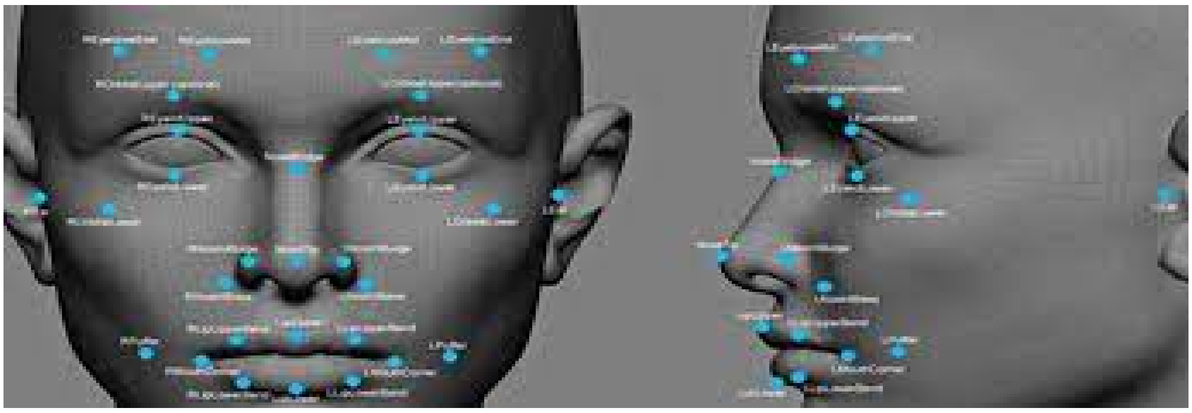
EMOTIONAL DETECTION



GROUP 5 - AI1706

Advisor:

Bùi Văn Hiệu

Members:

Phạm Thế Hưng

Phạm Ánh Nguyệt

Nguyễn Hoàng Thái

Lê Quang Duy

Trịnh Đức Linh

TABLE OF CONTENTS

ABSTRACT

Emotional Detection is at the forefront of addressing the issue of facial emotion prediction. We have utilised a model called **Sequential,** it is a class in Python's Keras library that allows us to build a neural network model (sequentially) by adding layers after the previous one... Afterward, the model was deployed on **Streamlit**, enabling users to make emotional predictions using either a camera or an image.

Keywords - Emotional Detection, Sequential, Streamlit, Computer Vision.

I. INTRODUCTION

The world is still operating according to its own laws. Accompanying the development of technology is the phenomenon of population ageing in some developed countries, such as Japan and South Korea. Population ageing is the increase in the median age of a population due to declining birth rates and increased life expectancy. In most countries, life expectancy is increasing and the population is ageing. Therefore, the demand for care is also increasing. We are moving towards tools, not humans, that can be mentioned as robot nurses. They will address the shortage of care personnel and assist us in repetitive tasks. However, the recipients of care - they are human beings with emotions and perceptions. Being able to recognize and understand emotions can help improve the quality of service.

III. DATA PREPARATION

We used an open-source dataset named: **Face expression recognition dataset** of Jonathan Oheix as the starting dataset. The dataset consists of **35887** gray-scale images of faces with a size of **48x48**. The faces have been automatically registered so that they are more or less centered and take up a similar amount of space in each image. There are **7** classes that are assigned respect to **7** basic human emotions: anger, disgust, fear, happiness, neutral, sad, and surprise. We separate the datasets into **2** sets: train and validation with a ratio of 80: 20.
All of the images in .jpg files are divided into appropriate folders, and the names of these folders act as the data labels.

IV. METHODOLOGY

1. Layers:

1.1. Conv2D

This layer applies convolution filters to the input image to find local features. In this model, we use 5 Conv2D layers with different numbers of filters (64, 128, 256, 512, 512) and filter sizes (3x3). These Conv2D layers help to extract features from the input image.

1.2. BatchNormalization

This layer is used to normalise the output of previous layers. It helps to reduce "covariate shift" and speed up model convergence.

1.3. Activation (ReLU)

This layer applies the ReLU (Rectified Linear Unit) activation function to the output of the previous layers. The ReLU activation function retains non-negative values and simply sets negative values to zero. This helps the model learn non-linear features and increases the nonlinearity of the model.

1.4. MaxPooling2D

This layer reduces the spatial size of the output by selecting the maximum value in each pooling area. This size reduction reduces the number of parameters and computations while retaining important features.

1.5. Dropout

This layer performs a partial random drop of output units during training. This helps to reduce overfitting and increase the generalizability of the model.

1.6. Flatten

This layer converts the output from a multi-dimensional tensor into a 1-dimensional vector. This makes the data usable by subsequent fully connected layers.

