2.1 - Analyze images

Overview

Computer vision is an area of artificial intelligence that deals with visual perception. **Azure AI Vision** includes multiple services that support common computer vision scenarios.

Introduction

Azure AI Vision is a branch of artificial intelligence (AI) in which software interprets visual input, often from images or video feeds.

In this module, you'll learn how to use the **Azure AI Vision** service to extract information from images.

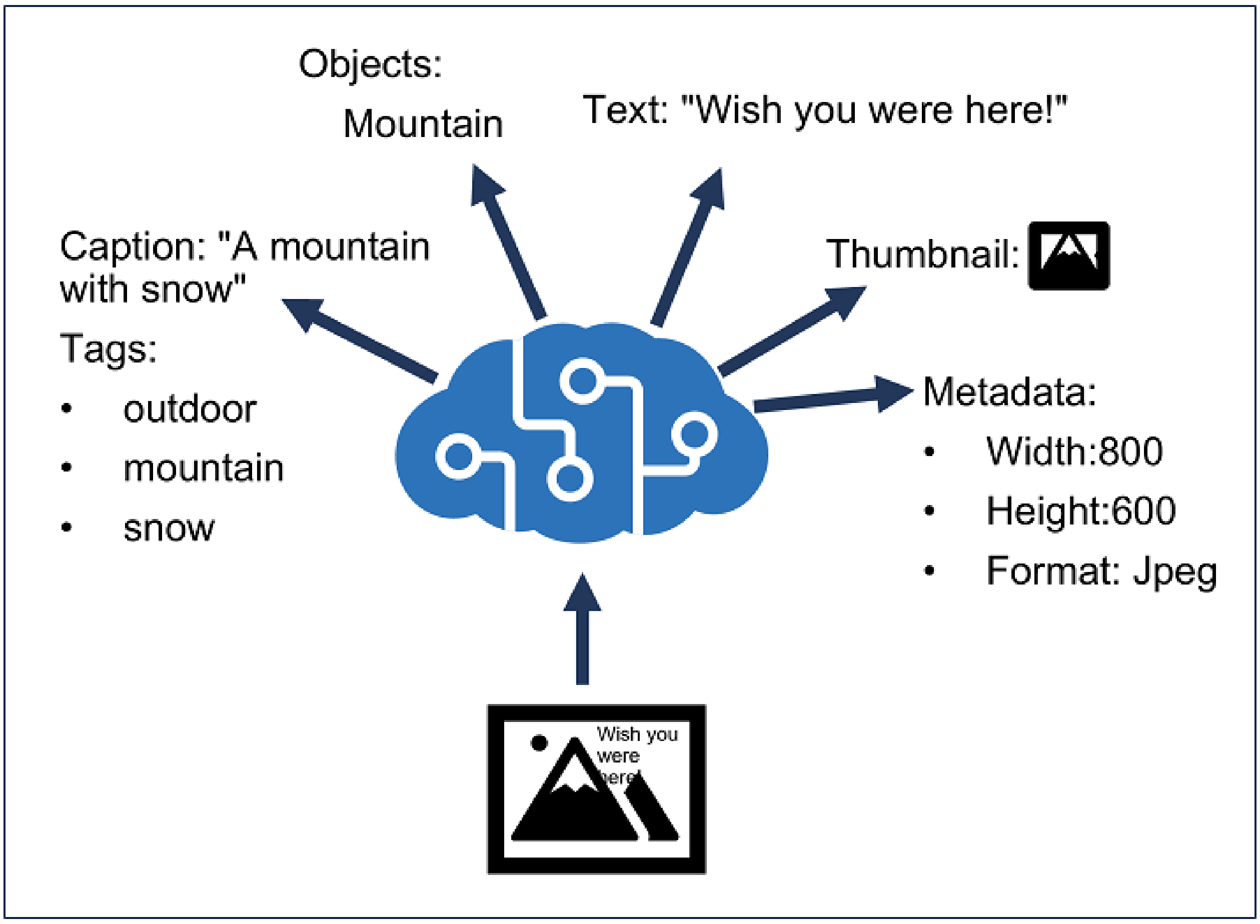
After completing this module, you'll be able to:

- Provision an Azure AI Vision resource.
- Analyze an image.
- Remove an image background.
- Generate a smart cropped thumbnail.

