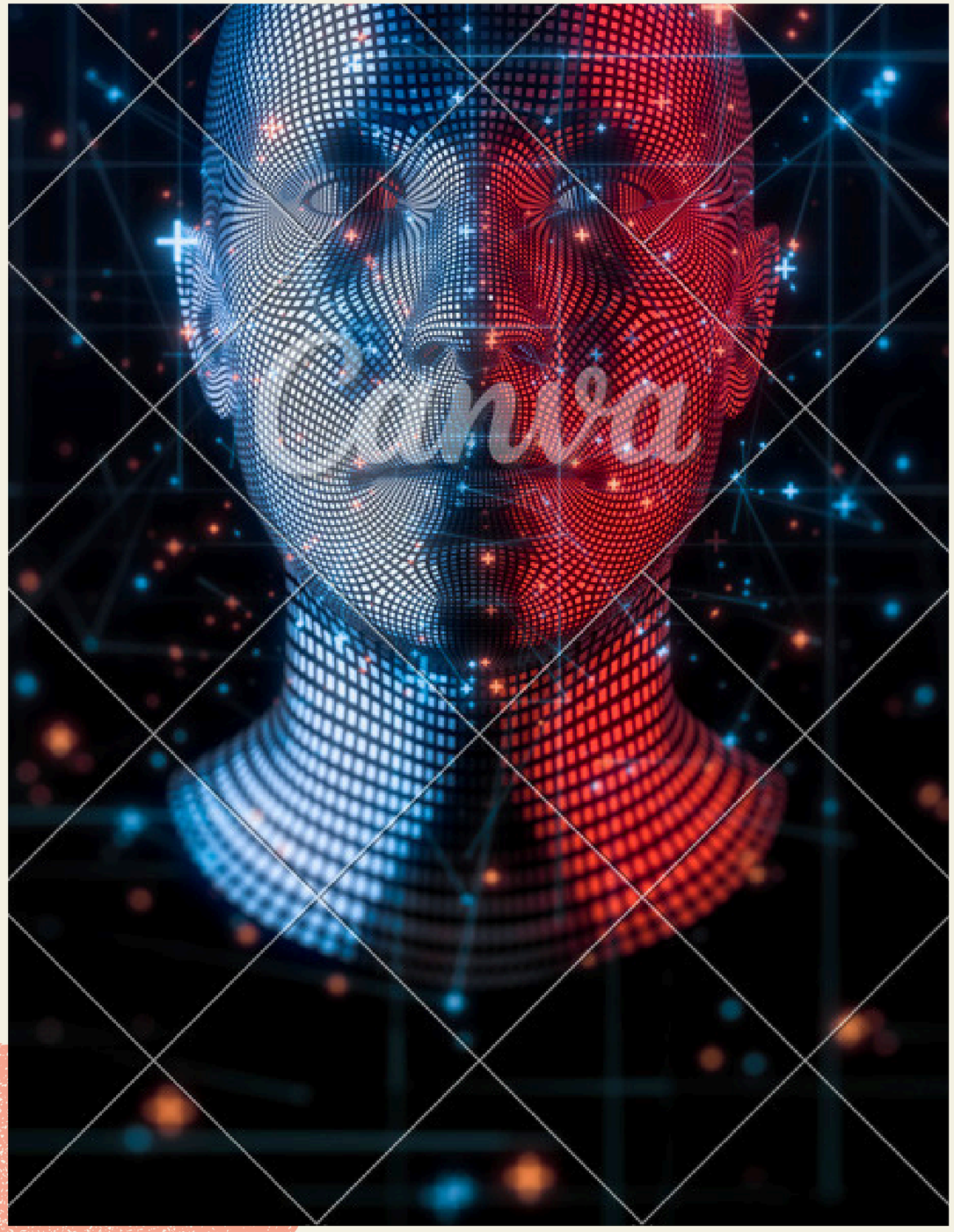
Emotion Detection CNN Classification Project

