2.6 - Read Text in images and documents with the Azure AI Vision Service

---

Overview

Azure's AI Vision service uses algorithms to process images and return information. This module teaches you how to use the Image Analysis API for optical character recognition (OCR).

Learning objectives

In this module, you'll learn how to:

- Read text from images using OCR
- Use the Azure AI Vision service Image Analysis with SDKs and the REST API
- Develop an application that can read printed and handwritten text

---

Introduction

Suppose you are given thousands of images and asked to transfer the text on the images to a computer database. The scanned images have text organized in different formats and contain multiple languages. What are some ways you could complete the project in a reasonable time frame and make sure the data is entered with a high degree of accuracy?

Companies around the world are tackling similar scenarios every day. Without AI services, it would be challenging to complete the project, especially if it were to change in scale.

Using AI services, we can treat this project as an Azure AI Vision scenario and apply Optical Character Recognition (OCR). OCR allows you to extract text from images, such as photos of street signs and products, as well as from documents — such as handwritten or unstructured documents.

To build an automated AI solution, you need to train machine learning models to cover many use cases. Azure AI Vision service gives access to advanced algorithms for processing images and returns data to

secure storage.

In this module, you'll learn how to:

- Identify how the Azure AI Vision service enables you to read text from images
- Use the Azure AI Vision service with SDKs and the REST API
- Develop an application that can read printed and handwritten text

---

Explore Azure AI Vision options for reading text

Azure AI provides two different features that read text from documents and images, one in the Azure AI Vision Service, the other in Azure AI Document Intelligence. There is overlap in what each service provides, however each is optimized for results depending on what the input is.

- **Image Analysis** Optical character recognition (OCR):
  - Use this feature for general, unstructured documents with smaller amount of text, or images that contain text.
  - Results are returned immediately (synchronous) from a single API call.
  - Has functionality for analyzing images past extracting text, including object detection, describing or categorizing an image, generating smart-cropped thumbnails and more.
  - Examples include: street signs, handwritten notes, and store signs.
- **Document Intelligence**:
  - Use this service to read small to large volumes of text from images and PDF documents.
  - This service uses context and structure of the document to improve accuracy.
  - The initial function call returns an asynchronous operation ID, which must be used in a subsequent call to retrieve the results.
  - Examples include: receipts, articles, and invoices.

You can access both technologies via the REST API or a client library. In this module, we'll focus on the OCR feature in **Image Analysis**. If you'd like to learn more about **Document Intelligence**, reading this module will provide a good introduction.

Use the Read API

To use the Read OCR feature, call the **ImageAnalysis** function (REST API or equivalent SDK method), passing the image URL or binary data, and optionally specifying a gender neutral caption or the language the text is written in (with a default value of **en** for English).

To make an OCR request to **ImageAnalysis**, **specify the visual feature as** `READ`.

C#

```
ImageAnalysisResult result = client.Analyze(
    <image-to-analyze>,
    VisualFeatures.Read);
```

Python

```
result = client.analyze(
    image_url=<image_to_analyze>,
    visual_features=[VisualFeatures.READ]
)
```

If using the REST API, specify the feature as `read`.

```
https://<endpoint>/computervision/imageanalysis:analyze?features=read&...
```

The results of the Read OCR function are returned synchronously, either as JSON or the language specific object of a similar structure. These results are broken down in *blocks* (with the current service only using one block), then *lines*, and then *words*.

Additionally, the text values are included at both the *line* and *word* levels, making it easier to read entire lines of text if you don't need to extract text at the individual *word* level.

```
{
    "metadata":
    {
        "width": 500,
        "height": 430
    },
    "readResult":
    {
        "blocks":
        [
            {
                "lines":
                [
                    {
                        "text": "Hello World!",
                        "boundingPolygon":
                        [
                            {"x":251,"y":265},
                            {"x":673,"y":260},
                            {"x":674,"y":308},
                            {"x":252,"y":318}
                        ],
                        "words":
                        [
                            {
                                "text":"Hello",
                                "boundingPolygon":
                                [
                                    {"x":252,"y":267},
                                    {"x":307,"y":265},
                                    {"x":307,"y":318},
```

```
                                {"x":253,"y":318}
                            ],
                        "confidence":0.996
                        },
                        {
                            "text":"World!",
                            "boundingPolygon":
                            [
                                {"x":318,"y":264},
                                {"x":386,"y":263},
                                {"x":387,"y":316},
                                {"x":319,"y":318}
                            ],
                            "confidence":0.99
                        }
                    ]
                },
            ]
        }
    ]
}
}
```

