**CCT College Dublin**

|  |  |
| --- | --- |
| **Module Title:** | *Advanced Data Analysis*  *Big Data Storage Processing* |
| **Assessment Title:** | *Integrated CA1 Semester 2 MSc in Data Analytics* |
| **Lecturer Name:** | *David McQuaid*  *Muhammad Iqbal* |
| **Student Full Name:** | *Mahmut Zahid Malkoc* |
| **Student Number:** | *2023034* |
| **Assessment Due Date:** | *06/09/2023* |
| **Date of Submission:** | *06/10/2023* |

**Declaration**

|  |
| --- |
| By submitting this assessment, I confirm that I have read the CCT policy on Academic Misconduct and understand the implications of submitting work that is not my own or does not appropriately reference material taken from a third party or other source. I declare it to be my own work and that all material from third parties has been appropriately referenced. I further confirm that this work has not previously been submitted for assessment by myself or someone else in CCT College Dublin or any other higher education institution. |

**Face Emotion Recognition with CNN (Convolutional Neural Networks)**

Mahmut Zahid Malkoc   
*CCT College Dublin*Dublin, Ireland  
2023034@student.cct.ie

***Abstract—Face emotion recognition, one of the deep learning models, CNN (Convolutional Neural Networks), has been employed in conjunction with big data processing technologies in this study. This research investigates the utilization of CNN and deep learning methods in emotion analysis and recognition applications using big data processing tools such as Hadoop Distributed File System (HDFS) and Apache Spark. HDFS and Apache Spark offer ideal platforms for processing large-scale datasets and performing parallel computations. CNN provides high precision and accuracy in facial emotion recognition tasks. This study explores an effective approach to emotion analysis using deep learning models by processing facial expression data from large data sources. In conclusion, this study aims to enhance efficiency and accuracy in the field of facial emotion recognition by combining big data and deep learning technologies.***

***Keywords***—***Face Emotion Recognition, Hadoop Distributed File System, Apache Spark, Convolutional Neural Networks (CNN), Deep Learning.***

1. INTRODUCTION

The present era witnesses swift advancements in computer science and artificial intelligence, which have a significant impact on crucial domains such as facial emotion recognition and face recognition. The latter is an extensively used technology for identification and authentication of individuals. In contrast, the former seeks to comprehend the emotional state of people by scrutinizing their facial expressions. These two technologies are widely utilized in multiple sectors such as healthcare, security, entertainment, education, among others.

The main objective of this article is to emphasize the significance of deep learning-based methodologies in the areas of facial emotion recognition and face recognition. Deep learning is an efficacious artificial intelligence technique that involves the usage of multi-layered neural networks for performing intricate pattern recognition and data analysis tasks. This study will primarily concentrate on the influence of deep learning algorithms on the task of facial emotion recognition and face recognition.

Identity recognition is accomplished by utilizing facial features captured through face recognition technology. The field of face recognition has seen significant research in both academia and industry. Success in face recognition applications has been demonstrated by researchers such as Akhand et al. (2021), Benkaddour et al. (2021), and Ko (2018) through their use of deep convolutional neural networks (CNNs). The potential of face recognition technology has been acknowledged in various fields, including security systems, automation applications, and healthcare, where patient identification is of utmost importance.

Facial emotion recognition is a field of study that seeks to identify various emotional states, including fear, anger, sadness, happiness, and more, by analyzing facial expressions. The research in this field is fascinating. The works of Graves et al. (2008), Mollahosseini et al. (2016), and Taghi Zadeh et al. (2019) have shown that deep neural networks can substantially improve the accuracy and effectiveness of facial emotion recognition. This presents substantial potential, especially in applications that require emotional analysis, such as psychology research, marketing strategies, and human-machine interaction.

The following article presents a comprehensive analysis on notable research carried out in the areas of facial recognition and identifying emotions through facial expressions. Furthermore, it will delve into the influence of deep learning methods on these applications and how they can be utilized in everyday life and industrial settings. Additionally, it will discuss possible advancements that may arise in the future.

