The Hong Kong Polytechnic University

Department of Computing

COMP4913 Capstone Project

Report (Final)

REAL-TIME EMOTION MONITORING

Student Name: Lam Chi Wai

Student ID No.: 22067588D

Programme-Stream Code: 61431-SYF

Supervisor: Dr LOU Wei

Co-Examiner: Dr FAN Wen qi

2nd Assessor: Dr LUK Wing Pong Robert

Submission Date: 12 April 2024

Abstract

The project addresses a critical gap in education by developing a real-time student emotion monitoring and teaching material assessment system. Traditional teaching methods need immediate feedback on students' feelings and the effectiveness of teaching materials, hindering the learning process. The introduction of instant emotion monitoring provides valuable insights into students' emotional responses, enabling timely interventions and personalized support. This report highlights algorithmic differences and their performance impact among the transfer learning models and face detection algorithm. By combining the customized VGG16 model with the "mmod_human_face_detector" algorithm, it obtained a test accuracy of 78.29% on a cleaned and balanced FER2013 dataset.

TABLE OF CONTENTS

LIST OF FIGURE AND TABLE

CURRENT PROBLEM

LACK OF EMOTION RECOGNITION METHOD

Current emotion recognition methods mainly focus on static facial emotions, which means simultaneously analyzing and identifying facial expressions. However, this method ignores the temporal characteristics of facial expressions, that is, the changing process of expressions. Take the dynamic environment of the classroom as an example. Students' emotions in the classroom will change differently depending on the progress of the teacher's teaching. For example, they may feel bored because the teaching materials are simple, and they may feel fearful because the teaching materials are complicated. Therefore, relying only on static emotion recognition methods will lead to inaccurate results and cannot accurately analyze students' emotional changes.

RESTRICTED EMOTIONAL ANALYSIS

The imbalance problem in facial emotion recognition data sets is an important issue that leads to biased results. Most face emotion recognition systems use well-known datasets as their input, such as FER 2013, CK+, ExpW, etc. However, these datasets have the problem of uneven distribution of facial expressions. This imbalance means that the system is better at identifying emotions in more photos and has difficulty identifying those with fewer samples, limiting the system's ability to identify different emotions accurately.

METHODOLOGY

DATASET SELECTION & METHOD

FIRST ATTEMPT

Initially, the original FER 2013 dataset was employed to train the model, but it became apparent that the system could only effectively recognize certain specific emotions like happiness and surprise. Examining the distribution of emotion labels within the dataset revealed an inconsistency in the number of photos associated with each label. Some emotions had excessive images, while others had too few. This imbalance posed a challenge for the system in distinguishing all emotions accurately.

Figure 1 Training Set of FER 2013

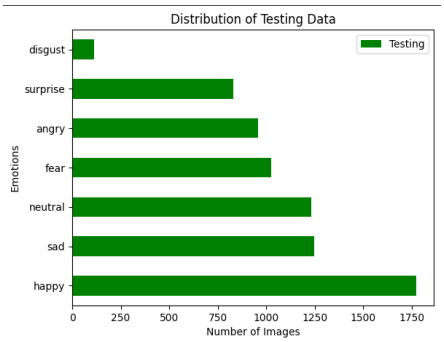


Figure 2 Test Set of FER 2013

SECOND ATTEMPT

To address the above issue, an attempt was made to employ SMOTE (Synthetic Minority Over-sampling Technique), a method capable of generating new images for underrepresented classes without altering the number of photos in the majority class. However, in the picture below, we can see that the newly generated images were found to be unreadable and could not pass through the system, rendering this approach unsuitable for resolving the problem.

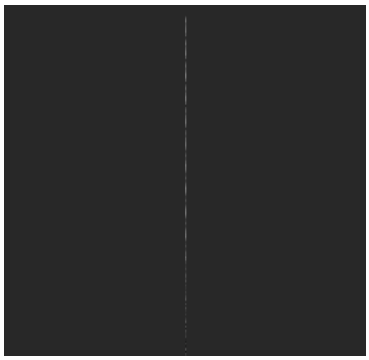


Figure 3 Problem Image by SMOTE

THIRD ATTEMPT

In addition to solving the problem of data imbalance, we also need to consider the unique needs of the classrooms where the system will be installed. This means the system must have unique capabilities to accurately detect appearances from different angles to capture the student's emotions at every moment. Therefore, I tried the ExpW (Expression in the Wild) dataset, which contains a comprehensive and diverse collection of facial images that demonstrate spontaneous expressions in real-world scenarios.

The ExpW dataset was not utilized in this project due to certain limitations. The dataset comprises images collected from diverse, uncontrolled environments, resulting in inconsistent lighting conditions and varying photo quality. These factors pose challenges that can make the training process more complex and less reliable. Moreover, the dataset predominantly consists of images featuring individuals from Western backgrounds, lacking sufficient samples representing different races. This limitation raises concerns about potential biases in the model and its ability to accurately recognize the facial expressions of students from diverse cultural backgrounds worldwide.

FOURTH ATTEMPT

In the above three attempts, I found that the main problem with the dataset is that there are many irrelevant photos in it, and the number of images owned by each label is unbalanced. Therefore, the behaviours of data cleaning and balancing dataset are necessary. While looking for a dataset, I

found a dataset on Kaggle that did both the data cleaning and the balancing steps (PRIZKER, 2022). Therefore, I will use this dataset as input for this system.

MACHINE LEARNING MODEL SELECTION

VGG16 MODEL

While doing research on how to perform the model training, there was a journal that published in 2023, which about using modified VGG16 model to perform the face emotion classification task (Yang, 2023). In the paper, it gave an insight for me to design a better version on the pre-trained VGG 16 model, by adding different layers and parameters settings.

EFFICIENTNETB3 MODEL

To have a better performance on the system, the step of comparison on different models is important. Through researching different models on internet, I have found a model called EfficientNet. EfficientNet is a model with an advanced deep neural network architecture, which introduces a compound scaling method so that the depth, width and resolution of the network can be scaled uniformly. This method can ensure effective allocation of model parameters, thereby improving accuracy with minimal computational cost. There was also a research paper that having an experiment on performance of different version of EfficientNet, it shown that version B3 having the best accuracy among 7 versions. I have tried this state-of-the-art deep in the system, but unfortunately it performed bad on the emotion recognition task. Therefore, I chose the customized VGG16 model to perform the task.

FACE DETECTION ALGORITHM

OPENCV CASCADE CLASSIFIER

It computes Haar-like features at different scales and locations in the image and applies a sliding window technique to evaluate the features. The cascade classifier consists of multiple stages with weak classifiers, gradually discarding regions unlikely to contain faces, enabling instant and efficient identification and localization of faces. Below is a flow of how to detect human face:

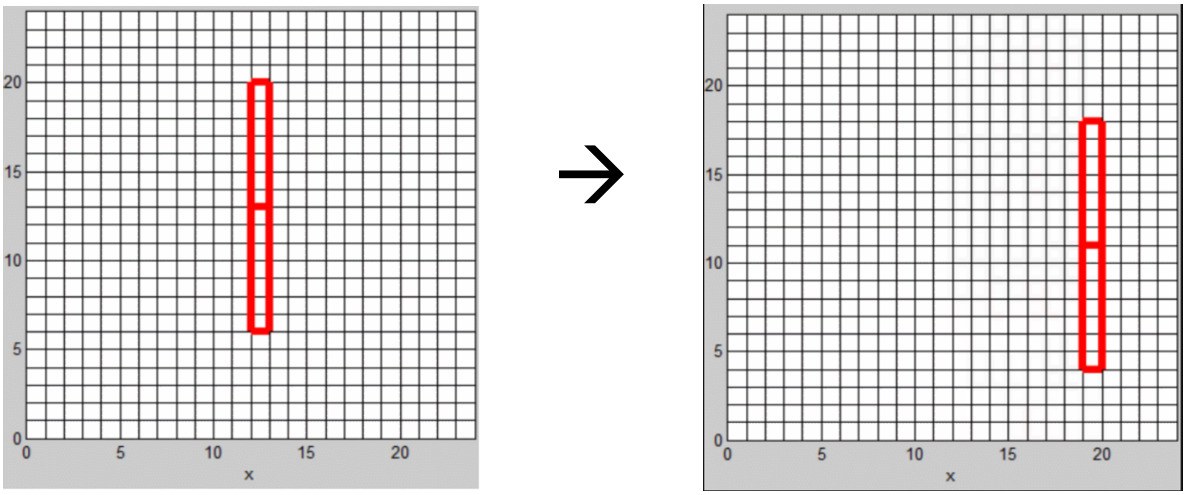


Figure 4 Haar Cascade Face Detection Illustration

DLIB LIBRARY

Dlib is a popular open-source library that efficiently implements facial landmark detection algorithms. It includes detecting the frontal faces and “mmod_human_face_detector”, which can detect frontal and small faces in real-time. Combining different functions and algorithms makes it possible to ensure that the system can quickly detect both large and small faces, regardless of distance. Through the experiemnt on comparing Haar Cascade Classifier and Dlib that conducted by Jia-Rou Lee, Kok-Why Ng and Yih-Jian Yoong in 2023, it shown that Dlib having a better performance on recognizing human faces, which means Dlib is a worth trying method to develop the system successfully.

