect on the output, as they do not contribute to the convolution result at those positions. At each position, the element-wise product is calculated between the kernel matrix and the corresponding section of the input tensor. For example, at the top-left position, the element-wise product is:

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 0 & 0 \\ 1 & 0 & 2 \end{bmatrix} \odot \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}.$$

Thus, the value of the feature map at this position is calculated as:

$$1(1) + 2(0) + 1(1) + 2(0) + 0(1) + 1(1) + 0(0) + 0(0) + 2(1) = 5 \quad (1)$$

Since the value of the stride is 1, the convolutional filter shifts horizontally, one position at a time. This process repeats until a complete 3 × 3 feature map of the input tensor is obtained, as illustrated in Fig. 4. This method effectively captures local patterns and features, making it pivotal in tasks such as image recognition and feature extraction in convolutional neural networks.

2.1.2 ReLU Activation Function

The Rectified Linear Unit (ReLU) activation function is commonly used in Convolutional Neural Networks (CNNs). It outputs zero for negative inputs and the input value for positive ones. ReLU helps prevent the vanishing gradient

problem, allowing for more efficient training of deep networks by maintaining a constant gradient for positive values. The activation function is defined as

$$R(x) = \begin{cases} x, & x \geq 0, \\ 0, & x < 0. \end{cases} \tag{2}$$

2.1.3 Pooling Layer

Pooling layers reduce the spatial dimension of feature maps. The two types of pooling methods are max pooling and average pooling. Max pooling selects the maximum value of each local region. Fig. 5 shows an example of the max pooling operation.

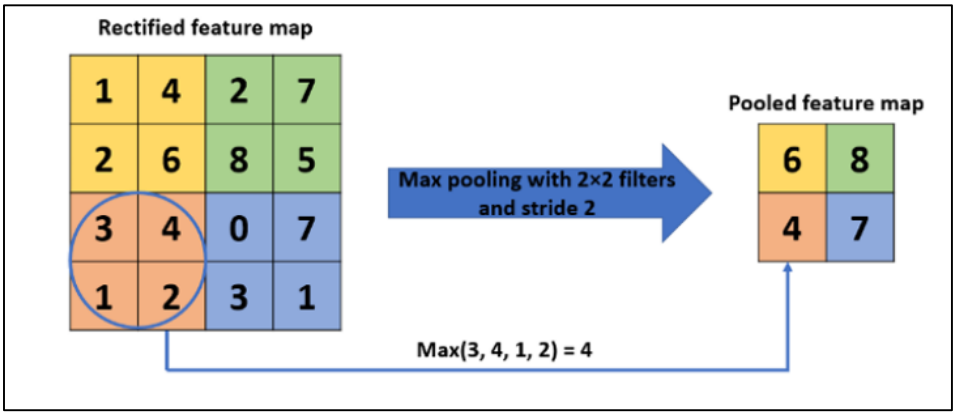


Fig. 5 Max pooling operation [10]

Fig. 5 demonstrates a max pooling operation. In this operation, each region of the input tensor, such as the orange-highlighted region, is processed independently. Within the highlighted region, which contains the numbers 3, 4, 1, and 2, max pooling selects the largest number, which is 4, to represent that specific region in the resulting feature map. This selection process repeats across the entire input tensor, with each pooling region independently choosing the maximum value. Hence, a 2×2 feature map is generated. This operation helps preserve the important features while reducing the size of the feature map, making the model more efficient.

2.1.4 Fully Connected Layer

After feature extraction, Convolutional Neural Networks (CNNs) use a fully connected layer, where neurons (the basic units of the network) link the input and output layers through one or more hidden layers. The connections between neurons are represented by weights, which are initially set to random values. Additionally, biases are extra parameters added to each neuron to help the network make better predictions. This layer helps the network combine the features it learned earlier and use them to make final decisions. A typical fully connected layer is illustrated in Fig. 6.

