(SVM)TERM_PROJECT_CS171

DS SPECIALIZATION 29

Marc Jason M. Chan

Troy Andrei Paytan

```
#Import necessary libraries

import os
import cv2
import dlib
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
import PIL.Image
from google.colab import drive
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle
import os
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

Step 1 - Dataset Labeling

Load the dataset and label each instance with its corresponding emotion class. For example, if an image represents a happy expression, label it as 'happy.' Similarly, label other images with their respective emotion classes.

```
data_dir = "/content/drive/MyDrive/Module 3 dataset/train"  # Replace this with the path to your main folder
emotion_classes = ["happy", "disgust", "fear", "surprise", "angry", "neutral", "sad"]

# Initialize lists to hold images and labels
images = []
labels = []

for emotion_class in emotion_classes:
    class_dir = os.path.join(data_dir, emotion_class)
    for image_file in os.listdir(class_dir):
        image_path = os.path.join(class_dir, image_file)
        image = Image.open(image_path)
        images.append(image)
        labels.append(emotion_class)


print("Number of images:", len(images))
print("Number of labels:", len(labels))
```

```
Number of images: 420
Number of labels: 420
```

Step 2- Feature Extraction

We will use OpenCV's built-in face detector (Haar Cascade) and facial landmark detector to extract the necessary facial features.

```
# Load pre-trained face detector and facial landmark detector
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')
landmark_model = "/content/drive/MyDrive/shape_predictor_68_face_landmarks.dat"  # Replace this with the path to the facial landmark model

# Load the facial landmark detector model
predictor = dlib.shape_predictor(landmark_model)

# Function to detect facial landmarks
def detect_landmarks(image, circle_size=1, circle_color=(0, 0, 255)):
    image_copy = image.copy()  # Make a copy of the original image
    gray = cv2.cvtColor(image_copy, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))

    for (x, y, w, h) in faces:
        face_roi = image_copy[y:y + h, x:x + w]
        gray_roi = gray[y:y + h, x:x + w]

        # Detect facial landmarks
        landmarks = predictor(gray_roi, dlib.rectangle(0, 0, w, h))

        # Convert landmarks to numpy array
        landmarks_np = np.array([[p.x, p.y] for p in landmarks.parts()])

        # Draw facial landmarks on the image copy
        for (x, y) in landmarks_np:
            cv2.circle(image_copy, (x, y), circle_size, circle_color, -1)

    return image_copy
```

The purpose of the code above is to load a pre-trained face detection model and a facial landmark detection model. It defines a function called detect_landmarks that takes an input image and detects facial landmarks using the loaded models. The function identifies faces in the image using the face detection model, extracts facial regions of interest, applies the facial landmark detection model to those regions, and then draws circles at the detected landmarks on a copy of the input image. The function returns the modified image with visualized facial landmarks. This code is useful for extracting and visualizing facial landmarks on input images, which is a crucial step in various facial analysis tasks, such as emotion recognition.

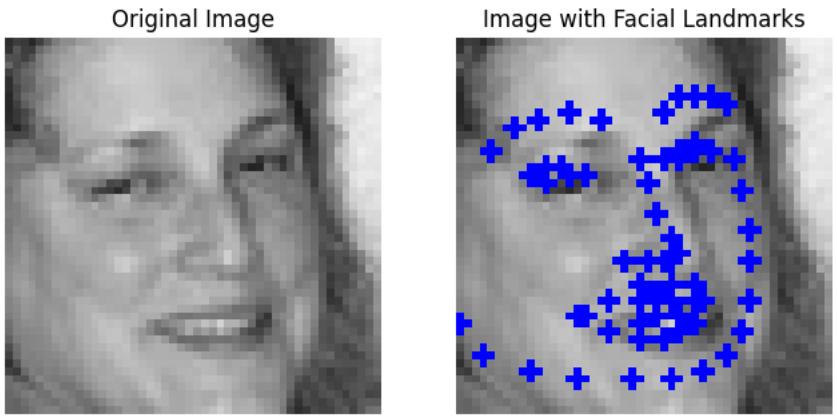
```
# Extract and visualize facial landmarks for the first image in the dataset
image_path = "/content/drive/MyDrive/Module 3 dataset/train/happy/165.jpg"
image = cv2.imread(image_path)

# Adjust the precision of plotting facial landmarks by modifying circle_size and circle_color
landmarked_image = detect_landmarks(image, circle_size=1, circle_color=(255, 0, 0))

# Display the original image using matplotlib
plt.figure(figsize=(8, 4))
plt.subplot(121)
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.title("Original Image")
plt.axis("off")

# Display the image with facial landmarks using matplotlib
plt.subplot(122)
plt.imshow(cv2.cvtColor(landmarked_image, cv2.COLOR_BGR2RGB))
plt.title("Image with Facial Landmarks")
plt.axis("off")

plt.show()
```



Facial landmarks was extracted and visualized, specifically for the emotion of "happy," from an image in a dataset. It uses OpenCV to read and process the image, detects facial landmarks using the detect_landmarks function, adjusts the precision of landmark visualization, and then displays both the original image and the image with highlighted facial landmarks side by side.

Step 3 - Dataset Splitting

Split the dataset into two parts: training and test sets. Use a stratified sampling technique to ensure that each emotion class is proportionally represented in both sets. 75-25 ratio was used for training and testing data

```
# Split the dataset into training (75%) and test (25%) sets with stratified sampling
train_images, test_images, train_labels, test_labels = train_test_split(images, labels, test_size=0.25, random_state=42, stratify=labels)