Provision an Azure AI Vision resource

The **Azure AI Vision** service is designed to help you extract information from images. It provides functionality that you can use for:

- *Description and tag generation* - determining an appropriate caption for an image, and identifying relevant "tags" that can be used as keywords to indicate its subject.
- *Object detection* - detecting the presence and location of specific objects within the image.
- *People detection* - detecting the presence, location, and features of people in the image.
- *Image metadata, color, and type analysis* - determining the format and size of an image, its dominant color palette, and whether it contains clip art.
- *Category identification* - identifying an appropriate categorization for the image, and if it contains any known landmarks.
- *Background removal* - detecting the background in an image and output the image with the background transparent or a greyscale alpha matte image.
- *Moderation rating* - determine if the image includes any adult or violent content.
- *Optical character recognition* - reading text in the image.
- *Smart thumbnail generation* - identifying the main region of interest in the image to create a smaller "thumbnail" version.



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

Note: In this module, we'll focus on the image analysis and thumbnail generation capabilities of the Azure AI Vision service. To learn how to use the Azure AI Vision service for optical character recognition, check out the [Read Text in images and documents with the Azure AI Vision service] module.

Analyze an image

To analyze an image, you can use the **Analyze Image** REST method or the equivalent method in the **SDK** for your preferred programming language, specifying the visual features you want to include in the analysis (and if you select categories, whether or not to include details of celebrities or landmarks). This method returns a JSON document containing the requested information.

> Note: Detection of celebrities will require getting approved through a Limited Access policy. You can read more about the addition of this policy to our Responsible AI standard. Celebrity recognition is seen in some screenshots, however is not included in the lab.

```
from azure.ai.vision.imageanalysis import ImageAnalysisClient
from azure.ai.vision.imageanalysis.models import VisualFeatures
from azure.core.credentials import AzureKeyCredential

client = ImageAnalysisClient(
    endpoint=os.environ["ENDPOINT"],
    credential=AzureKeyCredential(os.environ["KEY"])
)

result = client.analyze(
    image_url="<url>",
    visual_features=[VisualFeatures.CAPTION, VisualFeatures.READ],
    gender_neutral_caption=True,
    language="en",
)
```

> Note: For `AzureKeyCredential` , can use either `Key1` or `Key2` from the Azure AI Vision resource

Available visual features are contained in the `VisualFeatures` Enum:

- VisualFeatures.**TAGS**: Identifies tags about the image, including objects, scenery, setting, and actions
- VisualFeatures.**OBJECTS**: Returns the bounding box for each detected object
- VisualFeatures.**CAPTION**: Generates a caption of the image in natural language
- VisualFeatures.**DENSE_CAPTIONS**: Generates more detailed captions for the objects detected
- VisualFeatures.**PEOPLE**: Returns the bounding box for detected people
- VisualFeatures.**SMART_CROPS**: Returns the bounding box of the specified aspect ratio for the area of interest
- VisualFeatures.**READ**: Extracts readable text

Specifying the visual features you want analyzed in the image determines what information the response will contain. Most responses will contain a bounding box (if a location in the image is reasonable) or a confidence score (for features such as tags or captions).