,

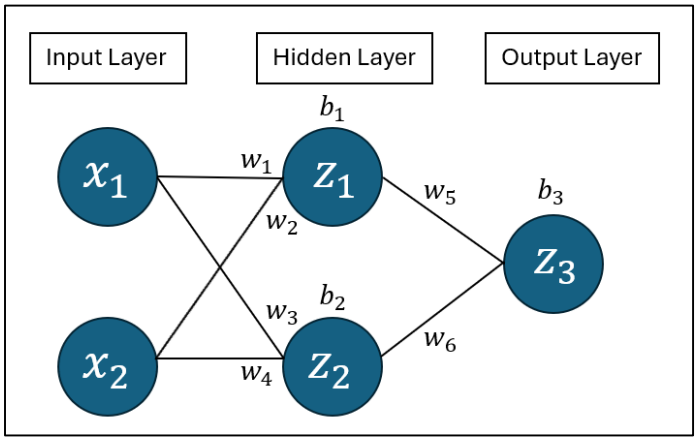


Fig. 6 A fully connected layer

From Fig. 6, let a_1 be the neuron value of z_1. The value of a_1 is calculated as:

$$a_1 = x_1 w_1 + x_2 w_2 + b_1$$

z = The output vector of the layer,
w = The weight matrix (each row corresponds to the weights of a neuron),
x = The input vector,
b = The bias vector.

For n sized inputs, the general form of a_i is as follows:

$$a_i = \sum_{j=1}^{n} x_j w_j + b_i, \qquad i = 1,2, \ldots \tag{3}$$

From (3), it can be concluded that all the neuron values are simple linear functions. They cannot learn or recognise complex mappings from the data because of their limited complexity. To address this limitation, the nonlinear activation function introduces nonlinear properties [11]. Within the hidden ReLU function, $f(z)$ is used as the activation function. Thus, the new value of z_j is:

$$z_j = f\left(\sum_{i=1}^{n} x_i w_i + b_j\right). \tag{4}$$

For the output layer, the Softmax function is used as the activation function. The function can convert raw output scores, also known as logits, into probabilities that sum up to one [12]. The Softmax function is defined as follows:

$$f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}}, \qquad i = 1,2, \ldots, K \; where \; x \in R. \tag{5}$$

2.2 Preparation of Dataset

The Facial Expression Recognition 2013 Dataset (FER2013) dataset is divided into training and test sets, with 28,709 images used for training and 7,178 images used for testing, resulting in an approximately 80-20 train-test split. This split provides a good balance between effectively training the model and ensuring sufficient data for reliable performance evaluation.

Dataset preparation for training also involves a preprocessing technique called normalisation. During normalisation, each image in the dataset is scaled to the range [0, 1]. This step involves dividing each pixel value by 255 (the maximum pixel value in an 8-bit image). Normalising the pixel value stabilises and speeds up the training process, enabling the neural network to converge faster during optimisation.

Furthermore, data augmentation techniques are used to introduce more variability into the training dataset, which helps improve the model's accuracy and generalisation. These techniques include random transformations of the images, such as rotation by $\pm 10^{\circ}$, vertical shifting by $\pm 10\%$, horizontal shifting by $\pm 10\%$, zooming by $\pm 10\%$ and horizontal flipping. As a result, the model learns more robust features, enhancing its generalisation ability and increasing its accuracy.

2.3 Implementation of the Custom CNN Model

In this research, the custom Convolutional Neural Network (CNN) model developed using TensorFlow in Python consists of a total of 13 layers, including 4 convolutional layers, 3 max-pooling layers, 3 dropout layers, 1 flatten layer and 2 dense layers. Table 1 below shows the detailed architecture of the model:

Table 1 Custom CNN model architecture

Name	Layer Type	Input Shape	Output Shape
Conv2d_1	Convolutional	(48, 48, 1)	(46, 46, 32)
Conv2d_2	Convolutional	(46, 46, 32)	(44, 44, 64)
Maxpooling2d_1	Pooling	(44, 44, 64)	(22, 22, 64)
Dropout_1	Dropout	(22, 22, 64)	(22, 22, 64)
Conv2d_3	Convolutional	(22, 22, 64)	(20, 20, 128)
Maxpooling2d_2	Max Pooling	(20, 20, 128)	(10, 10, 128)

