Final Project Report

on

Emotion Detector using CNN

(A Convolutional Neural Network Approach for Facial Emotion Classification)

Submitted by: **Bushra Khatoon**

August 20, 2025

Table of Contents

Abstract

This report documents the development of a facial emotion recognition system using a Convolutional Neural Network (CNN). Images from the 'emotion-detection-fer' dataset were preprocessed in grayscale and resized to 100×100 pixels. A three-block CNN with batch normalization, max pooling, and fully connected layers was trained using the Adam optimizer and categorical cross-entropy loss for 50 epochs. While the training accuracy rose to approximately 93.9%, validation accuracy plateaued near ~52–53%, and the held-out test accuracy was approximately 52.4%. The results highlight significant overfitting and class imbalance, with comparatively strong performance on 'happy' and 'neutral' classes and weaker performance on minority or harder classes such as 'disgusted'. Recommendations include data augmentation, class weighting, stronger regularization, and transfer learning.

Introduction

Human beings are inherently emotional, and emotions guide cognition, decision-making, and social interaction. Computationally detecting emotions has long been a challenge, but recent advances in deep learning have opened doors to practical applications (Binali et al., 2010; Jaiswal et al., 2020; Kratzwald et al., 2018). Facial expressions, as one of the most universal indicators of emotions, provide a fertile ground for automated detection systems (Fathallah et al., 2017a; Ge et al., 2022; Jaiswal et al., 2020). In this project, we attempt to harness CNN-based models for classifying emotions from facial images.

Traditional machine learning approaches required feature extraction techniques such as Histogram of Oriented Gradients (HOG), Local Binary Patterns (LBP), or Gabor filters (Guhdar and Melhum, 2020; Xiang et al., 2018). These methods, while effective to some extent, often struggled with variation in lighting, facial orientation, and subtle emotional cues. Deep learning, specifically CNNs, addresses these shortcomings by automatically learning hierarchical features from raw images, ranging from low-level edges and textures to high-level semantic patterns (Chen et al., 2018; Li and Deng, 2022). Convolutional Neural Networks (CNNs) are the de-facto standard for image classification and have become especially influential in facial expression recognition (FER) (Agrawal and Mittal, 2020; Fathallah et al., 2017b). By stacking convolution, nonlinearity, and pooling operations, CNNs learn hierarchical features—from edges and facial contours in early layers to expression-relevant patterns (e.g., furrowed brows, lip curvature, eye shape) in deeper layers (Agrawal and Mittal, 2020, 2020; Chen et al., 2018). This hierarchy makes CNNs well suited to discriminate among subtle, often overlapping facial cues required for seven-class FER (angry, disgusted, fearful, happy, neutral, sad, surprised) (Agrawal and Mittal, 2020; Chen et al., 2018; Fathallah et al., 2017).

The introduction also emphasizes the growing demand for emotion detection in real-time applications. For instance, online education platforms could adapt lesson difficulty based on student frustration, customer service bots could gauge user satisfaction in real time, and healthcare monitoring tools could track stress and depression levels in patients. By presenting this work, we showcase not only a technical implementation but also the societal relevance of FER systems.

In FER pipelines, CNNs typically operate on standardized inputs (cropped and aligned faces, normalized intensities), and produce class probabilities via a softmax layer (Shin et al., 2016; Singh and Nasoz, 2020). Training is performed end-to-end with a cross-entropy objective, allowing the network to discover task-specific filters that are more robust than hand-engineered features (Shin et al., 2016; Singh and Nasoz, 2020). Despite their power, CNNs in FER face recurring challenges: intra-class variability (the same emotion looks different across people, poses, lighting), inter-class similarity (e.g., angry vs. sad vs. fearful), occlusions (hands, glasses, masks), and class imbalance (some emotions are much rarer than others). Effective systems therefore combine architectural choices (batch normalization, dropout/SpatialDropout), data strategies (augmentation, class-aware sampling/weights), and, where possible, transfer learning from large face or general-image backbones.

Literature Review

The field of facial emotion recognition has evolved significantly over the past two decades. Early research focused on hand-engineered features combined with classifiers such as Support Vector Machines (SVM) or Random Forests. These approaches relied heavily on domain expertise to craft features but lacked scalability across diverse datasets. Viola and Jones' (2001) Haar cascade-based detector revolutionized object detection and inspired similar work in face detection, but emotion classification remained limited.

With the rise of deep learning, particularly convolutional architectures (LeCun et al., 1998; Krizhevsky et al., 2012), FER systems began to demonstrate remarkable improvements. Models such as VGGNet and ResNet provided robust architectures capable of handling variations in facial geometry, lighting, and occlusion. More recent literature also explores hybrid models combining CNNs with Recurrent Neural Networks (RNNs) or Transformers to capture temporal dynamics in videos, since emotions often unfold over time rather than single frames (Agrawal and Mittal, 2020; Shin et al., 2016).

Another critical aspect discussed in the literature is dataset availability. FER2013, JAFFE, CK+, and Kaggle FER datasets have enabled consistent benchmarking. Despite these advances, challenges such as class imbalance, cultural variability in expressions, and subtlety in emotions persist. This project situates itself within this trajectory, focusing on applying CNN architectures to the FER dataset, analyzing their performance, and suggesting improvements based on contemporary insights from the literature.

System Requirements

The project was implemented in Google Colab using Python 3. Libraries and frameworks used include TensorFlow, Keras, NumPy, Pandas, Matplotlib, Seaborn, and OpenCV. A GPU-enabled runtime was utilized to accelerate training.

Methodology

The methodology for this project is systematic and structured to ensure reliable experimentation and reproducibility. It comprises the following stages:

(i) Dataset Acquisition
(ii) Data Preprocessing
(iii) Model Design
(iv) Training Procedure
(v) Evaluation Metrics
(vi) Prediction on Custom Images

Dataset Acquisition

The dataset was fetched programmatically from Kaggle using the Kaggle API. The dataset consists of approximately 28,000 training images and 7,000 testing images spread across seven emotion categories. The seven classes and their training and test image counts observed were:

Train: surprised (3171), disgusted (436), angry (3995), sad (4830), fearful (4097), neutral (4965), happy (7215)

Test: surprised (831), disgusted (111), angry (958), sad (1247), fearful (1024), neutral (1233), happy (1774)

Data Preprocessing

Images were converted to greyscale and resized to 100 ×100 pixels to maintain uniform input dimensions. Normalization (scaling pixel values between 0 and 1) was applied to stabilize training. Data splitting was performed to maintain separation between training and testing subsets.

Model Design

A custom CNN was implemented with three convolutional blocks (Conv2D → Batch Normalization → MaxPooling) followed by fully connected layers. Dropout layers were included to reduce overfitting, and ReLU activation functions were used for hidden layers, while sigmoid activation was applied at the output layer.

Training Procedure

TensorFlow/Keras ImageDataGenerator was used with a validation split for training/validation generators and a separate generator for the test set. The Adam optimizer was selected for efficient gradient descent. Sparse categorical cross-entropy loss was used as the loss function, suitable for multi-class classification. The model was trained for 50 epochs with a batch size of 64.

Evaluation Metrics

Accuracy and loss values were monitored for both training and validation sets. Visualization of learning curves allowed diagnosis of overfitting and underfitting trends.

Prediction on Custom Images

After training, the model was tested on external images (angry, happy, neutral, fearful) to validate generalization in real-world settings. This systematic methodology provided a clear pathway from dataset preparation to experimental validation.

Reference Python Codes

This section provides the canonical, runnable Python snippets that reproduce our results end-to-end. The code is organized into concise listings covering (i) dataset download/setup (Kaggle API), (ii) preprocessing and directory-based data loaders with a validation split, (iii) the baseline CNN architecture definition, (iv) compilation and training with and optional class weight for imbalance, (v) evaluation utilities (confusion matrix, per-class precision/recall/F1), and (vi) inference on custom images using the exact training preprocessing pipeline. All snippets use the same hyperparameters reported in the paper and include the saved class–index mapping to avoid label drift across environments. For strict reproducibility, we also note package versions and random seeds; full scripts and environment files are provided in the Appendix.