The main objective of this article is to illuminate the progress that has been made in the area of face recognition and facial emotion recognition through the use of deep learning techniques, as well as to suggest how these technologies may continue to gain traction and become more broadly applied in the coming years. Face recognition and the recognition of emotions through facial expressions are both integral topics in the field of artificial intelligence, and they are expected to receive even more attention and advancement in the future. This article will provide valuable insights into the current status of these domains as well as their future potential.

1. CONVOLUTIONAL NEURAL NETWORK

At the moment, neural networks are among the most significant algorithms for pattern recognition and machine learning. With Convolutional Neural Networks (CNNs) being a significant type of artificial neural network frequently used for image analysis, it has been amply demonstrated over time that neural networks outperform other algorithms in terms of sensitivity and speed, particularly in common application areas like image classification, segmentation, or object detection.

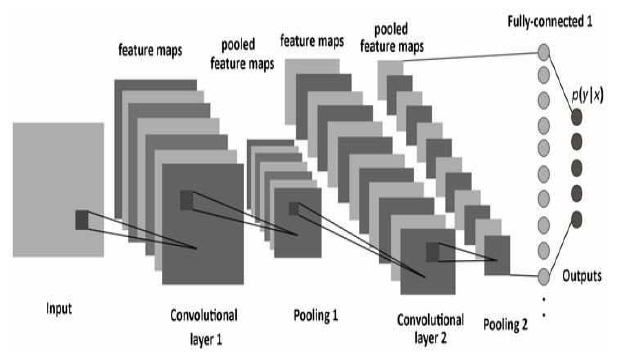
The feature extraction step is the first of two key stages in CNNs. In order to extract features, the network goes through a sequence of convolutional layers and pooling processes. On top of this feature set, fully connected layers' classification component functions as a classifier, assigning a probability to how accurately the algorithm guesses the item in the image, as illustrated in Figure 1.

Fig. 1. Convolutional Neural Network Architecture

CNNs consist of four types of layers: convolutional, pooling, ReLU (Rectified Linear Unit), and fully connected (FC) layers. An input image is processed through a series of filters in the convolutional layer, which produces a feature map. This output is then sent to a pooling layer that reduces the size of the feature map, helping to focus on the most relevant results and reduce computation time. Depending on the network, convolutional layers and pooling stages are repeated multiple times, and during this process, the output feature map of each operation is sent to a series of FC layers. These FC layers combine the maps and compare the likelihood of each operation harmonizing with the others before calculating the best classification. The Rectified Linear Unit (ReLU) layer is used to add non-linearity on top of the convolution, essentially performing element-wise multiplication and addition, introducing non-linearity to the network.

1. RESEARCH AND REVIEW OF DATASET

In this project, a CNN model was used to process multiple datasets for Image Processing. The Gender Recognition dataset was initially chosen, and it was uploaded to the Hadoop Distributed File System as a 450 MB file. Subsequently, it was observed that the pixel dimensions of each image in this dataset were different, and the pixels were quite large. Therefore, it was decided to modify this dataset, and the project continued by switching to the Face Emotion Recognition dataset for further processing. This chosen dataset is approximately 300 MB. This dataset consists of 7 different emotions: angry, disgust, fear, happy, sad, surprise, and neutral, comprising 7 distinct classes. Additionally, this dataset contains approximately 35,000 image pixels. To facilitate the use of the dataset in machine learning or deep learning models, it has been divided into three main categories: Training, Public Test, and Private Test. These divisions are also assigned to a column called "Usage." As can be seen, this dataset consists of three primary columns: emotion, pixels, and usage. This project involves the implementation of a deep learning model. Furthermore, this project necessitates big data storage and processing operations. As it can be seen, dealing with a large amount of data, the data will be stored in the Hadoop Distributed File System. Additionally, the processing of this large-sized dataset will be carried out using Apache Spark.

1. PREPROCESSING OF DATASET

Overall, in this project, big data storage processing and deep learning with Convolutional Neural Networks (CNNs) will be conducted. Therefore, the initial step involves configuring the Hadoop Distributed File System, which serves as the data storage system for the project. Once these configurations are completed, the dataset is transferred to the Hadoop Distributed File System. After importing the necessary libraries, a Spark session is initiated to read data from the Hadoop Distributed File System using Spark.

