1. **Labeled Faces in the Wild (LFW):**

محیط: برای تشخیص چهره بدون محدودیت در موقعیت های دنیای واقعی طراحی شده است. این شامل تصاویر جمع آوری شده از اینترنت، با تغییرات در ژست، بیان، روشنایی، و پس زمینه است.

تعداد تصاویر: حاوی بیش از 13000 تصویر برچسب گذاری شده از چهره از 5749 فرد است.

2. **پایگاه داده چهره ها ORL** :

محیط: مجموعه داده ORL برای تشخیص چهره در شرایط کنترل شده طراحی شده است. این در یک محیط آزمایشگاهی با تنوع در نور، حالات چهره، و جزئیات صورت جمع آوری شد.

تعداد تصاویر: مجموعه داده شامل 40 موضوع است که هر کدام دارای 10 تصویر است که در مجموع 400 تصویر ایجاد می شود.

3. **پایگاه داده چهره ها Yale** :

محیط: پایگاه داده چهره Yale نیز برای آزمایش های کنترل شده تشخیص چهره طراحی شده است. در یک محیط داخلی با تنوع در نور، بیان و ژست جمع آوری شد.

تعداد تصاویر: مجموعه داده شامل 15 موضوع است که هر کدام دارای 11 تصویر است که در مجموع 165 تصویر ایجاد می شود.

4. پایگاه داده پیشرفته **Yale Face B** :

محیط: شبیه به پایگاه داده اصلی Yale Face ، این مجموعه داده برای آزمایش‌های کنترل‌شده تشخیص چهره طراحی شده است. این شامل تصاویری با تنوع در نور، بیان و ژست است.

تعداد تصاویر: پایگاه داده توسعه یافته چهره Yale B شامل 38 موضوع است که هر کدام دارای 64 تصویر است که در مجموع 2416 تصویر به دست می آید.

5. پایگاه داده چهره ها AR :

محیط: پایگاه داده چهره AR برای تحقیقات تشخیص چهره طراحی شده است و شامل تصاویری با تغییرات در حالت، بیان، نور و انسداد است.

تعداد تصاویر: شامل بیش از 4000 تصویر رنگی از 126 نفر است که هر فرد به طور متوسط 26 تصویر دارد.

6. پایگاه داده چهره ها BioID :

محیط: پایگاه داده چهره BioID برای تحقیق در مورد تشخیص و تشخیص چهره طراحی شده است. این شامل تصاویر گرفته شده در شرایط مختلف دنیای واقعی، با تغییرات نور و پس زمینه است.

تعداد تصاویر: مجموعه داده شامل 1521 تصویر در مقیاس خاکستری از 23 فرد است.

1) برای پیاده سازی سیستم شناسایی چهره براساس ویژگی بافت Local Binary Pattern (LBP) :

```python
def calculate_texture_feature(image, method="uniform"):

    # Convert the input image to grayscale
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Set parameters for Local Binary Pattern (LBP) computation
    radius = 1
    n_points = 8 * radius

    # Compute LBP using the specified method
    lbp = feature.local_binary_pattern(gray, n_points, radius, method=method)

    # Calculate the histogram of LBP features
    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()

    return hist

def compare_texture_features(feature1, feature2):

    # Calculate the chi-square histogram distance between two texture feature vectors
    distance = np.sum((feature1 - feature2) ** 2 / (feature1 + feature2 + 1e-10))
    return distance
```

ابتدا یک تابع برای محاسبه ی ویژگی LBP به نام calculate_texture_feature تعریف میکنیم :

ورودی: یک تصویر و یک پارامتر برای روش محاسبه ی ویژگی LBP می گیرد (پیش فرض "uniform" است) .

تصویر ورودی را با استفاده از تابع cv2.cvtColor به مقیاس خاکستری تبدیل میکنیم

پارامترهایی را برای محاسبه LBP مانند شعاع و تعداد نقاط تنظیم میکنیم .و سپس LBP را با استفاده از روش مشخص شده ("uniform") محاسبه میکنیم.

هیستوگرام ویژگی های LBP را محاسبه میکنیم .هیستوگرام را با تقسیم هر تعداد bin بر تعداد کل نرمال سازی میکنیم .

خروجی: هیستوگرام نرمال شده ویژگی های LBP را برمی گرداند.

