2025-2026

Welcome to the 6th Lab session of CO3519 module.

For Two Hours Lab Session:

#	Task	Time (m)	Guidance
A	Please give the concepts we covered in the lecture a short overview.	5	Just grab the concepts.
	Please implement/run the code for facial expression recognition. The details about each snippet, dataset, and subsequent steps are given. Please carefully run and understand each snippet, as this will help you in your assignment.	45	Please follow the instruction on Page **2** to Page **8**. Single image test is provided for both steps/methods separately. Please read the detailed instructions given there for each snippet. (Instructions followed by snippet). Skip Great Task.
	Break	10	
B	Please perform the **Deep Dive Analysis Task**.	40	The **Deep Dive Analysis Task** is located on Page 8. This is the same task done in Task A but with different dataset "JAFFE-[70,30]". This is Japanese FER dataset where 70% is training and 20% for testing. HoG with SVM is asked too.

For One Hour Lab Session:

#	Task	Time (m)	Guidance
A	Please perform the A task from the above two hours sessions.	50	See the related guide for it in the above two hours session Task A.

Note: I always recommend downloading the sheet and then you will be able to clearly see each variable/value in the code.

Facial emotion recognition

In previous lab sheet, we implemented to learned building a facial expression recognition system/method where Basic steps: Face Detection using (haarcascade_frontalface_default.xml), features extraction using Local Binary Pattern (LBP), Classification using SVM classifier. (Support Vector Machine) were done. Today, we will use ***LBP features*** with **Decision Tree DT classifier** and then Histogram of Oriented Gradient ***HoG features*** with **DT classifier**. But here, the code snippets are provided with comments for each line code. Some code might be repeated you already used in previous lab sheet, so much details for those are not added.

Today lab sheet is consisting of two methods, each of which is given below:

1. First, we will construct the system with *Decision Tree DT 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. Here, we will also import **Decision Tree Classifier** which will be used for classification.

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

```

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

```
# Function to load images, detect faces, and extract cropped face regions
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:
                    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]
                        images.append(face_region)
                        labels.append(label)
    return images, labels
```

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)
```

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.
Subsequently, extract the features from the extracted faces.

```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, detect faces, and extract features for training data
X_train_faces, y_train = load_and_detect_faces(train_folder_path)
X_train_features = extract_lbp_features(X_train_faces)

# Load, detect faces, and extract features for testing data
X_test_faces, y_test = load_and_detect_faces(test_folder_path)
X_test_features = extract_lbp_features(X_test_faces)
```

Initialize and Train Decision Tree Classifier with a maximum depth of 10 to avoid overfitting. The classifier is trained on the LBP feature vectors (X_train_features) and their corresponding labels (y_train).

```python
# Initialize the Decision Tree Classifier
tree_classifier = DecisionTreeClassifier(max_depth=10, random_state=42)

# Train the classifier on the training data
tree_classifier.fit(X_train_features, y_train)

```

```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=10, random_state=42)
```

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 on the test set
y_pred = tree_classifier.predict(X_test_features)

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

# Display Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
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.)

```
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 of *week6* folder). Make sure the path is correct.

At first, we will detect face and extract features from the detection face with some pre-processing as done previously in training. The below two code fragments/images are given separately but they are same code and continue.

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

    # Detect face
    faces = face_cascade.detectMultiScale(img, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))
    if len(faces) == 0:
        print("No face detected.")
        return

    for (x, y, w, h) in faces:
        face_region = img[y:y+h, x:x+w]
        face_region = cv2.resize(face_region, (64, 64))
        lbp = local_binary_pattern(face_region, 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))
        hist = hist.astype("float")
        hist /= (hist.sum() + 1e-6)

        # Predict expression
        features = hist.reshape(1, -1)
        predicted_expression = model.predict(features)[0]

        img_rgb = cv2.cvtColor(cv2.imread(image_path), cv2.COLOR_BGR2RGB)

            # Draw a rectangle around the detected face in the color image
        img_with_box = img_rgb.copy()
        cv2.rectangle(img_with_box, (x, y), (x + w, y + h), (255, 0, 0), 1)  # Red rectangle


        plt.figure(figsize=(18, 6))

        # Display the image with predicted label

        plt.subplot(1, 3, 1)
        plt.imshow(img_with_box)
        plt.title(f"Face detection in the original image")
        plt.axis('off')

        # 1st subplot: Original image with predicted expression label
        plt.subplot(1, 3, 2)
        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/24.tiff"
predict_emotion(image_path, tree_classifier)
```

2. **Now, we will construct the system with *Decision Tree DT Classifier* and *HoG Features*.**

Now we will be using same DT classifier as used in previous code but with HoG Features. HOG (Histogram of Oriented Gradients) features are a widely used method in facial emotion recognition (FER) that capture the distribution of gradient orientations in localized image areas. They effectively represent facial contours and textures essential for distinguishing emotions, making them robust to variations in lighting and facial orientation. This representation helps classifiers accurately identify emotions in computer vision applications.