PREPROCESSING STEPS TO DATA

DATA GENERATOR AND AUGMENTATION

Given the dataset's extensive size and intricate nature, it is unfeasible to simultaneously load all the images into memory. To address this challenge, I employed data generators that facilitate the on-the-fly generation of image batches during training, which ensures efficient utilization of the complete dataset while avoiding memory constraints.

Also, I used a range of data augmentation techniques to enhance the training procedure further. These techniques include rotation, shifting, flipping, and zooming. By applying such transformations, we can augment our training set, expanding its size and introducing greater diversity. The integration of data augmentation enhances the performance of our deep learning model by introducing variability into the training process. It mitigates the risk of overfitting, where the model becomes excessively specialized to the training data and needs to perform better on novel, unseen examples.

DATA EXPLORATION

EMOTION CATEGORIES

```
# Showing different categories in the dataset
categories_train = os.listdir(train_path)
categories_test = os.listdir(test_path)
print(categories_train)
print(categories_test)
```

```
['Surprise', 'Fear', 'Neutral', 'Sad', 'Disgust', 'Happy', 'Anger']
['Surprise', 'Fear', 'Neutral', 'Sad', 'Disgust', 'Happy', 'Anger']
```

Figure 5 Emotion Categories in Dataset

From the above image, the dataset contains 7 categories of emotion, which are "Angry", "Disgust", "Fear", "Happy", "Neutral", "Sad", and "Surprise".

NUMBER OF IMAGES IN CATEGORIES

Through the data cleaning and data balancing process are done, it has a smaller number than the original FER 2013 dataset, but it solved the problem of data unbalanced. Here is the total images number in training and test set respectively:

```
Total number of training images: 23251
Total number of test images: 5772
```

Figure 6 Total images in training and test set

Here is the distribution of the images in different emotion categories:

Figure 7 Image Distribution in Training Set

Figure 8 Image Distribution in Test Set

IMPLEMENTATION

DATA GENERATOR AND AUGMENTATION

Data generator:

```
# Source : https://www.geeksforgeeks.org/python-data-augmentation/
# Source : https://www.analyticsvidhya.com/blog/2020/08/image-augmentation-on-the-fly-using-keras-imagedatagenerator/
train_datagen = ImageDataGenerator(
    rescale = 1 / 255.,        # Rescale pixel values to be between 0 and 1
    rotation_range = 10,       # Rotates image with the number of degrees that we assigned
    zoom_range = 0.2,          # Zoom to our object randomly
    horizontal_flip = True,    # Randomly flip images horizontally
    height_shift_range = 0.1,  # Shift the height of images by up to 10% randomly
    width_shift_range = 0.1,   # Shift the width of images by up to 10% randomly
    fill_mode = 'nearest',     # Replaces the empty area with the nearest pixel values
    validation_split=0.2    # set the validation split
)

test_datagen = ImageDataGenerator(
    rescale = 1 / 255.
)
```

Figure 9 Code of Data Generator

Data Augmentation:

```
# Train generator
print('Generate Training Set - ',end = ' ')
train_generator = train_datagen.flow_from_directory(
    directory = train_path,
    class_mode = "categorical",
    target_size = (img_resize,img_resize),
    color_mode = 'rgb',
    batch_size = new_batch_size,
    shuffle = True,
    subset = 'training'

)

print('Generate Validation Set - ',end = ' ')
validation_generator = train_datagen.flow_from_directory(
    directory = train_path,
    target_size = (img_resize,img_resize),
    batch_size = new_batch_size,
    class_mode = 'categorical',
    color_mode = 'rgb',
    shuffle = False,
    subset = 'validation' # set as validation data
   )

# Test generator
print('Generate Shuffled Test Set - ', end = ' ')
test_generator = test_datagen.flow_from_directory(
    directory = test_path,
    class_mode = 'categorical',
    target_size = (img_resize,img_resize),
    color_mode = 'rgb',
    batch_size = new_batch_size,
    shuffle = True
)
```

```
Generate Training Set -  Found 18603 images belonging to 7 classes.
Generate Validation Set -  Found 4648 images belonging to 7 classes.
Generate Shuffled Test Set -  Found 5772 images belonging to 7 classes.
```

Figure 10 Code of Data Augmentation

To implement data augmentation, an instance of the “ImageDataGenerator” class is created and configured with various augmentation parameters. These parameters define the transformations to be applied to the training images. For example,” rotation_range” specifies the range of degrees for

random rotations, “zoom_range” controls the amount of random zooming, “horizontal_flip” enables random horizontal flipping, and “fill_mode” determines how to fill empty areas created by the transformations. The rescale parameter is used to normalize the pixel values of the images. Next, the “flow_from_directory” method is called on the “train_datagen” object to create a generator for the training data. This method takes the directory path of the training data, training_data_path, as input. Additional parameters such as “class_mode”, “target_size”, “color_mode”, “batch_size”, “shuffle”, and “subset” are specified to configure the generator's behaviour.

There is a generator called “validation_generator”, which is created for validation set that having 20% of training set’s images as it splits from the data generator. The reason why splitting 20% of training set into validation set is that I have tried different splitting number, such as 0.2, 0.15 and 0.1. Below is the comparison table:

Data Split	Train Accuracy	Validation Accuracy	Test Accuracy
0.1	76.03%	75.84%	75.54%
0.15	70.05%	73.43%	65.40%
0.2	80.59%	79.13%	78.29%

Table 1 Comparison on Data Split

We can see that split 20% of the training data into validation data having the best performance on different dataset. Therefore, I chose 0.2 as the splitting ratio.

Similarly, a separate generator, called "test_generator", is created for the testing data using the "test_datagen" object. The "flow_from_directory" method is called with the directory path of the testing data, "testing_data_path", and other relevant parameters. But it is worth mentioning that this test set is shuffled. Randomly reordering samples in the test set can mitigate potential bias or ordering effects, and the model's performance can be fairly assessed. This practice helps eliminate reliance on specific patterns or sequences in the test material, ensuring the assessment results are reliable and unbiased.

These data generators efficiently load and preprocess the training, validation and test data in batches during model training. The generators automatically apply the specified data augmentation techniques to the training images, enabling the model to learn from diverse and augmented data samples.

The output indicates that the code found 18603 images belonging to 7 classes in the training data directory, 4648 images in validation data directory and 5772 images in the same seven classes in the testing data directory. It also ensures that “flow_from_directory” method successfully locates the image files in the specified directories and identifies the number of images and classes in each directory.

MODEL CREATION

CUSTOMIZED VGG16 MODEL

VGG16 is a convolutional neural network (CNN) architecture that consists of 16 layers, which includes 13 convolutional layers and 3 fully connected layers. Here is the architecture of the original VGG16 model:

