2025-2026

Welcome to the 5th Lab session of CO3519 module.

For Two Hours Lab Session:

#	Task	Time (m)	Guidance
A	Please continue and finish the left over from the last week lab sheet.	50	There are some of the tasks remained from the last week. Also, the second step of Task B in Lab_Session_4_week_4. If you have completed that, please continue here with Task B below.
	Break	10	
B	Please explore the concepts we covered in the lecture related to the lab sheet.	10	You can think and contextualise the concepts towards your assessment.
	Please implement/run the code for facial expression recognition. The details about each snippets, dataset, and subsequent steps are given. Please carefully run and understand each snippet, as this will help you in your assignment.	40	Please follow the instruction on Page **2** to Page **5**. Page **6** for single image test. Please read the detailed instructions given there for each snippet. (Instructions followed by snippet).

For One Hour Lab Session:

#	Task	Time (m)	Guidance
1	Please perform the B task from the above two hours sessions.	50	Think and contextualise the concepts. Then follow the instructions on P2 to P5 for step1.

Facial emotion recognition

In today lab sheet, you will be directed to build a facial expression recognition system/method. Basic steps to be consider includes Face Detection using (haarcascade_frontalface_default.xml) which is a pre-trained cascade for face detection. After the face is detection, the features are extracted using Local Binary Pattern (LBP) features which has the ability to extract features effectively while maintaining computational simplicity. Lastly, we use SVM classifier. (Support Vector Machine).

1. Firstly, we will build the system with SVM classifier and LBP Features.

Import essential libraries where some of them you already knew. **CV2**: is OpenCV library, widely used for computer vision tasks such as reading, processing, and manipulating images. **Skimage.feature**: This library contains functions for feature extraction methods. We will use the Local Binary Pattern (LBP) method. **matplotlib.pyplot**: is used for plotting data and visualizing results. It will help us display images and graphs, such as the confusion matrix.

```
import cv2
import numpy as np
import os
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
from skimage.feature import local_binary_pattern
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

As first step is the face detection, we will load the Haar Cascade for face detection. (Based on Viola Jones algorithm- Recap today's lecture). Make sure you place "haarcascade_frontalface_default.xml" file in the same folder where you placed this main code file. (Otherwise take care of the path.)

```
# Load Haar Cascade for face detection
face_cascade = cv2.CascadeClassifier("/content/drive/My Drive/haarcascade_frontalface_default.xml")
```

This code fragment will read images from a directory structure organized by the class labels, detects faces in each image, crops each face, and stores the cropped face along with its label in two lists.

images = [], labels = []: These lists will store the detected face images and their corresponding labels.
for label in os.listdir(folder_path): This loop iterates through each folder within the folder_path. Each folder corresponds to a specific class (e.g., "happy," "sad," etc.).
label_path = os.path.join(folder_path, label): Combines the base path (folder_path) and the label folder name to get the path for each label folder.

```python
# Function to load images, detect faces, and prepare them for feature extraction
def load_and_detect_faces(folder_path):
    images = []
    labels = []
    for label in os.listdir(folder_path):
        label_path = os.path.join(folder_path, label)
        if os.path.isdir(label_path):
            for img_file in os.listdir(label_path):
                img_path = os.path.join(label_path, img_file)
                img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
                if img is not None:
                    # Detect face
                    faces = face_cascade.detectMultiScale(img, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))
                    for (x, y, w, h) in faces:
                        face_region = img[y:y+h, x:x+w]  # Crop to face region
                        images.append(face_region)
                        labels.append(label)
    return images, labels
```

Set paths for training and testing folders. Please adjust the path if you place the dataset into another folder.

train_folder_path and test_folder_path: Directory paths for the training and testing datasets, respectively.

CK_dataset/train: is the folder where training images are located.

CK_dataset/test: is the folder for testing images.

```python

# Set paths for training and testing folders
train_folder_path = "/content/drive/My Drive/CK_dataset/CK_dataset/train"
test_folder_path = "/content/drive/My Drive/CK_dataset/CK_dataset/test"

# Load and detect faces in training and testing data
X_train, y_train = load_and_detect_faces(train_folder_path)
X_test, y_test = load_and_detect_faces(test_folder_path)

```

This part will load and pre-process images from the specified training and testing paths, returning face images and their labels for further processing.

X_train contain a list of face images (in grayscale) from the training set.

y_train contain a list of corresponding labels (class names like "happy," "sad," etc.) for each training image.

X_test contain a list of face images from the testing set.

y_test contain the labels for each testing image.

The following snippet performs histogram equalization on a list of images to enhance contrast and potentially improve feature extraction quality. Using histogram equalization helps the subsequent feature extraction method (like LBP or ORB) by increasing image contrast, which can make facial features more distinguishable and improve classifier performance. This is useful in cases where lighting conditions in the images vary significantly. Similarly, StandardScaler and make_pipeline are part of scikit-learn and are useful for creating a pipeline where data preprocessing and model training can be done sequentially.

```python
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline

# Preprocess the images with histogram equalization
def preprocess_images(images):
    preprocessed_images = []
    for img in images:
        # Apply histogram equalization to improve contrast
        img = cv2.equalizeHist(img)
        preprocessed_images.append(img)
    return preprocessed_images
```

This following snippet will get feature vectors using Local Binary Patterns (LBP). Each face is resized to 64x64 pixels and processed to extract texture patterns, forming a histogram that represents the face's unique textures. This histogram is normalized to ensure consistency across varying lighting conditions. The resulting array of histograms provides a compact and effective representation of facial textures, ideal for training a classifier to recognize different expressions.

