VIETNAM ACADEMY OF SCIENCE AND TECHNOLOGY
UNIVERSITY OF SCIENCE AND TECHNOLOGY OF HANOI

Course: Machine Learning and Data Mining

Machine Learning Group Report

Project Title: Face Emotion Recognition Using Machine Learning

Group Members:

1, Nguyễn Đức Tâm - 22BI13400

2, Trần Thọ Thăng - 22BI13404

3, Đặng Trần Trung Thành – 22BI13406

4, Nguyễn Quốc Thành – 22BI13410

5, Hoàng Kim Thành – 22BI13407

I. Introduction:

The project focused on implementing machine learning techniques for face emotion recognition. Emotion recognition from facial expressions plays a significant role in various fields such as human-computer interaction, healthcare, and security systems. In this project, we explored different machine learning algorithms and methodologies to develop a robust emotion recognition system.

II. Objectives:

1. To collect and preprocess a dataset of facial expressions.

2. To explore and implement various machine learning algorithms for emotion recognition.

3. To evaluate the performance of the implemented models using appropriate metrics.

4. To create a user-friendly interface for real-time emotion recognition.

III. Methodology:

1. **Data Collection**: We collected a dataset of facial images labeled with different emotions.

2. **Preprocessing**: The collected data underwent preprocessing steps, including face detection, alignment, and normalization.

3. **Feature Extraction**: Various features were extracted from the preprocessed images, including facial landmarks, texture features, and deep learning embeddings.

4. **Model Selection**: We apply **Convolutional Neural Networks (CNN)** model to implement this project.

5. **Model Training**: The selected models were trained on the extracted features using appropriate training strategies and hyper parameters.

6. **Evaluation**: The trained models were evaluated on a separate test dataset using metrics such as accuracy, precision, recall, specifier and F1-score.

IV. Implementation:

1. Set up:

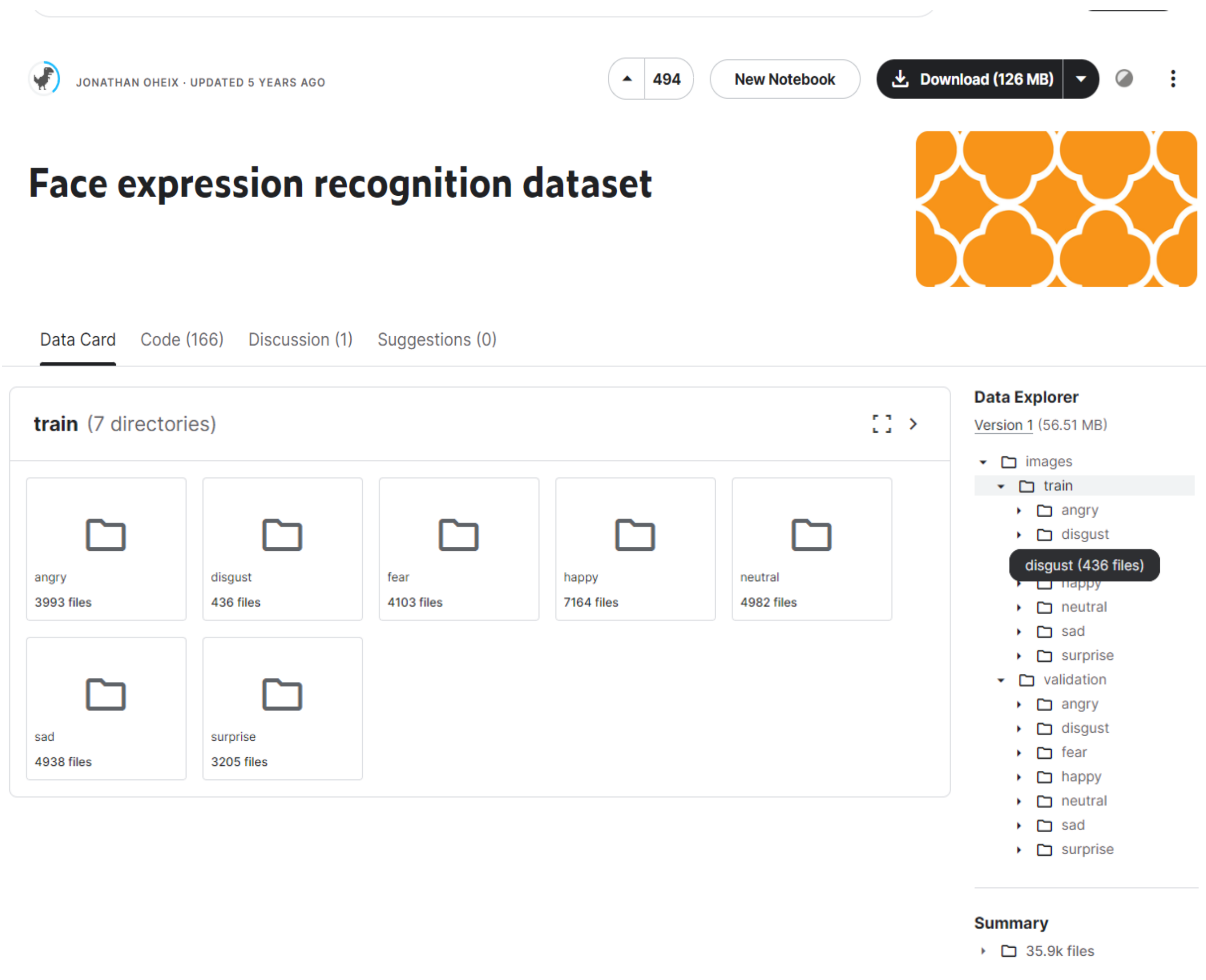
Install all the following module in your IDE:

requirement.txt

```
tensorflow
keras
numpy
tqdm
sklearn
pandas
matplotlib
opencv-contrib-python
```

2. Prepare for dataset:

Dataset we apply on project (available dataset):



The dataset is divided into three parts:

- Training Data: 28821 images
- Validation Data: 7066 images
- Test Data: 5000 images

3. Code implementation:

+) File emotion_detector_trainer.py:

This is the program we used to create the model

- Import all the necessary module and library:

```
import numpy as np
np.object = np.object_          # Resolve deprecation warnings for numpy data types
np.bool = np.bool_
np.int = np.int32
import matplotlib.pyplot as plt
import os
import pandas as pd
from tqdm import tqdm
from tensorflow.keras.utils import to_categorical
from keras_preprocessing.image import load_img
from keras.models import Sequential
from keras.layers import Dense, Conv2D, Dropout, Flatten, MaxPooling2D
from keras.callbacks import EarlyStopping, ModelCheckpoint
from sklearn.preprocessing import LabelEncoder
```

- Direct suitable folder paths:

```
# Directory paths for training and testing images
TRAIN_DIR = 'images/train'
TEST_DIR = 'images/test'
```

- Create function name create_data_frame to create data frame for training and testing:

```python
# Function to create a DataFrame with image paths and labels
def create_data_frame(dir):
    image_paths = []
    labels = []
    for label in os.listdir(dir):
        for image_name in os.listdir(os.path.join(dir,label)):
            image_paths.append(os.path.join(dir,label,image_name))
            labels.append(label)
        print(label, "completed")
    return image_paths, labels

# Creating DataFrames for training and testing data
train =pd.DataFrame()
train['image'], train['label'] = create_data_frame(TRAIN_DIR)

test = pd.DataFrame()
test['image'], test['label'] = create_data_frame(TEST_DIR)
```

- Extracting features:

```
# Function to extract features from images
def extract_features(images):
    features = []
    for image in tqdm(images):
        img = load_img(image,grayscale = True)
        img = np.array(img)
        features.append(img)
    features = np.array(features)
    features = features.reshape(len(features), 48, 48, 1)
    return features

# Extracting features for training and testing datasets
train_features = extract_features(train['image'])
test_features = extract_features(test['image'])
```

- Feature rescale:
  We scale all features to the same scale, in order to have easier computation

```
# Normalizing pixel values to range [0, 1]
x_train = train_features/255.0
x_test = test_features/255.0
```

- Create an instance of the "LabelEncoder" to encode the labels and fit the "LabelEncoder" to the training data's labels. Then, transform the training set labels into encoded numerical values. This means that each original label in **train['label']** is replaced with its corresponding encoded numerical value:

```
# Encoding labels to integers
le = LabelEncoder()
le.fit(train['label'])

y_train = le.transform(train['label'])
y_test = le.transform(test['label'])
```