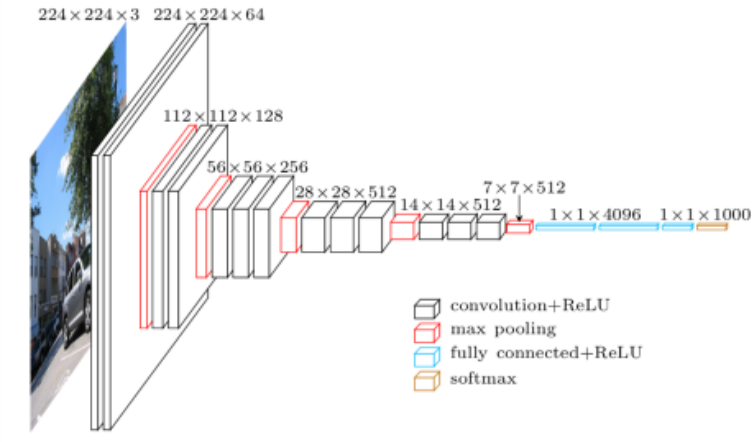


Figure 11 Architecture of VGG16

To have better accuracy, I have based on the source that I have found on the internet that introduces how to customise the layer of the VGG16 model, which was created by Malli in Github, to determine the parameters and add new layers to the model. The following images are the codes of how I have done it:

```
# Set the batch size for training
new_batch_size = 128

# Define the dropout rate for regularization
dropout_rate = 0.5

# Specify the number of frozen layers in the model
frozen_layer = 170

# Set the learning rate for the Adam optimizer
adam_lr = 0.001

# Set the learning rate for the SGD optimizer
sgd_lr = 0.01

# Specify the target size for resizing input images
img_resize = 64
```

```python
# Create the VGG16 model with specified input shape and average pooling
vgg_notop = VGG16(input_shape = (img_resize, img_resize, 3), include_top = False, pooling = 'avg')

# Get the output of the last layer of the VGG16 model
last_layer = vgg_notop.output

# Flatten the output
x = Flatten(name = 'flatten')(last_layer)

# Apply Dropout regularization
x = Dropout(dropout_rate)(x)

# Add a fully connected layer with 4096 units and ReLU activation
x = Dense(4096, activation = 'relu', name = 'fc6')(x)

# Apply Dropout regularization
x = Dropout(dropout_rate)(x)

# Add another fully connected layer with 1024 units and ReLU activation
x = Dense(1024, activation = 'relu', name = 'fc7')(x)

# Apply Dropout regularization
x = Dropout(dropout_rate)(x)

# Define the indices of layers to freeze during training
batch_norm_indices = [2, 6, 9, 13, 14, 18, 21, 24, 28, 31, 34, 38, 41, 45, 46, 53, 56, 60, 63, 66, 70, 73, 76, 80, 83, 87, 88, 92,

# Freeze the specified layers by setting their trainable attribute to False
for i in range(frozen_layer):
    if i < len(vgg_notop.layers) and i not in batch_norm_indices:
        vgg_notop.layers[i].trainable = False

# Add a dense layer with 7 units and softmax activation as the classifier layer
x = Dense(7, activation = 'softmax', name = 'classifier')(x)

# Create the vgg16 model with the modified layers
vgg16_model = Model(vgg_notop.input, x)


optim = tf.keras.optimizers.Adam(learning_rate=adam_lr, beta_1=0.9, beta_2=0.999, epsilon=1e-08)

sgd = tf.keras.optimizers.SGD(learning_rate=sgd_lr, momentum=0.9, nesterov=True)

# Compile the model with the specified optimizer, loss function, and metrics
vgg16_model.compile(optimizer = sgd, loss = 'categorical_crossentropy', metrics = ['accuracy'])
```

Figure 12 Customized VGG16 Parameters and Layers

MODEL ARCHITECTURE

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 64, 64, 3)	0
block1_conv1 (Conv2D)	(None, 64, 64, 64)	1,792
block1_conv2 (Conv2D)	(None, 64, 64, 64)	36,928
block1_pool (MaxPooling2D)	(None, 32, 32, 64)	0
block2_conv1 (Conv2D)	(None, 32, 32, 128)	73,856
block2_conv2 (Conv2D)	(None, 32, 32, 128)	147,584
block2_pool (MaxPooling2D)	(None, 16, 16, 128)	0
block3_conv1 (Conv2D)	(None, 16, 16, 256)	295,168
block3_conv2 (Conv2D)	(None, 16, 16, 256)	590,080
block3_conv3 (Conv2D)	(None, 16, 16, 256)	590,080
block3_pool (MaxPooling2D)	(None, 8, 8, 256)	0
block4_conv1 (Conv2D)	(None, 8, 8, 512)	1,180,160
block4_conv2 (Conv2D)	(None, 8, 8, 512)	2,359,808
block4_conv3 (Conv2D)	(None, 8, 8, 512)	2,359,808
block4_pool (MaxPooling2D)	(None, 4, 4, 512)	0
block5_conv1 (Conv2D)	(None, 4, 4, 512)	2,359,808
block5_conv2 (Conv2D)	(None, 4, 4, 512)	2,359,808
block5_conv3 (Conv2D)	(None, 4, 4, 512)	2,359,808
block5_pool (MaxPooling2D)	(None, 2, 2, 512)	0
global_average_pooling2d (GlobalAveragePooling2D)	(None, 512)	0
flatten (Flatten)	(None, 512)	0
dropout (Dropout)	(None, 512)	0
fc6 (Dense)	(None, 4096)	2,101,248
dropout_1 (Dropout)	(None, 4096)	0
fc7 (Dense)	(None, 1024)	4,195,328
dropout_2 (Dropout)	(None, 1024)	0
classifier (Dense)	(None, 7)	7,175

Total params: 21,018,439 (80.18 MB)
Trainable params: 9,290,567 (35.44 MB)
Non-trainable params: 11,727,872 (44.74 MB)

Figure 13 Customized Model Architecture

EXPLANATION

1. **Setting Parameters:** By adjusting the different values in the process of training the model, such as batch size, regularization strength, freezing layers, and controlling the learning rates for other optimizers (Adam and Stochastic Gradient Descent), it can achieve better performance and convergence.

2. **Creating VGG16 base Model**: Initializing with the specified input shape of the images, the "include_top" is set to False to exclude the fully connected layers at the top of the VGG16 model. The pooling parameter is set to 'avg', which applies global average pooling to the output of the last convolutional layer.

3. **Modifying the Model:** It begins by flattening the output of the model using the Flatten layer. Then, I applied dropout regularization with a specified dropout_rate. Also, I added two fully connected layers (fc6 and fc7) with 4096 and 1024 units, respectively, and ReLU activation functions.

4. **Freezing Layers:** I created a list of indices (batch_norm_indices) corresponding to the batch normalization layers in the model that won't be frozen during the model training. By specifying the number of layers to freeze, a portion of the model is frozen, allowing the remaining layers to be fine-tuned for specific tasks.

5. **Defining Optimizers:** Two optimizers are defined using the Adam and SGD. I initialized the Adam optimizer with a specified learning rate (adam_lr), beta values, and other parameters. I also initialized the SGD optimizer with a fixed learning rate (sgd_lr), momentum, decay, and nesterov parameters.

MODEL EVALUATION

The customized VGG16 model is chosen and here are the evaluations.

Accuracy comparison on training and validation set:

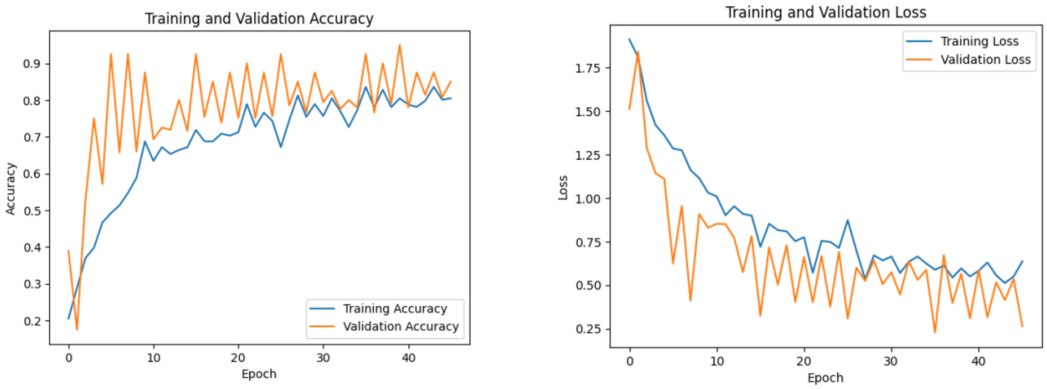


Figure 14 Accuracy and Loss of Training and Validation

From the accuracy graph, we can see that the accuracy of validation is slightly higher than the training accuracy, which may occur an overfitting problem as the model performs better on validation set instead of training set. But the model performs well on both set, which can ignore the problem and use this model to detect human emotion afterwards.

Last accuracy test on training, validation and test set, and here are the results.