Results

- Training reached ~0.939 accuracy by epoch 50, while validation accuracy fluctuated around 0.51–0.53, peaking near ~0.529.
- Held-out test accuracy: 0.5242.
- Per-class metrics from the test classification report:

Class (index)	Precision	Recall	F1-score	Support
angry (0)	0.3995	0.4603	0.4277	958
disgusted (1)	0.4024	0.2973	0.3420	111
fearful (2)	0.3609	0.3799	0.3701	1024
happy (3)	0.7494	0.7317	0.7404	1774
neutral (4)	0.4985	0.4047	0.4467	1233
sad (5)	0.3830	0.4306	0.4054	1247
surprised (6)	0.7266	0.6811	0.7031	831
accuracy			0.5242	7178
macro avg	0.5029	0.4837	0.4908	7178
weighted avg	0.5325	0.5242	0.5267	7178

The confusion matrix (not reproduced here) showed stronger recognition for ‘happy’ and ‘surprised’, with notable confusion among ‘angry’, ‘fearful’, ‘neutral’, and ‘sad’. The ‘disgusted’ class suffered from data scarcity (only 436 training images and 111 test images), contributing to lower recall.

Discussion

The gap between training and validation/test accuracy indicates overfitting. Contributing factors include:

1. Class Imbalance: The dataset is skewed (e.g., ‘happy’ has 7215 train images vs. ‘disgusted’ with only 436). These bias favors majority classes.
2. Limited Regularization: Dropout of 0.1 may be insufficient; no L2 weight decay was used.
3. No Data Augmentation: Realistic augmentations (cropping, rotation, brightness/contrast jitter, horizontal flip) can improve generalization.

Recommended remedies:

1. Apply class weights or focal loss to address imbalance.
2. Add data augmentation aggressively (but realistically) and consider color input if beneficial.
3. Use stronger regularization (higher dropout, L2, early stopping, and learning-rate schedules).
4. Explore transfer learning using face-focused backbones (e.g., MobileNetV2 or EfficientNet) and fine-tuning on this dataset.
5. Improve face alignment/cropping to reduce background variation.

Conclusion

This baseline CNN achieves ~52% test accuracy on seven-class facial emotion recognition. While it captures prominent expressions (e.g., happy, surprised), the model struggles on minority and subtle classes. Future work should prioritize better data balance, augmentation, and transfer learning, alongside careful regularization and training-monitoring (early stopping, checkpointing).

References

1. Agrawal, A., Mittal, N., 2020. Using CNN for facial expression recognition: a study of the effects of kernel size and number of filters on accuracy. Vis. Comput. 36, 405–412. https://doi.org/10.1007/s00371-019-01630-9
2. Binali, H., Wu, C., Potdar, V., 2010. Computational approaches for emotion detection in text, in: 4th IEEE International Conference on Digital Ecosystems and Technologies. Presented at the 4th IEEE International Conference on Digital Ecosystems and Technologies, pp. 172–177. https://doi.org/10.1109/DEST.2010.5610650
3. Chen, Z., Huang, D., Wang, Y., Chen, L., 2018. Fast and Light Manifold CNN based 3D Facial Expression Recognition across Pose Variations, in: Proceedings of the 26th ACM International Conference on Multimedia. Presented at the MM '18: ACM Multimedia Conference, ACM, Seoul Republic of Korea, pp. 229–238. https://doi.org/10.1145/3240508.3240568
4. Fathallah, A., Abdi, L., Douik, A., 2017a. Facial Expression Recognition via Deep Learning, in: 2017 IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA). Presented at the 2017 IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA), pp. 745–750. https://doi.org/10.1109/AICCSA.2017.124
5. Fathallah, A., Abdi, L., Douik, A., 2017b. Facial Expression Recognition via Deep Learning, in: 2017 IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA). Presented at the 2017 IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA), pp. 745–750. https://doi.org/10.1109/AICCSA.2017.124
6. Ge, H., Zhu, Z., Dai, Y., Wang, B., Wu, X., 2022. Facial expression recognition based on deep learning. Comput. Methods Programs Biomed. 215, 106621. https://doi.org/10.1016/j.cmpb.2022.106621
7. Guhdar, M., Melhum, A., 2020. Implementation of HOG Feature Extraction with Tuned Parameters for Human Face Detection, International Journal of Machine Learning and Computing. https://doi.org/10.18178/ijmlc.2020.10.5.987
8. Jaiswal, A., Krishnama Raju, A., Deb, S., 2020. Facial Emotion Detection Using Deep Learning, in: 2020 International Conference for Emerging Technology (INCET). Presented at the 2020 International Conference for Emerging Technology (INCET), pp. 1–5. https://doi.org/10.1109/INCET49848.2020.9154121
9. Kratzwald, B., Ilić, S., Kraus, M., Feuerriegel, S., Prendinger, H., 2018. Deep learning for affective computing: Text-based emotion recognition in decision support. Decis. Support Syst. 115, 24–35. https://doi.org/10.1016/j.dss.2018.09.002

10. Li, S., Deng, W., 2022. Deep Facial Expression Recognition: A Survey. IEEE Trans. Affect. Comput. 13, 1195–1215. https://doi.org/10.1109/TAFFC.2020.2981446
11. Shin, M., Kim, M., Kwon, D.-S., 2016. Baseline CNN structure analysis for facial expression recognition, in: 2016 25th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN). Presented at the 2016 25th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN), pp. 724–729. https://doi.org/10.1109/ROMAN.2016.7745199
12. Singh, S., Nasoz, F., 2020. Facial Expression Recognition with Convolutional Neural Networks, in: 2020 10th Annual Computing and Communication Workshop and Conference (CCWC). Presented at the 2020 10th Annual Computing and Communication Workshop and Conference (CCWC), pp. 0324–0328. https://doi.org/10.1109/CCWC47524.2020.9031283
13. Xiang, Z., Tan, H., Ye, W., 2018. The Excellent Properties of a Dense Grid-Based HOG Feature on Face Recognition Compared to Gabor and LBP. IEEE Access 6, 29306–29319. https://doi.org/10.1109/ACCESS.2018.2813395
14. Kaggle Dataset: 'emotion-detection-fer' (used for training and testing), https://www.kaggle.com/datasets/ananthu017/emotion-detection-fer/code.
15. Viola, P., & Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, I–511–I–518.
16. Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems (NeurIPS)*, 1097–1105.
17. TensorFlow / Keras: High-level APIs for deep learning (model, layers, ImageDataGenerator).
18. Scikit-learn: Metrics and evaluation utilities (classification_report, confusion matrix).

