CHAPTER 5

Implementing Computer Vision Solutions with Azure AI

Computer vision isn't just about teaching machines to see; it's about rewriting the rules of how industries operate. Take your local supermarket: in November 2021, Carrefour launched its Flash 10/10 pilot in Paris to leverage AiFi's camera-only computer vision platform—which consists of ceiling-mounted cameras and AI-driven tracking—to let customers grab items and simply walk out while purchases are automatically tallied and charged. This tech isn't limited to groceries. Imagine a hospital where AI scans thousands of X-rays overnight, flagging subtle fractures radiologists might miss during a hectic morning shift. From spotting potholes in real time for city maintenance trucks to helping filmmakers animate lifelike CGI characters, computer vision acts as the unsung hero, transforming raw pixels into decisions that save time, money, and even lives.

Next, let's zoom in on cars. Modern vehicles are packed with cameras that do more than just help with parallel parking. Systems like Tesla's Autopilot analyze lane markings and pedestrians with the precision of a hyper-caffeinated copilot, and they make split-second adjustments to keep drivers safe. But here's the catch: building these systems requires more than just clever algorithms. On Azure, tools like AI Custom Vision let engineers train models to recognize everything from stop signs that are obscured by fog to debris on highways, while Video Indexer extracts metadata and insights from stored video and audio files. Want to prototype a shelf-monitoring system for retailers? You can spin up a model in hours that alerts staff when the last jar of artisanal pickles leaves the aisle.

In this chapter, through hands-on exercises and walkthroughs of these services, you'll gain practical knowledge of how to use Azure AI tools to craft the right computer vision solutions for your specific needs. Understanding all the configuration options

and best practices will enable you to make informed decisions and ensure that your solutions are both effective and scalable.

Introduction to Azure AI Vision

Before you can begin crafting implementations with Azure AI Vision, you need a firm grasp of the fundamentals of computer vision and its capabilities within the Azure ecosystem. In this section, we'll discuss what computer vision is, how Azure AI Vision contributes to it, and the relevant architecture.

Azure AI Vision is part of the broader Azure Cognitive Services suite, which currently offers features in general availability (GA) for standard subscription tiers through versions V3 and V4 of the Computer Vision API. Some advanced functionalities (such as the latest text extraction and background removal features) may require a Premium tier or a specific region for early preview access. Microsoft typically updates these APIs quarterly, so you should check the release notes for the relevant Azure AI service—for example, on the "What's New in Azure AI Search" page (*https://oreil.ly/UoY4v*)—for version changes and region-specific feature availability.

You can also refer to Table 5-1, which is a simplified compatibility matrix that shows which service tiers typically support which key features of Azure AI Vision.

Table 5-1. Compatibility matrix showing different service tiers' support for Azure AI Vision

Feature	Free tier	Standard (S0)	Standard (S1/premium)
Image analysis (basic tags)	Yes	Yes	Yes
Background removal (preview)	No	Yes	Yes
OCR text extraction	Limited	Yes	Yes
Landmark recognition	Yes	Yes	Yes (for faster performance)
Custom vision models	No	Yes (on a custom tier)	Yes (on a custom tier)

Note that the availability of features may vary by region. For detailed subscription requirements, consult Azure's official documentation on pricing tiers and region availability for Computer Vision and related AI services.

What Is Computer Vision?

Computer vision is a field of AI that enables computers and systems to extract meaningful insights from digital images, videos, and other relevant formats, then provide recommendations or perform actions based on that information. Computer vision algorithms are designed to process, analyze, and interpret visual data, with the goal of enabling AI systems to perceive and understand images on a level comparable to human perception and understanding.

There are many applications that make use of computer vision, including facial recognition and medical image analysis systems. Computer vision is also closely integrated with other types of AI, such as generative AI, which can process images, provide insights into medical data, and enable interpretations at a level that's understandable by doctors. Core tasks in computer vision include image recognition, image generation, and image restoration.

What Is Azure AI Vision?

Azure AI Vision is a service that offers developers and data scientists prebuilt models and tools that they can use to integrate computer vision capabilities into their applications without having to first gain a deep knowledge of machine learning or AI. It provides a range of functionalities, such as generating descriptions and tagging, identifying objects, and recognizing people in images (see Table 5-2).

Table 5-2. Capabilities of Azure AI Vision

Capability	Example use case
Generating descriptions and tagging	Creating suitable titles for images and pinpointing relevant keywords that reflect an image's content
Identifying objects in images	Recognizing and pinpointing the locations of specific items
Recognizing people in images	Identifying the presence, positions, and characteristics of individuals in photographs
Analyzing image characteristics, colors, and formats	Assessing an image's dimensions, format, and predominant color schemes, and determining if the image includes clip art
Classifying images	Assigning images to proper categories and recognizing if the images feature recognizable landmarks
Eliminating backgrounds from images	Identifying and removing the backdrop from pictures, thus providing images with either a transparent background or a grayscale alpha channel
Assessing content for moderation	Evaluating images to identify any content that may be considered adult or violent
Extracting text from images	Interpreting and reading text contained in photographs
Creating condensed versions of images	Determining the focal point of an image to produce a reduced thumbnail version

To give you a better idea of how these capabilities work, let's examine a case study of a production workflow that uses some of them.

Case Study

To gain a clearer understanding of what AI Vision workflows look like, consider the example depicted in Figure 5-1. By integrating various Azure services—such as Blob Storage for scalable data management, Machine Learning for model inference, and the Computer Vision API for image analysis—this architecture demonstrates a seamless and automated workflow for analyzing video content.

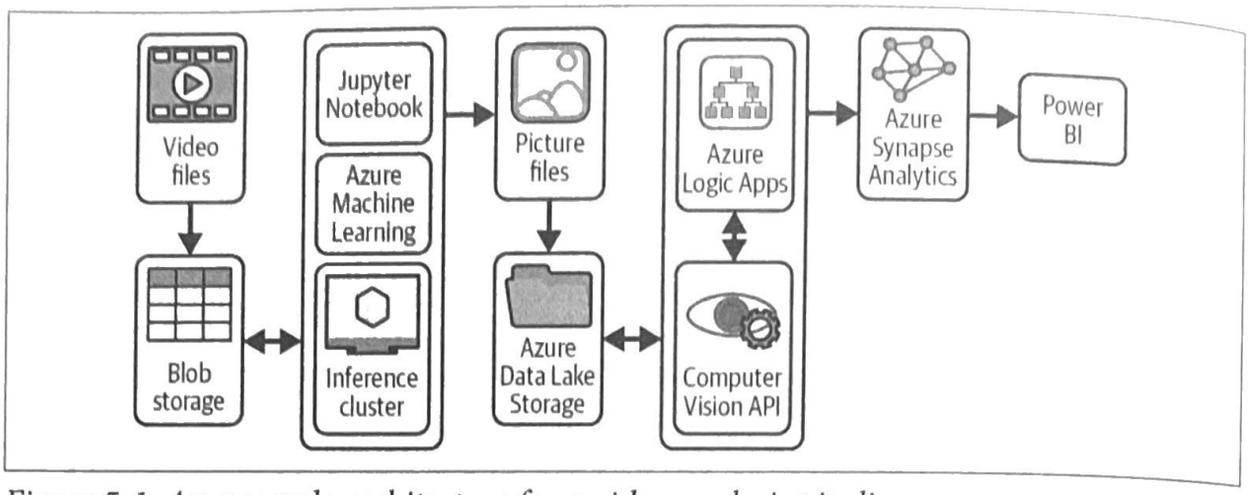

Figure 5-1. An example architecture for a video analysis pipeline

The architecture begins with the ingestion of video files, which are stored in a machine learning storage account. The stored videos are then sent to a machine learning pipeline for initial processing.

In the transformation stage, a Jupyter Notebook in the Azure Machine Learning environment contains scripts or code that orchestrate the subsequent processing of the data. FFmpeg, a versatile multimedia framework, is employed to convert the video files into individual image frames. These picture files are stored in Azure Data Lake Storage, which is designed to hold large volumes of data in its native format.

Once the data is transformed into image files, the enrichment and serving phase begins. Azure Logic Apps, which facilitate the creation of automated workflows, are used to manage the processing of these files. The image data is then analyzed by either the Custom Vision API or the Computer Vision API, both of which are part of Azure Cognitive Services. These APIs extract insights from the images, and the output, typically in JSON format, is parsed for further processing.

Finally, the processed data is channeled into Azure Synapse Analytics, a service that manages and analyzes large datasets. The last step in the workflow is the visualization of the data in Power BI, which allows for the creation of interactive reports and dashboards. With Power BI, users can explore an present the insights derived from the analyzed image data, completing the end-to-end workflow from video ingestion to data visualization.

Image Analysis with Azure AI Vision

Image analysis is a pivotal aspect of modern technology that transforms how we interact with and interpret visual data. Azure AI Vision offers a comprehensive suite of tools for image analysis that leverage advanced machine learning models to extract meaningful insights from images. These tools are designed to be robust and versatile so that they can cater to various applications in multiple industries. By utilizing Azure

AI Vision, businesses and developers can automate complex image processing tasks, enhance operational efficiency, and derive actionable intelligence from visual content.

The Fundamentals of Image Analysis

At its core, *image analysis* involves extracting useful information from images. This process encompasses several fundamental steps, starting with *image acquisition*, in which digital images are captured with devices such as cameras and scanners. Once acquired, these images undergo *preprocessing* to enhance their quality and make them suitable for analysis. Preprocessing techniques may include noise reduction, contrast adjustment, and image resizing.