Firstly, the content of this dataset has been inspected. Due to the size of the dataset, not all of it could be displayed; therefore, several processing operations were carried out on the dataset using Spark. This allowed for determining the number of different emotions present in the dataset. Initially, this dataset has been divided into seven classes of emotions: angry, disgust, fear, happy, sad, surprise, and neutral. It is evident that all these emotions have been assigned numerical labels. Additionally, all the images in the dataset are represented as pixels. The data consists of thousands of numbers arranged in a string format for each pixel in each image, ranging from 0 to 255. The most crucial task here is to convert this string structure in each image into integers. The primary purpose of this operation is to reduce the numbers to integers so that they can be taught to the machine. After this reduction to integers, the dataset is then normalized. This way, we obtain data that can be used for training and testing a deep learning model. Normalization typically rescales pixel values to be within the range of [0, 1] or [-1, 1]. It helps achieve better results when training with deep learning. Pixel values generally perform better within these ranges. As seen in this project, pixel values have been normalized to the [0, 1] range.

The "reshaped\_pixels" column is necessary to transform the data into a more suitable format before using it as input for a machine learning model. Typically, many deep learning models expect input data to be in the form of a matrix or tensor rather than a single vector, especially when dealing with images. Therefore, the "reshaped\_pixels" column is used to convert each image into this appropriate format. For example, consider an image where each pixel contains RGB color components, usually represented as a vector of three separate numbers. However, many deep learning models expect the image to be in the form of a matrix, where each pixel contains RGB components within a matrix. The "reshaped\_pixels" column contains the structure used to transform each image into this matrix format. This reshaping process is important for the model to learn better and achieve more effective results because it adapts the data into a format that the model can understand. This is why the reshaping process has been applied to this dataset.

One-Hot Encoding is a data transformation method used to convert categorical data into a numerical format. This transformation is necessary, especially in models that work with numerical data, such as deep learning. According to this assessment, labels such as 'Angry', 'Disgust', 'Fear', 'Happy', 'Sad', 'Surprise', and 'Neutral' cannot be directly given to a deep learning model without being converted into a numerical format. Therefore, one-hot encoding is used to convert categorical data into an appropriate numerical format. Furthermore, one-hot encoding ensures that all classes have equal weight. In other words, it indicates that each class is equally important for the model. This balances the learning process of the model and ensures that each class is represented accurately.

One-hot encoding is often used as part of the output layer in deep learning models, especially in classification problems and multi-class classification tasks. For instance, if you want to predict which emotion or category a piece of text belongs to, you can use one-hot encoding by creating a column for each emotion or category. The output layer of a deep learning model consists of one neuron for each class or category. Each neuron represents a class, and predictions are made based on the activations of these neurons. If the prediction for a class is close to 1, it means that the input belongs to that class, while if it is close to 0, it indicates that it does not belong. One-hot encoding enables the model to accurately learn the relationships between classes, making it a common data transformation for classification and categorization problems.

1. DATA SPLITTING AND PREPARATION OF DATASET FOR CNN MODEL

Data Splitting, an important data preprocessing step, is used to divide data into different subsets for the purpose of training and evaluating machine learning and deep learning models. This code snippet typically involves three main sections: creating training, validation, and test datasets.

Firstly, the dataset is processed by selecting a large portion of it to be used as training data. This step is performed to utilize the majority of the dataset for the model's learning process. In this case, approximately 80% of the data is selected as training data using the df.sample(False, 0.8, seed=123) code, and the remaining dataset is determined.

Subsequently, the remaining dataset is created, which is the portion of data that remains after extracting the training data. This step is a fundamental part of data segmentation, enabling the model to be tested on other datasets in addition to the training data.

Once the training data is identified, the remaining dataset is divided into validation and test datasets. This stage creates datasets referred to as validation\_data and test\_data. Initially, half of the remaining data is selected as validation data using the remaining\_data.sample(False, 0.5, seed=123) code, while the other half is designated as test data. Validation data is used to evaluate the model's performance during the training process and to fine-tune hyperparameters. Hyperparameters are configuration settings that affect the model's structure and performance, and validation data helps in finding the best hyperparameter settings for the model.

