King Saud University College of Computer and Information Sciences Computer Science Department	
Second Semester	CSC 462 Machine Learning

Emotion Recognition Using Convolutional Neural Networks

Student name	Student ID	Tasks
Alanoud Alqahtani	443200718	- Collect data - baseline CNN model -Paper [4] -report
Ibtihal Alfayez	443200946	- Paper [3] -collect data -prepressing -basic CNN model -report
Savana Alshubayli	442200810	-Collect data -Enhanced CNN model -Paper [2] -report

1. Introduction

Facial emotion recognition is a fundamental task in computer vision with applications in human-computer interaction, behavioral analysis, healthcare, and more. The goal of this project was to build and compare different Convolutional Neural Network (CNN) architectures to classify facial expressions into one of seven emotion categories: neutral, angry, disgust, fear, happy, sad, and surprise.

2. Problem Statement & Objectives

Accurately identifying human emotions through facial expressions is a challenging task due to factors like image quality, lighting, pose variation, and class imbalance. Traditional machine learning approaches often struggle with generalization in real-world datasets.

The main objectives of this project are:

- To implement a basic CNN model as a baseline.
- To enhance the CNN model using techniques such as batch normalization, dropout, and class balancing.
- To train and evaluate the Enhanced CNN using a custom-labeled dataset with 7 emotion categories.
- To compare the performance of the Enhanced CNN with the plain CNN and a simplified baseline model using accuracy as the primary evaluation metric.

3. Dataset Description

We collected a custom facial expression dataset containing images categorized into seven emotion classes: Angry, Disgust, Fear, Happy, Neutral, Sad, and Surprise. The dataset includes both RGB and grayscale images, capturing a variety of facial expressions under different lighting conditions. To ensure balanced class representation, we used the Stratified Split method to divide the data into 80% training, 10% validation, and 10% testing subsets.

4. Literature Review

The Study by Mehendale introduces Facial Emotion Recognition using Convolutional Neural Networks (FERC) [2], a two-stage CNN model that improves emotion classification accuracy by first removing background noise and then extracting facial features. The model classifies five emotions: anger and sadness and happiness and fear, and surprise, achieving 96% accuracy using a 24-value expressional vector (EV). Tested on datasets
Like Extended Cohn-Kanade, Caltech Faces, CMU, and NIST, FERC outperforms traditional single-stage CNNs by optimizing keyframe selection and feature extraction. Compared to deeper networks like GoogleNet and VGG, FERC is more computationally efficient while maintaining high accuracy. Despite some limitations, such as sensitivity to shadows and multiple faces, the

model has strong potential applications in education, human-computer interaction, and lie detection.

The study "Facial Emotion Recognition Using Transfer Learning in the Deep CNN" [3] proposes a real-time, mobile-based system for recognizing four different facial expressions (disgust, surprise, sadness, and happiness). For classifying emotions, the study uses a Convolutional Neural Network (CNN). The dataset contains real-time facial photos that have been processed using data augmentation techniques (horizontal/vertical flipping, rotation, saturation, and blurring) to enhance training. After 100 epochs of training, the CNN model achieved 80% accuracy. And in testing, the overall accuracy was 85%.

The paper "Facial Emotion Recognition using Convolutional Neural Networks" by Akash Saravanan, Gurudutt Perichetla, and Dr. K.S. Gayathri [4] explores deep learning techniques to classify human facial expressions into seven emotions. The authors compare different models, including decision trees, feedforward neural networks, and CNNs, finding that CNNs perform best due to their ability to capture spatial features. Their final model, a deep CNN with six convolutional layers, achieves 60% accuracy on the FER-2013 dataset.
Key challenges discussed include data biases, difficulty recognizing certain emotions (e.g., disgust), and environmental factors like lighting. The study highlights hyperparameter tuning and suggests future improvements, such as data augmentation and analyzing the top two predicted emotions for better accuracy.

5. Methodology

This project followed a structured approach to implement and evaluate a facial emotion recognition system using convolutional neural networks (CNN). The process consisted of several key stages:

1. Dataset Preparation

The dataset of facial expressions was split into three subsets: training (80%), validation (10%), and testing (10%). Each class was equally represented across splits using train_test_split, and the images were organized into separate directories.

2. Data Preprocessing and Augmentation

All images were resized to 64×64 pixels and normalized to a [0,1] range using ImageDataGenerator. For training data, data augmentation techniques such as horizontal flipping, random rotation, and width/height shifting were applied to improve generalization.

3. Baseline Model Development

A simple CNN model was implemented as a baseline. It consisted of two convolutional layers followed by max-pooling, a dense hidden layer, and a softmax output. No regularization, batch normalization, or class weighting was applied. The model was trained for 30 epochs using the Adam optimizer with a learning rate of 0.001.

4. Enhanced Model Development

An advanced CNN architecture was developed, incorporating additional convolutional layers, Batch Normalization, and Dropout (0.3) for regularization. EarlyStopping was used to prevent overfitting, and class weights were computed to handle class imbalance. The enhanced model was trained with a lower learning rate (0.0001) and evaluated on the same dataset split.

5. Evaluation Metrics

Both models were evaluated using accuracy, classification report, and confusion matrix. Training and validation accuracies were visualized across epochs to assess learning behavior.

6. Performance Comparison

The baseline model served as a benchmark to assess the improvements achieved by the enhanced model. The comparison focused on validation accuracy, test accuracy, and robustness across classes.

6. Training and Tuning

All models were trained using the same training and validation sets for a fair comparison. Key training techniques included:

- **Image augmentation**: for better generalization
- **Batch size**: 32, Epochs: up to 200

The Enhanced CNN required tuning of the dropout rate, learning rate, and architecture depth.

