Detect, Analyze, and Recognize Faces

The ability to detect, analyze, and recognize human faces is a core AI capability. In this exercise, you'll explore two Azure Cognitive Services that you can use to work with faces in images: the **Computer Vision** service, and the **Face** service.

Clone the repository for this course

If you have not already done so, you must clone the code repository for this course:

1. Start Visual Studio Code.
2. Open the palette (SHIFT+CTRL+P) and run a **Git: Clone** command to clone the `https://github.com/MicrosoftLearning/AI-102-AIEngineer` repository to a local folder (it doesn't matter which folder).
3. When the repository has been cloned, open the folder in Visual Studio Code.
4. Wait while additional files are installed to support the C# code projects in the repo.

> **Note**: If you are prompted to add required assets to build and debug, select **Not Now**.

Provision a Cognitive Services resource

If you don't already have one in your subscription, you'll need to provision a **Cognitive Services** resource.

1. Open the Azure portal at `https://portal.azure.com`, and sign in using the Microsoft account associated with your Azure subscription.
2. Select the **+ Create a resource** button, search for *cognitive services*, and create a **Cognitive Services** resource with the following settings:
    - **Subscription**: *Your Azure subscription*
    - **Resource group**: *Choose or create a resource group (if you are using a restricted subscription, you may not have permission to create a new resource group - use the one provided)*
    - **Region**: *Choose any available region*
    - **Name**: *Enter a unique name*
    - **Pricing tier**: Standard S0
3. Select the required checkboxes and create the resource.
4. Wait for deployment to complete, and then view the deployment details.
5. When the resource has been deployed, go to it and view its **Keys and Endpoint** page. You will need the endpoint and one of the keys from this page in the next procedure.

Prepare to use the Computer Vision SDK

In this exercise, you'll complete a partially implemented client application that uses the Computer Vision SDK to analyze faces in an image.

> **Note**: You can choose to use the SDK for either **C#** or **Python**. In the steps below, perform the actions appropriate for your preferred language.

1. In Visual Studio Code, in the **Explorer** pane, browse to the **19-face** folder and expand the **C-Sharp** or **Python** folder depending on your language preference.
2. Right-click the **computer-vision** folder and open an integrated terminal. Then install the Computer Vision SDK package by running the appropriate command for your language preference:

    C#

    Code Copy

```
dotnet add package Microsoft.Azure.CognitiveServices.Vision.ComputerVision --version 6.0.0
```

Python

Code Copy

```
pip install azure-cognitiveservices-vision-computervision==0.7.0
```

3. View the contents of the **computer-vision** folder, and note that it contains a file for configuration settings:
    - **C#**: appsettings.json
    - **Python**: .env
4. Open the configuration file and update the configuration values it contains to reflect the **endpoint** and an authentication **key** for your cognitive services resource. Save your changes.
5. Note that the **computer-vision** folder contains a code file for the client application:
    - **C#**: Program.cs
    - **Python**: detect-faces.py
6. Open the code file and at the top, under the existing namespace references, find the comment **Import namespaces**. Then, under this comment, add the following language-specific code to import the namespaces you will need to use the Computer Vision SDK:

C#

Code Copy

```
// import namespaces
using Microsoft.Azure.CognitiveServices.Vision.ComputerVision;
using Microsoft.Azure.CognitiveServices.Vision.ComputerVision.Models;
```

Python

Code Copy

```
# import namespaces
from azure.cognitiveservices.vision.computervision import ComputerVisionClient
from azure.cognitiveservices.vision.computervision.models import VisualFeatureTypes
from msrest.authentication import CognitiveServicesCredentials
```

View the image you will analyze

In this exercise, you will use the Computer Vision service to analyze an image of people.

1. In Visual Studio Code, expand the **computer-vision** folder and the **images** folder it contains.
2. Select the **people.jpg** image to view it.

Detect faces in an image

Now you're ready to use the SDK to call the Computer Vision service and detect faces in an image.

