2.5 - Detect, analyze, and recognize faces

- Overview
- Learning objectives
- Introduction
- Identify options for face detection analysis and identification
  - The Azure AI Vision service
  - The Face service
- Understand considerations for face analysis
- Detect faces with the Azure AI Vision service
- Understand capabilities of the face service
- Compare and match detected faces
- Implement facial recognition
- Exercise - Detect, analyze, and identify faces
  - Prepare to use the Azure AI Vision SDK
  - View the image you will analyze
  - Detect faces in an image
  - Prepare to use the Face SDK
  - Detect and analyze faces
  - More information
- Knowledge Check
- Summary

Overview

The ability for applications to detect human faces, analyze facial features and emotions, and identify individuals is a key artificial intelligence capability.

Learning objectives

After completing this module, you'll be able to:

- Identify options for face detection, analysis, and identification.
- Understand considerations for face analysis.
- Detect faces with the Computer Vision service.
- Understand capabilities of the Face service.
- Compare and match detected faces.
- Implement facial recognition.

Introduction

Face detection, analysis, and recognition are all common computer vision challenges for AI systems. The ability to detect when a person is present, identify a person's facial location, or recognize an individual based on their facial features is a key way in which AI systems can exhibit human-like behavior and build empathy with users.

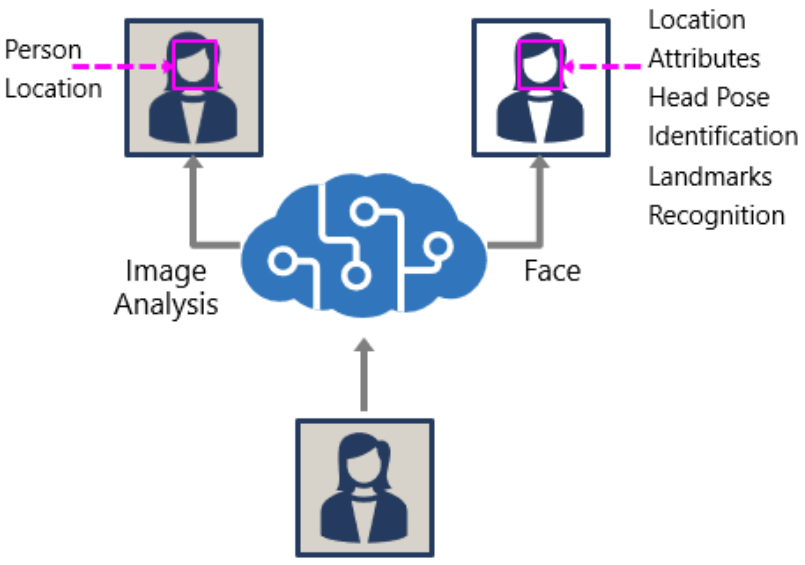
In this module, you'll learn how to detect, analyze, and recognize faces using Azure AI Services.

After completing this module, you'll be able to:

- Identify options for face detection analysis.
- Describe considerations for face analysis.
- Detect faces with the Azure AI Vision service.
- Describe the capabilities of the Face service.
- Learn about facial recognition.

Identify options for face detection analysis and identification

There are two Azure AI services that you can use to build solutions that detect faces or people in images.



The Azure AI Vision service

The **Azure AI Vision** service enables you to detect people in an image, as well as returning a bounding box for its location.

The Face service

The **Face** service offers **more comprehensive facial analysis** capabilities than the Azure AI Vision service, including:

- Face detection (with bounding box).
- Comprehensive facial feature analysis (including head pose, presence of spectacles, blur, facial landmarks, occlusion and others).
- Face comparison and verification.

- Facial recognition.

> Important: Usage of facial recognition, identification, comparison, and verification will require getting approved through a Limited Access policy. You can read more about the addition of this policy to our Responsible AI standard. Facial recognition will be touched on in the rest of this module, but will be unavailable without applying for Limited Access.

Understand considerations for face analysis

While all applications of artificial intelligence require considerations for responsible and ethical use, system that rely on facial data can be particularly problematic.

