O'REILLY®

Building Intelligent Apps with Cognitive APIs

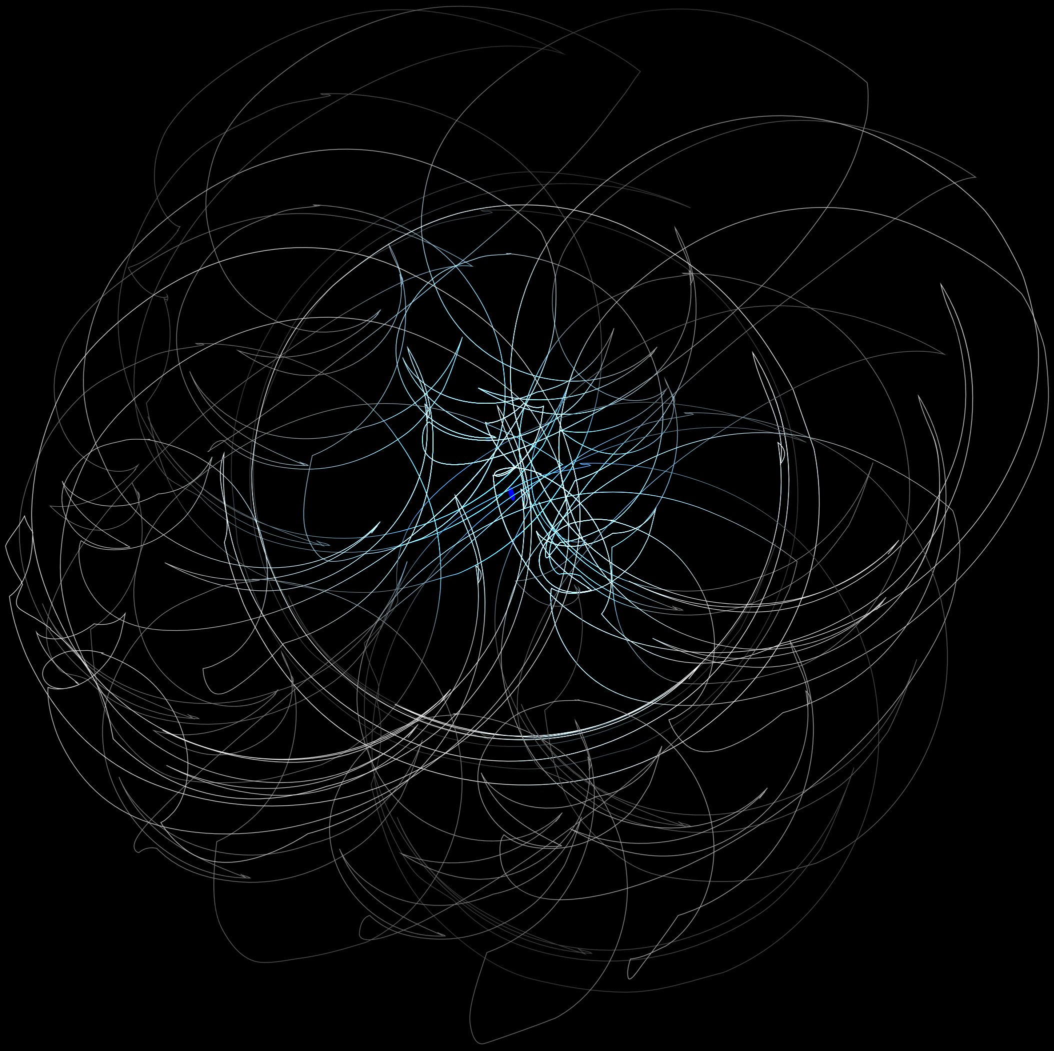
Develop Your First Intelligent Application Using Azure Cognitive Services

Anand Raman & Chris Hoder

REPORT

The future,
faster

Make your vision real. Experiment in the cloud with 12 AI services—free for 12 months with your account. Start free >



Get help with your project. Talk to a sales specialist >

Microsoft Azure

Building Intelligent Apps with Cognitive APIs

Develop Your First Intelligent Application Using Azure Cognitive Services

Anand Raman and Chris Hoder

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®

Building Intelligent Apps with Cognitive APIs
by Anand Raman and Chris Hoder

Copyright © 2020 O'Reilly Media. All rights reserved.

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://oreilly.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Acquisitions Editor: Jonathan Hassell
Development Editor: Nicole Taché
Production Editor: Deborah Baker
Copyeditor: Rachel Head
Proofreader: Gary Gorski
Interior Designer: David Futato
Cover Designer: Karen Montgomery
Illustrator: Rebecca Demarest

November 2019: First Edition

Revision History for the First Edition
2019-11-04: First Release

978-1-492-05860-1

[LSI]

Table of Contents

1. What You'll Learn in This Report. 1

2. The Microsoft AI Platform. 5
Machine Learning Services in Azure 6

3. Understanding Azure Cognitive Services. 9
How to Call a Cognitive Services API 10
New Breakthroughs as a Service 13

4. Vision. 15
Computer Vision 15
Custom Vision 30
Face 36
Form Recognizer 38
Ink Recognizer 39
Video Indexer 39

5. Speech. 43
Speech to Text 44
Text to Speech 46
Translation and Unified Speech 49
Speaker Verification and Identification 49

6. Language. 53
Text Analytics 53
Language Understanding with LUIS 59
QnA Maker 62

Spell Check 63
Translator Text 65
Immersive Reader 66

7. Decision. 69
Anomaly Detector 69
Personalizer 70
Content Moderator 71

8. Web Search. 75
Bing Web Search 75
Bing Custom Search 76
Bing Visual Search 77
Bing Autosuggest 78
Bing Video and News Search 78
Bing Entity Search 79
Bing Local Business Search 80
Azure Search and Cognitive Search 81

9. Paving the Road Ahead. 83
Schneider Electric 83
Seeing AI 84
AI Ethics and Microsoft’s Principles 85

10. Where to Go Next. 89

CHAPTER 1

What You'll Learn in This Report

Infusing AI into an application can be difficult and time-consuming. Until recently, you needed both a deep understanding of machine learning and months of development to acquire data, train models, and deploy them at scale. Even then, success was not guaranteed. The path was filled with blockers, gotchas, and pitfalls causing teams to fail to realize value from their AI investments.

Microsoft Azure Cognitive Services remove these challenges. They allow you to infuse your apps, websites, and bots with intelligence using just a few simple lines of code and without a large data science team. With these services you can quickly create applications that hear, speak, understand, and even begin to reason. These capabilities will unlock new experiences and applications for your business. Imagine an app that allows users to take a photograph of a menu and then automatically translate it into another language and retrieve reviews, pictures, and relevant recipes. Or a chatbot that can talk with your users in a customized voice that matches your brand. Imagine infusing this chatbot with even more intelligence, such as being able to recognize pictures of your products, identify any defects, and streamline the return process. These are just a few of the AI-powered features companies are building using the Cognitive Services.

Even companies with deep expertise in AI turn to these services rather than creating their own. When Uber looked at ways to verify a driver's identity—even if they'd recently cut their hair or changed their glasses—they chose to use the Microsoft Azure Cognitive Services Face API rather than build their own solution, despite their deep knowledge and use of machine learning across the company.

There are several intelligent APIs to choose from: Apple, Amazon, Google, and Microsoft all have offerings. In this report, however, we'll show you how to work with the Microsoft Azure Cognitive Services to quickly add intelligence to your applications. We will look at a wide range of scenarios, from kiosks that can greet visitors and recommend products to ways to gather real-time insights about crowd behavior or monitor driver safety on the road. Along the way, we will provide C# code snippets showing you how to call a few of the APIs.[1] We assume you have a basic understanding of the Azure platform. If you are completely new to Azure, you can find training materials on the website. We are constantly updating and improving our services, so be sure to check the documentation to learn more about the latest features and functionality for each service.

Here's a breakdown of what we cover in this report:

Chapter 2, The Microsoft AI Platform
: Cognitive Services is just one part of Microsoft's AI platform, which also includes frameworks, tools, infrastructure, and services for developing AI applications and bringing them to familiar systems like SQL Server and Power BI.

Chapter 3, Understanding Azure Cognitive Services
: Today, there are more than 20 Cognitive Services within Microsoft Azure, each with multiple features and options. These services allow you to quickly and simply bring the latest breakthroughs from research into your apps. They're divided into five categories: Vision, Speech, Language, Decision, and Web Search. We'll show you what you can achieve with each one and how to build them into your apps.

Chapter 4, Vision
: Want to analyze an image or a video? The various Vision services provide a powerful tool for extracting data from images. Recognizing and describing faces, objects, and text are just some of the many features they offer. You get the power of a

1 The authors would like to thank Winona Azure for creating most of the code snippets for this report.

fully trained deep learning image recognition model and can even customize it to recognize your specific objects.

Chapter 5, Speech
: The Speech services cover speech-to-text, text-to-speech, and real-time translation across several languages. You can customize speech models for specific acoustic environments, like a factory floor, or train the service to recognize and pronounce your business's unique jargon.

Chapter 6, Language
: The Language services enable you to analyze, understand, and translate text. You can turn your FAQ into an interactive chatbot with the QnA Maker, extract sentiment and key phrases using Text Analytics, or understand the meaning of a user's comment using the Language Understanding service.

Chapter 7, Decision
: With the Decision services, you can build apps that surface recommendations for informed and efficient decision making. You can use the Personalizer service to provide relevant, engaging, and unique experiences to every user, improving app satisfaction, usability, and engagement, and you can quickly identify problems in time series data using the Anomaly Detector service.

Chapter 8, Web Search
: Whether you want to search for an image or use an image to do a search, you can use the Bing Search APIs to bring the power of Microsoft's Bing search engine to your app.

Chapter 9, Paving the Road Ahead
: In this chapter we will provide some examples of customers leveraging the Cognitive Services to add intelligence to their offerings and transform the way they do business.

Chapter 10, Where to Go Next
: Finally, we provide some pointers to resources you can turn to if you'd like to broaden your understanding.

CHAPTER 2

The Microsoft AI Platform

Microsoft uses AI broadly in its own products and services, and makes its learnings, techniques, and products available to developers through a variety of services, infrastructure, and tools. It all starts with Microsoft Research, which has been navigating the cutting edge of AI for over 25 years. Every day, thousands of researchers explore new ideas, tackle unsolved challenges, and develop innovative techniques that repeatedly set new records. These records range from designing ResNet (the algorithm that now underpins many image recognition systems) to matching human abilities in translation and in understanding images, speech, text, and questions.

The techniques developed by Microsoft Research already drive features in Microsoft tools like Windows and Office. These AI-powered features range from protecting your account from attack to suggesting the best layout for your PowerPoint slides. The techniques and services are also made available as tools for all types of users, ranging from data scientists looking to boost their productivity or take advantage of Azure's cloud scale to developers who want to quickly infuse intelligent capabilities into their solutions.

Microsoft has also added AI into familiar products. For example, SQL Server Machine Learning Services offers an analytics engine that supports R and Python libraries inside SQL Server so developers can use machine learning like any other database functions they're writing. Running machine learning models where the data resides delivers the lowest latency and highest performance because the data doesn't have to move around.

Figure 2-1 summarizes the machine learning tools available to developers and data scientists in Azure. Microsoft provides a comprehensive set of capabilities, from the hardware to SaaS offerings.

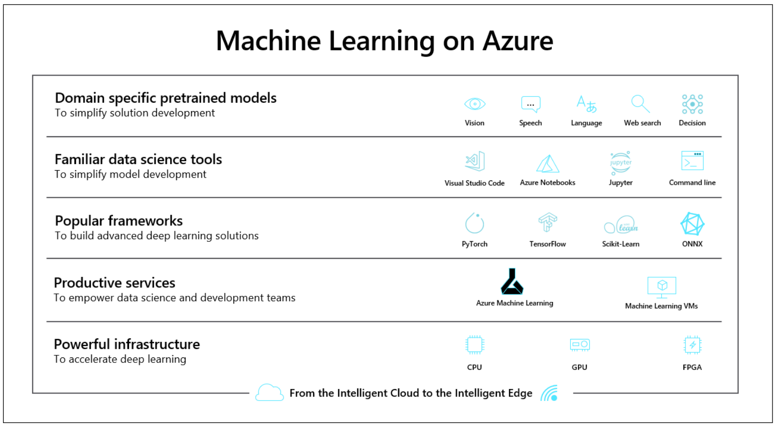

Machine Learning on Azure
Domain specific pretrained models
To simplify solution development
Vision
Speech
Language
Web search
Decision
Familiar data science tools
To simplify model development
Visual Studio Code
Azure Notebooks
Jupyter
Command line
Popular frameworks
To build advanced deep learning solutions
PyTorch
TensorFlow
Scikit-Learn
ONNX
Productive services
To empower data science and development teams
Azure Machine Learning
Machine Learning VMs
Powerful infrastructure
To accelerate deep learning
CPU
GPU
FPGA
From the Intelligent Cloud to the Intelligent Edge

Figure 2-1. A high-level overview of Microsoft's cloud AI offerings

The aim of the Microsoft AI Platform is to bring AI to every developer, empowering developers to innovate and accelerate with a variety of services, infrastructure, and tools. From Azure Cognitive Services to Azure Machine Learning (AML) for building custom AI models, the Microsoft AI Platform meets developers where they are and lets them use the tool and language of their choice. To help you get started, you can leverage the resources that are available on the Azure AI website. In the next section, we explore how you can develop your next intelligent application using the Microsoft AI Platform.

Machine Learning Services in Azure

Microsoft offers several machine learning cloud services. Each product targets a different level of expertise and desired way of working. The first choice you have to make is whether you are interested in building your own model or prefer to leverage prebuilt ones provided by Microsoft. In this report, we assume you want to use prebuilt models.

For highly custom use cases, you may need more control over the models. For these scenarios, developers should look at the Azure Machine Learning services. AML is a managed cloud service that allows you to train, deploy, and manage models in the cloud or on edge devices, using Python and tools like Jupyter notebooks. You can even deploy some TensorFlow image classification and recognition models (using a variety of deep neural networks) to Microsoft's Project Brainwave FPGA hardware in Azure for inference and training, which offers high-scale throughput and low latency. Using 800 FPGAs on Azure, Microsoft was able to process 20 TB of images in just over 10 minutes, generating over 400,000 inferences a second with just 1.8 milliseconds latency—enough images to analyze the entire continental United States with a 1-meter-per-pixel resolution.

The flowchart in Figure 2-2 is a useful tool for deciding which products to use within Azure Cognitive Services.

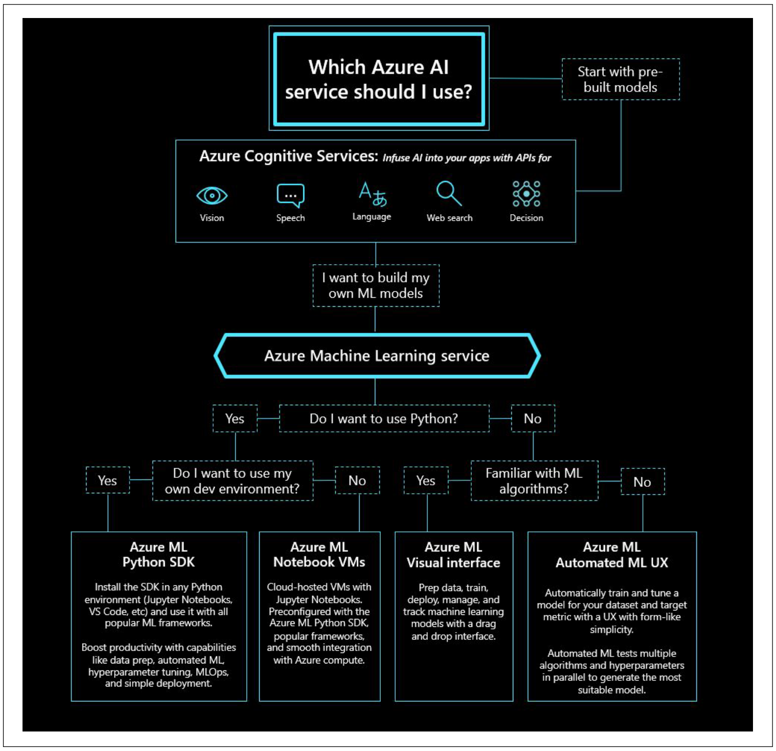

Which Azure AI service should I use?
Start with pre-built models
Azure Cognitive Services: Infuse AI into your apps with APIs for
Vision
Speech
Language
Web search
Decision
I want to build my own ML models
Azure Machine Learning service
Yes
Do I want to use Python?
No
Yes
Do I want to use my own dev environment?
No
Yes
Familiar with ML algorithms?
No
Azure ML Python SDK
Install the SDK in any Python environment (Jupyter Notebooks, VS Code, etc) and use it with all popular ML frameworks.
Boost productivity with capabilities like data prep, automated ML, hyperparameter tuning, MLOps, and simple deployment.
Azure ML Notebook VMs
Cloud-hosted VMs with Jupyter Notebooks. Preconfigured with the Azure ML Python SDK, popular frameworks, and smooth integration with Azure compute.
Azure ML Visual interface
Prep data, train, deploy, manage, and track machine learning models with a drag and drop interface.
Azure ML Automated ML UX
Automatically train and tune a model for your dataset and target metric with a UX with form-like simplicity.
Automated ML tests multiple algorithms and hyperparameters in parallel to generate the most suitable model.