سپس یک تابع برای مقایسه هیستوگرام ویژگی های LBP به نام compare_texture_features تعریف میکنیم :

ورودی: دو بردار ویژگی LBP را می گیرد.(از خروجی تابع compare_texture_features بدست می آید)

فاصله هیستوگرام chi-square بین دو بردار ویژگی ورودی را محاسبه میکنیم . فاصله chi-square با استفاده از فرمول محاسبه می شود :

$$\sum_{i=1}^{n} \frac{(X_i - Y_i)^2}{X_i + Y_i + \epsilon}$$:

که Xi و Yi در آن مقادیر bin دو هیستوگرام مقایسه می شوند وε یک مقدار کوچک $(1\varepsilon - 10)$ است که برای جلوگیری از تقسیم بر صفر اضافه شده است.

خروجی: فاصله chi-square را به عنوان معیار عدم تشابه بین دو بردار ویژگی LBP برمی گرداند.

```
def analyze_similarity(input_photo, input_feature, distances):

    # Sort distances in ascending order
    distances.sort(key=lambda x: x[0])

    # Select the top 10 photos with the smallest distances
    top_10_photos = distances[:10]

    # Create titles for the top 10 photos
    top_10_titles = [f"({i+1})  label: {label}" for i, (dist, label, _) in enumerate(top_10_photos)]

    # Display the input photo and the top 10 similar photos
    display_images(input_photo, [photo for _, _, photo in top_10_photos], top_10_titles)

    # Count the number of correct matches in the top 10 and calculate accuracy
    correct_count = sum(1 for _, label, _ in top_10_photos if label == distances[0][1])
    accuracy_percentage = (correct_count / 10) * 100
    print(f"\033[1m{correct_count}\033[0m photos of those ten are in the same group as the input photo: \033[1m{accuracy_percent
```

حال یک تابع برای آنالیز شباهت بین تصویر ورودی و مجموعه ای از عکس ها، از جمله نمایش ۱۰ تصویر برتر مشابه همراه با برچسب آن ها و محاسبه دقت به نام analyze_similarity تعریف میکنیم :

این تابع 3 پارامتر به عنوان ورودی میگیرد : input_photo (تصویر ورودی)، input_feature (بردار ویژگی تصویر ورودی)، distances(برای ذخیره فواصل). در اینجا توضیحی در مورد هر بخش از تابع آورده شده است :

مرتب سازی فواصل : فواصل بین هیستوگرام ها ویژگی ها LBP به ترتیب صعودی با استفاده از distances.sort(key=lambda x: x[0])مرتب می شوند .

10عکس برتر : 10عکس برتر با کمترین فاصله از لیست فواصل مرتب شده انتخاب می شوند.تابع display_images برای نمایش عکس ورودی و 10 عکس برتر مشابه در یک شبکه فراخوانی می شود .عناوین هر تصویر بر اساس برچسب ها ایجاد می شود .

محاسبه دقت : تعداد عکس هایی که به درستی تطبیق داده شده اند (دارای برچسب مشابه با مشابه ترین عکس) در 10 عکس برتر محاسبه می شود.و همچنین درصد دقت بر اساس تعداد عکس هایی که به درستی تطبیق داده شده اند محاسبه می شود.

```python
def display_images(input_photo, similar_photos, titles, rows=4, cols=4):

    # Display input photo and similar photos with titles
    plt.figure(figsize=(20, 15))
    plt.subplot(rows, cols, 1)
    plt.imshow(cv2.cvtColor(input_photo, cv2.COLOR_BGR2RGB))
    plt.title("Input Photo")
    plt.axis("off")

    for i, (photo, title) in enumerate(zip(similar_photos, titles), start=1):
        plt.subplot(rows, cols, i + 1)
        plt.imshow(cv2.cvtColor(photo, cv2.COLOR_BGR2RGB))
        plt.title(title)
        plt.axis("off")

    plt.show()
```

یک تابع نیز برای نمایش بصری از عکس ورودی و مجموعه ای از عکس های مشابه با عناوین مربوطه به نام display_images تعریف میکنیم :

این تابع 4 پارامتر به عنوان ورودی میگیرد : input_photo (تصویر ورودی)، similar_photos (تصاویر مشابه)، titles (عناوین تصاویر) ، rows (تعداد سطرها) و cols (تعداد ستون ها) .

```python
def load_dataset(dataset_path):

    # Load images and corresponding labels from the dataset
    X = []  # List to store texture features
    y = []  # List to store corresponding labels

    # Loop through each folder in the dataset path
    for folder in os.listdir(dataset_path):
        folder_path = os.path.join(dataset_path, folder)

        # Check if the item in the folder path is a directory
        if os.path.isdir(folder_path):

            # Loop through each file in the folder
            photos = [filename for filename in os.listdir(folder_path) if os.path.isfile(os.path.join(folder_path, filename))]
            np.random.shuffle(photos)
            train_size = int(len(photos) * 0.8)
            train_data = photos[:train_size]

            for filename in train_data:
                file_path = os.path.join(folder_path, filename)

                if os.path.isfile(file_path):

                    # Read image, compute texture feature, and append to X and y
                    photo = cv2.imread(file_path)
                    photo_feature = calculate_texture_feature(photo)
                    X.append(photo_feature)
                    y.append(folder)

    return np.array(X), np.array(y)
```

یک تابع به نام load_dataset تعریف میکنیم که تصاویر را از یک مسیر داده مشخص شده می خواند، ویژگی های LBP را برای هر تصویر محاسبه می کند:

ورودی: آدرس قایل دیتاست (dataset_path) را به عنوان ورودی میگیرد.