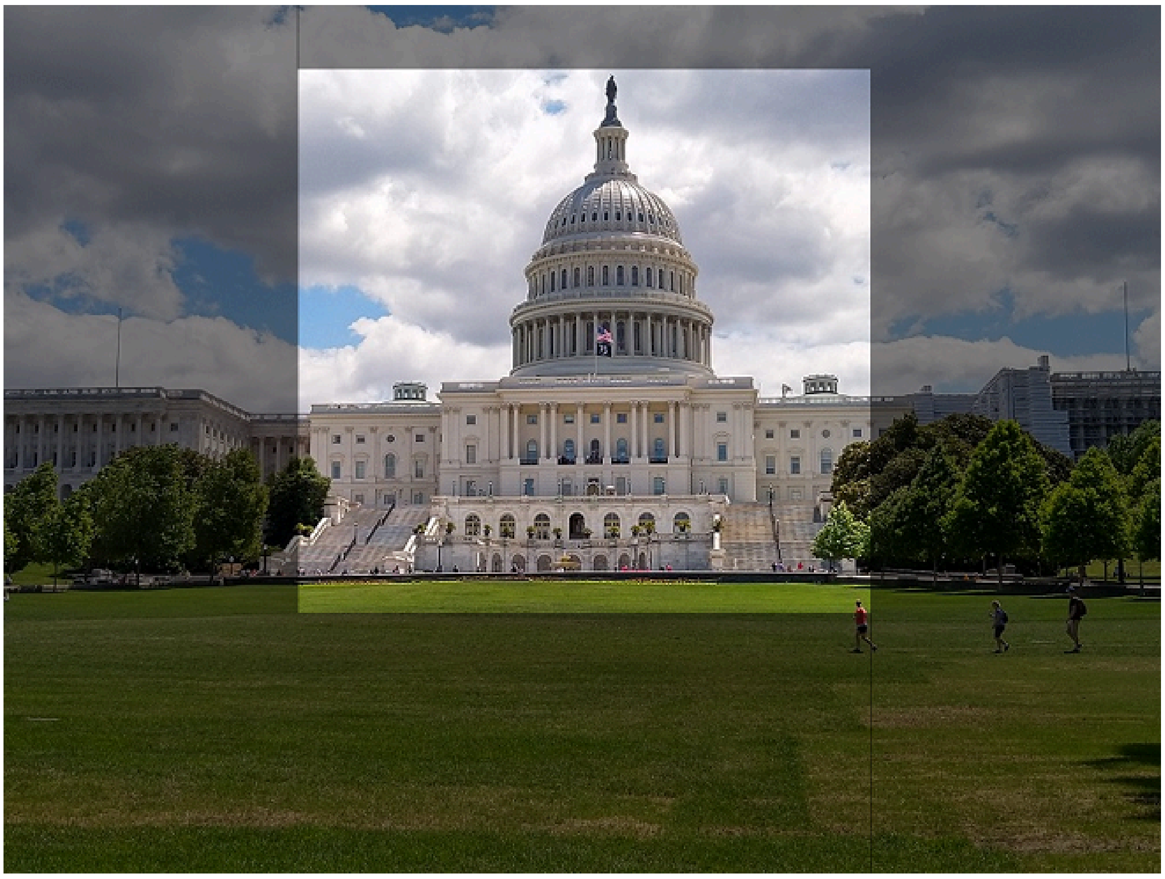
The JSON response for image analysis looks similar to this example, depending on your requested features:

```
{
  "apim-request-id": "abcde-1234-5678-9012-f1g2h3i4j5k6",
  "modelVersion": "<version>",
  "denseCaptionsResult": {
    "values": [
      {
        "text": "a house in the woods",
        "confidence": 0.7055229544639587,
        "boundingBox": {
          "x": 0,
          "y": 0,
          "w": 640,
          "h": 640
        }
      },
      {
        "text": "a trailer with a door and windows",
        "confidence": 0.6675070524215698,
        "boundingBox": {
          "x": 214,
          "y": 434,
          "w": 154,
          "h": 108
        }
      }
    ]
  },
  "metadata": {
    "width": 640,
    "height": 640
  }
}
```

Generate a smart-cropped thumbnail and remove background

Thumbnails are often used to provide smaller versions of images in applications and websites. For example, a tourism site might display a list of tourist attractions in a city with a small, representative thumbnail image for each attraction; and only display the full image when the user selects the "details" page for an individual attraction.

The Azure AI Vision service enables you to create a thumbnail with different dimensions (and aspect ratio) from the source image, and **optionally to use image analysis to determine the *region of interest* in the image (its main subject) and make that the focus of the thumbnail.** This ability to determine the region of interest is especially useful when cropping the image to change its aspect ratio.