Figure 2-2. A flowchart for deciding which Azure services to use for your use case

Cloud services can also be a great way for organizations to take advantage of the latest breakthroughs, using products that require little or no coding. For example, a few years ago, Microsoft Research created an AI support agent that handles support requests for

Microsoft. This agent resolves up to 40% of these interactions without needing to involve a human. The same agent is now available as a service in Dynamics 365 and is used by companies like HP and Macy's.

For organizations that do not have a team of developers but are looking to apply intelligence to their business processes, the Microsoft Power Platform provides a set of low-code tools such as Power BI, PowerApps, and Microsoft Flow that let you quickly and easily analyze data and act on it through custom applications and automated business processes. Many of these tools contain integrations with the Cognitive Services and other machine learning tools.

Power BI, for example, includes a data preparation workflow and can now use automated machine learning from the Azure Machine Learning service to allow business analysts to train, validate, and call binary prediction, classification, and regression models to analyze their data. Power BI automatically finds the most relevant features to include in the model, selects the right algorithm, tunes the model, and generates a report explaining the performance and contributing factors.

One of Microsoft's newest products, AI Builder, provides a low-code way for users to train, build, and deploy their own models within a business process without writing a single line of code. Using a simple wizard-like experience, users can perform binary classification, text classification, object detection, and data extraction from forms.

CHAPTER 3

Understanding Azure Cognitive Services

Azure Cognitive Services are designed to be productive, enterprise-ready, and trusted. They make it possible for you to build on the latest breakthroughs in AI without building and deploying your own models.

The Cognitive Services portfolio is growing fast, and services are currently grouped into five categories: Vision, Speech, Language, Decision, and Web Search. We have organized this report around these groupings, but when you are building your own app, you are likely to use several services together across multiple categories. There are no restrictions on calling different services together.

You can use Cognitive Services whether you're building traditional apps or taking the low-code approach. If you're writing serverless code, you can call services to process events. For example, you can use Microsoft Flow and the Language Understanding service to create automation that will schedule a meeting whenever you receive a text that says something like "set up a meeting." You can also build the insights from these services into your Power BI dashboards for more informative and predictive reports.

To get started, each Cognitive Service offers a free seven-day trial, or you can create an Azure account to get higher quotas and a longer trial (you get $200 in credits if you're new to Azure).

If you're building your own machine learning systems, operationalizing them for use in production can take as long as—or longer than—developing the model in the first place. The data used to train and run machine learning models can be very personal, or critical to your business, so you need to deploy those systems with strong security. Machine learning models also require ongoing maintenance. The way users interact with your systems will evolve over time, and to maintain model performance you will need to keep them up to date and retrain them regularly. All this effort can be costly and complex to manage.

The Cognitive Services manage many of these requirements for you. The services are globally available in 25 regions and backed by over 70 industry certifications. Microsoft also regularly updates the models behind each service to ensure that they stay relevant and can work with as wide a range of source materials as possible.

You can also customize the data model in many of the services. For example, if your app needs to understand common technical terms in your industry or recognize your internal product names, you can do the following:

- Create a Custom Vision model to recognize specific people, objects, or places—for example, to identify products you make or different plants in your garden.
- Teach the Language Understanding service your unique slang and regional colloquialisms that often confuse automated systems.
- Make your speech recognition system more accurate by modeling the acoustics in the environment where your app is used.
- Create custom voices so generated speech both sounds natural and matches the voice of your brand and organization.

How to Call a Cognitive Services API

There are several different ways to use the Azure Cognitive Services. The documentation provides QuickStart tutorials for getting up and running and reference material for both the SDKs and REST APIs.

There are SDKs to help you get started in many popular languages, such as C#, Python, Java, JavaScript, and Go. The SDKs enable you to call the APIs directly from your code using standard methods within the library. They also handle working with response data and formatting it in the appropriate object types. The available SDKs vary from service to service, though in most cases you'll find a C# SDK from NuGet. A complete list of Azure C# SDKs can be found on GitHub.

You can also access the Cognitive Services directly by calling their REST API URLs. REST APIs make it easy to call the services in whatever language or environment you choose.

If you prefer to work with tools like Postman to build and test API queries, Microsoft provides OpenAPI Swagger definitions for the REST APIs within the documentation.

Throughout this report, we will provide example code snippets using the C# SDKs. Each service has a different library, and you will need to include the ones you need with `using` statements at the top of your file. For example, to use the face detection feature in the Face API, you would include these statements to load the libraries:

```
using Microsoft.Azure.CognitiveServices.Vision.Face;
using Microsoft.Azure.CognitiveServices.Vision.Face.Models;
```

Or, for the Text Analytics API, you would include these statements:

```
using Microsoft.Azure.CognitiveServices.Language.TextAnalytics;
using Microsoft.Azure.CognitiveServices.Language
   .TextAnalytics.Models;
```

NOTE

To save space in the book, we will not be including the `using` statements that load the various C# libraries and SDKs used in the code. We have also omitted some code surrounding the actual SDK method calls.

You can see the full versions of the code samples included in this book on GitHub or in the QuickStarts included in the Cognitive Services documentation.

You will also need to authenticate all the requests you make to Cognitive Services by providing a subscription key or an authentication token or by using Azure Active Directory. For production applications you will need to properly secure these keys, for example by using Azure Key Vault. You can find more details on authentication for the Cognitive Services on the Microsoft Azure website.

To illustrate the structure of a typical Cognitive Services REST API call, let's look at the URL for the Analyze operation within the Computer Vision service:

```
{endpoint}/vision/v2.0/analyze[?visualFeatures]
```

In July 2019, the Cognitive Services switched to using a custom subdomain name as part of the endpoint for each Cognitive Services instance you create. With this new feature, the endpoint name is unique and based on the name you choose when you deploy a Cognitive Services resource or instance. URLs using the custom subdomain will have the following structure: *https://{Resource Name}.cognitiveservices.azure.com*. Some services, however, still rely on the older region-based endpoint names, which have the following structure: *https://{region}.api.cognitive.microsoft.com*. The region is the same one you chose when you created your resource. You can determine which endpoint to use by finding the Endpoint field when viewing the resource in the Azure Portal. Be sure to check the service's documentation to make sure you are using the correct endpoint to call the service.

The next part of the URL specifies the service you want to call, the API version, and the specific operation. In the preceding example, you would be calling version 2.0 of the Analyze operation within the Computer Vision service. APIs are versioned, and you should keep aware of Microsoft's deprecation and update policies to ensure there is no downtime in your application. The specific operation being called is followed by optional configuration details.

Not all Cognitive Services APIs have the same hierarchy, so ensure that you've read the documentation for the specific API or SDK you are using. Depending on the requirements, additional content is sent either in the request body or as request parameters.

NOTE

The APIs return data as JSON; you'll need to make sure that your code can parse the responses and deal with any errors.

Some APIs have a namespace that contains multiple operations for each different feature. For example, the Computer Vision API has operations like `/detect`, `/tag`, and `/describe`. Each operation provides a different set of features. However, since you will often want to call multiple features at once, many services provide an operation to group them together. For example, the Computer Vision API provides the `/analyze` feature, where you can use request parameters to choose the features you want to use.

Running in a hyperscale cloud data center means the Cognitive Services can handle a very large call volume. They run with strict service-level agreements (SLAs) and are guaranteed to be available at least 99.9% of the time. However, some use cases and regulations make it difficult or impractical to use a cloud service. Fortunately, many of the Cognitive Services can be exported as containers and run locally. Containers are ideal for remote environments where connectivity is slow and expensive but Internet of Things (IoT) devices are the most useful. Running models locally also addresses questions of data governance, since you do not have to worry about taking data outside your environment or data center.

New Breakthroughs as a Service

Cognitive Services covers areas where the state of the art in research is advancing rapidly, and their capabilities are updated frequently to bring that innovation to you. In some cases, developers get access to new models as quickly as the internal teams at Microsoft.

The new neural text-to-speech capabilities can generate speech that sounds almost exactly like a person speaking; that's important because research shows it's much less tiring to listen to results, directions, or something longer like an audiobook with the natural intonations of a human voice and with all the words articulated clearly.

Using deep neural networks to do voice synthesis and prosody (matching the patterns of stress and intonation in speech) together, rather than as separate steps, produces more natural and fluid speech. This is a relatively new development that was in research labs just a couple of years ago, and new research papers are still coming out with refinements. But several months before the Bing team added neural voice synthesis to their mobile app, the Cognitive Services Speech Services APIs already included a preview of two neural text-to-speech voices in English, followed by Chinese, German, and Italian voices.

Microsoft also offers an early preview of some of the more experimental capabilities through cognitive research technologies. These aren't production-grade Azure services, but instead an early look at some of the ongoing research and development at the company. At the time of writing they include the following services:

- Controlling apps using custom hand gestures
- Giving chatbots more personality by adding small talk or teaching them new behaviors by providing examples of conversations
- Creating interactive search experiences for structured data

There's no guarantee that any of the cognitive research technologies will develop into a production service, but several of them have already moved into public preview, including Ink Recognizer (for working with digital ink), Personalizer (which uses reinforcement learning to personalize the experience for each one of your users), and Anomaly Detector (which detects anomalies in time series data). You can learn more about the cognitive research technologies on the Microsoft website.

CHAPTER 4

Vision

We live in a world of objects, but identifying them in pictures today can be challenging. Digital images are represented as arrays of pixels and color values with no data describing the objects that those pixels represent. However, advancements in machine learning on images are removing this barrier by providing powerful tools for extracting meaning and information from these pixels.

The Cognitive Services Vision APIs provide operations that take image data as input and return labeled content you can use in your app, whether it's text from a menu, the expression on someone's face, or a description of what's going on in a video. These same services are used to power Bing's image search, extract optical character recognition (OCR) text from images in OneNote, and index video in Azure Streams, making them tried and tested at scale.

The Vision category includes six services: Computer Vision, Custom Vision, Face, Form Recognizer, Ink Recognizer, and Video Indexer. We will provide a brief introduction to each.

Computer Vision

Computer Vision provides tools for analyzing images enabling a long list of insights including detection of objects, faces, color composition, tags, and landmarks. Behind the APIs are a set of deep neural networks trained to perform functions like image classification, scene and activity recognition, celebrity and landmark recognition, OCR, and handwriting recognition.

Many of the computer vision tasks are provided by the Analyze Image API, which supports the most common image recognition scenarios. When you make a call to the different endpoints in the API namespace, the appropriate neural network is used to classify your image. In some cases, this may mean the image passes through more than one model, first to recognize an object and then to extract additional information.

Bundling all these features into one operation means you can make one call and accomplish many tasks. For example, using a picture of a shelf in a supermarket you can identify the packaging types on display, the brands being sold, and even whether the specific products are laid out in the right order (something that is often both time-consuming and expensive to audit manually).

The Analyze Image API attempts to detect and tag various visual features, marking detected objects with a bounding box. The tasks it performs include:

- Tagging visual features
- Detecting objects
- Detecting brands
- Categorizing images
- Describing images
- Detecting faces
- Detecting image types
- Detecting domain-specific content
- Detecting color schemes
- Generating thumbnails
- Detecting areas of interest

The process of working with an API through an SDK is much the same for every API. Using version 5 of the C# SDK, do the following:

1. Create a client, specifying your subscription key and endpoint:

```
ComputerVisionClient computerVision =
  new ComputerVisionClient(
    new ApiKeyServiceClientCredentials(
      "<Your Subscription Key>")
```

```
        )
       {Endpoint = "<Your Service Endpoint>"};
```

2. Choose the features you want to analyze in the image:

```
private static readonly List<VisualFeatureTypes>
features = new List<VisualFeatureTypes>()
{
   VisualFeatureTypes.Categories,
   VisualFeatureTypes.Description,
   VisualFeatureTypes.Faces,
   VisualFeatureTypes.ImageType,
   VisualFeatureTypes.Tags
};
```

3. Call the API:

```
ImageAnalysis analysis =
   await computerVision.AnalyzeImageAsync(
   "http://example.com/image.jpg", features
 );
```

4. Extract the response information. Here we extract the caption, but many other features are also returned:

```
Console.WriteLine(
   analysis.Description.Captions[0].Text + "\n"
);
```

Tagging Visual Features

Tagging an image is one of the most obvious uses of the Computer Vision service. This functionality provides an easy way to extract descriptors of the image that can be used later by your application. By providing many different tags for each image, you can create complex indexes for your image sets that can then be used, for example, to describe the scene depicted or find images of specific people, objects, or logos in an archive.

To use this feature, you need to upload a still image or provide a link to an image. The API returns a JSON document that contains a list of recognized objects, along with a confidence score for each. For example, an excerpt of the tags from the response for a picture of a home with a lawn (Figure 4-1) will look something like this:

```
"tags": [
    {
        "name": "tree",
        "confidence": 0.9999969005584717
        },
        {
            "name": "grass",
            "confidence": 0.9999740123748779
        }
    ]
```

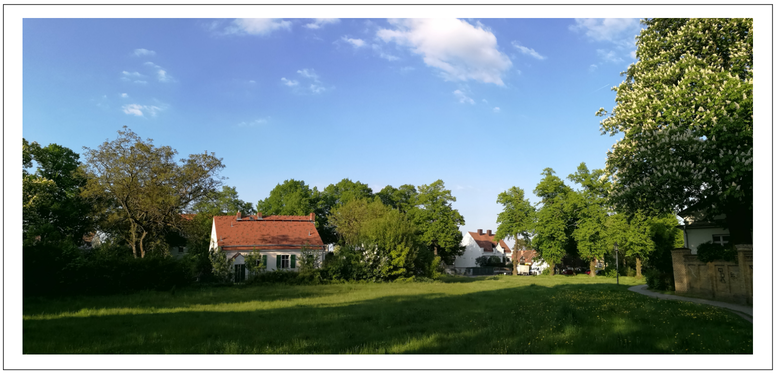


Figure 4-1. Image of a house with a lawn

The names of the objects are easy enough to extract, and you can use the confidence score as a cutoff to define when to apply a tag (or when to show the tag to your users). The threshold choice is up to you and your specific use case. We suggest using a high threshold to avoid false positives and poor matches cluttering up the tags and search results.

When you call the `/tag` endpoint, tags that could have multiple meanings may include a hint to scope the tag to a usage. When a picture of a cyclist is tagged "riding," the hint will note that the domain is sport (rather than geography, to avoid confusion with the Ridings, which are areas of Yorkshire), for example.

You may want to add code to convert the image tags into different terms that are more specific to your application before showing them to users, or at least go beyond a basic list structure.

Object Detection

Like the tagging API, the object detection API takes an image or an image URL and returns a JSON document with a list of detected objects, which in this case are accompanied by bounding box coordinates. The coordinates let you understand how objects are related. For example, you can determine if a cup is to the right or the left of a vase. You can also see how many instances of an object there are in a picture: unlike with the tagging API, which just returns "truck" even if there are multiple trucks in a picture, with the object detection API you get the location of each one. There are some limitations to be aware of, however; for example, it's not possible to detect small objects or objects that are close together.

You call the object detection API via the Analyze Image API by setting the query type to `"objects"` in the `visualFeatures` requests parameter, or via the standalone endpoint `/detect`. Here's an excerpt of the JSON response for one of the objects in Figure 4-2:

```
"objects": [
    {
        "rectangle": {
            "x": 1678,
            "y": 806,
            "w": 246,
            "h": 468
        },
        "object": "vase",
        "confidence": 0.757,
        "parent": {
            "object": "Container",
            "confidence": 0.759
        }
    },
]
```

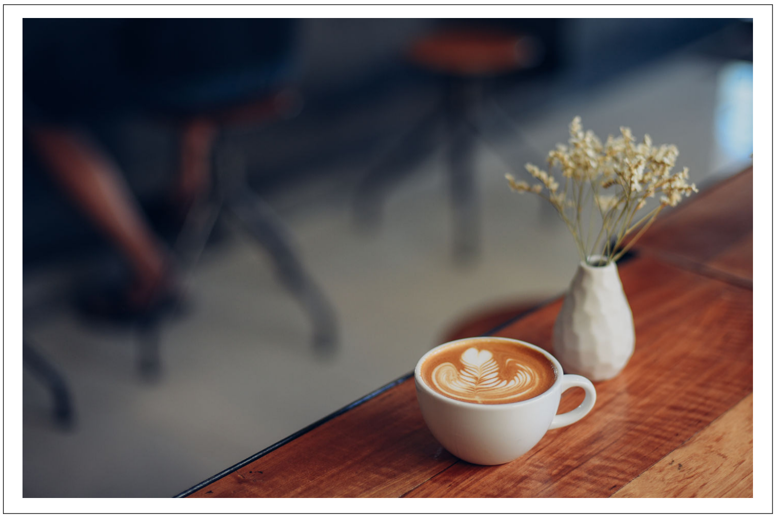


Figure 4-2. An image of a coffee cup and vase

As with the tagging API, the service returns hints to put classifications in context, in this case showing that a "vase" is a "container."

Detecting Brands

The brand detection API is a specialized version of the object detection API that has been trained on thousands of different product logos from around the world and can be used to detect brands in both still images and video.