Conv2d_4	Convolutional	(10, 10, 128)	(8, 8, 128)
Maxpooling2d_3	Max Pooling	(8, 8, 128)	(4, 4, 128)
Dropout_2	Dropout	(4, 4, 128)	(4, 4, 128)
Flatten_1	Flatten	(4, 4, 128)	(2048)
Dense_1	Dense	(2048)	(1024)
Dropout_3	Dropout	(1024)	(1024)
Dense_2	Dense	(1024)	(7)

Before training, the custom CNN model is compiled using the Adam optimiser with a learning rate of 0.0001. The model minimises the categorical cross-entropy loss function, which is suitable for multi-class classification, and uses accuracy as the evaluation metric. The accuracy of the model in this research is calculated based on the model's predictions on the test set of the FER2013 dataset. The formula for accuracy is as follows:

$$\text{Accuracy} = \frac{\text{Number of Correct Prediction}}{\text{Total Predictions}} \tag{6}$$

The model's efficiency is evaluated based on its training time. Two types of hyperparameters, epochs and batch size, are tested to determine how they affect the model's training time and accuracy. The values used for epochs are 10, 50, 100, 150, 200, 250, 300, 350, 400, 450, and 500, while the values used for batch size are 16, 32, 64 and 128. The best model is selected based on the highest accuracy, then followed by the shortest training time. The overview of the training and evaluation steps for the custom CNN model is shown in Fig. 7.

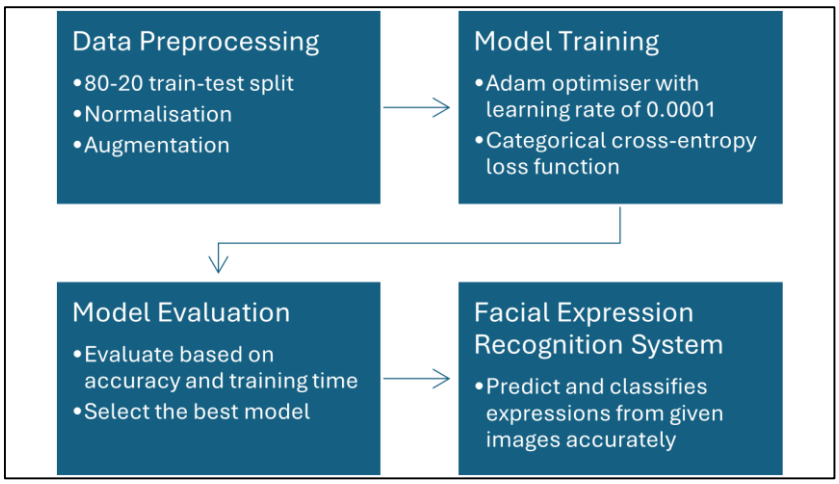


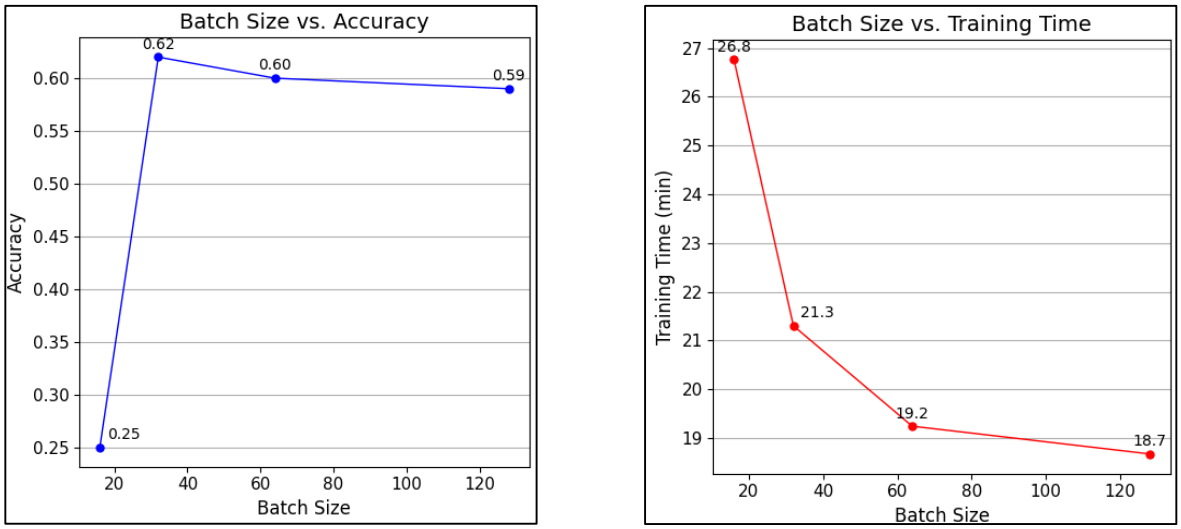
Fig. 7 Development of a facial expression recognition system