Training Set

```
vgg16_Score1 = final_vgg16_model.evaluate(train_generator)

print("    Test Loss: {:.5f}".format(vgg16_Score1[0]))
print("Train Accuracy: {:.2f}%".format(vgg16_Score1[1] * 100))
```

```
  2/146 ━━━━━━━━━━━━━━━━━━━━ 8s 59ms/step - accuracy: 0.8105 - loss: 0.5769
W0000 00:00:1712297417.464108      90 graph_launch.cc:671] Fallback to op-by-op mode because memset node breaks graph update
146/146 ━━━━━━━━━━━━━━━━━━━━ 45s 301ms/step - accuracy: 0.8078 - loss: 0.5336
    Test Loss: 0.53179
Train Accuracy: 80.59%
W0000 00:00:1712297461.135618      89 graph_launch.cc:671] Fallback to op-by-op mode because memset node breaks graph update
```

+ Code + Markdown

Validation Set

```
vgg16_Score2 = final_vgg16_model.evaluate(validation_generator)

print("    Test Loss: {:.5f}".format(vgg16_Score2[0]))
print("Validation Accuracy: {:.2f}%".format(vgg16_Score2[1] * 100))
```

```
37/37 ━━━━━━━━━━━━━━━━━━━━ 11s 299ms/step - accuracy: 0.7833 - loss: 0.6040
    Test Loss: 0.57344
Validation Accuracy: 79.13%
W0000 00:00:1712297472.641651      87 graph_launch.cc:671] Fallback to op-by-op mode because memset node breaks graph update
```

Test Set

```
vgg16_Score3 = final_vgg16_model.evaluate(test_generator)

print("    Test Loss: {:.5f}".format(vgg16_Score3[0]))
print("Test Accuracy: {:.2f}%".format(vgg16_Score3[1] * 100))
```

```
46/46 ━━━━━━━━━━━━━━━━━━━━ 6s 130ms/step - accuracy: 0.7736 - loss: 0.5908
W0000 00:00:1712297478.865444      90 graph_launch.cc:671] Fallback to op-by-op mode because memset node breaks graph update
    Test Loss: 0.58704
Test Accuracy: 78.29%
```

Figure 15 Accuracy on Training, Validation and Test Set

The above data shows that the model performs well on the training and test set when the test set is shuffled. This means that the model can effectively learn and identify patterns that represent the problem domain without relying on the behaviour of specific patterns or sequences in the test data, thereby making accurate predictions for new and unseen instances.

Below are the confusion matrix of the model to show how it perform well on classifying different emotions in different dataset.

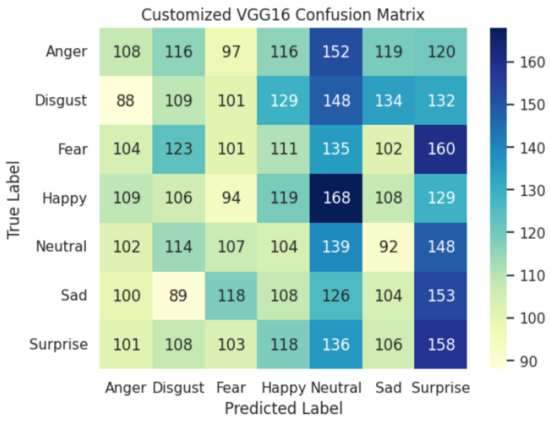


Figure 16 Current Confusion Matrix

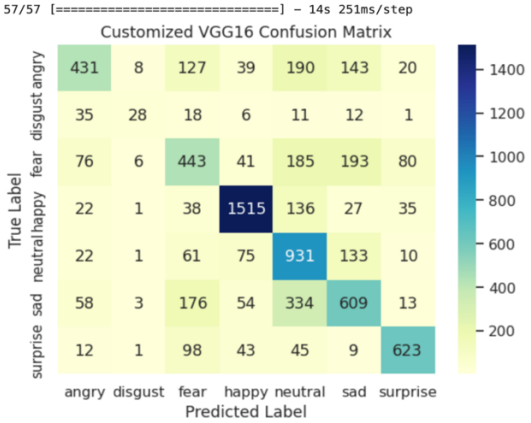


Figure 17 Past Confusion Matrix

We can see that on the right-hand side, the model has a high chance of predicting correct emotions in the unseen data because it has a balanced dataset. But on the left-hand side, the model can only perform well on the specific emotions, “Happy” and “Neutral”, because these two emotions have more training images than others and cause the imbalance problem. Therefore, balancing the dataset is the main reason the model can perform well on unseen data, and I chose the balanced dataset to ensure the model performance is the best.

First Attempt

I tried to use the algorithm of OpenCV Haar Cascade Classifier to do the face detection on the images. As the testing images are all frontal faces, which means having a large face for the algorithm to detect, it can work adequately to detect faces.

After testing on the images, I started to implement it in the video testing. Although it can detect the faces' emotions correctly in the video, there occurred an issue about it also detecting things that were not related to the human faces. Therefore, adjustment to the parameters of the Haar Cascade Classifier is needed.

Second Attempt

Based on the official documentation provided by OpenCV, I have learned how to set parameters for the Haar Cascade Classifier to perform well on face detection. By setting the numbers of the parameters of "scaleFactor", "minNeighbors", and "minSize", it performed better on focusing the human faces. Another problem occurred: while people were moving a bit in the video, it could not detect the face, and no emotion could be predicted. This is a big problem as the system is in a real-time classroom environment; we cannot assume the students are having a lesson without any movements. Therefore, adjustments are needed.

Third Attempt

Due to different problems in the above attempts, it is reflected that this algorithm can only detect frontal faces and cannot track the human face with various movements. Therefore, using Dlib is another solution. The reason for choosing Dlib is because its face detection model can provide higher accuracy and robustness when detecting faces, such as the HOG + SVM detector or MMOD framework, allowing Dlib to detect faces in different situations effectively. This results in more reliable and accurate face detection results than Haar Cascade.

Using the function "get_frontal_face_detector" in the Dlib library, it performed well on detecting faces while movements appeared and won't detect objects unrelated to human faces. However,

after I tried to implement it in the video that the human faces are far from the camera, it cannot detect any faces in the video, and no emotions can be predicted.

Final Attempt

By checking the Dlib library documentation, I found a function called "cnn_face_detection_model_v1", which can load the face detection model from a file. Also, it provided a pre-trained model on this function called "mmod_human_face_detector". To have a better performance on face detection, I chose to use the CNN face detection function combined with the pre-trained model to perform the task.

After implementing it with the testing video, it can detect faces with emotions at a certain distance, significantly improving the face detection task. Therefore, I chose this attempt as the final implementation of the face detection task.

EMOTION COUNTER AND ALERT

As this is a classroom analysis system, it is essential to have an emotion counter to tell the teachers how many students feel about that specific emotion. Due to the situation that counting of emotions is processed frame by frame, I have made a dictionary that stores 7 different emotions that will be detected in the system and set those counting numbers as 0. Below is the code of how I implement the function:

```
    # Reset emotion counts for each frame
    emotion_counts = {
        "Angry": 0,
        "Disgust": 0,
        "Fear": 0,
        "Happy": 0,
        "Neutral": 0,
        "Sad": 0,
        "Surprise": 0
    }

    for face in faces:
        x, y, w, h = face.rect.left(), face.rect.top(), face.rect.width(), face.rect.height()

        # Draw a rectangle around the face
        cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)

        # Extract the face region
        face_region = frame[y:y+h, x:x+w]

        # Predict the face
        predictions = predict_faces([face_region])[0]

        # Find the index of the emotion with the highest predicted score
        max_index = np.argmax(predictions)

        # Get the emotion label with the highest predicted score
        emotion_label = emotion_dict[max_index]

        # Increment the count for the detected emotion
        emotion_counts[emotion_label] += 1

        # Display the emotion label
        cv2.putText(frame, emotion_label, (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 255, 0), 2)

            # Check for high counts and display an alert if necessary
        if emotion_counts["Sad"] > alert_threshold or emotion_counts["Disgust"] > alert_threshold or emotion_counts["Fear"] > alert_threshold or
            alert_text = "Alert: Teaching adjustment required!"
            cv2.putText(frame, alert_text, (x, y - 40), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)

    # Display the emotion counts in the top right corner
    counts_text = ", ".join([f"{emotion}: {count}" for emotion, count in emotion_counts.items()])
    text_width, text_height = cv2.getTextSize(counts_text, cv2.FONT_HERSHEY_SIMPLEX, 0.7, 2)[0]
    text_x = frame_width - text_width - 10
    text_y = 30

    cv2.putText(frame, counts_text, (text_x, text_y), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)

    # Write the annotated frame to the output video
    output_video.write(frame)
```

Figure 18 Code of Emotion Counter

Flow of Emotion Counter:

- The code initializes a dictionary called emotion_counts to track each detected emotion's counts.
- When a face is detected and an emotion is recognized, the corresponding emotion label is obtained (emotion_label).
- The count for that specific emotion is incremented in the emotion_counts dictionary using emotion_counts[emotion_label] += 1.
- The counts are updated for each frame, ensuring that the number of occurrences for each emotion is accumulated in video.
- The updated counts are displayed on each frame using cv2.putText() in the top right corner of the frame.