Like the object detection API, it returns details of the brand detected and the bounding box coordinates indicating where in the image it can be found. For example, you could run both object and brand detection on an image to identify a computer on a table as a Microsoft Surface laptop.

You call the object detection API with the Analyze Image API, setting the query type to `"brands"` in the `visualFeatures` request parameter. Objects are returned in the the JSON document's "brands" block. The response for Figure 4-3 can be seen below the image.

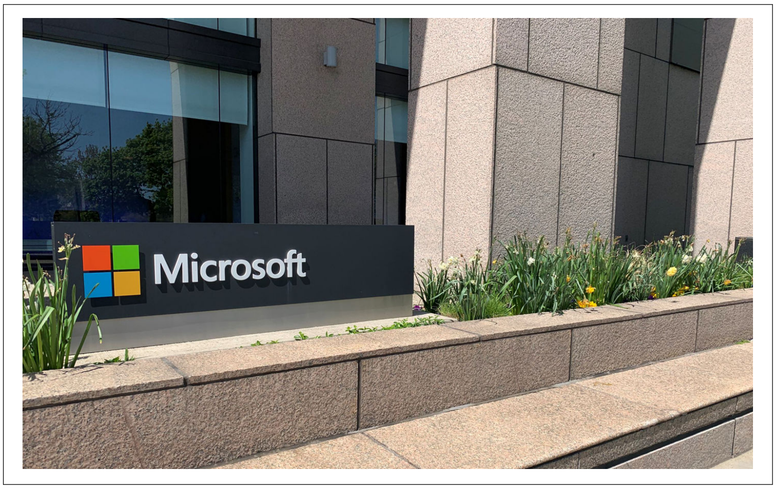


Figure 4-3. Example of a brand you may want to detect in photos

```
"brands": [
    {
        "name": "Microsoft",
        "confidence": 0.659,
        "rectangle": {
            "x": 177,
                "y": 707,
                "w": 223,
                "h": 235
            }
        }
    ]
```

Categorizing an Image

The Computer Vision API can also categorize an image. This is a high-level approach, useful for filtering a large image set to quickly determine if an image is relevant and whether you should be using more complex algorithms.

There are 86 different categories, organized in a parent/child hierarchy. For example, you can get an image category of "food_pizza" in the "food_" hierarchy. If you're building a tool to determine pizza quality to assess whether restaurant franchises are following specifications, any image that doesn't fit the category because it's not a pizza can be rejected without spending more time on it.

It's a quick and easy API to use, and one that is ideal for quickly parsing a large catalog of images, as well as for an initial filter. If you need more powerful categorization tools for images, PDFs, and other documents, consider the Cognitive Search tools covered in Chapter 7. An excerpt from the JSON response returned for the photograph of a crowd of people shown in Figure 4-4 follows the image.

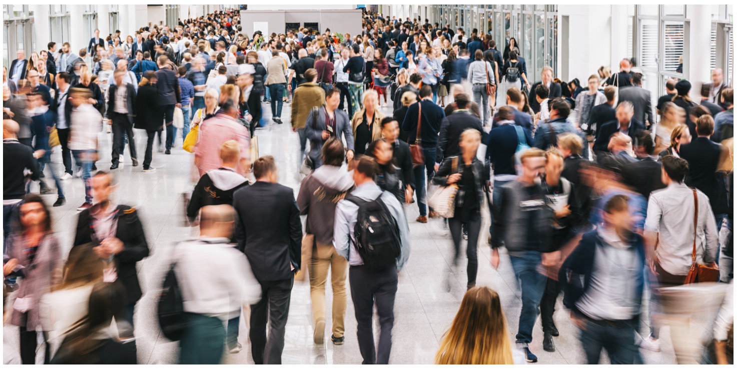


Figure 4-4. A crowd photograph you might categorize using Computer Vision

```
"categories": [
    {
        "name": "people_crowd",
        "score": 0.9453125
    }
]
```

Describing an Image

Most of the Computer Vision tools return machine-readable information, using JSON documents to deliver results that can then be processed by your code to deliver the results you need. However, you may at times need a more human-oriented response, like text that can be used as a caption. This is ideal for assistive technologies, or for providing the human-readable elements of an image catalog.

You can access the image description feature via either the `/analyze` endpoint or the standalone `/describe` endpoint. Descriptions are returned in a JSON document as a list ordered by confidence, with associated tags that can give additional context. Following is an excerpt of the response for a photograph of the New York City skyline (see Figure 4-5):

```
"description": {
  "tags": [
     "outdoor", "photo", "large", "white", "city", "building",
     "black", "sitting", "water", "big", "tall", "skyscraper",
     "old", "boat", "bird", "street", "parked", "river"
   ],
   "captions": [
      {
        "text": "a black and white photo of a large city",
        "confidence": 0.9244712774886765
      }
   ]
}
```

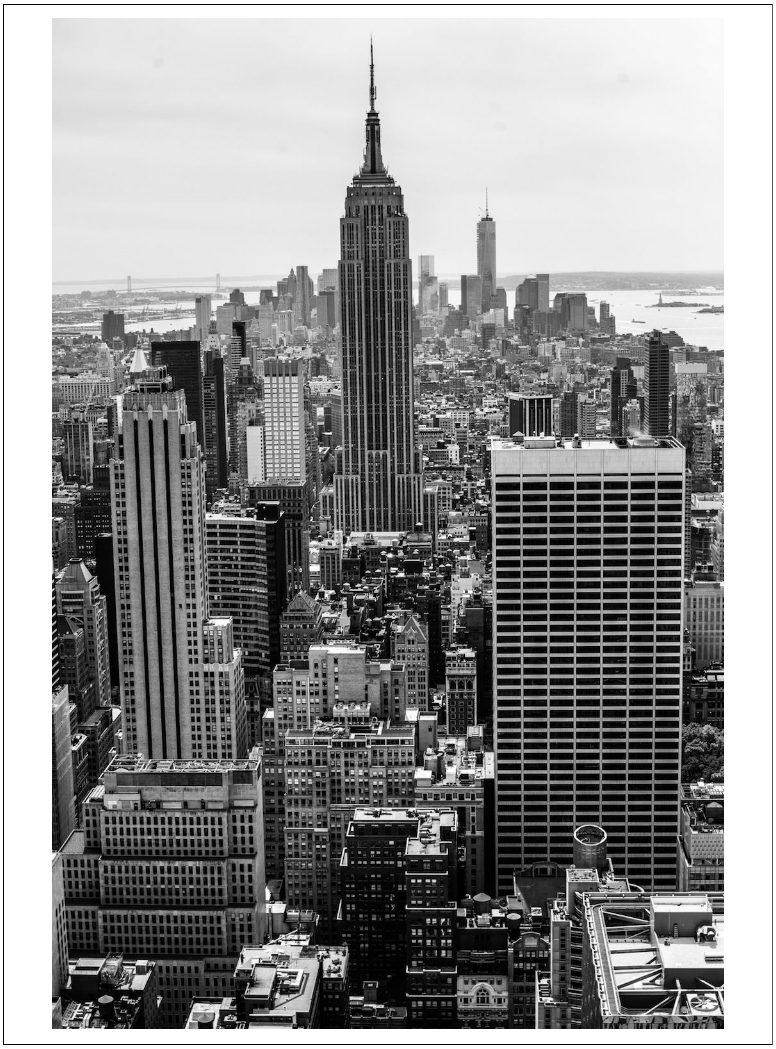


Figure 4-5. Image of the New York skyline

You can use the confidence level to have the tool automatically choose the highest-ranked description if you always want to get a single result, or you may prefer to show users multiple possible descriptions when the confidence levels are lower so that they can pick the most appropriate one manually.

Detecting Faces

While the Face API offers a suite of more powerful face recognition services, you can get quick access to basic facial analysis capabilities through the Analyze Image API. This detects the faces in an image, along with an indication of age and gender and bounding box coordinates.

Data is returned using the familiar JSON document format, with different responses for single and multiple faces. Your code will need to be able to work with responses with one or more face blocks, because images may contain multiple faces (as in Figure 4-6).

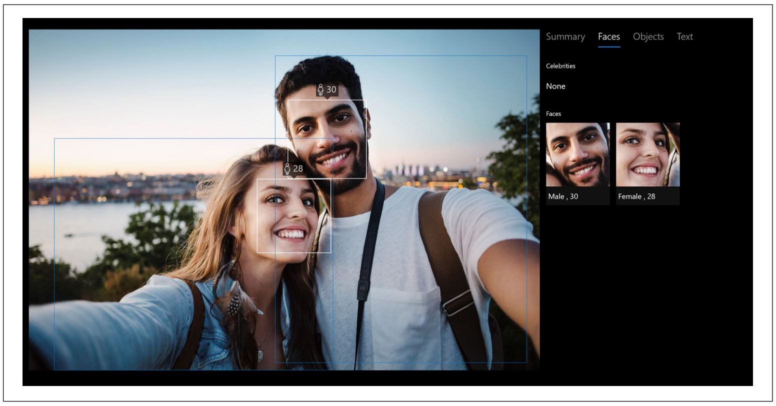


Figure 4-6. Image of a man and a woman with faces detected by bounding boxes

Here is an example of the "faces" block of the JSON response for the picture of two people in Figure 4-6:

```
"faces": [
    {
        "age": 30,
        "gender": "Male",
        "faceRectangle": {
            "left": 1074,
            "top": 292,
            "width": 328,
            "height": 328
        }
    },
    {
        "age": 28,
        "gender": "Female",
        "faceRectangle": {
                "left": 947,
                "top": 619,
                "width": 308,
                "height": 308
            }
        }
    ]
```

You may find this familiar—this API was the basis of the popular "How Old" service.

Detecting Image Types

Sometimes it's useful to be able to categorize the type of image that's being analyzed. The Analyze Image API can detect whether an image is clip art or a line drawing, returning the responses (on a simple 0 to 3 scale) in the `imageType` field. A value of 0 indicates that it's not, while a value of 3 indicates high likelihood that it is clip art or a line drawing (as in Figure 4-7).

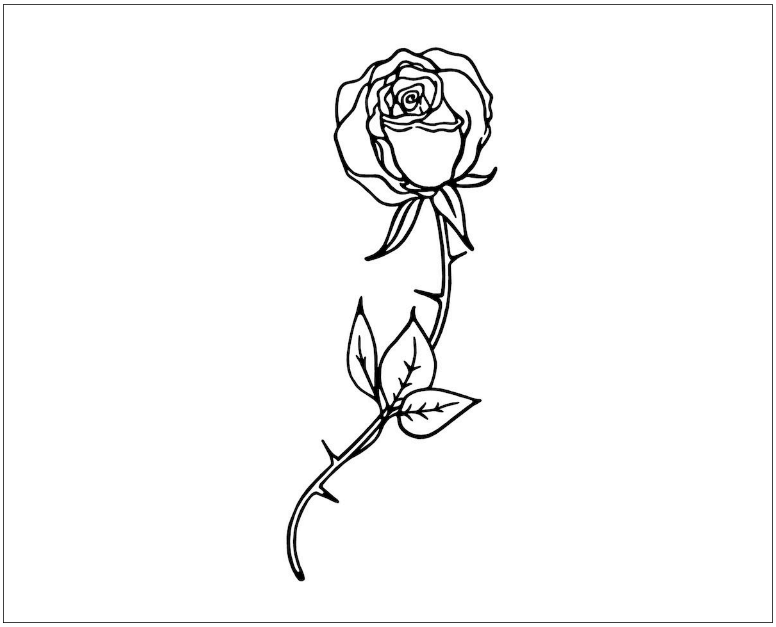


Figure 4-7. Image of a sketch of a rose

A sketch of a rose like the one in Figure 4-7 might return the following image type information in the JSON response:

```
    "imageType": {
        "clipArtType": 3,
        "lineDrawingType": 1
    }
```

To detect photographs, use the same API: a 0 return value for both image types is an indication it's neither clip art nor a line drawing.

Detecting Domain-Specific Content

While most of the Computer Vision tools are designed for general-purpose image classification, a small set of APIs are trained to work against specific image sets. Currently there are two domain-specific models available: for celebrities and for landmarks. You can use them as standalone categorization tools, or as an extension to the existing toolset.

Like the other APIs, these domain-specific models can be called by REST APIs, using the *models/<model>/analyze* URI with the Computer Vision namespace. Results are in the standard JSON document format and include a bounding box for the recognized object, the name, and a confidence level.

Detecting the Color Scheme

Image analysis isn't only useful for detecting people or objects; much of the information in an image can be used in your applications. For example, if you're looking for anomalies using computer vision, a change in color can be a useful indicator. The color scheme analysis feature in the Analyze Image API extracts the dominant foreground and background colors, as well as a set of dominant colors for an image. It also details the most vibrant color in the image as an accent color. Dominant colors are chosen from a set of 12 possibilities, while the accent is shown as an HTML color code.

The JSON response also contains a Boolean value, `isBwImg`, that is is used to indicate whether an image is in color or black and white. Here is an example excerpt of the JSON response for the sunset image in Figure 4-8:

```
    "color": {
        "dominantColorForeground": "Brown",
        "dominantColorBackground": "Black",
        "dominantColors": [
            "Brown",
            "Black"
        ],
        "accentColor": "C69405",
        "isBWImg": false
    }
```

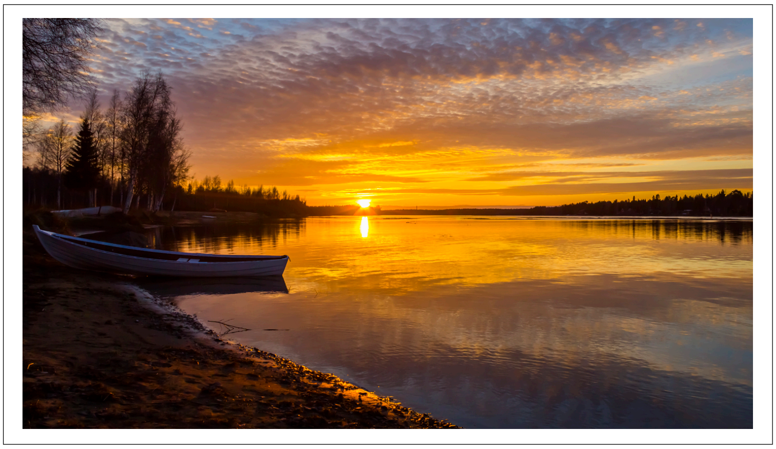


Figure 4-8. Image of a sunset on a lake

Generating a Thumbnail

Naively cropping an image or reducing the resolution to create thumbnails can lead to the loss of valuable information. The thumbnail API allows you to first identify the area of interest in the image for cropping. The result is a more useful thumbnail for your users. For example, if you start with a photograph of a hummingbird feeding at a flower, the API will generate a thumbnail showing the bird. The response from the service is the binary data of a cropped and resized image you can download and use in your applications.

Getting the Area of Interest

If you want to highlight an area of the image for further processing rather than cropping it, the area of interest API uses the same underlying algorithm as generating a thumbnail but returns the bounding box coordinates for you to work with.

Extracting Text from Images

The Computer Vision API has three different tools for handling text in images. The first, OCR, is an older model that uses synchronous recognition to extract small amounts of text from images. Using a standard uploaded image, it will recognize text that's rotated up to 40 degrees from any vertical. It can return coordinates of the bounding boxes around words, so you can reconstruct sentences. However, there are issues with partial recognition and busy images. It's best used when there's a small amount of text in an image.

A second API, Recognize Text, is in preview and being deprecated in favor of the newer, more modern Read API; however, it's still available if you need it. The Read API offers the best performance, but currently only supports English. Designed for text-heavy documents, it can recognize a range of text styles in both printed (PDF) and handwritten documents. The API follows a standard asynchronous process, which can take some time. The initial call returns the `operationLocation` that is used to construct a URL to check the retrieve recognized text.

If a process is running it will return a "running" status code. Once you get "succeeded" you will also receive a JSON object that contains the recognized text as a string along with document analytics. Each separate word has a bounding box, a rotation, and an indicator of whether the recognition has a low confidence score.