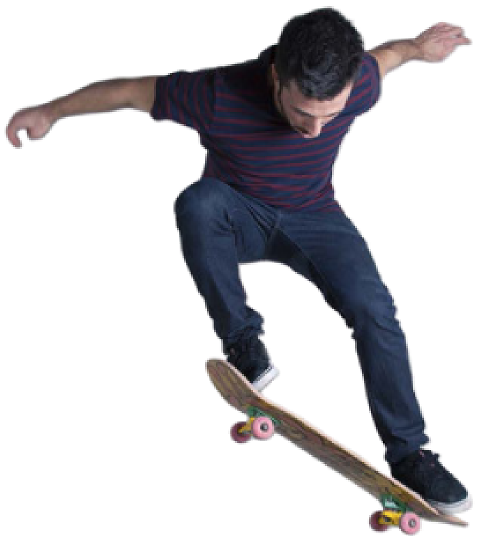
You can specify the aspect ratio of the cropped image (width / height), ranging from `0.75` to `1.80`.

Remove image background

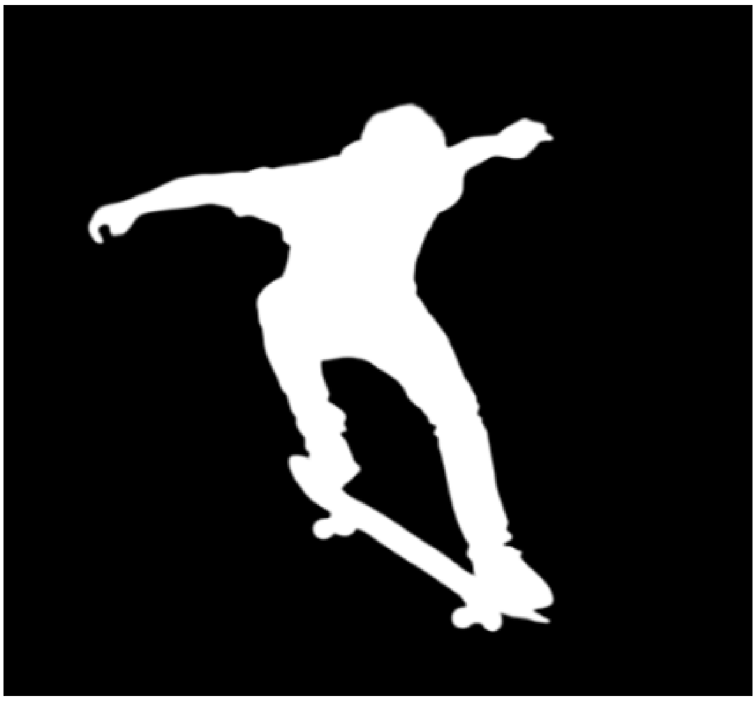
The background removal feature can split the image into the subject in the foreground, and everything else that is considered background. Azure AI Vision achieves this feature by creating an *alpha matte* of the foreground subject, which is then used to return either the foreground or the background.

For example, take this image original of a skateboarder.

With the background removed, we get just the skateboarder on a transparent background.



When creating an **alpha matte** of an image, the result is the foreground in all white, with a black background.



Alpha matte images are helpful when client applications intend to do further processing of an image that requires **separation of foreground and background objects**.

Exercise - Analyze images with Azure AI Vision

Azure AI Vision is an artificial intelligence capability that enables software systems to interpret visual input by analyzing images. In Microsoft Azure, the **Vision** Azure AI service provides **pre-built models for common computer vision tasks**, including analysis of images to suggest captions and tags, detection of common objects and people. You can also use the Azure AI Vision service to remove the background or create a foreground matting of images.

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 images.

