

# Table of Contents

## Computer Vision Documentation

### Overview

### How-to

- Obtain subscription keys

- Call the Computer Vision API

- Analyze videos in real time

### Quickstarts

- cURL

- C#

- Java

- JavaScript

- PHP

- Python

- Ruby

### Tutorials

- C#

- Java

- JavaScript

- Python

### Reference

- API reference

- SDKs

  - Android

  - Swift

  - Windows

### Resources

- Azure Roadmap

- Samples

- Category taxonomy

[FAQ](#)

[Research papers](#)

# Computer Vision Documentation

The cloud-based Computer Vision API provides developers with access to advanced algorithms for processing images and returning information. By uploading an image or specifying an image URL, Microsoft Computer Vision algorithms can analyze visual content in different ways based on inputs and user choices. Learn how to analyze visual content in different ways with our quickstarts, tutorials, and samples.

## 5-Minute Quickstarts

Analyze images, generate thumbnails, and extract text from an image using:

[C#](#)

[cURL](#)

[Java](#)

[JavaScript](#)

[PHP](#)

[Python](#)

[Ruby](#)

## Step-by-Step Tutorials

Develop applications using the Computer Vision API:

1. [C# Tutorial](#)
2. [Python Tutorial](#)

## Reference

### APIs

[API Reference](#)

### SDKs

[Android](#)

[Swift](#)

[Windows](#)

# Computer Vision API Version 1.0

12/7/2017 • 9 min to read • [Edit Online](#)

The cloud-based Computer Vision API provides developers with access to advanced algorithms for processing images and returning information. By uploading an image or specifying an image URL, Microsoft Computer Vision algorithms can analyze visual content in different ways based on inputs and user choices. With the Computer Vision API users can analyze images to:

- [Tag images based on content.](#)
- [Categorize images.](#)
- [Identify the type and quality of images.](#)
- [Detect human faces and return their coordinates.](#)
- [Recognize domain-specific content.](#)
- [Generate descriptions of the content.](#)
- [Use optical character recognition to identify printed text found in images.](#)
- [Recognize handwritten text.](#)
- [Distinguish color schemes.](#)
- [Flag adult content.](#)
- [Crop photos to be used as thumbnails.](#)

## Requirements

- Supported input methods: Raw image binary in the form of an application/octet stream or image URL.
- Supported image formats: JPEG, PNG, GIF, BMP.
- Image file size: Less than 4 MB.
- Image dimension: Greater than 50 x 50 pixels.

## Tagging Images

Computer Vision API returns tags based on more than 2000 recognizable objects, living beings, scenery, and actions. When tags are ambiguous or not common knowledge, the API response provides 'hints' to clarify the meaning of the tag in context of a known setting. Tags are not organized as a taxonomy and no inheritance hierarchies exist. A collection of content tags forms the foundation for an image 'description' displayed as human readable language formatted in complete sentences. Note, that at this point English is the only supported language for image description.

After uploading an image or specifying an image URL, Computer Vision API's algorithms output tags based on the objects, living beings, and actions identified in the image. Tagging is not limited to the main subject, such as a person in the foreground, but also includes the setting (indoor or outdoor), furniture, tools, plants, animals, accessories, gadgets etc.

### Example



Returned Json

```
{
  'tags':[
    {
      "name": "grass",
      "confidence": 0.999999761581421
    },
    {
      "name": "outdoor",
      "confidence": 0.999970674514771
    },
    {
      "name": "sky",
      "confidence": 0.99289751052856
    },
    {
      "name": "building",
      "confidence": 0.996463239192963
    },
    {
      "name": "house",
      "confidence": 0.992798030376434
    },
    {
      "name": "lawn",
      "confidence": 0.822680294513702
    },
    {
      "name": "green",
      "confidence": 0.641222536563873
    },
    {
      "name": "residential",
      "confidence": 0.314032256603241
    },
  ],
}
```

## Categorizing Images

In addition to tagging and descriptions, Computer Vision API returns the taxonomy-based categories defined in previous versions. These categories are organized as a taxonomy with parent/child hereditary hierarchies. All categories are in English. They can be used alone or with our new models.

### The 86-category concept

Based on a list of 86 concepts seen in the following diagram, visual features found in an image can be categorized ranging from broad to specific. For the full taxonomy in text format, see [Category Taxonomy](#).

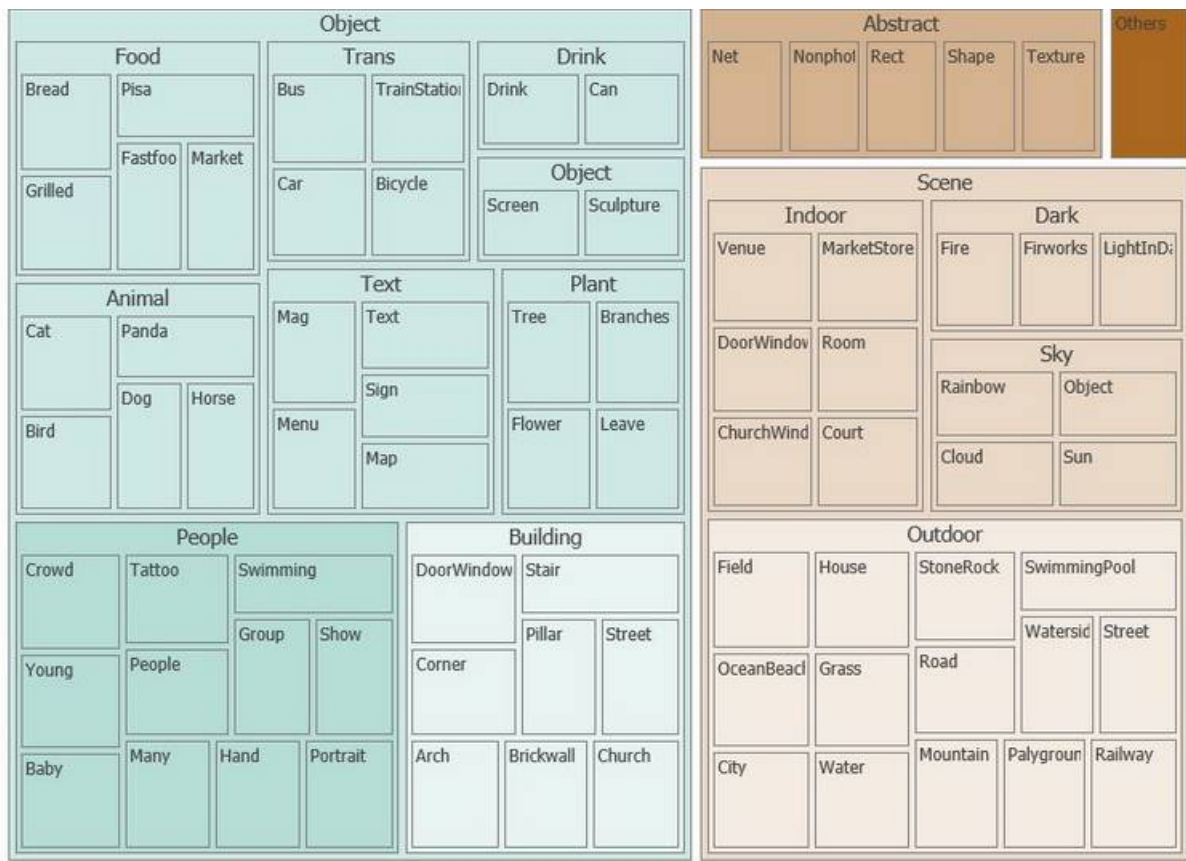







IMAGE	RESPONSE
	people
	people_crowd

IMAGE	RESPONSE
	animal_dog
	outdoor_mountain
	food_bread



## Identifying Image Types

There are several ways to categorize images. Computer Vision API can set a boolean flag to indicate whether an image is black and white or color. It can also set a flag to indicate whether an image is a line drawing or not. It can also indicate whether an image is clip art or not and indicate its quality as such on a scale of 0-3.

### Clip-art type



Detects whether an image is clip art or not.

VALUE	MEANING
0	Non-clip-art
1	ambiguous
2	normal-clip-art
3	good-clip-art

IMAGE	RESPONSE
	3 good-clip-art
	0 Non-clip-art

### Line drawing type

Detects whether an image is a line drawing or not.


IMAGE	RESPONSE
	True
	False

### Faces

Detects human faces within a picture and generates the face coordinates, the rectangle for the face, gender, and age. These visual features are a subset of metadata generated for face. For more extensive metadata generated for faces



(facial identification, pose detection, and more), use the Face API.

IMAGE	RESPONSE
	[ { "age": 23, "gender": "Female", "faceRectangle": { "left": 1379, "top": 320, "width": 310, "height": 310 } } ]
	[ { "age": 28, "gender": "Female", "faceRectangle": { "left": 447, "top": 195, "width": 162, "height": 162 } }, { "age": 10, "gender": "Male", "faceRectangle": { "left": 355, "top": 87, "width": 143, "height": 143 } } ]
	[ { "age": 11, "gender": "Male", "faceRectangle": { "left": 113, "top": 314, "width": 222, "height": 222 } }, { "age": 11, "gender": "Female", "faceRectangle": { "left": 1200, "top": 632, "width": 215, "height": 215 } }, { "age": 41, "gender": "Male", "faceRectangle": { "left": 514, "top": 223, "width": 205, "height": 205 } }, { "age": 37, "gender": "Female", "faceRectangle": { "left": 1008, "top": 277, "width": 201, "height": 201 } } ]

## Domain-Specific Content

In addition to tagging and top-level categorization, Computer Vision API also supports specialized (or domain-specific) information. Specialized information can be implemented as a standalone method or with the high-level categorization. It functions as a means to further refine the 86-category taxonomy through the addition of domain-specific models.

