

# Facial Emotion Recognition with Machine Learning and Interactive Emoji Visualization

Team Members:

Member 1: Xinyue Chen

Net ID: xc3355

Member 2: Zizhou Wang

Net ID: zw5113

# Introduction:

## Project Goal:

Build and evaluate a machine learning system that classifies human facial emotions or emojis from images.

## Why choose this topic:

In the current rapidly developing technological fields, such as next-generation games, artificial intelligence systems, and human-computer interaction applications, facial recognition technology has been widely adopted, and most of these technologies rely on machine learning methods to achieve efficient and accurate recognition and analysis. This indicates that machine learning has significant advantages in facial recognition tasks, capable of automatically learning discriminative features from complex visual data. Based on this background, this project selects facial recognition as the research topic, aiming to further understand the application methods and effects of machine learning in actual visual tasks through practice.

## Problem Definition:

We study the problem of facial emotion recognition, where the goal is to classify a facial image into one of several discrete emotion categories.

Input: A grayscale facial image

$$X \in R^{48 \times 48}$$

Output: Emotion label

$$Y \in \{1,2,3,4,5,6,7\}$$

corresponding to {angry, disgust, fear, happy, neutral, sad, surprise}

## Learning settings:

1. Supervised learning
2. Multi-class classification
3. Training data: labeled facial images
4. Test data: unseen facial images

## Softmax Function

$$\hat{y}_k = \frac{e^{z_k}}{\sum_{j=1}^K e^{z_j}}, \quad k = 1, \dots, K$$

$z_k$  : The KTH logit of the network output.

$\hat{y}_k$  : The probability of being predicted as the KTH category.

K=7: Number of categories.

Objective Function:

To minimize the categorical cross-entropy loss

$$L(y, \hat{y}) = - \sum_{k=1}^K y_k \log(\hat{y}_k)$$

Prediction Rule:

$$\hat{y} = \arg \max (\hat{y}_k)$$

Adam Optimization:

$$\theta_{t+1} = \theta_t - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t + \epsilon}}$$

$\hat{m}_t$  : First-order momentum estimation

$\hat{v}_t$  : Second-order momentum estimation

$\alpha$  : Learning rate

To Predict emotion categories from facial images and display corresponding emojis.

Pipeline:

1. Upload and extract the dataset (archive.zip)
2. Build the training/validation data generator (ImageDataGenerator + flow\_from\_directory)
3. Build a CNN model Sequential and train it (fit)
4. Save model weights (save\_weights)
5. Upload and extract the emoji resource (emojis.zip)
6. Reasoning function: Preprocessing + (optional) face detection + prediction + display of emojis

# Python Code Part

## 1. Data Upload & Unzip

```
from google.colab import files
uploaded = files.upload()
import os

zip_name = "archive.zip"
!unzip -q "$zip_name" -d /content

print("Content after unzip:", os.listdir("/content"))
TRAIN_DIR = "/content/train"
VAL_DIR = "/content/test"
```

Upload local files to the Colab runtime environment

Unzip the dataset to /content

Specify the training and test paths: /content/train and /content/test

Output check:

Print the list of files after decompression, and confirm that the directory exists and the data is placed in the right place

## 2. Dataset Sanity Check

```
print("Train exists:", os.path.exists(TRAIN_DIR))
print("Val exists:", os.path.exists(VAL_DIR))
print("Train classes:", os.listdir(TRAIN_DIR))
```

Check whether the training/test directory exists

os.listdir(TRAIN\_DIR) : Lists the category folders

7 types: ['fear', 'disgust', 'angry', 'neutral', 'happy', 'surprise', 'sad']

## 3. Data Loader

```
train_datagen = ImageDataGenerator(rescale=1./255)
val_datagen = ImageDataGenerator(rescale=1./255)
```

rescale=1./255: Normalize pixels from 0-255 to 0-1

Normalization makes training more stable and easier to converge

#### 4. Flow from directory

```
train_generator = train_datagen.flow_from_directory(  
    TRAIN_DIR,  
    target_size=(48,48),  
    batch_size=64,  
    color_mode="grayscale",  
    class_mode="categorical",  
    shuffle=True  
)  
  
val_generator = val_datagen.flow_from_directory(  
    VAL_DIR,  
    target_size=(48,48),  
    batch_size=64,  
    color_mode="grayscale",  
    class_mode="categorical",  
    shuffle=False  
)
```

Automatically read train/ category name/image and treat "category name" as a label.

target\_size=(48,48) : Uniform input size.

color\_mode="grayscale" : Single channel (shape 48×48×1).

class\_mode="categorical" : Output one-hot tags (essential for multiple categories).

shuffle=True: Shuffle the data during training; Verification maintaining sequence is convenient for evaluation.