To make a call to the Read API using the C# SDK, you first need to instantiate the client:

```
ComputerVisionClient computerVision =
 new ComputerVisionClient(
  new ApiKeyServiceClientCredentials("<Your Subscription Key>"),
      new System.Net.Http.DelegatingHandler[] { }
  );
{Endpoint = "<Your Service Endpoint>"};
```

Then you can start the async process to extract the text. Here we show how to do this with an image URL, but you could also upload an image from a file:

```
const string imageUrl = "https://example.com/image.jpg";
BatchReadFileHeaders textHeaders = await
  computerVision.BatchReadFileAsync(imageUrl,
          TextRecognitionMode.Handwritten
  );
```

Since this process happens asynchronously, the service will respond with an `OperationId` that can be used to check the status of your request:

```
const int numberOfCharsInOperationId = 36;
string operationId = operationLocation.Substring(
            operationLocation.Length - numberOfCharsInOperationId
        );
    int i = 0;
    int maxRetries = 10;
    while ((result.Status == TextOperationStatusCodes.Running ||
            result.Status == TextOperationStatusCodes.NotStarted)
            && i++ < maxRetries
        )
    {
        result = await
          computerVision.GetReadOperationResultAsync(operationId);
    }
```

Once the service is done, you can display the results. Here we just print the extracted text, but other information such as the bounding box for each word may be included in the response:

```
    var recResults = result.RecognitionResults;
    foreach (TextRecognitionResult recResult in recResults)
    {
        foreach (Line line in recResult.Lines)
        {
            Console.WriteLine(line.Text);
        }
    }
```

Here is an excerpt from the JSON response for the words recognized in the image of a shopping list in Figure 4-9:

```
    {
        "boundingBox": [
            2260,
            841,
            2796,
            850,
            2796,
            994,
            2259,
            998
        ],
        "text": "Grocery"
    },
```

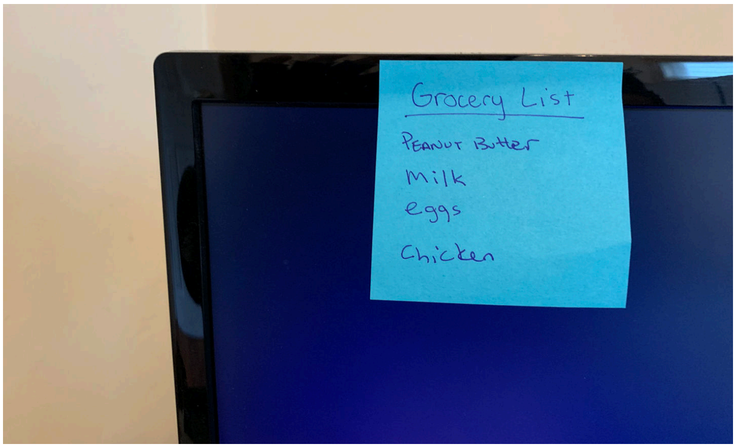


Figure 4-9. Image of a handwritten shopping list

Custom Vision

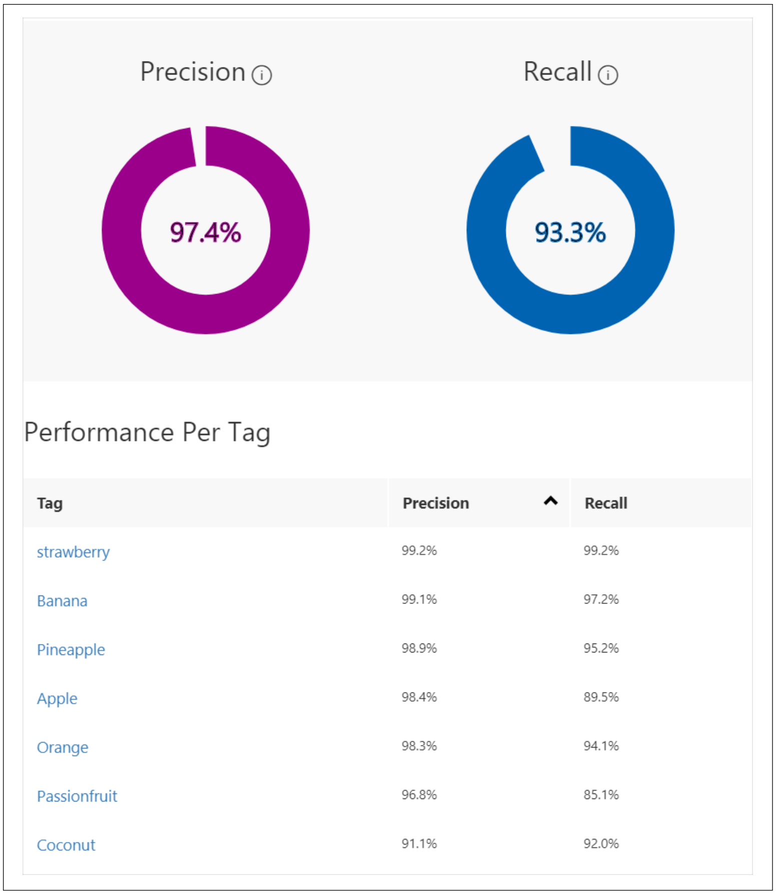
For many business-specific use cases, you may find that the general image tagging and object detection services provided by the Computer Vision API are not accurate enough. The Custom Vision API solves this problem by letting you build your own custom classifier based on a relatively small set of labeled images that show the objects, conditions, and concepts you need to recognize. For example, you can use this service for very specific use cases like identifying a circuit board that wasn't soldered correctly or distinguishing between an infected leaf and a healthy one. You can even export these models to a smartphone and give employees an app that provides real-time feedback.

Custom Vision uses a machine learning technique called *transfer learning* to fine-tune the generalized models based on the sample images you provide. This process lets you get great performance using only a small number of images (versus the millions used to train the general classifier). For the best results, your training set needs at least 30 to 50 images, ideally with a good range of camera angles, lighting, and backgrounds. These images should match how they will be captured in your production application. If the camera angle or background will be fixed, label common objects that will always be in the shot.

To get started building a Custom Vision model, you first need to choose whether you want a model for detecting objects or classifying the entire image. If your use case is particularly complex, you can create multiple models and layer them to improve discrimination in classes that are easy to confuse (like tomatoes and bell peppers or sandwiches and layer cakes). Where possible, it's important to have a similar number of images for each tag. You can also add images that you tag as "negative samples," to tell the classifier that it shouldn't match any of your tags to these types of images.

After you choose which models you are going to create, you need to provide training data or examples of the objects or classes to the service. You can create the model and upload images through the service's website, or in code via API calls.

After training (which takes only a few minutes), you can see the precision and recall performance of your model on the website. *Precision* shows what percentage of classifications are correct. To illustrate this concept, imagine if the model identified 1,000 images as bananas, but only 974 were actually pictures of bananas—that's 97.4% precision. *Recall* measures the percentage of all examples of a class that were correctly identified. For example, if you had 1,000 images of bananas but the model only identified 933, the recall would be 93.3%; if the model correctly identified 992 of the 1,000 images of bananas, then the recall would be 99.2%. Figure 4-10 shows the easy-to-read graphic in the Custom Vision portal. Here you can see a breakdown of the overall precision and recall of the model, as well as the performance per tag.



Tag	Precision	Recall
strawberry	99.2%	99.2%
Banana	99.1%	97.2%
Pineapple	98.9%	95.2%
Apple	98.4%	89.5%
Orange	98.3%	94.1%
Passionfruit	96.8%	85.1%
Coconut	91.1%	92.0%

Figure 4-10. Screenshot of a model's performance from https://www.customvision.ai/

Classifications are determined by a threshold you set on the returned probability for each class. For each image the model analyzes, it will return the predicted classifications (or objects) and a corresponding probability between 0 and 1. The probability is a measure of the model's confidence that the classification is correct. A probability of 1 means the model is very confident. The service will consider any predictions with a probability greater than the threshold you set as a predicted class. Setting the threshold high favors precision over recall—classifications will be more accurate, but fewer of them will be found. Setting it low will favor recall—most of the classifications will be found, but there will be more false positives. Experiment with this and use the threshold value that best suits your project. Before launching your application, you'll want to test your model with new images and verify performance.

For challenging data sets or where you need very fine-grained classification, the Advanced Training option in the portal lets you specify how long you want the Custom Vision service to spend training the model. In general, the longer the model trains, the better the performance is. Once you're happy with the performance of a model, you can publish it as a prediction API from the Performance tab in the portal (or via an API) and get the prediction URL and prediction key to call in your code.

How to Train and Call a Custom Vision Model

The following code snippet shows how to train and call a Custom Vision model using version 1 of the C# SDK. First-time users may also want to walk through the steps on the website.

First, instantiate the client:

```
CustomVisionTrainingClient trainingApi =
  new CustomVisionTrainingClient()
  {
  ApiKey = "<Your Training Key>",
  Endpoint = "<Your Service Endpoint>"
  };
```

Next, create a new project:

```
var project = trainingApi.CreateProject("My New Project");
```

Create the image tags to apply to recognized images:

```
var japaneseCherryTag =
  trainingApi.CreateTag(project.Id, "Japanese Cherry");
```

And load the training images from disk. It is often helpful to put images with different classes in separate folders:

```
japaneseCherryImages =
  Directory.GetFiles(Path.Combine("Images", "Japanese Cherry"))
    .ToList();
```

We're uploading the images in a single batch:

```
var imageFiles = japaneseCherryImages.Select(img => new
  ImageFileCreateEntry(Path.GetFileName(img),
  File.ReadAllBytes(img))).ToList();
trainingApi.CreateImagesFromFiles(project.Id, new
  ImageFileCreateBatch(imageFiles,
  new List<Guid>() { japaneseCherryTag.Id }));
```

Now we can start training the Custom Vision model:

```
var iteration = trainingApi.TrainProject(project.Id);
```

Training happens asynchronously, and we will keep querying to find out when the training is complete:

```
while (iteration.Status == "Training")
{
  Thread.Sleep(1000);
  iteration =
  trainingApi.GetIteration(project.Id, iteration.Id);
}
```

Once the iteration is trained, we publish it to the prediction endpoint (you can find the prediction resource ID in the Custom Vision portal under Settings):

```
var publishedModelName = "<Published Model Name>";
var predictionResourceId = "<Prediction Resource ID>";
trainingApi.PublishIteration(
  project.Id,
  iteration.Id,
  publishedModelName,
  predictionResourceId
  );
```

Now we can start making predictions that classify images. First, we need to create a new prediction client:

```
CustomVisionPredictionClient endpoint =
  new CustomVisionPredictionClient()
  {
     ApiKey = "<Your Prediction Key>",
     Endpoint = "<Your Service Endpoint>"
  };
```

Then we can make a prediction:

```
var result =
  endpoint.ClassifyImage(
     project.Id, publishedModelName, testImage
     );
```

And we can loop over each prediction and write out the results:

```
foreach (var c in result.Predictions)
{
  Console.WriteLine($"\t{c.TagName}: {c.Probability:P1}");
}
```

Edge Deployment Options for Vision (and Other Cognitive Services APIs)

If you want to embed your Custom Vision classifier in an app to run it locally on a device, you can export it as TensorFlow for Android, CoreML for iOS 11, ONNX for Windows ML, or as a Windows or Linux container. Only models generated by the "compact" domain types can be exported. You choose this option when first creating your project. These models are smaller, which makes exporting easier but also means they are slightly less accurate than the standard model types. If you didn't choose a compact domain to start with, you can convert but the models will need to be retrained.

Beyond exporting Custom Vision models, an increasing number of the Cognitive Services can be hosted in containers. Containers enable you to deploy these models at the edge, whether that's on a local computer, in your company's data center, or on an IoT device. If you can't send your data to the cloud because of bandwidth and latency issues or regulations, containers let you process the data locally. Running in the cloud has the advantage that you don't need to rebuild your app when you update a model, but local deployment may be a better choice for image and video recognition, where high latency can cause problems. Your data is not sent to the cloud, but the containers do need to connect to Azure to verify your account and send billing data.

There's no transactions-per-second cap on Cognitive Services running in containers, so you can scale up or out as necessary to manage demand. Using containers also gives you more flexibility for versioning and updating the models you deploy. You can create a portable application architecture that you can deploy in Azure or on premises in a Kubernetes cluster.

New container offerings are constantly being added. Currently, seven services offer some of their capabilities in containers: Anomaly Detector, Computer Vision, Face, Form Recognizer, Language Understanding, Speech Services, and Text Analytics. A single container often does not contain a full service. For example, Text Analytics has a separate container for each of its three endpoints: Key Phrase Extraction, Language Detection, and Sentiment Analysis. You can find the most up-to-date list in the Microsoft documentation. This architecture means you can scale up specific operations to the scale you need in your application.

The container images for Cognitive Services can be pulled from the Microsoft Container Registry or Docker Hub, but you currently need to request access to the Face and Recognize Text containers through a questionnaire (using a Microsoft or Azure AD account), then use the credentials provided to pull the images from a private Azure Container Registry.

Face

The Face API delivers much more detailed information than the simple face recognition feature included in the Computer Vision API. You can also use it to compare two faces or to search by face for images of the same person.

At the heart of the Face API is a set of detection tools to extract the human faces from an image and provide a bounding box to indicate where each face is in the image. It'll also give you additional information, including details on the pose position, gender, approximate age, emotion, smile intensity, facial hair, and whether or not the person is wearing glasses (and what type). You can even extract a 27-point array of face landmarks, which can be used to give further information about a face.

The API is powerful: up to 64 different faces can be returned per image. The more faces you're detecting, though, the more time detection takes—especially if you are extracting additional attributes. For large groups it's better to get the minimum information your app needs, and then run deeper analysis on a face-by-face basis.

The face verification endpoint lets you verify whether two faces belong to the same person or whether a face image belongs to a specific person. This is a useful tool for identifying a user and providing a personalized experience. For offline scenarios, this endpoint is available as a container.

The tools for finding similar faces might seem like those used for face verification, but they don't operate at the same level. Here a verified target face is compared to an array of candidate faces, helping you track down other instances of that person. Faces returned may or may not be the same person. You can switch between a more accurate `matchPerson` mode and a less accurate `matchFace`, which only looks for similarities.

In both cases, the API also returns a confidence score that you can use to set the cutoff point for verification or similar faces. You will need to think about what level of confidence is acceptable in your scenario. For example, do you need to err on the side of protecting sensitive information and resources?

The person identification capability is a more generalized case of face verification. For this scenario, a large database of tagged data is used to identify individuals—for example, identifying people in a photo library where a known group of friends can be used to automatically apply tags as the images are uploaded. This can be a large-scale database, with up to a million people in a group and with up to 248 different faces per person. You will need to train the API with your source data, and once trained it can be used to identify individuals in an uploaded image.

If you've got a group of faces and no verified images to test against, you have the option of using the face grouping tools to extract similar faces from a set of faces. The results are returned in several groups, though the same person may appear in multiple groups as they are being sorted by a specific trait (for example, a group where all the members are smiling, or one where they all have blond hair).

How to Use the Face API

The following sample code shows how to use the Face API. Don't forget to substitute in your subscription key and image URL, as well as choosing an Azure Cognitive Services endpoint.

First, we initialize a Face client:

```
FaceClient faceClient = new FaceClient(
  new ApiKeyServiceClientCredentials("<Your Subscription Key>"),
  new System.Net.Http.DelegatingHandler[] { });
faceClient.Endpoint = "<Your Service Endpoint>";
```

Now we can detect faces and extract attributes:

```
IList<DetectedFace> faceList =
      faceClient.Face.DetectWithUrlAsync(
      "<Remote Image URL>", true, false,
      { FaceAttributeType.Age, FaceAttributeType.Gender }
  );
```

Here we extract the age and gender attributes returned by the model:

```
string attributes = string.Empty;
foreach (DetectedFace face in faceList)
{
   double? age = face.FaceAttributes.Age;
   string gender = face.FaceAttributes.Gender.ToString();
   attributes += gender + " " + age + "   ";
}
```

We can display the face attributes like so:

```
Console.WriteLine(<Remote Image URL>);
Console.WriteLine(attributes + "\n");
```

Form Recognizer

Many businesses have mountains of unstructured data sitting in PDFs, images, and paper documents. While these resources may contain the data and insights needed to drive the business forward, they often sit unutilized due to the immense cost and complexity of converting them into structured data. Form Recognizer lowers this barrier and can accelerate your business processes by automating the information extraction steps. Using this service you can turn PDFs or images of forms into usable data at a fraction of the usual time and cost, so you can focus on acting on the information rather than compiling it.

The service uses advanced machine learning techniques to accurately extract text, key/value pairs, and tables from documents. It includes a prebuilt model for reading sales receipts that pulls out key information such as the time and date of the transaction, merchant information, amount of tax, and total cost—and with just a few samples, you can customize the model to understand your own documents. When you submit your input data, the algorithm clusters the forms by type, discovers what keys and tables are present, and associates values to keys and entries to tables. The service then outputs the results as structured data that includes the relationships in the original file. After you train the model, you can test and retrain it and eventually use it to reliably extract data from more forms according to your needs.

As with all the Cognitive Services, you can use the trained model by calling the simple REST APIs or using the client libraries.

Ink Recognizer

Natural user interfaces are the next evolution in the way we interact with computers. A natural interface is one that mimics or aligns with our own natural behavior, relying for example on speech, hand gestures, or handwriting detection. One of the barriers to providing a seamless natural interface for users is understanding and digitizing a person's writings and drawings. The Ink Recognizer service provides a powerful ready-to-use tool for recognizing and understanding digital ink content. Unlike other services that analyze an image of the drawing, it uses digital ink stroke data as input. Digital ink strokes are time-ordered sets of 2D points (x,y coordinates) that represent the motion of input tools such as digital pens or fingers. The service analyzes this data, recognizes the shapes and handwritten content, and returns a JSON response containing all the recognized entities (as shown in Figure 4-11).

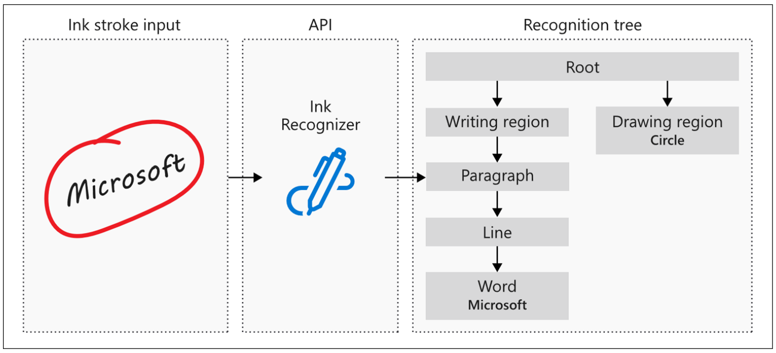


Figure 4-11. Overview of the Ink Recognizer Cognitive Service

This powerful tool lets you easily create applications with capabilities like converting handwriting to text and making inked content searchable.