Exploring Deep Learning in Emotions

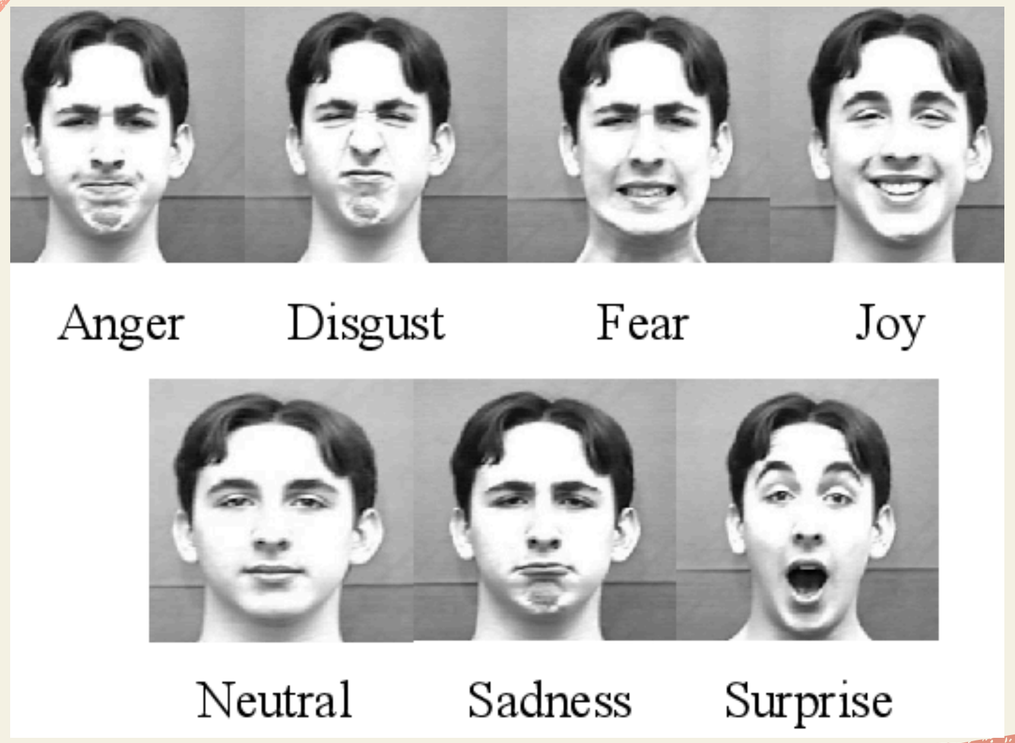


Unveiling Emotion Detection Machine Learning Model

The aim of this project is to create a deep learning model that can correctly identify seven different types of facial expressions: surprised, disgust, fear, happy, neutral, sadness, and anger.

In our method, the CK+ dataset from Kaggle was used to train a convolutional neural networks (CNNs) model.

This model can be deployed for use in customer service visual interactions apps to enable company get an exit emotion of the customer at the end of an engagement to determine their level of satisfaction aside the verbal feedback.



Emotion Detection Overview

Exploring Emotion Detection in Deep Learning Models

1: Data Collection

- The CK+ dataset consists of 48x48 pixel grayscale images of faces.
- The dataset contains 981 images and was obtained from Kaggle

2: Data Preprocessing

- Utilized both data generators and data augmentation to train the model to increase the diversity of the dataset.

3: Model Structure

- Developed the model architecture by adding the following:
- Input layer, Convolutional layer, Max-pooling layers, Batch normalization, Dense layer, Output layer

Model Architecture

Model: "sequential_2"

Layer (type)	Output Shape	Param #
conv2d_4 (Conv2D)	(None, 46, 46, 32)	320
max_pooling2d_4 (MaxPooling2D)	(None, 23, 23, 32)	0
conv2d_5 (Conv2D)	(None, 21, 21, 64)	18,496
batch_normalization_5 (BatchNormalization)	(None, 21, 21, 64)	256
max_pooling2d_5 (MaxPooling2D)	(None, 10, 10, 64)	0
flatten_2 (Flatten)	(None, 6400)	0
dense_6 (Dense)	(None, 128)	819,328
batch_normalization_6 (BatchNormalization)	(None, 128)	512
dense_7 (Dense)	(None, 7)	903

Total params: 839,815 (3.20 MB)