When building a solution that uses facial data, considerations include (but aren't limited to):

- **Data privacy and security**. Facial data is personally identifiable, and should be considered sensitive and private. You should ensure that you have implemented adequate protection for facial data used for model training and inferencing.
- **Transparency**. Ensure that users are informed about how their facial data is used, and who will have access to it.
- **Fairness and inclusiveness**. Ensure that your face-based system can't be used in a manner that is prejudicial to individuals based on their appearance, or to unfairly target individuals.

Detect faces with the Azure AI Vision service

To detect and analyze faces with the Azure AI Vision service, call the **Analyze Image** function (SDK or equivalent REST method), specifying **People** as one of the visual features to be returned.

> *People detection* - detecting the presence, location, and features of people in the image.

In images that contain one or more people, the response includes details of their location in the image and the attributes of the detected person, like this:


```
{
  "modelVersion": "2023-10-01",
  "metadata": {
    "width": 400,
    "height": 600
  },
  "peopleResult": {
    "values": [
      {
        "boundingBox": {
          "x": 0,
          "y": 56,
          "w": 101,
          "h": 189
        },
```

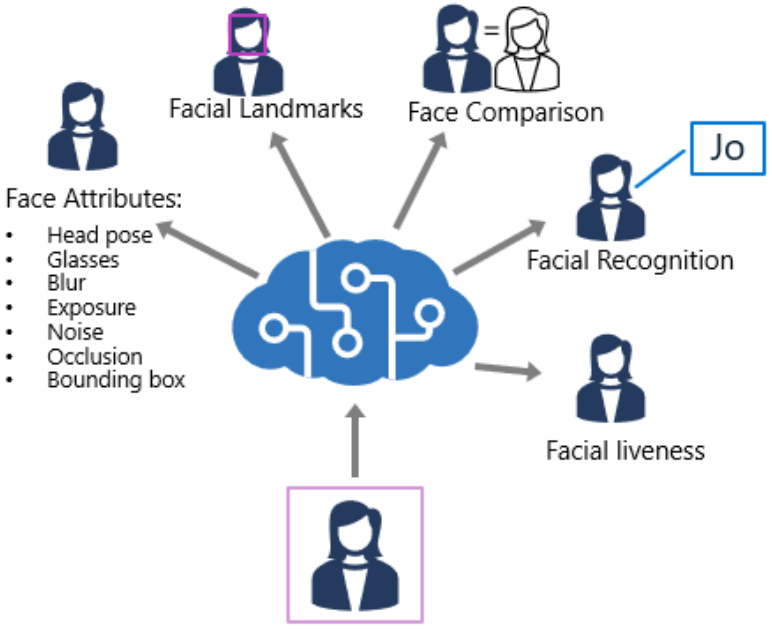
```
        "confidence": 0.9474349617958069
      },
      {
        "boundingBox": {
          "x": 402,
          "y": 96,
          "w": 124,
          "h": 156
        },
        "confidence": 0.9310565276194865
      },
    ...
    ]
  }
}
```


For more information on Azure AI Vision people detection, see the People detection concept page

> **Note**: Azure AI Vision previously included age and gender prediction, however that has been removed as a safeguard for responsible use. You can read more about our Responsible AI Investments here.

Understand capabilities of the face service

The **Face** service provides comprehensive facial detection, analysis, and recognition capabilities.



The Face service provides functionality that you can use for:

- *Face detection* - for each detected face, the results include an ID that identifies the face and the bounding box coordinates indicating its location in the image.
- *Face attribute analysis* - you can return a wide range of facial attributes, including:
  - Head pose (*pitch*, *roll*, and *yaw* orientation in 3D space)
  - Glasses (*NoGlasses*, *ReadingGlasses*, *Sunglasses*, or *Swimming Goggles*)

    - Blur (*low*, *medium*, or *high*)
    - Exposure (*underExposure*, *goodExposure*, or *overExposure*)
    - Noise (visual noise in the image)
    - Occlusion (objects obscuring the face)
    - Accessories (glasses, headwear, mask)
    - QualityForRecognition (*low*, *medium*, or *high*)
- *Facial landmark location* - coordinates for key landmarks in relation to facial features (for example, eye corners, pupils, tip of nose, and so on)
- *Face comparison* - you can compare faces across multiple images for similarity (to find individuals with similar facial features) and verification (to determine that a face in one image is the same person as a face in another image)
- *Facial recognition* - you can train a model with a collection of faces belonging to specific individuals, and use the model to identify those people in new images.
- *Facial liveness* - liveness can be used to determine if the input video is a real stream or a fake to prevent bad intentioned individuals from spoofing the recognition system.