راه اندازی لیست ها :دو لیست خالی X و y به ترتیب برای ذخیره ویژگی های LBP و برچسب های مربوطه ایجاد میکنیم .

حلقه زدن روی پوشه های مجموعه داده : تابع از طریق هر آیتم در data_path مشخص شده تکرار می شود .برای هر مورد، بررسی می کند که آیا یک دایرکتوری است (با استفاده از os.path.isdir)

Shuffle کردن و تقسیم داده های آموزشی : برای هر دایرکتوری، فایل‌ها (عکس‌ها) را در پوشه فهرست می‌کند، فهرست را به‌طور تصادفی به هم می‌زند(np.random.shuffle) ، و سپس 80 درصد عکس‌ها را به عنوان داده‌های آموزشی انتخاب می‌کند.

پردازش هر عکس آموزشی : برای هر فایل در داده های آموزشی، مسیر فایل کامل (file_path) را می سازد .بررسی می کند که آیا فایل وجود دارد و واقعاً یک فایل است (با استفاده ازos.path.isfile) اگر فایل معتبر باشد، تصویر را با استفاده از cv2.imread می خواند، ویژگی LBP را با استفاده از تابع account_texture_feature محاسبه می کند. و ویژگی را به لیست X اضافه می کند .برچسب (نام پوشه) نیز به لیست y اضافه می شود .

خروجی: در نهایت، این تابع آرایه‌های NumPy را برای X (ویژگی‌های LBP) و y (برچسب‌ها) برمی‌گرداند.

```python
if __name__ == "__main__":

    # Example usage of the functions
    input_photo_path = r"D:\Computer Vision\att_faces\s23\6.pgm"
    dataset_path = r"D:\Computer Vision\att_faces"

    input_photo = cv2.imread(input_photo_path)

    # Compute texture feature for the input photo
    input_feature = calculate_texture_feature(input_photo)

    distances = []
    for folder in os.listdir(dataset_path):
        folder_path = os.path.join(dataset_path, folder)

        if os.path.isdir(folder_path):

            for filename in os.listdir(folder_path):
                file_path = os.path.join(folder_path, filename)

                if os.path.isfile(file_path):

                    # Read image, compute texture feature, and calculate distance
                    photo = cv2.imread(file_path)
                    photo_feature = calculate_texture_feature(photo)
                    dist = compare_texture_features(input_feature, photo_feature)
                    distances.append((dist, folder, photo))

    # Analyze the similarity between the input photo and the dataset
    analyze_similarity(input_photo, input_feature, distances)
```

حال یک تابع main تعریف کرده تا از توابع نوشته شده برای مجموعه داده ATT استفاده کنیم :

مسیرهای عکس ورودی (input_photo_path) و مجموعه داده (dataset_path) مشخص میکنیم .

عکس ورودی با استفاده از cv2.imread از مسیر مشخص شده خوانده می شود.

ویژگی LBP برای عکس ورودی :ویژگی LBP برای عکس ورودی با استفاده از تابع account_texture_feature محاسبه می شود .

مقایسه عکس ورودی با مجموعه داده : سپس کد از طریق هر پوشه در مجموعه داده و هر فایل در آن پوشه ها تکرار می شود . برای هر فایل تصویر معتبر، تصویر را می‌خواند، ویژگی LBP آن را محاسبه می‌کند، و فاصله بین ویژگی LBP عکس ورودی و عکس فعلی در مجموعه داده را با استفاده از تابع compare_texture_features محاسبه می‌کند .فاصله به همراه نام پوشه و خود عکس به لیست فاصله ها اضافه می شود .