Currently, the only specialized information supported are celebrity recognition and landmark recognition. They are domain-specific refinements for the people and people group categories, and landmarks around the world.

There are two options for using the domain-specific models:

### Option One - Scoped Analysis

Analyze only a chosen model, by invoking an HTTP POST call. For this option, if you know which model you want to use, you specify the model's name, and you only get information relevant to that model. For example, you can use this option to only look for celebrity-recognition. The response contains a list of potential matching celebrities, accompanied by their confidence scores.

### Option Two - Enhanced Analysis

Analyze to provide additional details related to categories from the 86-category taxonomy. This option is available for use in applications where users want to get generic image analysis in addition to details from one or more domain-specific models. When this method is invoked, the 86-category taxonomy classifier is called first. If any of the categories match that of known/matching models, a second pass of classifier invocations follows. For example, if 'details=all' or "details" include 'celebrities', the method calls the celebrity classifier after the 86-category classifier is called. The result includes tags starting with 'people\_'.

## Generating Descriptions

Computer Vision API's algorithms analyze the content in an image. This analysis forms the foundation for a 'description' displayed as human-readable language in complete sentences. The description summarizes what is found in the image. Computer Vision API's algorithms generate various descriptions based on the objects identified in the image. The descriptions are each evaluated and a confidence score generated. A list is then returned ordered from highest confidence score to lowest. An example of a bot that uses this technology to generate image captions can be found [here](#).

### Example Description Generation



Returned Json

```
'description':{
  "captions":[
    {
      "type": "phrase",
      'text': 'a black and white photo of a large city',
      'confidence': 0.607638706850331
    }
  ]
  "captions":[
    {
      "type": "phrase",
      'text': 'a photo of a large city',
      'confidence': 0.577256764264197
    }
  ]
  "captions":[
    {
      "type": "phrase",
      'text': 'a black and white photo of a city',
      'confidence': 0.538493271791207
    }
  ]
'description':[
  "tags":{
    "outdoor",
    "city",
    "building",
    "photo",
    "large",
  }
]
```

## Perceiving Color Schemes

The Computer Vision algorithm extracts colors from an image. The colors are analyzed in three different contexts: foreground, background, and whole. They are grouped into twelve 12 dominant accent colors. Those accent colors are black, blue, brown, gray, green, orange, pink, purple, red, teal, white, and yellow. Depending on the colors in an image, simple black and white or accent colors may be returned in hexadecimal color codes.







IMAGE	FOREGROUND	BACKGROUND	COLORS
	Black	Black	White
	Black	White	White, Black, Green

IMAGE	FOREGROUND	BACKGROUND	COLORS
	Black	Black	Black



**Accent color**

Color extracted from an image designed to represent the most eye-popping color to users via a mix of dominant colors and saturation.

IMAGE	RESPONSE
	#BC6F0F
	#CAA501
	#484B83

**Black & White**

Boolean flag that indicates whether an image is black&white or not.

IMAGE	RESPONSE
	True
	False

## Flagging Adult Content

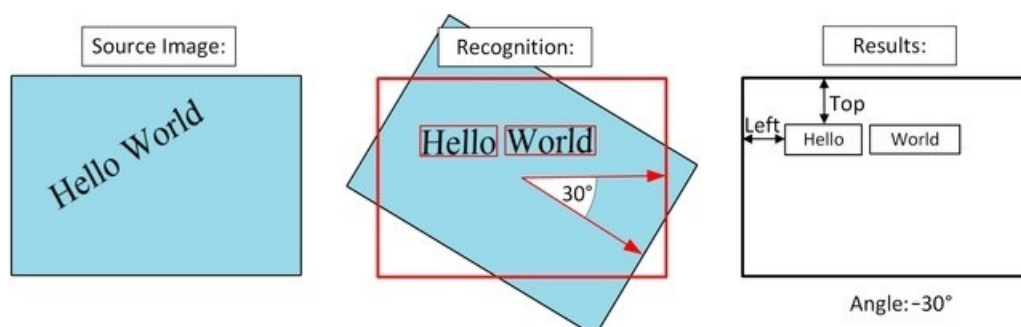
Among the various visual categories is the adult and racy group, which enables detection of adult materials and restricts the display of images containing sexual content. The filter for adult and racy content detection can be set on a sliding scale to accommodate the user's preference.

## Optical Character Recognition (OCR)

OCR technology detects text content in an image and extracts the identified text into a machine-readable character stream. You can use the result for search and numerous other purposes like medical records, security, and banking. It automatically detects the language. OCR saves time and provides convenience for users by allowing them to take photos of text instead of transcribing the text.

OCR supports 25 languages. These languages are: Arabic, Chinese Simplified, Chinese Traditional, Czech, Danish, Dutch, English, Finnish, French, German, Greek, Hungarian, Italian, Japanese, Korean, Norwegian, Polish, Portuguese, Romanian, Russian, Serbian (Cyrillic and Latin), Slovak, Spanish, Swedish, and Turkish.

If needed, OCR corrects the rotation of the recognized text, in degrees, around the horizontal image axis. OCR provides the frame coordinates of each word as seen in below illustration.



Requirements for

OCR:

- The size of the input image must be between 40 x 40 and 3200 x 3200 pixels.
- The image cannot be bigger than 10 megapixels.

Input image can be rotated by any multiple of 90 degrees plus a small angle of up to '40 degrees.

The accuracy of text recognition depends on the quality of the image. An inaccurate reading may be caused by the following situations:

- Blurry images.
- Handwritten or cursive text.
- Artistic font styles.
- Small text size.
- Complex backgrounds, shadows, or glare over text or perspective distortion.
- Oversized or missing capital letters at the beginnings of words
- Subscript, superscript, or strikethrough text.

Limitations: On photos where text is dominant, false positives may come from partially recognized words. On some photos, especially photos without any text, precision can vary a lot depending on the type of image.

## Recognize Handwritten Text

This technology allows you to detect and extract handwritten text from notes, letters, essays, whiteboards, forms, etc. It works with different surfaces and backgrounds, such as white paper, yellow sticky notes, and whiteboards.

Handwritten text recognition saves time and effort and can make you more productive by allowing you to take images of text, rather than having to transcribe it. It makes it possible to digitize notes. This digitization allows you to implement quick and easy search. It also reduces paper clutter.

Input requirements:

- Supported image formats: JPEG, PNG, and BMP.
- Image file size must be less than 4 MB.
- Image dimensions must be at least 40 x 40, at most 3200 x 3200.

Note: this technology is currently in preview and is only available for English text.

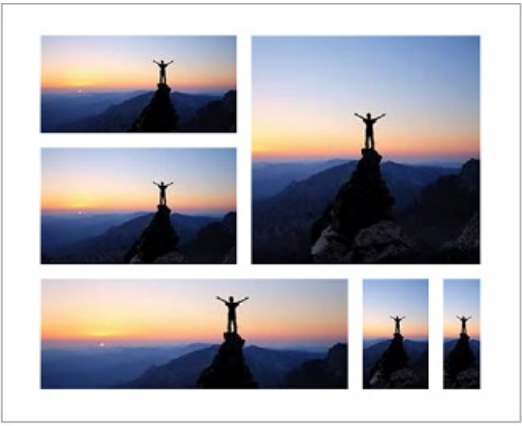
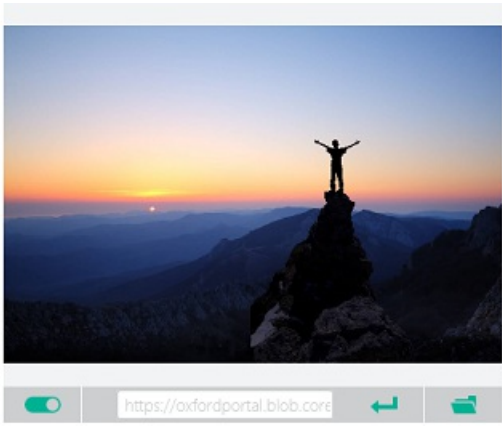
## Generating Thumbnails

A thumbnail is a small representation of a full-size image. Varied devices such as phones, tablets, and PCs create a need for different user experience (UX) layouts and thumbnail sizes. Using smart cropping, this Computer Vision API feature helps solve the problem.

After uploading an image, a high-quality thumbnail gets generated and the Computer Vision API algorithm analyzes the objects within the image. It then crops the image to fit the requirements of the 'region of interest' (ROI). The output gets displayed within a special framework as seen in below illustration. The generated thumbnail can be presented using an aspect ratio that is different from the aspect ratio of the original image to accommodate a user's needs.