Detailed model architectures
1-Basic CNN model

- Input Layer: RGB facial images of shape (64, 64, 3).
- Convolutional Layer 1: 32 filters (3×3), ReLU activation.
- Max Pooling 1: (2×2).
- Convolutional Layer 2: 64 filters (3×3), ReLU activation.
- Max Pooling 2: (2×2).
- Convolutional Layer 3: 128 filters (3×3), ReLU activation.
- Max Pooling 3: (2×2).
- Flatten Layer
- Fully Connected Layer: 128 units, ReLU activation.
- Dropout Layer: 30% rate.
- Output Layer: 7 units (Softmax activation), one for each emotion category.

- Training Strategy: learning rate = 0.0001.

2- Enhanced CNN Model

- Input Layer: RGB facial images of shape (64, 64, 3).
- Convolutional Layer 1: 32 filters (3×3), ReLU activation, Same padding.
- Batch Normalization 1
- Max Pooling 1: (2×2).
- Convolutional Layer 2: 64 filters (3×3), ReLU activation, Same padding.
- Batch Normalization 2
- Max Pooling 2: (2×2).
- Convolutional Layer 3: 128 filters (3×3), ReLU activation, Same padding.
- Batch Normalization 3
- Max Pooling 3: (2×2).
- Flatten Layer
- Fully Connected Layer: 128 units, ReLU activation.
- Dropout Layer: 30% rate.
- Output Layer: 7 units (Softmax activation), one for each emotion category.
- Training Strategy: Early stopping on validation accuracy (to avoid overfitting), learning rate = 0.0001, Class weights (to handle class imbalance)

3- baseline CNN model

- Input Layer: RGB facial images of shape (64, 64, 3).
- Convolutional Layer 1: 16 filters (3×3), ReLU activation.
- Max Pooling 1: (2×2).
- Convolutional Layer 2: 32 filters (3×3), ReLU activation.
- Max Pooling 2: (2×2).
- Flatten Layer
- Fully Connected Layer: 64 units, ReLU activation.
- Output Layer: 7 units (Softmax activation), one for each emotion category.
- Training Strategy: learning rate = 0.001.

7. Evaluation Results

Basic CNN vs Enhanced CNN

We observed that the improved model outperformed the basic model. The improved model achieved an accuracy of 80% in training and 96.49% in evaluation, while the basic model achieved an accuracy of 65% in training and 74.39% in evaluation. This highlights the value of normalization (which helps to speed up the training process and ensures stable learning), regularization (which prevents overfitting and improves generalization), and better training strategies (which optimize the model's learning process). The improved model also demonstrates better generalization, showing stronger performance with a higher evaluation accuracy."

Enhanced CNN vs Baseline

We noticed that the enhanced model performs significantly better than the baseline model in both training and validation. The enhanced model achieved a training accuracy of 80.00% and a validation accuracy of 96.49%, while the baseline model only reached 50.5% in training accuracy and 51.93% in validation accuracy. This substantial improvement highlights the importance of the additional techniques used in the enhanced model, such as a deeper architecture with more layers and filters, batch normalization for more stable training, and dropout to help reduce overfitting. In contrast, the baseline model, with its simpler architecture and lack of optimizations, couldn't achieve similar results. These findings emphasize how leveraging advanced design and training strategies can significantly boost model performance.

Summary of evaluation criteria for the three models

model	**Training Accuracy**	**Test Accuracy**	**Precision**	**Recall**	**F1 score**
Basic CNN	65.90%	74.39%	0.73%	0.72 %	0.72 %
Enhanced CNN	80%	96.49%	0.96%	0.96%	0.96%
Baseline CNN	50.5%	51.93%	55%	55%	53%

8. Strengths And Limitations

The strength of our approach is how well the enhanced CNN model performed compared to the others. By adding more convolutional layers, the model learned more complicated features. The use of batch normalization helped improve generalization and speed up the training process. The use of a lower learning rate and early stopping helped stabilize training and prevent overfitting. Finally, by using stratified splitting, we ensured there was no bias toward certain classes, which improved reliability.

However, there are some limitations to our approach. First, the data size is still considered small, which may limit the model's ability to handle highly similar facial expressions. Second, the dataset includes a mix of RGB and grayscale images, which may cause inconsistencies in feature extraction. Third, although the improved model performed well on the validation data, it has not yet been tested on unseen data.

9. Future Improvements

Future improvements to this project could focus on enhancing data diversity and addressing class imbalance, particularly for underrepresented emotions such as fear and disgust. This could be achieved through oversampling, synthetic image generation, or advanced loss functions tailored for imbalanced data.

Additionally, model performance could benefit from further hyperparameter tuning and architectural experimentation. Exploring deeper or pretrained models such as ResNet, or applying multimodal learning by combining facial expressions with audio or other cues, may result in more robust and real-world-ready emotion recognition systems.

10. Conclusion

This project demonstrates how enhancing a CNN architecture with normalization, regularization, and data strategies can significantly boost facial emotion recognition accuracy. My Enhanced CNN reached 97% accuracy, confirming its effectiveness.