On the other hand, test data is employed to objectively assess the model's performance after training. This dataset is used to measure how the model responds to real-world data and to evaluate its generalization ability. Performance metrics such as accuracy, precision, and others are calculated using the test data.

In conclusion, this data splitting process enables the model to conduct training, tuning, and performance evaluation more robustly and reliably. Consequently, it mitigates the risk of overfitting and yields more dependable results.

After performing the data splitting process, there are two methods to define and create train\_X and train\_y. The first method involves selecting the relevant columns using PySpark's DataFrame API, followed by collecting these columns into lists. Consequently, Python lists named train\_X and train\_y are created, with each feature or label being an element in the list. In the second method, PySpark DataFrame API is used again to collect all the data into a list. However, this time, the list is converted into a NumPy array. Each column is stored within a NumPy array.

When examining these two methods, the second method utilizes NumPy's fast and optimized operations instead of PySpark DataFrame operations. Therefore, it can be more efficient, especially when working with large datasets. Particularly in computation-intensive applications like deep learning, NumPy's speed can significantly accelerate processes. Additionally, the second method stores each feature and label separately within arrays, which is a more suitable data structure for processing data and is commonly used for feeding deep learning models.

Furthermore, NumPy arrays are independent of PySpark DataFrame operations since NumPy is a separate Python package. This independence provides greater flexibility when working with data and makes it easier to use the same data in other analyses or modeling processes.

In conclusion, especially in the context of large datasets and deep learning projects, the second method is considered more efficient, faster, and flexible. However, both methods can be used depending on data sizes and specific project requirements.

The reshaping of the split data is performed to restructure image data within a dataset. Each set of data contains images that are transformed to have dimensions of 48 pixels in width, 48 pixels in height, and a single channel (grayscale representation). This is done to prepare input data that adheres to specific requirements, particularly for deep learning models. By doing so, it ensures that the model can work with the correct data structure during training, testing, and validation processes, ultimately leading to improved results.

1. CREATING AND CONFIGURING OF CNN MODEL

First of all, initial parameters used at different parts of the CNN model are assigned. These are configurations that define the starting points of a machine learning or deep learning model. Number of classes (num\_classes) specifies the number of classes in the dataset; in this project, it is necessary to detect 7 different emotions or classes. Width and Height set the dimensions of the images to 48x48 pixels. Number of epochs (num\_epochs) determines how many times the model should see the entire dataset, and the number of epochs has been chosen based on trial and error. Batch size (batch\_size) defines the size of the data batch to be processed in each training step and is set to 64. Finally, number of features (num\_features) represents the number of internal features of the model, reflecting its complexity. These parameters are carefully chosen based on the project's requirements, dataset characteristics, and experimentation processes to contribute to the successful training of the model.

This CNN (Convolutional Neural Network) model consists of three different modules, and each module comprises a sequence of layers typically used to extract more complex and abstract features.

In the first module, a Conv2D (2D convolution) layer with `2\*2\*num\_features` filters is added. This layer helps extract low-level features (e.g., edges). Next, a BatchNormalization layer is added, which normalizes the weights, making the network train faster and more stable. ReLU (Rectified Linear Unit) is used as the activation function, which linearly activates the layer's outputs. Then, another Conv2D layer with the same filters is added, but this time "same" padding is used, ensuring that the output dimensions match the input dimensions. BatchNormalization and ReLU activation are used again. Finally, a MaxPooling2D layer is added, which reduces the size of the data and emphasizes important features.

Modules 2 and 3 have a similar structure. Both consist of Conv2D, BatchNormalization, ReLU, and MaxPooling2D layers to extract more complex features. However, the number of filters in the Conv2D layers is halved compared to the previous module (`2\*num\_features` and `num\_features`). These modules include layers that progressively extract more abstract and high-level features.

The reason for using three modules is to extract features at each level and help the model understand and learn both low-level and high-level features. This enables the model to successfully handle more complex tasks. Additionally, reducing the number of filters in Conv2D layers in each module allows the model to process previously extracted features more thoroughly and reduces the risk of overfitting.