The thumbnail algorithm works as follows:

1. Removes distracting elements from the image and recognizes the main object, the 'region of interest' (ROI).
2. Crops the image based on the identified region of interest.
3. Changes the aspect ratio to fit the target thumbnail dimensions.





# Obtaining Subscription Keys

6/27/2017 • 1 min to read • [Edit Online](#)

Computer Vision services require special subscription keys. Every call to the Computer Vision API requires a subscription key. This key needs to be either passed through a query string parameter or specified in the request header.

To sign up for subscription keys, see [Subscriptions](#). It's free to sign up. Pricing for these services is subject to change.

## NOTE

Your subscription keys are valid for only one of these [Microsoft Azure Regions](#).

REGION	ADDRESS
West US	westus.api.cognitive.microsoft.com
East US 2	eastus2.api.cognitive.microsoft.com
West Central US	westcentralus.api.cognitive.microsoft.com
West Europe	westeurope.api.cognitive.microsoft.com
Southeast Asia	southeastasia.api.cognitive.microsoft.com

If you sign up using the Computer Vision free trial, your subscription keys are valid for the **westcentral** region ( `https://westcentralus.api.cognitive.microsoft.com/vision/v1.0/` ). That is the most common case. However, if you sign-up for Computer Vision with your Microsoft Azure account through the <https://azure.microsoft.com/> website, you specify the region for your trial from the preceding list of regions.

For example, if you sign up for Computer Vision with your Microsoft Azure account and you specify `westus` for your region, you must use the `westus` region for your REST API calls ( `https://westus.api.cognitive.microsoft.com/vision/v1.0/` ).

If you forget the region for your subscription key after obtaining your trial key, you can find your region at <https://azure.microsoft.com/try/cognitive-services/my-apis/>.

Subscriptions

## Related Links:

- [Pricing Options for Microsoft Cognitive APIs](#)



# How to Call Computer Vision API

6/27/2017 • 5 min to read • [Edit Online](#)

This guide demonstrates how to call Computer Vision API using REST. The samples are written both in C# using the Computer Vision API client library, and as HTTP POST/GET calls. We will focus on:

- How to get "Tags", "Description" and "Categories".
- How to get "Domain-specific" information (celebrities).

Image URL or path to locally stored image.

- Supported input methods: Raw image binary in the form of an application/octet stream or image URL
- Supported image formats: JPEG, PNG, GIF, BMP
- Image file size: Less than 4MB
- Image dimension: Greater than 50 x 50 pixels

In the examples below, the following features are demonstrated:

1. Analyzing an image and getting an array of tags and a description returned.
2. Analyzing an image with a domain-specific model (specifically, "celebrities" model) and getting the corresponding result in JSON return.

Features are broken down on:

- **Option One:** Scoped Analysis - Analyze only a given model
- **Option Two:** Enhanced Analysis - Analyze to provide additional details with [86-categories taxonomy](#)

Every call to the Computer Vision API requires a subscription key. This key needs to be either passed through a query string parameter or specified in the request header.

To obtain a subscription key, see [How to Obtain Subscription Keys](#).

1. Passing the subscription key through a query string, see below as a Computer Vision API example:

```
https://westus.api.cognitive.microsoft.com/vision/v1.0/analyze?visualFeatures=Description,Tags&subscription-key=<Your subscription key>
```

2. Passing the subscription key can also be specified in the HTTP request header:

```
ocp-apim-subscription-key: <Your subscription key>
```

3. When using the client library, the subscription key is passed in through the constructor of VisionServiceClient:

```
var visionClient = new VisionServiceClient("Your subscriptionKey");
```

The basic way to perform the Computer Vision API call is by uploading an image directly. This is done by sending a "POST" request with application/octet-stream content type together with the data read from the image. For "Tags" and "Description", this upload method will be the same for all the Computer Vision API calls. The only difference will be the query parameters the user specifies.

Here's how to get "Tags" and "Description" for a given image:

**Option One:** Get list of "Tags" and one "Description"

```
POST https://westus.api.cognitive.microsoft.com/vision/v1.0/analyze?visualFeatures=Description,Tags&subscription-key=<Your subscription key>
```

```
using Microsoft.ProjectOxford.Vision;
using Microsoft.ProjectOxford.Vision.Contract;
using System.IO;

AnalysisResult analysisResult;
var features = new VisualFeature[] { VisualFeature.Tags, VisualFeature.Description };

using (var fs = new FileStream(@"C:\Vision\Sample.jpg", FileMode.Open))
{
    analysisResult = await visionClient.AnalyzeImageAsync(fs, features);
}
```

### Option Two Get list of "Tags" only, or list of "Description" only:

Tags-only:

```
POST https://westus.api.cognitive.microsoft.com/vision/v1.0/tag&subscription-key=<Your subscription key>
var analysisResult = await visionClient.GetTagsAsync("http://contoso.com/example.jpg");
```

Description-only:

```
POST https://westus.api.cognitive.microsoft.com/vision/v1.0/describe&subscription-key=<Your subscription key>
using (var fs = new FileStream(@"C:\Vision\Sample.jpg", FileMode.Open))
{
    analysisResult = await visionClient.DescribeAsync(fs);
}
```

### Here is how to get domain-specific analysis (in our case, for celebrities).

#### Option One: Scoped Analysis - Analyze only a given model

```
POST https://westus.api.cognitive.microsoft.com/vision/v1.0/models/celebrities/analyze
var celebritiesResult = await visionClient.AnalyzeImageInDomainAsync(url, "celebrities");
```

For this option, all other query parameters {visualFeatures, details} are not valid. If you want to see all supported models, use:

```
GET https://westus.api.cognitive.microsoft.com/vision/v1.0/models
var models = await visionClient.ListModelsAsync();
```

#### Option Two: Enhanced Analysis - Analyze to provide additional details with [86-categories taxonomy](#)

For applications where you want to get generic image analysis in addition to details from one or more domain-specific models, we extend the v1 API with the models query parameter.

```
POST https://westus.api.cognitive.microsoft.com/vision/v1.0/analyze?details=celebrities
```

When this method is invoked, we will call the 86-category classifier first. If any of the categories match that of a known/matching model, a second pass of classifier invocations will occur. For example, if "details=all", or "details" include 'celebrities', we will call the celebrities model after the 86-category classifier is called and the result includes the category person. This will increase latency for users interested in celebrities, compared to Option One.

All v1 query parameters will behave the same in this case. If visualFeatures=categories is not specified, it will be

implicitly enabled.

Here's an example:

```
{
  "tags": [
    {
      "name": "outdoor",
      "score": 0.976
    },
    {
      "name": "bird",
      "score": 0.95
    }
  ],
  "description": {
    "tags": [
      "outdoor",
      "bird"
    ],
    "captions": [
      {
        "text": "partridge in a pear tree",
        "confidence": 0.96
      }
    ]
  }
}
```

FIELD	TYPE	CONTENT
Tags	object	Top-level object for array of tags
tags[].Name	string	Keyword from tags classifier
tags[].Score	number	Confidence score, between 0 and 1.
description	object	Top-level object for a description.
description.tags[]	string	List of tags. If there insufficient confidence in the ability to produce a caption, the tags maybe the only information available to the caller.
description.captions[].text	string	A phrase describing the image.
description.captions[].confidence	number	Confidence for the phrase.

**Option One:** Scoped Analysis - Analyze only a given model

The output will be an array of tags, an example will be like this example:

```
{
  "result": [
    {
      "name": "golden retriever",
      "score": 0.98
    },
    {
      "name": "Labrador retriever",
      "score": 0.78
    }
  ]
}
```

## Option Two: Enhanced Analysis - Analyze to provide additional details with 86-categories taxonomy

For domain-specific models using Option Two (Enhanced Analysis), the categories return type is extended. An example follows:

```
{
  "requestId": "87e44580-925a-49c8-b661-d1c54d1b83b5",
  "metadata": {
    "width": 640,
    "height": 430,
    "format": "Jpeg"
  },
  "result": {
    "celebrities": [
      {
        "name": "Richard Nixon",
        "faceRectangle": {
          "left": 107,
          "top": 98,
          "width": 165,
          "height": 165
        },
        "confidence": 0.9999827
      }
    ]
  }
}
```

The categories field is a list of one or more of the [86-categories](#) in the original taxonomy. Note also that categories ending in an underscore will match that category and its children (for example, people\_ as well as people\_group, for celebrities model).

FIELD	TYPE	CONTENT
categories	object	Top-level object
categories[].name	string	Name from 86-category taxonomy
categories[].score	number	Confidence score, between 0 and 1
categories[].detail	object?	Optional detail object

Note that if multiple categories match (for example, 86-category classifier returns a score for both people\_ and people\_young when model=celebrities), the details are attached to the most general level match (people\_ in that example.)

These are identical to vision.analyze, with the additional error of NotSupportedModel error (HTTP 400), which may

be returned in both Option One and Option Two scenarios. For Option Two (Enhanced Analysis), if any of the models specified in details are not recognized, the API will return a `NotSupportedModel`, even if one or more of them are valid. Users can call `listModels` to find out what models are supported.

These are the basic functionalities of the Computer Vision API: how you can upload images and retrieve valuable metadata in return.

To use the REST API, go to [Computer Vision API Reference](#).