1. In the code file for your client application (**Program.cs** or **detect-faces.py**), in the **Main** function, note that the code to load the configuration settings has been provided. Then find the comment **Authenticate Computer Vision client**. Then, under this comment, add the following language-specific code to create and authenticate a Computer Vision client object:

C#

Code Copy

```
// Authenticate Computer Vision client
ApiKeyServiceClientCredentials credentials = new ApiKeyServiceClientCredentials(cogSvcKey);
cvClient = new ComputerVisionClient(credentials)
{
    Endpoint = cogSvcEndpoint
};
```

Python

Code Copy

```
# Authenticate Computer Vision client
credential = CognitiveServicesCredentials(cog_key)
cv_client = ComputerVisionClient(cog_endpoint, credential)
```

2. In the **Main** function, under the code you just added, note that the code specifies the path to an image file and then passes the image path to a function named **AnalyzeFaces**. This function is not yet fully implemented.

3. In the **AnalyzeFaces** function, under the comment **Specify features to be retrieved (faces)**, add the following code:

C#

Code Copy

```
// Specify features to be retrieved (faces)
List<VisualFeatureTypes?> features = new List<VisualFeatureTypes?>()
{
    VisualFeatureTypes.Faces
};
```

Python

Code Copy

```
# Specify features to be retrieved (faces)
features = [VisualFeatureTypes.faces]
```

4. In the **AnalyzeFaces** function, under the comment **Get image analysis**, add the following code:

C#

Code Copy

```
// Get image analysis
using (var imageData = File.OpenRead(imageFile))
{
    var analysis = await cvClient.AnalyzeImageInStreamAsync(imageData, features);

    // Get faces
    if (analysis.Faces.Count > 0)
    {
        Console.WriteLine($"{analysis.Faces.Count} faces detected.");

        // Prepare image for drawing
        Image image = Image.FromFile(imageFile);
        Graphics graphics = Graphics.FromImage(image);
        Pen pen = new Pen(Color.LightGreen, 3);
        Font font = new Font("Arial", 3);
        SolidBrush brush = new SolidBrush(Color.LightGreen);

        // Draw and annotate each face
        foreach (var face in analysis.Faces)
        {
            var r = face.FaceRectangle;
            Rectangle rect = new Rectangle(r.Left, r.Top, r.Width, r.Height);
            graphics.DrawRectangle(pen, rect);
            string annotation = $"Person aged approximately {face.Age}";
            graphics.DrawString(annotation,font,brush,r.Left, r.Top);
        }

        // Save annotated image
        String output_file = "detected_faces.jpg";
        image.Save(output_file);
        Console.WriteLine(" Results saved in " + output_file);
    }
}
```

Python

Code Copy

```
# Get image analysis
with open(image_file, mode="rb") as image_data:
    analysis = cv_client.analyze_image_in_stream(image_data , features)

    # Get faces
    if analysis.faces:
        print(len(analysis.faces), 'faces detected.')

        # Prepare image for drawing
        fig = plt.figure(figsize=(8, 6))
        plt.axis('off')
        image = Image.open(image_file)
        draw = ImageDraw.Draw(image)
        color = 'lightgreen'

        # Draw and annotate each face
        for face in analysis.faces:
            r = face.face_rectangle
            bounding_box = ((r.left, r.top), (r.left + r.width, r.top + r.height))
            draw = ImageDraw.Draw(image)
            draw.rectangle(bounding_box, outline=color, width=5)
            annotation = 'Person aged approximately {}'.format(face.age)
            plt.annotate(annotation,(r.left, r.top), backgroundcolor=color)

        # Save annotated image
        plt.imshow(image)
        outputfile = 'detected_faces.jpg'
        fig.savefig(outputfile)

        print('Results saved in', outputfile)
```

1. Save your changes and return to the integrated terminal for the **computer-vision** folder, and enter the following command to run the program:

   C#

   Code Copy

   ```
   dotnet run
   ```

   Python

   Code Copy

   ```
   python detect-faces.py
   ```

2. Observe the output, which should indicate the number of faces detected.
3. View the **detected_faces.jpg** file that is generated in the same folder as your code file to see the annotated faces. In this case, your code has used the attributes of the face to estimate the age of each person in the image, and the bounding box coordinates to draw a rectangle around each face.

Prepare to use the Face SDK

While the **Computer Vision** service offers basic face detection (along with many other image analysis capabilities), the **Face** service provides more comprehensive functionality for facial analysis and recognition.