Appendix

Final Project on **Emotion Detector**

Submitted by: **Bushra Khatoon**

In [1]:
```
from google.colab import files
files.upload()   # upload kaggle.json from your computer
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving kaggle.json to kaggle.json
```

Out[1]:

```
{'kaggle.json': b'{"username":"bushrakhatoon","key":"17a83efd40bfd25d5e5f5157cbde8078"}'}
```


In [2]:
```
!mkdir -p ~/.kaggle
!mv kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json
```

In [3]:
```
!pip -q install kaggle
import kaggle
!kaggle --version
```

```
Kaggle API 1.7.4.5
```

In [10]:
```
# Download the file from Kaggle link
# Use the dataset slug from your link
slug = "ananthu017/emotion-detection-fer"

# Choose a target folder
DATA_DIR = "/content/data/emotion_fer"
!mkdir -p "$DATA_DIR"

# Download and unzip (if --unzip fails on some setups, we unzip manually below)
!kaggle datasets download -d $slug -p "$DATA_DIR" --unzip
```

```
Dataset URL: https://www.kaggle.com/datasets/ananthu017/emotion-detection-fer
License(s): CC0-1.0
Downloading emotion-detection-fer.zip to /content/data/emotion_fer
  0% 0.00/65.2M [00:00<?, ?B/s]
100% 65.2M/65.2M [00:00<00:00, 1.21GB/s]
```

In [22]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os
import glob
import cv2
from PIL import Image
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow import keras
from keras import Sequential
from keras.layers import Dense, Conv2D, MaxPooling2D, Flatten, Activation, BatchNormalization, Drop
from tensorflow.keras import models, layers
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.utils import class_weight
import warnings
warnings.filterwarnings('ignore')
```

In [6]:
```python
# Inspect the directory structure
for dirpath, dirnames, filenames in os.walk("/content/data"):
    print(f"Directory: {dirpath}, Files: {len(filenames)}")
```

```
Directory: /content/data, Files: 0
Directory: /content/data/emotion_fer, Files: 0
Directory: /content/data/emotion_fer/test, Files: 0
Directory: /content/data/emotion_fer/test/surprised, Files: 831
Directory: /content/data/emotion_fer/test/disgusted, Files: 111
Directory: /content/data/emotion_fer/test/angry, Files: 958
Directory: /content/data/emotion_fer/test/sad, Files: 1247
Directory: /content/data/emotion_fer/test/fearful, Files: 1024
Directory: /content/data/emotion_fer/test/neutral, Files: 1233
Directory: /content/data/emotion_fer/test/happy, Files: 1774
Directory: /content/data/emotion_fer/train, Files: 0
Directory: /content/data/emotion_fer/train/surprised, Files: 3171
Directory: /content/data/emotion_fer/train/disgusted, Files: 436
Directory: /content/data/emotion_fer/train/angry, Files: 3995
Directory: /content/data/emotion_fer/train/sad, Files: 4830
Directory: /content/data/emotion_fer/train/fearful, Files: 4097
Directory: /content/data/emotion_fer/train/neutral, Files: 4965
Directory: /content/data/emotion_fer/train/happy, Files: 7215
```

In [17]:
```python
# Total number of images in train and test folders
total_train_images = sum(train_counts.values())
total_test_images = sum(test_counts.values())

print(f"Total images in train folder: {total_train_images}")
print(f"Total images in test folder: {total_test_images}")
```

```
Total images in train folder: 28709
Total images in test folder: 7178
```

In [15]:
```python
# Total number of images for all 7-categories in train and test directories.
train_counts = {}
test_counts = {}

train_dir = os.path.join(DATA_DIR, "train")
test_dir = os.path.join(DATA_DIR, "test")

if os.path.isdir(train_dir):
    for emotion_folder in os.listdir(train_dir):
        emotion_path = os.path.join(train_dir, emotion_folder)
        if os.path.isdir(emotion_path):
            train_counts[emotion_folder] = len(glob.glob(os.path.join(emotion_path, "*")))

if os.path.isdir(test_dir):
    for emotion_folder in os.listdir(test_dir):
        emotion_path = os.path.join(test_dir, emotion_folder)
        if os.path.isdir(emotion_path):
            test_counts[emotion_folder] = len(glob.glob(os.path.join(emotion_path, "*")))

print("Train counts:")
for emotion, count in train_counts.items():
    print(f"{emotion}: {count}")

print("\nTest counts:")
for emotion, count in test_counts.items():
    print(f"{emotion}: {count}")
```

```
Train counts:
surprised: 3171
disgusted: 436
angry: 3995
sad: 4830
fearful: 4097
neutral: 4965
happy: 7215

Test counts:
surprised: 831
disgusted: 111
angry: 958
sad: 1247
fearful: 1024
neutral: 1233
happy: 1774
```

In [13]:

```
# ✅ Show random images from the emotion_fer dataset
import glob, random, math
from PIL import Image

# 👇 Change this if your dataset lives elsewhere
DATA_DIR = "/content/data/emotion_fer"   # e.g., where you unzipped ananthu017/emotion-detection-fe

# Pick a split folder if it exists; otherwise search the root
base = None
for split in ["train", "validation", "val", "test"]:
    p = os.path.join(DATA_DIR, split)
    if os.path.isdir(p):
        base = p
        break
if base is None:
    base = DATA_DIR  # fallback to root

# Collect image paths
exts = ("*.jpg", "*.jpeg", "*.png", "*.bmp", "*.tif", "*.tiff")
paths = []
# common structure: base/<class>/*.jpg
class_dirs = [d for d in os.listdir(base) if os.path.isdir(os.path.join(base, d))]
if class_dirs:
    for cls in sorted(class_dirs):
        for e in exts:
            paths.extend(glob.glob(os.path.join(base, cls, e)))
# fallback: recursive search (in case the structure is different)
if not paths:
    for e in exts:
        paths.extend(glob.glob(os.path.join(DATA_DIR, "**", e), recursive=True))

if not paths:
    raise FileNotFoundError(f"No images found under {DATA_DIR}. Check your dataset path/folders.")

# Show a class balance snapshot (top 10)
labels = [os.path.basename(os.path.dirname(p)) for p in paths]
class_counts = pd.Series(labels).value_counts().head(10)
display(class_counts.to_frame("images"))
```

	images
happy	7215
neutral	4965
sad	4830
fearful	4097
angry	3995
surprised	3171
disgusted	436

In [18]:

```
# Sample & plot
K = 16  # number of images to show
sample = random.sample(paths, min(K, len(paths)))

cols = 4
rows = math.ceil(len(sample) / cols)
plt.figure(figsize=(cols * 3, rows * 3))
for i, p in enumerate(sample, 1):
    with Image.open(p) as img:
        img = img.convert("RGB")
    plt.subplot(rows, cols, i)
    plt.imshow(img)
    title = os.path.basename(os.path.dirname(p))
    plt.title(title[:16])
    plt.axis("off")
plt.tight_layout()
plt.show()

print(f"Showing {len(sample)} random images from: {base}")
```


```
Showing 16 random images from: /content/data/emotion_fer/train
```

Data preparation for CNN model with Tensorflow

Data generation and data normalization

In [21]:
```python
# Using IDG to load images from directory
train_idg = ImageDataGenerator(rescale=1./255, validation_split=0.25)
                                        # 25% validation split for taining
test_idg = ImageDataGenerator(rescale=1./255)

# Specify parameters for data generation
img_size = (100, 100)
batch_size = 64

arg_train = {'target_size': img_size,
             'color_mode': 'grayscale',
             'class_mode' : 'categorical',
             'batch_size': batch_size}
arg_test = {'target_size': img_size,
            'color_mode': 'grayscale',
```

```
            'class_mode' : 'categorical',
            'batch_size': batch_size,
            'shuffle': False}

train = train_idg.flow_from_directory(directory=train_dir, subset='training', **arg_train)
valid = train_idg.flow_from_directory(directory=train_dir, subset='validation', **arg_train)
test  = test_idg.flow_from_directory(directory=test_dir, **arg_test)
```

```
Found 21535 images belonging to 7 classes.
Found 7174 images belonging to 7 classes.
Found 7178 images belonging to 7 classes.
```

In [42]:

```
### Creat CNN Model
model=Sequential()
model.add(Conv2D(32,kernel_size=(3,3),padding='valid',activation='relu',input_shape=(100,100,1)))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2),strides=2,padding='valid'))

model.add(Conv2D(64,kernel_size=(3,3),padding='valid',activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2),strides=2,padding='valid'))

model.add(Conv2D(128,kernel_size=(3,3),padding='valid',activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2),strides=2,padding='valid'))


model.add(Flatten())
```

In [43]:

```
model.add(Dense(128,activation='relu'))
model.add(Dropout(0.1))
model.add(Dense(64,activation='relu'))
model.add(Dropout(0.1))
model.add(Dense(7,activation='softmax')) # Changed activation to softmax
model.summary()
```

Model: "sequential_6"

Layer (type)	Output Shape	Param #
conv2d_19 (Conv2D)	(None, 98, 98, 32)	320
batch_normalization_19 (BatchNormalization)	(None, 98, 98, 32)	128
max_pooling2d_19 (MaxPooling2D)	(None, 49, 49, 32)	0
conv2d_20 (Conv2D)	(None, 47, 47, 64)	18,496
batch_normalization_20 (BatchNormalization)	(None, 47, 47, 64)	256
max_pooling2d_20 (MaxPooling2D)	(None, 23, 23, 64)	0
conv2d_21 (Conv2D)	(None, 21, 21, 128)	73,856
batch_normalization_21 (BatchNormalization)	(None, 21, 21, 128)	512
max_pooling2d_21 (MaxPooling2D)	(None, 10, 10, 128)	0
flatten_6 (Flatten)	(None, 12800)	0
dense_18 (Dense)	(None, 128)	1,638,528
dropout_12 (Dropout)	(None, 128)	0
dense_19 (Dense)	(None, 64)	8,256
dropout_13 (Dropout)	(None, 64)	0
dense_20 (Dense)	(None, 7)	455

Total params: 1,740,807 (6.64 MB)

Trainable params: 1,740,359 (6.64 MB)

Non-trainable params: 448 (1.75 KB)

In [44]:
```
### To train the model
model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])

history=model.fit(train,epochs=50,validation_data=valid)
```

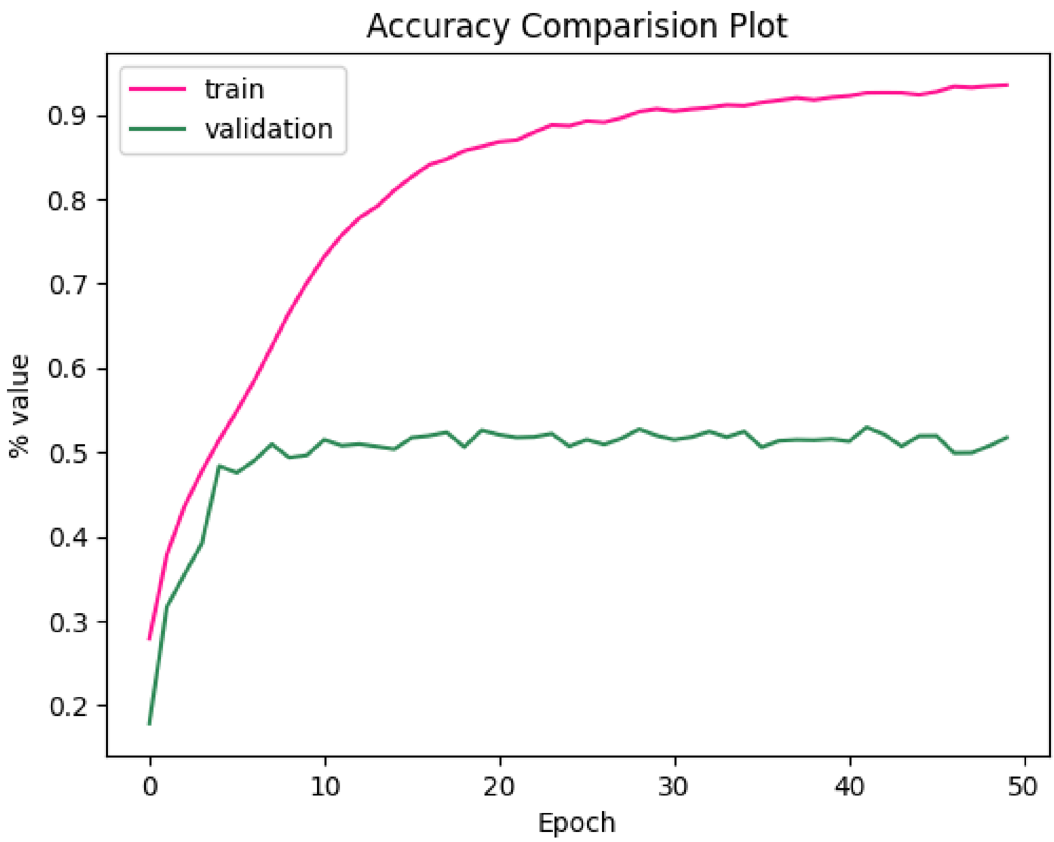
```
Epoch 1/50
337/337 ━━━━━━━━━━━━━━━━━━━━ 27s 58ms/step - accuracy: 0.2387 - loss: 2.1775 - val_accuracy: 0.1781
- val_loss: 4.8952
Epoch 2/50
337/337 ━━━━━━━━━━━━━━━━━━━━ 14s 42ms/step - accuracy: 0.3672 - loss: 1.5993 - val_accuracy: 0.3163
- val_loss: 1.6647
Epoch 3/50
337/337 ━━━━━━━━━━━━━━━━━━━━ 15s 45ms/step - accuracy: 0.4331 - loss: 1.4534 - val_accuracy: 0.3545
- val_loss: 1.6969
Epoch 4/50
337/337 ━━━━━━━━━━━━━━━━━━━━ 17s 49ms/step - accuracy: 0.4727 - loss: 1.3406 - val_accuracy: 0.3914
- val_loss: 1.5253
Epoch 5/50
337/337 ━━━━━━━━━━━━━━━━━━━━ 15s 44ms/step - accuracy: 0.5109 - loss: 1.2656 - val_accuracy: 0.4834
- val_loss: 1.3692
Epoch 6/50
337/337 ━━━━━━━━━━━━━━━━━━━━ 18s 54ms/step - accuracy: 0.5606 - loss: 1.1455 - val_accuracy: 0.4755
- val_loss: 1.3738
Epoch 7/50
337/337 ━━━━━━━━━━━━━━━━━━━━ 14s 42ms/step - accuracy: 0.5876 - loss: 1.0743 - val_accuracy: 0.4898
- val_loss: 1.3684
Epoch 8/50
337/337 ━━━━━━━━━━━━━━━━━━━━ 17s 50ms/step - accuracy: 0.6323 - loss: 0.9636 - val_accuracy: 0.5095
- val_loss: 1.3732
Epoch 9/50
337/337 ━━━━━━━━━━━━━━━━━━━━ 14s 42ms/step - accuracy: 0.6721 - loss: 0.8669 - val_accuracy: 0.4933
- val_loss: 1.4446
Epoch 10/50
337/337 ━━━━━━━━━━━━━━━━━━━━ 18s 54ms/step - accuracy: 0.7080 - loss: 0.7710 - val_accuracy: 0.4962
- val_loss: 1.4794
Epoch 11/50
337/337 ━━━━━━━━━━━━━━━━━━━━ 17s 49ms/step - accuracy: 0.7448 - loss: 0.6882 - val_accuracy: 0.5148
- val_loss: 1.5206
Epoch 12/50
337/337 ━━━━━━━━━━━━━━━━━━━━ 15s 43ms/step - accuracy: 0.7640 - loss: 0.6243 - val_accuracy: 0.5077
- val_loss: 1.5753
Epoch 13/50
337/337 ━━━━━━━━━━━━━━━━━━━━ 21s 46ms/step - accuracy: 0.7833 - loss: 0.5811 - val_accuracy: 0.5096
- val_loss: 1.7648
Epoch 14/50
337/337 ━━━━━━━━━━━━━━━━━━━━ 15s 45ms/step - accuracy: 0.7972 - loss: 0.5419 - val_accuracy: 0.5066
- val_loss: 1.7592
Epoch 15/50
337/337 ━━━━━━━━━━━━━━━━━━━━ 17s 49ms/step - accuracy: 0.8140 - loss: 0.5063 - val_accuracy: 0.5035
- val_loss: 1.7623
Epoch 16/50
337/337 ━━━━━━━━━━━━━━━━━━━━ 17s 49ms/step - accuracy: 0.8330 - loss: 0.4551 - val_accuracy: 0.5171
- val_loss: 1.8615
Epoch 17/50
337/337 ━━━━━━━━━━━━━━━━━━━━ 22s 66ms/step - accuracy: 0.8504 - loss: 0.4152 - val_accuracy: 0.5194
- val_loss: 1.8980
Epoch 18/50
337/337 ━━━━━━━━━━━━━━━━━━━━ 18s 52ms/step - accuracy: 0.8524 - loss: 0.4085 - val_accuracy: 0.5234
- val_loss: 1.9531
Epoch 19/50
337/337 ━━━━━━━━━━━━━━━━━━━━ 19s 55ms/step - accuracy: 0.8634 - loss: 0.3752 - val_accuracy: 0.5061
- val_loss: 1.9956
Epoch 20/50
337/337 ━━━━━━━━━━━━━━━━━━━━ 15s 46ms/step - accuracy: 0.8658 - loss: 0.3751 - val_accuracy: 0.5259
- val_loss: 1.9560
Epoch 21/50
337/337 ━━━━━━━━━━━━━━━━━━━━ 14s 42ms/step - accuracy: 0.8767 - loss: 0.3456 - val_accuracy: 0.5205
- val_loss: 2.1393
Epoch 22/50
337/337 ━━━━━━━━━━━━━━━━━━━━ 15s 45ms/step - accuracy: 0.8724 - loss: 0.3566 - val_accuracy: 0.5173
- val_loss: 2.1757
Epoch 23/50
```