The Flatten layer (model.add(Flatten())) in this context is used to convert the feature maps produced by the Convolutional Neural Network (CNN) at the end into a flat vector. In other words, it transforms 3D data, typically represented as feature maps, into a 1D vector. This is necessary because the subsequent fully connected layers (Dense layers) expect their input in the form of a flattened vector.

The Dense layers (Fully Connected layers) are conventional neural network layers. They receive the flattened feature vector as their input. Dense 1 (model.add(Dense(2\*2\*2\*num\_features))) layer takes in a large feature vector and consists of 2\*2\*2\*num\_features neurons. The purpose of this layer is to start learning higher-level features from the data. Dense 2 (model.add(Dense(2\*2\*num\_features))) layer takes the output from the previous layer and contains 2\*2\*num\_features neurons. It is used to represent features with even greater complexity. Dense 3 (model.add(Dense(2\*num\_features))) layer takes the output from the previous layer and contains 2\*num\_features neurons. Its role is to learn even more abstract representations of the features.

The Output Layer (model.add(Dense(num\_classes, activation='softmax')) is used to make predictions with the model. This layer consists of neurons equal to the number of classes in the dataset (i.e., num\_classes), and it employs the softmax activation function to transform the outputs into a probability distribution for classification. In other words, it calculates the estimated probabilities for each class. These fully connected layers take in abstract representations of the features and ultimately produce the model's predictions. The number of neurons and complexity between layers can significantly impact the model's capabilities and is often adjusted experimentally.

The compilation of a deep learning model is done using model.compile(), which determines how the model will be configured for training. The loss function computes an error metric by comparing the model's predictions to the actual labels. In this code, the chosen loss function is "sparse categorical cross-entropy," which is suitable for multi-class classification problems. The model tries to minimize this loss function by learning to make accurate classifications. The optimization algorithm is used to update the model's parameters (weights) during training. In this code, the optimizer algorithm used is "Adam." Adam is a gradient-based optimization algorithm that takes important hyperparameters such as the learning rate. The learning rate is set to 0.001, and other Adam algorithm hyperparameters are pre-defined. The metrics parameter specifies the metrics to be tracked during training. In this code, "accuracy" is chosen as the metric, which means that the model's accuracy in making predictions will be calculated and monitored during the training process. "model.summary()" displays a summary of the model's architecture. It shows the model's layers, the number of parameters in each layer, and the total number of parameters. This information is valuable for understanding the structure and dimensions of the model.

A deep learning model has been created for a multi-class classification problem, and the necessary configurations for training have been set. The choice of the loss function, optimization algorithm, and tracked metrics should be selected and tuned based on the specific problem and dataset. These values can be adjusted through trial and error to improve the model's performance.

Data augmentation is a technique used to diversify training data by applying various transformations. Especially when you have a limited amount of data or your model tends to overfit, data augmentation becomes valuable. Data augmentation helps the model learn more generalizable features and perform better.

Rotation Range is randomly rotating images to create examples representing different viewpoints. Width Shift Range and Height Shift Range are shifting images horizontally and vertically to generate data points at new positions. Zoom Range is randomly zooming in or out representing different zoom levels. Horizontal Flip is flipping images horizontally creating mirrored examples. These techniques are particularly useful for classification problems, improving the model's ability to generalize.

The Early Stopping call back monitors the model's training process and automatically stops it if a certain condition is met (typically, an increase or no decrease in validation loss). This helps prevent overfitting and allows the model to stop at the point where it performs the best.

This call back watches for situations where the validation loss doesn't improve for a specified duration, typically controlled by the "patience" parameter. If the validation loss doesn't improve within the patience period, the training is automatically halted, and the model is saved at the point where it achieves the best performance. The Early Stopping call back prevents the model from training for unnecessarily long periods and helps it generalize better. "model.fit()" performs the training of the model. The "data\_generator.flow()" function prepares the training data using data augmentation techniques. steps\_per\_epoch specifies the number of training steps, and it trains for the specified number of epochs (num\_epochs). The callbacks parameter specifies the callbacks to be used during training, and here, the early stopping callback (es) is used. Finally, the training process is monitored using validation data.