1. In Visual Studio Code, in the **Explorer** pane, browse to the **19-face** folder and expand the **C-Sharp** or **Python** folder depending on your language preference.

2. Right-click the **face-api** folder and open an integrated terminal. Then install the Face SDK package by running the appropriate command for your language preference:

   C#

   Code Copy

   ```
   dotnet add package Microsoft.Azure.CognitiveServices.Vision.Face --version 2.6.0-preview.1
   ```

   Python

   Code Copy

   ```
   pip install azure-cognitiveservices-vision-face==0.4.1
   ```

3. View the contents of the **face-api** folder, and note that it contains a file for configuration settings:

   - **C#**: appsettings.json
   - **Python**: .env

4. Open the configuration file and update the configuration values it contains to reflect the **endpoint** and an authentication **key** for your cognitive services resource. Save your changes.

5. Note that the **face-api** folder contains a code file for the client application:

   - **C#**: Program.cs
   - **Python**: analyze-faces.py

6. Open the code file and at the top, under the existing namespace references, find the comment **Import namespaces**. Then, under this comment, add the following language-specific code to import the namespaces you will need to use the Computer Vision SDK:

   C#

   Code Copy

   ```
   // Import namespaces
   using Microsoft.Azure.CognitiveServices.Vision.Face;
   using Microsoft.Azure.CognitiveServices.Vision.Face.Models;
   ```

   Python

   Code Copy

   ```
   # Import namespaces
   from azure.cognitiveservices.vision.face import FaceClient
   from azure.cognitiveservices.vision.face.models import FaceAttributeType
   from msrest.authentication import CognitiveServicesCredentials
   ```

7. In the **Main** function, note that the code to load the configuration settings has been provided. Then find the comment **Authenticate Face client**. Then, under this comment, add the following language-specific code to create and authenticate a **FaceClient** object:

   C#

   Code Copy

```
// Authenticate Face client
ApiKeyServiceClientCredentials credentials = new ApiKeyServiceClientCredentials(cogSvcKey);
faceClient = new FaceClient(credentials)
{
    Endpoint = cogSvcEndpoint
};
```

Python

Code Copy

```
# Authenticate Face client
credentials = CognitiveServicesCredentials(cog_key)
face_client = FaceClient(cog_endpoint, credentials)
```

8. In the **Main** function, under the code you just added, note that the code displays a menu that enables you to call functions in your code to explore the capabilities of the Face service. You will implement these functions in the remainder of this exercise.

Detect and analyze faces

One of the most fundamental capabilities of the Face service is to detect faces in an image, and determine their attributes, such as age, emotional expressions, hair color, the presence of spectacles, and so on.

1. In the code file for your application, in the **Main** function, examine the code that runs if the user selects menu option **1**. This code calls the **DetectFaces** function, passing the path to an image file.
2. Find the **DetectFaces** function in the code file, and under the comment **Specify facial features to be retrieved**, add the following code:

C#

Code Copy

```
// Specify facial features to be retrieved
List<FaceAttributeType?> features = new List<FaceAttributeType?>
{
    FaceAttributeType.Age,
    FaceAttributeType.Emotion,
    FaceAttributeType.Glasses
};
```

Python

Code Copy

```
# Specify facial features to be retrieved
features = [FaceAttributeType.age,
            FaceAttributeType.emotion,
            FaceAttributeType.glasses]
```

3. In the **DetectFaces** function, under the code you just added, find the comment **Get faces** and add the following code:

C#

Code Copy

```
// Get faces
using (var imageData = File.OpenRead(imageFile))
{
    var detected_faces = await faceClient.Face.DetectWithStreamAsync(imageData,
returnFaceAttributes: features);

    if (detected_faces.Count > 0)
    {
        Console.WriteLine($"{detected_faces.Count} faces detected.");

        // Prepare image for drawing
        Image image = Image.FromFile(imageFile);
        Graphics graphics = Graphics.FromImage(image);
        Pen pen = new Pen(Color.LightGreen, 3);
        Font font = new Font("Arial", 4);
        SolidBrush brush = new SolidBrush(Color.Black);

        // Draw and annotate each face
        foreach (var face in detected_faces)
        {
            // Get face properties
            Console.WriteLine($"\nFace ID: {face.FaceId}");
            Console.WriteLine($" - Age: {face.FaceAttributes.Age}");
            Console.WriteLine($" - Emotions:");
            foreach (var emotion in face.FaceAttributes.Emotion.ToRankedList())
            {
                Console.WriteLine($"    - {emotion}");
            }

            Console.WriteLine($" - Glasses: {face.FaceAttributes.Glasses}");

            // Draw and annotate face
            var r = face.FaceRectangle;
            Rectangle rect = new Rectangle(r.Left, r.Top, r.Width, r.Height);
            graphics.DrawRectangle(pen, rect);
            string annotation = $"Face ID: {face.FaceId}";
            graphics.DrawString(annotation,font,brush,r.Left, r.Top);
        }

        // Save annotated image
        String output_file = "detected_faces.jpg";
        image.Save(output_file);
        Console.WriteLine(" Results saved in " + output_file);
    }
}
```

Python

Code Copy

```
# Get faces
with open(image_file, mode="rb") as image_data:
    detected_faces = face_client.face.detect_with_stream(image=image_data,
                                                         return_face_attributes=features)

    if len(detected_faces) > 0:
        print(len(detected_faces), 'faces detected.')

        # Prepare image for drawing
        fig = plt.figure(figsize=(8, 6))
        plt.axis('off')
        image = Image.open(image_file)
        draw = ImageDraw.Draw(image)
        color = 'lightgreen'

        # Draw and annotate each face
        for face in detected_faces:

            # Get face properties
            print('\nFace ID: {}'.format(face.face_id))
            detected_attributes = face.face_attributes.as_dict()
            age = 'age unknown' if 'age' not in detected_attributes.keys() else
int(detected_attributes['age'])
            print(' - Age: {}'.format(age))

            if 'emotion' in detected_attributes:
                print(' - Emotions:')
                for emotion_name in detected_attributes['emotion']:
                    print('   - {}: {}'.format(emotion_name, detected_attributes['emotion']
[emotion_name]))

            if 'glasses' in detected_attributes:
                print(' - Glasses:{}'.format(detected_attributes['glasses']))

            # Draw and annotate face
            r = face.face_rectangle
            bounding_box = ((r.left, r.top), (r.left + r.width, r.top + r.height))
            draw = ImageDraw.Draw(image)
            draw.rectangle(bounding_box, outline=color, width=5)
            annotation = 'Face ID: {}'.format(face.face_id)
            plt.annotate(annotation,(r.left, r.top), backgroundcolor=color)

        # Save annotated image
        plt.imshow(image)
        outputfile = 'detected_faces.jpg'
        fig.savefig(outputfile)

        print('\nResults saved in', outputfile)
```

1. Examine the code you added to the **DetectFaces** function. It analyzes an image file and detects any faces it contains, including attributes for age, emotions, and the presence of spectacles. The details of each face are displayed, including a unique face identifier that is assigned to each face; and the location of the faces is indicated on the image using a bounding box.

2. Save your changes and return to the integrated terminal for the **face-api** folder, and enter the following command to run the program:

   C#

Code Copy

```
dotnet run
```

*The C# output may display warnings about asynchronous functions now using the **await** operator. You can ignore these.*

Python

Code Copy

```
python analyze-faces.py
```

3. When prompted, enter **1** and observe the output, which should include the ID and attributes of each face detected.
4. View the **detected_faces.jpg** file that is generated in the same folder as your code file to see the annotated faces.

Compare faces

A common task is to compare faces, and find faces that are similar. In this scenario, you do not need to *identify* the person in the images, just determine whether multiple images show the *same* person (or at least someone who looks similar).