Next begins the actual *analysis*, in which the image is examined to identify specific features. Azure AI Vision employs state-of-the-art algorithms to detect and classify objects, recognize text, analyze colors, and even generate descriptive captions. This ability to identify and label objects within an image is essential for applications like automated tagging, inventory management, and security surveillance.

One of the critical components of image analysis is *optical character recognition* (OCR), which converts text within images into machine-readable text. This capability is crucial for digitizing documents, extracting information from scanned forms, and enhancing accessibility by converting printed text into formats that are readable by screen readers.

Azure AI Vision also incorporates advanced capabilities such as *facial recognition*, which can identify and verify individuals based on their facial features. These technologies enable applications in security, customer behavior analysis, and augmented reality.

What follows is a minimal example of how you might apply preprocessing steps like noise reduction, contrast adjustment, and image normalization prior to sending an image to Azure AI Vision. Although Azure AI Vision can handle various input qualities, cleaning images first can improve analysis accuracy:

```
import cv2
import numpy as np

def preprocess_image(image_path):
    # Read image with OpenCV
    img = cv2.imread(image_path)
    img_denoised = cv2.GaussianBlur(img, (3, 3), 0)
    yuv_img = cv2.cvtColor(img_denoised, cv2.COLOR_BGR2YUV)
    yuv_img[:, :, 0] = cv2.equalizeHist(yuv_img[:, :, 0])
    img_contrast = cv2.cvtColor(yuv_img, cv2.COLOR_YUV2BGR)
    img_normalized = img_contrast.astype(np.float32) / 255.0
    return img_normalized

processed_image = preprocess_image("sample.jpg")
```

In this example, we use `OpenCV` to demonstrate basic noise reduction (Gaussian blur), contrast adjustment (histogram equalization), and normalization (scaling pixel values). For color-based analyses, you need to ensure that your transformations align with Azure's expected input format. You can also extend this approach to other scenarios, including resizing large images or compressing them to fit memory constraints.

When you're extracting specific features (e.g., edges, corners, custom embeddings), you can cache intermediate representations to avoid recomputing them in subsequent operations. If memory is a bottleneck, consider processing images in smaller batches or using streaming APIs. In GPU-accelerated environments, ensure that data transfers between CPUs and GPUs are minimized to reduce overhead. Data augmentation, such as by performing random rotations or flipping, can help with model generalization but might increase computational load. You'll need to balance these trade-offs based on your application's accuracy and performance needs.

Azure AI Vision supports multiple formats, including JPEG, PNG, GIF, and BMP. However, high-resolution TIFF or RAW images may require conversion to a more common format. You can downsample large images to reduce cost and latency, but you might lose important details in the process—evaluate your use case carefully to determine whether lower-resolution images will suffice. Consider tiling extremely large images (e.g., satellite imagery) into manageable sections.

If you're troubleshooting preprocessing issues, you'll need to perform careful adjustments to maintain image quality and prevent processing errors. For instance, if noise reduction is too aggressive, excessive blur, artifacts, and loss of critical details can occur. To mitigate this, try adjusting kernel sizes or experimenting with different noise-reduction techniques to help preserve important features. Another common issue is overequalization—applying histogram equalization to an already well-lit image can create unnatural brightness patterns, but adaptive methods like contrast limited adaptive histogram equalization (CLAHE) offer more controlled brightness distribution.

Memory crashes may also occur when you're processing large datasets. These can often be addressed by batching image processing tasks or using streaming techniques to handle data incrementally. Finally, mismatched color spaces can cause color distortions or misinterpretations during analysis, but by ensuring consistency among color spaces (such as RGB, BGR, or YUV) throughout the pipeline, you can help maintain accurate processing and interpretation of image data.

Performing Image Analysis

Performing image analysis with Azure AI Vision involves several steps, each designed to harness the full potential of Azure's machine learning capabilities. You start the process by setting up the environment and authenticating the Azure AI Vision

service. This involves creating a Vision Image Analysis client using the `AzureKeyCredential`, which securely manages the necessary credentials.

Once the environment is set up, images can be submitted for analysis through various methods, including uploading images from local storage and providing URLs for remote images. The `analyze` method is central to this process because it allows users to specify the visual features they want to analyze. These features include object detection, OCR, and caption generation.

For instance, to analyze an image for captions and text recognition, you invoke the `analyze` method with parameters that specify the desired features. This method processes the image and returns a detailed response containing the extracted information. The response will include detected objects, recognized text lines and words, generated captions, and confidence scores indicating the accuracy of each detection.

In practical applications, users can employ this process to automate a variety of tasks. Retailers can use object detection to manage inventory and monitor store layouts, ensuring that products are correctly placed and stock levels are maintained. Healthcare providers can use OCR to digitize patient records, making it easier to access and manage medical histories. Security applications can leverage facial recognition and insight extraction to enhance surveillance systems and improve safety.

Azure AI Vision also provides extensive customization options that allow users to tailor the analysis to their specific needs. For example, users can choose to generate gender-neutral captions or specify the language for text recognition. By using these customizations, users can ensure that their analysis results are relevant to and useful for their intended applications.

Selecting the appropriate visual features

When you're working with the Azure AI Vision Image Analysis client, you need to know which visual features the service can identify. This will help you understand how to integrate this service with more components in Azure to better work with it in a production workflow. Table 5-3 summarizes those features.

Table 5-3. Visual features that the Azure AI Vision Image Analysis client can work with

Features	Descriptions
Tags	Identify and tag visual elements in an image by drawing on a vast set of recognizable objects, living beings, types of scenery, and activities.
Objects	Detect and locate objects in an image and return their bounding box coordinates.
Descriptions	Generate human-readable captions for images in complete sentences.
Faces	Detect human faces in an image and provide details like coordinates, gender, and age.
Image types	Determine specific characteristics of an image, such as if it's a line drawing or clip art.
Color schemes	Analyze color usage within the image, identifying whether it's black and white or in color and what the dominant and accent colors are.

Features	Descriptions
Brands	Identify any commercial brands that are present in the image.
Adult content	Detect any adult content in the image and provide confidence scores for various classifications of such content.

We can capture these features and act on them based on specific thresholds. For instance, when detecting faces, we can determine how many faces are present in an image by evaluating the visual feature for detected faces.

The following code demonstrates how to use the Azure AI Vision Image Analysis client to analyze specific visual features in an image. This is particularly useful in tasks involving image recognition and processing that can leverage Azure's powerful AI capabilities:

```
from azure.ai.vision import ImageAnalysisClient as VisionAnalysisClient
from azure.core.credentials import AzureKeyCredential
from azure.ai.vision.models import VisualFeatures
import os
with open("owl.png", "rb") as f:
    image_data = f.read()
vision_client = VisionAnalysisClient(
    endpoint=os.environ.get("AZURE_VISION_ENDPOINT"),
    credential=AzureKeyCredential(os.environ.get("AZURE_SUBSCRIPTION_KEY"))
)

analysis_result = vision_client.analyze(
    image_data=image_data,
    visual_features=[VisualFeatures.CAPTION],
    gender_neutral_caption=True,
    language="en"
)
print(
    f"'{analysis_result.caption.text}', "
    f"Confidence {analysis_result.caption.confidence:.4f}"
)
```

Different feature combinations—e.g., Faces + Objects + Read—can increase CPU and GPU usage and latency. Tests show that each additional feature can add an average of approximately 10–20% extra latency and memory usage for medium-resolution images (e.g., 1024 × 768). For high-resolution images (e.g., 4K), memory usage can spike by an additional 50%, which will impact both local GPU memory and the cost of compute resources in Azure. Therefore, if cost is a concern, you should consider separating tasks (e.g., running OCR offline while performing near-real-time object detection).

Different feature combinations have varying impacts on system resources. For example, Tags + OCR requires moderate CPU usage with minimal GPU overhead, while Objects + Adult content demands higher GPU usage due to bounding box

computations and classification models. Faces + Brands + Captions is among the most resource-intensive feature combination options because it involves multiple deep-learning models operating simultaneously, requiring greater computational power and memory.

You can use a decision tree approach to help you choose the most efficient feature or feature combination for your specific use case. If you need to analyze text, include Read; otherwise, exclude it to save on processing time and cost. If you need to use object bounding boxes, enable Objects; otherwise, use Tags for broader classification. If you need to detect people or faces, enable Faces; otherwise, exclude them to optimize performance. If you need to perform brand analysis or adult content moderation, include Brands and Adult content; otherwise, omit them.

In retail applications, detecting products often requires Objects + Brands, while Faces may be optional unless you're analyzing customer interactions. For social media platforms, a combination of Tags + Adult content can help moderate user-generated content, and you can also add Read if you're scanning posted images for text-based policy violations. If you're performing document digitization, prioritize Read and include Faces only if you need to perform identity verification (e.g., scanning forms or IDs).

You can also batch multiple images into a single API call to reduce processing overhead and optimize costs. Employing asynchronous processing allows you to defer less critical feature extractions, such as brand detection, to off-peak times to reduce compute costs. To improve your overall cost efficiency, consider investigating regional pricing variations—certain Azure regions may offer lower costs or special promotions for Cognitive Services.

Note that to run the code in the previous example, you will have to configure your local `AZURE_VISION_ENDPOINT` and `AZURE_SUBSCRIPTION_KEY` environment variables to point to your Azure AI Vision service endpoint and your subscription key, respectively. You will also need to provide the URL for an image that Azure AI Vision can ingest.

The code first imports the required libraries, including the `ImageAnalysisClient` from the `azure.ai.vision` module and `AzureKeyCredential` from the `azure.core.credentials` module. Importing these libraries is essential for creating a client that can interact with Azure's image analysis services. It also imports the `os` module to securely manage environment variables, which store the endpoint and subscription key that are required for authentication.