- Convert integer labels to one-hot encoded format and create callbacks to stop early and save the best model:

```
# Converting integer labels to one-hot encoded format
y_train = to_categorical(y_train, num_classes = 7)
y_test = to_categorical(y_test, num_classes = 7)

# Callbacks for early stopping and saving the best model
call_back = EarlyStopping(monitor='val_loss', patience=5, verbose=1, mode='auto')
best_model_file = "best_model_1.keras"
best_model = ModelCheckpoint(best_model_file, monitor='val_accuracy', verbose=1, save_best_only=True)
```

- Build model:

```python
# Building the Convolutional Neural Network (CNN) model
model = Sequential()
# Convolutional layers with pooling and dropout for regularization
model.add(Conv2D(128, kernel_size = (3,3), activation = 'relu', input_shape = (48,48,1)))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Dropout(0.4))

model.add(Conv2D(256, kernel_size = (3,3), activation = 'relu'))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Dropout(0.4))

model.add(Conv2D(512, kernel_size = (3,3), activation = 'relu'))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Dropout(0.4))

model.add(Conv2D(512, kernel_size = (3,3), activation = 'relu'))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Dropout(0.4))

model.add(Flatten())
# Fully connected layers with dropout
model.add(Dense(512, activation = 'relu'))
model.add(Dropout(0.4))
model.add(Dense(256, activation = 'relu'))
model.add(Dropout(0.3))
# Output layer with softmax activation for multi-class classification
model.add(Dense(7, activation = 'softmax'))
```

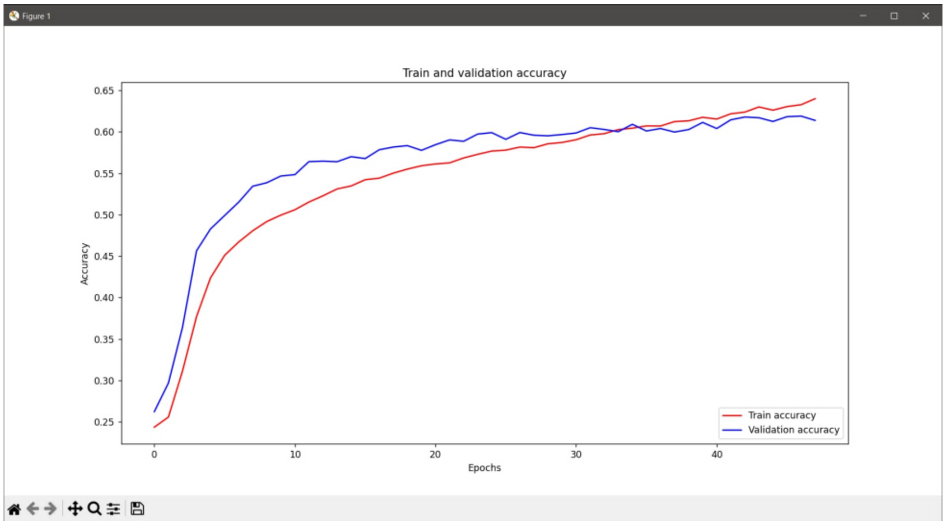
- We compile the model with optimizer attribute and train the model:

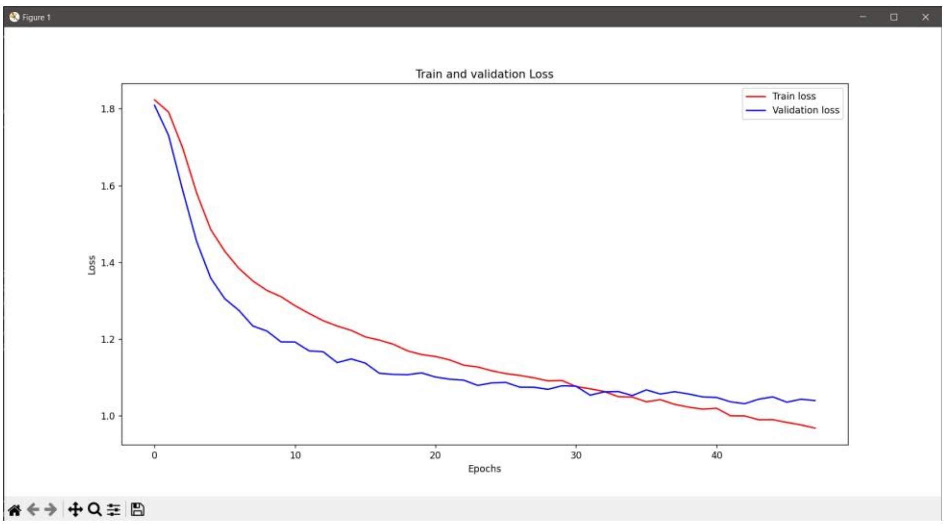
```python
# Compiling the model
model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'] )

# Training the model
history = model.fit(x = x_train, y = y_train, batch_size = 128, epochs = 100, validation_data = (x_test, y_test), callbacks=[call_back, best_model])
```