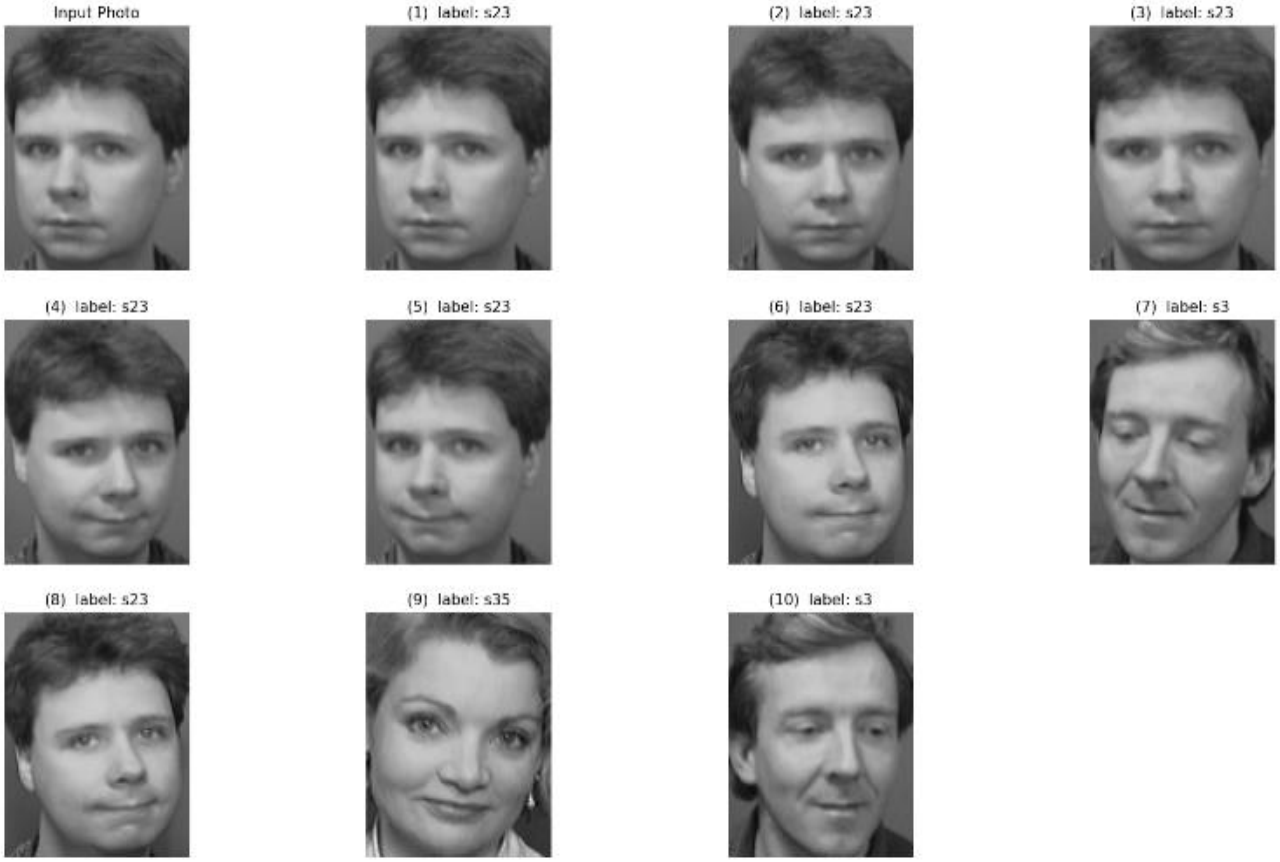
تجزیه و تحلیل شباهت : تابع analyzy_similarity با عکس ورودی، ویژگی محاسبه شده آن و لیست فاصله ها فراخوانی می شود .

نمایش نتایج : تابع analyzy_similarity به صورت داخلی عکس ورودی را به همراه 10 عکس مشابه برتر بر اساس فواصل محاسبه شده نمایش می دهد .همچنین تعداد موارد منطبق صحیح در 10 مورد برتر و درصد دقت را چاپ می کند.

خروجی کد های این قسمت برای مجموعه داده ATT به صورت زیر است :

خروجی برای عکس شماره ی **6** از پوشه ی **s23**:

10 تصویر برتر مشابه با تصویر ورودی به همراه lable مربوط به هر کدام در بالای آن مشخص شده است .