You can provision **Face** as a single-service resource, or you can use the **Face API in a multi-service Azure AI Services** resource.

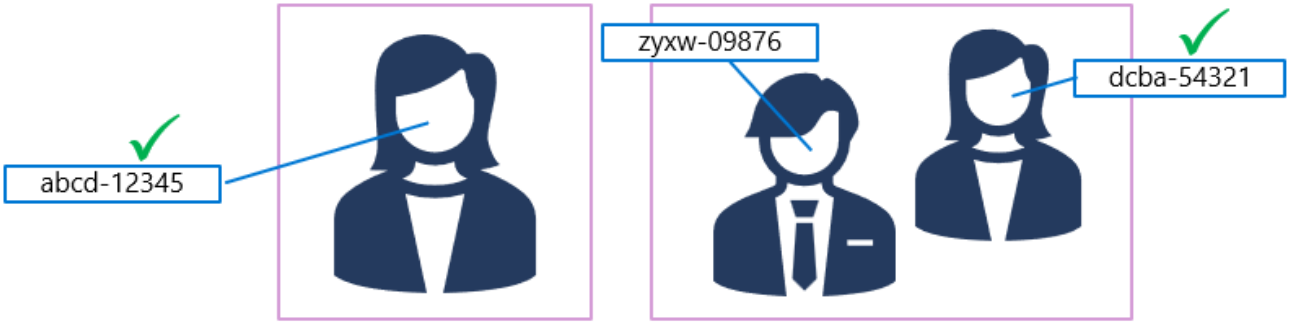
If you want to use the identification, recognition, and verification features of **Face**, you'll need to apply for the Limited Access policy and get approval before these features are available.

Compare and match detected faces

When a face is detected by the Face service, a unique ID is assigned to it and retained in the service resource for 24 hours. The ID is a GUID, with **no indication of the individual's identity other than their facial features.**

> Important: Usage of facial recognition, comparison, and verification will require getting approved through a Limited Access policy. You can read more about the addition of this policy to our Responsible AI standard. Facial recognition will be unavailable to new customers until they are granted the Limited Access policy.

While the detected face ID is cached, subsequent images can be used to compare the new faces to the cached identity and determine if they are *similar* (in other words, they share similar facial features) or to *verify* that the same person appears in two images.



This ability to compare faces anonymously can be useful in systems where it's important to confirm that the same person is present on two occasions, without the need to know the actual identity of the person. For example, by taking images of people as they enter and leave a secured space to verify that everyone who entered leaves.