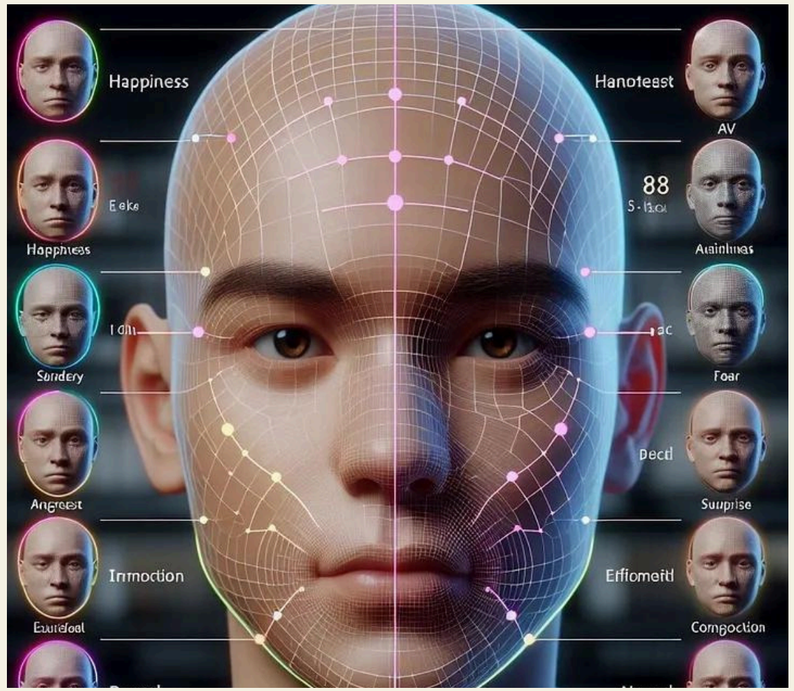
Trainable params: 839,431 (3.20 MB)

Non-trainable params: 384 (1.50 KB)

Enhancements

```
# Defining Callbacks
```

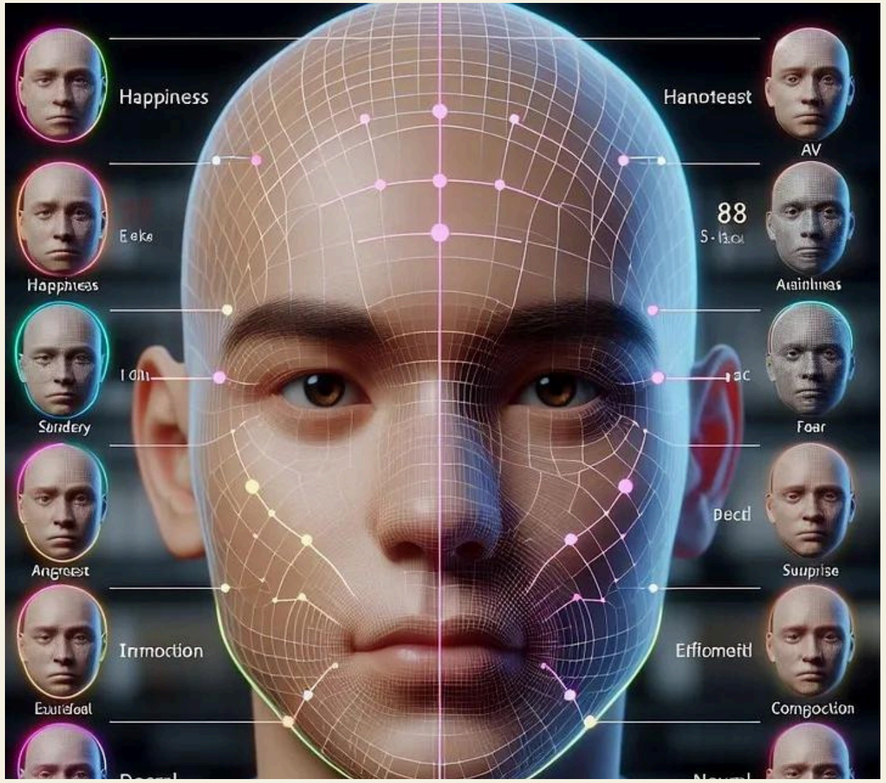
```
# Early stopping to prevent overfitting
early_stopping = EarlyStopping(monitor='val

# Learning rate scheduler
reduce_lr = ReduceLROnPlateau(monitor='val_
```



Training

```
# Training the Model
```

```
# Train the model using data augmentation with callbacks
history = model.fit(datagen.flow(X_train, y_train, batch_size=64),
                    epochs=100,
                    validation_data=(X_test, y_test),
                    callbacks=[early_stopping, reduce_lr])
```