The initialization of the `VisionAnalysisClient` is straightforward. By fetching the endpoint and subscription key from environment variables, the code ensures that sensitive information is not hardcoded, thus enhancing security. This setup allows the client to communicate with the Azure AI Vision service, enabling it to send requests and receive analysis results.

The core functionality of the code lies in the `analyze` method call. This method requests the analysis of specific visual features in an image loaded from local storage. In this example, the code specifies the Captions feature, which generates human-readable descriptions of the image. These can be particularly useful for improving accessibility, enhancing content metadata for search engines, or providing automatic descriptions in digital asset management systems. The Read feature, on the other hand, performs OCR to extract text from the image. It thus facilitates tasks such as document digitization, automated data entry, and content moderation by scanning for inappropriate or sensitive text.

The method also includes parameters for gender-neutral descriptions and language specification, ensuring that the generated captions are inclusive and tailored to the desired language (in this case, English). This level of customizability makes the Azure AI Vision service highly versatile for various applications across different industries.

This is a great example of how you can use the Azure AI Vision Image Analysis client to extract and utilize visual data from images. By leveraging these capabilities, you can automate and enhance your workflows.

Detecting objects in images and generating image tags

Object detection in Azure AI Vision involves identifying individual objects within an image and specifying their locations using bounding boxes. This feature is part of the Analyze Image API, and you can access it via the Azure SDKs or REST API. Including Objects in the `VisualFeatures` query parameter instructs the API to return detailed information about each detected object, including its type, coordinates within the image, and a confidence score indicating the certainty of the detection.

For example, an image containing a kitchen scene might return objects such as *kitchen appliance*, *computer keyboard*, and *person*, each with a specific confidence score and bounding box coordinates. This detailed output allows applications to process and understand the spatial relationships among objects, which is particularly useful for applications in security surveillance, automated inventory management, and the like.

The object detection feature is designed to handle various scenarios, though it has some limitations. For example, it has difficulty detecting small objects (objects that make up less than 5% of an image) or objects arranged closely together. However, despite these challenges, object detection remains a valuable tool for extracting information from images and supporting complex visual analysis tasks.

Image tagging is another essential feature of Azure AI Vision that automatically generates tags for recognizable objects, living beings, scenery, and actions within an image. Unlike object detection, which focuses on identifying and locating objects,

image tagging provides broader context by including tags for the setting, such as "indoor" or "outdoor," and for specific elements like furniture, plants, and gadgets.

The tagging process involves analyzing the image and returning a collection of tags, each with an associated confidence score. These tags help categorize and describe the content of the image, making it easier to manage and search through large collections of visual data. For instance, an image of a house might be tagged with terms like *grass*, *building*, *sky*, and *real estate*, each accompanied by a confidence score indicating the likelihood of the tag's accuracy.

Image tags form the foundation for generating descriptive sentences that can be used in various applications, from enhancing search engine optimization (SEO) to improving the accessibility of digital content by providing text descriptions for visually impaired users. The flexibility and accuracy of Azure AI Vision's image tagging make it a valuable tool for enriching metadata and improving content discoverability.

Interpreting image processing responses

Knowing how to interpret image processing responses go a long way toward helping you work effectively with Azure AI Vision, as you will need to process and act on the different attributes in the endpoint's response. The following example of a JSON response from Azure AI's image processing service illustrates how text is extracted and structured from images using OCR. This response offers a comprehensive view of the text that's been detected, including its location in the image and the confidence levels associated with the detections:

```
{
  "readResults": [
    {
      "lines": [
        {
          "text": "Town Hall",
          "boundingBox": [546, 180, 590, 190],
          "words": [
            {
              "text": "Town",
              "boundingBox": [547, 181, 568, 191],
              "confidence": 0.98
            },
            {
              "text": "Hall",
              "boundingBox": [570, 181, 590, 191],
              "confidence": 0.99
            }
          ]
        },
        {
          "text": "9:00 AM - 10:00 AM",
          "boundingBox": [546, 191, 596, 200],
```

```
            "words": [
              {"text": "9:00", "confidence": 0.09},
              {"text": "AM", "confidence": 0.99},
              {"text": "-", "confidence": 0.69},
              {"text": "10:00", "confidence": 0.88},
              {"text": "AM", "confidence": 0.99}
            ]
          }
        ]
      }
    ]
}
```

The `readResults` array forms the core of the JSON structure and encapsulates the results of the text extraction process. Each entry in this array represents a segment of the image that has been analyzed. This organization allows for a detailed and structured breakdown of multiple sections or pages within a single image, ensuring thorough text extraction.

Within each entry of the `readResults` array, there's a `lines` array that contains objects representing lines of text that have been detected in the image. For example, `"Town Hall"` is one such detected text line. Each line object includes several attributes, with the `text` attribute containing the actual string detected. The `boundingBox` attribute is an array of coordinates (e.g., `[546, 180, 590, 190]`) that delineate the location of the text within the image. This is essential for visually mapping the text on the image, and it supports applications like document digitization and layout analysis.

Each line object is further decomposed into individual words, each represented by a `words` array. Each word object also includes a `text` attribute (which specifies the detected word, such as `"Town"`), a `boundingBox` attribute (which gives the coordinates of the word's location within the image, such as `[547, 181, 568, 191]`), and a `confidence` attribute (a numerical value, such as `0.98` for `"Town"` and `0.99` for `"Hall"`, that indicates the OCR system's confidence in the accuracy of the detected word). High confidence scores suggest greater reliability of the OCR detection.

The JSON response also handles more complex text structures. For instance, the `"9:00 AM - 10:00 AM"` time range is broken down into its components, each with its own confidence level. This level of granularity allows for precise text analysis, even when some parts of the text have lower confidence. For example, the `"9:00"` component has a confidence of `0.09`, which indicates some uncertainty, while `"AM"` has a high confidence of `0.99`, indicating a high level of certainty.

Azure AI's OCR capabilities, as demonstrated in this JSON response, have significant practical applications across various industries. With document digitization, businesses can convert physical documents into digital text, making records searchable and editable. This is particularly valuable in the healthcare, legal, and finance sectors.

Automated data entry, such as extracting text from forms and receipts, can streamline processes, reduce errors, and increase efficiency, benefitting the retail and ecommerce sectors. Additionally, extracting and analyzing text from images aids content moderation on social media platforms, helping ensure adherence to community guidelines and prevent the spread of harmful content. Finally, converting text within images into readable formats enhances accessibility for visually impaired users by making content accessible to them through screen readers.

To help you implement these OCR capabilities appropriately, Azure provides a suite of tools, including Cognitive Services APIs like the Computer Vision API and Azure AI Document Intelligence. These tools allow users to submit images or documents for processing and retrieve structured text data, facilitating applications ranging from document management to automated workflows.

Extracting Text with Azure AI Vision

In today's data-driven world, businesses in numerous industries must be able to efficiently extract and process information from various forms of media. Azure AI Vision's text extraction capabilities play a crucial role in this process by enabling organizations to automate the extraction of text from images, documents, and videos. This functionality leverages advanced OCR and computer vision algorithms to accurately identify and convert text into a machine-readable format.

Choosing the workload

Choosing the right type of workload for extracting text is an important part of using Azure AI Vision in production environments. The two main approaches are optical character recognition and document intelligence.

OCR focuses on recognizing and extracting text from digital images, such as scanned documents and photos, and converting it into machine-readable text. It's designed to handle both printed and handwritten text from images and documents, which makes it very useful when you need to digitize paper-based information or extract insights from visual assets.

Examples of use cases for OCR include:

- Digitizing paper records
- Automating data entry processes
- Enhancing accessibility through the conversion of handwritten notes into text that can be read by a screen reader

Document intelligence expands on OCR by interpreting the structure, context, and semantics of the document, rather than just recognizing the text. It does this by

employing machine learning models to identify and extract key-value pairs, tables, and entities from forms and other documents.

Common use cases for document intelligence include:

- Automated form processing
- Understanding complex documents, such as legal documents and contracts
- Enhancing content management systems through categorization

To help you better understand how these features are used, let's walk through a practical exercise that implements OCR.

Practical: Extracting and converting handwritten text

The following Python script demonstrates how to use Azure AI Vision's OCR capabilities to extract both printed and handwritten text from images. It provides a practical example of how to integrate Azure's powerful OCR functionality into a Python application, enabling efficient text extraction from visual data. This approach can help you process large volumes of images quickly, improve automation, and enhance data accessibility.

Install the Azure AI Image Analysis package with the following command:

```
pip install azure-ai-vision-imageanalysis
```

We then can proceed by importing the libraries and modules that are required for the script to function, as follows:

```
import os
from os.path import join as join_paths
from azure.ai.vision.imageanalysis import ImageAnalysisClient
from azure.ai.vision.imageanalysis.models import VisualFeatures
from azure.core.credentials import AzureKeyCredential
from dotenv import load_dotenv
```

These include standard (`os`) libraries for file path manipulation, Azure SDK classes for image analysis, and *`dotenv`* to load environment variables from a *.env* file. The environment variables you defined earlier (`AZURE_VISION_ENDPOINT` and `AZURE_SUBSCRIPTION_KEY`) are essential for authenticating requests to the Azure AI Vision API. Note that we have created these environmental variables at an earlier exercise.