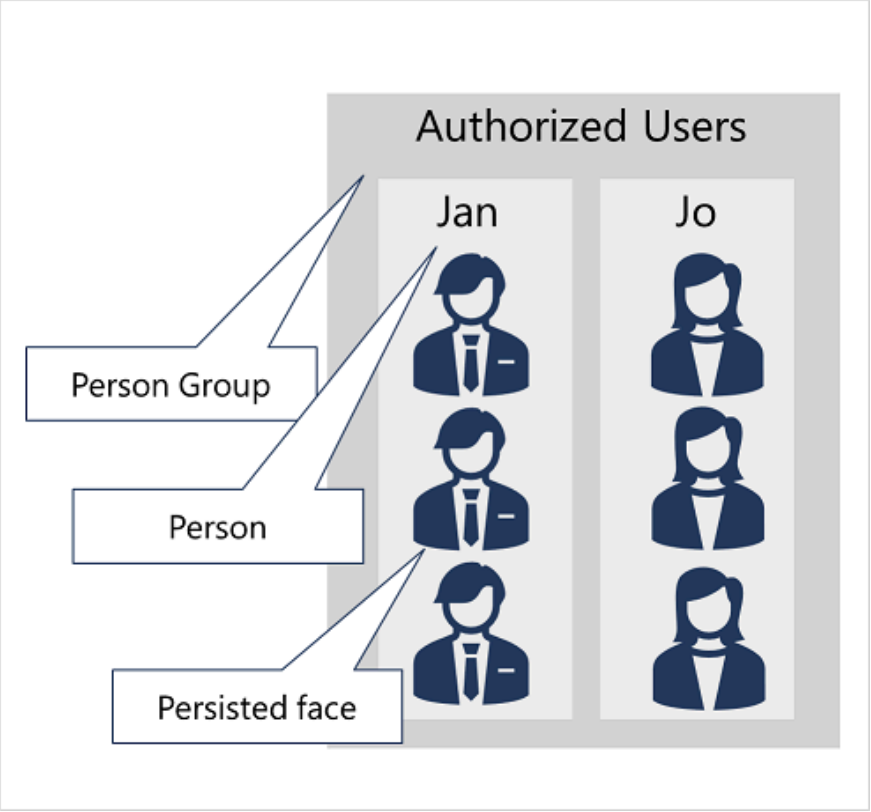
Implement facial recognition

For scenarios where you need to **positively identify individuals**, you can train a facial recognition model using face images.

> Note: As mentioned in the previous unit, recognition models will require getting approved through a Limited Access policy.

To train a facial recognition model with the Face service:

1. Create a **Person Group** that defines the set of individuals you want to identify (for example, *employees*).
2. Add a **Person** to the **Person Group** for each individual you want to identify.
3. Add detected faces from multiple images to each **person**, preferably in various poses. The IDs of these faces will no longer expire after 24 hours (so they're now referred to as *persisted* faces).
4. Train the model.



The trained model is stored in your Face (or Azure AI Services) resource, and can be used by client applications to:

- *Identify* individuals in images.
- *Verify* the identity of a detected face.
- Analyze new images to find faces that are *similar* to a known, persisted face.

Exercise - Detect, analyze, and identify faces

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

> **Important**: This lab can be completed without requesting any additional access to restricted features.

> **Note**: From June 21st 2022, capabilities of Azure AI services that return personally identifiable information are restricted to customers who have been granted limited access. Additionally, capabilities that infer emotional state are no longer available. For more details about the changes Microsoft has made, and why - see Responsible AI investments and safeguards for facial recognition.

Prepare to use the Azure AI Vision SDK

In this exercise, you'll complete a partially implemented client application that uses the Azure AI 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 **04-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 Azure AI Vision SDK package by running the appropriate command for your language preference:

   C#

   ```
   dotnet add package Azure.AI.Vision.ImageAnalysis -v 0.15.1-beta.1
   ```

   Python

   ```
   pip install azure-ai-vision==0.15.1b1
   ```

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 Azure AI 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-people.py (View here: detect-people.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 Azure AI Vision SDK:

   C#

   ```
   // import namespaces using Azure.AI.Vision.Common; using Azure.AI.Vision.ImageAnalysis;
   ```

Python

```
# import namespaces import azure.ai.vision as sdk
```

View the image you will analyze

In this exercise, you will use the **Azure AI 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 (Data and ML Engineering/Courses/Designing and Implementing a Microsoft Azure AI Solution (AI Engineer Exam)/mslearn-ai-vision/Labfiles/04-face/C-Sharp/computer-vision/images/people.jpg)

Detect faces in an image

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

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

   C#

   ```
   // Authenticate Azure AI Vision client var cvClient = new VisionServiceOptions( aiSvcEndpoint, new AzureKeyCredential(aiSvcKey));
   ```

   Python

   ```
   # Authenticate Azure AI Vision client cv_client = sdk.VisionServiceOptions(ai_endpoint, ai_key)
   ```

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 **AnalyzeImage**. This function is not yet fully implemented.
3. In the **AnalyzeImage** function, under the comment **Specify features to be retrieved (PEOPLE)**, add the following code:

   C#

   ```
   // Specify features to be retrieved (PEOPLE) Features = ImageAnalysisFeature.People
   ```

   Python

   ```
   # Specify features to be retrieved (PEOPLE) analysis_options = sdk.ImageAnalysisOptions() features = analysis_options.features = ( sdk.ImageAnalysisFeature.PEOPLE )
   ```