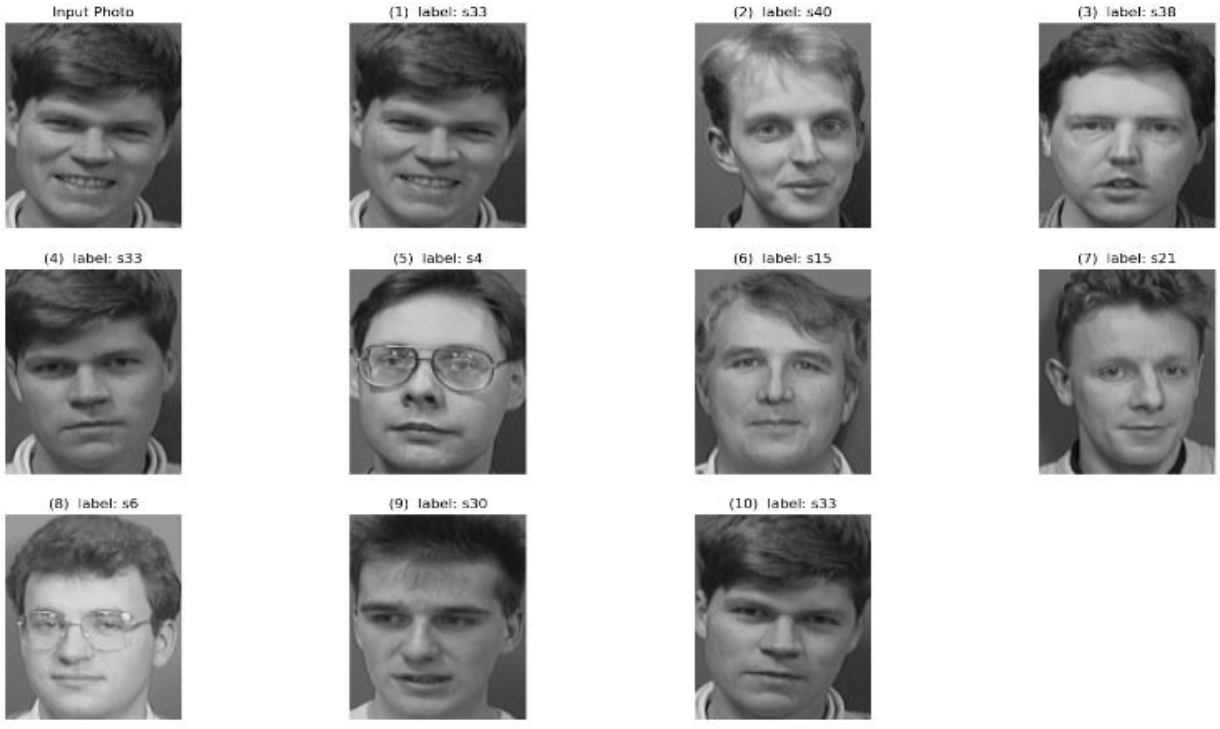


7 photos of those ten are in the same group as the input photo: **70.00%**

خروجی برای عکس شماره ی **5** از پوشه ی **s3**:

4 photos of those ten are in the same group as the input photo: **40.00%**

خروجی برای عکس شماره ی **4** از پوشه ی **s33**:



3 photos of those ten are in the same group as the input photo: **30.00%**

الف) برای اینکه درصد شناسایی را به دست بیاریم :

```
def train_knn(X_train, y_train, k=3):

    # Train a k-NN classifier with the specified number of neighbors
    knn_classifier = KNeighborsClassifier(n_neighbors=k)
    knn_classifier.fit(X_train, y_train)
    return knn_classifier

def evaluate_system(X_test, y_test, knn_classifier):

    # Evaluate the performance of the system on the test set
    y_pred = knn_classifier.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    confusion_mat = confusion_matrix(y_test, y_pred)
    return accuracy, confusion_mat
```

ابتدا یک تابع به نام train_knn تعریف میکنیم :

ورودی:

X_train: بردارهای ویژگی مجموعه آموزشی.

y_train: برچسب های مربوط به مجموعه آموزشی.

k: تعداد همسایگان طبقه‌بندی‌کننده k-NN (پیش‌فرض 3 است).

یک طبقه‌بندی‌کننده k-NN را با استفاده از KNeighborsClassifier از scikit-learn با تعداد همسایه‌های مشخص شده راه‌اندازی میکنیم.

طبقه بندی کننده را با استفاده از برازش به داده های آموزشی برازش می دهد.

خروجی:

k-NN آموزش دیده را برمی گرداند.

سپس یک تابع به نام evaluate_system تعریف میکنیم :

ورودی:

X_test: بردارهای ویژگی مجموعه تست.

y_test: برچسب های مربوط به مجموعه تست.

knn_classifier: k-NN آموزش دیده.

از طبقه‌بندی‌کننده k-NN آموزش‌دیده برای پیش‌بینی برچسب‌ها برای مجموعه آزمایشی با استفاده از پیش‌بینی استفاده میکنیم.

دقت پیش‌بینی‌ها را با استفاده از accuracy_score از scikit-learn محاسبه میکنیم.