This part uses data augmentation and early stopping techniques to make the model training more efficient and reduce the risk of overfitting. It helps in creating a deep learning model that can generalize better and achieve improved performance.

1. EXPERIMENTAL WORK AND RESULTS

The success of a machine learning project is evaluated through two stages: the training and validation process and the testing process.

In the first stage, the model is trained, and the validation process is undertaken. A large dataset is typically employed for the model to learn from the training data. During training, validation data is utilized to monitor the model's progress within each epoch and its performance on the training data. This allows the observation of how effectively the model is being trained and how the loss (error) and accuracy values on the training data are changing. Additionally, in this stage, the model's performance can be compared with validation data to detect issues such as overfitting or performance drops during training.

The second stage involves testing the model and evaluating its results. In this stage, a precise and objective measure of the model's performance is obtained. The model's performance on test data provides an official evaluation of the project's success metrics and outcomes. The performance metrics employed in this stage are determined based on the project's requirements. The model's performance on test data demonstrates how effectively it operates in real-world applications.

In conclusion, the assessment of the success of a machine learning project is facilitated by the training and validation process in conjunction with the testing process. The first stage allows the observation of the model's performance and progress on the training data, while the second stage enables the measurement of the project's official results and the evaluation of its success. Together, these two stages contribute to the successful completion of machine learning projects.

According to this assessment, the training has taken place over a total of 15 epochs, with each epoch being quite lengthy. For instance, the first epoch took 2,564 seconds. In total, approximately 28,728 seconds, or roughly 8 hours, were devoted to training the model. This is a typical scenario for training on a large dataset or with a complex model. At the end of each epoch, the model's loss on the training data is computed. It's evident that this loss decreased over time, indicating that the model performed better on the training data and experienced further learning. During training, the model's accuracy is also recorded. It increased over epochs, implying that the model started making better predictions on the training data. At the end of each epoch, the model's performance on a separate validation dataset is also evaluated. Considering the validation loss and validation accuracy, it provides insights into how well the model generalizes. In the initial epoch, the validation loss was high, and validation accuracy was low. However, in the final epoch, the validation loss decreased, and validation accuracy increased. This suggests that the model started to generalize better and performed well on test data.

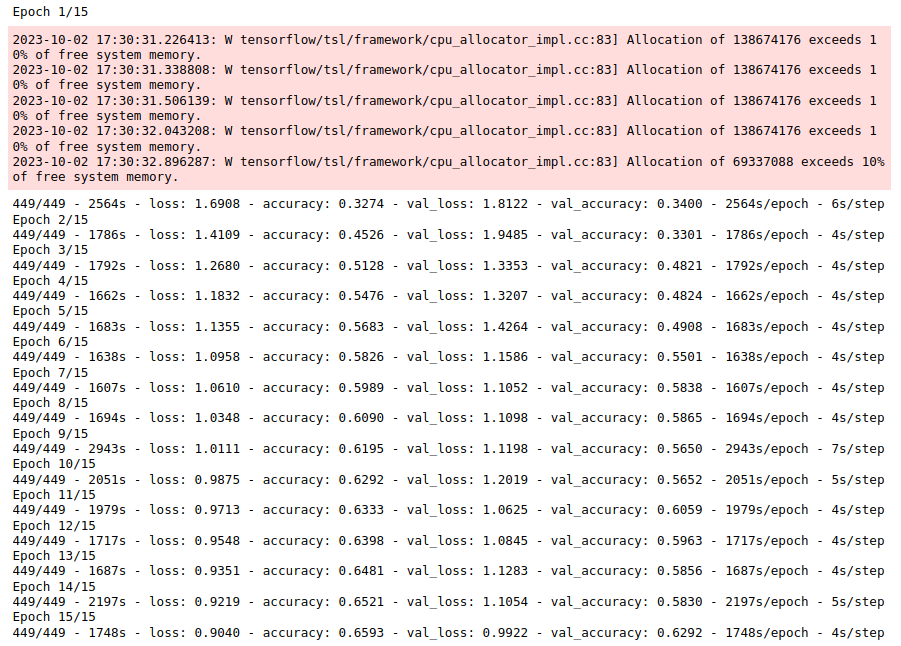


Fig. 2. The Results of This Applied CNN Model

In summary, these results indicate that the training process began with lower performance but improved as more training occurred. However, it's important to note that the performance of any model should be evaluated not only on the training data but also on separate test or validation data. In this context, the test values are acceptable for this model. When the train validation and test values are compared, it is seen that the model has the correct learning curve and the validation progresses with a positive result with the test values.