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

   C#

   ```
   // Get image analysis using var imageSource = VisionSource.FromFile(imageFile); using var analyzer = new ImageAnalyzer(serviceOptions, imageSource, analysisOptions); var result = analyzer.Analyze(); if (result.Reason == ImageAnalysisResultReason.Analyzed) { // Get people in the image if (result.People != null) { Console.WriteLine($" People:"); ``` // Prepare image for drawing System.Drawing.Image image = System.Drawing.Image.FromFile(imageFile); Graphics graphics = Graphics.FromImage(image); Pen pen = new Pen(Color.Cyan, 3); Font font = new Font("Arial", 16); SolidBrush brush = new SolidBrush(Color.WhiteSmoke); foreach (var person in result.People) { // Draw object
   ```

```
bounding box if confidence &gt; 50% if (person.Confidence &gt; 0.5) { // Draw object bounding box var r = person.BoundingBox; Rectangle rect = new Rectangle(r.X, r.Y, r.Width, r.Height); graphics.DrawRectangle(pen, rect); // Return the confidence of the person detected Console.WriteLine($" Bounding box {person.BoundingBox}, Confidence {person.Confidence:0.0000}"); } } // Save annotated image String output_file = "detected_people.jpg"; image.Save(output_file); Console.WriteLine(" Results saved in " + output_file + "\n"); } ``` } else { var errorDetails = ImageAnalysisErrorDetails.FromResult(result); Console.WriteLine(" Analysis failed."); Console.WriteLine($" Error reason : {errorDetails.Reason}"); Console.WriteLine($" Error code : {errorDetails.ErrorCode}"); Console.WriteLine($" Error message: {errorDetails.Message}\n"); }
```

Python

```
# Get image analysis image = sdk.VisionSource(image_file) image_analyzer = sdk.ImageAnalyzer(cv_client, image, analysis_options) result = image_analyzer.analyze() if result.reason == sdk.ImageAnalysisResultReason.ANALYZED: # Get people in the image if result.people is not None: print("\nPeople in image:") ``` # Prepare image for drawing image = Image.open(image_file) fig = plt.figure(figsize=(image.width/100, image.height/100)) plt.axis('off') draw = ImageDraw.Draw(image) color = 'cyan' for detected_people in result.people: # Draw object bounding box if confidence &gt; 50% if detected_people.confidence &gt; 0.5: # Draw object bounding box r = detected_people.bounding_box bounding_box = ((r.x, r.y), (r.x + r.w, r.y + r.h)) draw.rectangle(bounding_box, outline=color, width=3) # Return the confidence of the person detected print(" {} (confidence: {:.2f}%)".format(detected_people.bounding_box, detected_people.confidence * 100)) # Save annotated image plt.imshow(image) plt.tight_layout(pad=0) outputfile = 'detected_people.jpg' fig.savefig(outputfile) print(' Results saved in', outputfile) ``` else: error_details = sdk.ImageAnalysisErrorDetails.from_result(result) print(" Analysis failed.") print(" Error reason: {}".format(error_details.reason)) print(" Error code: {}".format(error_details.error_code)) print(" Error message: {}".format(error_details.message))
```

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

   C#

   ```
   dotnet run
   ```

   Python

   ```
   python detect-people.py
   ```

6. Observe the output, which should indicate the number of faces detected.
7. View the **detected_people.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 label the location of the top left of the box, and the bounding box coordinates to draw a rectangle around each face.

Prepare to use the Face SDK

While the **Azure AI 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 **04-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#

   ```
   dotnet add package Microsoft.Azure.CognitiveServices.Vision.Face --version 2.8.0-preview.3
   ```

   Python

   ```
   pip install azure-cognitiveservices-vision-face==0.6.0
   ```

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 Azure AI 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 (View here: 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 Vision SDK:
   C#
   `// Import namespaces using Microsoft.Azure.CognitiveServices.Vision.Face; using Microsoft.Azure.CognitiveServices.Vision.Face.Models;`

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

   Python
   `# 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 head pose, blur, 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#