Exercise - Read text in images

Optical character recognition (OCR) is a subset of computer vision that deals with reading text in images and documents. The **Azure AI Vision** service provides an API for reading text, which you'll explore in this exercise.

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 read text.

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

1. In Visual Studio Code, in the **Explorer** pane, browse to the **Labfiles\05-ocr** folder and expand the **C-Sharp** or **Python** folder depending on your language preference.
2. Right-click the **read-text** 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 **read-text** 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.

Use the Azure AI Vision SDK to read text from an image

One of the features of the **Azure AI Vision SDK** is to read text from an image. In this exercise, you'll complete a partially implemented client application that uses the Azure AI Vision SDK to read text from an image.

1. The **read-text** folder contains a code file for the client application:

   - **C#**: Program.cs
   - **Python**: read-text.py (View here: read-text.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'll 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
   ```

2. In the code file for your client application, in the **Main** function, 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 an 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) )
   ```

3. 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 the **GetTextRead** function. This function isn't yet fully implemented.
4. Let's add some code to the body of the **GetTextRead** function. Find the comment **Use Analyze image function to read text in image**. Then, under this comment, add the following language-specific code,

noting that the visual features are specified when calling the `Analyze` function:

C#

```
// Use Analyze image function to read text in image ImageAnalysisResult result = client.Analyze( BinaryData.FromStream(stream), // Specify the features to be retrieved VisualFeatures.Read); stream.Close(); // Display analysis results if (result.Read != null) { Console.WriteLine($"Text:"); ``` // 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); foreach (var line in result.Read.Blocks.SelectMany(block =&gt; block.Lines)) { // Return the text detected in the image } // Save image String output_file = "text.jpg"; image.Save(output_file); Console.WriteLine("\nResults saved in " + output_file + "\n"); ``` }
```

Python

```
# Use Analyze image function to read text in image result = cv_client.analyze( image_data=image_data, visual_features=[VisualFeatures.READ] ) # Display the image and overlay it with the extracted text if result.read is not None: print("\nText:") ``` # 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 line in result.read.blocks[0].lines: # Return the text detected in the image # Save image plt.imshow(image) plt.tight_layout(pad=0) outputfile = 'text.jpg' fig.savefig(outputfile) print('\n Results saved in', outputfile) ```
```

5. In the code you just added in the **GetTextRead** function, and under the **Return the text detected in the image** comment, add the following code (this code prints the image text to the console and generates the image **text.jpg** which highlights the image's text):

C#

```
// Return the text detected in the image Console.WriteLine($" '{line.Text}'"); // Draw bounding box around line var drawLinePolygon = true; // Return the position bounding box around each line // Return each word detected in the image and the position bounding box around each word with the confidence level of each word // Draw line bounding polygon if (drawLinePolygon) { var r = line.BoundingPolygon; ``` Point[] polygonPoints = { new Point(r[0].X, r[0].Y), new Point(r[1].X, r[1].Y), new Point(r[2].X, r[2].Y), new Point(r[3].X, r[3].Y) }; graphics.DrawPolygon(pen, polygonPoints); ``` }
```

Python

```
# Return the text detected in the image print(f" {line.text}") drawLinePolygon = True r = line.bounding_polygon bounding_polygon = ((r[0].x, r[0].y),(r[1].x, r[1].y),(r[2].x, r[2].y),(r[3].x, r[3].y)) # Return the position bounding box around each line # Return each word detected in the image and the position bounding box around each word with the confidence level of each word # Draw line bounding polygon if drawLinePolygon: draw.polygon(bounding_polygon, outline=color, width=3)
```