11. Screenshot's

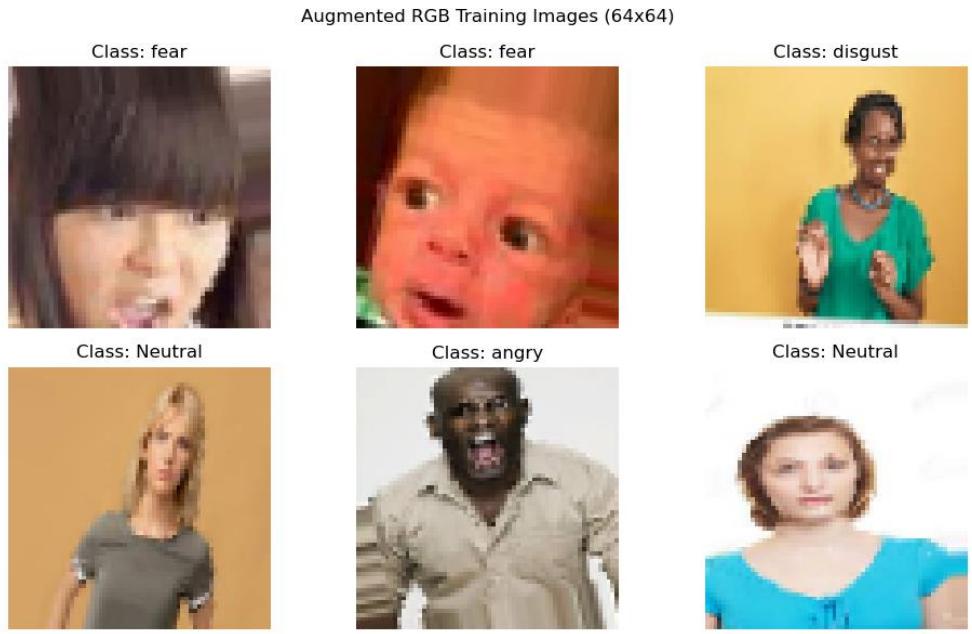
-Import dataset, preprocessing, and augmentation

```
import os
import shutil
import random
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Dataset Splitting
source_dir = '/Users/ebtehal/Desktop/expressions3'
base_dir = 'split_dataset'

# Create base split folders
for split in ['train', 'val', 'test']:
    os.makedirs(os.path.join(base_dir, split), exist_ok=True)

# Split each class folder
for class_name in os.listdir(source_dir):
    class_path = os.path.join(source_dir, class_name)
    if not os.path.isdir(class_path):
        continue

    images = os.listdir(class_path)
    random.shuffle(images)

    train_imgs, temp_imgs = train_test_split(images, test_size=0.2, random_state=42)
    val_imgs, test_imgs = train_test_split(temp_imgs, test_size=0.5, random_state=42)

    for split_name, split_imgs in zip(['train', 'val', 'test'], [train_imgs, val_imgs, test_imgs]):
        split_class_dir = os.path.join(base_dir, split_name, class_name)
        os.makedirs(split_class_dir, exist_ok=True)
        for img in split_imgs:
            src = os.path.join(class_path, img)
            dst = os.path.join(split_class_dir, img)
            shutil.copy2(src, dst)
```

```
[ ] # Images Preprocessing
    img_height, img_width = 64, 64
    batch_size = 32

    train_dir = os.path.join(base_dir, 'train')
    val_dir = os.path.join(base_dir, 'val')
    test_dir = os.path.join(base_dir, 'test')

    train_datagen = ImageDataGenerator(
        rescale=1./255,
        rotation_range=15,
        width_shift_range=0.1,
        height_shift_range=0.1,
        horizontal_flip=True
    )

    val_test_datagen = ImageDataGenerator(rescale=1./255)

    train_generator = train_datagen.flow_from_directory(
        train_dir, target_size=(img_height, img_width), batch_size=batch_size, class_mode='categorical')

    val_generator = val_test_datagen.flow_from_directory(
        val_dir, target_size=(img_height, img_width), batch_size=batch_size, class_mode='categorical')

    test_generator = val_test_datagen.flow_from_directory(
        test_dir, target_size=(img_height, img_width), batch_size=batch_size, class_mode='categorical', shuffle=False)
```

```
[ ]  print("Train samples:", train_generator.samples)
     print("Validation samples:", val_generator.samples)
     print("Test samples:", test_generator.samples)

     class_names = list(train_generator.class_indices.keys())
     images, labels = next(train_generator)

     plt.figure(figsize=(10, 6))
     for i in range(6):
         plt.subplot(2, 3, i+1)
```

```
Found 695 images belonging to 7 classes.
Found 285 images belonging to 7 classes.
Found 297 images belonging to 7 classes.
Train samples: 695
Validation samples: 285
Test samples: 297
```



2.CNN model with optimal hyperparameters

```
[ ]  import tensorflow as tf
     from tensorflow.keras import layers, models
     from tensorflow.keras.optimizers import Adam

     input_shape = (64, 64, 3)
     num_classes = train_generator.num_classes

     model = models.Sequential([
         # Conv Layer 1
         layers.Conv2D(32, (3, 3), activation='relu', input_shape=input_shape),
         layers.MaxPooling2D((2, 2)),

         # Conv Layer 2
         layers.Conv2D(64, (3, 3), activation='relu'),
         layers.MaxPooling2D((2, 2)),

         # Conv Layer 3
         layers.Conv2D(128, (3, 3), activation='relu'),
         layers.MaxPooling2D((2, 2)),

         layers.Flatten(),
         layers.Dense(128, activation='relu'),  # Fully Connected Layer
         layers.Dense(7, activation='softmax')  # Output Layer
     ])
```

Model: "sequential_23"

