Analyze Images with Computer Vision

Computer vision is an artificial intelligence capability that enables software systems to interpret visual input by analyzing images. In Microsoft Azure, the **Computer Vision** cognitive service provides pre-built models for common computer vision tasks, including analysis of images to suggest captions and tags, detection of common objects, landmarks, celebrities, brands, and the presence of adult content. You can also use the Computer Vision service to analyze image color and formats, and to generate "smart-cropped" thumbnail images.

Clone the repository for this course

If you have not already cloned **AI-102-AIEngineer** code repository to the environment where you're working on this lab, follow these steps to do so. Otherwise, open the cloned folder in Visual Studio Code.

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 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 **15-computer-vision** 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 Computer Vision SDK package by running the appropriate command for your language preference:

C#

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

Python

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

1. 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 cognitive services resource. Save your changes.

2. Note that the **image-analysis** folder contains a code file for the client application:
    - **C#**: Program.cs
    - **Python**: 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 Computer Vision SDK:

C#

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

Python

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

View the images you will analyze

In this exercise, you will use the Computer 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 then in Visual Studio Code.

Analyze an image to suggest a caption

Now you're ready to use the SDK to call the Computer 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**

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)
```

1. 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 **GetThumbnail**). These functions are not yet fully implemented.

2. In the **AnalyzeImage** function, under the comment **Specify features to be retrieved**, add the following code:

C#

Code Copy

```
// Specify features to be retrieved
List<VisualFeatureTypes?> features = new List<VisualFeatureTypes?>()
{
    VisualFeatureTypes.Description,
    VisualFeatureTypes.Tags,
    VisualFeatureTypes.Categories,
    VisualFeatureTypes.Brands,
    VisualFeatureTypes.Objects,
    VisualFeatureTypes.Adult
};
```

Python

Code Copy

```
# Specify features to be retrieved
features = [VisualFeatureTypes.description,
            VisualFeatureTypes.tags,
            VisualFeatureTypes.categories,
            VisualFeatureTypes.brands,
            VisualFeatureTypes.objects,
            VisualFeatureTypes.adult]
```

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

C#

Code Copy

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

    // get image captions
    foreach (var caption in analysis.Description.Captions)
    {
        Console.WriteLine($"Description: {caption.Text} (confidence:
{caption.Confidence.ToString("P")})");
    }

    // Get image tags


    // Get image categories


    // Get brands in the image


    // Get objects in the image


    // Get moderation ratings


}
```

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 image description
for caption in analysis.description.captions:
    print("Description: '{}' (confidence: {:.2f}%)".format(caption.text, caption.confidence *
100))

# Get image tags


# Get image categories


# Get brands in the image


# Get objects in the image


# Get moderation ratings
```

1. 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#

Code Copy

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

Python

Code Copy

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

1. Observe the output, which should include a suggested caption for the **street.jpg** image.
2. Run the program again, this time with the argument **images/building.jpg** to see the caption that gets generated for the **building.jpg** image.
3. 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#

Code Copy

```
// Get image tags
if (analysis.Tags.Count > 0)
{
    Console.WriteLine("Tags:");
    foreach (var tag in analysis.Tags)
    {
        Console.WriteLine($" -{tag.Name} (confidence: {tag.Confidence.ToString("P")})");
    }
}
```

Python

Code Copy

```
# Get image tags
if (len(analysis.tags) > 0):
    print("Tags: ")
    for tag in analysis.tags:
        print(" -'{}' (confidence: {:.2f}%)".format(tag.name, tag.confidence * 100))
```

1. 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.

Get image categories

The Computer Vision service can suggest *categories* for images, and within each category it can identify well-known landmarks or celebrities.

1. In the **AnalyzeImage** function, under the comment **Get image categories (including celebrities and landmarks)**, add the following code:

C#

Code Copy