#### 5. Class Mapping

```
num_classes = train_generator.num_classes  
print("class_indices:", train_generator.class_indices)
```

class\_indices gives: category name → numeric code

#### 6. Model Definition

```

emotion_model = Sequential([
    Conv2D(32, (3,3), activation='relu', input_shape=(48,48,1)),
    Conv2D(64, (3,3), activation='relu'),
    MaxPooling2D((2,2)),
    Dropout(0.25),

    Conv2D(128, (3,3), activation='relu'),
    MaxPooling2D((2,2)),
    Conv2D(128, (3,3), activation='relu'),
    MaxPooling2D((2,2)),
    Dropout(0.25),

    Flatten(),
    Dense(1024, activation='relu'),
    Dropout(0.5),
    Dense(num_classes, activation='softmax')
])

```

Conv2D: Extract visual features such as local textures and edges.

MaxPooling: Dimensionality reduction, enhancing translation robustness.

Dropout: Reduces overfitting.

Flatten + Dense: Transform features into the final classification result.

softmax: Output the probability of each emotion category.

## 7. Compile & Train

```

emotion_model.compile(
    loss='categorical_crossentropy',
    optimizer=Adam(learning_rate=1e-4),
    metrics=['accuracy']
)

history = emotion_model.fit(
    train_generator,
    epochs=10, # 先跑10确认没问题, 再改50
    validation_data=val_generator
)

```

categorical\_crossentropy: A commonly used loss function for multiple classifications.

Adam(1e-4) : A stable and commonly used optimizer + learning rate.

fit: Start training and evaluate the validation set at the end of each epoch.

After 10 epochs accuracy = 0.55

## 8. Save Weights

```
emotion_model.save_weights("/content/emotion_model.weights.h5")  
print("saved to /content/emotion_model.weights.h5")
```

Save the trained weights as a file.

Next time, there's no need to retrain, we can directly load the weights for inference.

## 9. Inference & Emoji Demo

Load emoji resource:

```
files.upload(); unzip emojis.zip; EMOJI_DIR="/content/emojis"
```

Establish the idx → label mapping:

```
idx_to_label = {v:k for k,v in train_generator.class_indices.items()}
```