و کانفیوژن ماتریس را با استفاده از confusion_matrix از scikit-learn محاسبه میکنیم.

خروجی:

یک تاپل حاوی دقت و کانفیوژن ماتریس را برمی‌گرداند.

ب) ابتدا یک تابع به نام find_cases_of_mistakes_together را تعریف میکنیم که جفت افرادی را که بیشتر با هم اشتباه می شوند را بر اساس ماتریس اشتباهات شناسایی کنیم :

```python
def find_cases_of_mistakes_together(mistakes_matrix, num_cases=2, labels=None):

    # Find pairs of people who make the most mistakes together
    flat_mistakes = mistakes_matrix.flatten()

    # Sort indices in descending order to identify pairs with the most mistakes
    sorted_indices = np.argsort(flat_mistakes)[::-1]

    selected_cases = []

    # Loop through sorted indices to find pairs
    for index in sorted_indices:
        person1, person2 = np.unravel_index(index, mistakes_matrix.shape)

        # Check if both individuals are from different classes
        if labels is not None and labels[person1] != labels[person2]:

            # Add 1 to convert from 0-based index to 1-based index
            selected_cases.append((person1 + 1, person2 + 1))

            # Stop when the desired number of pairs is reached
            if len(selected_cases) == num_cases:
                break

    if len(selected_cases) > 0:
        print("\nCases of People Who Make a Lot of Mistakes Together:")
        for case in selected_cases:
            print(f"People \033[1m{case[0]}\033[0m and \033[1m{case[1]}\033[0m")
```

ورودی: این تابع 3 پارامتر به عنوان ورودی میگیرد : mistakes_matrix (ماتریس اشتباهات)، num_cases (تعداد موارد منتخب)و labels (برچسب ها) .

flat_mistakes :با استفاده از روش flatten ماتریس اشتباهات را مسطح میکنیم . این کار مرتب‌سازی و یافتن جفت‌هایی را که بیشتر با هم اشتباه می شوند آسان‌تر می‌کند .

مرتب سازی شاخص ها به ترتیب نزولی sorted_indices :مرتب کردن ماتریس اشتباهات مسطح به ترتیب نزولی با استفاده از np.argsort(flat_mistakes)[::-1]به دست می آید .این شاخص های عناصر در ماتریس اشتباهات را به ترتیب نزولی مقادیر آنها ارائه می دهد.

سپس کد از میان شاخص های مرتب شده حلقه می زند تا جفت افرادی را پیدا کند که بیشتر با هم اشتباه می شوند .اندیس ها با استفاده از np.unravel_index (index,rrors_matrix.shape) به شاخص های ماتریس اصلی تبدیل می شوند .

کلاس های مختلف را بررسی میکنیم :اگر پارامتر labels ارائه شده باشد، بررسی می میکنیم که آیا هر دو فرد در جفت به کلاس های مختلف (برچسب) تعلق دارند یا خیر .این کار برای اطمینان از اینکه جفت های شناسایی شده از یک کلاس نیستند انجام میدهیم.

فهرستی از موارد منتخب ایجاد میکنیم :این تابع یک لیست selected_cases را ایجاد می کند که شامل چندین شاخص شخصی است که با هم اشتباه میشوند .

خروجی: در صورت وجود موارد منتخب، موارد افرادی را که بیشتر با هم اشتباه می شوند را چاپ می کند، از جمله شاخص های افراد درگیر.

در آخر با استفاده از توابع تعریف شده نتیجه را برای قسمت های **الف** و **ب** سوال جمع آوری کرده و خروجی را چاپ میکنیم.

```
def analyze_system(input_photo, input_feature, dataset_features, dataset_labels, k=3):

    # Split the dataset into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(dataset_features, dataset_labels, test_size=0.2, random_state=42)

    # Train a k-NN classifier
    knn_classifier = train_knn(X_train, y_train, k)

    # Evaluate the system's performance
    accuracy, confusion_mat = evaluate_system(X_test, y_test, knn_classifier)

    # Display results
    print("\033[1mA)\033[0m")
    print(f"\nAccuracy: \033[1m{accuracy * 100:.2f}%\033[0m\n")

    mistakes_matrix = confusion_mat - np.eye(len(confusion_mat))

    print("\033[1mB)\033[0m")
    print("\nMistakes Matrix:")
    print(mistakes_matrix)

    # Find cases where people make mistakes together
    find_cases_of_mistakes_together(mistakes_matrix, num_cases=2, labels=y_test)

if __name__ == "__main__":

    # Assuming load_dataset is defined elsewhere in your code
    X, y = load_dataset(dataset_path)

    # Analyze the system using the input photo and dataset
    analyze_system(input_photo, input_feature, X, y, k=3)
```

یک تابع به نام analyze_system تعریف میکنیم که تجزیه و تحلیل یک سیستم را برای شناسایی افراد بر اساس عکس های ورودی و ویژگی های آنها انجام می دهد. این تجزیه و تحلیل شامل آموزش یک طبقه‌بندی کننده k-NN، ارزیابی عملکرد آن و یافتن مواردی است که افراد با هم اشتباه میشوند.