```
337/337 ━━━━━━━━━━━━━━━━━━━━ 20s 45ms/step - accuracy: 0.8789 - loss: 0.3385 - val_accuracy: 0.5180
- val_loss: 2.1820
Epoch 24/50
337/337 ━━━━━━━━━━━━━━━━━━━━ 18s 54ms/step - accuracy: 0.8892 - loss: 0.3176 - val_accuracy: 0.5217
- val_loss: 2.1070
Epoch 25/50
337/337 ━━━━━━━━━━━━━━━━━━━━ 16s 48ms/step - accuracy: 0.8954 - loss: 0.3084 - val_accuracy: 0.5070
- val_loss: 2.2813
Epoch 26/50
337/337 ━━━━━━━━━━━━━━━━━━━━ 21s 63ms/step - accuracy: 0.8944 - loss: 0.3048 - val_accuracy: 0.5148
- val_loss: 2.3842
Epoch 27/50
337/337 ━━━━━━━━━━━━━━━━━━━━ 17s 49ms/step - accuracy: 0.8924 - loss: 0.3041 - val_accuracy: 0.5091
- val_loss: 2.3611
Epoch 28/50
337/337 ━━━━━━━━━━━━━━━━━━━━ 18s 53ms/step - accuracy: 0.8979 - loss: 0.2827 - val_accuracy: 0.5163
- val_loss: 2.3067
Epoch 29/50
337/337 ━━━━━━━━━━━━━━━━━━━━ 18s 53ms/step - accuracy: 0.9057 - loss: 0.2754 - val_accuracy: 0.5273
- val_loss: 2.3673
Epoch 30/50
337/337 ━━━━━━━━━━━━━━━━━━━━ 17s 50ms/step - accuracy: 0.9082 - loss: 0.2697 - val_accuracy: 0.5195
- val_loss: 2.2697
Epoch 31/50
337/337 ━━━━━━━━━━━━━━━━━━━━ 19s 57ms/step - accuracy: 0.9058 - loss: 0.2664 - val_accuracy: 0.5146
- val_loss: 2.5771
Epoch 32/50
337/337 ━━━━━━━━━━━━━━━━━━━━ 16s 49ms/step - accuracy: 0.9105 - loss: 0.2596 - val_accuracy: 0.5178
- val_loss: 2.4784
Epoch 33/50
337/337 ━━━━━━━━━━━━━━━━━━━━ 19s 55ms/step - accuracy: 0.9093 - loss: 0.2615 - val_accuracy: 0.5244
- val_loss: 2.4651
Epoch 34/50
337/337 ━━━━━━━━━━━━━━━━━━━━ 19s 50ms/step - accuracy: 0.9141 - loss: 0.2460 - val_accuracy: 0.5177
- val_loss: 2.6008
Epoch 35/50
337/337 ━━━━━━━━━━━━━━━━━━━━ 18s 53ms/step - accuracy: 0.9153 - loss: 0.2445 - val_accuracy: 0.5244
- val_loss: 2.5011
Epoch 36/50
337/337 ━━━━━━━━━━━━━━━━━━━━ 18s 52ms/step - accuracy: 0.9190 - loss: 0.2270 - val_accuracy: 0.5059
- val_loss: 2.5088
Epoch 37/50
337/337 ━━━━━━━━━━━━━━━━━━━━ 16s 48ms/step - accuracy: 0.9216 - loss: 0.2332 - val_accuracy: 0.5135
- val_loss: 2.6316
Epoch 38/50
337/337 ━━━━━━━━━━━━━━━━━━━━ 19s 55ms/step - accuracy: 0.9207 - loss: 0.2367 - val_accuracy: 0.5146
- val_loss: 2.4045
Epoch 39/50
337/337 ━━━━━━━━━━━━━━━━━━━━ 17s 50ms/step - accuracy: 0.9182 - loss: 0.2387 - val_accuracy: 0.5142
- val_loss: 2.4557
Epoch 40/50
337/337 ━━━━━━━━━━━━━━━━━━━━ 17s 51ms/step - accuracy: 0.9210 - loss: 0.2424 - val_accuracy: 0.5155
- val_loss: 2.8644
Epoch 41/50
337/337 ━━━━━━━━━━━━━━━━━━━━ 18s 53ms/step - accuracy: 0.9260 - loss: 0.2145 - val_accuracy: 0.5128
- val_loss: 2.2657
Epoch 42/50
337/337 ━━━━━━━━━━━━━━━━━━━━ 18s 52ms/step - accuracy: 0.9243 - loss: 0.2157 - val_accuracy: 0.5294
- val_loss: 2.7794
Epoch 43/50
337/337 ━━━━━━━━━━━━━━━━━━━━ 20s 59ms/step - accuracy: 0.9280 - loss: 0.2054 - val_accuracy: 0.5209
- val_loss: 2.6579
Epoch 44/50
337/337 ━━━━━━━━━━━━━━━━━━━━ 17s 51ms/step - accuracy: 0.9261 - loss: 0.2087 - val_accuracy: 0.5070
- val_loss: 2.5789
Epoch 45/50
337/337 ━━━━━━━━━━━━━━━━━━━━ 16s 48ms/step - accuracy: 0.9293 - loss: 0.2076 - val_accuracy: 0.5191
```