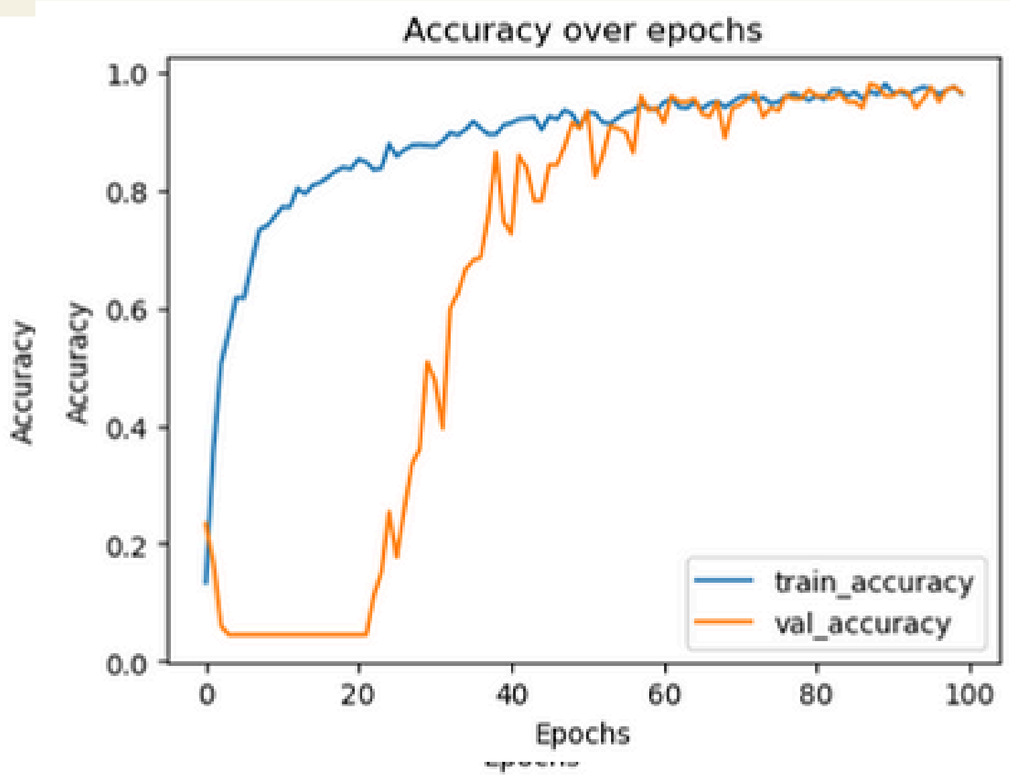
Evaluation

```
# Evaluating the Model
```

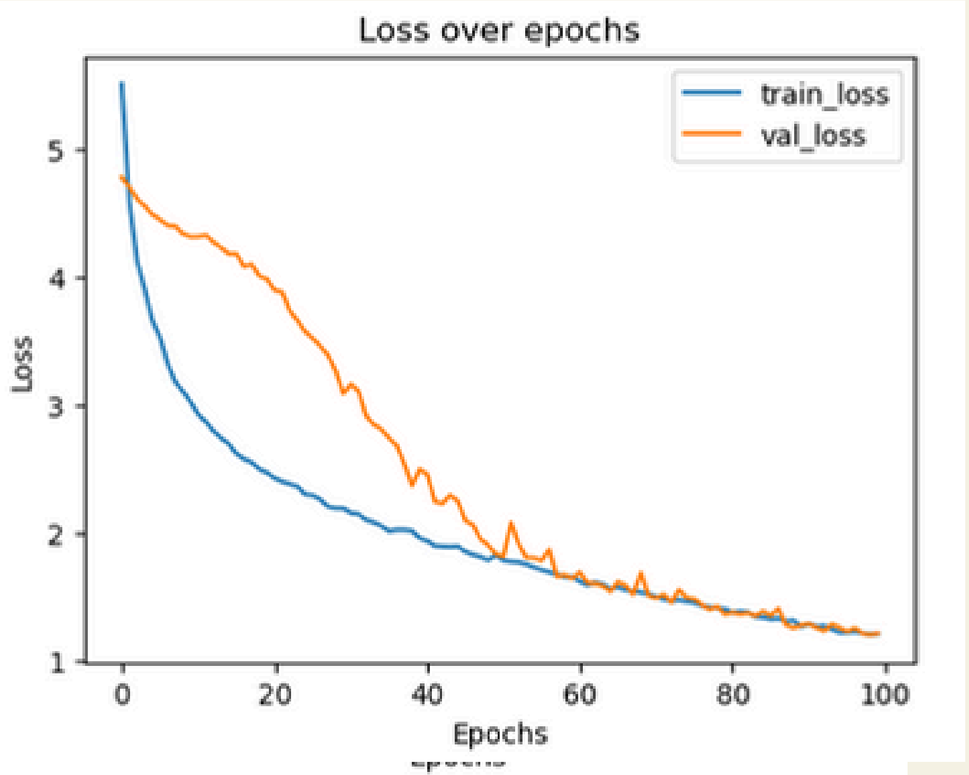
```
# Evaluate the model
train_loss, train_accuracy = model.evaluate(X_train,
print(f'Train accuracy: {train_accuracy*100:.2f}%')

test_loss, test_accuracy = model.evaluate(X_test, y_
print(f'Test accuracy: {test_accuracy*100:.2f}%')
```

```
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy:
Train accuracy: 98.98%
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.
Test accuracy: 96.95%
```