We then define an `execute_main_process` function, which is the core of the script and is responsible for orchestrating the OCR process:

```
def execute_main_process():
    load_dotenv()
    endpoint = os.getenv('AZURE_VISION_ENDPOINT')
    key = os.getenv('AZURE_SUBSCRIPTION_KEY')
```

```
    print(
        '\nOptions:\n'
        '1: Analyze "Lincoln.jpg" using OCR\n'
        '2: Decode handwriting in "Note.jpg"\n'
        'Press any other key to exit\n'
    )
    user_choice = input('Choose an option:')
    if user_choice == '1':
        path_to_image = join_paths('images', 'Lincoln.jpg')
        process_text_extraction(path_to_image, endpoint, key)
    elif user_choice == '2':
        path_to_image = join_paths('images', 'Note.jpg')
        process_text_extraction(path_to_image, endpoint, key)
    else:
        print("Exiting...")
```

This function loads the environment variables to get the Azure AI Vision API credentials, then presents the user with options for analyzing specific images. Depending on the user's choice, it calls the `process_text_extraction` function with the appropriate image path, endpoint, and key. If the user inputs anything other than one of the predefined options, the script simply exits.

Next, we define the `process_text_extraction` function, which handles the actual OCR processing:

```
def process_text_extraction(path_to_image, endpoint, key):
    credential = AzureKeyCredential(key)
    client = ImageAnalysisClient(endpoint=endpoint, credential=credential)
    with open(path_to_image, "rb") as image_file:
        image_data = image_file.read()
    result = client.analyze(
        image_data=image_data,
        visual_features=[VisualFeatures.READ]
    )

    if result.read is not None:
       for block in result.read.blocks:
            for line in block.lines:
                for word in line.words:
                    print(word.text)
    else:
        print("No text recognized.")
```

First, it creates an `ImageAnalysisClient` using the provided endpoint and API key. The image is then read from disk and converted into a byte stream, which is necessary for uploading it to the Azure service. Next, the function calls the Azure AI Vision API to analyze the image and extract text. If any text is detected, it prints each line to the console. If no text is found, it informs the user that no text was recognized.

Finally, we have the script's entry point:

```
if __name__ == "__main__":
    execute_main_process()
```

This ensures that `execute_main_process` is called only when the script is run directly, not when it's imported as a module in another script. You can now run the script and see the workload being executed.

Facial Recognition and Analysis

Facial recognition and analysis are advanced technologies that utilize algorithms to detect, recognize, and analyze human faces in images and videos. These systems identify unique facial features and patterns to determine a person's identity or assess attributes such as age, gender, and emotion. The technologies rely on machine learning models and computer vision techniques to process and interpret facial data with high accuracy and speed.

Facial recognition and analysis are important for several reasons. First, they enhance security and surveillance systems by allowing for the real-time identification and tracking of individuals. This capability is crucial for public safety, because it enables law enforcement agencies to quickly identify suspects and locate missing persons. Second, these technologies streamline authentication processes, offering a more secure and convenient alternative to traditional methods such as using passwords and PINs. For instance, facial recognition is widely used in smartphones and laptops for biometric authentication, providing users with a seamless login experience.

Law enforcement agencies and security organizations also use facial recognition to monitor public spaces, airports, and borders. This technology helps them identify criminals and respond to security threats, thereby enhancing public safety. For example, the Metropolitan Police in London has employed facial recognition technology to scan crowds for known offenders. Businesses and institutions also use facial recognition for access control to secure buildings and sensitive areas, so employees and authorized personnel can gain entry without key cards or passwords, which can be lost or stolen. Finally, airports such as Changi Airport in Singapore utilize facial recognition for seamless passenger check-ins and boarding.

Facial recognition and analysis technologies are transforming various industries by enhancing security, convenience, and personalization. In Azure, these advancements are critical because they empower organizations with scalable, reliable, and compliant AI solutions. Azure's AI Face service provides robust tools for real-time identification, emotion detection, and demographic analysis, all while ensuring data privacy and security. This makes Azure an ideal platform for integrating facial recognition capabilities into enterprise applications, enabling businesses to harness the power of AI with confidence and agility as they innovate in different sectors.

Fundamentals of Facial Recognition

Facial recognition technologies offer a range of capabilities that enable organizations to extract meaningful information from human faces across various types of media. These include:

- Detection of faces within an image, including the identification of facial boundaries
- Detailed analysis of facial attributes, such as the orientation of the head, the presence of eyewear, image clarity, identification of key facial points, coverage, and additional features
- Comparison and confirmation of facial identities
- Identification of individuals through facial analysis

When implementing facial recognition, organizations must enforce data protection strategies and encryption mechanisms to secure stored facial data. Azure Key Vault can store all necessary credentials and encryption keys, while customer-managed keys (CMKs) can help ensure compliance with regulations like GDPR and HIPAA. For data in transit, enforce TLS 1.2+ encryption and limit network exposure using private endpoints or virtual networks. Incorporate Azure Policy for resource governance to prevent noncompliant configurations at scale.

In real-time scenarios such as surveillance or operating interactive kiosks, consider employing GPU-based virtual machines (like the NC or ND series) and scaling with AKS to handle traffic bursts. You can also cache face embeddings and reuse them across comparisons to reduce redundant processing. Adopt asynchronous patterns or event-driven architectures (e.g., Azure Event Hubs and Azure Functions) to distribute workloads and minimize latency.

To ensure high availability for critical deployments (e.g., airport security), replicate facial recognition services across multiple Azure regions by using paired regions for geo-redundancy. Use Azure Front Door or Traffic Manager for global load balancing, and implement failover strategies so that if one region experiences downtime, facial recognition requests automatically route to a secondary region. Finally, log all failover events and integrate with Azure Monitor for alerts and dashboards that provide continuous insights into your service's health.

Considerations

When you're building facial recognition solutions, you need to address several key considerations that go beyond the technical capabilities. Privacy and security are paramount: since facial recognition inherently deals with sensitive personal data, you must ensure this data is handled with the highest level of confidentiality. The storage and processing of facial data must comply with relevant regulations, such as GDPR,

to prevent unauthorized access or misuse. This extends to the lifecycle of the data; you must ensure it's retained only for as long as necessary and then securely deleted.

Transparency is another critical consideration. Users and individuals whose data is being processed should be clearly informed about how their facial data is being used, the purpose behind its collection, and how long it will be retained. This will help trust with users and ensure that the deployment of facial recognition technology is in line with ethical standards. It's also important to offer ways for individuals to consent or opt out, where applicable, as this further supports the ethical use of this technology.

Fairness and inclusivity are equally important, particularly given the potential biases that can arise in AI systems. Azure AI Vision, like other AI platforms, needs to be trained and tested on diverse datasets to minimize biases related to race, gender, age, and other demographic factors. Failure to do so can result in disproportionate errors for certain groups, leading to unfair treatment or exclusion. Ensuring that the system is fair and inclusive requires continuous monitoring and updates to the underlying models to reflect the diversity of the populations being served.

Finally, you need to consider the implications of assigning an AI identifier to detected faces, which are stored for up to 24 hours. You must manage this retention period carefully to balance the need for accurate comparisons and verification with individuals' rights to privacy and control over their data. The reuse of facial data for comparisons can enhance the functionality of a system, but it also raises ethical questions about consent and data ownership, which you must address transparently and responsibly.

By taking such considerations into account, organizations can develop facial recognition solutions that are not only powerful and effective but also ethical and respectful of individual rights.

Functionalities

The Azure AI Face service has six main capabilities. These are detailed in Table 5-4.

Table 5-4. Capabilities of Azure AI Face

Functionality	Description
Face detection	Face detection is the fundamental functionality of the Azure AI Face service. It involves identifying and locating human faces in images. This feature can detect multiple faces within an image and provide coordinates for each detected face. It's crucial to use it in applications that need to identify the presence of faces before performing further analysis. Face detection is used in scenarios such as automated photo tagging, security surveillance, and audience measurement in marketing.
Face attribute analysis	Face attribute analysis extends the capabilities of face detection by identifying various attributes of detected faces, including age, gender, emotion, smiles, facial hair, head pose, and even accessories like glasses. This analysis helps users understand the demographics and emotional state of individuals in real time. It can be applied in customer service settings, targeted advertising, and interactive applications (e.g., retailers can analyze customer emotions to gauge satisfaction levels and tailor their services accordingly).

Functionality	Description
Facial landmark location	Facial landmark location involves identifying specific points on a face, such as the eyes, nose, mouth, and chin. Identifying these landmarks is essential for applications that require access to precise facial geometry to perform their tasks, such as apps that allow users to virtually try on glasses or makeup, those that analyze facial expressions, and those that provide augmented reality experiences. By accurately locating facial landmarks, developers can create applications that interact naturally with users' facial movements and expressions.
Face comparison	Face comparison functionality allows for the comparison of two facial images to determine if they belong to the same person. This is achieved by analyzing facial features and calculating similarity scores. Face comparison is used in authentication systems, such as those that verify users against their profile pictures or compare live images with stored records. This feature enhances security in applications that perform access control, identity verification, and fraud detection.
Facial recognition	Facial recognition builds on face detection and comparison by identifying or verifying individuals whose facial images are stored in a database of known faces. This functionality is critical for applications that rely on quick and accurate identification, such as security systems, time attendance tracking apps, and apps that provide personalized user experiences. Facial recognition technology can identify faces even under challenging conditions, such as poor lighting or partial occlusions, making it a reliable tool for identity verification and access management.
Facial liveness detection	Facial liveness detection is designed to prevent spoofing attacks by ensuring that the face presented to the camera is that of a live person and not a photo, video, or mask. This feature is essential for secure authentication systems, particularly in financial services and sensitive transactions. By using techniques like blink detection, head movement detection, and texture analysis, facial liveness detection helps ensure that the system is interacting with a real person and thereby enhances the security of biometric systems.