```
- val_loss: 2.2895
Epoch 46/50
337/337 ━━━━━━━━━━━━━━━━━━━━ 18s 54ms/step - accuracy: 0.9295 - loss: 0.2159 - val_accuracy: 0.5192
- val_loss: 2.4035
Epoch 47/50
337/337 ━━━━━━━━━━━━━━━━━━━━ 17s 50ms/step - accuracy: 0.9336 - loss: 0.1988 - val_accuracy: 0.4989
- val_loss: 3.0592
Epoch 48/50
337/337 ━━━━━━━━━━━━━━━━━━━━ 14s 43ms/step - accuracy: 0.9360 - loss: 0.1870 - val_accuracy: 0.4992
- val_loss: 2.9436
Epoch 49/50
337/337 ━━━━━━━━━━━━━━━━━━━━ 14s 42ms/step - accuracy: 0.9330 - loss: 0.1968 - val_accuracy: 0.5072
- val_loss: 2.5851
Epoch 50/50
337/337 ━━━━━━━━━━━━━━━━━━━━ 14s 43ms/step - accuracy: 0.9388 - loss: 0.1843 - val_accuracy: 0.5171
- val_loss: 2.6766
```

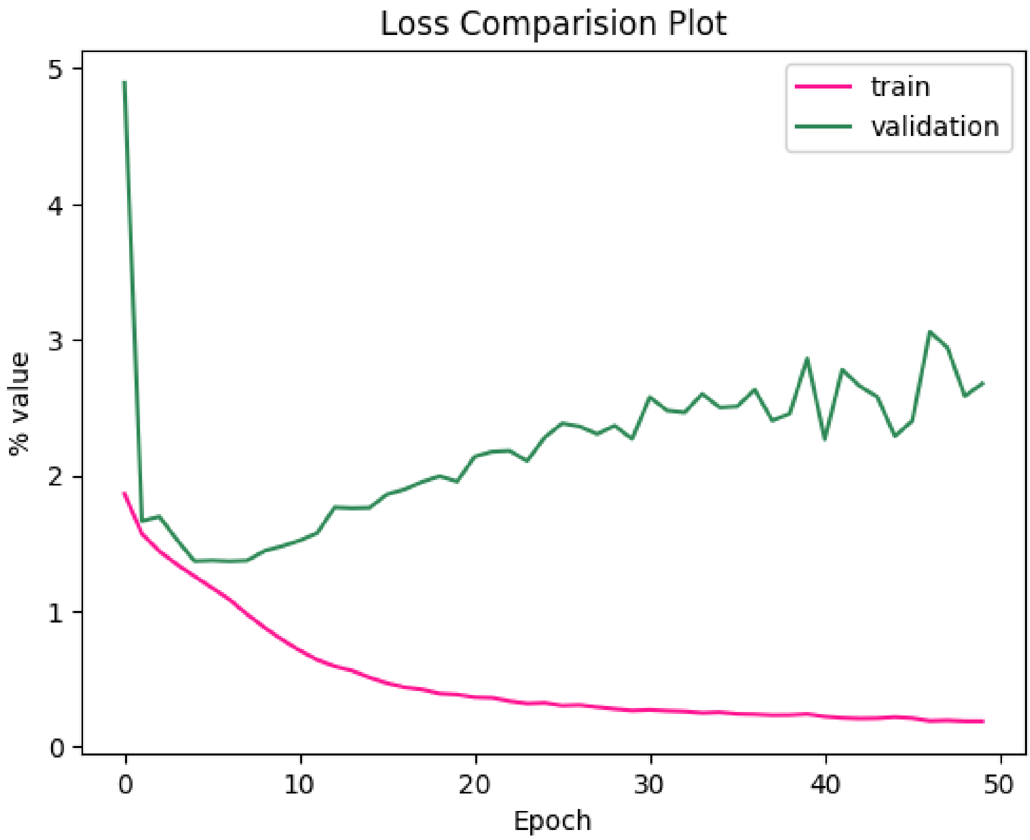
In [51]:

```python
# Accuracy comparison chart
plt.plot(history.history['accuracy'],color='deeppink',label='train')
plt.plot(history.history['val_accuracy'],color='seagreen',label='validation')
plt.title('Accuracy Comparision Plot')
plt.ylabel('% value')
plt.xlabel('Epoch')
plt.legend()
plt.show()
```



In [52]:

```python
# Loss comparison chart
plt.plot(history.history['loss'],color='deeppink',label='train')
plt.plot(history.history['val_loss'],color='seagreen',label='validation')
plt.title('Loss Comparision Plot')
plt.ylabel('% value')
plt.xlabel('Epoch')
plt.legend()
plt.show()
```

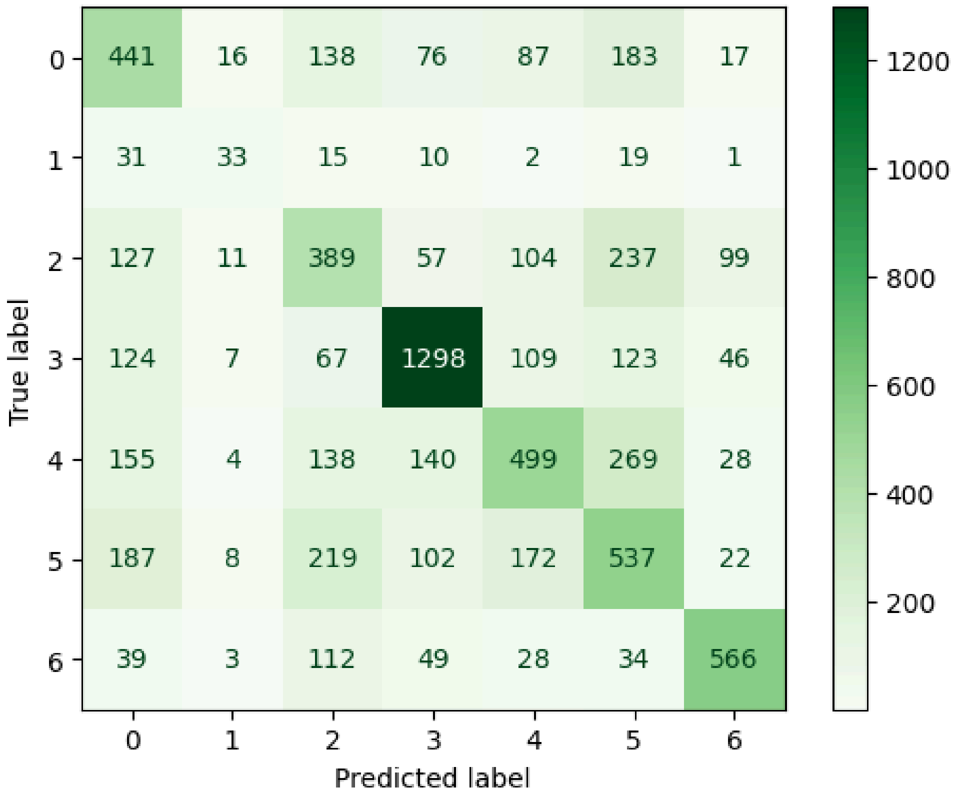


In [56]:
```
# Testing and evaluation
from sklearn import metrics # Import metrics module

y_pred = model.predict(test)
y_pred_labels = []
for i in y_pred:
    y_pred_labels.append(np.argmax(i))
y_actual = test.classes[test.index_array]

cm = metrics.confusion_matrix(y_actual, y_pred_labels)
disp = metrics.ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap=plt.cm.Greens)
plt.show()
```