Layer (type)	Output Shape	Param #
conv2d_66 (Conv2D)	(None, 62, 62, 32)	896
max_pooling2d_66 (MaxPooling2D)	(None, 31, 31, 32)	0
conv2d_67 (Conv2D)	(None, 29, 29, 64)	18,496
max_pooling2d_67 (MaxPooling2D)	(None, 14, 14, 64)	0
conv2d_68 (Conv2D)	(None, 12, 12, 128)	73,856
max_pooling2d_68 (MaxPooling2D)	(None, 6, 6, 128)	0
flatten_18 (Flatten)	(None, 4608)	0
dense_64 (Dense)	(None, 128)	589,952
dense_65 (Dense)	(None, 7)	903

Total params: 684,103 (2.61 MB)
Trainable params: 684,103 (2.61 MB)
Non-trainable params: 0 (0.00 B)

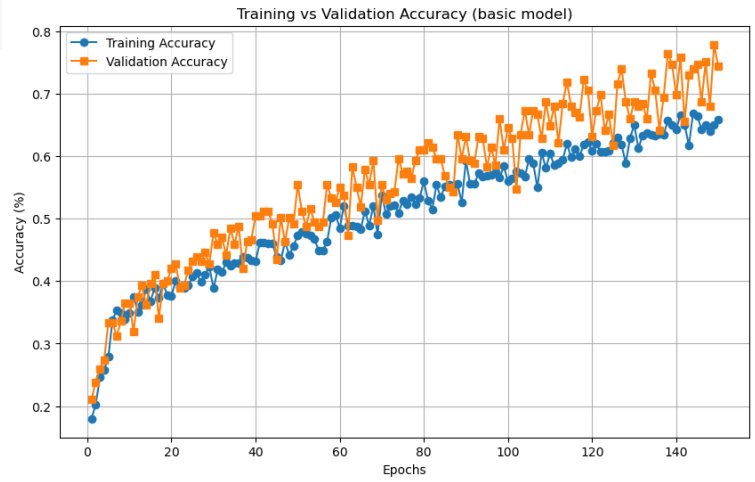
3.Train the basic CNN model

```
[ ]  epochs = 150

     history = model.fit(
         train_generator,
         epochs=epochs,
         validation_data=val_generator
     )
```

```
Epoch 1/150
/opt/anaconda3/lib/python3.12/site-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning:
  self._warn_if_super_not_called()
22/22 ━━━━━━━━━━━━━━━━━━━━ 1s 43ms/step - accuracy: 0.1658 - loss: 1.9420 - val_accuracy: 0.2105 - val_loss: 1.9160
Epoch 2/150
22/22 ━━━━━━━━━━━━━━━━━━━━ 1s 37ms/step - accuracy: 0.2135 - loss: 1.9096 - val_accuracy: 0.2386 - val_loss: 1.8956
Epoch 3/150
22/22 ━━━━━━━━━━━━━━━━━━━━ 1s 38ms/step - accuracy: 0.2398 - loss: 1.8894 - val_accuracy: 0.2596 - val_loss: 1.8562
Epoch 4/150
22/22 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.2665 - loss: 1.8600 - val_accuracy: 0.2737 - val_loss: 1.8158
Epoch 5/150
22/22 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.3000 - loss: 1.8058 - val_accuracy: 0.3333 - val_loss: 1.7513
Epoch 6/150
22/22 ━━━━━━━━━━━━━━━━━━━━ 1s 36ms/step - accuracy: 0.3501 - loss: 1.7666 - val_accuracy: 0.3333 - val_loss: 1.6955

Epoch 145/150
22/22 ━━━━━━━━━━━━━━━━━━━━ 1s 45ms/step - accuracy: 0.6963 - loss: 0.8722 - val_accuracy: 0.7474 - val_loss: 0.7942
Epoch 146/150
22/22 ━━━━━━━━━━━━━━━━━━━━ 1s 44ms/step - accuracy: 0.6413 - loss: 0.9148 - val_accuracy: 0.6877 - val_loss: 0.8506
Epoch 147/150
22/22 ━━━━━━━━━━━━━━━━━━━━ 1s 45ms/step - accuracy: 0.6753 - loss: 0.9116 - val_accuracy: 0.7509 - val_loss: 0.7964
Epoch 148/150
22/22 ━━━━━━━━━━━━━━━━━━━━ 1s 44ms/step - accuracy: 0.6468 - loss: 0.8976 - val_accuracy: 0.6807 - val_loss: 0.8792
Epoch 149/150
22/22 ━━━━━━━━━━━━━━━━━━━━ 1s 45ms/step - accuracy: 0.6556 - loss: 0.8779 - val_accuracy: 0.7789 - val_loss: 0.7817
Epoch 150/150
22/22 ━━━━━━━━━━━━━━━━━━━━ 1s 44ms/step - accuracy: 0.6501 - loss: 0.8758 - val_accuracy: 0.7439 - val_loss: 0.7852
```