3. Results and Discussion

This section examines the effects of epochs (10, 50, 100, 150, 200, 250, 300, 350, 400, 450, 500) and batch sizes (16, 32, 64, 128) on the accuracy and training time of the custom convolutional neural network (CNN) model. The best model is determined based on the highest accuracy achieved, followed by the shortest training time. Then, the model's strengths and weaknesses are discussed using a confusion matrix. Additionally, the CNN model is integrated into a graphic user interface (GUI) for the facial expression recognition (FER) application.

3.1 Effect of Batch Size on CNN Model Performance

Fig. 8 shows the relationship between batch size and the Convolutional Neural Network (CNN) model's performance. In this experiment, the learning rate (0.0001) and number of epochs (50) were kept constant, while the batch size varied from 16 to 128. The choice of 50 epochs is to ensure the model has enough time to learn without taking too long to train.



(a) (b)

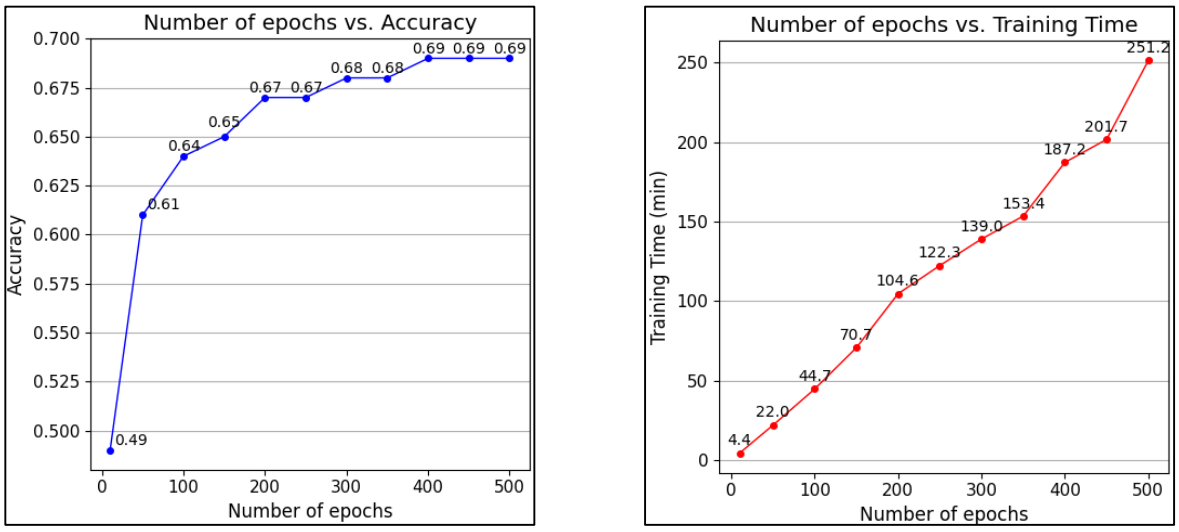
Fig. 8 (a) Relationship between batch size and accuracy; (b) Relationship between batch size and training time

From Fig. 8 (a), it can be observed that the model's accuracy increased from 25% at a batch size of 16 to 62% at a batch size of 32, showing a 37% improvement. However, it then decreased from 62% at a batch size of 32 to 59% at a batch size of 128, a 3% drop. This indicates that the optimal batch size for the custom CNN model is 32.