```
113/113 ━━━━━━━━━━━━━━━━━━━━ 5s 47ms/step
```



In [58]:
```
from sklearn.metrics import classification_report # Import classification_report

print(classification_report(y_actual, y_pred_labels, digits=4))
```

```
              precision    recall  f1-score   support

           0     0.3995    0.4603    0.4277       958
           1     0.4024    0.2973    0.3420       111
           2     0.3609    0.3799    0.3701      1024
           3     0.7494    0.7317    0.7404      1774
           4     0.4985    0.4047    0.4467      1233
           5     0.3830    0.4306    0.4054      1247
           6     0.7266    0.6811    0.7031       831

    accuracy                         0.5242      7178
   macro avg     0.5029    0.4837    0.4908      7178
weighted avg     0.5325    0.5242    0.5267      7178
```

In [152…
```
### Category recognition
train_gen = train_idg.flow_from_directory(
    '/content/data/emotion_fer/train',
    target_size=(100,100),
    batch_size=32,
    class_mode='categorical'
)

print(train_gen.class_indices)
```

```
Found 28709 images belonging to 7 classes.
{'angry': 0, 'disgusted': 1, 'fearful': 2, 'happy': 3, 'neutral': 4, 'sad': 5, 'surprised': 6}
```

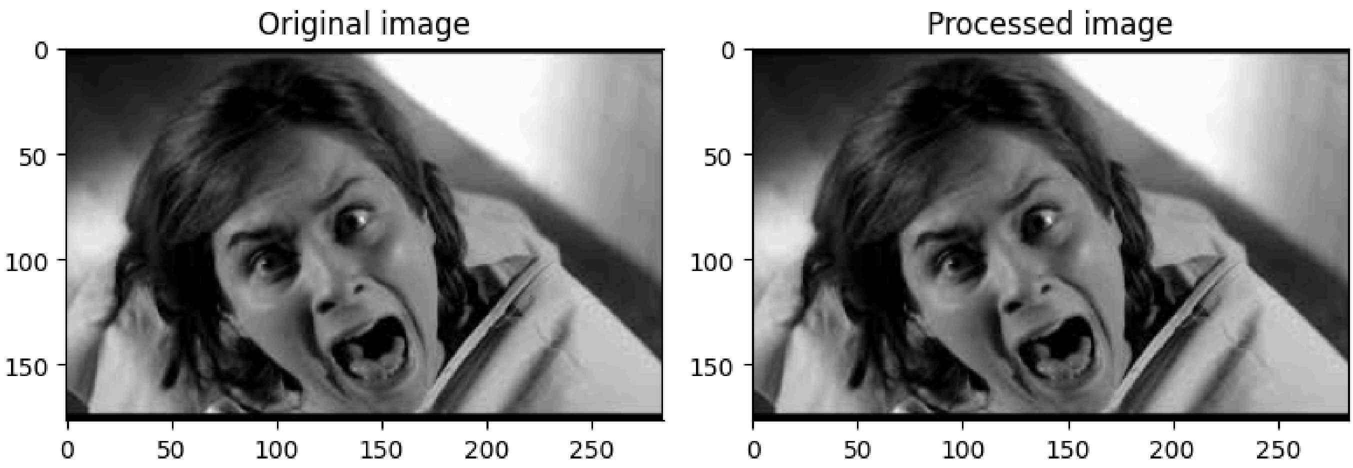
Found 28709 images belonging to 7 classes.

{'angry': 0, 'disgusted': 1, 'fearful': 2, 'happy': 3, 'neutral': 4, 'sad': 5, 'surprised': 6}

Prediction-1

In [162…
```
import cv2
from matplotlib import pyplot as plt
```

In [84]:
```
#reading the original image
test_img2 = cv2.imread('/content/2_fearful_face.jpeg')
#converting the image into grayscale
grey_img2  = rgb2gray(test_img2)
fig, axes = plt.subplots(1, 2, figsize=(8, 4))
ax = axes.ravel()
#setting the axes of the image
ax[0].imshow(test_img2) # Use test_img1 for the original image
ax[0].set_title("Original image")
ax[1].imshow(grey_img2, cmap=plt.cm.gray) # Use grey_img1 for the grayscale image
ax[1].set_title("Processed image")
#display the processed image
fig.tight_layout()
plt.show()
```



In [85]:
```
grey_img2_resized = cv2.resize(grey_img2, (100, 100))
test_input2 = grey_img2_resized.reshape(1, 100, 100, 1)
test_input2.shape
```

Out[85]:
```
(1, 100, 100, 1)
```

In [86]:
```
model.predict(test_input2)
```

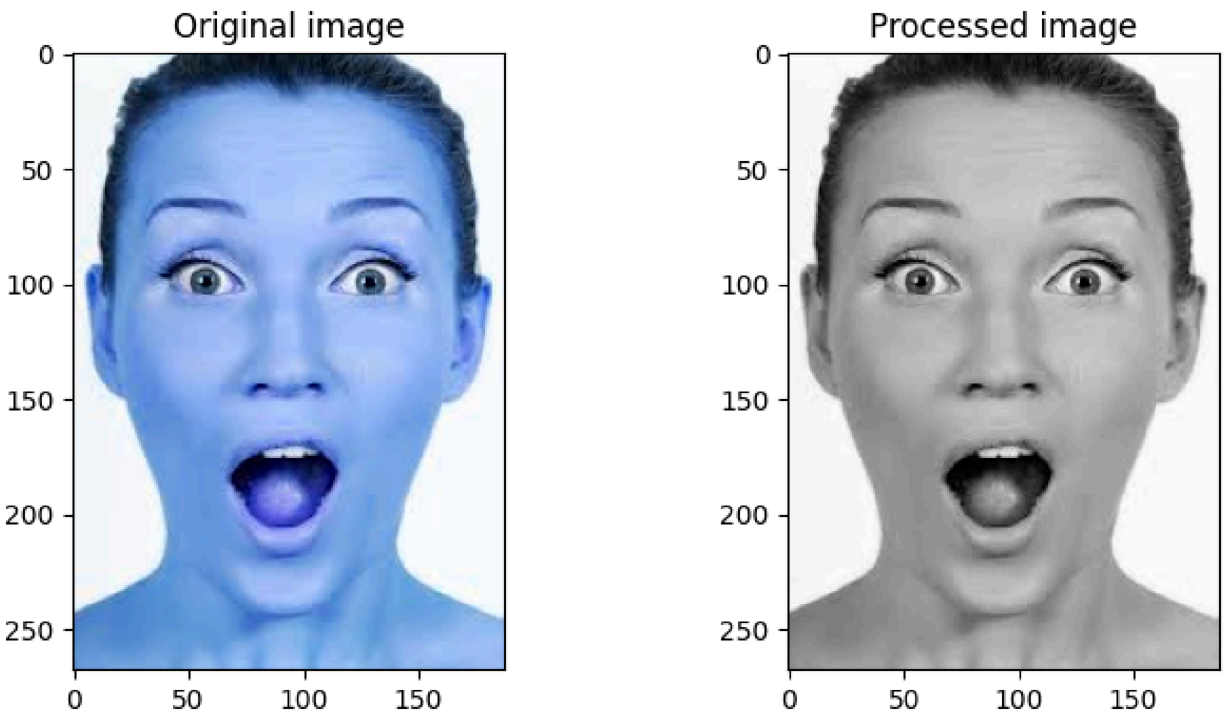
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step
```

Out[86]:
```
array([[0.12131992, 0.13538785, 0.6299236 , 0.04644792, 0.00874567,
        0.04605708, 0.01211797]], dtype=float32)
```

Prediction-2

In [175…
```
#reading the original image
test_img02 = cv2.imread('/content/surprised_face2.jpeg')
#converting the image into grayscale
grey_img02  = rgb2gray(test_img02)
fig, axes = plt.subplots(1, 2, figsize=(8, 4))
ax = axes.ravel()
#setting the axes of the image
ax[0].imshow(test_img02) # Use test_img1 for the original image
ax[0].set_title("Original image")
ax[1].imshow(grey_img02, cmap=plt.cm.gray) # Use grey_img1 for the grayscale image
ax[1].set_title("Processed image")
#display the processed image
```

```
fig.tight_layout()
plt.show()
```



In [176…
```
grey_img02_resized = cv2.resize(grey_img02, (100, 100))
test_input02 = grey_img02_resized.reshape(1, 100, 100, 1)
test_input02.shape
```