Here is an example that how the emotion counter works:

Figure 19 Example of Emotion Detection

From the image, we can see that there is a face that with “neutral” emotion, which expecting the counter of “neutral” will change from 0 to 1 as there is only one face detected in the image.

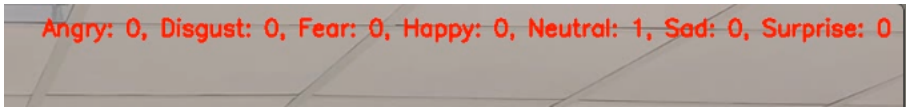


Figure 20 Example of Emotion Counter

From the right-hand corner, we can see that the number of “neutral” changed from 0 to 1, which has fulfilled the requirement of the system. Therefore, the system can count the emotions in real-time to give a teacher the class’s emotional information for a better teaching experience.

First image (Happy boy):

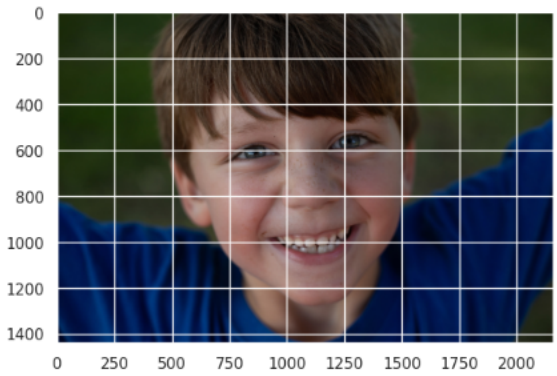


Figure 21 Happy Face

After applying "haarcascade_frontalface_default.xml", which is a haar cascade designed by OpenCV to detect the frontal face, the system can detect the face and use the cropping technique to crop out the unnecessary parts from the image.

Cropping process:

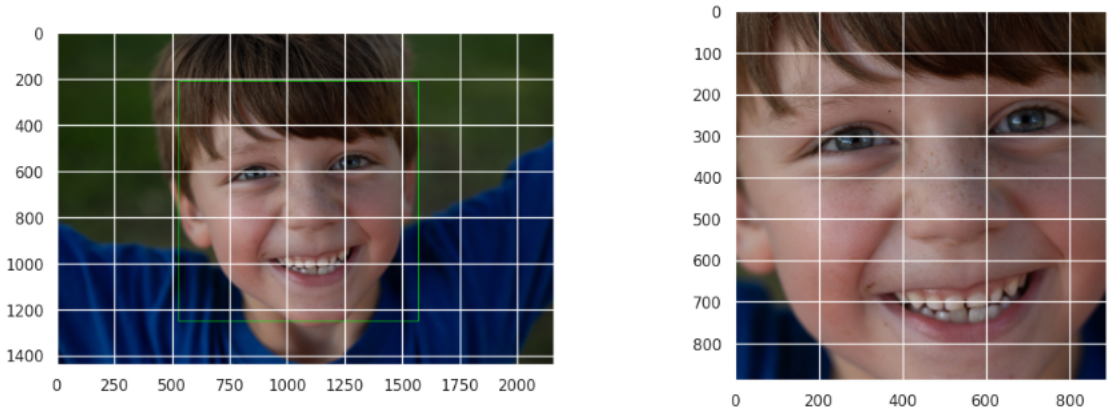


Figure 22 Result of Cropped Image

We can see that there is a green rectangle that is used to capture the human face. After it can detect the face from the image, applying the customized VGG16 model to recognize the emotion is the next step.

Here is the result:

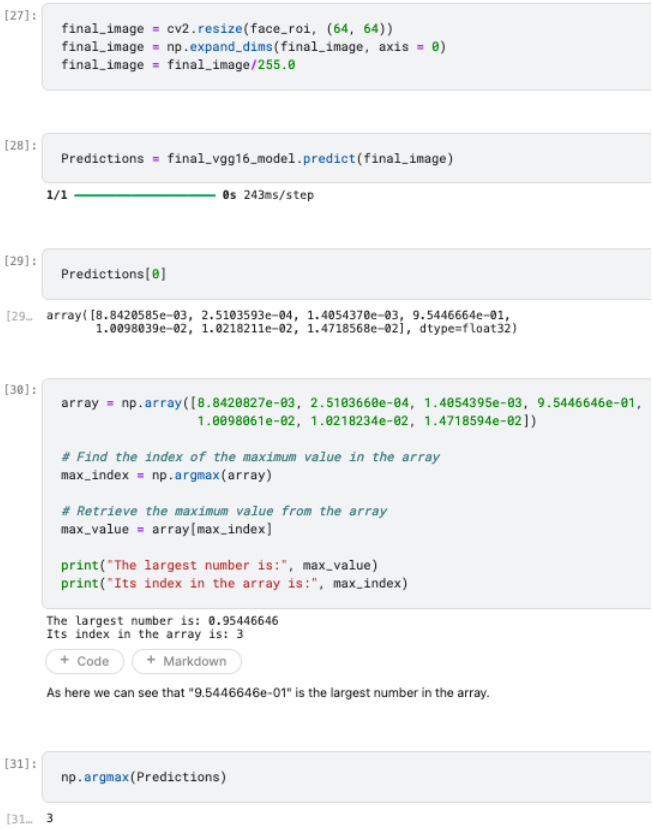


Figure 23Prediction on Happy Face

In the fer2013 dataset:

0 = angry, 1 = disgust, 2 = fear, 3 = happy, 4 = neutral, 5 = sad, 6 = surprise

Here, we can see that "9.5446646e-01" is the largest number in the array, and the output is showing "3", which means the model predicts that the image is equal to "happy". Therefore, it is a correct prediction as the image is a "happy boy".

Second Image (Surprise man):

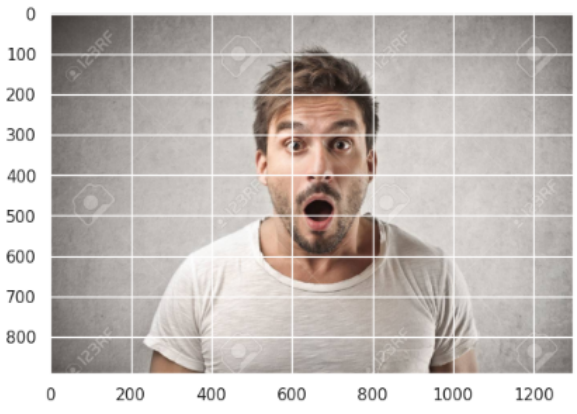


Figure 24 Surprise Face

Cropping process:

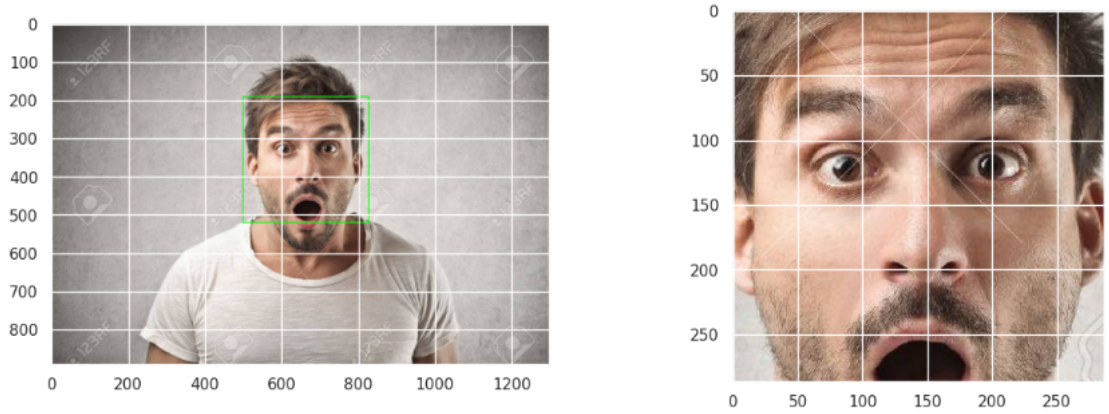


Figure 25 Result of Cropped Image

Result:

```
[37]:
final_image2 = cv2.resize(face_roi, (64, 64))
final_image2 = np.expand_dims(final_image2, axis = 0)
final_image2 = final_image2/255.0

Predictions2 = final_vgg16_model.predict(final_image2)

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step

+ Code  + Markdown

[38]:
Predictions2[0]

[38… array([2.8438913e-04, 2.5721736e-06, 2.8893180e-02, 5.6542864e-04,
       6.1503993e-03, 1.5998972e-03, 9.6250421e-01], dtype=float32)

[39]:
array = np.array([2.8438936e-04, 2.5721829e-06, 2.8893199e-02, 5.6542992e-04,
                  6.1503989e-03, 1.5998984e-03, 9.6250409e-01])

# Find the index of the maximum value in the array
max_index = np.argmax(array)

# Retrieve the maximum value from the array
max_value = array[max_index]

print("The largest number is:", max_value)
print("Its index in the array is:", max_index)

The largest number is: 0.96250409
Its index in the array is: 6

As here we can see that "9.6250409e-01" is the largest number in the array.

[40]:
np.argmax(Predictions2)

[40… 6
```

Here, we can see that " 9.6250409e-01" is the largest number in the array, and the output is showing “6", which means the model predicts that the image is equal to "surprise". Therefore, it is a correct prediction as the image is a "surprised man".