> **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 **Labfiles/01-analyze-images** folder and expand the **C-Sharp** or **Python** folder depending on your language preference.
2. Right-click the **image-analysis** 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 1.0.0-beta.1
   ```

   > **Note**: If you are prompted to install dev kit extensions, you can safely close the message.

   Python

```
pip install azure-ai-vision-imageanalysis==1.0.0b1
```

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

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

   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.
4. Note that the **image-analysis** folder contains a code file for the client application:

   - **C#**: Program.cs
   - **Python**: image-analysis.py (View here: image-analysis.py)

   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.ImageAnalysis;`

   Python
   `# import namespaces from azure.ai.vision.imageanalysis import ImageAnalysisClient from azure.ai.vision.imageanalysis.models import VisualFeatures from azure.core.credentials import AzureKeyCredential`

View the images you will analyze

In this exercise, you will use the Azure AI Vision service to analyze multiple images.

1. In Visual Studio Code, expand the **image-analysis** folder and the **images** folder it contains.
2. Select each of the image files in turn to view them in Visual Studio Code.

Analyze an image to suggest a caption

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

1. In the code file for your client application (**Program.cs** or **image-analysis.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 ImageAnalysisClient client = new ImageAnalysisClient( new Uri(aiSvcEndpoint), new AzureKeyCredential(aiSvcKey));`

   Python
   `# Authenticate Azure AI Vision client cv_client = ImageAnalysisClient( endpoint=ai_endpoint, credential=AzureKeyCredential(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 two other functions

(**AnalyzeImage** and **BackgroundForeground**). These functions are not yet fully implemented.

3. In the **AnalyzeImage** function, under the comment **Get result with specify features to be retrieved**, add the following code:

   C#

   ```
   // Get result with specified features to be retrieved ImageAnalysisResult result =
   client.Analyze( BinaryData.FromStream(stream), VisualFeatures.Caption |
   VisualFeatures.DenseCaptions | VisualFeatures.Objects | VisualFeatures.Tags |
   VisualFeatures.People);
   ```

   Python

   ```
   # Get result with specified features to be retrieved result = cv_client.analyze(
   image_data=image_data, visual_features=[ VisualFeatures.CAPTION,
   VisualFeatures.DENSE_CAPTIONS, VisualFeatures.TAGS, VisualFeatures.OBJECTS,
   VisualFeatures.PEOPLE], )
   ```

4. In the **AnalyzeImage** function, under the comment **Display analysis results**, add the following code (including the comments indicating where you will add more code later.):

   C#

   ```
   // Display analysis results // Get image captions if (result.Caption.Text != null) {
   Console.WriteLine(" Caption:"); Console.WriteLine($" \"{result.Caption.Text}\",
   Confidence {result.Caption.Confidence:0.00}\n"); } // Get image dense captions
   Console.WriteLine(" Dense Captions:"); foreach (DenseCaption denseCaption in
   result.DenseCaptions.Values) { Console.WriteLine($" Caption: '{denseCaption.Text}',
   Confidence: {denseCaption.Confidence:0.00}"); } // Get image tags // Get objects in the
   image // Get people in the image
   ```

   Python

   ```
   # Display analysis results # Get image captions if result.caption is not None:
   print("\nCaption:") print(" Caption: '{}' (confidence:
   {:.2f}%)".format(result.caption.text, result.caption.confidence * 100)) # Get image
   dense captions if result.dense_captions is not None: print("\nDense Captions:") for
   caption in result.dense_captions.list: print(" Caption: '{}' (confidence:
   {:.2f}%)".format(caption.text, caption.confidence * 100)) # Get image tags # Get objects
   in the image # Get people in the image
   ```

5. Save your changes and return to the integrated terminal for the **image-analysis** folder, and enter the following command to run the program with the argument **images/street.jpg**:

   C#

   ```
   dotnet run images/street.jpg
   ```

   Python

   ```
   python image-analysis.py images/street.jpg
   ```

6. Observe the output, which should include a suggested caption for the **street.jpg** image.
7. Run the program again, this time with the argument **images/building.jpg** to see the caption that gets generated for the **building.jpg** image.
8. Repeat the previous step to generate a caption for the **images/person.jpg** file.

Get suggested tags for an image

It can sometimes be useful to identify relevant *tags* that provide clues about the contents of an image.

1. In the **AnalyzeImage** function, under the comment **Get image tags**, add the following code:

   C#

   ```
   // Get image tags if (result.Tags.Values.Count > 0) { Console.WriteLine($"\n Tags:");
   foreach (DetectedTag tag in result.Tags.Values) { Console.WriteLine($" '{tag.Name}',
   Confidence: {tag.Confidence:F2}"); } }
   ```

   Python

   ```
   # Get image tags if result.tags is not None: print("\nTags:") for tag in
   result.tags.list: print(" Tag: '{}' (confidence: {:.2f}%)".format(tag.name,
   tag.confidence * 100))
   ```

2. Save your changes and run the program once for each of the image files in the **images** folder, observing that in addition to the image caption, a list of suggested tags is displayed.