Out[176…
```
(1, 100, 100, 1)
```

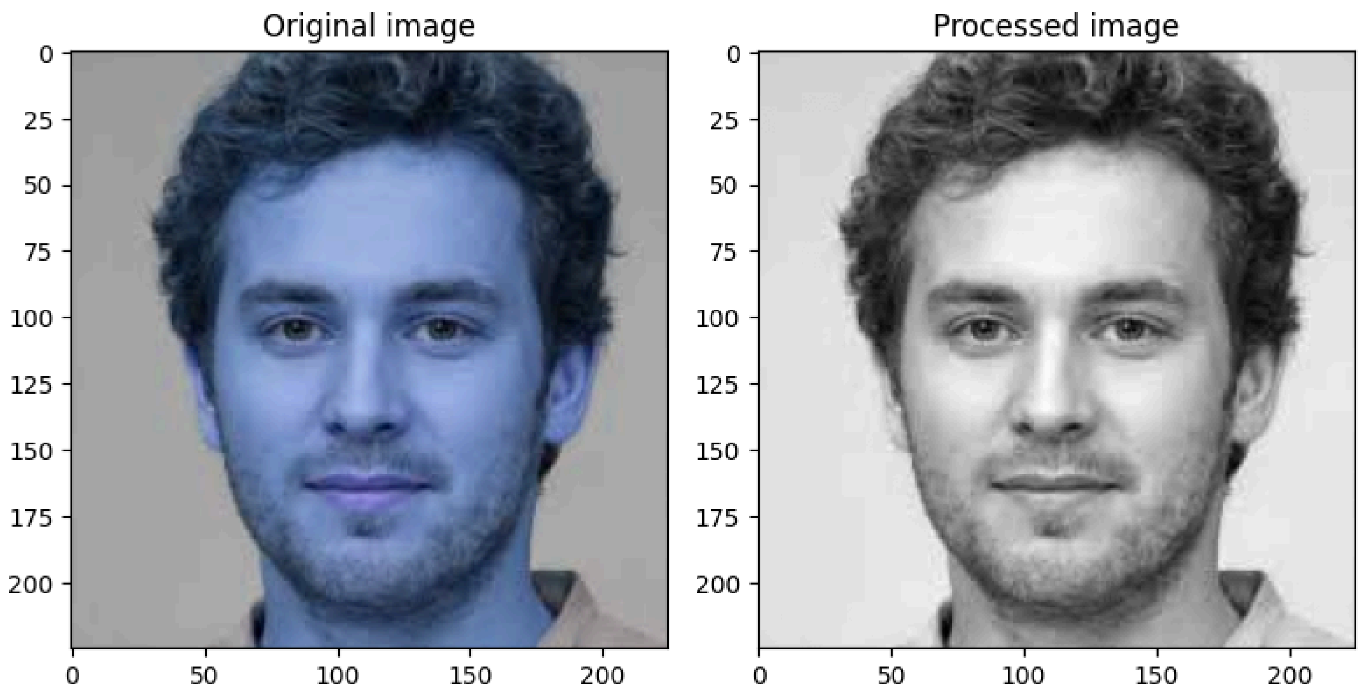
In [177…
```
model.predict(test_input02)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step
```

Out[177…
```
array([[5.4477328e-01, 5.7460260e-02, 3.7919796e-01, 1.7362081e-02,
        1.5343505e-06, 1.6526741e-04, 1.0397012e-03]], dtype=float32)
```

Prediction-3

In [166…
```
#reading the original image
test_img4 = cv2.imread('/content/nuteral_face2.jpeg')
#converting the image into grayscale
grey_img4  = rgb2gray(test_img4)
fig, axes = plt.subplots(1, 2, figsize=(8, 4))
ax = axes.ravel()
#setting the axes of the image
ax[0].imshow(test_img4) # Use test_img1 for the original image
ax[0].set_title("Original image")
ax[1].imshow(grey_img4, cmap=plt.cm.gray) # Use grey_img1 for the grayscale image
ax[1].set_title("Processed image")
#display the processed image
fig.tight_layout()
plt.show()
```



In [169…
```
grey_img4_resized = cv2.resize(grey_img4, (100, 100))
test_input4 = grey_img4_resized.reshape(1, 100, 100, 1)
test_input4.shape
```

Out[169…
```
(1, 100, 100, 1)
```

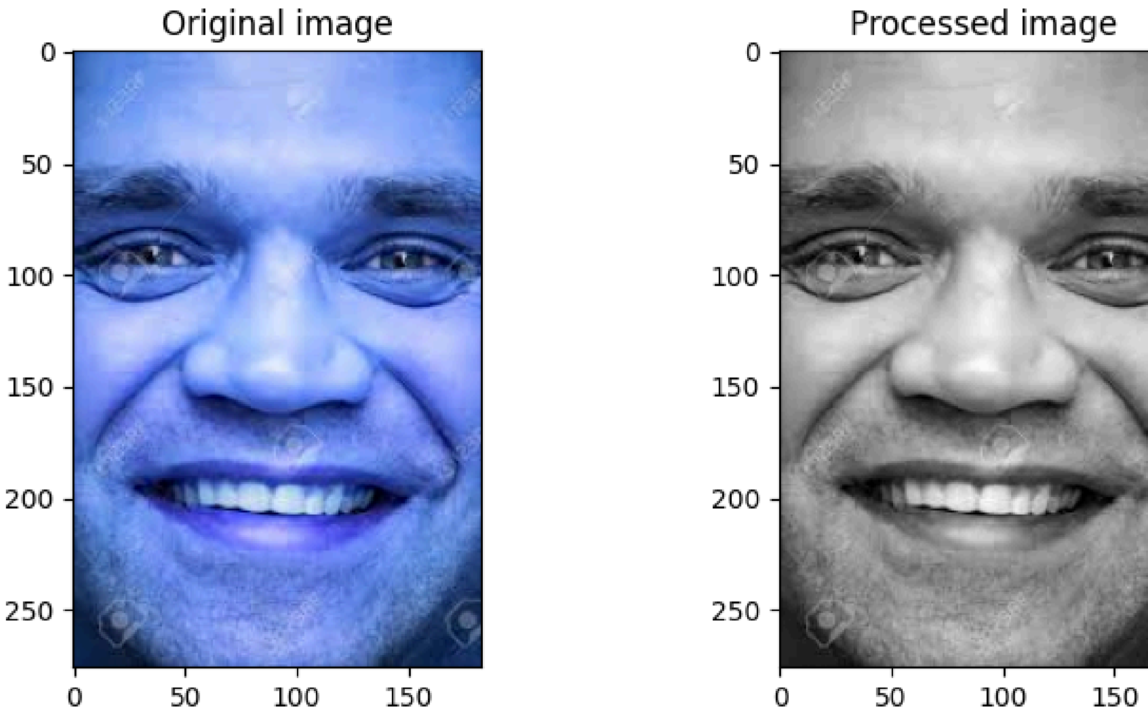
In [170…
```
model.predict(test_input4)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step
```

Out[170…
```
array([[6.9924456e-04, 2.5507687e-14, 6.6499088e-06, 5.0905120e-04,
        9.9419051e-01, 4.0052142e-03, 5.8924936e-04]], dtype=float32)
```

Prediction-4

In [178…
```
#reading the original image
test_img5 = cv2.imread('/content/happy_face4.jpeg')
#converting the image into grayscale
grey_img5  = rgb2gray(test_img5)
fig, axes = plt.subplots(1, 2, figsize=(8, 4))
ax = axes.ravel()
#setting the axes of the image
ax[0].imshow(test_img5) # Use test_img1 for the original image
ax[0].set_title("Original image")
ax[1].imshow(grey_img5, cmap=plt.cm.gray) # Use grey_img1 for the grayscale image
ax[1].set_title("Processed image")
#display the processed image
fig.tight_layout()
plt.show()
```



In [179…

```
grey_img5_resized = cv2.resize(grey_img5, (100, 100))
test_input5 = grey_img5_resized.reshape(1, 100, 100, 1)
test_input5.shape
```

Out[179…

```
(1, 100, 100, 1)
```

In [180…

```
model.predict(test_input5)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
```

Out[180…

```
array([[7.3643422e-01, 5.7453119e-11, 1.4249649e-06, 4.0472839e-02,
        6.7381113e-04, 2.2227617e-01, 1.4158608e-04]], dtype=float32)
```