(RANDOM FOREST)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"  # 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.

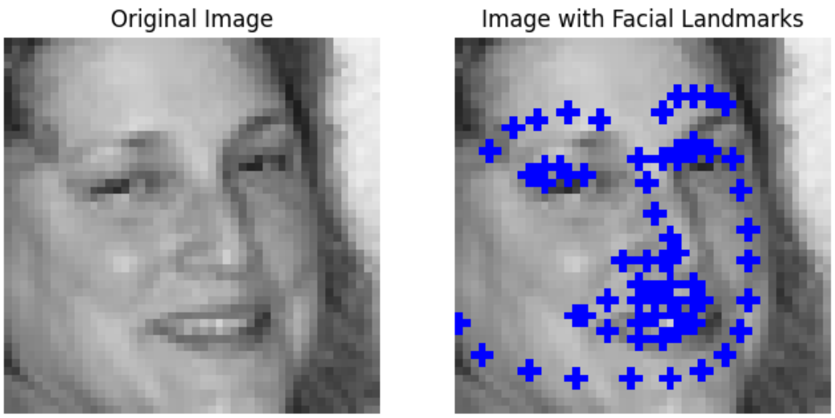
```
# Extract and visualize facial landmarks for the first image in the dataset
image_path = "/content/drive/MyDrive/Module 3 dataset/train/happy/165.jpg"
image = cv2.imread(image_path)

# 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 Feedforward Neural Network (FNN)

A Feedforward Neural Network, also known as a Multilayer Perceptron (MLP), is a type of artificial neural network that consists of an input layer, one or more hidden layers, and an output layer. It can be suitable for small datasets and simpler tasks like emotion recognition from handcrafted features.

we use LabelEncoder to convert the string labels into integer-encoded labels. Then, we use OneHotEncoder to one-hot encode the integer-encoded labels. This process ensures that your categorical labels are transformed into a suitable format for training the FNN.

```
from sklearn.preprocessing import LabelEncoder, OneHotEncoder

# Encode the labels into integers using LabelEncoder
label_encoder = LabelEncoder()
train_labels_encoded = label_encoder.fit_transform(train_labels)
test_labels_encoded = label_encoder.transform(test_labels)

# One-hot encode the integer labels using OneHotEncoder
onehot_encoder = OneHotEncoder(sparse=False)
train_labels_one_hot = onehot_encoder.fit_transform(train_labels_encoded.reshape(-1, 1))
test_labels_one_hot = onehot_encoder.transform(test_labels_encoded.reshape(-1, 1))

# Now `train_labels_one_hot` and `test_labels_one_hot` are one-hot encoded label matrices
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/preprocessing/_encoders.py:868: FutureWarning: `sparse` was renamed to `sparse_output` i
  warnings.warn(
```

```
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import StandardScaler

# Normalize the features (optional but recommended for FNN)
scaler = StandardScaler()
train_features_normalized = scaler.fit_transform(train_features)
test_features_normalized = scaler.transform(test_features)

# Initialize the FNN classifier
fnn_classifier = MLPClassifier(hidden_layer_sizes=(100, 50), activation='relu', solver='adam', random_state=42)

# Train the FNN classifier on the training data
fnn_classifier.fit(train_features_normalized, train_labels_one_hot)
```

```
MLPClassifier
MLPClassifier(hidden_layer_sizes=(100, 50), random_state=42)
```

Step 6: Model Testing

We will evaluate the performance of the trained classifier (FNN) 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 Feedforward Neural Network (FNN) Classifier:

```
# Decode one-hot encoded predictions to integer labels
fnn_predictions_decoded = np.argmax(fnn_predictions, axis=1)

# Test Feedforward Neural Network (FNN) Classifier
fnn_accuracy = accuracy_score(test_labels_encoded, fnn_predictions_decoded)
fnn_precision = precision_score(test_labels_encoded, fnn_predictions_decoded, average='weighted')
fnn_recall = recall_score(test_labels_encoded, fnn_predictions_decoded, average='weighted')
fnn_f1 = f1_score(test_labels_encoded, fnn_predictions_decoded, average='weighted')
fnn_conf_matrix = confusion_matrix(test_labels_encoded, fnn_predictions_decoded, labels=range(len(emotion_classes)))

print("FNN Accuracy:", fnn_accuracy)
print("FNN Precision:", fnn_precision)
print("FNN Recall:", fnn_recall)
print("FNN F1-score:", fnn_f1)
print("FNN Confusion Matrix:")
print(fnn_conf_matrix)
```

```
FNN Accuracy: 0.21904761904761905
FNN Precision: 0.3584767240229425
FNN Recall: 0.21904761904761905
FNN F1-score: 0.2151135718021572
FNN Confusion Matrix:
```

```
[[ 9  2  0  2  0  1  1]
 [11  2  1  0  0  1  0]
 [10  2  1  1  0  0  1]
 [11  1  0  3  0  0  0]
 [10  0  0  1  3  1  0]
 [11  1  0  0  1  1  1]
 [ 6  1  0  2  2  0  4]]
```

FNN Accuracy: 0.21905

> The accuracy of the Feedforward Neural Network (FNN) classifier is approximately 21.91%. This metric indicates the proportion of correctly classified instances out of the total number of instances in the test set.

FNN Precision: 0.3585

> The weighted average precision for the FNN classifier is about 0.3585. This suggests that, on average, when the FNN classifier predicts an emotion, it is correct about 35.85% of the time.

FNN Recall: 0.21905

> The weighted average recall for the FNN classifier is approximately 0.21905. It means that the classifier correctly identifies about 21.91% of the instances belonging to each emotion class.

FNN F1-score: 0.2151

> The weighted average F1-score for the FNN model is around 0.2151. This score indicates that the model achieves a balance between precision and recall, with a slight emphasis on balancing these metrics.

FNN Confusion Matrix:

> The confusion matrix provides insights into the classifier's performance for each emotion class. Key observations from the confusion matrix include:

- Instances on the diagonal (top-left to bottom-right) represent correctly classified instances for each emotion class.
- Off-diagonal values represent misclassifications. For instance, the element at row 1 and column 2 (1,2) indicates that two instances of class 1 were misclassified as class 2.
- Certain emotions, such as classes 4 and 6, exhibit higher misclassifications with other classes.

Interpretation and Insights:

1. The FNN model's accuracy, precision, recall, and F1-score are consistent with the performance of the previous classifiers. This suggests that, like the other models, the FNN model encounters challenges in effectively distinguishing between emotion classes.
2. The confusion matrix reinforces the model's struggles in accurately classifying certain emotions, as indicated by non-diagonal values.
3. The higher misclassifications for certain emotion classes could be attributed to either class imbalance or inherent similarities between those emotions.
4. Similar to the other classifiers, class 5 (neutral) seems to have relatively higher correct predictions compared to other classes.
5. The FNN model's performance aligns with the SVM and Random Forest models, indicating that further optimization or alternative approaches may be necessary to improve the model's performance.

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