From Fig. 8 (b), it can be observed that the model's training time decreases with larger batch sizes. The training time of the model decreased from 26.8 minutes at a batch size of 16 to 18.7 minutes at a batch size of 128. This suggests that while larger batch sizes speed up training by processing more data simultaneously, they might not always lead to better model performance in terms of accuracy.

3.2 Effect of Number of Epochs on CNN Model Performance

Fig. 9 shows the relationship between the number of training epochs and the Convolutional Neural Network (CNN) model's performance. In this experiment, the learning rate (0.0001) and batch size (32) were kept constant, while the number of epochs varied from 10 to 500.



(a) (b)

Fig. 9 (a) Relationship between epochs and accuracy; (b) Relationship between epochs and training time

From Fig. 9 (a), it can be observed that as the number of epochs increases, the accuracy of the model also increases. From 10 to 400 epochs, the model's accuracy increased from 49% to 69%, observing a 20% increase. The model's accuracy reached a plateau at 400 epochs, as further training up to 500 epochs resulted in no improvement, with the accuracy remaining at 69%.

From Fig. 9 (b), it can be observed that as the number of epochs increases, the training time of the model also increases. From 10 to 400 epochs, the model's training time increased significantly from 4.4 to 251.2 minutes. This trend shows that the computational efficiency of the model decreases as the number of epochs increases.

After testing with different numbers of epochs and batch sizes, the optimal hyperparameters for the best model were determined. Using 400 epochs, a batch size of 32, and a learning rate of 0.0001, the best model achieved an accuracy of 69% with a training time of 187.2 minutes.

3.3 CNN Model Evaluation Using a Confusion Matrix

The confusion matrix in Fig. 10 was obtained by comparing the predicted class labels with the actual class labels from the test set. Each row in the 7 × 7 matrix represents one of the seven actual classes, while each column represents the predicted classes. The diagonal elements show the number of correct predictions (true positives), where the predicted class matches the actual class, and the off-diagonal elements represent misclassifications.

From Fig. 10, it is evident that the model performed well in predicting the "Happy" class, with an 88.3% true positive rate. This shows that the model correctly classified 88.3% of all actual "Happy" instances. The "Surprise" class comes in second, with an 83.4% true positive rate, followed by "Neutral" with a 72.6% true positive rate. Conversely, the "Sad" class has a low true positive rate of only 54.4%, with a notable 20.8% false positive rate on the "Neutral" class. This shows that the model frequently misclassifies the "Sad" class as the "Neutral" class. Lastly, the "Fear" class showed the poorest performance, with the lowest true positive rate of 41.7%. Out of all the false positives for this class, the majority are misclassified as "Sad" (19.3%) and "Neutral" (14.2%). This indicates that the model often misclassifies the "Fear" class as either "Sad" or "Neutral".