- Plot the training and validation accuracy and loss:

```
# Plotting training and validation accuracy
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(acc))

fig = plt.figure(figsize=(14,7))
plt.plot(epochs, acc , 'r', label="Train accuracy")
plt.plot(epochs, val_acc , 'b', label="Validation accuracy")
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Train and validation accuracy')
plt.legend(loc='lower right')
plt.show()

# Plotting training and validation loss
fig2 = plt.figure(figsize=(14,7))
plt.plot(epochs, loss , 'r', label="Train loss")
plt.plot(epochs, val_loss , 'b', label="Validation loss")
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Train and validation Loss')
plt.legend(loc='upper right')
plt.show()
```

➔ Result:





Our model achieves the following result when apply CNN model:

- Test Accuracy: 65%
- Test Loss: 1.8

+) Program *predict_unseen_image.py:*

This is the program we used to test the model with images

- Import all the necessary module and library:

```
import os
import cv2
import numpy as np
np.object = np.object_              # Resolve deprecation warnings for numpy data types
np.bool = np.bool_
np.int = np.int32
from tensorflow.keras.models import load_model
```

- Load the trained model and define the emotion labels and base folder to contain all emotion folders:

```
# Load the trained model
loaded_model = load_model("best_model.keras")
print("Model loaded successfully")

# Define the emotion labels
emotion_labels = ['angry', 'disgust', 'fear', 'happy', 'neutral', 'sad', 'surprise']

# Define the base folder containing all emotion folders
base_folder = 'images2/validation'
```

- Define function predict_emotion( ):

```
def predict_emotion(base_folder, label):
    # Initialize counters
    angry_count = 0
    disgust_count = 0
    fear_count = 0
    happy_count = 0
    neutral_count = 0
    sad_count = 0
    surprise_count = 0
    total_count = 0
```

```python
    # Define the folder for the given label
    label_folder = os.path.join(base_folder, label)

    # Loop through all images in the specified label folder
    for filename in os.listdir(label_folder):
        if filename.endswith(".jpg") or filename.endswith(".png"):
            image_path = os.path.join(label_folder, filename)

            # Read the image
            image = cv2.imread(image_path)
            if image is None:
                print(f"Error: Unable to read image {filename}")
                continue
            # Convert to grayscale
            gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
            # Resize to match model input size
            resized = cv2.resize(gray, (48, 48))
            # Convert grayscale image to 3-channel image
            rgb_image = cv2.cvtColor(resized, cv2.COLOR_GRAY2RGB)
            # Normalize pixel values
            normalized = rgb_image / 255.0
            # Make prediction
            prediction = loaded_model.predict(np.expand_dims(normalized, axis=0))
            # Get the predicted emotion label
            predicted_label = emotion_labels[np.argmax(prediction)]
```

```python
            print(f"Image: {filename}, Predicted Emotion: {predicted_label}")
            total_count += 1
            if predicted_label == 'angry':
                angry_count += 1
            if predicted_label == 'disgust':
                disgust_count += 1
            if predicted_label == 'fear':
                fear_count += 1
            if predicted_label == 'happy':
                happy_count += 1
            if predicted_label == 'neutral':
                neutral_count += 1
            if predicted_label == 'sad':
                sad_count += 1
            if predicted_label == 'surprise':
                surprise_count += 1

    # Return the counts
    return total_count, angry_count, disgust_count, fear_count, happy_count, neutral_count, sad_count, surprise_count
```