# How to Analyze Videos in Real-time

12/7/2017 • 7 min to read • [Edit Online](#)

This guide will demonstrate how to perform near-real-time analysis on frames taken from a live video stream. The basic components in such a system are:

- Acquire frames from a video source
- Select which frames to analyze
- Submit these frames to the API
- Consume each analysis result that is returned from the API call

These samples are written in C# and the code can be found on GitHub here:

<https://github.com/Microsoft/Cognitive-Samples-VideoFrameAnalysis>.

## The Approach

There are multiple ways to solve the problem of running near-real-time analysis on video streams. We will start by outlining three approaches in increasing levels of sophistication.

### A Simple Approach

The simplest design for a near-real-time analysis system is an infinite loop, where in each iteration we grab a frame, analyze it, and then consume the result:

```
while (true)
{
    Frame f = GrabFrame();
    if (ShouldAnalyze(f))
    {
        AnalysisResult r = await Analyze(f);
        ConsumeResult(r);
    }
}
```

If our analysis consisted of a lightweight client-side algorithm, this approach would be suitable. However, when our analysis is happening in the cloud, the latency involved means that an API call might take several seconds, during which time we are not capturing images, and our thread is essentially doing nothing. Our maximum frame-rate is limited by the latency of the API calls.

### Parallelizing API Calls

While a simple single-threaded loop makes sense for a lightweight client-side algorithm, it doesn't fit well with the latency involved in cloud API calls. The solution to this problem is to allow the long-running API calls to execute in parallel with the frame-grabbing. In C#, we could achieve this using Task-based parallelism, for example:

```

while (true)
{
    Frame f = GrabFrame();
    if (ShouldAnalyze(f))
    {
        var t = Task.Run(async () =>
        {
            AnalysisResult r = await Analyze(f);
            ConsumeResult(r);
        })
    }
}

```

This launches each analysis in a separate Task, which can run in the background while we continue grabbing new frames. This avoids blocking the main thread while waiting for an API call to return, however we have lost some of the guarantees that the simple version provided -- multiple API calls might occur in parallel, and the results might get returned in the wrong order. This could also cause multiple threads to enter the ConsumeResult() function simultaneously, which could be dangerous, if the function is not thread-safe. Finally, this simple code does not keep track of the Tasks that get created, so exceptions will silently disappear. Thus, the final ingredient for us to add is a "consumer" thread that will track the analysis tasks, raise exceptions, kill long-running tasks, and ensure the results get consumed in the correct order, one at a time.

### A Producer-Consumer Design

In our final "producer-consumer" system, we have a producer thread that looks very similar to our previous infinite loop. However, instead of consuming analysis results as soon as they are available, the producer simply puts the tasks into a queue to keep track of them.

```

// Queue that will contain the API call tasks.
var taskQueue = new BlockingCollection<Task<ResultWrapper>>();

// Producer thread.
while (true)
{
    // Grab a frame.
    Frame f = GrabFrame();

    // Decide whether to analyze the frame.
    if (ShouldAnalyze(f))
    {
        // Start a task that will run in parallel with this thread.
        var analysisTask = Task.Run(async () =>
        {
            // Put the frame, and the result/exception into a wrapper object.
            var output = new ResultWrapper(f);
            try
            {
                output.Analysis = await Analyze(f);
            }
            catch (Exception e)
            {
                output.Exception = e;
            }
            return output;
        })

        // Push the task onto the queue.
        taskQueue.Add(analysisTask);
    }
}

```

We also have a consumer thread, that is taking tasks off the queue, waiting for them to finish, and either displaying

the result or raising the exception that was thrown. By using the queue, we can guarantee that results get consumed one at a time, in the correct order, without limiting the maximum frame-rate of the system.

```
// Consumer thread.
while (true)
{
    // Get the oldest task.
    Task<ResultWrapper> analysisTask = taskQueue.Take();

    // Await until the task is completed.
    var output = await analysisTask;

    // Consume the exception or result.
    if (output.Exception != null)
    {
        throw output.Exception;
    }
    else
    {
        ConsumeResult(output.Analysis);
    }
}
```

## Implementing the Solution

### Getting Started

To get your app up and running as quickly as possible, we have implemented the system described above, intending it to be flexible enough to implement many scenarios, while being easy to use. To access the code, go to <https://github.com/Microsoft/Cognitive-Samples-VideoFrameAnalysis>.

The library contains the class `FrameGrabber`, which implements the producer-consumer system discussed above to process video frames from a webcam. The user can specify the exact form of the API call, and the class uses events to let the calling code know when a new frame is acquired, or a new analysis result is available.

To illustrate some of the possibilities, there are two sample apps that use the library. The first is a simple console app, and a simplified version of this is reproduced below. It grabs frames from the default webcam, and submits them to the Face API for face detection.



```

using System;
using VideoFrameAnalyzer;
using Microsoft.ProjectOxford.Face;
using Microsoft.ProjectOxford.Face.Contract;

namespace VideoFrameConsoleApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            // Create grabber, with analysis type Face[].
            FrameGrabber<Face[]> grabber = new FrameGrabber<Face[]>();

            // Create Face API Client. Insert your Face API key here.
            FaceServiceClient faceClient = new FaceServiceClient("<subscription key>");

            // Set up our Face API call.
            grabber.AnalysisFunction = async frame => return await faceClient.DetectAsync(frame.Image.ToMemoryStream(".jpg"));

            // Set up a listener for when we receive a new result from an API call.
            grabber.NewResultAvailable += (s, e) =>
            {
                if (e.Analysis != null)
                    Console.WriteLine("New result received for frame acquired at {0}. {1} faces detected", e.Frame.Metadata.Timestamp,
e.Analysis.Length);
            };

            // Tell grabber to call the Face API every 3 seconds.
            grabber.TriggerAnalysisOnInterval(TimeSpan.FromMilliseconds(3000));

            // Start running.
            grabber.StartProcessingCameraAsync().Wait();

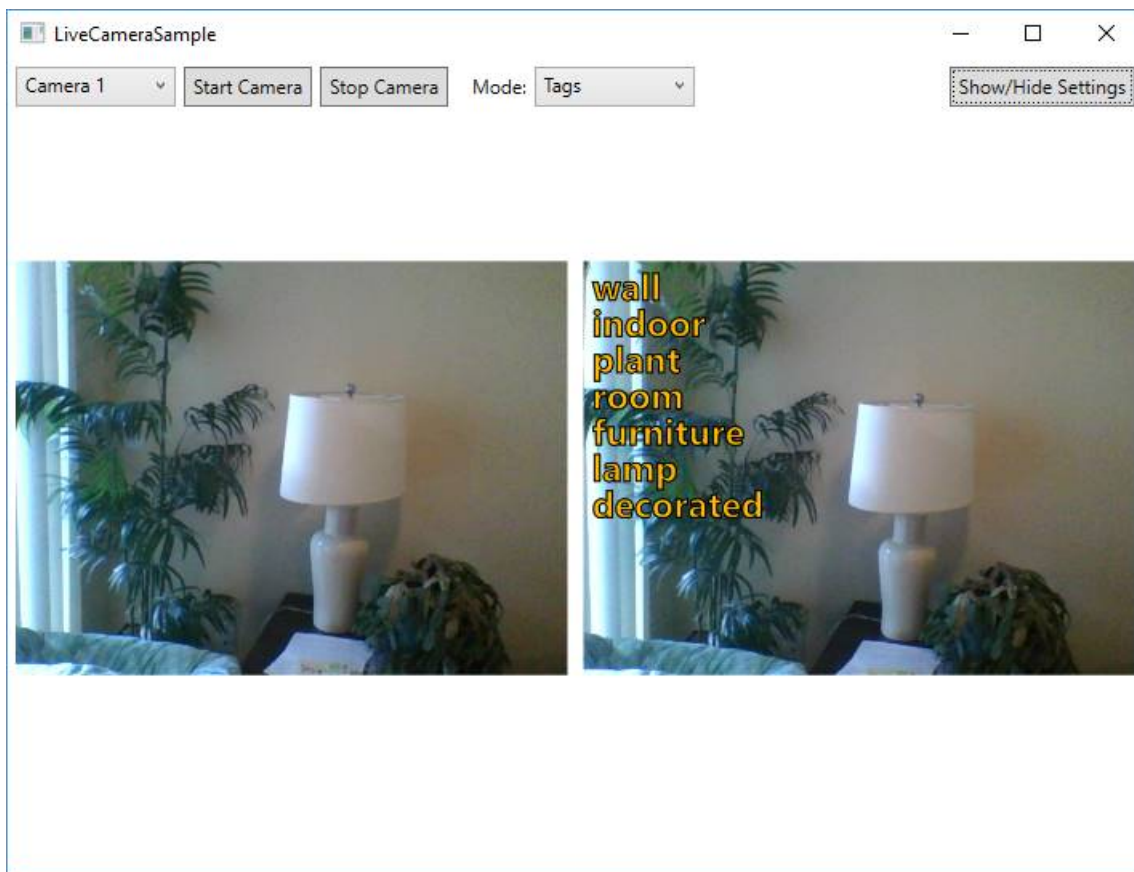
            // Wait for keypress to stop
            Console.WriteLine("Press any key to stop...");
            Console.ReadKey();

            // Stop, blocking until done.
            grabber.StopProcessingAsync().Wait();
        }
    }
}

```

The second sample app is a bit more interesting, and allows you to choose which API to call on the video frames. On the left hand side, the app shows a preview of the live video, on the right hand side it shows the most recent API result overlaid on the corresponding frame.