4.Accuracy graph

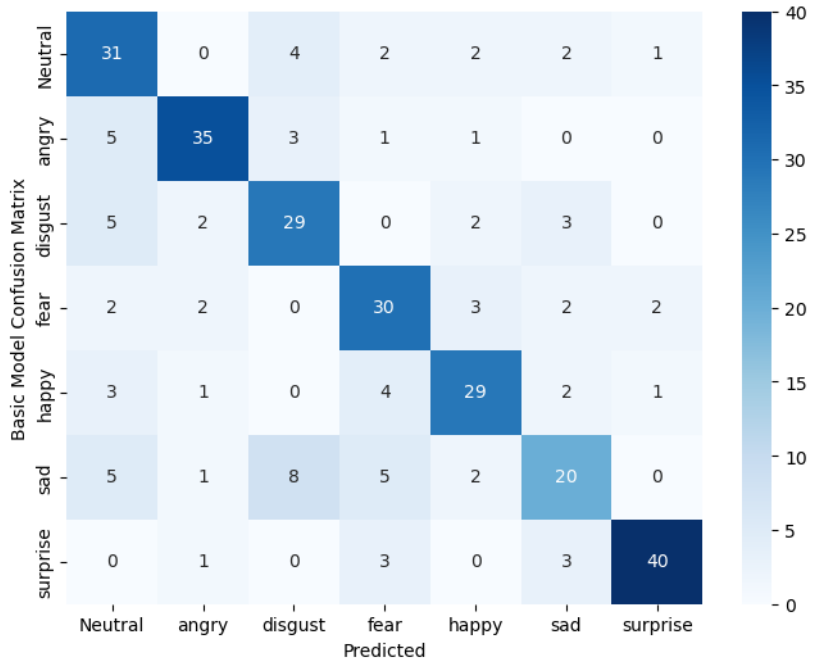
```
[ ]  acc = history.history['accuracy']
     val_acc = history.history['val_accuracy']
     epochs_range = range(1, len(acc) + 1)

     plt.figure(figsize=(10, 6))
     plt.plot(epochs_range, acc, 'o-', label='Training Accuracy')
     plt.plot(epochs_range, val_acc, 's-', label='Validation Accuracy')
     plt.title('Training vs Validation Accuracy (basic model)')
     plt.xlabel('Epochs')
     plt.ylabel('Accuracy (%)')
     plt.legend()
     plt.grid(True)
```



5.Evaluate model (classification_report, confusion_matrix)

```
[ ]  from sklearn.metrics import classification_report, confusion_matrix , accuracy_score
     import seaborn as sns


     Y_pred = model.predict(test_generator)
     y_pred = np.argmax(Y_pred, axis=1)

     true_labels = test_generator.classes
     class_names = list(test_generator.class_indices.keys())

     # Classification report
     print(classification_report(true_labels, y_pred, target_names=class_names))

     # Training and Validation Accuracy
     train_acc = history.history['accuracy'][-1]
     val_acc = history.history['val_accuracy'][-1]

     print(f"Final Training Accuracy: {train_acc * 100:.2f}%")
     print(f"Final Validation Accuracy: {val_acc * 100:.2f}%")

     # Confusion Matrix
     cm = confusion_matrix(true_labels, y_pred)
     plt.figure(figsize=(8,6))
     sns.heatmap(cm, annot=True, fmt='d', xticklabels=class_names, yticklabels=class_names, cmap="Blues")
     plt.xlabel('Predicted')
     plt.ylabel('Basic Model Confusion Matrix')
     plt.show()
```

```
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
              precision    recall  f1-score   support

     Neutral       0.61      0.74      0.67        42
       angry       0.83      0.78      0.80        45
     disgust       0.66      0.71      0.68        41
        fear       0.67      0.73      0.70        41
       happy       0.74      0.72      0.73        40
         sad       0.62      0.49      0.55        41
    surprise       0.91      0.85      0.88        47

    accuracy                           0.72       297
   macro avg       0.72      0.72      0.72       297
weighted avg       0.73      0.72      0.72       297

Final Training Accuracy: 65.90%
Final Validation Accuracy: 74.39%
```



-Enhancement CNN model

6.balance learning to avoid bias

```
[ ] from sklearn.utils.class_weight import compute_class_weight

    y_train = train_generator.classes
    class_weights = compute_class_weight(class_weight='balanced', classes=np.unique(y_train), y=y_train)
    class_weights_dict = dict(enumerate(class_weights))

    print("Class Weights:", class_weights_dict)
```

```
Class Weights: {0: 1.0342261904761905, 1: 0.9193121693121693, 2: 0.9108781127129751, 3: 0.9928571428571429, 4: 0.9928571428571429, 5: 1.1962134251290877, 6: 1.002886002886003}
```

7.Enhancment model

```
[ ] import tensorflow as tf
    from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNormalization

    model = Sequential()

    model.add(Conv2D(32, (3, 3), padding='same', activation='relu', input_shape=(64, 64, 3)))
    model.add(BatchNormalization())
    model.add(MaxPooling2D((2, 2)))

    model.add(Conv2D(64, (3, 3), padding='same', activation='relu'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D((2, 2)))

    model.add(Conv2D(128, (3, 3), padding='same', activation='relu'))
    model.add(BatchNormalization())
    model.add(MaxPooling2D((2, 2)))
```

Model: "sequential_24"

Layer (type)	Output Shape	Param #
conv2d_69 (Conv2D)	(None, 64, 64, 32)	896
batch_normalization_36 (BatchNormalization)	(None, 64, 64, 32)	128
max_pooling2d_69 (MaxPooling2D)	(None, 32, 32, 32)	0
conv2d_70 (Conv2D)	(None, 32, 32, 64)	18,496
batch_normalization_37 (BatchNormalization)	(None, 32, 32, 64)	256
max_pooling2d_70 (MaxPooling2D)	(None, 16, 16, 64)	0
conv2d_71 (Conv2D)	(None, 16, 16, 128)	73,856
batch_normalization_38 (BatchNormalization)	(None, 16, 16, 128)	512
max_pooling2d_71 (MaxPooling2D)	(None, 8, 8, 128)	0
flatten_19 (Flatten)	(None, 8192)	0
dense_66 (Dense)	(None, 128)	1,048,704
dropout_53 (Dropout)	(None, 128)	0
dense_67 (Dense)	(None, 7)	903

Total params: 1,143,751 (4.36 MB)
Trainable params: 1,143,303 (4.36 MB)
Non-trainable params: 448 (1.75 KB)