- Predict images for each emotion category and print the results:

```
# Predict unseen images for each emotion category
angry_total, angry_angry, angry_disgust, angry_fear, angry_happy, angry_neutral, angry_sad, angry_surprise = predict_emotion(base_folder, 'angry')
disgust_total, disgust_angry, disgust_disgust, disgust_fear, disgust_happy, disgust_neutral, disgust_sad, disgust_surprise = predict_emotion(base_folder, 'disgust')
fear_total, fear_angry, fear_disgust, fear_fear, fear_happy, fear_neutral, fear_sad, fear_surprise = predict_emotion(base_folder, 'fear')
happy_total, happy_angry, happy_disgust, happy_fear, happy_happy, happy_neutral, happy_sad, happy_surprise = predict_emotion(base_folder, 'happy')
neutral_total, neutral_angry, neutral_disgust, neutral_fear, neutral_happy, neutral_neutral, neutral_sad, neutral_surprise = predict_emotion(base_folder, 'neutral')
sad_total, sad_angry, sad_disgust, sad_fear, sad_happy, sad_neutral, sad_sad, sad_surprise = predict_emotion(base_folder, 'sad')
surprise_total, surprise_angry, surprise_disgust, surprise_fear, surprise_happy, surprise_neutral, surprise_sad, surprise_surprise = predict_emotion(base_folder, 'surprise')

# Print the results
print(f"In the Angry folder:\nTotal Angry Images: {angry_total}\nAngry: {angry_angry}\nDisgust: {angry_disgust}\nFear: {angry_fear}\nHappy: {angry_happy}\nNeutral: {angry_neutral}\nSad: {angry_sad}\nSurprise: {angry_surprise}\n")
print(f"In the Disgust folder:\nTotal Disgust Images: {disgust_total}\nAngry: {disgust_angry}\nDisgust: {disgust_disgust}\nFear: {disgust_fear}\nHappy: {disgust_happy}\nNeutral: {disgust_neutral}\nSad: {disgust_sad}\nSurprise: {disgust_surprise}\n")
print(f"In the Fear folder:\nTotal Fear Images: {fear_total}\nAngry: {fear_angry}\nDisgust: {fear_disgust}\nFear: {fear_fear}\nHappy: {fear_happy}\nNeutral: {fear_neutral}\nSad: {fear_sad}\nSurprise: {fear_surprise}\n")
print(f"In the Happy folder:\nTotal Happy Images: {happy_total}\nAngry: {happy_angry}\nDisgust: {happy_disgust}\nFear: {happy_fear}\nHappy: {happy_happy}\nNeutral: {happy_neutral}\nSad: {happy_sad}\nSurprise: {happy_surprise}\n")
print(f"In the Neutral folder:\nTotal Neutral Images: {neutral_total}\nAngry: {neutral_angry}\nDisgust: {neutral_disgust}\nFear: {neutral_fear}\nHappy: {neutral_happy}\nNeutral: {neutral_neutral}\nSad: {neutral_sad}\nSurprise: {neutral_surprise}\n")
print(f"In the Sad folder:\nTotal Sad Images: {sad_total}\nAngry: {sad_angry}\nDisgust: {sad_disgust}\nFear: {sad_fear}\nHappy: {sad_happy}\nNeutral: {sad_neutral}\nSad: {sad_sad}\nSurprise: {sad_surprise}\n")
print(f"In the Surprise folder:\nTotal Surprise Images: {surprise_total}\nAngry: {surprise_angry}\nDisgust: {surprise_disgust}\nFear: {surprise_fear}\nHappy: {surprise_happy}\nNeutral: {surprise_neutral}\nSad: {surprise_sad}\nSurprise: {surprise_surprise}\n")
```

+) Program *predict_with_camera.py:*

This is the program we used to use the model to predict emotions with data from the PC's camera in real time.