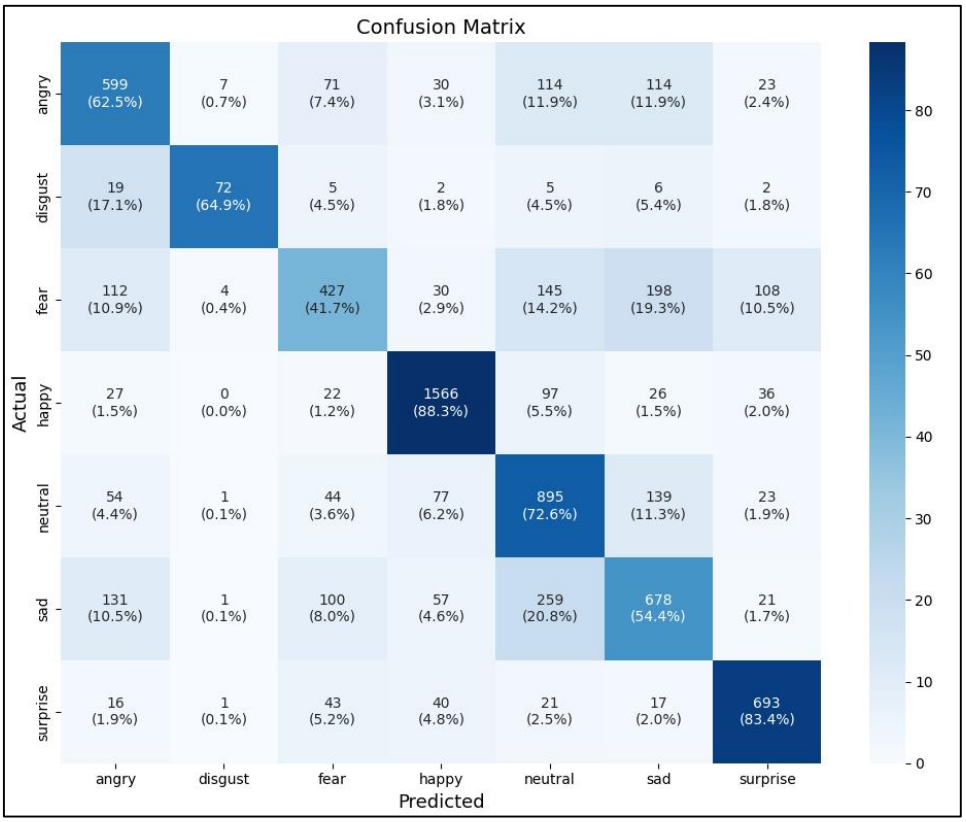
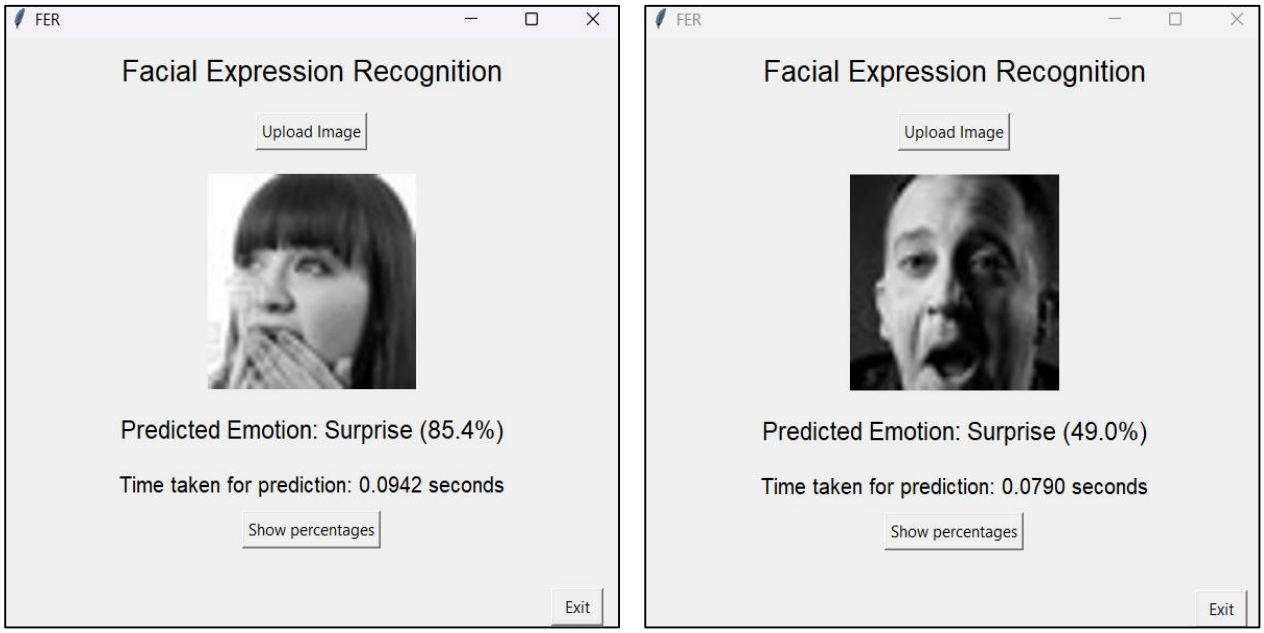