Response from performing the analysis

The JSON response from the Azure AI Face service provides a structured summary of the facial analysis performed on an image. It includes information about the model used, metadata about the image, and detailed attributes for each detected face. The following JSON snippet illustrates how the service structures its output, giving a clear view of the information captured during facial analysis:

```
{
  "modelVersion": "2024-02-01",
  "metadata": {
    "width": 500,
    "height": 700
  },
  "peopleResult": {
    "values": [
      {
        "boundingBox": {
          "x": 10,
          "y": 40,
          "w": 110,
          "h": 200
        },
        "confidence": 0.953217489
      },
```

```
        {
          "boundingBox": {
            "x": 380,
            "y": 120,
            "w": 130,
            "h": 170
          },
          "confidence": 0.925743289
        }
      ]
    }
  }
```

The `modelVersion` field specifies the version of the model used for the analysis, which is `2024-02-01` in this instance. This information is important for ensuring compatibility and understanding the specific features and improvements included in that version. The `metadata` section provides basic details about the image, such as its width and height (`500` and `700` pixels, respectively, in this case). This helps users understand the context and scale of the detected faces in the image.

The core of the response is the `peopleResult` section, which contains an array of detected faces from the image. Each entry in this array includes a `boundingBox` and a confidence score. The bounding box provides the coordinates and dimensions of the detected face, which indicate its exact location within the image. For instance, one of the detected faces has a bounding box starting at `(10, 40)` with a width of `110` pixels and a height of `200` pixels. The confidence score represents the model's level of certainty that it has detected a face; here, the scores of `0.953` and `0.926` indicate high confidence.

This structured output from the Azure AI Face service can be utilized in various practical applications. In security and surveillance systems, it enables the detection and monitoring of individuals in real time and thus enhances safety measures. In access control systems, the technology facilitates secure, touchless entry by recognizing authorized individuals. In retail, face detection can be used to analyze customer demographics and behaviors, thus aiding in personalized marketing and improving customer experiences.

Practical: Implementing Facial Recognition in Your Application

In this section, we'll walk through a hands-on example of integrating facial recognition capabilities directly into an application, exploring key implementation steps and best practices. To get started, follow these steps:

1. In the Azure portal, click "Create a resource" and search for "Face." Select the Face API from the search results, and create a new instance.

2. Go to the resource, and make a note of the subscription key and endpoint URL because you'll need them to interact with the API.

3. Install the Azure Face client library for Python using `pip`:

```
pip install azure-ai-vision-face
```

4. Create a new script called *face_recognition.py*. Import the necessary libraries and create a `FaceClient` object with your subscription key and endpoint:

```
from azure.ai.vision.face import FaceClient, FaceAdministrationClient
from azure.ai.vision.face.models import (
    FaceDetectionModel, FaceRecognitionModel,
    FaceAttributeTypeDetection03, FaceAttributeTypeRecognition04,
)
from azure.core.credentials import AzureKeyCredential
KEY = 'your_subscription_key_here'
ENDPOINT = 'your_endpoint_url_here'
face_client = FaceClient(ENDPOINT, AzureKeyCredential(KEY))
```

5. Use the `FaceClient`'s `detect_with_url` or `detect_with_stream` method to detect faces in an image. Include the `return_face_attributes` parameter to get details like age, gender, and emotions:

```
image_url = 'url_to_your_image'
detected_faces = face_client.detect_from_url(
    image_url,
    detection_model=FaceDetectionModel.DETECTION03,
    recognition_model=FaceRecognitionModel.RECOGNITION04,
    return_face_id=True,
    return_face_attributes=[
        FaceAttributeTypeDetection03.HEAD_POSE,
        FaceAttributeTypeDetection03.MASK,
        FaceAttributeTypeRecognition04.QUALITY_FOR_RECOGNITION,
    ],
)
```

6. Analyze the response by iterating over `detected_faces`. You can extract and print attributes like face ID, age, and gender for each detected face:

```
for face in detected_faces:
    print(f"Face ID: {face.face_id}")
    print(face.as_dict())
```

7. Identify faces (optional). If you're working with known individuals, you can create a `person_group`, add persons, and register faces to these persons. Then, you can use the `identify` method to match detected faces with registered individuals:

```
person_group_id = 'your_person_group_id'
face_ids = [face.face_id for face in detected_faces]
results = face_client.identify_from_large_person_group(
    face_ids=face_ids, large_person_group_id=person_group_id
)
face_client = FaceAdministrationClient(ENDPOINT, AzureKeyCredential(KEY))
```

```
for result in results:
    print(f"Face ID: {result.face_id}")
    if not result.candidates:
        print("No person identified for the face.")
        continue
    top_candidate = result.candidates[0]
    person_id = top_candidate.person_id
    confidence = top_candidate.confidence
    person = face_client.large_person_group.get_person(
        person_group_id,
        top.person_id,
    )
    print(
        f"Person identified: {person.name} "
        f"with confidence {confidence:.2f}"
    )
```

8. Run your Python script and analyze the output to review the detected faces' attributes and, if applicable, the identification results.
9. Remember to clean up any resources you've created in the Azure portal to avoid unnecessary charges.

And with that, you've implemented facial recognition in your application!

Custom Vision and Object Detection

The Azure AI Custom Vision service provides powerful tools for creating custom image classification and object detection models that are tailored to specific use cases. These capabilities are essential for developing applications that require precise image analysis, such as identifying objects in a photo, recognizing brand logos, and detecting defects in manufacturing processes. By leveraging this service, developers can build robust and accurate models that enhance automation and decision making in various industries.

Building Custom Image Classification Models

Building custom image classification models involves several key steps. Initially, you need to gather and label a substantial dataset of images that are relevant to the task at hand. You'll use these images to train the model to accurately recognize and classify different categories—and Azure AI provides a user-friendly interface you can use to upload and label images, which makes this process straightforward.

Data collection and preparation

The foundation of a successful image classification model lies in the quality and diversity of the training dataset. Therefore, when gathering images, you must ensure

that the dataset is representative of all categories that the model needs to recognize. You should start with at least 50 images per label for initial prototyping. However, to improve the robustness of the model, especially in real-world applications, we recommend that you use a larger and more varied dataset. Azure AI Custom Vision facilitates the uploading and labeling of images. The system is designed to handle both multiclass (with a single label per image) and multilabel (with multiple labels per image) classifications, making it versatile for different use cases.

Once you've prepared the dataset, the next step is to use the Custom Vision service to create a new project that will define the scope and objectives of the model. During the training phase, the model learns to identify patterns and features within the labeled images. Azure's advanced machine learning algorithms facilitate this learning process by ensuring that the model improves its accuracy over time. Then, after training, you can test the model with new images to evaluate its performance. Azure AI offers tools that you can use to refine and optimize the model based on these tests, enhancing its reliability and success for real-world applications.

When classes in your dataset are unevenly represented, you can use techniques such as oversampling minority classes and undersampling majority classes to mitigate bias. Azure AI Custom Vision also supports data augmentation—such as random cropping, flipping, and rotation—to artificially expand your dataset. The following is a minimal Python snippet that illustrates a simple augmentation approach outside Custom Vision (in this case, using the Pillow library):

```
from PIL import Image, ImageOps
import os
def augment_image(image_path, output_dir):
    img = Image.open(image_path)
    flipped = ImageOps.mirror(img)
    rotated = img.rotate(15, expand=True)
    basename = os.path.splitext(os.path.basename(image_path))[0]
    flipped.save(os.path.join(output_dir, f"{basename}_flip.jpg"))
    rotated.save(os.path.join(output_dir, f"{basename}_rotate.jpg"))
```

By applying augmentation, you can increase dataset diversity and significantly improve model generalization. You should also monitor class distributions after augmentation to ensure balanced coverage.

Use a dedicated validation set or cross-validation to assess model performance during training. For cross-validation, you partition your dataset into *K folds*, train on $K-1$ folds, and validate on the remaining fold. You then cycle through all the other folds in the same manner. This approach gives a more robust estimate of your model's performance and helps detect overfitting or class-specific weaknesses.

Each training run (iteration) in Custom Vision can produce a distinct model version, so you should tag these versions with meaningful iteration names and track their performance metrics (accuracy, precision, and recall) in a version control system or

Azure DevOps. For deployment, consider using separate staging and production endpoints. Finally, to ensure that there's minimal disruption, test new model versions in staging before promoting them to production.

What follows is a simple code snippet that polls the Custom Vision training status and logs metrics to Azure Application Insights:

```
import logging
from azure.cognitiveservices.vision.customvision.training import (
    CustomVisionTrainingClient
)

logging.basicConfig(level=logging.INFO)

def monitor_training(project_id, iteration_id):
    iteration = trainer.get_iteration(project_id, iteration_id)
    logging.info(f"Iteration status: {iteration.status}")
    logging.info(
        f"Precision: {iteration.precision}, "
        f"Recall: {iteration.recall}, "
        f"mAP: {iteration.average_precision}"
    )
```

You can schedule this monitoring in a pipeline or cron job, and you can store the iteration metrics in Azure Storage or a database for long-term tracking. Resource optimization strategies include leveraging GPU-based compute in Azure ML for faster training times, especially for large datasets.

Choosing the appropriate model

Azure AI Custom Vision allows you to train three types of models:

Image classification models
: These categorize images into predefined classes. They are ideal for tasks where the goal is to identify the overall content or subject of an image, such as distinguishing between different types of plants, animals, or products. These models are beneficial in scenarios where understanding the general category of the image is sufficient. To improve the model's accuracy and generalization, ensure that the training dataset is diverse and representative of all possible categories.

Object detection models
: These go beyond classification by identifying and locating multiple objects within a single image. These models not only classify objects but also provide their positions using bounding boxes, making them ideal for tasks that require detailed analysis of complex scenes with multiple items. Object detection is particularly useful in applications that require precise localization, such as surveillance, where identifying and tracking multiple objects is necessary, and manufacturing, where detecting defects or anomalies is critical.