- Import all the necessary modules and libraries:

```
import numpy as np
np.object = np.object_            # Resolve deprecation warnings for numpy data types
np.bool = np.bool_
np.int = np.int32
import cv2
from tensorflow.keras.models import load_model
```

- Load the pre-trained model and Haar Cascade:

```
# Load the pre-trained model
model = load_model("best_model.keras")

# Load Haar Cascade for face detection
haar_file=cv2.data.haarcascades + 'haarcascade_frontalface_default.xml'
face_cascade=cv2.CascadeClassifier(haar_file)
```

- Define the function extract_features( ):

```
# Function to extract features from the image
def extract_features(image):
    image = cv2.resize(image, (48, 48))  # Ensure the image is resized to 48x48
    feature = cv2.cvtColor(image, cv2.COLOR_GRAY2RGB)  # Convert grayscale to RGB
    feature = np.array(feature)
    feature = feature.reshape(1, 48, 48, 3)
    return feature / 255.0
```

- Initialize webcam, define emotion and set loop for real-time emotion detect:

```
# Initialize webcam
webcam=cv2.VideoCapture(0)
# Define emotion labels corresponding to the model's output classes
labels = {0 : 'angry', 1 : 'disgust', 2 : 'fear', 3 : 'happy', 4 : 'neutral', 5 : 'sad', 6 : 'surprise'}
```

```
# Loop for real-time emotion detection
while True:
    # Capture frame from webcam
    i,im=webcam.read()
    # Convert frame to grayscale
    gray=cv2.cvtColor(im,cv2.COLOR_BGR2GRAY)
    # Detect faces in the frame
    faces=face_cascade.detectMultiScale(im,1.3,5)
    try:
        # Iterate over detected faces
        for (p,q,r,s) in faces:
            # Extract the face region
            image = gray[q:q+s,p:p+r]
            # Draw rectangle around face
            cv2.rectangle(im,(p,q),(p+r,q+s),(255,0,0),2)
            # Resize face to 48x48
            image = cv2.resize(image,(48,48))
            # Extract features from the face
            img = extract_features(image)
            # Predict emotion
            pred = model.predict(img)
            # Get the predicted emotion label
            prediction_label = labels[pred.argmax()]
            # print("Predicted Output:", prediction_label)
            cv2.putText(im, '% s' %(prediction_label), (p-10, q-10),cv2.FONT_HERSHEY_COMPLEX_SMALL,2, (0,0,255))
        # Display the frame with predictions
        cv2.imshow("Output",im)
        # Break the loop if 'Esc' key is pressed
        cv2.waitKey(27)
    except cv2.error:
        pass
```

V. Model Evaluation:

We evaluate the model with the results from the program based on accuracy, precision, recall, specifier and F1-score indexes.

		Actual						
		Angry	Disgust	Fear	Happy	Neutral	Sad	Surprise
Predicted	Angry	2199	124	471	223	228	369	80
	Disgust	39	174	21	20	6	11	4
	Fear	272	30	1244	142	97	223	211
	Happy	120	6	143	6002	260	145	179
	Neutral	583	17	571	465	3533	1107	140
	Sad	684	79	1080	238	750	2914	61
	Surprise	98	6	567	125	91	61	2496

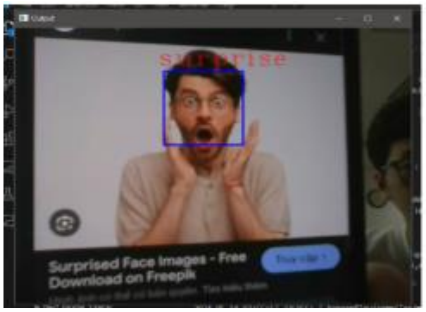
Total: 28708

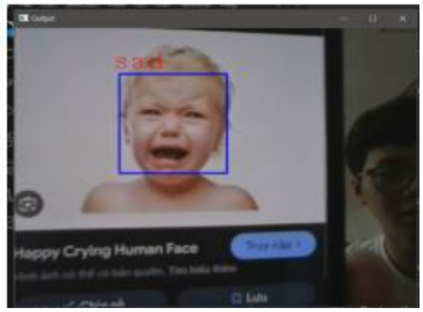
	Angry	Disgust	Fear	Happy	Neutral	Sad	Surprise
TP	2199	174	1244	6002	3533	2914	2496
FN	1796	262	2853	1213	1432	1916	675
FP	1495	101	975	853	2883	2892	948
TN	23218	28171	23636	20640	20860	20986	24589