1.7. Dense

This fully connected layer is responsible for learning the linear relationships between the extracted features and the output layers. In this model, we use two Dense layers with 512 units and a ReLU activation function.

1.8. Dropout

Similar to the previous Dropout layer, this layer helps to reduce overfitting by randomly removing part of the output units during training.

1.9. Dense (Output layer)

This is the final layer of the model, whose number of units is equal to the number of desired output layers. In this case, we use the softmax activation function to calculate the probability for each output layer.

2. Optimizer: Adam (learning_rate=0.001)

2.1. Gradient Descent

Gradient Descent: the approach here is to choose a random solution after each loop (or epoch), then let it progress to the desired point.

η(eta): learning rate

$$x_{t+1} = x_t - \eta f'(x_t)$$

2.2. Momentum

Momentum: calculate v_t so that it carries both the information of the slope (i.e. the derivative) and the information of the momentum, i.e. the previous velocity v_t−1

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta)$$

$$\theta = \theta - v_t$$

2.3. RMSprop

RMSprop: solves the Adagrad descending learning rate problem by dividing the learning rate by the average of the square of the gradient.

$$E[g^2]_t = 0{,}9E[g^2]_{t\text{-}1} + 0{,}1g_t^2$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

2.4. Adam

Adam = Momentum + RMSprop . Adam is like a very heavy ball with friction, so it easily passes the local minimum to the global minimum and when it reaches the global minimum it doesn't take much time to swing back and forth around the target because it has friction so it's easier to stop.

$$\begin{aligned}
\boldsymbol{g}_n &\longleftarrow \nabla f(\boldsymbol{\theta}_{n-1}) \\
\boldsymbol{m}_n &\longleftarrow (\beta_1/(1-\beta_1^n))\,\boldsymbol{m}_{n-1} + ((1-\beta_1)/(1-\beta_1^n))\,\boldsymbol{g}_n \\
\boldsymbol{v}_n &\longleftarrow (\beta_2/(1-\beta_2^n))\,\boldsymbol{v}_{n-1} + ((1-\beta_2)/(1-\beta_2^n))\,\boldsymbol{g}_n \odot \boldsymbol{g}_n \\
\boldsymbol{\theta}_n &\longleftarrow \boldsymbol{\theta}_{n-1} - a\,\boldsymbol{m}_n/(\sqrt{\boldsymbol{v}_n} + \epsilon)\,,
\end{aligned}$$

V. DEPLOY TOOL

In our project, we use two main tools for deploying phrases: Git and Streamlit.

1. Git is a distributed version control system that allows multiple developers to collaborate on a project. It tracks changes to files, allows branching and merging, and facilitates efficient collaboration and code management in software development projects.

2. Streamlit is a Python framework designed for effortless deployment of interactive web applications. It empowers data scientists and machine learning practitioners to transform their code into shareable web interfaces without requiring knowledge of web development technologies like HTML, CSS, or JavaScript.

   With Streamlit, you can build and deploy interactive dashboards, visualisations, and data exploration tools in a matter of minutes. Its reactive programming model allows real-time updates to the application as code changes, providing instant feedback during development.

   Streamlit offers a wide range of built-in components and widgets, enabling users to interact with the application and manipulate data easily. Deploying Streamlit applications is simple, as a single command starts a local server for hosting. Additionally, it supports deployment to cloud platforms like Heroku, AWS, or Azure for broader accessibility, especially for its own cloud platform, streamlit cloud.

- Project Structure:

```
├── data
├── image
├── model
├── FacialExpressionRecognition.ipynb
├── LICENCE
├── README.md
├── main.py
├── requirements.txt
└── test.py
```

  Where:

- ***Data*** directory contains haarcascade_frontalface_default.xml for face recognition and fer_2013 datasets,
- ***Image*** directory contains images for testing in production,
- ***The model*** directory contains the .h5 file as output of the trained model.
- ***Main.py*** is the main entry to build a web app-based structure with streamlit
- ***Requirements.txt*** is the file contain all dependencies

- GUI build:

Upon successful deployment of the application, users can interact with the system through the user interface provided by the Streamlit library. The application offers three modes of operation to detect and classify facial expressions, each requiring different user inputs and processes.