تقسیم مجموعه داده :ویژگی‌های مجموعه داده و برچسب آن‌ها با استفاده از train_test_split از scikit-learn به مجموعه‌های آموزشی و آزمایشی تقسیم میکنیم. 80٪ از داده ها را برای آموزش (X_train, y_train) و 20٪ را برای تست (X_test, y_test) ذخیره میکنیم .

آموزش یک طبقه بندی کننده :k-NN با استفاده از تابع train_knn با مجموعه آموزشی (X_train, y_train) آموزش داده می شود .پارامتر k تعداد همسایگان را مشخص می کند

ارزیابی عملکرد سیستم :عملکرد سیستم بر روی مجموعه تست (X_test, y_test) با استفاده از تابع evaluate_system ارزیابی می شود .این تابع دقت را محاسبه می کند و یک کانفیوژن ماتریس ایجاد می کند

نمایش نتایج (قسمت **الف**): accuracy به عنوان بخش A از نتایج چاپ می شود.

ایجاد ماتریس اشتباهات(قسمت **ب**): ماتریس اشتباهات با کم کردن ماتریس identity از کانفیوژن ماتریس ایجاد می شود .این ماتریس نشان دهنده مواردی است که طبقه بندی کننده اشتباه کرده است (عناصر خارج از مورب) .ماتریس اشتباهات به عنوان قسمت B از نتایج چاپ می شود.

یافتن موارد اشتباه با هم: تابع find_cases_of_mistakes_together برای شناسایی جفت افرادی که با هم اشتباه می کنند فراخوانی می شود. پارامتر labels برای در نظر گرفتن اطلاعات کلاس روی y_test تنظیم شده است.

حال تابع main را تعریف میکنیم:

بارگذاری مجموعه داده :تابع load_dataset برای بارگذاری مجموعه داده و به دست آوردن بردارهای ویژگی (X) و برچسب های مربوطه (y) فراخوانی می شود.

سپس تابع analyzy_system با آرگومان های زیر فراخوانی می شود

input_photo : عکس ورودی که قرار است عملکرد سیستم برای آن تحلیل شود.

input_feature: ویژگی بافت محاسبه شده برای عکس ورودی

X: بردارهای ویژگی کل مجموعه داده

y : برچسب های مربوط به کل مجموعه داده

k=3 : تعداد همسایگان طبقه‌بندی‌کننده k-NN (پیش‌فرض 3 است).

خروجی کد های قسمت **الف** و **ب** برای عکس شماره ی **6** از پوشه ی **s23**:

```
A)

Accuracy: 29.69%

B)

Mistakes Matrix:
[[-1.  1.  0. ...  0.  0.  0.]
 [ 0. -1.  0. ...  0.  0.  0.]
 [ 0.  0.  0. ...  0.  0.  0.]
 ...
 [ 0.  0.  0. ...  0.  0.  0.]
 [ 0.  0.  0. ...  0.  0.  0.]
 [ 0.  0.  0. ...  0.  0. -1.]]

Cases of People Who Make a Lot of Mistakes Together:
People 10 and 7
People 31 and 3
```

(2

```python
def calculate_texture_feature_blocks(image, method="uniform", block_size=(3, 3)):

    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    radius = 1
    n_points = 8 * radius

    # Divide the image into blocks
    blocks = [gray[i:i + block_size[0], j:j + block_size[1]] for i in range(0, gray.shape[0], block_size[0]) for j in
              range(0, gray.shape[1], block_size[1])]

    # Calculate LBP features for each block
    block_features = []
    for block in blocks:
        lbp = feature.local_binary_pattern(block, n_points, radius, method=method)
        hist, _ = np.histogram(lbp.ravel(), bins=np.arange(0, n_points + 3), range=(0, n_points + 2))
        hist = hist.astype("float")
        hist /= hist.sum()
        block_features.append(hist)

    # Combine the block features into a single feature vector
    combined_feature = np.concatenate(block_features)
    return combined_feature
```

برای قسمت 2 سوال ابتدا یک تابع به نام calculate_texture_feature_blocks تعریف میکنیم که ویژگی های بافت یک تصویر را با تقسیم آن به بلوک ها و محاسبه ویژگی های LBP برای هر بلوک محاسبه می کند. بردار ویژگی به دست آمده ترکیبی از هیستوگرام های LBP برای همه بلوک ها است:

ورودی: یک تصویر، یک پارامتر برای روش محاسبه ی ویژگی LBP (پیش فرض "uniform " است)و سایز بلوک ها(در اینجا 3x3 در نظر گرفته شده است) به عنوان ورودی میگیرد.