```python
# Define parameters for LBP
radius = 1
n_points = 8 * radius

# Function to extract LBP features from the cropped face images
def extract_lbp_features(images):
    lbp_features = []
    for img in images:
        # Resize face image to a standard size if needed
        img = cv2.resize(img, (64, 64))  # Standardize size
        lbp = local_binary_pattern(img, n_points, radius, method="uniform")
        # Histogram of LBP patterns
        (hist, _) = np.histogram(lbp.ravel(), bins=np.arange(0, n_points + 3), range=(0, n_points + 2))
        # Normalize the histogram
        hist = hist.astype("float")
        hist /= (hist.sum() + 1e-6)
        lbp_features.append(hist)
    return np.array(lbp_features)
```

Extract LBP features from training and testing face regions

```python
# Extract LBP features from training and testing face regions
X_train_features = extract_lbp_features(X_train)
X_test_features = extract_lbp_features(X_test)
```

Now, this snippet is to set up and train an SVM classifier for recognizing facial emotion from feature vectors extracted from images. By utilizing a pipeline that includes feature scaling, the code ensures that the model receives well-prepared input data, which enhances its performance and accuracy in predicting facial expressions based on the trained features.

SVC: Initializes the SVM classifier with a radial basis function (RBF) kernel, which is effective for non-linear decision boundaries. The **C** parameter controls the trade-off between achieving a low training

error and a low testing error, while gamma='scale' automatically determines the gamma value based on the number of features.

```

from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline

# Create an SVM pipeline with scaling and RBF kernel
svm_classifier = make_pipeline(StandardScaler(), SVC(kernel='rbf', C=7, gamma='scale'))

# Train and evaluate the model
svm_classifier.fit(X_train_features, y_train)
y_pred = svm_classifier.predict(X_test_features)

# Train the classifier on the training features
svm_classifier.fit(X_train_features, y_train)

```

Evaluation on train dataset. Please remember, sometimes you will get better performance on one class than the other ones. Try to think why it happen. See the images in the test folders, and analyse. Number of images, expressions, etc.

```
# Predict the test set results
y_pred = svm_classifier.predict(X_test_features)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")

# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print(len(y_test))
print(conf_matrix)
```

Optional: You can also, visualize the confusion matrix to see clearly see the correct and incorrect predictions with class names. (I kept this optional for you because you will be able to see the confusion matrix from previous snippet.)

```python
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.metrics import confusion_matrix, accuracy_score

# Assuming class names are defined
class_names = ["Angry", "Fear", "Happy", "Neutral", "Sadness", "Surprise"]

# Compute confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Calculate percentages
conf_matrix_percent = conf_matrix / conf_matrix.sum(axis=1, keepdims=True) * 100

# Plot confusion matrix with class names
plt.figure(figsize=(8, 8))
ax = sns.heatmap(conf_matrix, annot=False, cmap="BuGn", xticklabels=class_names, yticklabels=class_names, cbar=False)

# Add annotations with both counts and percentages
for i in range(conf_matrix.shape[0]):
    for j in range(conf_matrix.shape[1]):
        count = conf_matrix[i, j]
        ax.text(j + 0.5, i + 0.5, f"{count}", ha="center", va="center", color="black", fontsize=10)

plt.xlabel("Predicted Class")
plt.ylabel("True Class")
plt.title("Confusion Matrix")
plt.show()
```

Single Image Test:

After training, we can predict a single test image. (You can test images placed in 'testimages' folder) Firstly, the libraries are imported. At first, we will detect face and extract features from the detection face with some pre-processing as done previously in training.

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt
from skimage.feature import local_binary_pattern

# Define parameters for LBP
radius = 1
n_points = 8 * radius

# Load the Haar Cascade classifier for face detection
face_cascade = cv2.CascadeClassifier('/content/drive/My Drive/haarcascade_frontalface_default.xml')

# Function to preprocess and extract LBP features from a face image
def preprocess_and_extract_features(face_img):
    # Resize the face image to a standard size
    face_img = cv2.resize(face_img, (64, 64))

    # Apply LBP
    lbp = local_binary_pattern(face_img, n_points, radius, method="uniform")

    # Compute the histogram of LBP patterns
    (hist, _) = np.histogram(lbp.ravel(), bins=np.arange(0, n_points + 3), range=(0, n_points + 2))

    # Normalize the histogram
    hist = hist.astype("float")
    hist /= (hist.sum() + 1e-6)

    return hist.reshape(1, -1)  # Return as a 2D array for model input

```

The following is the function defined to predict and display the image. The comments added are now helping you what each snippet does.

```python
# Function to predict and display the expression of a single image
def predict_emotion(image_path, model):
    # Load the image in color
    img = cv2.imread(image_path)
    if img is None:
        raise ValueError("Image not found or could not be loaded.")

    # Convert the image to grayscale for face detection
    gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # Detect faces in the image
    faces = face_cascade.detectMultiScale(gray_img, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))

    # If no faces are detected, return
    if len(faces) == 0:
        print("No faces detected in the image.")
        return
```

Please continue the below fragment within the above function. Next, I have shown an example of how the predicted image will look like.

```python
    # Loop over the detected faces
    for (x, y, w, h) in faces:
        face_region = gray_img[y:y+h, x:x+w]  # Crop the detected face region

        # Extract features from the face region
        features = preprocess_and_extract_features(face_region)

        # Predict the expression (directly get the predicted label as a string)
        predicted_expression = model.predict(features)[0]

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

        # Display the predicted label above the rectangle
        cv2.putText(img, predicted_expression, (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.3, (255, 0, 0), 2)

    # Convert to RGB for display
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

    # Display the image with predictions
    plt.imshow(img_rgb)
    plt.title(f"Predicted Expression: {predicted_expression}")
    plt.axis('off')
    plt.show()

# Example path to an image
image_path = "/content/drive/My Drive/testimages/testimages/27.tiff"
predict_emotion(image_path, svm_classifier)
```