Video Indexer

Not every image you'll want to analyze is a still image. The Video Indexer service provides both APIs and an interactive website to extract information from videos. Once you upload a video, the service will run it through a large number of models to extract useful data such as faces, emotions, and detected objects. This metadata can then be used to index or control playback. Put it all together and you can take a one-hour video, extract the people and topics, add captions, and put in links that start the video playing in the right place.

Video Indexer is a cloud application built on top of the Media Analytics service, Azure Search, and Cognitive Services. You can explore all the features the service has to offer through the website, and you can also automate video processing using the REST APIs.

Some features, like face identification, let you create custom models, but most work in much the same way as the still image analysis tools available through the Cognitive Services.

As the Video Indexer is a mix of services, you'll need to register and obtain tokens before you can use it. You will need to generate a new access token every hour. Videos are best uploaded to public cloud services like OneDrive or an Azure Blob. Once uploaded, you must provide the location URL to the Video Indexer APIs.

Once you've logged in, you can start to process your videos and gain new insights. The insights cover both video and audio. Video insights include detecting faces and individuals, extracting thumbnail images, identifying objects, and extracting text. There are also insights specific to produced videos, such as identifying the opening or closing credits of a show, key frames, and blank frames (see Figure 4-12).

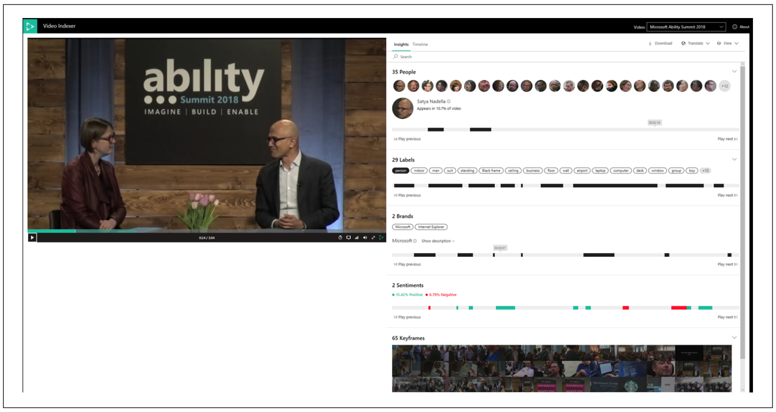


Figure 4-12. Example of the interfaces and intelligence you can extract using video indexer

Audio insights include detecting language, transcribing audio (with the option of using custom language models), creating captions (with translation), detecting sounds like clapping (or silence), detecting emotions, and even identifying who speaks which words and generating statistics for how often each person speaks. You can also clean up noisy audio using Skype filters.

The Video Indexer is one of the more complex offerings in the Cognitive Services. It can require a considerable amount of programming to navigate the index object and extract usable data. You can simplify the process by working in the portal or using Microsoft's own widgets in your applications.

CHAPTER 5
Speech

Speech recognition has long been one of the more complex computer science problems—but years of research and recent breakthroughs with deep learning neural networks have turned this from a research problem into a set of easy-to-use services. The very first successful implementation of deep learning instead of the traditional speech recognition algorithms was funded by Microsoft Research. In 2017, a system built by Microsoft researchers outperformed not just individuals but a more accurate multitranscriber process at transcribing recorded phone conversations.

The Cognitive Services Speech Services are built upon these innovations: they provide a set of pretrained speech APIs that work across multiple speakers and many different languages. Add them to your code and you're using the same engines that power Microsoft's own services, from Skype's real-time translation tools to PowerPoint's live captioning.

The Speech Services include speech-to-text, text-to-speech, voice identification, and real-time translation capabilities. Combined, these features make it easy to add natural interaction to your apps and let your users communicate in whatever way they find convenient.

The services are available through the Speech SDK, the Speech Devices SDK, or REST APIs. These cloud APIs enable speech-to-text translation in just a few lines of code, making it economical to add these capabilities in applications where client-side translation services would have been considered too expensive.

Speech to Text

It used to require hours of time and specialized equipment for a trained human to translate speech into text. Often transcribers used a system that was more like drawing gestures than normal typing. It was expensive, and even commercial services didn't always reach high enough accuracy.

The Speech to Text tool works with real-time streamed audio data or prerecorded audio files. It's the same underlying technology as that used in Cortana, so it's been proven in a wide range of conditions, with many accents and in multiple languages. The list of supported languages is long and continues to grow, covering most European languages, Arabic, Thai, Chinese, and Japanese. Not all languages offer the same level of customization, however.

Speech to Text is available through a set of SDKs and REST APIs. As the service is primarily intended to be used with streamed data, it's easiest to use the SDKs. These libraries give you direct access to audio streams, including device microphones and local audio recording files. The REST APIs are useful for quick speech commands (say, for adding speech controls to mobile apps or websites). If you've built custom language understanding models in LUIS, you can use these in conjunction with the Speech Services to extract the speaker's intent, making it easier to deliver what your user is asking for.

.NET uses the `Microsoft.Cognitive.Services.Speech` namespace to expose all interactions with the service. The key base class is the `Recognizer` that controls the connection to the service, sending speech data and detecting start and end events. Calls to the `SpeechRecognizer` are asynchronous, and the SDK handles the connection to your microphone and recognizing data until a preset length of silence is found. Calls to the service can either include short speech snippets or long utterances for recognition. There is also a continuous recognition model. The SDK returns recognized speech as a string, with error handling for failed recognitions.

Here's a snippet of what this looks like using version 1 of the C# SDK. First, configure your Speech credentials by filling in your own subscription key and service region (e.g., `"westus"`):

```
var config = SpeechConfig.FromSubscription(
        "<Your Subscription Key>", "<Your Service Region>"
    );
```

Next, create a `SpeechRecognizer`:

```
var recognizer = new SpeechRecognizer(config);
```

This calls the API to capture short utterances and convert them to text—for long-running multiutterance recognition, use `StartContinuousRecognitionAsync` instead:

```
var result = await recognizer.RecognizeOnceAsync();
```

Now, check to see if the speech was recognized (other options are `NoMatch`, `Cancelled`, and `Error`):

```
if (result.Reason == ResultReason.RecognizedSpeech)
{
    Console.WriteLine($"We recognized: {result.Text}");
}
```

One of the difficulties with speech recognition is the many ways people speak. Speech styles, prosody, accents, and vocabulary vary considerably. Your business also likely has unique names and jargon that are not found in general dictionaries. To address these challenges, you can customize the speech models to understand accents or work with specific vocabularies. Like the Custom Vision service, these customizations build on top of the existing trained models and allow you to create use case-specific models without the burdensome data requirements of creating your own from scratch.

The places in which speech is being recorded can pose challenges too. For example, the background noise at a drive-through or the acoustics of a mall are both very different from someone speaking into their phone in a quiet room. You can add acoustic models to account for the complexities of varied environments where accurate recognition is essential: in vehicles, on the factory floor, or out in the field. Adding a custom acoustic model will be necessary if you're building code for use in a predictably noisy environment.

To get started building your own custom models, you will need samples recorded in the same conditions in which your application will be recognizing speech. That means people talking in the environment or into the device you plan to use. You can also use this method to tune speech recognition to a single voice. This technique is useful for transcribing podcasts or other audio sources.

Data needs to be in 8 KHz or 16 KHz WAV files, using mono recordings. Split them up into 10- to 12-second chunks for the best results, starting and finishing with silence. Each file needs a unique name, and should contain a single utterance: a query, a name, or a short sentence. Package the files in a single zipped folder that's less than 2 GB, and upload that to the Custom Speech website. Each file needs to be accompanied by the transcription in the correct format: a single line of text in a file that starts with the audio file's name, then a tab, then the text. You will then need to walk through the process of training the model on the website.

Building custom language models, either for a specific technical vocabulary or to improve recognition of accented speech, also requires labeled data. However, this data consists of a list of sentences and phrases as text rather than voice recordings. For best results, include text that uses your specific vocabulary in different sentences and contexts that cover the ways you expect the terms to be used. You can provide up to 1.5 GB of raw text data. The service's website provides a walkthrough on how to create these custom acoustic models.

Text to Speech

We can't always be looking at screens. In many cases this can be a dangerous distraction, diverting attention away from hazardous environments or expensive equipment. When people need to interact with devices in such environments, one option is to use speech synthesis, perhaps paired with speech recognition for input. Speech synthesis can also provide accessibility tooling for the visually impaired or be used to deliver information in an augmented reality tool.

The following code snippet will take text from the console input and play the resulting speech through your default audio device.

First you will need to configure your Speech credentials (fill in your own subscription key and service region (e.g., `"westus"`):

```
var config = SpeechConfig.FromSubscription(
     "<Your Subscription Key>", "<Your Service Region>"
 );
```

Then, create a `SpeechSynthesizer` that uses your device speaker:

```
var synthesizer = new SpeechSynthesizer(config);
```

Here we're going to take the text to be spoken from console input:

```
string text = Console.ReadLine();
```

Call the API and synthesize the text to audio:

```
var result = await synthesizer.SpeakTextAsync(text);
```

Then check to see if the speech was synthesized (the other options are `Cancelled` and `Error`):

```
if (result.Reason == ResultReason.SynthesizingAudioCompleted)
{
   Console.WriteLine(
       $"Speech synthesized to speaker for text [{text}]"
   );
}
```

Neural and Custom Voices

The Text to Speech service converts text into synthesized speech that's natural and sounds near human. You can pick from a set of standard and higher-quality "neural" voices, or if you want to express your brand's personality you can create your own voices.

Currently five neural voices are available in English, German, Italian, and Chinese. You can also choose from more than 75 standard voices in over 45 languages and locales. The standard voices are created using statistical parametric synthesis and/or concatenation synthesis techniques.

Neural text to speech is a powerful new improvement over standard speech synthesis, offering human-sounding inflection and articulation. The result is computer-generated speech that is less tiring to listen to. It's ideal if you're using speech to deliver long-form content, for example when narrating scenes for the visually impaired or generating audiobooks from web content. It's also a useful tool when you're expecting a lot of human interaction, such as for high-end chatbots or virtual assistants.

Standard speech synthesis supports many more languages, but it's clearly artificial. You can experiment to find the right set of parameters to give it the feel you want, tuning speed, pitch, and other settings—including adding pauses.

To generate your own custom voices, known as *voice fonts*, you need studio recordings, preferably made by a professional voice actor, and a set of scripts to create the training data. It is possible to use public recordings, but they will require significant editing to remove filler sounds and ambient noise. The best results come when you use an expressive voice at a consistent volume, speaking rate, and pitch. Voice fonts can only be single language: either English (US), Chinese (Mainland), French, German, or Italian.

Custom voices can be configured using the service's website. Audio files for creating a voice need to be WAV files sampled with a sample rate of at least 16 KHz, in 16-bit PCM, bundled into a ZIP file less than 200 MB in size. Files need to be single utterances: either a single sentence or a section of any dialog you wish to construct, with a maximum length of 15 seconds. As with custom recognition, you also need a script file that ties the voice to text. You can upload multiple speech files, with free users limited to 2 GB and subscription users to 5 GB.

To turn your uploaded voice data set into a voice font, you need to set the locale and gender for the voice to match the data set. Training can take a significant amount of time, depending on the volume of data (from 30 minutes to 40 hours). Once the voice font has been trained, you can try it out from the portal.

Text sent via the REST API must use the Speech Synthesis Markup Language (SSML), which controls how voices operate. Start by setting the voice you're using, then add text in an XML format. You can add breaks, using time in milliseconds, and change the speaking rate (defined as prosody, and increased and decreased using percentages). You can also change how a voice pronounces a word, altering the phonemes used. Other options let you control volume and pitch, and even switch between voices. Constructing speech as SSML can take time, but it gives you a wide range of options and helps deliver a more natural experience.

Here's the SSML for an enthusiastic rather than neutral response:

```
<speak version='1.0'
      xmlns="https://www.w3.org/2001/10/synthesis"
      xmlns:mstts="https://www.w3.org/2001/mstts"
      xml:lang="en-US">
 <voice name='en-US-JessaNeural'>
   <mstts:express-as type="cheerful">
      That'd be just amazing!
    </mstts:express-as>
  </voice>
</speak>
```

Translation and Unified Speech

Real-time speech translation was one of the first deep learning speech services Microsoft showcased, with a 2012 demonstration showing an English speaker communicating with a Chinese speaker. In just a few years, those translation services have gone from research to product to service. Using neural machine translation techniques, rather than the traditional statistical approach, allows them to deliver higher-quality translations. (Not all language pairs in Azure Speech use neural translation; some still depend on the statistical operations.)

The Speech Translation tool uses a four-step process, starting with speech recognition to convert spoken words into text. The transcribed text is then passed through a TrueText engine to normalize the speech and make it more suitable for translation. Next, the text is passed through the machine translation tools and converted to the target language. Finally, the translated text is sent through the Text to Speech service to produce the final audio.

Speech Translation works in a similar fashion to the standard speech recognition tools, using a `TranslationRecognizer` object to work with audio data. By default it uses the local microphone, though you can configure it to use alternative audio sources. To make a translation, you set both the source and target languages, using the standard Windows language types (even if your app doesn't run on Windows).

Translations are delivered as events, so your code needs to subscribe to the response stream. The streamed data can be displayed as text in real time, or you can use it to produce a synthesized translation, using neural speech if available. Custom Translator lets you extend the default translation models to cover industry-specific terms or language that's essential to your business. We go into more depth on Custom Translator in the next chapter.

Speaker Verification and Identification

In the 1992 movie *Sneakers*, the villain Cosmo protects his computer systems with a speech-controlled door lock using the passphrase "My voice is my passport. Verify me." The Speaker Recognition API promises to turn that fiction into fact. While still in preview, the service makes it easy to identify and even verify the person speaking.

Speaker verification uses the unique characteristics of a person's voice to identify them, just like a fingerprint. Speaker identification uses enrolled voices to identify a specific voice based on speech patterns. By playing back selected audio, it can determine who in the speech database is speaking.

Speaker Verification

We all speak differently; our voices have unique characteristics that make them suitable for use as biometric identifiers. Like face recognition, voice recognition is a quick and easy alternative to the traditional password, simplifying login and access. You enroll users with a spoken passphrase that they will use when they want to be verified. Once they're enrolled with at least three samples, the passphrase can be processed to build a voice signature.

When a user speaks their passphrase for verification, it's processed to create a voice signature, which is then compared with the stored phrase. If the phrase and voice match, the user is verified.

You need to create a profile for each user, including their language; currently only US English is supported. When the profile is created through the REST API, it returns a GUID that you use to enroll the user. While the service is in preview, you can only register a thousand profiles per subscription.

There's a preset list of supported passphrases (which is quite short for the preview), and each user will need to choose one—they don't need to be unique to each user, though. You can retrieve the list through the API and either present one randomly or let the users pick from a list. Each user can then provide three samples of them saying the selected phrase, which must be recorded in mono WAV format at 16 KHz and in 16-bit PCM. Each sample is uploaded separately, along with a count and the phrase being used. Once all three have been loaded, the profile is then trained, and when training is complete, it's ready for use.

A single REST API call handles verifying a user, but you still need to handle usernames and give users a way to enter these in your application so you can retrieve the appropriate GUID from a local store. They then speak their passphrase, which is sent to the speaker verification service. Three fields are returned: the result of the verification (accept or reject), the confidence level of the verification (low, normal, or high), and the recognized passphrase.

If a user is recognized with high confidence, you can let them into your application. If confidence is low, you may want to ask them to use an alternative authentication method. You may also want to use a similar fallback approach if they are rejected with low confidence rather than locking them out.

Speaker Identification

Speaker identification lets you automatically identify the person speaking in an audio file from a given group of potential speakers. This functionality is useful in dictation applications and other scenarios where you need to differentiate between speakers. You can use the speaker identification service in parallel with the other Speech Services to build an annotated transcript of a conversation, something that's hard to do using conventional speech recognition products.

Because you're not converting speech to text, all you need are speech samples to train the identification service. Think of this as the voice equivalent of music recognition services like Shazam. The service identifies the "voice fingerprint" of an individual and then compares this fingerprint with new recordings to determine if the same person is speaking. Multiple recordings can be compared with the same voice fingerprint, or different voices extracted from the same file.

Speaker identification uses the same underlying approach as speaker verification. You first create a speaker identification profile that can then be enrolled in the service. Enrollment for identification is different from verification because you're uploading a single file, which can be up to five minutes long. The minimum recommended length is 30 seconds, though you have the option of uploading a shorter file. Again, you need to use mono WAV files, recorded at 16 KHz in 16-bit PCM.

To identify a voice, you send a request to the Speaker Identification API with the speech you wish to identify along with a list of the 10 profiles you wish to compare it against. The service runs asynchronously and will return a URI that can be queried to get the status. A separate API returns an identified voice GUID, along with a confidence level. If no one is identified, a dummy zero GUID is then returned.