# Print the number of samples in the training and test sets
print("Number of samples in the training set:", len(train_images))
print("Number of samples in the test set:", len(test_images))
```

```
Number of samples in the training set: 315
Number of samples in the test set: 105
```

```
def convert_pil_image_to_numpy_array(image):
    image_np = np.array(image, dtype=np.uint8)
    image_np = cv2.cvtColor(image_np, cv2.COLOR_RGB2BGR)
    return image_np

images = np.array([convert_pil_image_to_numpy_array(image) for image in train_images])
```

Step 4: Model Development

Before training the models, preprocessing may be needed for the extracted facial landmarks or features and convert them into a suitable format for training the classifiers. Typically, you would flatten the features and create a feature matrix as input for the classifiers

```
train_features = np.array([np.array(image).flatten() for image in train_images])
test_features = np.array([np.array(image).flatten() for image in test_images])
```

Train Support Vector Machine (SVM) Classifier

SVM is a powerful classification algorithm that can be effective for image-based tasks. We'll use scikit-learn's SVC class for training.

```
from sklearn.svm import SVC

# Initialize the SVM classifier
svm_classifier = SVC(kernel='linear', C=1.0, random_state=42)

# Train the SVM classifier on the training data
# Train the SVM classifier on the training data
svm_classifier.fit(train_features, train_labels)
```

```
▾                SVC
SVC(kernel='linear', random_state=42)
```

Step 6: Model Testing

We will evaluate the performance of the trained classifier (SVM) on the test set and analyze the results.

```
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix,precision_score, recall_score, f1_score
import seaborn as sns
```

Test Support Vector Machine (SVM) Classifier:

```
# Test Support Vector Machine (SVM) Classifier
svm_predictions = svm_classifier.predict(test_features)
svm_accuracy = accuracy_score(test_labels, svm_predictions)
svm_precision = precision_score(test_labels, svm_predictions, average='weighted')
svm_recall = recall_score(test_labels, svm_predictions, average='weighted')
svm_f1 = f1_score(test_labels, svm_predictions, average='weighted')
svm_conf_matrix = confusion_matrix(test_labels, svm_predictions, labels=emotion_classes)

print("SVM Accuracy:", svm_accuracy)
print("SVM Precision:", svm_precision)
print("SVM Recall:", svm_recall)
print("SVM F1-score:", svm_f1)
print("SVM Confusion Matrix:")
print(svm_conf_matrix)
```

```
SVM Accuracy: 0.23809523809523808
SVM Precision: 0.23847933112638994
SVM Recall: 0.23809523809523808
SVM F1-score: 0.23488050885866957
SVM Confusion Matrix:
[[4 3 3 1 3 1 0]
 [1 2 4 0 4 3 1]
 [3 6 4 2 0 0 0]
 [3 1 2 4 1 1 3]
 [2 2 2 1 4 3 1]
 [1 2 0 0 4 6 2]
 [2 2 0 3 4 3 1]]
```

SVM Accuracy: 0.2381

> The accuracy of the Support Vector Machine (SVM) classifier is approximately 23.81%. This metric indicates the proportion of correctly classified instances out of the total number of instances in the test set.

SVM Precision: 0.2385

> Precision is the ratio of correctly predicted positive observations to the total predicted positives. In this case, the weighted average precision is about 0.2385. It suggests that when the SVM classifier predicts a certain emotion, it is correct about 23.85% of the time on average.

SVM Recall: 0.2381

> Recall, also known as sensitivity or true positive rate, is the ratio of correctly predicted positive observations to all observations in the actual class. The weighted average recall for this SVM model is approximately 0.2381. It means that the classifier correctly identifies about 23.81% of the instances belonging to each emotion class.

SVM F1-score: 0.2349

> The F1-score is the harmonic mean of precision and recall. It provides a balance between these two metrics. The weighted average F1-score for the SVM model is around 0.2349. This indicates that the model achieves a reasonable balance between precision and recall, although the overall value is not very high.

SVM Confusion Matrix:

> The confusion matrix provides a more detailed view of the classifier's performance for each emotion class. Rows represent the true class labels, while columns represent the predicted class labels. Here are some key observations from the confusion matrix:
>
> - The diagonal values (top-left to bottom-right) represent the number of instances correctly classified for each emotion.
> - Off-diagonal values represent misclassifications. For example, the element at row 1 and column 2 (1,2) indicates that one instance of class 1 was misclassified as class 2.
> - Some emotions, such as classes 2 and 3, have relatively higher misclassifications with other classes.
> - The class 5 (neutral) seems to have higher correct predictions compared to other classes.

Interpretation and Insights:

1. The SVM model's accuracy, precision, recall, and F1-score are all relatively low. This suggests that the model struggles to distinguish between different emotion classes effectively.
2. The confusion matrix highlights the challenges the model faces in correctly classifying certain emotions, as evidenced by non-diagonal values.
3. The higher misclassifications for some emotions could indicate either class imbalance or inherent similarity between those emotions.
4. The relatively higher performance for the neutral class (class 5) suggests that the model might have an easier time identifying neutral expressions compared to other emotions.

References:

1. https://www.analyticsvidhya.com/blog/2022/10/face-detection-using-haar-cascade-using-python/
2. https://github.com/italojs/facial-landmarks-recognition/blob/master/shape_predictor_68_face_landmarks.dat
3. https://www.kaggle.com/code/prashant111/svm-classifier-tutorial
4. https://www.kaggle.com/code/prashant111/random-forest-classifier-tutorial