تصویر ورودی با استفاده از cv2.cvtColor به مقیاس خاکستری تبدیل میکنیم

پارامترهایی را برای محاسبه LBP مانند شعاع و تعداد نقاط تنظیم میکنیم

سپس تصویر را با استفاده از حلقه های تو در تو برای تکرار در ردیف ها و ستون های تصویر به بلوک ها تقسیم میکنیم و برای هر بلوک، ویژگی های LBP با استفاده از تابع feature.local_binary_pattern محاسبه میکنیم .

هیستوگرام ویژگی های LBP با استفاده از np.histogram محاسبه و نرمال میکنیم.

هیستوگرام های نرمال شده برای همه بلوک ها در یک بردار ویژگی واحد (combined_feature) الحاق می شوند.

خروجی: بردار ویژگی ترکیبی از هیستوگرام های LBP برای همه بلوک ها را برمی گرداند.

حال تابع main را تعریف میکنیم:

```
if __name__ == "__main__":

    # Calculate texture feature for 3x3 blocks
    input_feature_blocks = calculate_texture_feature_blocks(input_photo)

    distances = []
    for folder in os.listdir(dataset_path):
        folder_path = os.path.join(dataset_path, folder)
        if os.path.isdir(folder_path):
            for filename in os.listdir(folder_path):
                file_path = os.path.join(folder_path, filename)
                if os.path.isfile(file_path):
                    photo = cv2.imread(file_path)

                    # Calculate texture feature for 3x3 blocks for each photo
                    photo_feature_blocks = calculate_texture_feature_blocks(photo)

                    # Compare the features
                    dist = compare_texture_features(input_feature_blocks, photo_feature_blocks)
                    distances.append((dist, folder, photo))

    analyze_similarity(input_photo, input_feature, distances)
    X, y = load_dataset(dataset_path)
    analyze_system(input_photo, input_feature_blocks, X, y, k=3)
```

تابع account_texture_feature_blocks برای محاسبه ویژگی های LBP برای عکس ورودی با استفاده از بلوک های 3x3 فراخوانی میکنیم .بردار ویژگی به دست آمده در input_feature_block ذخیره می شود .

سپس کد از طریق هر تصویر در مجموعه داده تکرار می‌شود، ویژگی‌های LBP را برای بلوک‌های 3×3 برای هر تصویر محاسبه می‌کند، ویژگی‌ها را با ویژگی‌های ورودی با استفاده از تابع compare_texture_features مقایسه می‌کند، و فاصله‌ها را همراه با اطلاعات پوشه و عکس در فهرست فاصله‌ها ذخیره می‌کند .

سپس تابع analyzy_similarity برای تجزیه و تحلیل شباهت بین عکس ورودی و مجموعه داده بر اساس فواصل محاسبه شده فراخوانی میکنیم .

سپس تابع load_dataset برای بارگذاری کل مجموعه داده فراخوانی میکنیم .

در نهایت، تابع analyzy_system برای ارزیابی عملکرد سیستم با استفاده از ویژگی‌های LBP عکس ورودی محاسبه شده با بلوک‌های3x3 (input_feature_blocks)، کل مجموعه داده (X and y) و یک k-NN با k=3 فراخوانی میکنیم.

خروجی کد های این قسمت برای مجموعه داده ATT به صورت زیر است :

خروجی برای عکس شماره ی **6** از پوشه ی **s23** :

10 تصویر برتر مشابه با تصویر ورودی به همراه lable مربوط به هر کدام در بالای آن مشخص شده است .

و همچنین خروجی قسمت **الف** و **ب** برای عکس شماره ی **6** از پوشه ی **s23**:

```
A)

Accuracy: 31.25%

B)

Mistakes Matrix:
[[ 0.  0.  0. ...  0.  0.  0.]
 [ 0.  0.  0. ...  0.  0.  0.]
 [ 0.  0.  0. ...  0.  0.  0.]
 ...
 [ 0.  0.  1. ... -1.  0.  0.]
 [ 0.  0.  0. ...  0. -1.  0.]
 [ 0.  0.  0. ...  0.  0.  0.]]

Cases of People Who Make a Lot of Mistakes Together:
People 37 and 7
People 26 and 9
```