In most modes, there will be a visible delay between the live video on the left, and the visualized analysis on the right. This delay is the time taken to make the API call. The exception to this is in the "EmotionsWithClientFaceDetect" mode, which performs face detection locally on the client computer using OpenCV, before submitting any images to Cognitive Services. By doing this, we can visualize the detected face immediately, and then update the emotions later once the API call returns. This demonstrates the possibility of a "hybrid" approach, where some simple processing can be performed on the client, and then Cognitive Services APIs can be used to augment this with more advanced analysis when necessary.



## Integrating into your codebase

To get started with this sample, follow these steps:

1. Get API keys for the Vision APIs from [Subscriptions](#). For video frame analysis, the applicable APIs are:
  - [Computer Vision API](#)
  - [Emotion API](#)
  - [Face API](#)
2. Clone the [Cognitive-Samples-VideoFrameAnalysis](#) GitHub repo
3. Open the sample in Visual Studio 2015, build and run the sample applications:
  - For BasicConsoleSample, the Face API key is hard-coded directly in [BasicConsoleSample/Program.cs](#).
  - For LiveCameraSample, the keys should be entered into the Settings pane of the app. They will be persisted across sessions as user data.

When you're ready to integrate, **simply reference the VideoFrameAnalyzer library from your own projects.**

## Developer Code of Conduct

As with all the Cognitive Services, Developers developing with our APIs and samples are required to follow the "[Developer Code of Conduct for Microsoft Cognitive Services](#)."

The image, voice, video or text understanding capabilities of VideoFrameAnalyzer uses Microsoft Cognitive Services. Microsoft will receive the images, audio, video, and other data that you upload (via this app) and may use them for service improvement purposes. We ask for your help in protecting the people whose data your app sends to Microsoft Cognitive Services.

## Summary

In this guide, you learned how to run near-real-time analysis on live video streams using the Face, Computer Vision, and Emotion APIs, and how you can use our sample code to get started. You can get started building your app with

free API keys at the [Microsoft Cognitive Services sign-up page](#).

Please feel free to provide feedback and suggestions in the [GitHub repository](#), or for more broad API feedback, on our [UserVoice site](#).

# Computer Vision cURL Quick Starts

11/30/2017 • 4 min to read • [Edit Online](#)

This article provides information and code samples to help you quickly get started using the Computer Vision API with cURL to accomplish the following tasks:

- [Analyze an image](#)
- [Intelligently generate a thumbnail](#)
- [Detect and extract text from an Image](#)

Learn more about obtaining free Subscription Keys [here](#)

## Analyze an Image With Computer Vision API Using cURL

With the [Analyze Image method](#), you can extract visual features based on image content. You can upload an image or specify an image URL and choose which features to return, including:

- The category defined in this [taxonomy](#).
- A detailed list of tags related to the image content.
- A description of image content in a complete sentence.
- The coordinates, gender, and age of any faces contained in the image.
- The ImageType (clipart or a line drawing)
- The dominant color, the accent color, or whether an image is black & white.
- Whether the image contains pornographic or sexually suggestive content.

### NOTE

In the following samples, the parameter `--data-ascii "{body}"` has a placeholder `{body}` which should be replaced with a JSON formatted string that defines a URL for a web-accessible JPG file. For example:

```
--data-ascii '{"url": "https://domain/path/to/my/photo.jpg"}'
```

### Analyze an Image curl Example Request

Change the URL to use the location where you obtained your subscription keys. Replace the "Ocp-Apim-Subscription-Key" value with your valid subscription key, and replace `{string}` with "Celebrities" or "Landmark," or alternatively, remove the entire parameter `&details={string}`.

### NOTE

You must use the same location in your REST call as you used to obtain your subscription keys. For example, if you obtained your subscription keys from westus, replace "westcentralus" in the URL below with "westus".

@ECHO OFF

```
curl -v -X POST "https://westcentralus.api.cognitive.microsoft.com/vision/v1.0/analyze?visualFeatures=Categories&details={string}&language=en"  
-H "Content-Type: application/json"  
-H "Ocp-Apim-Subscription-Key: {subscription key}"  
  
--data-ascii "{body}"
```

## Analyze an Image Response

A successful response is returned in JSON. Following is an example of a successful response:

```
{
  "categories": [
    {
      "name": "abstract_",
      "score": 0.00390625
    },
    {
      "name": "people_",
      "score": 0.83984375,
      "detail": {
        "celebrities": [
          {
            "name": "Satya Nadella",
            "faceRectangle": {
              "left": 597,
              "top": 162,
              "width": 248,
              "height": 248
            },
            "confidence": 0.999028444
          }
        ]
      }
    }
  ],
  "adult": {
    "isAdultContent": false,
    "isRacyContent": false,
    "adultScore": 0.0934349000453949,
    "racyScore": 0.068613491952419281
  },
  "tags": [
    {
      "name": "person",
      "confidence": 0.98979085683822632
    },
    {
      "name": "man",
      "confidence": 0.94493889808654785
    },
    {
      "name": "outdoor",
      "confidence": 0.938492476940155
    },
    {
      "name": "window",
      "confidence": 0.89513939619064331
    }
  ],
  "description": {
    "tags": [
      "person",
      "man",
      "outdoor",
      "window",
      "glasses"
    ],
    "captions": [
      {
        "text": "Satya Nadella sitting on a bench",
        "confidence": 0.48293603002174407
      }
    ]
  },
  "imageMetadata": {
    "format": "JPEG",
    "width": 1024,
    "height": 1024,
    "size": 1048576
  }
}
```

```

"requestId": "0dbec5ad-a3d3-4f7e-96b4-dfd57efe967d",
"metadata": {
  "width": 1500,
  "height": 1000,
  "format": "Jpeg"
},
"faces": [
  {
    "age": 44,
    "gender": "Male",
    "faceRectangle": {
      "left": 593,
      "top": 160,
      "width": 250,
      "height": 250
    }
  }
],
"color": {
  "dominantColorForeground": "Brown",
  "dominantColorBackground": "Brown",
  "dominantColors": [
    "Brown",
    "Black"
  ],
  "accentColor": "873B59",
  "isBwImg": false
},
"imageType": {
  "clipArtType": 0,
  "lineDrawingType": 0
}
}

```

## Get a Thumbnail with Computer Vision API Using curl

Use the [Get Thumbnail method](#) to crop an image based on its region of interest (ROI) to the height and width you desire, even if the aspect ratio differs from the input image.

### Get a Thumbnail curl Example Request

Change the URL to use the location where you obtained your subscription keys, and replace the "Ocp-Apim-Subscription-Key" value with your valid subscription key.

#### NOTE

You must use the same location in your REST call as you used to obtain your subscription keys. For example, if you obtained your subscription keys from westus, replace "westcentralus" in the URL below with "westus".

@ECHO OFF

```

curl -v -X POST "https://westcentralus.api.cognitive.microsoft.com/vision/v1.0/generateThumbnail?width={number}&height={number}&smartCropping=true"
-H "Content-Type: application/json"
-H "Ocp-Apim-Subscription-Key: {subscription key}"

--data-ascii "{body}"

```

### Get a Thumbnail Response

A successful response contains the thumbnail image binary. If the request failed, the response contains an error code and a message to help determine what went wrong.

# Optical Character Recognition (OCR) with Computer Vision API Using curl

Use the [Optical Character Recognition \(OCR\) method](#) to detect text in an image and extract recognized characters into a machine-usable character stream.

## OCR curl Example Request

Change the URL to use the location where you obtained your subscription keys, and replace the "Ocp-Apim-Subscription-Key" value with your valid subscription key.

### NOTE

You must use the same location in your REST call as you used to obtain your subscription keys. For example, if you obtained your subscription keys from westus, replace "westcentralus" in the URL below with "westus".

@ECHO OFF

```
curl -v -X POST "https://westcentralus.api.cognitive.microsoft.com/vision/v1.0/ocr?language=unk&detectOrientation=true"
-H "Content-Type: application/json"
-H "Ocp-Apim-Subscription-Key: {subscription key}"

--data-ascii "{body}"
```

## OCR Example Response

Upon success, the OCR results returned include text, bounding box for regions, lines, and words.

```
{
  "language": "en",
  "textAngle": -2.0000000000000338,
  "orientation": "Up",
  "regions": [
    {
      "boundingBox": "462,379,497,258",
      "lines": [
        {
          "boundingBox": "462,379,497,74",
          "words": [
            {
              "boundingBox": "462,379,41,73",
              "text": "A"
            },
            {
              "boundingBox": "523,379,153,73",
              "text": "GOAL"
            },
            {
              "boundingBox": "694,379,265,74",
              "text": "WITHOUT"
            }
          ]
        },
        {
          "boundingBox": "565,471,289,74",
          "words": [
            {
              "boundingBox": "565,471,41,73",
              "text": "A"
            },
            {
              "boundingBox": "626,471,150,73",
              "text": "PLAN"
            },
            {
              "boundingBox": "801,472,53,73",
              "text": "IS"
            }
          ]
        },
        {
          "boundingBox": "519,563,375,74",
          "words": [
            {
              "boundingBox": "519,563,149,74",
              "text": "JUST"
            },
            {
              "boundingBox": "683,564,41,72",
              "text": "A"
            },
            {
              "boundingBox": "741,564,153,73",
              "text": "WISH"
            }
          ]
        }
      ]
    }
  ]
}
```



# Computer Vision C# Quick Starts

2/15/2018 • 21 min to read • [Edit Online](#)

This article provides information and code samples to help you quickly get started using the Computer Vision API with C# to accomplish the following tasks:

- [Analyze an image](#)
- [Use a Domain-Specific Model](#)
- [Intelligently generate a thumbnail](#)
- [Detect and extract printed text from an image](#)
- [Detect and extract handwritten text from an image](#)

## Prerequisites

- To use the Computer Vision API, you need a subscription key. You can get free subscription keys [here](#).