Product recognition models
: These specialize in identifying and categorizing products within images, making them highly valuable for retail and ecommerce applications. The models can recognize various products, even with subtle differences, enabling efficient inventory management and enhancing the shopping experience by providing detailed product information. Product recognition models can also help with automating checkout processes, reducing manual intervention, and improving accuracy in product listings.

When deciding whether to use an image classification, object detection, or product recognition model, consider the expected inference latencies and memory usage of each. For example, object detection typically requires more compute resources and can exhibit 20–30% slower inference than simple classification on comparable hardware. Product recognition models can also incur additional overhead if they rely on large catalogs or advanced feature embeddings.

Accuracy trade-offs and decision frameworks

The type of model you should select depends on the specific requirements of your use case. If your image typically contain only one major item and do not require bounding boxes, then an image classification model will be the simplest and fastest solution. However, if you need to identify multiple objects within an image and also define their locations, an object detection model will be the best choice. In general, while image classification is faster and requires fewer computational resources, object recognition provides greater detail at the cost of higher compute power. In retail and ecommerce scenarios, where precise product matching is essential, a product recognition model will offer you a more specialized solution. Though highly effective, these models often require extensive labeled databases of SKUs or brand-specific data, which means it will take longer for you to prepare the dataset.

Performance benchmarks for different types of hardware

The hardware you select will significantly impact the performance of your model. CPU-based systems can manage simple classification in real time, but they struggle with object detection at high resolutions due to the increased computational demands. GPU-based systems (such as those using Azure NC-series instances) are two to five times faster than CPU-based systems at object detection tasks, which makes them ideal for real-time applications or large-scale batch processing. Edge devices, particularly ARM-based hardware, can run pretrained classification models with reduced precision. Performance will be lower (by about 20–40%) compared to GPUs, but they remain suitable for low-latency, localized processing in situations where cloud connectivity is limited.

Model optimization techniques

There are several techniques you can use to optimize models for efficiency:

Quantization
: This technique reduces precision from FP32 to FP16 or INT8, thus lowering inference latency on GPUs while maintaining reasonable accuracy.

Pruning
: This technique eliminates unused neurons to create a smaller model, which can improve speed but may slightly degrade accuracy.

Knowledge distillation
: This powerful technique involves a smaller "student" model learning from a larger "teacher" model to balance resource efficiency and performance.

You can apply these optimization techniques to tailor your models to the available hardware while ensuring acceptable accuracy for real-world applications.

Labeling images

Labeling your images appropriately and accurately for training will significantly boost the performance of your model. The quality of a model depends heavily on the accuracy of labeled data and how well balanced the dataset is across different categories.

Within Vision Studio, you can create an Azure Machine Learning data labeling project where you add categories to images and objects. You'll should label at least three to five images in each category. You can then use ML-assisted labeling to leverage the labels that you have provided; the app will attempt to label the remaining images automatically, and you can review and correct these as needed.

Implementing Object Detection

To implement object detection with Azure AI Custom Vision, you need to complete several steps, including setting up the environment, uploading and tagging images, training the model, and evaluating its performance. This process allows you to build robust models that can accurately identify and locate objects within images. Here, we'll focus on training and evaluating the model, then publishing it. You'll walk through a hands-on example in the next section.

Training the custom image model

To train a custom image model, you need to prepare a dataset by collecting and labeling images. This dataset should be diverse and representative of the different scenarios the model will encounter. The Azure AI Custom Vision service provides an intuitive platform for uploading and tagging images. Applying tags also helps the

model learn to identify specific objects within the images, which is crucial for the appropriate training.

Once you've completed these steps, you can initiate the training process. You do this by selecting a domain that optimizes the model for specific tasks, such as general object detection, logo recognition, or detecting products on shelves. The training process uses the labeled images to create a model that can accurately detect and classify objects. Training usually takes a few minutes, though the duration can vary depending on the size and complexity of the dataset. During training, the system continuously improves by learning the features and patterns associated with the tagged objects.

Evaluating the custom vision model metrics

You need to evaluate the performance of your Custom Vision model to ensure that it's accurate and reliable. can use various Table 5-5 lists the metrics available for assessing your model.

Table 5-5. Metrics for assessing a Custom Vision model

Metrics	Descriptions
Accuracy	This metric indicates how often the model's predictions are correct. High accuracy means the model correctly identifies the objects in the images most of the time.
Precision and recall	*Precision* measures the proportion of true positive predictions relative to all positive predictions made by the model, while *recall* measures the proportion of true positive predictions relative to all actual positives in the dataset. High precision indicates that the model makes fewer false-positive errors, and high recall indicates that it captures most of the true positives.
F1 score	This is the harmonic mean of precision and recall. It provides a single metric that balances both aspects and is particularly useful when the class distribution is imbalanced.
Confusion matrix	This matrix provides a detailed breakdown of the model's performance by showing actual versus predicted classifications. It can help you identify specific areas where the model may be making errors.
ROC curve and AUC	The *receiver operating characteristic* (ROC) curve plots the true positive rate against the false positive rate at various threshold settings, while the *area under the curve* (AUC) represents the model's ability to discriminate between positive and negative classes.
Model loss	This metric measures the difference between the predicted and actual values. Lower loss indicates a better-performing model.

By evaluating the relevant metrics, developers can identify strengths and weaknesses in a model and make the necessary adjustments to improve its performance. This iterative process of training, testing, and refining is essential for developing a successful and reliable object detection model.

Publishing and consuming a custom vision model

After ensuring that your model performs well on the validation set, you can move on to publishing it. Note that the model must be associated with a prediction resource, because that's what your applications will use to query the model.

Once it's ready, you can publish your model from the Performance tab in your Custom Vision project by selecting the Publish option and providing a name for the iteration. After publishing it, be sure to link the model to your prediction resource. Finally, take note of the endpoint URL and authentication keys that are provided. You'll use these for authentication when consuming requests.

You can integrate the model into your application by making HTTP requests to the Azure AI Custom Vision endpoint or using an SDK (such as the Python SDK). Azure's monitoring tools will track the usage and performance of the model in production while you continue to iteratively develop and improve it. You can also manage model versions by controlling the published iterations.

Practical: Creating a Custom Vision Object Detection Solution

This practical exercise will equip you with hands-on experience in setting up, training, and deploying a model that can detect and categorize different types of objects with Azure's powerful AI tools. Object detection goes beyond mere classification by not only identifying objects within an image but also pinpointing their exact locations. This capability has important applications in various industries, from automated quality control in manufacturing to inventory management to enhancing the precision of surveillance systems. By developing a custom object detection model, you can tailor your solution to recognize and differentiate between specific types of objects that are relevant to your use case, thereby improving the accuracy and efficiency of automated processes.

Start by defining a Custom Vision object detection solution. Follow these steps:

1. In the Azure portal, search for "Custom Vision" and select the service from the search results.
2. Create a new Custom Vision resource and fill in the required fields (see Figure 5-2).
3. On the Network tab, allow all networks, including the internet, to access the resource.
4. Review the terms and your configurations and create the resource if everything looks good to you.
5. Navigate to your resource group and locate the Custom Vision resource you just created. Note that your training key and endpoint will be required in step 8.