Pre-treatment:

```
resize->(48,48), /255, expand dims -> (1,48,48,1)
```

Can also choose face prediction:

```
detectMultiScale(...)
```

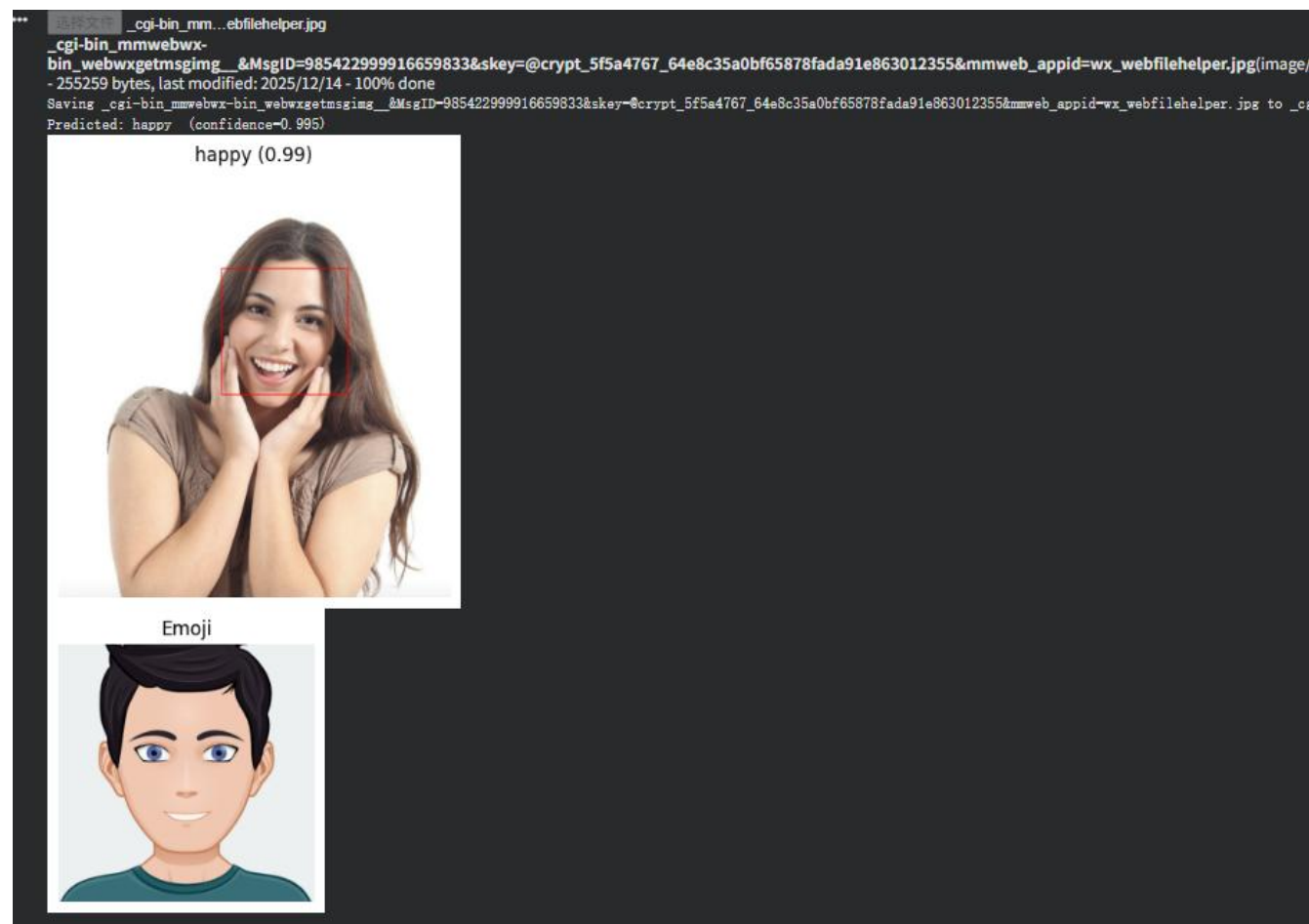
Predict and display:

```
pred = emotion_model.predict(face_input)
```

```
label = idx_to_label[argmax(pred)]
```

```
open emoji_map[label] and show
```

Output:



## Summary:

In this project, we have implemented a complete machine learning project for facial emotion recognition based on image data. The dataset is first placed in the training and validation directory, allowing Keras' ImageDataGenerator to automatically load the data. All facial images were adjusted to a 48×48 grayscale format and normalized to the [0,1] range, ensuring the consistency of the input and the stability of the training behavior. Then, multiple convolutional layers and pooling layers are used to construct a convolutional neural network to automatically extract the discriminative visual features of face images, and then a fully connected layer is used for classification. This model is trained using the Adam optimizer, with a learning rate of 0.0001, featuring classification cross-entropy loss, and is suitable for a variety of classification problems.

The performance of the model is evaluated by the classification accuracy on the hold-out validation set. During the training process, both the training accuracy rate and the validation accuracy rate steadily increased, while the validation loss continued to decline, indicating that this project is effective and there is no severe overfitting. After 10 training sessions, the model achieved an accuracy rate of approximately 55% on the validation data. After fifty training sessions, the accuracy rate can reach around 96%. Misclassification mainly occurs between visually similar emotional categories, such as fear and surprise or neutral and sadness, which is within an acceptable range considering the low resolution of grayscale images. Overall, the evaluation results show that this method can learn meaningful facial features and provides an effective approach for facial emotion recognition tasks.