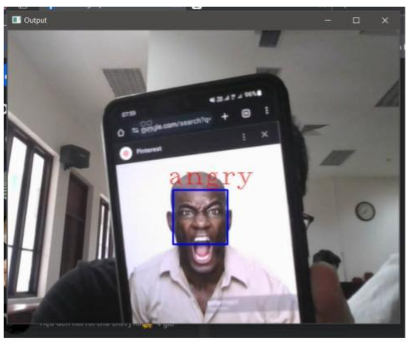
Measure	Angry	Disgust	Fear	Happy	Neutral	Sad	Surprise
Precision	0.59529	0.632727	0.560613	0.875565	0.550655	0.501895	0.724739
Recall	0.550438	0.399083	0.303637	0.831878	0.711581	0.603313	0.787133
Specificity	0.939506	0.996428	0.960384	0.960313	0.878575	0.878884	0.962877
Accuracy	0.885363	0.987355	0.866657	0.928034	0.849693	0.832521	0.943465
F-score	0.571986	0.489451	0.39392	0.853163	0.620859	0.54795	0.754649

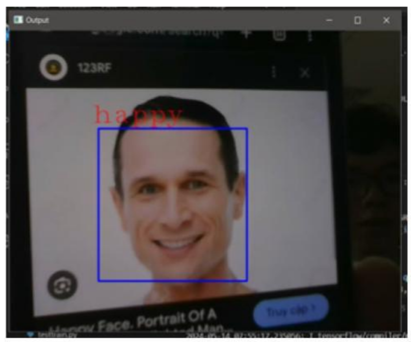
VI. Results:

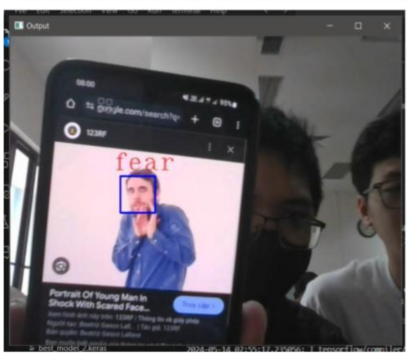
These are the result we from the program *predict_with_camera.py* in real time with images from both the Internet and real faces.

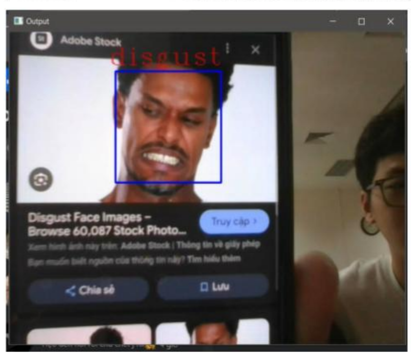


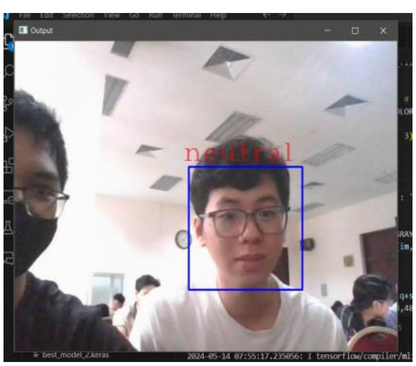












VII. References:

[List of references used in the project]

[1] The train and valid dataset:
https://www.kaggle.com/datasets/jonathanoheix/face-expression-recognition-dataset?fbclid=IwZXh0bgNhZW0CMTAAAR39z9oNqaAVe9koGb7sX3N-jtMpI0w0OVWLqt3XCmKFJPeJ2VSPh9eKZ4s_aem_AZso_0xWMiEoyY9KwR60k80EJlIXSnuLw_Sdv3aCZbKMAriQhUTJ4bvRXvA5vm1pp953haowpiaFHW02HFpuLurS.
[2] The test dataset:
https://www.kaggle.com/datasets/msambare/fer2013
[3] The reference code:
https://github.com/kumarvivek9088/Face_Emotion_Recognition_Machine_Learning

VIII. Conclusion:

In conclusion, the project successfully implemented a face emotion recognition system using machine learning techniques. The developed models showcased promising results in accurately identifying emotions from facial expressions. This project contributes to the growing field of emotion recognition technology and holds potential applications in various domains including healthcare, education, and entertainment.