1. DISCUSSION

During the training process, the model achieves an accuracy of approximately 65.93% on the training dataset, indicating that it has learned the training data quite well. Monitoring the accuracy on the validation dataset is crucial for tracking the model's performance and detecting overfitting. The validation accuracy is approximately 62.92%. Compared to the training accuracy, it is evident that the model experiences some performance drop on the validation dataset. Finally, to evaluate the model on real-world data, the test dataset is used, and the test accuracy is found to be approximately 62.04%. The test accuracy indicates how well the model generalizes to new and unknown data. Considering the difference between the test accuracy and the validation accuracy, it can be inferred that the model may suffer from overfitting or may require adjustments to hyperparameters or training strategies to achieve better generalization at a certain stage.

To improve the model, the following strategies can be applied:

* More Training: Training the model for more epochs may help achieve better results at a certain stage.
* Model Complexity: Consider making the model more complex or deeper.
* Hyperparameter Tuning: Experiment with adjusting hyperparameters such as learning rate, batch size etc.
* Data Augmentation: Create more training data using data augmentation techniques.
* Different Models: Try different model architectures or utilize transfer learning methods.

In conclusion, the test accuracy may serve as an acceptable starting point in the current state, but it appears that the model has further improvement potential. The above-mentioned strategies and methods can be explored to achieve better performance and enhancements from the model.

REFERENCES

Akhand, M. a. H. et al., 2021. Facial Emotion Recognition Using Transfer Learning in the Deep CNN. *Electronics,* 10(9).

Benkaddour, M. K., Lahlali, S. & Trabelsi, M., 2021. *Human Age And Gender Classification using Convolutional Neural Network.* Boumerdes, Algeria, 2020 2nd International Workshop on Human-Centric Smart Environments for Health and Well-being (IHSH).

De Silva, L., Miyasato, T. & Nakatsu, R., 1997. *Facial emotion recognition using multi-modal information.* Singapore, IEEE.

Graves, A. et al., 2008. Facial Expression Recognition with Recurrent Neural Networks. *In Proceedings of the International Workshop on Cognition for Technical Systems.*

Jain, D. K., Shamsolmoali, P. & Sehdev, P., 2019. Extended deep neural network for facial emotion recognition. *Pattern Recognition Letters,* Volume 120, pp. 69-74.

Ko, B. C., 2018. A Brief Review of Facial Emotion Recognition Based on Visual Information. *Sensors,* 18(2).

Liew, S. S., Khalil-Hani, M., Ahmad Radzi, S. & Bakhteri, R., 2016. Gender classification: a convolutional neural network approach. *Turkish Journal of Electrical Engineering & Computer Sciences,* Volume 24, pp. 1248-1264.

Mellouk, W. & Handouzi, W., 2020. Facial emotion recognition using deep learning: review and insights. *Procedia Computer Science,* Volume 175, pp. 689-694.

Mollahosseini, A., Chan, D. & Mahoor, M. H., 2016. *Going deeper in facial expression recognition using deep neural networks.* Lake Placid, NY, USA, 2016 IEEE Winter Conference on Applications of Computer Vision (WACV).

Müller, T. R. et al., 2022. A survey on facial emotion recognition techniques: A state-of-the-art literature review. *Information Sciences,* Volume 582, pp. 593-617.

Taghi Zadeh, M. M., Imani, M. & B., M., 2019. *Fast Facial emotion recognition Using Convolutional Neural Networks and Gabor Filters.* Tehran, Iran, 5th Conference on Knowledge Based Engineering and Innovation (KBEI).

Yuda, R. P., Aroef, C., Rustam, Z. & Alatas, H., 2020. Gender Classification Based on Face Recognition using Convolutional Neural Networks (CNNs). *Journal of Physics: Conference Series,* 1490(1).