Fig. 10 Confusion matrix for the CNN model

3.4 Facial Expression Recognition (FER) Application

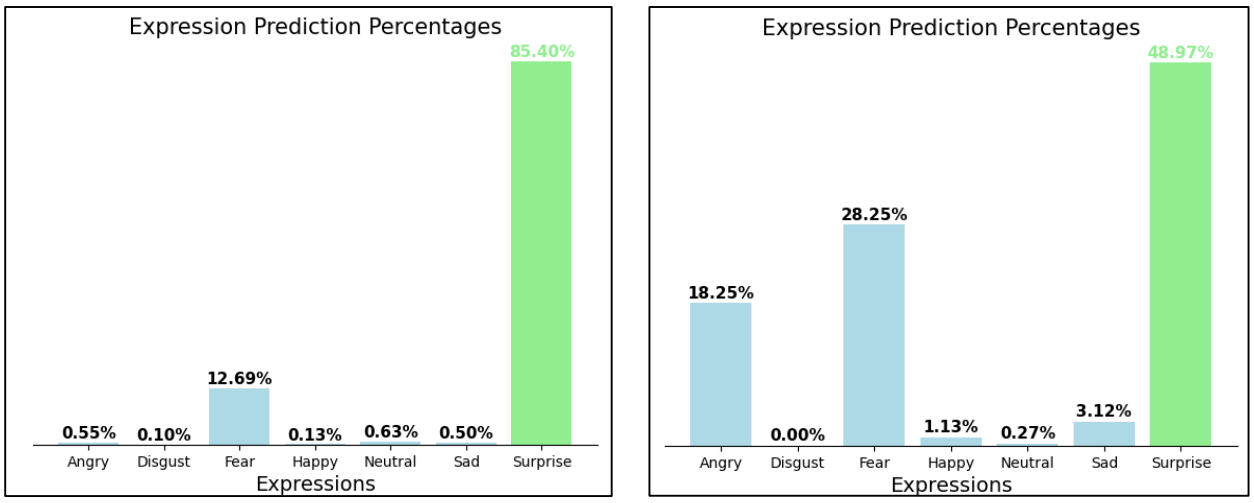
With the best pre-trained model integrated, a graphical user interface (GUI) was created for the facial expression recognition (FER) system to improve user experience. This was done using the tkinter library in Python. The main page of the application is shown in Fig. 11. On this page, the user can press the "Upload Image" button to upload an image for testing. The model will then predict the facial expression of the input image and display the predicted expression (as shown in Fig. 11 (a), where the predicted expression is "Surprise"). The application also displays the time taken by the CNN model to make the prediction (0.0942 seconds). The exit button allows the user to exit the program and close the application window.



(a) (b)

Fig. 11 (a) The main page of FER application with Image 1; (b) The main page of FER application with Image 2

Additionally, the user can press the "Show Percentage" button to view a bar chart displaying the percentage breakdown of all seven expressions in a separate window. As shown in Fig. 12 (a), the expression with the highest percentage (Surprise: 85.4%) is highlighted in green, indicating the predicted expression. This feature is useful for providing a detailed breakdown of the predicted probabilities for each expression in the uploaded image. For example, in Fig. 12 (b), when the predicted expression is Surprise with a lower percentage of 48.97%, the breakdown allows the user to assess the margin between the predicted expression and the second highest, Fear (28.25%). This comparison can help the user understand the confidence level of the model's prediction.