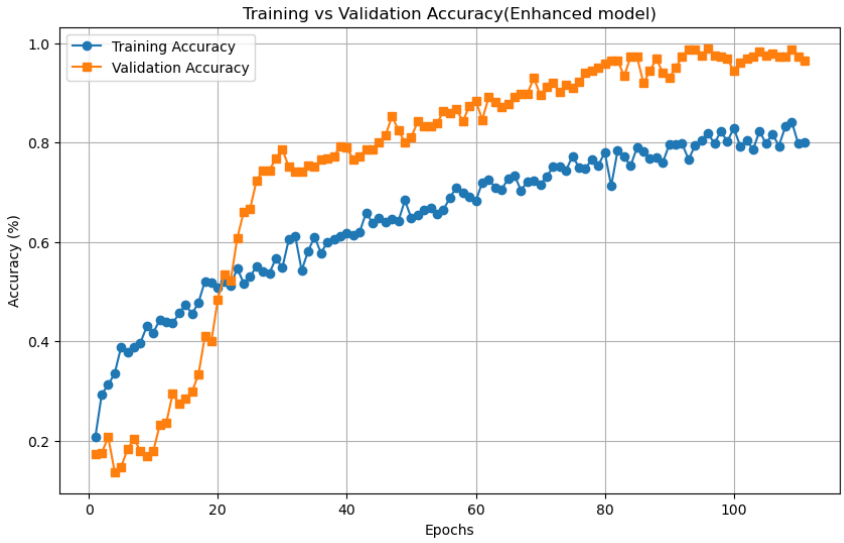
8.Train enhancment model

```
[ ] from tensorflow.keras.callbacks import EarlyStopping

    early_stop = EarlyStopping(monitor='val_accuracy', patience=15, restore_best_weights=True)

    history = model.fit(
        train_generator,
        validation_data=val_generator,
        epochs=150,
        callbacks=[early_stop],
        class_weight=class_weights_dict
    )
```

```
Epoch 1/150
22/22 ━━━━━━━━━━━━━━━━━━━━ 2s 69ms/step - accuracy: 0.2006 - loss: 2.7381 - val_accuracy: 0.1719 - val_loss: 1.9411
Epoch 2/150
22/22 ━━━━━━━━━━━━━━━━━━━━ 1s 67ms/step - accuracy: 0.2822 - loss: 1.9369 - val_accuracy: 0.1754 - val_loss: 2.0791
Epoch 3/150
22/22 ━━━━━━━━━━━━━━━━━━━━ 2s 72ms/step - accuracy: 0.3273 - loss: 1.6962 - val_accuracy: 0.2070 - val_loss: 2.2961
Epoch 4/150
22/22 ━━━━━━━━━━━━━━━━━━━━ 2s 74ms/step - accuracy: 0.3337 - loss: 1.6781 - val_accuracy: 0.1368 - val_loss: 2.4802
Epoch 5/150
22/22 ━━━━━━━━━━━━━━━━━━━━ 2s 71ms/step - accuracy: 0.3762 - loss: 1.5614 - val_accuracy: 0.1474 - val_loss: 2.7199

Epoch 107/150
22/22 ━━━━━━━━━━━━━━━━━━━━ 2s 78ms/step - accuracy: 0.7860 - loss: 0.5192 - val_accuracy: 0.9719 - val_loss: 0.1502
Epoch 108/150
22/22 ━━━━━━━━━━━━━━━━━━━━ 2s 80ms/step - accuracy: 0.8455 - loss: 0.4626 - val_accuracy: 0.9719 - val_loss: 0.1479
Epoch 109/150
22/22 ━━━━━━━━━━━━━━━━━━━━ 2s 79ms/step - accuracy: 0.8439 - loss: 0.4666 - val_accuracy: 0.9860 - val_loss: 0.1419
Epoch 110/150
22/22 ━━━━━━━━━━━━━━━━━━━━ 2s 73ms/step - accuracy: 0.8154 - loss: 0.4695 - val_accuracy: 0.9719 - val_loss: 0.1716
Epoch 111/150
22/22 ━━━━━━━━━━━━━━━━━━━━ 2s 73ms/step - accuracy: 0.7937 - loss: 0.5315 - val_accuracy: 0.9649 - val_loss: 0.1825
```

9.Accuracy graph for enhancment model

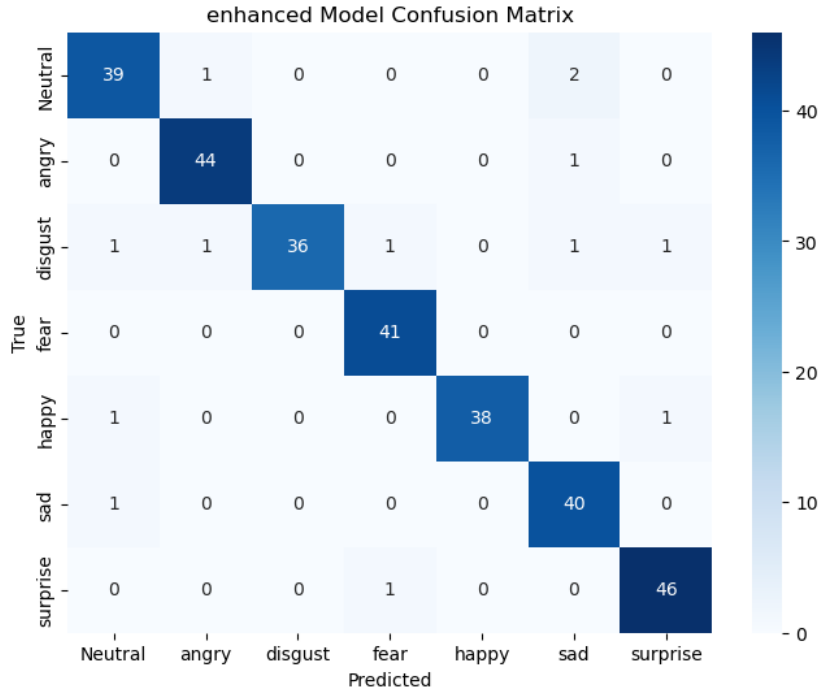
```
[ ]  acc = history.history['accuracy']
     val_acc = history.history['val_accuracy']
     epochs_range = range(1, len(acc) + 1)

     plt.figure(figsize=(10, 6))
     plt.plot(epochs_range, acc, 'o-', label='Training Accuracy')
     plt.plot(epochs_range, val_acc, 's-', label='Validation Accuracy')
     plt.title('Training vs Validation Accuracy(Enhanced model)')
     plt.xlabel('Epochs')
     plt.ylabel('Accuracy (%)')
     plt.legend()
     plt.grid(True)
     plt.show()
```