## Analyze an Image With Computer Vision API using C#

With the [Analyze Image method](#), you can extract visual features based on image content. You can upload an image or specify an image URL and choose which features to return, including:

- A detailed list of tags related to the image content.
- A description of image content in a complete sentence.
- The coordinates, gender, and age of any faces contained in the image.
- The ImageType (clip art or a line drawing).
- The dominant color, the accent color, or whether an image is black & white.
- The category defined in this [taxonomy](#).
- Does the image contain adult or sexually suggestive content?

### Analyze an image C# example request

Create a new Console solution in Visual Studio, then replace Program.cs with the following code. Change the `uriBase` to use the location where you obtained your subscription keys, and replace the `subscriptionKey` value with your valid subscription key.

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Text;

namespace CSHttpClientSample
{
    static class Program
    {
        // *****
        // *** Update or verify the following values. ***
        // *****

        // Replace the subscriptionKey string value with your valid subscription key.
        const string subscriptionKey = "13hc77781f7e4b19b5fcdd72a8df7156";
```

```

// Replace or verify the region.
//
// You must use the same region in your REST API call as you used to obtain your subscription keys.
// For example, if you obtained your subscription keys from the westus region, replace
// "westcentralus" in the URI below with "westus".
//
// NOTE: Free trial subscription keys are generated in the westcentralus region, so if you are using
// a free trial subscription key, you should not need to change this region.
const string uriBase = "https://westcentralus.api.cognitive.microsoft.com/vision/v1.0/analyze";

static void Main()
{
    // Get the path and filename to process from the user.
    Console.WriteLine("Analyze an image:");
    Console.Write("Enter the path to an image you wish to analyze: ");
    string imageFilePath = Console.ReadLine();

    // Execute the REST API call.
    MakeAnalysisRequest(imageFilePath);

    Console.WriteLine("\nPlease wait a moment for the results to appear. Then, press Enter to exit...\n");
    Console.ReadLine();
}

/// <summary>
/// Gets the analysis of the specified image file by using the Computer Vision REST API.
/// </summary>
/// <param name="imageFilePath">The image file.</param>
static async void MakeAnalysisRequest(string imageFilePath)
{
    HttpClient client = new HttpClient();

    // Request headers.
    client.DefaultRequestHeaders.Add("Ocp-Apim-Subscription-Key", subscriptionKey);

    // Request parameters. A third optional parameter is "details".
    string requestParameters = "visualFeatures=Categories,Description,Color&language=en";

    // Assemble the URI for the REST API Call.
    string uri = uriBase + "?" + requestParameters;

    HttpResponseMessage response;

    // Request body. Posts a locally stored JPEG image.
    byte[] byteData = GetImageAsByteArray(imageFilePath);

    using (ByteArrayContent content = new ByteArrayContent(byteData))
    {
        // This example uses content type "application/octet-stream".
        // The other content types you can use are "application/json" and "multipart/form-data".
        content.Headers.ContentType = new MediaTypeHeaderValue("application/octet-stream");

        // Execute the REST API call.
        response = await client.PostAsync(uri, content);

        // Get the JSON response.
        string contentString = await response.Content.ReadAsStringAsync();

        // Display the JSON response.
        Console.WriteLine("\nResponse:\n");
        Console.WriteLine(JsonPrettyPrint(contentString));
    }
}

/// <summary>
/// Returns the contents of the specified file as a byte array.

```

```

/// </summary>
/// <param name="imageFilePath">The image file to read.</param>
/// <returns>The byte array of the image data.</returns>
static byte[] GetImageAsByteArray(string imageFilePath)
{
    FileStream fileStream = new FileStream(imageFilePath, FileMode.Open, FileAccess.Read);
    BinaryReader binaryReader = new BinaryReader(fileStream);
    return binaryReader.ReadBytes((int)fileStream.Length);
}

/// </summary>
/// Formats the given JSON string by adding line breaks and indents.
/// </summary>
/// <param name="json">The raw JSON string to format.</param>
/// <returns>The formatted JSON string.</returns>
static string JsonPrettyPrint(string json)
{
    if (string.IsNullOrEmpty(json))
        return string.Empty;

    json = json.Replace(Environment.NewLine, "").Replace("\t", "");

    string INDENT_STRING = "  ";
    var indent = 0;
    var quoted = false;
    var sb = new StringBuilder();
    for (var i = 0; i < json.Length; i++)
    {
        var ch = json[i];
        switch (ch)
        {
            case '{':
            case '[':
                sb.Append(ch);
                if (!quoted)
                {
                    sb.AppendLine();
                    Enumerable.Range(0, ++indent).ForEach(item => sb.Append(INDENT_STRING));
                }
                break;
            case '}':
            case ']':
                if (!quoted)
                {
                    sb.AppendLine();
                    Enumerable.Range(0, --indent).ForEach(item => sb.Append(INDENT_STRING));
                }
                sb.Append(ch);
                break;
            case '"':
                sb.Append(ch);
                bool escaped = false;
                var index = i;
                while (index > 0 && json[--index] == '\\')
                    escaped = !escaped;
                if (!escaped)
                    quoted = !quoted;
                break;
            case ',':
                sb.Append(ch);
                if (!quoted)
                {
                    sb.AppendLine();
                    Enumerable.Range(0, indent).ForEach(item => sb.Append(INDENT_STRING));
                }
                break;
            case ':':
                sb.Append(ch);

```

```
        if (!quoted)
            sb.Append(" ");
        break;
    default:
        sb.Append(ch);
        break;
    }
}
return sb.ToString();
}
}
static class Extensions
{
    public static void ForEach<T>(this IEnumerable<T> ie, Action<T> action)
    {
        foreach (var i in ie)
        {
            action(i);
        }
    }
}
```

### Analyze an Image response

A successful response is returned in JSON. Following is an example of a successful response:

```
{
  "categories": [
    {
      "name": "abstract _",
      "score": 0.00390625
    },
    {
      "name": "others _",
      "score": 0.0234375
    },
    {
      "name": "outdoor _",
      "score": 0.00390625
    }
  ],
  "description": {
    "tags": [
      "road",
      "building",
      "outdoor",
      "street",
      "night",
      "black",
      "city",
      "white",
      "light",
      "sitting",
      "riding",
      "man",
      "side",
      "empty",
      "rain",
      "corner",
      "traffic",
      "lit",
      "hydrant",
      "stop",
      "board",
      "parked",
      "bus",
      "tall"
    ],
    "captions": [
      {
        "text": "a close up of an empty city street at night",
        "confidence": 0.7965622853462756
      }
    ]
  },
  "requestId": "dddflac9-7e66-4c47-bdef-222f3fe5aa23",
  "metadata": {
    "width": 3733,
    "height": 1986,
    "format": "Jpeg"
  },
  "color": {
    "dominantColorForeground": "Black",
    "dominantColorBackground": "Black",
    "dominantColors": [
      "Black",
      "Grey"
    ],
    "accentColor": "666666",
    "isBWImg": true
  }
}
```

# Use a Domain-Specific Model

The Domain-Specific Model is a model trained to identify a specific set of objects in an image. The two domain-specific models that are currently available are celebrities and landmarks. The following example identifies a landmark in an image.

## Landmark C# example request

Create a new Console solution in Visual Studio, then replace Program.cs with the following code. Change the `uriBase` to use the location where you obtained your subscription keys, and replace the `subscriptionKey` value with your valid subscription key.

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Text;

namespace CSHttpClientSample
{
    static class Program
    {
        // *****
        // *** Update or verify the following values. ***
        // *****

        // Replace the subscriptionKey string value with your valid subscription key.
        const string subscriptionKey = "13hc77781f7e4b19b5fcdd72a8df7156";

        // Replace or verify the region.
        //
        // You must use the same region in your REST API call as you used to obtain your subscription keys.
        // For example, if you obtained your subscription keys from the westus region, replace
        // "westcentralus" in the URI below with "westus".
        //
        // NOTE: Free trial subscription keys are generated in the westcentralus region, so if you are using
        // a free trial subscription key, you should not need to change this region.
        //
        // Also, if you want to use the celebrities model, change "landmarks" to "celebrities" here and in
        // requestParameters to use the Celebrities model.
        const string uriBase = "https://westcentralus.api.cognitive.microsoft.com/vision/v1.0/models/landmarks/analyze";

        static void Main()
        {
            // Get the path and filename to process from the user.
            Console.WriteLine("Domain-Specific Model:");
            Console.WriteLine("Enter the path to an image you wish to analyze for landmarks: ");
            string imageFilePath = Console.ReadLine();

            // Execute the REST API call.
            MakeAnalysisRequest(imageFilePath);

            Console.WriteLine("\nPlease wait a moment for the results to appear. Then, press Enter to exit ...\n");
            Console.ReadLine();
        }

        /// <summary>
        /// Gets a thumbnail image from the specified image file by using the Computer Vision REST API.
        /// </summary>
        /// <param name="imageFilePath">The image file to use to create the thumbnail image.</param>
        static async void MakeAnalysisRequest(string imageFilePath)
        {
            HttpClient client = new HttpClient();
```

```

// Request headers.
client.DefaultRequestHeaders.Add("Ocp-Apim-Subscription-Key", subscriptionKey);

// Request parameters. Change "landmarks" to "celebrities" here and in uriBase to use the Celebrities model.
string requestParameters = "model=landmarks";

// Assemble the URI for the REST API Call.
string uri = uriBase + "?" + requestParameters;

HttpResponseMessage response;

// Request body. Posts a locally stored JPEG image.
byte[] byteData = GetImageAsByteArray(imageFilePath);

using (ByteArrayContent content = new ByteArrayContent(byteData))
{
    // This example uses content type "application/octet-stream".
    // The other content types you can use are "application/json" and "multipart/form-data".
    content.Headers.ContentType = new MediaTypeHeaderValue("application/octet-stream");

    // Execute the REST API call.
    response = await client.PostAsync(uri, content);

    // Get the JSON response.
    string contentString = await response.Content.ReadAsStringAsync();

    // Display the JSON response.
    Console.WriteLine("\nResponse:\n");
    Console.WriteLine(JsonPrettyPrint(contentString));
}
}

/// <summary>
/// Returns the contents of the specified file as a byte array.
/// </summary>
/// <param name="imageFilePath">The image file to read.</param>
/// <returns>The byte array of the image data.</returns>
static byte[] GetImageAsByteArray(string imageFilePath)
{
    FileStream fileStream = new FileStream(imageFilePath, FileMode.Open, FileAccess.Read);
    BinaryReader binaryReader = new BinaryReader(fileStream);
    return binaryReader.ReadBytes((int)fileStream.Length);
}

/// <summary>
/// Formats the given JSON string by adding line breaks and indents.
/// </summary>
/// <param name="json">The raw JSON string to format.</param>
/// <returns>The formatted JSON string.</returns>
static string JsonPrettyPrint(string json)
{
    if (string.IsNullOrEmpty(json))
        return string.Empty;

    json = json.Replace(Environment.NewLine, "").Replace("\t", "");

    string INDENT_STRING = "  ";
    var indent = 0;
    var quoted = false;
    var sb = new StringBuilder();
    for (var i = 0; i < json.Length; i++)
    {
        var ch = json[i];
        switch (ch)
        {
            case '{':

```

```

        case '[':
            sb.Append(ch);
            if (!quoted)
            {
                sb.AppendLine();
                Enumerable.Range(0, ++indent).ForEach(item => sb.Append(INDENT_STRING));
            }
            break;
        case '}':
        case ']':
            if (!quoted)
            {
                sb.AppendLine();
                Enumerable.Range(0, --indent).ForEach(item => sb.Append(INDENT_STRING));
            }
            sb.Append(ch);
            break;
        case '"':
            sb.Append(ch);
            bool escaped = false;
            var index = i;
            while (index > 0 && json[--index] == '\\')
                escaped = !escaped;
            if (!escaped)
                quoted = !quoted;
            break;
        case ',':
            sb.Append(ch);
            if (!quoted)
            {
                sb.AppendLine();
                Enumerable.Range(0, indent).ForEach(item => sb.Append(INDENT_STRING));
            }
            break;
        case ':':
            sb.Append(ch);
            if (!quoted)
                sb.Append(" ");
            break;
        default:
            sb.Append(ch);
            break;
    }
}
return sb.ToString();
}
}

static class Extensions
{
    public static void ForEach<T>(this IEnumerable<T> ie, Action<T> action)
    {
        foreach (var i in ie)
        {
            action(i);
        }
    }
}
}

```

### Landmark example response

A successful response is returned in JSON. Following is an example of a successful response:



```
{
  "requestId": "cfe3d4eb-4d9c-4dda-ae63-7d3a27ce6d27",
  "metadata": {
    "width": 1024,
    "height": 680,
    "format": "Jpeg"
  },
  "result": {
    "landmarks": [
      {
        "name": "Space Needle",
        "confidence": 0.9448209
      }
    ]
  }
}
```

## Get a thumbnail with Computer Vision API using C#

Use the [Get Thumbnail method](#) to crop an image based on its region of interest (ROI) to the height and width you desire. You can even pick an aspect ratio that differs from the aspect ratio of the input image.

### Get a thumbnail C# example request

Create a new Console solution in Visual Studio, then replace Program.cs with the following code. Change the `uriBase` to use the location where you obtained your subscription keys, and replace the `subscriptionKey` value with your valid subscription key.

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Text;

namespace CSHttpClientSample
{
    static class Program
    {
        // *****
        // *** Update or verify the following values. ***
        // *****

        // Replace the subscriptionKey string value with your valid subscription key.
        const string subscriptionKey = "13hc77781f7e4b19b5fcdd72a8df7156";

        // Replace or verify the region.
        //
        // You must use the same region in your REST API call as you used to obtain your subscription keys.
        // For example, if you obtained your subscription keys from the westus region, replace
        // "westcentralus" in the URI below with "westus".
        //
        // NOTE: Free trial subscription keys are generated in the westcentralus region, so if you are using
        // a free trial subscription key, you should not need to change this region.
        const string uriBase = "https://westcentralus.api.cognitive.microsoft.com/vision/v1.0/generateThumbnail";

        static void Main()
        {
            // Get the path and filename to process from the user.
            Console.WriteLine("Thumbnail:");
            Console.Write("Enter the path to an image you wish to use to create a thumbnail image: ");
            string imageFilePath = Console.ReadLine();
        }
    }
}
```

```
// Execute the REST API call.
MakeThumbNailRequest(imageFilePath);

Console.WriteLine("\nPlease wait a moment for the results to appear. Then, press Enter to exit ... \n");
Console.ReadLine();
}
```

```
/// <summary>
/// Gets a thumbnail image from the specified image file by using the Computer Vision REST API.
/// </summary>
/// <param name="imageFilePath">The image file to use to create the thumbnail image.</param>
static async void MakeThumbNailRequest(string imageFilePath)
{
    HttpClient client = new HttpClient();

    // Request headers.
    client.DefaultRequestHeaders.Add("Ocp-Apim-Subscription-Key", subscriptionKey);

    // Request parameters.
    string requestParameters = "width=200&height=150&smartCropping=true";

    // Assemble the URI for the REST API Call.
    string uri = uriBase + "?" + requestParameters;

    HttpResponseMessage response;

    // Request body. Posts a locally stored JPEG image.
    byte[] byteData = GetImageAsByteArray(imageFilePath);

    using (ByteArrayContent content = new ByteArrayContent(byteData))
    {
        // This example uses content type "application/octet-stream".
        // The other content types you can use are "application/json" and "multipart/form-data".
        content.Headers.ContentType = new MediaTypeHeaderValue("application/octet-stream");

        // Execute the REST API call.
        response = await client.PostAsync(uri, content);

        if (response.IsSuccessStatusCode)
        {
            // Display the response data.
            Console.WriteLine("\nResponse:\n");
            Console.WriteLine(response);

            // Get the image data.
            byte[] thumbnailImageData = await response.Content.ReadAsByteArrayAsync();
        }
        else
        {
            // Display the JSON error data.
            Console.WriteLine("\nError:\n");
            Console.WriteLine(JsonPrettyPrint(await response.Content.ReadAsStringAsync()));
        }
    }
}
```

```
/// <summary>
/// Returns the contents of the specified file as a byte array.
/// </summary>
/// <param name="imageFilePath">The image file to read.</param>
/// <returns>The byte array of the image data.</returns>
static byte[] GetImageAsByteArray(string imageFilePath)
{
    FileStream fileStream = new FileStream(imageFilePath, FileMode.Open, FileAccess.Read);
    BinaryReader binaryReader = new BinaryReader(fileStream);
    return binaryReader.ReadBytes((int)fileStream.Length);
}
```