Both services are still in preview and currently suitable for trial applications rather than production use. You can test and adopt them in production when the service moves to general availability.

CHAPTER 6
Language

Much of what we do online involves text, and Cognitive Services offers tools to work with text across all your applications: from providing analytics on text content to understanding what a user really wants to do. Instead of forcing users to check boxes, click radio buttons, or choose from a drop-down list, you can use these services to make your app understand the meaning of unstructured text or recognize the intent of what your users are saying no matter how they express it. The Language services also integrate well with the Speech Services we covered in the previous chapter.

Text Analytics

The Text Analytics service takes raw text and extracts four kinds of structured information: the sentiment, the key phrases, the language, and the entities the text refers to. You can use all four endpoints independently, but combining them can enable a higher level of analysis. The four APIs are all very straightforward, and there is no option to tune or customize the service with your own data. All the Text Analytics endpoints are available as containers, meaning you can take advantage of these capabilities in your own data center or edge devices.

The following code snippet creates a Text Analytics client with version 3 of their SDK, ready for use with your content. It can be used with the standard endpoints to extract, for example, sentiment:

```
class ApiKeyServiceClientCredentials : ServiceClientCredentials
{
        public override Task ProcessHttpRequestAsync(
          HttpRequestMessage request,
          CancellationToken cancellationToken)
          {
           request.Headers.Add(
               "Ocp-Apim-Subscription-Key",
                   "Your Subscription Key"
             );
           return base.ProcessHttpRequestAsync(
               request, cancellationToken
             );
          }
    }
```

You can create a Text Analytics client as follows—make sure to use the correct service endpoint for your Text Analytics subscription:

```
    TextAnalyticsClient client = new
        TextAnalyticsClient(new ApiKeyServiceClientCredentials())
        {Endpoint = "<Your Service Endpoint>"};
```

Sentiment Analysis

A common use for analyzing sentiment is to parse a social media feed like Twitter, looking for references to a product and examining how customers are talking about it. You can use the Sentiment Analysis API to filter live tweets that you show at events, or you can take the results and feed them into an analytical tool like Power BI to generate actionable insights. If a tweet says that a user had problems, you can see not only their low sentiment rating, but also phrases like "didn't work" or "caught fire," allowing you to respond appropriately to a possible customer emergency.

The Sentiment Analysis API takes a text input and gives you a sentiment score of between 0 and 1, where 0 is negative and 1 is positive. The service works with English, German, Spanish, and French, and other languages are being added.

NOTE

Detecting irony and sarcasm can be tricky: a comment that someone is "really delighted" that something happened doesn't always represent genuine enthusiasm, so be careful when filtering on sentiment score.

The model used in the service works with small chunks of data. If you are working with large text collections, you will need to break them up into smaller chunks. The service can process up to 1,000 blocks at a time, and each block can be up to 5,120 characters long. Real-time data is rate-limited to under 100 requests a minute.

Building on the previous code snippet, the following code shows how to call the Sentiment Analysis API to extract the sentiment in multiple languages.

First, create the input example:

```
MultiLanguageBatchInput multiLanguageBatchInput =
 new MultiLanguageBatchInput(
  new List<MultiLanguageInput>()
  {
    new MultiLanguageInput("ja","1", "猫は幸せ"),
    new MultiLanguageInput("de", "2",
        "Fahrt nach Stuttgart und dann zum Hotel zu Fu."),
    new MultiLanguageInput("en", "3",
        "My cat is stiff as a rock."),
    new MultiLanguageInput("es", "4",
        "A mi me encanta el fútbol!")
    }
);
```

Now perform the sentiment analysis:

```
SentimentBatchResult sentimentBatchResult = await
        client.SentimentAsync(null, multiLanguageBatchInput);
```

You can then display the sentiment for each document analyzed:

```
foreach (var document in sentimentBatchResult.Documents)
        {
          Console.WriteLine(
           $"Document ID: {document.Id}," +
           $", Sentiment: {document.Score}"
          );
        };
```

Key Phrase Extraction

The Key Phrase Extraction API can be used to find relevant phrases in unstructured text, providing a great way to quickly identify the main points. The service works on text in English, Japanese, German, and Spanish. Analyzing this paragraph would return phrases like "relevant phrases" and "English," "German," "Japanese," and "Spanish."

Unlike the Sentiment Analysis API, the Key Phrase Extraction API requires large documents, and if you are running the two on the same content you will need to break up the text differently for each service. Results are returned as JSON. Nonessential phrases are dropped and single terms that are the subject or object are retained, so if you are using this as a tool for summarizing content, you will need to add your own code to integrate the returned values into your application.

The following snippet uses the Text Analytics client we defined earlier to extract key phrases from a multilanguage input.

First, we'll create the input. We can use the same example as earlier:

```
MultiLanguageBatchInput multiLanguageBatchInput =
   new MultiLanguageBatchInput(
    new List<MultiLanguageInput>()
    {
      new MultiLanguageInput("ja","1", "猫は幸せ"),
      new MultiLanguageInput("de", "2",
          "Fahrt nach Stuttgart und dann zum Hotel zu Fu."),
      new MultiLanguageInput("en", "3",
          "My cat is stiff as a rock."),
      new MultiLanguageInput("es", "4",
          "A mi me encanta el fútbol!")
  }
);
```

Now we'll extract the key phrases:

```
KeyPhraseBatchResult keyPhraseBatchResult =
   await client.KeyPhrasesAsync(null, multiLanguageBatchInput);
```

We can then display the key phrases for each document to be analyzed as follows:

```
foreach (var document in keyPhraseBatchResult.Documents)
{
    Console.WriteLine(
        $"Document ID: {document.Id}," +
        $" Key Phrases: " +
        $"{ string.Join(", ", document.KeyPhrases)}"
      );
}
```

Combining the Sentiment Analysis and Key Phrase Extraction APIs can help reveal root causes; if negative comments talking about fires have common key phrases like "plug" and "power supply," you can narrow down the problems behind the complaints.

Language Detection

The Language Detection API can simplify using other parts of the Cognitive Services suite of APIs. Many services attempt to autodetect language, but if you need to use a language-specific service to ensure accuracy, you can pass a snippet of the text through the Language Detection API. The service currently supports more than 120 languages.

All you need to do is send a JSON-formatted document via a POST request to the API endpoint. The content in the document will be scanned, and the response will contain a list of the detected languages and their associated ISO codes, as well as a confidence score for each.

The following code snippet uses our Text Analytics client to detect the language used in three different text snippets.

First, we create the input:

```
LanguageBatchInput languageBatchInput =
     new LanguageBatchInput(
       new List<LanguageInput>()
       {
         new LanguageInput(null, "1",
           "This is a documentwritten in English."),
         new LanguageInput(null, "2",
           "Este es un document escrito en Español."),
         new LanguageInput(null, "3",
           "这是一个用中文写的文件")
       }
       );
```

Now we'll detect the language in each snippet:

```
LanguageBatchResult languageBatchResult = await
     client.DetectLanguageAsync(null, languageBatchInput);
```

Finally, we can display the language detected for each document:

```
foreach (var document in languageBatchResult.Documents)
        {
          Console.WriteLine($"Document ID: {document.Id}," +
            $"Language:" +
            $"{ document.DetectedLanguages[0].Name}");
         }
```

Entity Recognition

The Entity Recognition API (`/entities` endpoint) takes unstructured text and returns a list of disambiguated entities with links to more information on the web. For example, "Mars" could be a reference to the planet or the British candy bar. Using entity linking with Wikipedia as a knowledge base, Text Analytics can differentiate between the two based on the surrounding text. It can also find time zones, temperatures, numbers, and percentages. The Named Entity Recognition feature allows you to identify places, people, quantities, businesses, dates and times, URLs, and email addresses. All these features help you turn a stream of text into more structured data points.

The service works in the same way as the other Text Analytics services. The following code snippet shows how to extract entities from a multilanguage document, again using the Text Analytics client we created earlier.

First we create the input. We'll use the same text examples we used for the other operations:

```
MultiLanguageBatchInput multiLanguageBatchInput =
   new MultiLanguageBatchInput(
      new List<MultiLanguageInput>()
      {
       new MultiLanguageInput("ja", "1", "猫は幸せ"),
       new MultiLanguageInput("de", "2",
           "Fahrt nach Stuttgart und dann zum Hotel zu Fu."),
       new MultiLanguageInput("en", "3",
           "My cat is stiff asa rock."),
       new MultiLanguageInput("es", "4",
           "A mi me encanta el fútbol!")
      }
      );
```

Now we'll extract the entities from the documents:

```
EntitiesBatchResult entitiesBatchResult = await
  client.EntitiesAsync(null, multiLanguageBatchInput);
```

Finally, we can display the extracted entities for each document:

```
foreach (var document in entitiesBatchResult.Documents)
 {
   Console.WriteLine($"Document ID: {document.Id}");
   foreach(var entity in document.Entities)
   {
       Console.WriteLine(
             $"Name: {entity.Name}, Type: {entity.Type}"
             );
        }
     }
```

Language Understanding with LUIS

Perhaps the biggest problem facing anyone trying to implement a natural language interface like a chatbot is trying to understand what a user *means*. We all have different ways of talking, typing, and speaking. We use different words to mean the same thing, and sometimes the same word to mean different things. When someone ordering a pizza through a chat interface asks for it to be delivered to their digs, what do they mean? Could they really want their pizza in a hole?

The Language Understanding service, or LUIS, is designed to resolve these situations by determining the user's intentions and turning the conversation into a list of ways an application can respond. Building on an understanding of the context of a conversation, LUIS can examine an utterance or input from the user, and find appropriate keywords. LUIS then maps these keywords to a list of intents and entities that can be consumed by the application.

An intent represents a task or action the user wants to perform. The intent is the purpose or goal expressed in a user's utterance. The entities are the words or phrases in the utterance that you want extracted. They often represent the important context needed for acting on the user's intent. For example, consider the utterance "buy 3 tickets to New York." The intent of the user could be "to buy" and the entities could be "3" and "New York."

Your code can then take the returned intents and entities and apply appropriate rules, handing them off to specific actions. It's important to note that it's your code that makes the decisions; LUIS just returns the information you need in the format you require.

Let's look at another example. Imagine adding LUIS to a travel agency chatbot. LUIS can help you identify the information needed to help a customer from their input. A statement like "I need a flight to Heathrow" will be parsed with the intent "BookTravel" and entities "flight" and "London." Prefilling those entities into a booking engine starts the booking process, while the chatbot prompts the user for additional information, like dates, class of travel, and the departure airport.

LUIS is not a general-purpose ML model; to get the most out of it, you must train it with domain-specific data for your industry, location, and scenario. A set of prebuilt models for specific domains can help you get started, but for the best results, they'll need additional training.

You can access LUIS through its website or APIs. After creating an account on the service website, create an app and walk through the initial steps of adding data to train your own model. When you create an app, you will be prompted to choose a "culture." The culture is the language that your app understands and speaks. This is perhaps the most important choice you make, as it defines the language used for understanding context, which sets up the basic model used by LUIS (and cannot be changed later).

A library of preconfigured domains for common LUIS scenarios helps you get started quickly. Add any of these to your LUIS instance from the portal and you can explore the available intents included in the library, with the associated sample utterances.

Once you've selected the domains that fit your scenarios, you can train LUIS directly from the website. This can take some time, especially if you're using more than one domain. Once training is complete, you can also try out your model right from the site. An interactive testing pane displays the highest-scoring intent for each test utterance.

When you're happy with the model you have created, you can publish it and create REST API endpoints that can be called directly from your code. The URIs are of the following form:

```
https://<region>.api.cognitive.microsoft.com/luis/v2.0
  /apps/<appID>?verbose=true&subscription-key=<YOUR_KEY>&
  <optional-name-value-pairs>&q=<user-utterance-text>
```

The resulting data can be parsed and used in your apps, handling queries or driving the next step in a bot conversation. The following code snippet shows how you might use LUIS to add voice control to a home IoT system.

First, configure your credentials:

```
var credentials =
  new ApiKeyServiceClientCredentials("<Your Subscription Key>");
```

and create a LUIS client:

```
var luisClient = new LUISRuntimeClient(
    credentials,
      new System.Net.Http.DelegatingHandler[] { }
);
```

Then set the endpoint, using your correct service region:

```
luisClient.Endpoint = "<Your LUIS Endpoint>";
```

Next, create a Prediction client:

```
var prediction = new Prediction(luisClient);
```

Now return a prediction (this query is for a home automation app):

```
var luisPrediction = await prediction.ResolveAsync(
    appId = "<Your App ID>",
    query = "turn on the bedroom light",
    timezoneOffset = null,
    verbose = true,
    staging = false,
    spellCheck = false,
    bingSpellCheckKey = null,
    log = false,
    CancellationToken.None
);
```

and get the result:

```
var luisResult = luisPrediction.Result;
```

You can display the results as follows:

```
Console.WriteLine(
    luisResult.Query,
    luisResult.TopScoringIntent.Intent, // top intent
    luisResult.TopScoringIntent.Score,
);
```

And you can display all the entities detected in the query like this:

```
foreach (var entity in luisResult.Entities)
{
    Console.WriteLine(
    "{0}:'{1}' begins at position {2}" +
    "and ends at position {3}",
    entity.Type, entity.Entity, entity.StartIndex,
    entity.EndIndex
    );
}
```

Production LUIS models can be retrained automatically, using active learning to work with user utterances that LUIS marks as needing validation (because the intent detected has a low score, or two intents have such close scores that neither is definitive). You can map utterances to the right intent one at a time or in batches. You can also manually add any new entities that appear in the utterances. Make this validation part of your regular app lifecycle management to keep your LUIS model current for what users are asking.

You can also provide a list of words or phrases that are significant to your app. This *phrase list* might include synonyms for important terms, to help the model handle different ways of asking the same question. For example, a car might be referred to as a sedan, coupe, automobile, or motor.

NOTE

Generating your own data for LUIS requires you to map example utterances to your planned intents and entities. If you collect all the examples in a JSON file, you can create the LUIS app programmatically, adding intents and entities via the API. A batch API lets you add 100 utterances at a time, and once the input has been processed, you can start to test your domain model from the LUIS portal before writing an app to work with the endpoints.

QnA Maker

Customer support is expensive. Discussing problems with customers and looking up the right information to help each user requires a lot of manual effort and time. The QnA Maker service lets you deliver faster answers to customers and reduce your support costs by transforming semi-structured data such as frequently asked question (FAQ) pages into a conversational interface. The service can be integrated with Microsoft's Bot Framework to create a chatbot that can respond through a wide selection of channels, from text messages to Slack.

QnA Maker has two main services: the first extracts question-and-answer pairs and the second matches answers to a user's question. Extraction discovers questions and answers in the various documents you upload. Matching delivers an endpoint that accepts user questions and responds with the most appropriate answer. The service also provides a confidence score that tells you how confident the service is in the match.

Unlike the other Cognitive Services, QnA Maker relies on other Azure services that are deployed when you create a resource. The first service is the QnA Maker Management service that orchestrates the service and controls the training and publishing experiences. The second is an Azure Search instance, which is used to store any data uploaded to the service. Finally, the endpoints are deployed as an Azure App Service. If you need additional logging, you can set up an App Insights link to your QnA Maker service. All these services will appear as separate resources in the Azure portal when you deploy an instance of the QnA Maker service.

Once you have deployed the resources, you can get started via the website. The portal provides an easy-to-use interface for creating, managing, training, and publishing your own custom QnA Maker model without any developer experience. Developers can also perform any of these functions using the service's REST APIs or client libraries.

The QnA Maker service is ideal for integrating into a chatbot, and the website even provides functionality for creating a bot. Bots can be given different personalities using QnA Maker's chit-chat option, which lets you choose between Professional, Friend, and Comic personas. The service now also supports bots for specific industry verticals, including healthcare.

Spell Check

Most spelling checkers use large dictionaries and rule sets. As a result, they tend to be hard to update and quickly become static. The Bing Spell Check API takes advantage of the Bing search index, which quickly picks up new terms, company names, and jargon that get published to the web and uses a combination of advanced machine learning and statistical techniques to deliver more accurate and relevant corrections.

The Spell Check API brings the power of Bing's spell checking capabilities to your application. The service offers multiple modes for checking spelling and grammar. It can also easily handle slang and other informal terms. It offers features including multiple spell check modes, slang recognition, homophone detection, and brand and proper noun recognition.

The two modes you can select from are Spell and Proof. Proof not only corrects spelling but also adds basic punctuation, capitalization, and other grammar checks. However, it's only available for the US English, Spanish, and Portuguese markets and truncates the query text at 4,096 characters. For a quicker response or use in other regions, use Spell. Spell only offers word-level spelling checks (for queries of 65 or 130 characters depending on the language) and won't find some of the grammar mistakes Proof does. You can choose the language of the spell check, which is especially useful if you're building a site or service using multiple languages.