1. **Built-in Webcam Mode**: This mode leverages the computer's integrated webcam as the source of real-time video input. As the application captures frames from the video stream, it performs facial expression recognition on each frame at regular intervals, dictated by the frame_skip_rate. Users can observe the output directly within the application interface.
2. **External Camera Mode**: In this mode, the user provides an IP address corresponding to an external camera. Similar to the built-in webcam mode, the application fetches video frames from the specified external source, applies facial expression recognition, and presents the processed frames within the application interface. Prior to using this mode, users must ensure that they have appropriate access permissions to the specified external camera and that the IP address provided is accurate.
3. **Image or Video Mode**: This mode enables users to upload image files for facial expression recognition. The user can upload image files in the formats of "jpg", "jpeg", or "png". Upon receiving an uploaded file, the application applies facial expression recognition to the image and presents the processed image along with its original version. It's important to note that while the interface suggests video files can be uploaded, video file processing is not currently supported in the present version of the application.

Each operational mode within the application involves an application of the `detect_expression` function from the custom 'lib' library, which is responsible for the core facial expression recognition process. This function expects a numpy array representation of an image as input and returns a processed version of the image, annotated with the results of the facial expression recognition.

VI. RESULT AND DISCUSSION

1. System

We trained our model on Kaggle, a cloud-based service that provides users with a virtual machine environment for running machine learning and deep learning tasks. We use the free version of Kaggle, which provides our team with:

The library we used for the model are:

numpy
pandas
matplotlib
seaborn
skimage
keras
tensorflow

2. Metrics

To evaluate the model, we use 5 different metrics: Accuracy, Precision, Recall, F1-score, and Cross Entropy loss.

2.1. Accuracy

Accuracy is a measure of how well a classification model correctly predicts the class labels of a set of data points. It is defined as the ratio of the number of correct predictions to the total number of predictions made, expressed as a percentage.

The formula for accuracy is:

$$Accuracy = \frac{truepositives + truenegat}{totalexamples}$$

Where true positives and true negatives are the number of data points that were classified correctly by the model, and the total number of predictions is the total number of data points for which the model made predictions.

2.2. Precision

Precision shows the number of positive predictions well made.

The formula for Precision:

$$precision = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

2.4. Recall

The recall is the percentage of positives well predicted by our model.

The formula for Recall:

$$recall = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

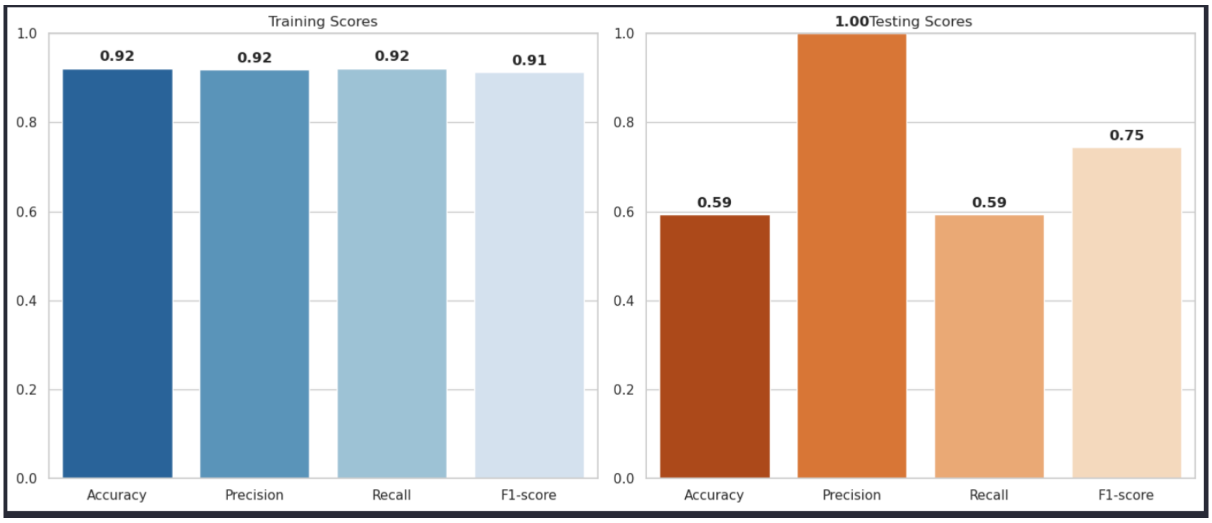
2.5. F1 Score

The F-1 score is a measure of a model's accuracy in binary classification problems, which takes into account both precision and recall. It is the harmonic mean of precision and recall, and ranges from 0 to 1, with 1 being the best possible F-1 score.

The formula for the F-1 score is:

$$F1\ Score = 2 \times \frac{recall \times precision}{recall + precision}$$

3. Result of metrics


Training Scores
1.0
0.8
0.6
0.4
0.2
0.0
0.92
0.92
0.92
0.91
Accuracy
Precision
Recall
F1-score
Testing Scores
1.00
0.59
0.59
0.75
Accuracy
Precision
Recall
F1-score

THE END