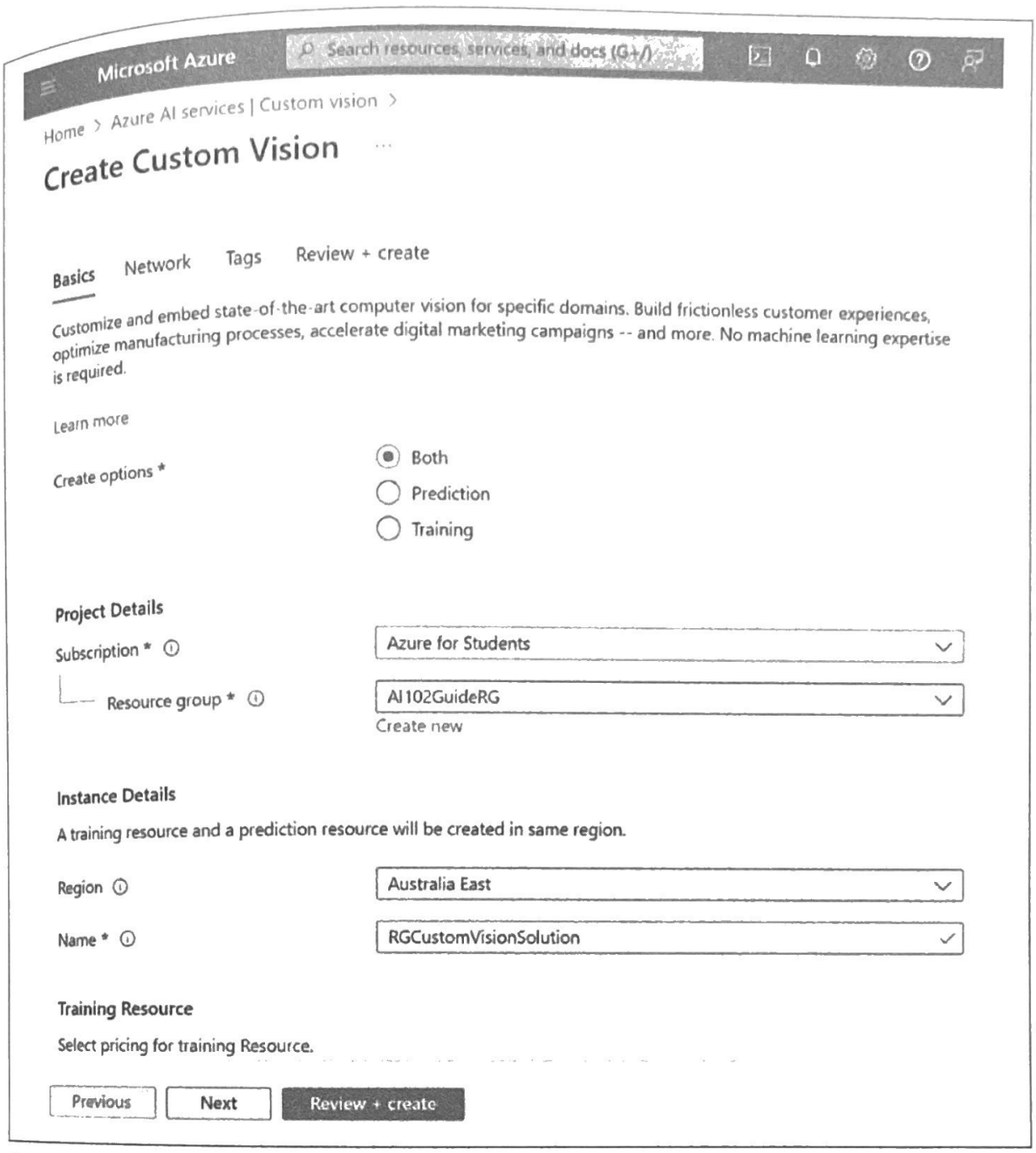

Figure 5-2. Creating a Custom Vision resource

6. In the "Keys and Endpoint" section, you'll find your training key and endpoint URL. Make a note of these values.
7. Use `pip` to install the Azure Cognitive Services Custom Vision SDK in your Python environment:

```
pip install azure-cognitiveservices-vision-customvision
```

8. Create a new script called *custom_vision_object_detection.py*. Begin by using the following code to import the required dependencies:

```
from azure.cognitiveservices.vision.customvision.training import (
    CustomVisionTrainingClient
)
```

```
from azure.cognitiveservices.vision.customvision.training.models import (
    ImageFileCreateBatch,
    ImageFileCreateEntry,
    Region
)
from msrest.authentication import ApiKeyCredentials
import time
```

9. Initialize the training client: use the `CustomVisionTrainingClient` class from the SDK and authenticate with your training key and endpoint. Replace the placeholders here with your actual values:

```
credentials = ApiKeyCredentials(
    in_headers={
        "Training-key": "your_training_key"
    }
)
trainer = CustomVisionTrainingClient("your_endpoint", credentials)
```

10. Use the `trainer.get_domains` method to retrieve a list of available domains. Each domain is optimized for different types of projects, such as classification or object detection. Loop through the domains to find the one that matches object detection. The domain you're looking for will have a type property set to `ObjectDetection`:

```
domains = trainer.get_domains()
obj_detection_domain = next(
    domain
    for domain in domains
    if domain.type == "ObjectDetection"
)
```

11. Create a new project:

```
project = trainer.create_project(
    "My Object Detection Project",
    domain_id=obj_detection_domain.id
)
```

12. Define the tags that represent the object categories you want to train your model to recognize:

```
fork_tag = trainer.create_tag(project.id, "fork")
scissors_tag = trainer.create_tag(project.id, "scissors")
```

13. Initialize an empty `image_list` array, then loop through the image files to read and append them to the list for training:

```
image_list = []
for image_num in range(1, 31):
    file_name = f"image_{image_num}.jpg"
    with open(f"flowers/{file_name}", "rb") as image_contents:
        image_list.append(
            ImageFileCreateEntry(
```

```
                name=file_name,
                contents=image_contents.read(),
                regions=[
                    Region(
                        tag_id=fork_tag.id, left=0.1, top=0.1,
                        width=0.8, height=0.8
                    )
                ]
            )
        )

    file_name = f"image_{image_num}.jpg"
    with open(f"flowers/{file_name}", "rb") as image_contents:
        image_list.append(
            ImageFileCreateEntry(
                name=file_name,
                contents=image_contents.read(),
                regions=[
                    Region(
                        tag_id=scissors_tag.id, left=0.1, top=0.1,
                        width=0.8, height=0.8
                    )
                ]
            )
        )
```

While training a model in Azure AI Custom Vision, it's recommended that you use at least 30 images per tag in the initial training set. This helps improve training quality and leads to better classification results. For more details on training image requirements, visit the "What is Custom Vision" page (*https://oreil.ly/-ewgE*) in the official Azure documentation.

14. Use the `ImageFileCreateBatch` object to create a batch of images before uploading them. This involves grouping your `image_list` into an `ImageFileCreateBatch` and then passing it to the `create_images_from_files` method:

```
batch = ImageFileCreateBatch(images=image_list)
upload_result = trainer.create_images_from_files(project.id, batch=batch)
```

15. Train the model:

```
print("Training...")
iteration = trainer.train_project(project.id)
while (iteration.status != "Completed"):
    iteration = trainer.get_iteration(project.id, iteration.id)
    print("Training status: " + iteration.status)
    time.sleep(1)
```

16. Publish the model:

```
trainer.publish_iteration(
    project.id,
    iteration.id,
    "myModel",
    "YOUR_SUBSCRIPTION_ID"
)

print(
    "Model trained and published. "
    "You can now use it to predict objects in new images."
)
```

17. View the Custom Vision projects (*https://oreil.ly/njDe_*), and locate your project (see Figure 5-3).

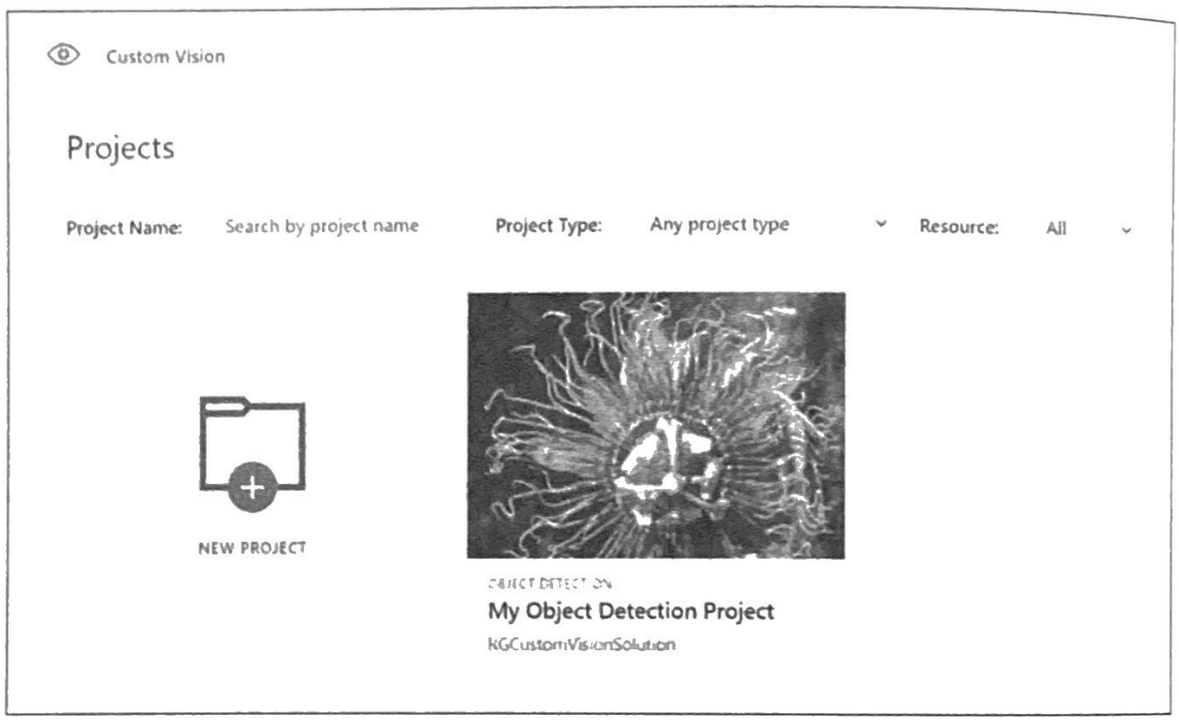

Figure 5-3. Finding your Custom Vision projects in the Custom Vision portal

18. You can now make predictions with your model, either through the CLI or via the portal. Navigate to the Predictions tab to start sending images to the endpoint and see the results (see Figure 5-4).