1. In the code file for your application, in the **Main** function, examine the code that runs if the user selects menu option **2**. This code calls the **CompareFaces** function, passing the path to two image files (**person1.jpg** and **people.jpg**).
2. Find the **CompareFaces** function in the code file, and under the existing code that prints a message to the console, add the following code:

C#

Code Copy

```
// Determine if the face in image 1 is also in image 2
DetectedFace image_i_face;
using (var image1Data = File.OpenRead(image1))
{
    // Get the first face in image 1
    var image1_faces = await faceClient.Face.DetectWithStreamAsync(image1Data);
    if (image1_faces.Count > 0)
    {
        image_i_face = image1_faces[0];
        Image img1 = Image.FromFile(image1);
        Graphics graphics = Graphics.FromImage(img1);
        Pen pen = new Pen(Color.LightGreen, 3);
        var r = image_i_face.FaceRectangle;
        Rectangle rect = new Rectangle(r.Left, r.Top, r.Width, r.Height);
        graphics.DrawRectangle(pen, rect);
        String output_file = "face_to_match.jpg";
        img1.Save(output_file);
        Console.WriteLine(" Results saved in " + output_file);

        //Get all the faces in image 2
        using (var image2Data = File.OpenRead(image2))
        {
            var image2Faces = await faceClient.Face.DetectWithStreamAsync(image2Data);

            // Get faces
            if (image2Faces.Count > 0)
            {

                var image2FaceIds = image2Faces.Select(f => f.FaceId).ToList<Guid?>();
                var similarFaces = await
faceClient.Face.FindSimilarAsync((Guid)image_i_face.FaceId,faceIds:image2FaceIds);
                var similarFaceIds = similarFaces.Select(f => f.FaceId).ToList<Guid?>();

                // Prepare image for drawing
                Image img2 = Image.FromFile(image2);
                Graphics graphics2 = Graphics.FromImage(img2);
                Pen pen2 = new Pen(Color.LightGreen, 3);
                Font font2 = new Font("Arial", 4);
                SolidBrush brush2 = new SolidBrush(Color.Black);

                // Draw and annotate each face
                foreach (var face in image2Faces)
                {
                    if (similarFaceIds.Contains(face.FaceId))
                    {
                        // Draw and annotate face
                        var r2 = face.FaceRectangle;
                        Rectangle rect2 = new Rectangle(r2.Left, r2.Top, r2.Width, r2.Height);
                        graphics2.DrawRectangle(pen2, rect2);
                        string annotation = "Match!";
                        graphics2.DrawString(annotation,font2,brush2,r2.Left, r2.Top);
                    }
                }

                // Save annotated image
                String output_file2 = "matched_faces.jpg";
                img2.Save(output_file2);
                Console.WriteLine(" Results saved in " + output_file2);
```

```
            }
        }

    }
}
```

Python

Code Copy

```
# Determine if the face in image 1 is also in image 2
with open(image_1, mode="rb") as image_data:
    # Get the first face in image 1
    image_1_faces = face_client.face.detect_with_stream(image=image_data)
    image_1_face = image_1_faces[0]

    # Highlight the face in the image
    fig = plt.figure(figsize=(8, 6))
    plt.axis('off')
    image = Image.open(image_1)
    draw = ImageDraw.Draw(image)
    color = 'lightgreen'
    r = image_1_face.face_rectangle
    bounding_box = ((r.left, r.top), (r.left + r.width, r.top + r.height))
    draw = ImageDraw.Draw(image)
    draw.rectangle(bounding_box, outline=color, width=5)
    plt.imshow(image)
    outputfile = 'face_to_match.jpg'
    fig.savefig(outputfile)

# Get all the faces in image 2
with open(image_2, mode="rb") as image_data:
    image_2_faces = face_client.face.detect_with_stream(image=image_data)
    image_2_face_ids = list(map(lambda face: face.face_id, image_2_faces))

    # Find faces in image 2 that are similar to the one in image 1
    similar_faces = face_client.face.find_similar(face_id=image_1_face.face_id,
face_ids=image_2_face_ids)
    similar_face_ids = list(map(lambda face: face.face_id, similar_faces))

    # Prepare image for drawing
    fig = plt.figure(figsize=(8, 6))
    plt.axis('off')
    image = Image.open(image_2)
    draw = ImageDraw.Draw(image)

    # Draw and annotate matching faces
    for face in image_2_faces:
        if face.face_id in similar_face_ids:
            r = face.face_rectangle
            bounding_box = ((r.left, r.top), (r.left + r.width, r.top + r.height))
            draw = ImageDraw.Draw(image)
            draw.rectangle(bounding_box, outline='lightgreen', width=10)
            plt.annotate('Match!',(r.left, r.top + r.height + 15), backgroundcolor='white')

    # Save annotated image
    plt.imshow(image)
    outputfile = 'matched_faces.jpg'
    fig.savefig(outputfile)
```

1. Examine the code you added to the **CompareFaces** function. It finds the first face in image 1 and annotates it in a new image file named **face_to_match.jpg**. Then it finds all of the faces in image 2, and uses their face IDs to find the ones that are similar to image 1. The similar ones are annotated and saved in a new image named **matched_faces.jpg**.