In the following code, you will see we have used the same code as used in point 1 but only updated for HoG features. So, you can use the same code as above but just update the features extraction part and make necessary

changes. If not, you can just continue with the following code. Apart of this, all the details what each snippet does can be seen from point.

```python
# Import necessary libraries
import cv2
import numpy as np
import os
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
import matplotlib.pyplot as plt
from skimage.feature import hog
```

```python
# Function to extract HOG features
def extract_hog_features(images):
    hog_features = []
    for img in images:
        img = cv2.resize(img, (64, 64))  # Resize to standard size
        # Compute HOG features
        hog_feat = hog(img, orientations=8, pixels_per_cell=(8, 8),
                       cells_per_block=(2, 2), block_norm='L2-Hys')
        hog_features.append(hog_feat)
    return np.array(hog_features)
```

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

# Function to load images, detect faces, and extract cropped face regions
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:
                    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]
                        images.append(face_region)
                        labels.append(label)
    return images, labels
```

```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, detect faces, and extract features for training data
X_train_faces, y_train = load_and_detect_faces(train_folder_path)
X_train_features = extract_hog_features(X_train_faces)


# Load, detect faces, and extract features for testing data
X_test_faces, y_test = load_and_detect_faces(test_folder_path)
X_test_features = extract_hog_features(X_test_faces)

```

```python
# Initialize the Decision Tree Classifier
tree_classifier = DecisionTreeClassifier(max_depth=10, random_state=42)

# Train the classifier on the training data
tree_classifier.fit(X_train_features, y_train)

```

```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=10, random_state=42)
```

```python
# Predict on the test set
y_pred = tree_classifier.predict(X_test_features)

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

# Display Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)
```

Single Image Test: HoG features and DT Classifier

After training, we can predict a single test image. (You can test images placed in 'testimages' folder of *week6* folder). Make sure the path is correct.

At first, we will detect face in the image (I used real world image which in this case is mine. After training you can test any image with single face in them) and extract features from the detection face with some pre-processing as done previously in training. The below two code fragments/images are given separately but they are same code and continue.

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

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

# Function to extract HOG features
def extract_hog_features(img):
    # Resize image to a standard size
    img = cv2.resize(img, (64, 64))

    # Extract HOG features and HOG image for visualization
    hog_features, hog_image = hog(img, orientations=8, pixels_per_cell=(8, 8),
                                  cells_per_block=(2, 2), block_norm='L2-Hys',
                                  visualize=True)
    return hog_features, hog_image

# Function to preprocess, detect face, extract HOG features, and predict emotion
def predict_emotion_hog(image_path, model):
    # Load the image in grayscale
    img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    if img is None:
        raise ValueError("Image not found or could not be loaded.")

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

    if len(faces) == 0:
        raise ValueError("No face detected in the image.")

    # Assume first detected face (for single face case)
    (x, y, w, h) = faces[0]
    face_region = img[y:y+h, x:x+w]
```

```python

    # Extract HOG features from the detected face region
    features, hog_image = extract_hog_features(face_region)

    # Reshape features for model input
    features = features.reshape(1, -1)

    # Predict the expression using the model
    predicted_expression = model.predict(features)[0]

    # Load the original image in color for visualization
    img_rgb = cv2.cvtColor(cv2.imread(image_path), cv2.COLOR_BGR2RGB)

    # Draw a rectangle around the detected face in the color image
    img_with_box = img_rgb.copy()
    cv2.rectangle(img_with_box, (x, y), (x + w, y + h), (255, 0, 0), 3)  # Red rectangle
        # Display the predicted label above the rectangle
    cv2.putText(img_with_box, predicted_expression, (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 3, (255, 0, 0), 20)

    # Display three subplots: prediction, face with bounding box, and HOG features
    plt.figure(figsize=(18, 6))

    # 1st subplot: Original image with predicted expression label
    plt.subplot(1, 3, 1)
    plt.imshow(img_rgb)
    plt.title(f"Original Input Image")
    plt.axis('off')

    # 2nd subplot: Face detected with bounding box
    plt.subplot(1, 3, 2)
    plt.imshow(img_with_box)
    # plt.title(f"Predicted Expression: {predicted_expression} with detection", fontweight='bold')
    plt.title(f"Predicted Expression: $\mathbf{{{predicted_expression}}}$ with detected bounding box")
    plt.axis('off')

    # 3rd subplot: HOG features
    plt.subplot(1, 3, 3)
    plt.imshow(hog_image, cmap='gray')
    plt.title("HOG Features")
    plt.axis('off')
    plt.show()

# Example usage with an image path and trained model
image_path = "/content/drive/My Drive/testimages/testimages/test1.JPG"
predict_emotion_hog(image_path, tree_classifier)
```

----------++++++++++++++++++++++++++++++++++----------
----------++++++++++++++++++++++++++++++++++----------

Deep Dive Analysis: Please perform the steps (**1**: LBP Features with DT Classifier, **2**: HoG Features with DT Classifier) for another data named "JAFFE-[70,30]" located in Week6 folder. If you successfully executed this task, please use HoG features with SVM classifier for JAFFE-[70,30] dataset.