10.Evaluate model (classification_report, confusion_matrix)

```
[ ]  from sklearn.metrics import  accuracy_score

     Y_pred = model.predict(test_generator)
     y_pred = np.argmax(Y_pred, axis=1)

     true_labels = test_generator.classes
     class_names = list(test_generator.class_indices.keys())

     # Classification report
     print(classification_report(true_labels, y_pred, target_names=class_names))

     # Training and Validation Accuracy
     train_acc = history.history['accuracy'][-1]
     val_acc = history.history['val_accuracy'][-1]

     print(f"Final Training Accuracy: {train_acc * 100:.2f}%")
     print(f"Final Validation Accuracy: {val_acc * 100:.2f}%")

     # Confusion Matrix
     cm = confusion_matrix(true_labels, y_pred)
     plt.figure(figsize=(8,6))
     sns.heatmap(cm, annot=True, fmt='d', xticklabels=class_names, yticklabels=class_names, cmap="Blues")
     plt.xlabel('Predicted')
     plt.ylabel('True')
     plt.title('enhanced Model Confusion Matrix')
     plt.show()
```

```
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step
              precision    recall  f1-score   support

     Neutral       0.93      0.93      0.93        42
       angry       0.96      0.98      0.97        45
     disgust       1.00      0.88      0.94        41
        fear       0.95      1.00      0.98        41
       happy       1.00      0.95      0.97        40
         sad       0.91      0.98      0.94        41
    surprise       0.96      0.98      0.97        47

    accuracy                           0.96       297
   macro avg       0.96      0.96      0.96       297
weighted avg       0.96      0.96      0.96       297

Final Training Accuracy: 80.00%
Final Validation Accuracy: 96.49%
```



-Baseline model

11.baseline model

```
[ ]  from tensorflow.keras import layers, models
     from tensorflow.keras.optimizers import Adam

     baseline_model = models.Sequential([
         layers.Conv2D(16, (3, 3), activation='relu', input_shape=(64, 64, 3)),
         layers.MaxPooling2D((2, 2)),

         layers.Conv2D(32, (3, 3), activation='relu'),
         layers.MaxPooling2D((2, 2)),

         layers.Flatten(),
         layers.Dense(64, activation='relu'),
         layers.Dense(7, activation='softmax')
     ])

     baseline_model.compile(
```

Model: "sequential_25"

Layer (type)	Output Shape	Param #
conv2d_72 (Conv2D)	(None, 62, 62, 16)	448
max_pooling2d_72 (MaxPooling2D)	(None, 31, 31, 16)	0
conv2d_73 (Conv2D)	(None, 29, 29, 32)	4,640
max_pooling2d_73 (MaxPooling2D)	(None, 14, 14, 32)	0
flatten_20 (Flatten)	(None, 6272)	0
dense_68 (Dense)	(None, 64)	401,472
dense_69 (Dense)	(None, 7)	455

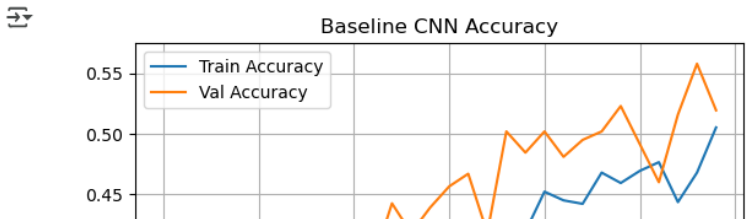
12.Train baseline model

```
[ ]  baseline_history = baseline_model.fit(
         train_generator,
         validation_data=val_generator,
         epochs=30
     )
```

```
Epoch 1/30
22/22 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 0.1907 - loss: 2.0065 - val_accuracy: 0.3018 - val_loss: 1.8452
Epoch 2/30
22/22 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - accuracy: 0.2833 - loss: 1.8369 - val_accuracy: 0.3333 - val_loss: 1.7722
Epoch 3/30
22/22 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - accuracy: 0.3377 - loss: 1.7562 - val_accuracy: 0.3404 - val_loss: 1.6713
Epoch 4/30
22/22 ━━━━━━━━━━━━━━━━━━━━ 1s 31ms/step - accuracy: 0.3046 - loss: 1.6850 - val_accuracy: 0.3333 - val_loss: 1.6321
Epoch 5/30
22/22 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.3255 - loss: 1.6568 - val_accuracy: 0.3579 - val_loss: 1.6573

Epoch 27/30
22/22 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.4572 - loss: 1.3192 - val_accuracy: 0.4596 - val_loss: 1.4201
Epoch 28/30
22/22 ━━━━━━━━━━━━━━━━━━━━ 1s 32ms/step - accuracy: 0.4370 - loss: 1.3178 - val_accuracy: 0.5158 - val_loss: 1.2232
Epoch 29/30
22/22 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.4573 - loss: 1.3162 - val_accuracy: 0.5579 - val_loss: 1.1874
Epoch 30/30
22/22 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 0.5168 - loss: 1.2546 - val_accuracy: 0.5193 - val_loss: 1.1947
```