Third Image (Animal):

I have also tried the algorithm on animals photo to make sure that the system can only work on the human facial expression.

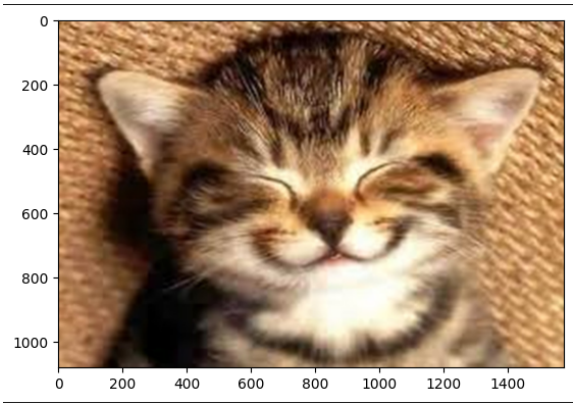


Figure 26 Cat Smiling Face

Result:

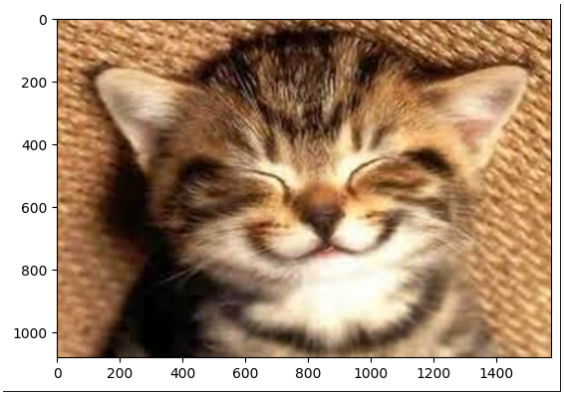


Figure 27 Result of Cropped Image

After applying the algorithm, we can see that the model and algorithm cannot detect the animal's emotions because humans and animals' facial structures and expressions differ. Therefore, the system can effectively recognize and analyze human facial emotions instead of animal emotions.

PROBLEM AND SOLUTION

From the above experiment, it reflected the Haar Classifier algorithm can perform well on detecting the frontal faces that the human faces is large. Then, I tried to implement it in the image that having smaller faces and testing video, there were different problems occurred.

Problem 1: Couldn't track the human face continuously.

Problem 2: Detected objects that are not related to human face.

Problem 3: Couldn't detect face on a certain distance.

Figure 28 Normal Harr Classifier on Video

I have captured the above from the video and it shows that even if the little girl in the left-hand side showing her frontal face, the algorithm still cannot track her face and emotion due to the reason of she is doing some movements.

Also, the algorithm detected not only the human faces, but also other objects. Below is a table of some objects that are not related to faces.

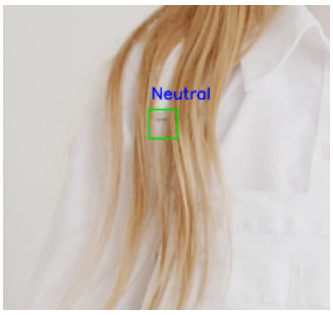
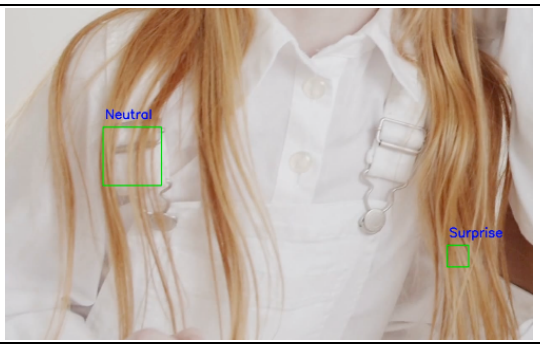
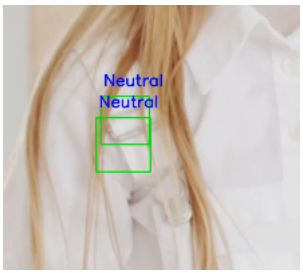
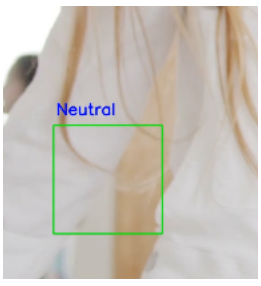
	
	

Table 2 Samples of Bad Detection

Also, I have tried to implement it to crop the face in the images that having smaller faces and it showed the message of cannot detect any face, which we can see there is no any green rectangle to crop the face in the image below:

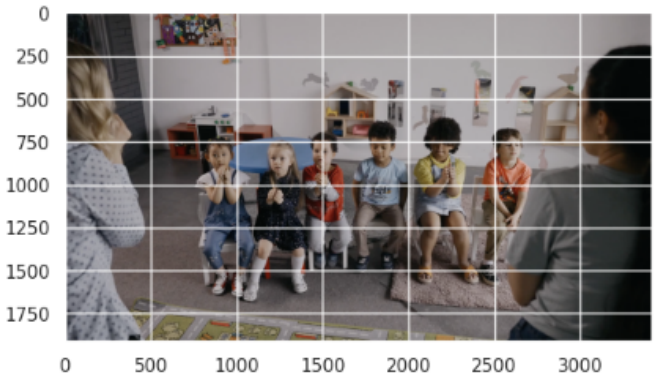


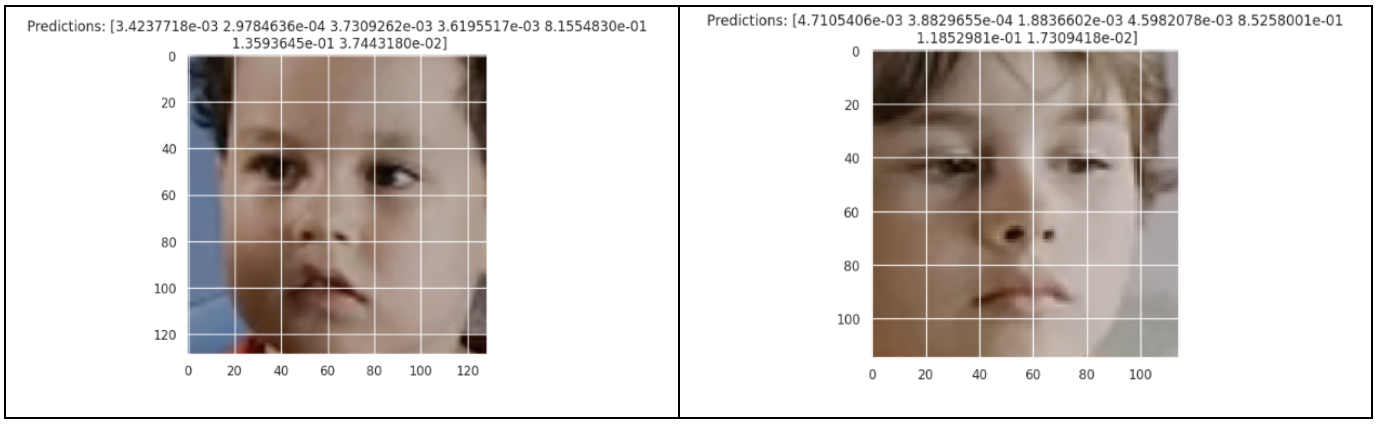
Figure 29 Test image without any Cropping

Solution 1: Setting parameter of Haar Cascade

Through using the function of "detectMultiScal" in the OpenCV library and set the follow parameters:

$$scaleFactor = 1.29, minNeighbors = 6, minSize = (50,50)$$

It solved the problem of detecting objects that are not related to human faces and it can detect the smaller faces in the image showing above. Here is the result:



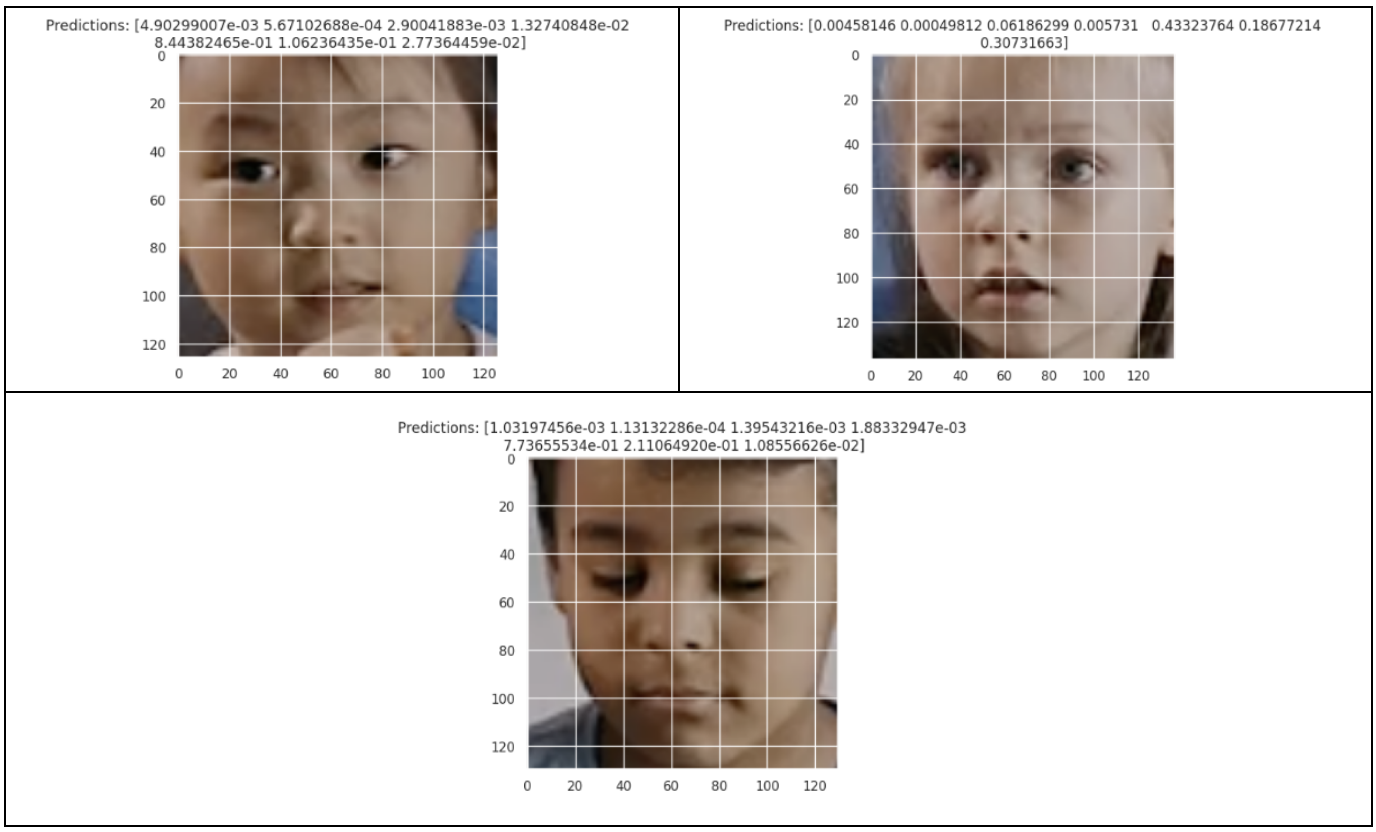


Table 3 Cropped Faces in Image

In figure 29, we can see that there are 6 human face should be detected by the algorithm. Although it can detect 5 of them after applying the parameters, but it still cannot detect the remaining face due to reason of the student's hair covered half of his face that showing below:

Figure 30 Face of Covered with Hair

Also, the function of track the human face continuously is still can't provide by this solution. To address this issue, I have tried another face detection algorithm, which is named as Dlib.

Solution2: Using Dlib library to track human face continuously.

By using the function in Dlib library called ” get_frontal_face_detector” and here is the result.

Figure 31 Using Dlib Frontal Face Detection

The result shows that it solved the problem in Solution 1, which is about tracking the human face continuously. While the children did lots of movement, the algorithm can still keep track of the face, which performs better than the Haar Cascade Classifier algorithm in detecting frontal faces.

Solution 3: Using Dlib pre-trained model to detect small faces

By combining the frontal face detection with the pre-trained model “mmod_human_face_detector” in the Dlib library, it can detect the remaining face, which solved the problem of solution 1.

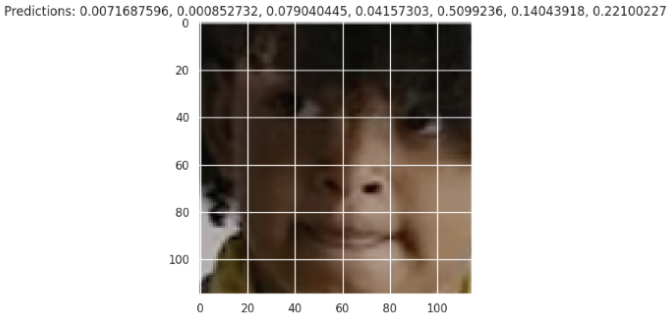


Figure 32 Detection for the Last Face

Therefore, different experiments comparing these two face detection algorithms reflected that using the Dlib pre-trained algorithm with the frontal face detection function performs the best among all the tests as it can solve all the problems mentioned above.

ANGLE TEST

To make sure that the face emotion detection model works well on predicting the emotions in real time, I have make an experiment on trying an angle test in frontal face to see what is the angle limitation and here are the results.

Right:

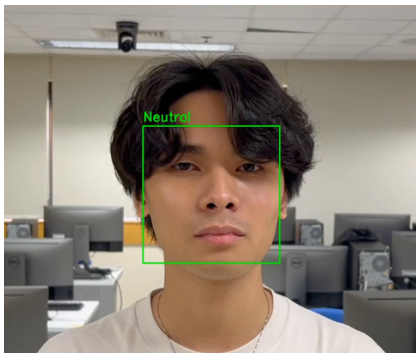
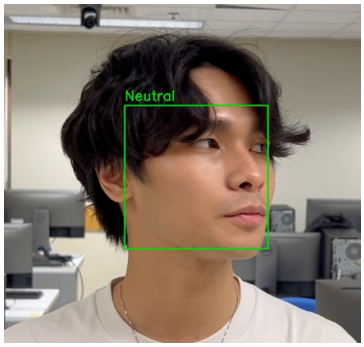
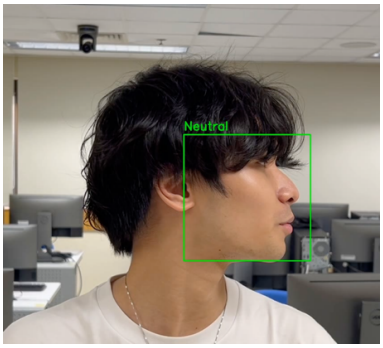
		
90 degree	45 degree	0 degree

Table 4 Angle Test from Right

Left:

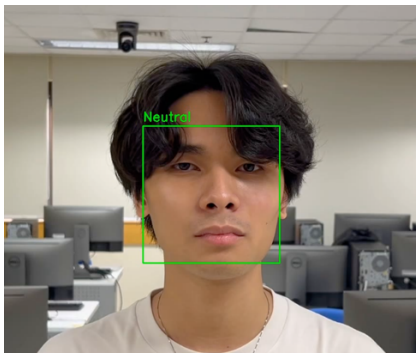
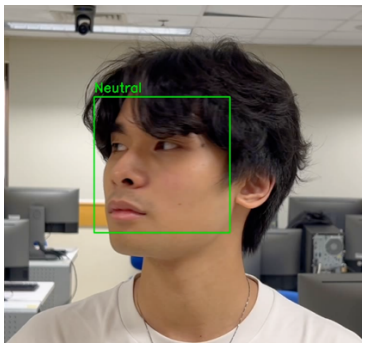
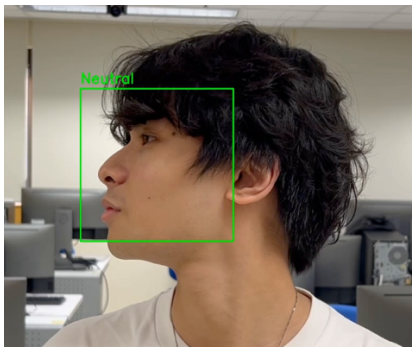
		
90 degree	45 degree	0 degree

Table 5 Angle Test from Left

Through the testing results, we can see that the system can detect the frontal faces in total 180 degrees from left to right.