Detect and locate objects in an image

Object detection is a specific form of computer vision in which individual objects within an image are identified and their location indicated by a bounding box..

1. In the **AnalyzeImage** function, under the comment **Get objects in the image**, add the following code:

   C#

   ```
   // Get objects in the image if (result.Objects.Values.Count > 0) { Console.WriteLine("
   Objects:"); // Prepare image for drawing stream.Close(); 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 (DetectedObject detectedObject in
   result.Objects.Values) { Console.WriteLine($" \"{detectedObject.Tags[0].Name}\""); //
   Draw object bounding box var r = detectedObject.BoundingBox; Rectangle rect = new
   Rectangle(r.X, r.Y, r.Width, r.Height); graphics.DrawRectangle(pen, rect);
   graphics.DrawString(detectedObject.Tags[0].Name,font,brush,(float)r.X, (float)r.Y); } //
   Save annotated image String output_file = "objects.jpg"; image.Save(output_file);
   Console.WriteLine(" Results saved in " + output_file + "\n"); }
   ```

   Python

   ```
   # Get objects in the image if result.objects is not None: print("\nObjects in image:") #
   Prepare image for drawing image = Image.open(image_filename) fig = plt.figure(figsize=
   (image.width/100, image.height/100)) plt.axis('off') draw = ImageDraw.Draw(image) color
   = 'cyan' for detected_object in result.objects.list: # Print object name print(" {}
   (confidence: {:.2f}%)".format(detected_object.tags[0].name,
   detected_object.tags[0].confidence * 100)) # Draw object bounding box r =
   detected_object.bounding_box bounding_box = ((r.x, r.y), (r.x + r.width, r.y +
   r.height)) draw.rectangle(bounding_box, outline=color, width=3)
   plt.annotate(detected_object.tags[0].name,(r.x, r.y), backgroundcolor=color) # Save
   annotated image plt.imshow(image) plt.tight_layout(pad=0) outputfile = 'objects.jpg'
   fig.savefig(outputfile) print(' Results saved in', outputfile)
   ```

2. Save your changes and run the program once for each of the image files in the **images** folder, observing any objects that are detected. After each run, view the **objects.jpg** file that is generated in the same folder as your code file to see the annotated objects.

Detect and locate people in an image

People detection is a specific form of computer vision in which individual people within an image are identified and their location indicated by a bounding box.

1. In the **AnalyzeImage** function, under the comment **Get people in the image**, add the following code:

   C#
   ```
   // Get people in the image if (result.People.Values.Count > 0) { Console.WriteLine($"
   People:"); // Prepare image for drawing System.Drawing.Image image =
   System.Drawing.Image.FromFile(imageFile); Graphics graphics = Graphics.FromImage(image);
   Pen pen = new Pen(Color.Cyan, 3); Font font = new Font("Arial", 16); SolidBrush brush =
   new SolidBrush(Color.WhiteSmoke); foreach (DetectedPerson person in
   result.People.Values) { // Draw object bounding box var r = person.BoundingBox;
   Rectangle rect = new Rectangle(r.X, r.Y, r.Width, r.Height); graphics.DrawRectangle(pen,
   rect); // Return the confidence of the person detected //Console.WriteLine($" Bounding
   box {person.BoundingBox.ToString()}, Confidence: {person.Confidence:F2}"); } // Save
   annotated image String output_file = "persons.jpg"; image.Save(output_file);
   Console.WriteLine(" Results saved in " + output_file + "\n"); }
   ```

   Python
   `# Get people in the image if result.people is not None: print("\nPeople in image:") # Prepare image for drawing image = Image.open(image_filename) 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.list: # Draw object bounding box r = detected_people.bounding_box bounding_box = ((r.x, r.y), (r.x + r.width, r.y + r.height)) 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 = 'people.jpg' fig.savefig(outputfile) print(' Results saved in', outputfile)```
2. (Optional) Uncomment the **Console.Writeline** command under the **Return the confidence of the person detected** section to review the confidence level returned that a person was detected at a particular position of the image.
3. Save your changes and run the program once for each of the image files in the **images** folder, observing any objects that are detected. After each run, view the **objects.jpg** file that is generated in the same folder as your code file to see the annotated objects.

> **Note**: In the preceding tasks, you used a single method to analyze the image, and then incrementally added code to parse and display the results. The SDK also provides individual methods for suggesting captions, identifying tags, detecting objects, and so on - meaning that you can use the most appropriate method to return only the information you need, reducing the size of the data payload that needs to be returned. See the .NET SDK documentation or Python SDK documentation for more details.

Remove the background or generate a foreground matte of an image

In some cases, you may need to create remove the background of an image or might want to create a foreground matte of that image. Let's start with the background removal.

1. In your code file, find the **BackgroundForeground** function; and under the comment **Remove the background from the image or generate a foreground matte**, add the following code:

C#