The following code snippet shows how to call the Spell Check API with version 4 of the SDK. First, create a client:

```
var client = new SpellCheckClient(
  new ApiKeyServiceClientCredentials("<Your Subscription Key>")
  );
```

Then perform the spell check, displaying the errors found and the suggestions for replacing them, showing the confidence score for each one:

```
string testSentence = "tset sentense";
SpellCheckModel spellCheckModel =
  await client.SpellCheckerAsync(testSentence);
// for each error detected
foreach (var token in spellCheckModel.FlaggedTokens)
{
    Console.WriteLine($"Error: {token.Token}");
    // print the suggestions & confidence score
    foreach (var suggestion in token.Suggestions)
    {
        Console.WriteLine(
                $"Suggestion: {suggestion.Suggestion},
                Score: {suggestion.Score}"
            );
    }
}
```

As with all the services, there are numerous configuration options that can be passed along with the input text. One of the more interesting ones is `preContextText`, which allows you to provide a string that gives context to the words being checked. For example, if you didn't set a context, "petal" would be marked a perfectly correct spelling, but if you set `preContextText` to "bike," then the spell-checker would suggest changing it to "pedal."

Translator Text

Microsoft Translator is a cloud-based machine translation service with multilanguage support for translation, transliteration, language detection, and dictionaries. The core service is the Translator Text API, which powers a number of Microsoft products and services (including the speech translation feature in the Azure Speech Services, which we looked at in the previous chapter).

At the heart of the Translator Text API is a deep learning–powered neural machine translation algorithm. This technique has revolutionized machine translation over the last decade. By translating words as part of a full sentence rather than considering them individually, the service provides more accurate, natural, and fluent results. The result is a more human-sounding translation, particularly with non-Western languages. Translator Text also provides transliteration services that let you display text in different alphabets, making it easier to read. For example, transliterating can be useful for converting Japanese or Chinese pictographs to a Western alphabet.

Building an application with translation capabilities is simple. The latest version 3 release consolidates most of the functions into a single REST API, making it easier to access the models. The APIs are hosted in the Americas, in Asia Pacific, and in Europe. There is also a Global option that routes calls dynamically to any data center, usually the one closest to the request location.

You can have more than one target language in your translation request, with each translation in the same JSON return. The response contents indicate the detected source language and include a translation block with text and a language identifier for each selected target language.

An alternative use of the service is to simply identify the language being used. That's useful if you have some content to translate but don't want to waste API calls on the wrong language pairing or content that can't be translated. With a basic app, you can make a simple detect call to the REST API, sending your sample text in the JSON body of the request. The API will return the identified language and an indicator of whether the service can translate it for you, so you can use the language identifier to make an appropriate translation or transliteration request.

You can also combine the different capabilities to create your own translation service: for example, detecting Japanese text, transliterating it to Western script at the same time as translating it, while also displaying any alternate translations that might be available.

Customizable Translation

The Custom Translator feature allows you to build a neural translation system that understands the terminology used in your own business and industry. The service works by providing a labeled data set that reflects your domain-specific terminology and style. The data must contain the content translated into all your targeted languages, and it is a best practice to include at least 10,000 parallel sentences. You can upload these data sets and create your own model using the service's website. Once trained, you can see various metrics to understand the accuracy of your model.

Immersive Reader

The newest Cognitive Service, Immersive Reader, seeks to empower users of all ages and abilities. Based on extensive research on inclusivity and accessibility, the service offers a suite of features focused on enhancing text reading and comprehension including reading the text aloud, translation, and focused attention views. Immersive Reader is a standalone web app that can be integrated within your existing app via an iframe through the JavaScript SDK. Figure 6-1 is a screenshot of the experience highlighting nouns, verbs, adjectives, and adverbs for the user.

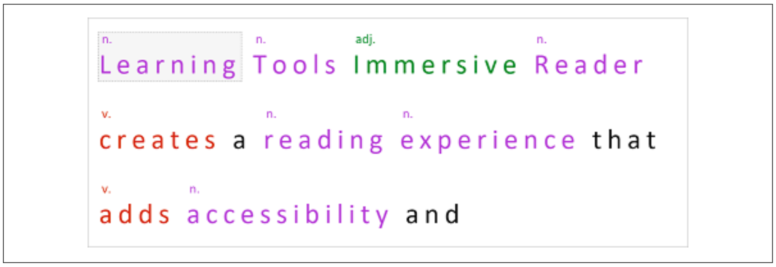


Figure 6-1. Screenshot of the Immersive Reader service

Immersive Reader helps users unlock knowledge from text to achieve gains in the classroom and office. Over 15 million users rely on Microsoft's immersive reading technologies across 18 apps and platforms including Microsoft Learning Tools, Word, Outlook, and Teams. Now, developers can deliver this literacy-enhancing experience to their users too.

NOTE

Andrzej, a child with dyslexia, learned to read with the Immersive Reader experience embedded into apps like Microsoft Learning Tools. You can listen to his mother, Mitra, sharing their story.

CHAPTER 7

Decision

The end goal of most machine learning applications is not just to extract information and insights, but to make decisions and take actions based on the new knowledge. The services within the Decision category, which we'll explore in this chapter, aim to enable these goals by providing higher-level capabilities such as detecting anomalies, personalizing recommendations, and identifying inappropriate content. These services are intended to help the user more quickly move from raw data to impactful action.

Anomaly Detector

Many applications need to detect anomalous behavior and know when to take corrective action. Anomaly Detector simplifies the process of detecting anomalies in time series data. The service examines either real-time or historical data to determine whether a data point is an anomaly, automatically selecting the best algorithm for your data to ensure high accuracy for your scenario. Over 200 product teams within Microsoft rely on this service to deliver accurate anomaly detection, and the service is a powerful tool in many use cases such as spotting fraud, alerting when an IoT device is failing, or identifying suspicious user activity. Here's an example showing how you can integrate the service in just a few lines of code.

First, create a client:

```
var client = new AnomalyDetectorClient(
  new ApiKeyServiceClientCredentials("Your Subscription Key"))
  {Endpoint = "<Your Service Endpoint>"};
```

Now create the list of points to analyze. You will need at least 12 points:

```
Request request = new Request(
    new List<Point>()
    {
    new Point(
          DateTime.Parse("2018-03-01T00:00:00Z"),32858923
          ),
    },
    Granularity.Daily
);
```

You can then add code to detect if there is an anomaly on the last point in the time series:

```
LastDetectResponse lastDetect =
  await client.LastDetectAsync(request);
```

and display the results:

```
Console.WriteLine($"Expected Value:" +
  $"{lastDetect.ExpectedValue}, +
  $", Is Anomaly? {lastDetect.IsAnomaly}");
```

Personalizer

Reaching and engaging customers is all about providing a personalized, unique experience to your users. Providing pertinent news articles, optimized ad placements, or relevant recommended items on a retail page or suggesting useful features in your application can delight your users and increase their engagement.

Personalizer enables you to create personalized experiences like these with no machine learning expertise required. The service uses state-of-the-art reinforcement learning techniques to learn in real time from your users' interactions and optimizes toward a direct business goal you provide.

It works by providing a set of recommendations based on information you supply about your user's context. Depending on the user's action, you then provide a reward back to the model, which is used to improve the model. Since this service trains in real time, you do not need to provide clean, labeled data before using Personalizer.

Content Moderator

Content Moderator checks text, image, audio, and video content for material that is potentially offensive, risky, or otherwise inappropriate. When the service finds such material, it applies labels or flags to the content that your app can use to take action. The service is applicable in a wide variety of scenarios, from moderating product catalogs in open online marketplaces to monitoring user-generated content in games.

Whether you want to keep a chat room family friendly or make sure your ecommerce site doesn't offer products with unfortunate or offensive phrases printed on them, this service can help—but it's best used in conjunction with human moderators. To help with that, Content Moderator also includes a web-based review tool that enables you to build human evaluations directly into your workflow. The human input does not directly train the service, but the combined work of the service and human review teams allows developers to strike the right balance between efficiency and accuracy.

To get started, you can either walk through the documentation or access the service via its REST APIs and client libraries. The following code snippet shows how to use version 2 of the C# SDK to check an image for adult content.

First, identify the URL of the image to check:

```
var url = new BodyModel("URL", "<URL of an Image>".Trim());
```

There are three steps you need to take when evaluating an image:

1. The `Evaluate` operation predicts whether the image contains adult or racy content.
2. The `OCR` operation predicts the presence of text content in the image that may be inappropriate.
3. The `FindFacesUrlInput` operation detects faces in an image to help protect personal data.

The following code snippets walk through these steps using the C# SDKs. First, you need to create a class to handle the results:

```
private static EvaluationData
{
    Evaluate ImageModerating;
    OCR TextDetection;
```

```
        FoundFaces FaceDetection;
    }
```

Then create an instance for the evaluation:

```
    var imageData = new EvaluationData();
```

To evaluate the image using the Image Moderation APIs, first check for adult and racy content:

```
    imageData.ImageModerating =
        client.ImageModeration.
        EvaluateUrlInput("application/json", url, true);
```

Next, detect and extract any text:

```
    imageData.TextDetection =
        client.ImageModeration.
        OCRUrlInput("eng", "application/json", url, true);
```

Finally, detect any faces:

```
    imageData.FaceDetection =
        client.ImageModeration.
        FindFacesUrlInput("application/json", url, true);
```

You can then display the results:

```
    Console.WriteLine(
       JsonConvert.SerializeObject(imageData, Formatting.Indented)
       );
```

Content Moderator isn't only for images; it can also work with text content. The text moderation service identifies more than just adult and racy content; it will also scan for offensive language and personal information such as Social Security numbers. Here is a code snippet showing how to run text moderation over a sample set of text.

First create and initialize an instance of the Content Moderator API wrapper, using your own subscription key and the correct endpoint:

```
    ContentModeratorClient client =
     new ContentModeratorClient(
      new ApiKeyServiceClientCredentials("<Your Subscription Key>")
      );
    client.Endpoint = "<Your Service Endpoint>";
```

Now, send the text to evaluate:

```
    string text = "Is this a garbage or crap email abcdef@abcd.com,
    phone: 6657789887, IP: 255.255.255.255, 1 Microsoft Way,
    Redmond, WA 98052. These are all UK phone numbers, the last
    two being Microsoft UK support numbers: +44 870 608 4000 or
    0344 800 2400 or 0800 820 3300. Also, 999-99-9999 looks like
    a social security number (SSN).";
```

Before the content can be sent to the service, you need to prepare the text and convert it to bytes:

```
text = text.Replace(System.Environment.NewLine, " ");
byte[] byteArray = System.Text.Encoding.UTF8.GetBytes(text);
MemoryStream stream = new MemoryStream(byteArray);
```

The API can evaluate the text for typos, matching terms, and personally identifying information (PII), and classify it:

```
var screenResult = client.TextModeration.ScreenText(
        "text/plain", stream, "eng", true, true, null, true
    );
```

You can then display the results:

```
Console.WriteLine(JsonConvert.SerializeObject(screenResult,
    Formatting.Indented));
```

A typical JSON response for checking for PII would look like this:

```
"PII": {
    "Email": [{
        "Detected": "abcdef@abcd.com",
        "SubType": "Regular",
        "Text": "abcdef@abcd.com",
        "Index": 32
        }],
    "IPA": [{
        "SubType": "IPV4",
        "Text": "255.255.255.255",
        "Index": 72
        }],
    "Phone": [{
        "CountryCode": "US",
        "Text": "6657789887",
        "Index": 56
        }, {
        "CountryCode": "US",
        "Text": "870 608 4000",
        "Index": 212
        }, {
        "CountryCode": "UK",
        "Text": "+44 870 608 4000",
        "Index": 208
        }, {
        "CountryCode": "UK",
        "Text": "0344 800 2400",
        "Index": 228
        }, {
```

```
        "CountryCode": "UK",
        "Text": "0800 820 3300",
        "Index": 245
        }],
    "Address": [{
        "Text": "1 Microsoft Way, Redmond, WA 98052",
        "Index": 89
        }],
    "SSN": [{
        "Text": "999999999",
        "Index": 56
        }, {
        "Text": "999-99-9999",
        "Index": 267
        }]
    }
```

CHAPTER 8

Web Search

The Cognitive Services Search APIs enable you to bring the power of Bing's search capabilities to your applications. Through these APIs you can deliver instant answers to user queries. There are some restrictions on the usage of the Bing Search APIs: you cannot copy or store data from the service unless it's as part of a limited search history, nor can you use the results as training data for machine learning services. Any API results must also be used for internet searches, and any use of the data has to be initiated by a user.

If you want to search your own content, Azure Search provides a comprehensive search-as-a-service cloud solution.

Bing Web Search

The Bing Web Search API is a simple way of adding ad-free search to your websites or applications. The service is easily configurable, giving you control over how your users search and what responses are delivered. Bing Web Search uses the full Bing index, so it returns text, images, news, and video. The service also provides type-specific endpoints if you want to limit the responses to specific categories.

Other API options set search priorities familiar from any major search engine: you can manage the safety level of responses, control the number of results delivered, and set the time period you're interested in indexed documents from. If you want to get the most appropriate results for your region, you can either set a country and language individually or use the “market” option to set both automatically.

The following code snippet shows how to set up a web search endpoint and then run a query with version 2 of the C# SDK.

First, initialize the client:

```
var client = new WebSearchClient(
  new ApiKeyServiceClientCredentials("<Your Subscription Key>"),
  new System.Net.Http.DelegatingHandler[] { }
  );
client.Endpoint = "<Your Service Endpoint>";
```

And perform a web search:

```
private const string query = "Yosemite National Park";
SearchResponse webData = await client.Web.SearchAsync(webQuery);
```

Now you can extract data from the response. Here’s an example of pulling out the first web page result:

```
var firstWebPagesResult =
  webData.WebPages.Value.FirstOrDefault();
Console.WriteLine("Webpage Results # {0}",
  webData.WebPages.Value.Count);
Console.WriteLine("First web page name: {0} ",
  firstWebPagesResult.Name);
Console.WriteLine("First web page URL: {0} ",
  firstWebPagesResult.Url);
```

If you have a paid subscription to the Bing APIs, you get access to Bing Statistics, which shows you a dashboard on Azure with details of your users’ searches. You can filter to see search patterns in different apps or different markets, over specific time periods. Use the data in the reports to fine-tune filters in your code, or to understand what your users are interested in.

Bing Custom Search

If you want a lot more control over search results for specific topics, Bing Custom Search lets you create a tailored search experience for your users. Using the Custom Search portal, you can choose the domains, sites, and even pages that are searched or removed. The portal lets you tune results, applying weighting to search rankings, and generate a search UI that can be dropped straight into your code or web content.

Once you've created a Custom Search instance, you can make REST queries against its endpoints for general web search, custom image or video search, and autosuggestions for partial queries.

Bing Visual Search

The Bing Visual Search API lets your users search based on images they find online or take with their device's camera. This service provides an experience like that found on Bing.com/Images. Running a visual search is much the same as running any other Bing search: the process and flow are largely the same. The API can also identify a variety of details about an image you upload, including visually similar images and web pages that include that image.

Here's a short snippet showing you how to call the Visual Search API. First you will need to create a client:

```
var client = new VisualSearchClient(
   new ApiKeyServiceClientCredentials("Your Subscription Key")
   )
{Endpoint = "<Your Service Endpoint"};
```

Then provide the image you would like to use in the search. Here we are going to load the image from a file:

```
System.IO.FileStream stream = new FileStream(
       Path.Combine("TestImages", "image.jpg")), FileMode.Open
   );
var visualSearchResults =
   await client.Images.VisualSearchMethodAsync(
   image: stream, knowledgeRequest: (string)null
);
```

We can display the image insights token from the result:

```
Console.WriteLine($"Uploaded image insights token:" +
  $"{visualSearchResults.Image.ImageInsightsToken}");
```

Or we can display the tags from the first returned result:

```
var firstTagResult = visualSearchResults.Tags[0];
Console.WriteLine(
  $"Visual search tag count:" +
  $"{visualSearchResults.Tags.Count}");
```

Bing Autosuggest

All the Bing APIs support Autosuggest for autocompleting search queries. The Autosuggest API lets you send a partial search query term to Bing and get back a list of suggested queries that other users have searched on. The API is easy to integrate into your workflow: simply make a call whenever a user types a new character into your application's search box. As the user provides more information, the API will return more relevant suggestions.

You can use Autosuggest in conjunction with other Cognitive Services APIs, for example recognizing objects or extracting text from a photograph with the Computer Vision APIs and then using Autosuggest to find relevant search terms to get more information.

Bing Video and News Search

The Bing Video and News Search APIs are specialized versions of the standard Web Search API that return ad-free results for videos or news items related to the provided search query. As well as general news results, the News Search API can deliver top headlines or news stories, news in a specific category, or trending news (for the US and China only).

The following code runs a news query for the term "quantum computing." First, create a client:

```
var client = new NewsSearchClient(
   new ApiKeyServiceClientCredentials("Your Subscription Key")
   )
{Endpoint = "Your Service Endpoint"};
```

Then call the News Search API and write out the name of the first result returned:

```
var searchTerm = "Quantum Computing";
var newsResults = await client.News.SearchAsync(
        query: searchTerm, market: "en-us", count: 10
  );
```

You can also display a link to the first image:

```
Console.WriteLine($"Name from first news result returned:" +
    $"{newsResults.Value[0].Name}");
```

The Video Search API can return thumbnail URLs and the HTML needed to display an embedded video player for quick access to video content, as well as insights like related videos.