```
// Get image categories (including celebrities and landmarks)
List<LandmarksModel> landmarks = new List<LandmarksModel> {};
List<CelebritiesModel> celebrities = new List<CelebritiesModel> {};
Console.WriteLine("Categories:");
foreach (var category in analysis.Categories)
{
    // Print the category
    Console.WriteLine($" -{category.Name} (confidence: {category.Score.ToString("P")})");

    // Get landmarks in this category
    if (category.Detail?.Landmarks != null)
    {
        foreach (LandmarksModel landmark in category.Detail.Landmarks)
        {
            if (!landmarks.Any(item => item.Name == landmark.Name))
            {
                landmarks.Add(landmark);
            }
        }
    }

    // Get celebrities in this category
    if (category.Detail?.Celebrities != null)
    {
        foreach (CelebritiesModel celebrity in category.Detail.Celebrities)
        {
            if (!celebrities.Any(item => item.Name == celebrity.Name))
            {
                celebrities.Add(celebrity);
            }
        }
    }
}

// If there were landmarks, list them
if (landmarks.Count > 0)
{
    Console.WriteLine("Landmarks:");
    foreach(LandmarksModel landmark in landmarks)
    {
        Console.WriteLine($" -{landmark.Name} (confidence:
{landmark.Confidence.ToString("P")})");
    }
}

// If there were celebrities, list them
if (celebrities.Count > 0)
{
    Console.WriteLine("Celebrities:");
    foreach(CelebritiesModel celebrity in celebrities)
    {
        Console.WriteLine($" -{celebrity.Name} (confidence:
{celebrity.Confidence.ToString("P")})");
    }
}
```

Python

Code Copy

```
# Get image categories (including celebrities and landmarks)
if (len(analysis.categories) > 0):
    print("Categories:")
    landmarks = []
    celebrities = []
    for category in analysis.categories:
        # Print the category
        print(" -'{}' (confidence: {:.2f}%)".format(category.name, category.score * 100))
        if category.detail:
            # Get landmarks in this category
            if category.detail.landmarks:
                for landmark in category.detail.landmarks:
                    if landmark not in landmarks:
                        landmarks.append(landmark)

            # Get celebrities in this category
            if category.detail.celebrities:
                for celebrity in category.detail.celebrities:
                    if celebrity not in celebrities:
                        celebrities.append(celebrity)

    # If there were landmarks, list them
    if len(landmarks) > 0:
        print("Landmarks:")
        for landmark in landmarks:
            print(" -'{}' (confidence: {:.2f}%)".format(landmark.name, landmark.confidence *
100))

    # If there were celebrities, list them
    if len(celebrities) > 0:
        print("Celebrities:")
        for celebrity in celebrities:
            print(" -'{}' (confidence: {:.2f}%)".format(celebrity.name, celebrity.confidence *
100))
```

1. 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 and tags, a list of suggested categories is displayed along with any recognized landmarks or celebrities (in particular in the **building.jpg** and **person.jpg** images).

Get brands in an image

Some brands are visually recognizable from logo's, even when the name of the brand is not displayed. The Computer Vision service is trained to identify thousands of well-known brands.

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

C#

Code Copy

```
// Get brands in the image
if (analysis.Brands.Count > 0)
{
    Console.WriteLine("Brands:");
    foreach (var brand in analysis.Brands)
    {
        Console.WriteLine($" -{brand.Name} (confidence: {brand.Confidence.ToString("P")})");
    }
}
```

Python

Code Copy

```
# Get brands in the image
if (len(analysis.brands) > 0):
    print("Brands: ")
    for brand in analysis.brands:
        print(" -'{}' (confidence: {:.2f}%)".format(brand.name, brand.confidence * 100))
```

1. Save your changes and run the program once for each of the image files in the **images** folder, observing any brands that are identified (specifically, in the **person.jpg** image).

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#

Code Copy

```
// Get objects in the image
if (analysis.Objects.Count > 0)
{
    Console.WriteLine("Objects in image:");

    // Prepare image for drawing
    Image image = 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.Black);

    foreach (var detectedObject in analysis.Objects)
    {
        // Print object name
        Console.WriteLine($" -{detectedObject.ObjectProperty} (confidence:
{detectedObject.Confidence.ToString("P")})");