2. Save your changes and return to the integrated terminal for the **face-api** folder, and enter the following command to run the program:

C#

```
dotnet run
```

*The C# output may display warnings about asynchronous functions now using the **await** operator. You can ignore these.*

Python

```
python analyze-faces.py
```

3. When prompted, enter **2** and observe the output. Then view the **face_to_match.jpg** and **matched_faces.jpg** files that are generated in the same folder as your code file to see the matched faces.
4. Edit the code in the **Main** function for menu option **2** to compare **person2.jpg** to **people.jpg** and then re-run the program and view the results.

Train a facial recognition model

There may be scenarios where you need to maintain a model of specific people whose faces can be recognized by an AI application. For example, to facilitate a biometric security system that uses facial recognition to verify the identity of employees in a secure building.

1. In the code file for your application, in the **Main** function, examine the code that runs if the user selects menu option **3**. This code calls the **TrainModel** function, passing the name and ID for a new **PersonGroup** that will be registered in your cognitive services resource, and a list of employee names.
2. In the **face-api/images** folder, observe that there are folders with the same names as the employees. Each folder contains mutliple images of the named employee.
3. Find the **TrainModel** function in the code file, and under the existing code that prints a message to the console, add the following code:

C#

```
```

```
// Delete group if it already exists
var groups = await faceClient.PersonGroup.ListAsync();
foreach(var group in groups)
{
    if (group.PersonGroupId == groupId)
    {
        await faceClient.PersonGroup.DeleteAsync(groupId);
    }
}

// Create the group
await faceClient.PersonGroup.CreateAsync(groupId, groupName);
Console.WriteLine("Group created!");

// Add each person to the group
Console.Write("Adding people to the group...");
foreach(var personName in imageFolders)
{
    // Add the person
    var person = await faceClient.PersonGroupPerson.CreateAsync(groupId, personName);

    // Add multiple photo's of the person
    string[] images = Directory.GetFiles("images/" + personName);
    foreach(var image in images)
    {
        using (var imageData = File.OpenRead(image))
        {
            await faceClient.PersonGroupPerson.AddFaceFromStreamAsync(groupId, person.PersonId,
imageData);
        }
    }

}

    // Train the model
Console.WriteLine("Training model...");
await faceClient.PersonGroup.TrainAsync(groupId);

// Get the list of people in the group
Console.WriteLine("Facial recognition model trained with the following people:");
var people = await faceClient.PersonGroupPerson.ListAsync(groupId);
foreach(var person in people)
{
    Console.WriteLine($"-{person.Name}");
}
```

Python

Code Copy

```
# Delete group if it already exists
groups = face_client.person_group.list()
for group in groups:
    if group.person_group_id == group_id:
        face_client.person_group.delete(group_id)

# Create the group
face_client.person_group.create(group_id, group_name)
print ('Group created!')

# Add each person to the group
print('Adding people to the group...')
for person_name in image_folders:
    # Add the person
    person = face_client.person_group_person.create(group_id, person_name)

    # Add multiple photo's of the person
    folder = os.path.join('images', person_name)
    person_pics = os.listdir(folder)
    for pic in person_pics:
        img_path = os.path.join(folder, pic)
        img_stream = open(img_path, "rb")
        face_client.person_group_person.add_face_from_stream(group_id, person.person_id,
img_stream)

# Train the model
print('Training model...')
face_client.person_group.train(group_id)

# Get the list of people in the group
print('Facial recognition model trained with the following people:')
people = face_client.person_group_person.list(group_id)
for person in people:
    print('-', person.name)
```