The following code implements a video search, looking for videos about the SwiftKey mobile device keyboard software:

```
var client = new VideoSearchClient(
   new ApiKeyServiceClientCredentials("Your Subscription Key")
   )
{Endpoint = "Your Service Endpoint"};
var videoResults =
   await client.Videos.SearchAsync(query: "SwiftKey");
```

And you can show the content from the first video returned:

```
var firstVideoResult = videoResults.Value[0];
Console.WriteLine(
  $"\r\nVideo result count: {videoResults.Value.Count}");
Console.WriteLine(
  $"First video id: {firstVideoResult.VideoId}");
Console.WriteLine(
  $"First video name: {firstVideoResult.Name}");
Console.WriteLine(
  $"First video url: {firstVideoResult.ContentUrl}");
```

Bing Entity Search

Text analytics is about extracting meaning—but much of what humans say is subtext or shared knowledge. This makes it hard to infer meaning programmatically. The Text Analytics service can recognize entities in text, but if you want to tag places, businesses, and other well-known things, you'll want to look at the Bing Entity Search service, which lets you identify well-known entities in a given search query.

The service has many features, including being able to identify the most relevant entity based on the search term. For example, if a user searches for "Mount Rainier," you may want to differentiate between the national park and the mountain. If you can extract key phrases with the Text Analytics API to show what kind of entity a user is referring to (e.g., a book, a movie, or a character), passing that with the query can also help get the most relevant result.

Some responses are specific to certain types of query. For example, searching for "coffee shop" might generate a list of nearby venues. This adds additional information about entities, such as addresses or contact numbers. You can then use that data with additional services, such as Azure Maps, to plot locations on a map and provide routing details.

It's important to note that the returned data may contain additional contractual rules about the data. These rules may include image attribution, or content licenses. Similarly, you many need to apply provider information and other licensing information around text content, as the Bing index contains third-party information as well as public data.

The following code snippet shows you how to get started with entity search using the C# SDK. First, you need to instantiate your client:

```
var client =
  new EntitySearchClient(
   new ApiKeyServiceClientCredentials("<Your Subscription Key>")
   )
  {Endpoint = "<Your Resource Endpoint>"};
```

Then send your query using the `Entities.Search()` function. The method will return an `entityData` object that can either be delivered to the end user as part of an application output or parsed with specific elements used in your code:

```
var entityData =
  await client.Entities.SearchAsync(query: "Satya Nadella");
```

Next, filter the sequence to find the main entity:

```
var mainEntity = entityData.Entities.Value.Where(
     thing => thing.EntityPresentationInfo.EntityScenario ==
     EntityScenario.DominantEntity).FirstOrDefault();
```

You can then display the description of the main entity:

```
Console.WriteLine(mainEntity.Description);
```

Bing Local Business Search

While Bing Entity Search lets you look up people, objects, and places, sometimes you only want to give your customers information about relevant local businesses. Bing Local Business Search lets you get these localized results, enabling useful searches like "find restaurants near me" within your application. The responses include information such as the name, address, phone number, and website URL for each business.

Azure Search and Cognitive Search

The Bing APIs are for when you want to search the web. If you want to search your own content, you'll want to look at Azure Search, which provides a fully managed cloud service for building and managing your own search index.

The recently added Cognitive Search feature enables you to leverage AI to extract information from images, documents, and other unstructured data sources. By applying Cognitive Services features like natural language support, entity recognition, and text analytics, along with image processing, you can make your custom search engine much more powerful.

Cognitive Search works by adding AI skills as part of the indexing pipeline. The first step involves "cracking" the source data and extracting text. In addition to searching text-based content like PDFs and Word or PowerPoint documents, it uses AI to extract text from images. The extracted text can then be passed through a pipeline of *skillsets*, each made up of specific cognitive actions that add new fields to the index that aren't found in the source text.

Skillsets can be simple or complex. New fields are added to the source documents, enriching them with, for example, entity relationships. When all these skills are combined, you can turn a large set of documents into a graph of information, linked by people, products, and other entities.

Once your documents have been enriched and indexed, they can be searched using the Azure Search APIs. You can also save the enriched documents and metadata into a knowledge store. In a standard AI-based pipeline, enriched documents are transitory, used only during indexing and then discarded. With a knowledge store, these enrichments are then saved for subsequent evaluation and exploration.

Azure Search combined with the Cognitive Services allows you to look more deeply inside your content. You can make sure that all the useful information is extracted, indexed, and available to your users.

CHAPTER 9

Paving the Road Ahead

Five decades ago, the early inventors in AI could only dream of what most consumers take for granted today. From smartphones to voice-powered assistants like Cortana, Siri, or Alexa and self-driving cars, we seem to be living in the pages of a sci-fi book. What do the next two decades hold for us? What about the next five? Microsoft has made it its mission to advance AI innovation by democratizing AI tools in the same way that it democratized the power of computing in the mainframe era by envisioning a personal computer in every home, school, and workplace.

Artificial intelligence is rapidly becoming a mainstream technology that is helping transform and empower us in unexpected ways. Here are just a few highlights.

Schneider Electric

Schneider Electric, a French multinational manufacturing company in the energy sector, is now using new technologies like AI and the IoT to transform the breadth of its offering, and the same versatility that has long powered the company's growth now defines its AI strategy. Companies that survive for more than a century, as Schneider Electric has, share a key trait: an ability to adapt. After all, no enterprise can last that long without anticipating and planning for the challenges that will shape its journey. In Schneider's case, a business that began as a steel manufacturing firm in France at the height of the first Industrial Revolution has strategically transformed itself into a twenty-first-century global leader in digital energy management and industrial automation. One of the most striking things about Schneider's business today is the variety of industries it touches: healthcare, banking, education, commercial buildings, the smart grid, construction, oil and gas, food and beverages, transportation, clean energy, and more.

Seeing AI

Microsoft aims to develop AI systems that will empower people worldwide to more effectively address local and global challenges, and to help drive progress and economic opportunity. One example of how AI can make a difference is a recent Microsoft offering called Seeing AI (see Figure 9-1), available in the iOS app store, that can assist people with blindness and low vision as they navigate daily life.

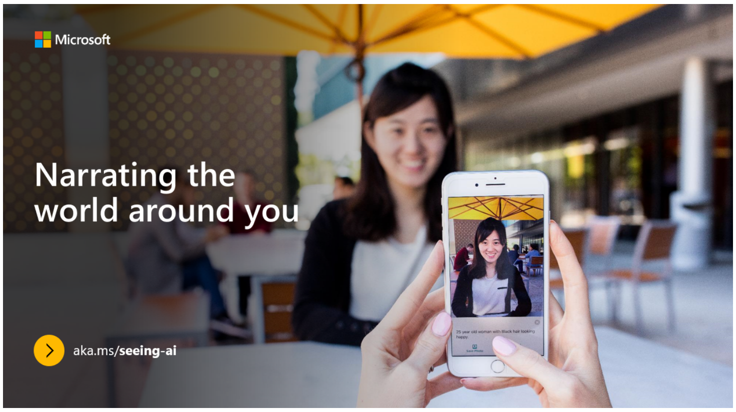


Figure 9-1. Seeing AI is a free app that narrates the world around you; it is designed for the low-vision community and harnesses the power of AI to describe people, text, and objects

Seeing AI was developed by a team that included a Microsoft engineer who lost his sight when he was seven years old. This powerful application demonstrates the potential for AI to empower people with disabilities by capturing images from the user's surroundings and instantly describing what is happening. For example, it can read signs and menus, recognize products through barcodes, interpret handwriting, count currency, describe scenes and objects in the vicinity, or, during a meeting, tell the user that a man and a woman are sitting across the table, smiling and paying close attention.

AI Ethics and Microsoft's Principles

Beyond paving the path by providing the tools and resources to democratize AI, Microsoft is also engaging with the challenging questions these powerful new technologies are forcing us to confront. The philosopher Marshall McLuhan said, "We become what we behold. We shape our tools and then our tools shape us." How do we ensure that AI is designed and used responsibly? How do we establish ethical principles to protect people? How should we govern its use? And how will AI affect employment and jobs? These questions cannot be answered by technologists alone; it is a societal responsibility that bears discussions across government, academia, business, civil society, and other stakeholders. Microsoft recently published a book, *The Future Computed: Artificial Intelligence and Its Role in Society*, that addresses the larger issues governing AI and the future. It also outlines the six guiding ethical principles Microsoft has identified for the cross-disciplinary development and use of artificial intelligence.

Fairness

The first principle underscores issues of social justice. How do we design AI systems to treat everyone in a fair and balanced way, when the training data we use might be marred with assumptions and biases? Can an AI system that provides guidance on loan applications or employment, for instance, be designed to make the same recommendations for everyone with similar financial circumstances or professional qualifications? As a developer, you will need to be cognizant of how biases can be introduced into the system, work on creating a representational data set, and, beyond that, educate end users to understand that sound human judgment must complement computer system recommendations to counter their inherent limitations.

Reliability and Safety

As we become more dependent on AI systems, how can we ensure that our systems can operate safely, reliably, and consistently? Consider the failure of an AI-powered system in a hospital, which could literally mean the difference between the lives and deaths of people. We must design carefully and conduct rigorous testing under various conditions, incorporating security considerations on how to counter cyberattacks and other malicious intents. Sometimes systems might react unexpectedly, depending on the data. Microsoft had a painful example of unexpected behavior when it unveiled Tay, a conversational chatbot on Twitter. Tay was an experiment in conversational AI that quickly went wrong when users began feeding the bot racist and sexist content that it quickly learned and reflected back to users. Microsoft took Tay down within 24 hours. Developers must teach end users what the expected behaviors within normal conditions are so that when things go wrong humans can quickly intervene to minimize the damage.

Privacy and Security

As our lives become more digitized, privacy and security take on additional significance. This discussion goes beyond what technologies are used to ensure the security of data; it must include regulations around how data is used and for what reasons.

Inclusivity

We want to ensure AI systems empower and engage people across the spectrum of abilities and access levels. AI-enabled services are already empowering people struggling with visual, hearing, and cognitive disabilities. Building systems that are context-aware with increasing emotional intelligence will pave the path for more empowerment.

Transparency

Designers and developers of AI systems need to create systems with maximum transparency. People need to understand how important decisions regarding their lives are made.

Accountability

The people who design and deploy AI systems must be accountable for how their systems operate. Companies should establish accountability norms for AI along with internal review boards to provide the necessary guidance and oversight on the systems they oversee.

Microsoft is committed to empowering you with the tools, ethical framework, and best practices to foster responsible development of AI systems that will enrich human lives and power innovations, which will in turn solve our most pressing problems today and help us anticipate the ones to come in the future. Finally, it's important to remember that while we often get bogged down in discussions about the exciting algorithms and tools, the real power of AI resides in the ideas and questions that precede it. It's the conservationist pondering how to create sustainable habitats, the doctor wondering how to better serve a patient, the astronomer's and citizen scientist's curiosity that expands our collective consciousness to the outer limits of the universe. AI has the potential to empower the noblest of human causes, and we are just at the very beginning of an exciting technological transformation.

CHAPTER 10

Where to Go Next

This report has introduced you to the key features and capabilities of Cognitive Services, along with some ideas of what you could do with them to make your apps smarter, more useful, easier to use, or just more fun. Cognitive Search is just one example of a powerful feature you can build by integrating the Cognitive Services with other products.

You may find the following sites helpful in your journey:

- Microsoft documentation: *https://docs.microsoft.com/en-us/azure/cognitive-services/*
- Complete code samples: *https://github.com/Azure-Samples/cognitive-services-dotnet-sdk-samples*
- Intelligent Kiosk demo: *https://github.com/Microsoft/Cognitive-Samples-IntelligentKiosk*
- Interactive demos: *https://azure.microsoft.com/en-us/services/cognitive-services/directory*

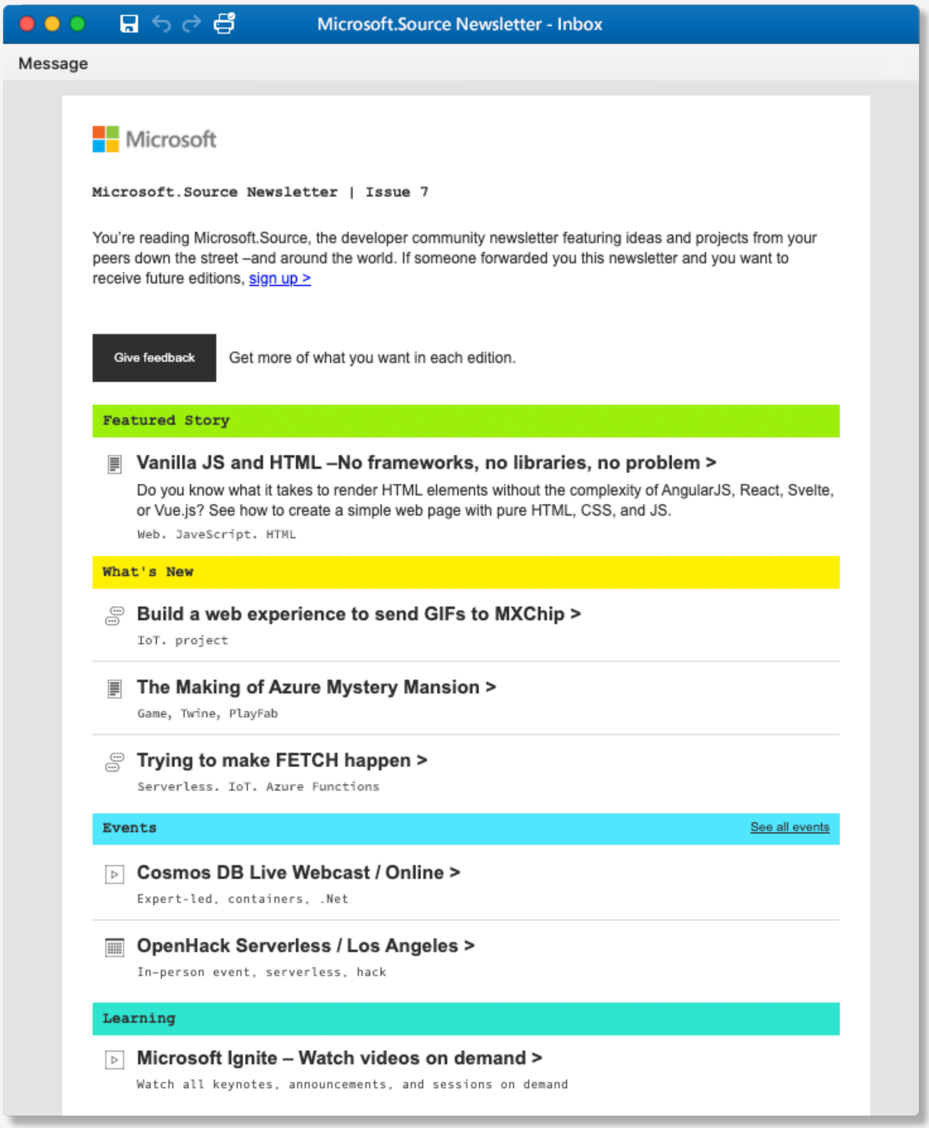
The Microsoft AI School also has online, interactive modules with QuickStarts, hands-on tutorials, and step-by-step guides for several of the Cognitive Services at *https://aischool.microsoft.com/en-us/services/learning-paths*.

About the Authors

Anand Raman is a senior director of product management for AI Platform at Microsoft. Previously he was the TA and chief of staff for Microsoft AI, covering Azure Data Group and Machine Learning. In the last decade, he ran the engineering and product management teams at Azure Data Services, Visual Studio, and Windows Server User Experience at Microsoft. Anand holds a PhD in computational fluid mechanics and worked several years as a researcher before joining Microsoft.

Chris Hoder is a program manager on the Cognitive Services team at Microsoft. Chris focuses on the end-to-end developer experience across the entire suite of services—from our API and SDK designs to the getting started documentation. In prior roles, Chris worked directly with customers to envision, design, build, and deploy AI-focused applications using Microsoft's AI stack.

Microsoft


Microsoft.Source Newsletter - Inbox

Message

Microsoft

Microsoft.Source Newsletter | Issue 7

You're reading Microsoft.Source, the developer community newsletter featuring ideas and projects from your peers down the street –and around the world. If someone forwarded you this newsletter and you want to receive future editions, sign up >

Give feedback Get more of what you want in each edition.

Featured Story

Vanilla JS and HTML –No frameworks, no libraries, no problem >
Do you know what it takes to render HTML elements without the complexity of AngularJS, React, Svelte, or Vue.js? See how to create a simple web page with pure HTML, CSS, and JS.
Web. JavaScript. HTML

What's New

Build a web experience to send GIFs to MXChip >
IoT. project

The Making of Azure Mystery Mansion >
Game, Twine, PlayFab

Trying to make FETCH happen >
Serverless. IoT. Azure Functions

Events See all events

Cosmos DB Live Webcast / Online >
Expert-led, containers, .Net

OpenHack Serverless / Los Angeles >
In-person event, serverless, hack

Learning

Microsoft Ignite – Watch videos on demand >
Watch all keynotes, announcements, and sessions on demand

By developers, for developers

Microsoft.Source newsletter

Get technical articles, sample code, and information on upcoming events in Microsoft.Source, the curated monthly developer community newsletter.

- Keep up on the latest technologies
- Connect with your peers at community events
- Learn with hands-on resources

Sign up