   ```
   // Specify facial features to be retrieved IList<FaceAttributeType> features = new FaceAttributeType[] { FaceAttributeType.Occlusion, FaceAttributeType.Blur, FaceAttributeType.Glasses };
   ```

   Python

   ```
   # Specify facial features to be retrieved features = [FaceAttributeType.occlusion, FaceAttributeType.blur, FaceAttributeType.glasses]
   ```

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

   C#

   ```
   // Get faces using (var imageData = File.OpenRead(imageFile)) { var detected_faces = await faceClient.Face.DetectWithStreamAsync(imageData, returnFaceAttributes: features, returnFaceId: false); if (detected_faces.Count() > 0) { Console.WriteLine($" {detected_faces.Count()} faces detected."); // Prepare image for drawing Image image = Image.FromFile(imageFile); Graphics graphics = Graphics.FromImage(image); Pen pen = new Pen(Color.LightGreen, 3); Font font = new Font("Arial", 4); SolidBrush brush = new SolidBrush(Color.White); int faceCount=0; // Draw and annotate each face foreach (var face in detected_faces) { faceCount++; Console.WriteLine($"\nFace number {faceCount}"); // Get face properties Console.WriteLine($" - Mouth Occluded: {face.FaceAttributes.Occlusion.MouthOccluded}"); Console.WriteLine($" - Eye Occluded: {face.FaceAttributes.Occlusion.EyeOccluded}"); Console.WriteLine($" - Blur: {face.FaceAttributes.Blur.BlurLevel}"); Console.WriteLine($" - 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 number {faceCount}"; 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

   ```
   # 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, return_face_id=False) 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'
   ```

```
face_count = 0 # Draw and annotate each face for face in detected_faces: # Get face properties face_count += 1 print('\nFace number {}'.format(face_count)) detected_attributes = face.face_attributes.as_dict() if 'blur' in detected_attributes: print(' - Blur:') for blur_name in detected_attributes['blur']: print(' - {}: {}'.format(blur_name, detected_attributes['blur'][blur_name])) if 'occlusion' in detected_attributes: print(' - Occlusion:') for occlusion_name in detected_attributes['occlusion']: print(' - {}: {}'.format(occlusion_name, detected_attributes['occlusion'][occlusion_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 number {}'.format(face_count) 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)
```

4. Examine the code you added to the **DetectFaces** function. It analyzes an image file and detects any faces it contains, including attributes for occlusion, blur, 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.
5. 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
   ```

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

More information

There are several additional features available within the **Face** service, but following the Responsible AI Standard those are restricted behind a Limited Access policy. These features include identifying, verifying, and creating facial recognition models. To learn more and apply for access, see the Limited Access for Azure AI Services.

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

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

Knowledge Check

1. Which of the following facial attributes can the Azure AI Vision service predict? *

- (•) Location.
  - ✔ That's correct. The Azure AI Vision service predicts location for a detected face.
- ( ) Type of eye-glasses.
- ( ) Occlusion.

2. You need to create a facial recognition solution to identify named employees. Which service should you use? *

- ( ) Azure AI Vision.
- ( ) Azure AI Custom Vision.
- (•) Face.
  - ✔ That's correct. Use the Face service to create a facial recognition solution.

3. You need to verify that the person in a photo taken at hospital reception is the same person in a photo taken at a ward entrance 10 minutes later. What should you do? *

- ( ) Create a People Group and add a person for every hospital visitor with multiple photographs to train a model.
- (•) Verify the face in the ward photo by comparing it to the detected face ID from the reception photo.
  - ✔ That's correct. The most efficient approach is to compare the two faces using the detected face ID within 24 hours.
- ( ) Compare the Age, head pose, and hair color for the faces in the reception and ward photo's.

Summary

In this module, you have learned how to detect, analyze, and recognize faces.

Now that you've completed this module, you can:

- Identify options for face detection analysis.
- Describe considerations for face analysis.
- Detect people with the Azure AI Vision service.
- Describe the capabilities of the Face service.
- Understand facial recognition.

To find out more about face detection with the Azure AI Vision service, see the Azure AI Vision documentation. To find out more about the Face service, see the Face documentation.

✍️ Compiled by Kenneth Leung (2025)