1. Examine the code you added to the **TrainModel** function. It performs the following tasks:
    - Gets a list of **PersonGroup**s registered in the service, and deletes the specified one if it exists.
    - Creates a group with the specified ID and name.
    - Adds a person to the group for each name specified, and adds the multiple images of each person.
    - Trains a facial recognition model based on the named people in the group and their face images.
    - Retrieves a list of the named people in the group and displays them.
2. Save your changes and return to the integrated terminal for the **face-api** folder, and enter the following command to run the program:

    C#

    Code Copy

    ```
    dotnet run
    ```

    *The C# output may display warnings about asynchronous functions now using the **await** operator. You can ignore these.*

    Python

    Code Copy

```
python analyze-faces.py
```

3. When prompted, enter **3** and observe the output, noting that the **PersonGroup** created includes two people.

Recognize faces

Now that you have defined a **PeopleGroup** and trained a facial recognition model, you can use it to recognize named individuals in an image.

1. In the code file for your application, in the **Main** function, examine the code that runs if the user selects menu option **4**. This code calls the **RecognizeFaces** function, passing the path to an image file (**people.jpg**) and the ID of the **PeopleGroup** to be used for face identification.
2. Find the **RecognizeFaces** function in the code file, and under the existing code that prints a message to the console, add the following code:

C#

Code Copy

```
// Detect faces in the image
using (var imageData = File.OpenRead(imageFile))
{
    var detectedFaces = await faceClient.Face.DetectWithStreamAsync(imageData);

    // Get faces
    if (detectedFaces.Count > 0)
    {

        // Get a list of face IDs
        var faceIds = detectedFaces.Select(f => f.FaceId).ToList<Guid?>();

        // Identify the faces in the people group
        var recognizedFaces = await faceClient.Face.IdentifyAsync(faceIds, groupId);

        // Get names for recognized faces
        var faceNames = new Dictionary<Guid?, string>();
        if (recognizedFaces.Count> 0)
        {
            foreach(var face in recognizedFaces)
            {
                var person = await faceClient.PersonGroupPerson.GetAsync(groupId,
face.Candidates[0].PersonId);
                Console.WriteLine($"-{person.Name}");
                faceNames.Add(face.FaceId, person.Name);

            }
        }


        // Annotate faces in image
        Image image = Image.FromFile(imageFile);
        Graphics graphics = Graphics.FromImage(image);
        Pen penYes = new Pen(Color.LightGreen, 3);
        Pen penNo = new Pen(Color.Magenta, 3);
        Font font = new Font("Arial", 4);
        SolidBrush brush = new SolidBrush(Color.Cyan);
        foreach (var face in detectedFaces)
        {
            var r = face.FaceRectangle;
            Rectangle rect = new Rectangle(r.Left, r.Top, r.Width, r.Height);
            if (faceNames.ContainsKey(face.FaceId))
            {
                // If the face is recognized, annotate in green with the name
                graphics.DrawRectangle(penYes, rect);
                string personName = faceNames[face.FaceId];
                graphics.DrawString(personName,font,brush,r.Left, r.Top);
            }
            else
            {
                // Otherwise, just annotate the unrecognized face in magenta
                graphics.DrawRectangle(penNo, rect);
            }
        }

        // Save annotated image
        String output_file = "recognized_faces.jpg";
        image.Save(output_file);
```

```
        Console.WriteLine("Results saved in " + output_file);
    }
}
```

Python

Code Copy

```
# Detect faces in the image
with open(image_file, mode="rb") as image_data:

    # Get faces
    detected_faces = face_client.face.detect_with_stream(image=image_data)

    # Get a list of face IDs
    face_ids = list(map(lambda face: face.face_id, detected_faces))

    # Identify the faces in the people group
    recognized_faces = face_client.face.identify(face_ids, group_id)

    # Get names for recognized faces
    face_names = {}
    if len(recognized_faces) > 0:
        print(len(recognized_faces), 'faces recognized.')
        for face in recognized_faces:
            person_name = face_client.person_group_person.get(group_id,
face.candidates[0].person_id).name
            print('-', person_name)
            face_names[face.face_id] = person_name

    # Annotate faces in image
    fig = plt.figure(figsize=(8, 6))
    plt.axis('off')
    image = Image.open(image_file)
    draw = ImageDraw.Draw(image)
    for face in detected_faces:
        r = face.face_rectangle
        bounding_box = ((r.left, r.top), (r.left + r.width, r.top + r.height))
        draw = ImageDraw.Draw(image)
        if face.face_id in face_names:
            # If the face is recognized, annotate in green with the name
            draw.rectangle(bounding_box, outline='lightgreen', width=3)
            plt.annotate(face_names[face.face_id],
                         (r.left, r.top + r.height + 15), backgroundcolor='white')
        else:
            # Otherwise, just annotate the unrecognized face in magenta
            draw.rectangle(bounding_box, outline='magenta', width=3)

    # Save annotated image
    plt.imshow(image)
    outputfile = 'recognized_faces.jpg'
    fig.savefig(outputfile)

    print('\nResults saved in', outputfile)
```