Train Accuracy Vs Val Accuracy



Train Loss Vs Val Loss



98% train accuracy and 96% test accuracy rate achieved

Detect emotions through Deep Learning trained model

981 images analyzed, 80:20 split

Identify various emotional expressions accurately

Reduce processing time efficiently

Enhance user experience significantly

The Phases of Emotion Detection Project

Success and Challenges

Dataset Selection

Selecting the data to train the model effectively and efficiently

Model Architecture and Development

Creating and fine-tuning the model for accurate emotion recognition

Training the Model

Training the model and reviewing the result metrics

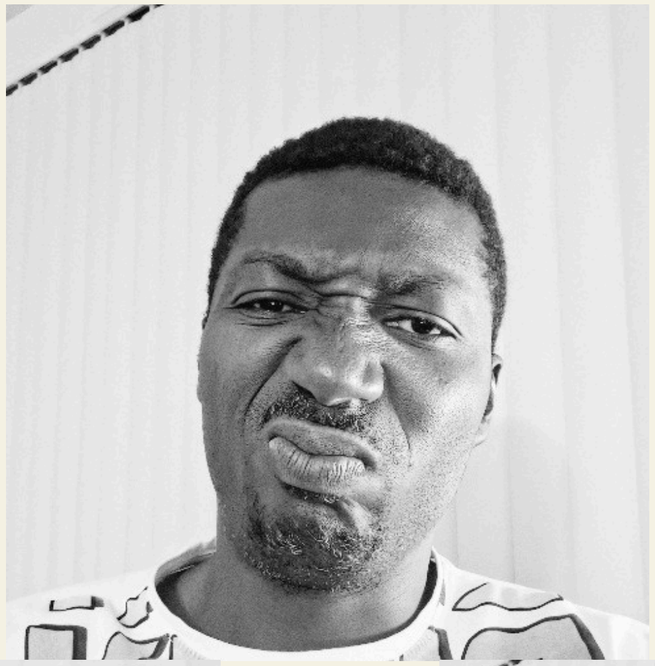
Testing and Validation

Evaluating model performance on various datasets to ensure reliability

"Emotions drive decisions, understand them to succeed."

Unknown, expert on human behavior and psychology

Unseen data used for project demo



Importing Necessary Libraries

[29]:
```
import numpy as np
import os
import cv2
import random
from PIL import Image
import tensorflow as tf
from tensorflow.keras.preprocessing.image import img_to_array, load_img, ImageDataGenerator
from tensorflow.keras.utils import to_categorical, plot_model
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNormalization
from tensorflow.keras.regularizers import l2
from tensorflow.keras.optimizers import Adam
from tensorflow.keras import mixed_precision
from sklearn.model_selection import train_test_split
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
import matplotlib.pyplot as plt
```

Defining Helper Functions

[30]:
```
def find_Class(directory_path):
    if not os.path.exists(directory_path):
        raise ValueError(f"The directory '{directory_path}' does not exist.")
    if not os.path.isdir(directory_path):
        raise ValueError(f"The path '{directory_path}' is not a directory.")

    # Get a list of all entries in the directory
    all_entries = os.listdir(directory_path)

    # Filter out only directories
    folders = [entry for entry in all_entries if os.path.isdir(os.path.join(directory_path, entry))]

    return folders
```