13.Accuracy graph for baseline model

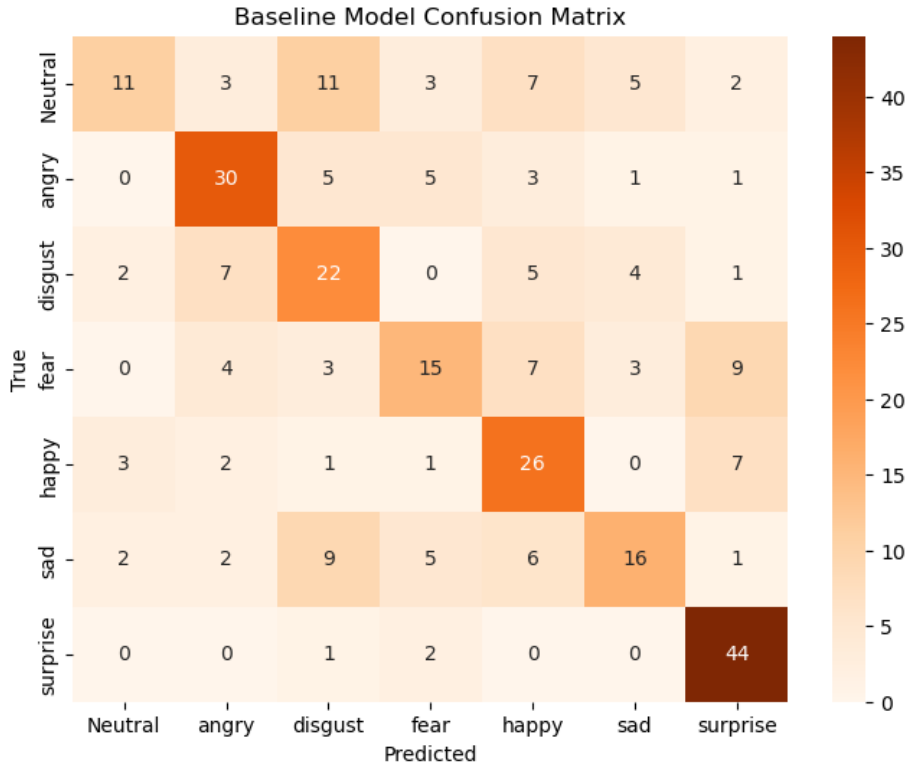
```
[ ]  plt.plot(baseline_history.history['accuracy'], label='Train Accuracy')
     plt.plot(baseline_history.history['val_accuracy'], label='Val Accuracy')
     plt.title('Baseline CNN Accuracy')
     plt.xlabel('Epochs')
     plt.ylabel('Accuracy')
     plt.legend()
     plt.grid(True)
     plt.show()
```



14.Evaluate model (classification_report, confusion_matrix)

```
[ ]  from sklearn.metrics import classification_report, confusion_matrix
     import matplotlib.pyplot as plt
     import seaborn as sns


     baseline_pred = baseline_model.predict(test_generator)
     baseline_y_pred = np.argmax(baseline_pred, axis=1)
     baseline_true_labels = test_generator.classes

     print(classification_report(baseline_true_labels, baseline_y_pred, target_names=class_names))

     train_acc = baseline_history.history['accuracy'][-1]
     val_acc = baseline_history.history['val_accuracy'][-1]

     print(f"Final Training Accuracy: {train_acc * 100:.2f}%")
     print(f"Final Validation Accuracy: {val_acc * 100:.2f}%")

     cm = confusion_matrix(baseline_true_labels, baseline_y_pred)
     plt.figure(figsize=(8,6))
     sns.heatmap(cm, annot=True, fmt='d', xticklabels=class_names, yticklabels=class_names, cmap="Oranges")
```

```
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
              precision    recall  f1-score   support

     Neutral       0.61      0.26      0.37        42
       angry       0.62      0.67      0.65        45
     disgust       0.42      0.54      0.47        41
        fear       0.48      0.37      0.42        41
       happy       0.48      0.65      0.55        40
         sad       0.55      0.39      0.46        41
    surprise       0.68      0.94      0.79        47

    accuracy                           0.55       297
   macro avg       0.55      0.54      0.53       297
weighted avg       0.55      0.55      0.53       297

Final Training Accuracy: 50.50%
Final Validation Accuracy: 51.93%
```



Refer

[1] https://www.kaggle.com/datasets/msambare/fer2013

[2] **N. Mehendale,** "Facial Emotion Recognition Using Convolutional Neural Networks (FERC)," *SN Applied Sciences*, vol. 2, no. 446, pp. 1-12, 2020, doi: 10.1007/s42452-020-2234-1.

[3] **Badrulhisham, N. A. S., & Mangshor, N. N. A.** (2021, July). Emotion recognition using a convolutional neural network (CNN). In *Journal of Physics: Conference Series* (Vol. 1962, No. 1, p. 012040). IOP Publishing.

[4] **Saravanan, A., Perichetla, G., & Gayathri, K. S.** (2019). Facial Emotion Recognition using Convolutional Neural Networks. Sri Venkateswara College of Engineering, Anna University, Chennai.