```
// Remove the background from the image or generate a foreground matte Console.WriteLine($"
Background removal:"); // Define the API version and mode string apiVersion = "2023-02-01-
preview"; string mode = "backgroundRemoval"; // Can be "foregroundMatting" or
"backgroundRemoval" string url = $"computervision/imageanalysis:segment?api-version=
{apiVersion}&mode={mode}"; // Make the REST call using (var client = new HttpClient()) {
var contentType = new MediaTypeWithQualityHeaderValue("application/json");
client.BaseAddress = new Uri(endpoint);
client.DefaultRequestHeaders.Accept.Add(contentType);
client.DefaultRequestHeaders.Add("Ocp-Apim-Subscription-Key", key); var data = new { url =
$"https://github.com/MicrosoftLearning/mslearn-ai-vision/blob/main/Labfiles/01-analyze-
images/Python/image-analysis/{imageFile}?raw=true" }; var jsonData =
JsonSerializer.Serialize(data); var contentData = new StringContent(jsonData,
Encoding.UTF8, contentType); var response = await client.PostAsync(url, contentData); if
(response.IsSuccessStatusCode) { File.WriteAllBytes("background.png",
response.Content.ReadAsByteArrayAsync().Result); Console.WriteLine(" Results saved in
background.png\n"); } else { Console.WriteLine($"API error: {response.ReasonPhrase} - Check
your body url, key, and endpoint."); } }
```

Python

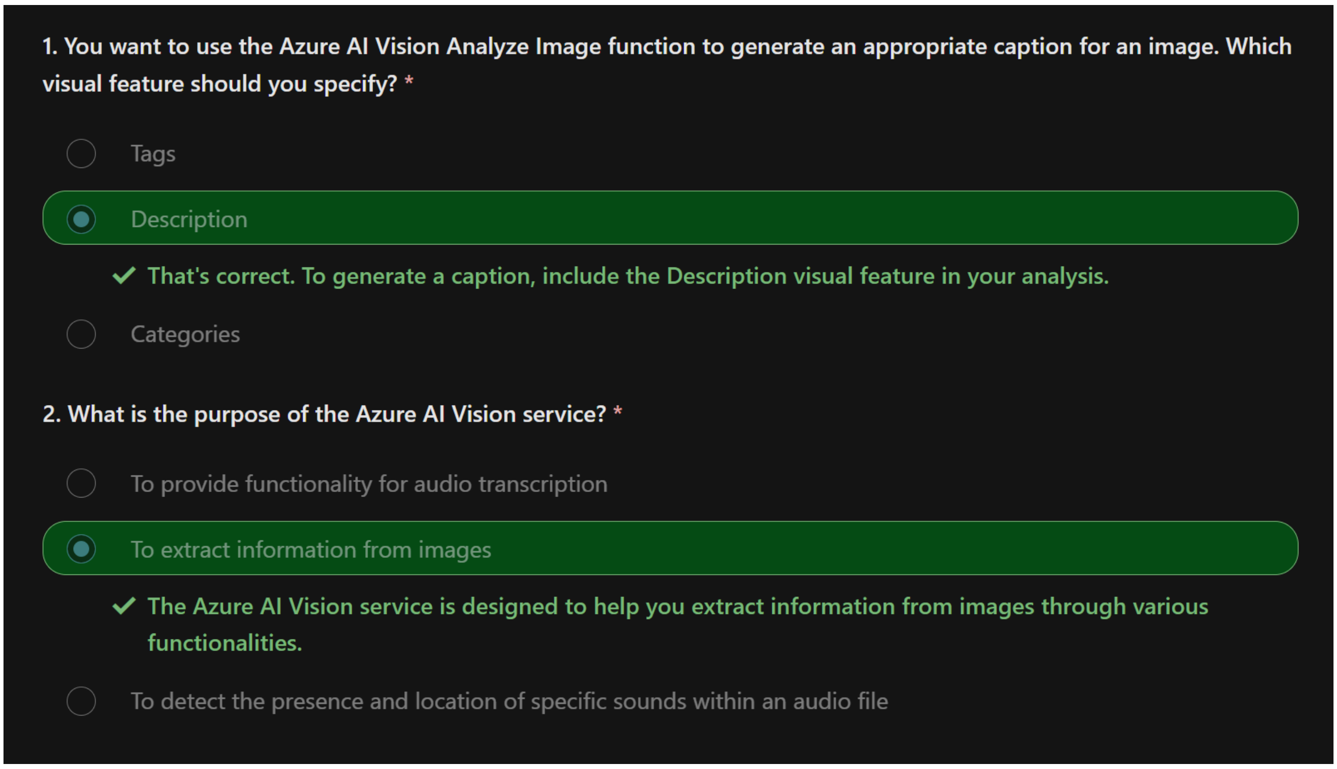
```
# Remove the background from the image or generate a foreground matte print('\nRemoving
background from image...') url = "{}computervision/imageanalysis:segment?api-version=
{}&mode={}".format(endpoint, api_version, mode) headers= { "Ocp-Apim-Subscription-Key":
key, "Content-Type": "application/json" }
image_url="https://github.com/MicrosoftLearning/mslearn-ai-vision/blob/main/Labfiles/01-
analyze-images/Python/image-analysis/{}?raw=true".format(image_file) body = { "url":
image_url, } response = requests.post(url, headers=headers, json=body)