1. Examine the code you added to the **RecognizeFaces** function. It finds the faces in the image and creates a list of their IDs. Then it uses the people group to try to identify the faces in the list of face IDs. The

recognized faces are annotated wuth the name of the identified person, and the results are saved in **recognized_faces.jpg**.

2. Save your changes and return to the integrated terminal for the **face-api** folder, and enter the following command to run the program:

   C#

   Code Copy

   ```
   dotnet run
   ```

   *The C# output may display warnings about asynchronous functions now using the **await** operator. You can ignore these.*

   Python

   Code Copy

   ```
   python analyze-faces.py
   ```

3. When prompted, enter **4** and observe the output. Then view the **recognized_faces.jpg** file that is generated in the same folder as your code file to see the identified people.
4. Edit the code in the **Main** function for menu option **4** to to recognize faces in **people2.jpg** and then re-run the program and view the results. The same people should be recognized.

Verify a face

Facial recognition is often used for identity verification. With the Face service, you can verify a face in an image by comparing it to another face, or from a person registered in a **PersonGroup**.

1. In the code file for your application, in the **Main** function, examine the code that runs if the user selects menu option **5**. This code calls the **VerifyFace** function, passing the path to an image file (**person1.jpg**), a name, and the ID of the **PeopleGroup** to be used for face identification.
2. Find the **VerifyFace** function in the code file, and under the comment **Get the ID of the person from the people group** (above the code that prints the result) add the following code:

C#

Code Copy

```
// Get the ID of the person from the people group
var people = await faceClient.PersonGroupPerson.ListAsync(groupId);
foreach(var person in people)
{
    if (person.Name == personName)
    {
        Guid personId = person.PersonId;

        // Get the first face in the image
        using (var imageData = File.OpenRead(personImage))
        {
            var faces = await faceClient.Face.DetectWithStreamAsync(imageData);
            if (faces.Count > 0)
            {
                Guid faceId = (Guid)faces[0].FaceId;

                //We have a face and an ID. Do they match?
                var verification = await faceClient.Face.VerifyFaceToPersonAsync(faceId,
personId, groupId);
                if (verification.IsIdentical)
                {
                    result = "Verified";
                }
            }
        }
    }
}
```

Python

Code Copy

```
# Get the ID of the person from the people group
people = face_client.person_group_person.list(group_id)
for person in people:
    if person.name == person_name:
        person_id = person.person_id

        # Get the first face in the image
        with open(person_image, mode="rb") as image_data:
            faces = face_client.face.detect_with_stream(image=image_data)
            if len(faces) > 0:
                face_id = faces[0].face_id

                # We have a face and an ID. Do they match?
                verification = face_client.face.verify_face_to_person(face_id, person_id,
group_id)
                if verification.is_identical:
                    result = 'Verified'
```

1. Examine the code you added to the **VerifyFace** function. It looks up the ID of the person in the group with the specified name. If the person exists, the code gets the face ID of the first face in the image. Finally, if there is a face in the image, the code verifies it against the ID of the specified person.

2. Save your changes and return to the integrated terminal for the **face-api** folder, and enter the following command to run the program:

C#

```
dotnet run
```

Python

```
python analyze-faces.py
```

3. When prompted, enter **5** and observe the result.
4. Edit the code in the **Main** function for menu option **5** to to experiment with different combinations of names and the images **person1.jpg** and **person2.jpg**.

More information

For more information about using the **Computer Vision** service for face detection, see the Computer Vision documentation.

To learn more about the **Face** service, see the Face documentation.