Setting Up Directory and Categories

[31]:
```
DIRECTORY = r'C:/Users/motun/Downloads/CK+_dataset'
CATAGORIES = []
try:
    folders = find_Class(DIRECTORY)
    print(f"Directories in '{DIRECTORY}':")
    for folder in folders:
        CATAGORIES.append(folder)
except ValueError as e:
    print(e)

# Define the image size
image_size = (48, 48)

# Define your categories and directory
CATAGORIES = ['anger', 'contempt', 'disgust', 'fear', 'happy', 'sadness', 'surprise']
DIRECTORY = 'C:/Users/motun/Downloads/CK+_dataset'  # Replace with the path to your images
```

Data Augmentation ¶

[33]:
```
datagen = ImageDataGenerator(
    rotation_range=10,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.1,
    zoom_range=0.1,
    horizontal_flip=True,
    fill_mode='nearest'
)

# Fit the data generator to the training data
datagen.fit(X_train)
```

Enabling Mixed Precision Training

[34]:
```
mixed_precision.set_global_policy('mixed_float16')
```

Loading and Preprocessing Data

[32]:
```
data = []
labels = []

for category in CATAGORIES:
    folder = os.path.join(DIRECTORY, category)
    label = CATAGORIES.index(category)

    for img in os.listdir(folder):
        img_path = os.path.join(folder, img)
        img_arr = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
        if img_arr is not None:  # Check if the image is successfully loaded
            img_arr = cv2.resize(img_arr, (48, 48))  # Resize to 48x48
            data.append(img_arr)
            labels.append(label)
        else:
            print(f"Failed to load image {img_path}")

data = np.array(data)
labels = np.array(labels)

# Normalize the images
data = data / 255.0

# Add a channel dimension
data = np.expand_dims(data, axis=-1)
```

```
# Convert labels to categorical
labels = to_categorical(labels, num_classes=len(CATAGORIES))

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.2, random_state=42)
```

[35]:
```
# Define the model
model = Sequential()

# Add convolutional layers with input shape specified in the first layer
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(48, 48, 1)))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3), activation='relu', kernel_regularizer=l2(0.01)))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))

# Add a flatten layer
model.add(Flatten())

# Add fully connected layers
model.add(Dense(128, activation='relu', kernel_regularizer=l2(0.01)))
model.add(BatchNormalization())
#model.add(Dropout(0.5))

# Add output layer
model.add(Dense(len(CATAGORIES), activation='softmax'))

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.0001), loss='categorical_crossentropy', metrics=['accuracy'])

# Print the model summary
model.summary()
```

Defining Callbacks

[36]:
```
# Early stopping to prevent overfitting
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)

# Learning rate scheduler
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, min_lr=0.00001)
```

Training the Model

[27]:
```
# Train the model using data augmentation with callbacks
history = model.fit(datagen.flow(X_train, y_train, batch_size=64),
                    epochs=100,
                    validation_data=(X_test, y_test),
                    callbacks=[early_stopping, reduce_lr])
```

Evaluating the Model

[28]:
```
# Evaluate the model
train_loss, train_accuracy = model.evaluate(X_train, y_train)
print(f'Train accuracy: {train_accuracy*100:.2f}%')

test_loss, test_accuracy = model.evaluate(X_test, y_test)
print(f'Test accuracy: {test_accuracy*100:.2f}%')
```

```
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.9876 - loss: 1.3401
Train accuracy: 98.98%
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - accuracy: 0.9626 - loss: 1.4295
Test accuracy: 96.95%
```

Save the model

[11]:
```
# Save the model
model.save('C:/Users/motun/Downloads/saved_model.keras')
```

Load and preprocess the image

[81]:
```
# Load and preprocess the image
from tensorflow.keras.preprocessing import image
img_pred = image.load_img(r"C:\Users\motun\OneDrive\Desktop\Random_test\surprise1.png", target_size=(48, 48),
img_pred = image.img_to_array(img_pred)
img_pred = img_pred / 255.0  # Normalize the image
img_pred = np.expand_dims(img_pred, axis=0)  # Add batch dimension
img_pred = np.expand_dims(img_pred, axis=-1)  # Add channel dimension if missing

# Make prediction
rslt = model.predict(img_pred)

# Print results
print(rslt)
print(CATAGORIES[np.argmax(rslt)])
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 236ms/step
[[0.1426 0.1404 0.1462 0.146  0.1323 0.1449 0.1473]]
surprise
```