image=response.content with open("backgroundForeground.png", "wb") as file:
file.write(image) print(' Results saved in backgroundForeground.png \n')
```

2. Save your changes and run the program once for each of the image files in the **images** folder, opening the **background.png** file that is generated in the same folder as your code file for each image. Notice how the background has been removed from each of the images.

Let's now generate a foreground matte for our images.

3. In your code file, find the **BackgroundForeground** function; and under the comment **Define the API version and mode**, change the mode variable to be `foregroundMatting`.
4. Save your changes and run the program once for each of the image files in the **images** folder, opening the **background.png** file that is generated in the same folder as your code file for each image. Notice how a foreground matte has been generated for your images.

Knowledge Check


1. You want to use the Azure AI Vision Analyze Image function to generate an appropriate caption for an image. Which visual feature should you specify? *

- Tags
- [✓ Description]
  ✓ That's correct. To generate a caption, include the Description visual feature in your analysis.
- Categories

2. What is the purpose of the Azure AI Vision service? *

- To provide functionality for audio transcription
- [✓ To extract information from images]
  ✓ The Azure AI Vision service is designed to help you extract information from images through various functionalities.
- To detect the presence and location of specific sounds within an audio file

---

Summary

In this module, you learned how to use the Azure AI Vision service to extract information from images.

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

- Provision an Azure AI Vision resource.
- Analyze an image.
- Generate a smart cropped thumbnail.

To learn more about the Azure AI Vision service, see the Azure AI Vision documentation.

✍️ Compiled by Kenneth Leung (2025)