DISTANCE TEST

To test the limitation on how far can the system can detect the facial expression, I have tried it and here are the results.

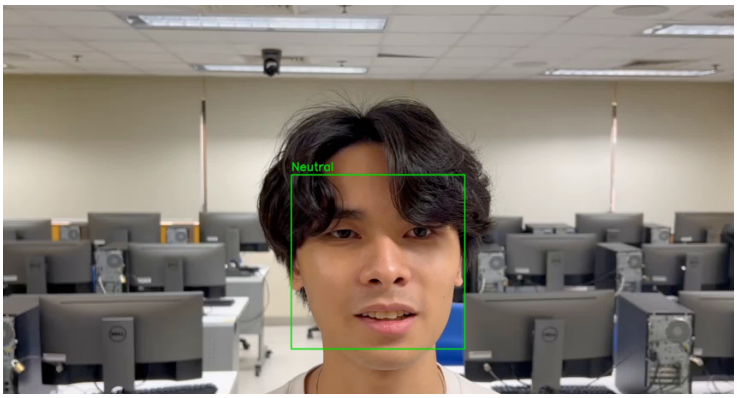
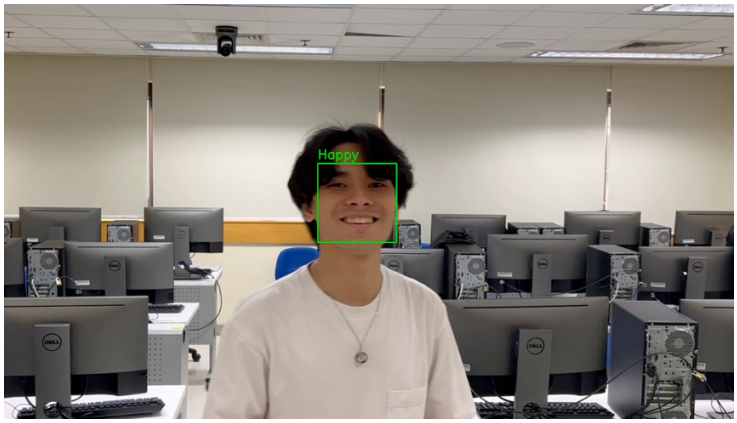
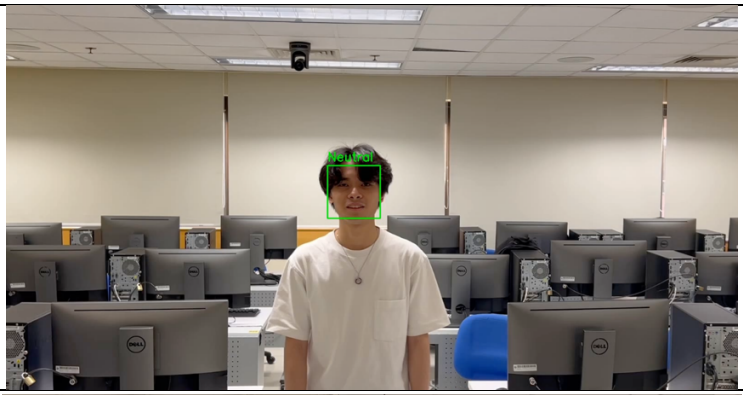
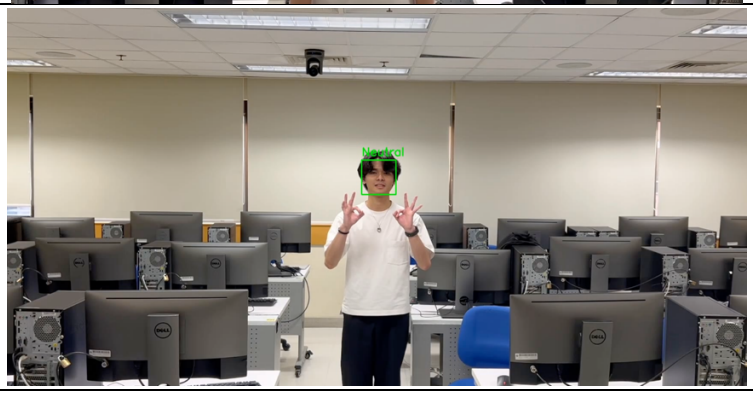





Table 6 Test on Different Distances

From the distance testing, it shown that my face can be detected while I am standing in row 2 of the classroom. This means that it can perform well on a small classroom that can detect all the faces of students.

CAMERA WITH HIGH-ANGLE SHOT

I have tried to test the system with the camera that is placed higher than the normal field of view, which shooting from top to bottom. Here is the results:

Table 7 Results in High-Angle Shot

We can see that both images are using high-angle shot method to do the facial expression recognition, but due to the reason that the distance of the camera is far, which cause the bottom image cannot detect any face with emotions.

CONCLUSION

Through different experiments comparing the machine learning models and face detection algorithms provided above, we can see that the system has improved from using the Dlib library's pre-trained model to perform the task of face detection instead of using OpenCV's Haar Cascade algorithm. Due to the performance of the Dlib library's pre-trained model, I have combined it with my customized VGG16 model to make sure that the function of facial expression recognition can be performed well in a small classroom.

In addition, with different tests on the system, such as angle tests, high-angle shot tests, and distance tests, I make sure that the system performs well on these tests by setting up different parameters to optimize the function of face detection and emotion recognition.

Therefore, this facial expression recognition system can be implemented in the environment of small size classrooms, which will help teachers adjust their teaching method and have a better teaching experience.

FUTURE WORK

1. System Alert

   The system now has the function of counting the emotions frame by frame, which allows the teacher to keep track of the class performance and adjust their teaching method. However, the system is missing the function of popping an alert message to warn the teacher if half of the class has negative emotions, like ”disgust”, “fear”, “angry”, and “sad”. Therefore, this function is needed.

2. Better algorithm on detecting faces in more far distance

   Due to the system's distance limitation, it cannot be implemented in a big classroom. Therefore, the system will perform better if a more robust face detection algorithm is used.

REFERENCE

Abbas, S. (2023, July 27). *Expression in-the-Wild (EXPW) dataset*. Kaggle. https://www.kaggle.com/datasets/shahzadabbas/expression-in-the-wild-expw-dataset

Classes. Dlib C++ Library. (n.d.). http://dlib.net/python/index.html

Classroom videos, download the best free 4K stock video footage & classroom HD video clips. (n.d.-a). https://www.pexels.com/search/videos/classroom/

CV::Cascadeclassifier class reference. OpenCV. (n.d.). https://docs.opencv.org/3.4/d1/de5/classcv_1_1CascadeClassifier.html

Dlib. (n.d.-b). http://dlib.net/files/mmod_human_face_detector.dat.bz2

Editor. (2021, May 18). *The gift of Happiness • Child Magazines*. CHILD Magazines. https://childmags.com.au/the-gift-of-happiness/

Lee, J.-R., Ng, K.-W., & Yoong, Y.-J. (2023). Face and facial expressions recognition system for blind people using Resnet50 architecture and CNN. *Journal of Informatics and Web Engineering*, *2*(2), 284–298. https://doi.org/10.33093/jiwe.2023.2.2.20

Piacquadio, M. (n.d.). *Surprised man*. 123RF. https://www.123rf.com/photo_22756908_surprised-man.html

P. Utami, R. Hartanto and I. Soesanti, "The EfficientNet Performance for Facial Expressions Recognition," *2022 5th International Seminar on Research of Information Technology and Intelligent Systems (ISRITI)*, Yogyakarta, Indonesia, 2022, pp. 756-762, doi: 10.1109/ISRITI56927.2022.10053007.

Prizker, I. (2022, May 30). *Fer2013plus-cleaned-AUGM-ballanced1*. Kaggle. https://www.kaggle.com/datasets/prilia/fer2013pluscleanedaugmballanced1/data

Rcmalli. (n.d.). *Rcmalli/keras-vggface: Vggface implementation with Keras Framework*. GitHub. https://github.com/rcmalli/keras-vggface

Sambare, M. (2020, July 19). *Fer-2013*. Kaggle. https://www.kaggle.com/datasets/msambare/fer2013

Tf.keras.preprocessing.image.imagedatagenerator : tensorflow V2.16.1. TensorFlow. (n.d.). https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator

Yang, F. (2023). Facial emotion classification based on the modified VGG and MobileNet. *Journal of Physics: Conference Series*, *2646*(1), 012038. https://doi.org/10.1088/1742-6596/2646/1/012038