        // Draw object bounding box
        var r = detectedObject.Rectangle;
        Rectangle rect = new Rectangle(r.X, r.Y, r.W, r.H);
        graphics.DrawRectangle(pen, rect);
        graphics.DrawString(detectedObject.ObjectProperty,font,brush,r.X, r.Y);

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

Python

Code Copy

```
# Get objects in the image
if len(analysis.objects) > 0:
    print("Objects in image:")

    # Prepare image for drawing
    fig = plt.figure(figsize=(8, 8))
    plt.axis('off')
    image = Image.open(image_file)
    draw = ImageDraw.Draw(image)
    color = 'cyan'
    for detected_object in analysis.objects:
        # Print object name
        print(" -{} (confidence: {:.2f}%)".format(detected_object.object_property,
detected_object.confidence * 100))

        # Draw object bounding box
        r = detected_object.rectangle
        bounding_box = ((r.x, r.y), (r.x + r.w, r.y + r.h))
        draw.rectangle(bounding_box, outline=color, width=3)
        plt.annotate(detected_object.object_property,(r.x, r.y), backgroundcolor=color)
    # Save annotated image
    plt.imshow(image)
    outputfile = 'objects.jpg'
    fig.savefig(outputfile)
    print('  Results saved in', outputfile)
```

1. 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.

Get moderation ratings for an image

Some images may not be suitable for all audiences, and you may need to apply some moderation to identify images that are adult or violent in nature.

1. In the **AnalyzeImage** function, under the comment **Get moderation ratings**, add the following code:

C#

Code Copy

```
// Get moderation ratings
string ratings = $"Ratings:\n -Adult: {analysis.Adult.IsAdultContent}\n -Racy:
{analysis.Adult.IsRacyContent}\n -Gore: {analysis.Adult.IsGoryContent}";
Console.WriteLine(ratings);
```

Python

Code Copy

```
# Get moderation ratings
ratings = 'Ratings:\n -Adult: {}\n -Racy: {}\n -Gore: {}'.format(analysis.adult.is_adult_content,

analysis.adult.is_racy_content,

analysis.adult.is_gory_content)
print(ratings)
```

1. Save your changes and run the program once for each of the image files in the **images** folder, observing the ratings for each image.

> **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.

Generate a thumbnail image

In some cases, you may need to create a smaller version of an image named a *thumbnail*, cropping it to include the main visual subject within new image dimensions.

1. In your code file, find the **GetThumbnail** function; and under the comment **Generate a thumbnail**, add the following code:

C#

Code Copy

```
// Generate a thumbnail
using (var imageData = File.OpenRead(imageFile))
{
    // Get thumbnail data
    var thumbnailStream = await cvClient.GenerateThumbnailInStreamAsync(100, 100,imageData,
true);

    // Save thumbnail image
    string thumbnailFileName = "thumbnail.png";
    using (Stream thumbnailFile = File.Create(thumbnailFileName))
    {
        thumbnailStream.CopyTo(thumbnailFile);
    }

    Console.WriteLine($"Thumbnail saved in {thumbnailFileName}");
}
```

Python

Code Copy

```
# Generate a thumbnail
with open(image_file, mode="rb") as image_data:
    # Get thumbnail data
    thumbnail_stream = cv_client.generate_thumbnail_in_stream(100, 100, image_data, True)

# Save thumbnail image
thumbnail_file_name = 'thumbnail.png'
with open(thumbnail_file_name, "wb") as thumbnail_file:
    for chunk in thumbnail_stream:
        thumbnail_file.write(chunk)

print('Thumbnail saved in.', thumbnail_file_name)
```

1. Save your changes and run the program once for each of the image files in the **images** folder, opening the **thumbnail.jpg** file that is generated in the same folder as your code file for each image.

More information

In this exercise, you explored some of the image analysis and manipulation capabilities of the Computer Vision service. The service also includes capabilities for reading text, detecting faces, and other computer vision tasks.

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