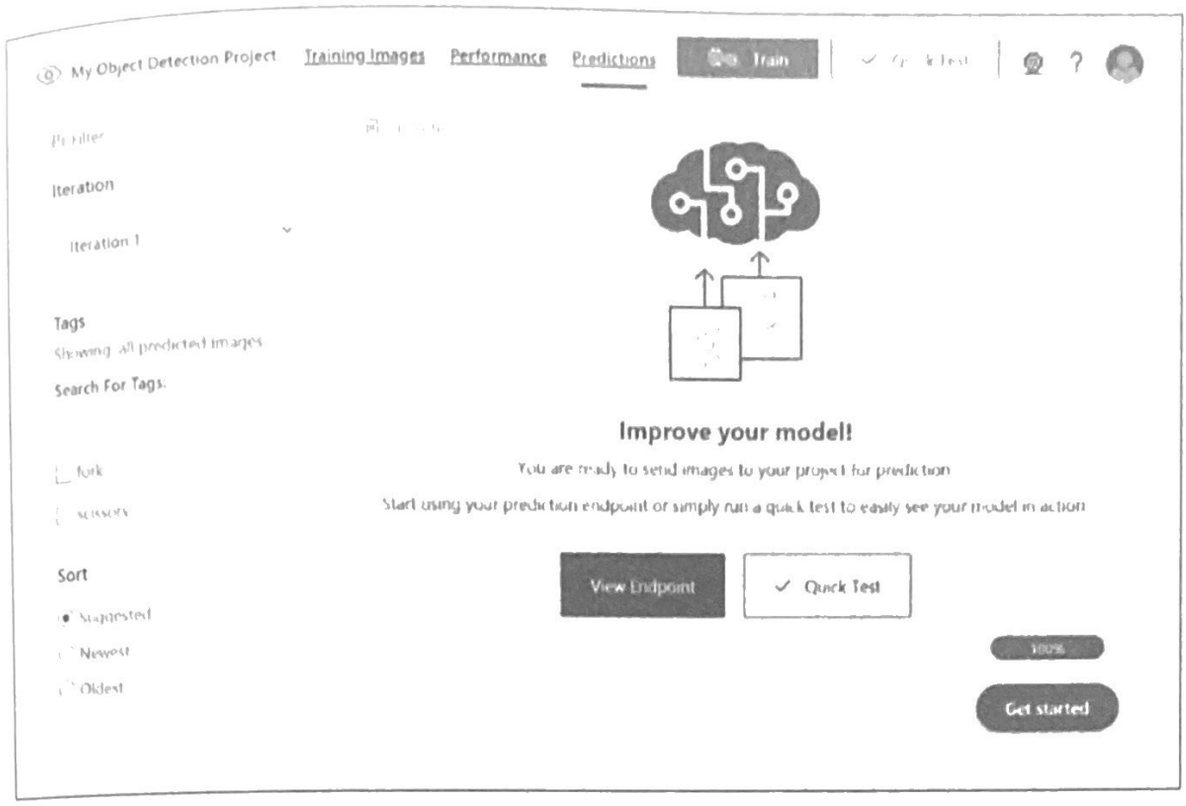

Figure 5-4. Using your prediction endpoint

And with that, you've successfully implemented your custom vision solution!

Working with Video Content

Azure AI provides powerful tools for analyzing video content that enable businesses to extract valuable insights from video data. By leveraging these tools, organizations can automate and enhance their video processing capabilities, improving efficiency and gaining deeper understanding from their video assets. Azure AI's video analysis features include detecting and identifying objects, transcribing speech, analyzing emotions, and recognizing faces. These functionalities are crucial across applications such as security, content management, and customer insights apps.

Azure AI also offers robust services to handle different aspects of video analysis, including object detection, scene segmentation, and transcription. These services automate tasks that would otherwise require extensive manual effort, such as identifying specific objects or scenes within a video, generating subtitles, and extracting meaningful data from video streams.

In this section, we will explore a powerful tool that provides many capabilities for video analysis: Azure AI Video Indexer.

Using Azure AI Video Indexer

One of the primary services for video analysis is Azure AI Video Indexer. This tool enables comprehensive analysis of video content and provides detailed insights and metadata. Video Indexer can identify and label various elements within a video, such as faces, emotions, and spoken words, and even detect scene changes and key topics discussed. Table 5-6 summarizes summarizes its capabilities.

Table 5-6. Capabilities of Azure AI Video Indexer

Capability	Description
Recognizing faces	Identifying unique individuals within an image, subject to restricted access consent
Text extraction	Interpreting written content within a video
Audio-to-text conversion	Generating written versions of spoken words in a video
Theme detection	Pinpointing principal subjects covered in a video
Emotional tone assessment	Evaluating the positivity or negativity of segments in a video
Tagging	Assigning labels to denote significant elements or concepts in a video
Content filtering	Identifying content with adult or violent themes in a video
Division of scenes	Decomposing a video into its basic scenes

You can also derive custom insights by creating models tailored to people, language, and brands. However, this capability requires limited access approval, so an application process is necessary.

Working with Azure AI Video Indexer is a great way to analyze and manage video assets. By leveraging its powerful features, organizations can automate video analysis, improve searchability accessibility, and gain valuable insights, all of which ultimately enhances the overall utility and impact of their video content.

Practical: Analyzing Video Content with Azure AI Video Indexer

In this section, we'll explore how to apply Azure AI Video Indexer in a real-world scenario, and demonstrating its powerful capabilities for analyzing and extracting insights from video content.

Start by navigating to the Azure AI Video Indexer website (*https://oreil.ly/3YCdk*). The first time you log in, a trial account will automatically be created for you, providing up to 2,400 free indexing minutes.

Follow these steps:

1. Sign in to the Azure AI Video Indexer API developer portal (*https://oreil.ly/dq1EC*). Use the same provider you used when signing up for Azure AI Video Indexer.

2. In the portal, select Products → Authorization and click Subscribe. New users are automatically subscribed to the Authorization product.
3. Access tokens are required for authentication against the Operations API. You can obtain user-level, account-level, or video-level access tokens through the Authorization API.
4. On the Azure AI Video Indexer website, select Upload to upload a sample video. Choose a video file from your system, or use a publicly accessible URL for the media file (see Figure 5-5). Make sure it's in a supported format and not larger than 2 GB for direct uploads. On the Azure AI Video Indexer website, select Upload and then choose your file source (system or URL). You should also configure the basic settings for indexing, like privacy and video source language.

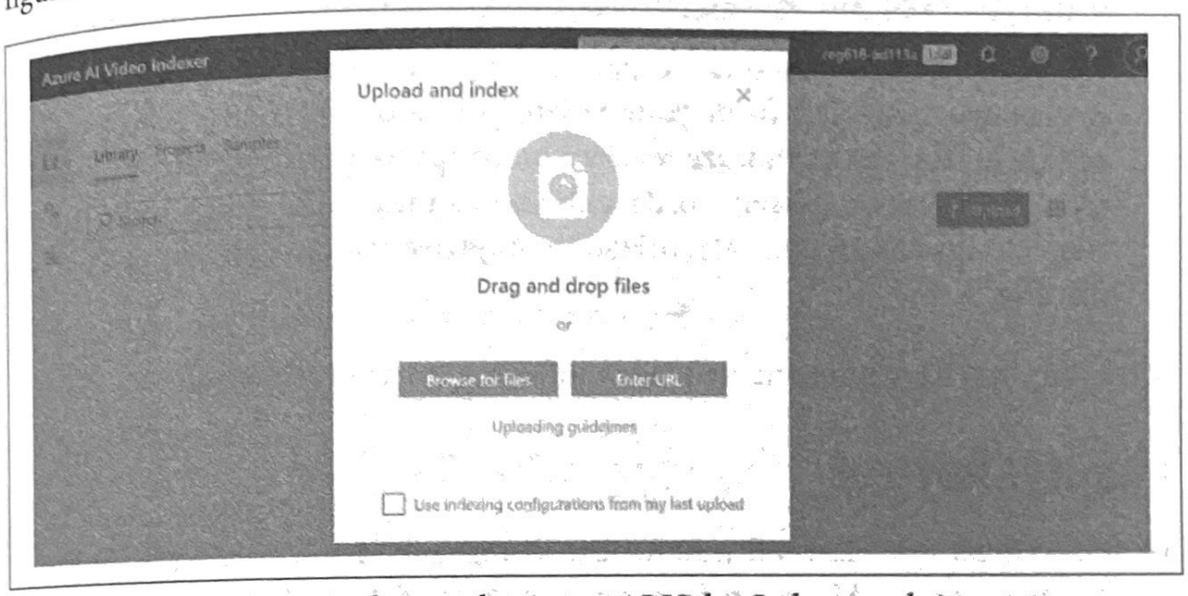

Figure 5-5. Uploading a video on the Azure AI Video Indexer website

5. After you upload the video, it will be indexed, which may take some time depending on the video length and content complexity.
6. Once the video is indexed, the API returns a detailed JSON output containing insights such as spoken words, detected faces, and emotions.
7. You can use Python to interact with the Azure AI Video Indexer API. Here's a basic Python script to start with:

```
import requests
account_id = 'your_account_id'
video_id = 'your_video_id'
api_key = os.getenv('VIDEO_INDEXER_API_KEY')
location = os.getenv('VIDEO_INDEXER_LOCATION')
token_url = (
    f"https://api.videoindexer.ai/Auth/{account_id}"
    f"/Videos/{video_id}/AccessToken"
    f"?allowEdit=false"
)
```

```
headers = {"Ocp-Apim-Subscription-Key": api_key}
access_token = requests.get(token_url, headers=headers).text.strip('"')
url = (
    f"https://api.videoindexer.ai/{account_id}/videos/{video_id}/Index"
    f"?accessToken={access_token}"
)
response = requests.get(url)
video_index = response.json()

print(video_index)
```

This script fetches the index of a specified video and provides insights into its content. You'll need to replace *your_account_id* with your actual account ID and *your_video_id* with your actual video ID, and make sure you've defined the `VIDEO_INDEXER_API_KEY` and `VIDEO_INDEXER_LOCATION` environment variables.

8. Explore the JSON response to gain a deeper understanding of the insights the system has extracted from your video, you should explore the detailed JSON response. You can customize the Python script to query specific insights, such as identified faces or spoken words, and you can use these insights in your application to enhance content categorization, improve searchability, or add accessibility features.

With that, you have implemented video indexing for your workload!

Chapter Review

In this chapter, you learned how to use computer vision solutions in Azure. You worked with Azure AI Vision to analyze images, detect objects, extract image features, retrieve text, and perform OCR. You also built custom vision models, for image classification and object detection, and you explored how to train, evaluate, and publish them. Lastly, you gained experience in video analysis.

To be successful on the exam, you need to be able to do the following things that we covered in this chapter:

- Implement image analysis solutions, including the retrieval of visual features, text, and objects.
- Implement custom computer vision models with Azure AI Vision.
- Analyze videos with AI Video Indexer.

In the next chapter, we'll look at different foundational NLP solutions and how to begin implementing them through hands-on exercises.