(a) (b)

Fig. 12 (a) Percentage breakdown of predicted expressions in the FER application for Image 1; (b) Percentage breakdown of predicted expressions in the FER application for Image 2

4. Conclusion

This research investigated the effects of epochs and batch size on the performance of a custom convolutional neural network (CNN) model for facial expression recognition (FER). The optimal value for batch size was found to be 32, as it achieved 62% accuracy when trained with 50 epochs. This suggests that if the batch size is smaller or larger than 32, the custom CNN model may underperform. It was found that the CNN model's accuracy reached a plateau at 69% after 400 training epochs. This suggests that the number of epochs has a positive correlation with the model's accuracy, but only up to a certain point, after which the accuracy no longer improves. With 400

epochs, a batch size of 32, and a learning rate of 0.0001, the best CNN model achieved a maximum accuracy of 69%. The confusion matrix analysis showed that the custom CNN model is the best at classifying the "Happy" expression with an 88.3% true positive rate but struggles the most with classifying the "Fear" expression with a low true positive rate of only 41.7%. Using Tkinter, an application was created for the FER system, which allows users to upload facial images, display the predicted expression, and view a detailed breakdown of the prediction probabilities of all seven expressions. Since the CNN model struggles with classifying the "Fear" expression, it is recommended to increase the dataset for this class so the model can better learn its features and improve its accuracy.

Acknowledgement

The authors would like to thank the Faculty of Applied Sciences and Technology, Universiti Tun Hussein Onn Malaysia, for its support.

Conflict of Interest

The authors declare that there is no conflict of interest regarding the publication of the paper.

Author Contribution

The authors confirm contribution to the paper as follows: ***study conception and design:*** *Pang Kou Yi, Lee Siaw Chong;* ***data collection:*** *Pang Kou Yi;* ***analysis and interpretation of results:*** *Pang Kou Yi, Lee Siaw Chong;* ***draft manuscript preparation:*** *Pang Kou Yi, Lee Siaw Chong. All authors reviewed the results and approved the final version of the manuscript.*

References

[1] S. Z. Li and A. K. Jain, *Handbook of Face Recognition*, 2nd ed. New York: Springer, 2011.
[2] A. T. Lopes, E. d. Aguiar, A. F. Souza, and T. Oliveira-Santos, "Facial expression recognition with convolutional neural networks: Coping with few data and the training sample order," *Pattern Recognition*, vol. 61, pp. 610–628, 2017.
[3] K. I. Ahamed and D. S. Akthar, "A study on neural network architectures," *Computer Engineering and Intelligent Systems*, vol. 7, no. 9, pp. 1–7, 2016.
[4] B. K. Kumar, K. Swaroopa, and T. R. Balaga, "Facial emotion recognition and detection using CNN," *Turkish Journal of Computer and Mathematics Education*, vol. 12, no. 14, pp. 5960–5968, 2021.
[5] F. K. Zaidan, A. J. Yousif, and G. J. Ahmed, "Ear recognition system based on CLAHE and convolutional neural network," *International Journal of Computer Science & Wireless Security*, vol. 9, no. 2, pp. 1–6, 2022.
[6] P. Goldsborough, "A tour of TensorFlow," *arXiv*, abs/1610.01178, 2016.
[7] I. J. Goodfellow et al., "Challenges in representation learning: A report on three machine learning contests," *Neural Networks*, vol. 64, pp. 59–63, 2015, doi: 10.1016/j.neunet.2014.09.005
[8] K. K. Gaurav Meena, "Sentiment analysis from images using VGG19-based transfer learning approach," *Procedia Computer Science*, vol. 204, pp. 4111–418, 2022.
[9] R. Yamashita, M. Nishio, R. K. Do, and K. Togashi, "Convolutional neural networks: An overview and application in radiology," *Insights into Imaging*, vol. 9, no. 4, pp. 611–629, 2018, doi: 10.1007/s13244-018-0639-9
[10] H. Gholamalinejad and H. Khosvari, "Pooling methods in deep neural networks: A review," *arXiv*, abs/2009.07485, 2020.
[11] S. Sharma and A. Athaiya, "Activation functions in neural networks," *International Journal of Engineering Applied Sciences and Technology*, vol. 4, no. 12, pp. 310–316, 2020, doi: 10.33564/ijeast.2020.v04i12.054
[12] Y. Zhou et al., "MPCE: A maximum probability based cross entropy loss function for neural network classification," *IEEE Access*, vol. 7, pp. 146331–146341, 2019, doi: 10.1109/ACCESS.2019.2946264