6. In the **read-text/images** folder, select **Lincoln.jpg** to view the file that your code processes.
7. 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 **GetTextRead** function, passing the path to the *Lincoln.jpg* image file.
8. Save your changes and return to the integrated terminal for the **read-text** folder, and enter the following command to run the program:

C#

```
dotnet run
```

Python

```
python read-text.py
```

9. When prompted, enter **1** and observe the output, which is the text extracted from the image.
10. In the **read-text** folder, select the **text.jpg** image and noticed how there's a polygon around each *line* of text.
11. Return to the code file in Visual Studio Code, and find the comment **Return the position bounding box around each line**. Then, under this comment, add the following code:

C#

```
// Return the position bounding box around each line Console.WriteLine($" Bounding Polygon: [{string.Join(" ", line.BoundingPolygon)}]");
```

Python

```
# Return the position bounding box around each line print(" Bounding Polygon: {}".format(bounding_polygon))
```

12. Save your changes and return to the integrated terminal for the **read-text** folder, and enter the following command to run the program:

C#

```
dotnet run
```

Python

```
python read-text.py
```

13. When prompted, enter **1** and observe the output, which should be each line of text in the image with their respective position in the image.
14. Return to the code file in Visual Studio Code, and find the comment **Return each word detected in the image and the position bounding box around each word with the confidence level of each word**. Then, under this comment, add the following code:

C#

```
// Return each word detected in the image and the position bounding box around each word with the confidence level of each word foreach (DetectedTextWord word in line.Words) { Console.WriteLine($" Word: '{word.Text}', Confidence {word.Confidence:F4}, Bounding Polygon: [{string.Join(" ", word.BoundingPolygon)}]"); ``` // Draw word bounding polygon drawLinePolygon = false; var r = word.BoundingPolygon; Point[] polygonPoints = { new Point(r[0].X, r[0].Y), new Point(r[1].X, r[1].Y), new Point(r[2].X, r[2].Y), new Point(r[3].X, r[3].Y) }; graphics.DrawPolygon(pen, polygonPoints); ``` }
```

Python

```
# Return each word detected in the image and the position bounding box around each word with the confidence level of each word for word in line.words: r = word.bounding_polygon bounding_polygon = ((r[0].x, r[0].y),(r[1].x, r[1].y),(r[2].x, r[2].y),(r[3].x, r[3].y))
```

`print(f" Word: '{word.text}', Bounding Polygon: {bounding_polygon}, Confidence: {word.confidence:.4f}") ``` # Draw word bounding polygon drawLinePolygon = False draw.polygon(bounding_polygon, outline=color, width=3) ````

15. Save your changes and return to the integrated terminal for the **read-text** folder, and enter the following command to run the program:

    C#

    ```
    dotnet run
    ```

    Python

    ```
    python read-text.py
    ```

16. When prompted, enter **1** and observe the output, which should be each word of text in the image with their respective position in the image. Notice how the confidence level of each word is also returned.
17. In the **read-text** folder, select the **text.jpg** image and noticed how there's a polygon around each *word*.

Use the Azure AI Vision SDK to read handwritten text from an image

In the previous exercise, you read well defined text from an image, but sometimes you might also want to read text from handwritten notes or papers. The good news is that the **Azure AI Vision SDK** can also read handwritten text with the same exact code you used to read well defined text. We'll use the same code from the previous exercise, but this time we'll use a different image.

1. In the **read-text/images** folder, select on **Note.jpg** to view the file that your code processes.
2. 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 **GetTextRead** function, passing the path to the *Note.jpg* image file.
3. From the integrated terminal for the **read-text** folder, enter the following command to run the program:

    C#

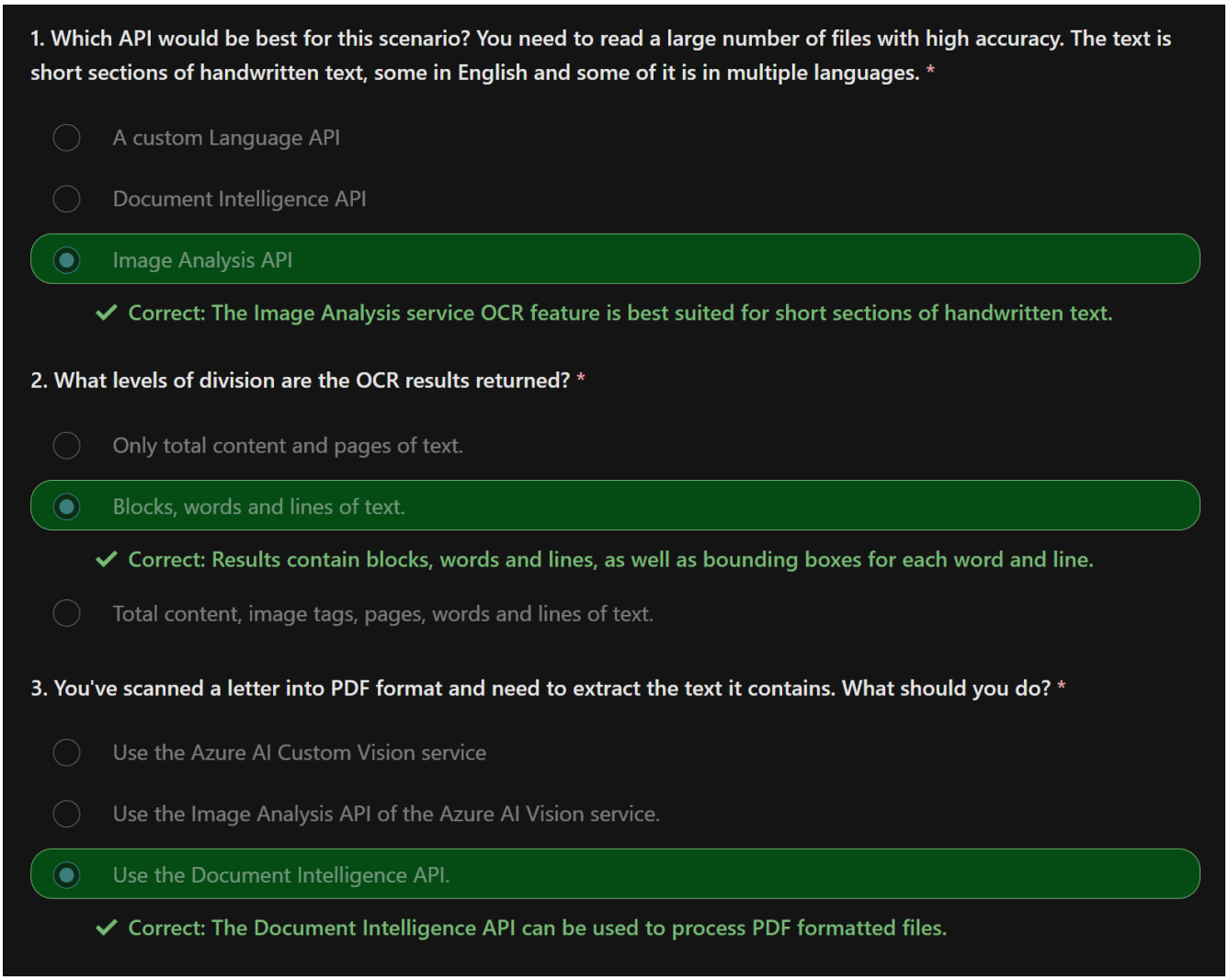
    ```
    dotnet run
    ```

    Python

    ```
    python read-text.py
    ```

4. When prompted, enter **2** and observe the output, which is the text extracted from the note image.
5. In the **read-text** folder, select the **text.jpg** image and noticed how there's a polygon around each *word* of the note.

Knowledge Check



> Note: Document Intelligence is the best choice for large amounts of structured text and multiple languages, however isn't the best choice for shorter, unstructured handwritten text. It can also be used to process PDF formatted files

---

Summary

In this module, you learned how to:

- Read text from images with **ImageAnalysis** READ feature
- Use the Azure AI Vision service with SDKs and the REST API
- Develop an application that can read printed and handwritten text

For more information, see the OCR documentation.

✍️ Compiled by Kenneth Leung (2025)