```

/// <summary>
/// Formats the given JSON string by adding line breaks and indents.
/// </summary>
/// <param name="json">The raw JSON string to format.</param>
/// <returns>The formatted JSON string.</returns>
static string JsonPrettyPrint(string json)
{
    if (string.IsNullOrEmpty(json))
        return string.Empty;

    json = json.Replace(Environment.NewLine, "").Replace("\t", "");

    string INDENT_STRING = "  ";
    var indent = 0;
    var quoted = false;
    var sb = new StringBuilder();
    for (var i = 0; i < json.Length; i++)
    {
        var ch = json[i];
        switch (ch)
        {
            case '{':
            case '[':
                sb.Append(ch);
                if (!quoted)
                {
                    sb.AppendLine();
                    Enumerable.Range(0, ++indent).ForEach(item => sb.Append(INDENT_STRING));
                }
                break;
            case '}':
            case ']':
                if (!quoted)
                {
                    sb.AppendLine();
                    Enumerable.Range(0, --indent).ForEach(item => sb.Append(INDENT_STRING));
                }
                sb.Append(ch);
                break;
            case '"':
                sb.Append(ch);
                bool escaped = false;
                var index = i;
                while (index > 0 && json[--index] == '\\')
                    escaped = !escaped;
                if (!escaped)
                    quoted = !quoted;
                break;
            case ',':
                sb.Append(ch);
                if (!quoted)
                {
                    sb.AppendLine();
                    Enumerable.Range(0, indent).ForEach(item => sb.Append(INDENT_STRING));
                }
                break;
            case ':':
                sb.Append(ch);
                if (!quoted)
                {
                    sb.AppendLine();
                    sb.Append(" ");
                }
                break;
            default:
                sb.Append(ch);
                break;
        }
    }
    return sb.ToString();
}

```

```

    }
}
static class Extensions
{
    public static void ForEach<T>(this IEnumerable<T> ie, Action<T> action)
    {
        foreach (var i in ie)
        {
            action(i);
        }
    }
}
}

```

## Get a Thumbnail response

A successful response contains the thumbnail image binary. If the request fails, the response contains an error code and a message to help determine what went wrong.

Response:

```

StatusCode: 200, ReasonPhrase: 'OK', Version: 1.1, Content: System.Net.Http.StreamContent, Headers:
{
    Pragma: no-cache
    apim-request-id: 131eb5b4-5807-466d-9656-4c1ef0a64c9b
    Strict-Transport-Security: max-age=31536000; includeSubDomains; preload
    x-content-type-options: nosniff
    Cache-Control: no-cache
    Date: Tue, 06 Jun 2017 20:54:07 GMT
    X-AspNet-Version: 4.0.30319
    X-Powered-By: ASP.NET
    Content-Length: 5800
    Content-Type: image/jpeg
    Expires: -1
}

```

# Optical Character Recognition (OCR) with Computer Vision API using C#

Use the [Optical Character Recognition \(OCR\) method](#) to detect printed text in an image and extract recognized characters into a machine-usable character stream.

## OCR C# example request

Create a new Console solution in Visual Studio, then replace Program.cs with the following code. Change the `uriBase` to use the location where you obtained your subscription keys, and replace the `subscriptionKey` value with your valid subscription key.

```

using System;
using System.Collections.Generic;
using System.IO;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Text;

namespace CSHttpClientSample
{
    static class Program
    {
        // *****
        // *** Update or verify the following values. ***
        // *****

```

```

// Replace the subscriptionKey string value with your valid subscription key.
const string subscriptionKey = "13hc77781f7e4b19b5fcdd72a8df7156";

// Replace or verify the region.
//
// You must use the same region in your REST API call as you used to obtain your subscription keys.
// For example, if you obtained your subscription keys from the westus region, replace
// "westcentralus" in the URI below with "westus".
//
// NOTE: Free trial subscription keys are generated in the westcentralus region, so if you are using
// a free trial subscription key, you should not need to change this region.
const string uriBase = "https://westcentralus.api.cognitive.microsoft.com/vision/v1.0/ocr";

static void Main()
{
    // Get the path and filename to process from the user.
    Console.WriteLine("Optical Character Recognition:");
    Console.Write("Enter the path to an image with text you wish to read: ");
    string imageFilePath = Console.ReadLine();

    // Execute the REST API call.
    MakeOCRRequest(imageFilePath);

    Console.WriteLine("\nPlease wait a moment for the results to appear. Then, press Enter to exit...\n");
    Console.ReadLine();
}

/// <summary>
/// Gets the text visible in the specified image file by using the Computer Vision REST API.
/// </summary>
/// <param name="imageFilePath">The image file.</param>
static async void MakeOCRRequest(string imageFilePath)
{
    HttpClient client = new HttpClient();

    // Request headers.
    client.DefaultRequestHeaders.Add("Ocp-Apim-Subscription-Key", subscriptionKey);

    // Request parameters.
    string requestParameters = "language=unk&detectOrientation=true";

    // Assemble the URI for the REST API Call.
    string uri = uriBase + "?" + requestParameters;

    HttpResponseMessage response;

    // Request body. Posts a locally stored JPEG image.
    byte[] byteData = GetImageAsByteArray(imageFilePath);

    using (ByteArrayContent content = new ByteArrayContent(byteData))
    {
        // This example uses content type "application/octet-stream".
        // The other content types you can use are "application/json" and "multipart/form-data".
        content.Headers.ContentType = new MediaTypeHeaderValue("application/octet-stream");

        // Execute the REST API call.
        response = await client.PostAsync(uri, content);

        // Get the JSON response.
        string contentString = await response.Content.ReadAsStringAsync();

        // Display the JSON response.
        Console.WriteLine("\nResponse:\n");
        Console.WriteLine(JsonPrettyPrint(contentString));
    }
}

```

```

/// <summary>
/// Returns the contents of the specified file as a byte array.
/// </summary>
/// <param name="imageFilePath">The image file to read.</param>
/// <returns>The byte array of the image data.</returns>
static byte[] GetImageAsByteArray(string imageFilePath)
{
    FileStream fileStream = new FileStream(imageFilePath, FileMode.Open, FileAccess.Read);
    BinaryReader binaryReader = new BinaryReader(fileStream);
    return binaryReader.ReadBytes((int)fileStream.Length);
}

/// <summary>
/// Formats the given JSON string by adding line breaks and indents.
/// </summary>
/// <param name="json">The raw JSON string to format.</param>
/// <returns>The formatted JSON string.</returns>
static string JsonPrettyPrint(string json)
{
    if (string.IsNullOrEmpty(json))
        return string.Empty;

    json = json.Replace(Environment.NewLine, "").Replace("\t", "");

    string INDENT_STRING = "  ";
    var indent = 0;
    var quoted = false;
    var sb = new StringBuilder();
    for (var i = 0; i < json.Length; i++)
    {
        var ch = json[i];
        switch (ch)
        {
            case '{':
            case '[':
                sb.Append(ch);
                if (!quoted)
                {
                    sb.AppendLine();
                    Enumerable.Range(0, ++indent).ForEach(item => sb.Append(INDENT_STRING));
                }
                break;
            case '}':
            case ']':
                if (!quoted)
                {
                    sb.AppendLine();
                    Enumerable.Range(0, --indent).ForEach(item => sb.Append(INDENT_STRING));
                }
                sb.Append(ch);
                break;
            case '"':
                sb.Append(ch);
                bool escaped = false;
                var index = i;
                while (index > 0 && json[--index] == '\\')
                    escaped = !escaped;
                if (!escaped)
                    quoted = !quoted;
                break;
            case ';':
                sb.Append(ch);
                if (!quoted)
                {
                    sb.AppendLine();
                    Enumerable.Range(0, indent).ForEach(item => sb.Append(INDENT_STRING));
                }
        }
    }
}

```

```

        break;
    case '!':
        sb.Append(ch);
        if (!quoted)
            sb.Append(" ");
        break;
    default:
        sb.Append(ch);
        break;
    }
}
return sb.ToString();
}
}
static class Extensions
{
    public static void ForEach<T>(this IEnumerable<T> ie, Action<T> action)
    {
        foreach (var i in ie)
        {
            action(i);
        }
    }
}
}

```

## OCR Example Response

Upon success, the OCR results returned include text, bounding box for regions, lines, and words.

```

{
  "language": "en",
  "textAngle": -1.5000000000000335,
  "orientation": "Up",
  "regions": [
    {
      "boundingBox": "154,49,351,575",
      "lines": [
        {
          "boundingBox": "165,49,340,117",
          "words": [
            {
              "boundingBox": "165,49,63,109",
              "text": "A"
            },
            {
              "boundingBox": "261,50,244,116",
              "text": "GOAL"
            }
          ]
        }
      ],
    },
    {
      "boundingBox": "165,169,339,93",
      "words": [
        {
          "boundingBox": "165,169,339,93",
          "text": "WITHOUT"
        }
      ]
    },
    {
      "boundingBox": "159,264,342,117",
      "words": [
        {
          "boundingBox": "159,264,64,110",
          "text": "A"
        }
      ],
    },
  ],
}

```

```

        {
            "boundingBox": "255,266,246,115",
            "text": "PLAN"
        }
    ],
    {
        "boundingBox": "161,384,338,119",
        "words": [
            {
                "boundingBox": "161,384,86,113",
                "text": "IS"
            },
            {
                "boundingBox": "274,387,225,116",
                "text": "JUST"
            }
        ]
    },
    {
        "boundingBox": "154,506,341,118",
        "words": [
            {
                "boundingBox": "154,506,62,111",
                "text": "A"
            },
            {
                "boundingBox": "248,508,247,116",
                "text": "WISH"
            }
        ]
    }
]
}
]
}
}

```

## Text recognition with Computer Vision API using C#

Use the [RecognizeText method](#) to detect handwritten or printed text in an image and extract recognized characters into a machine-usable character stream.

### Handwriting recognition C# example

Create a new Console solution in Visual Studio, then replace Program.cs with the following code. Change the `uriBase` to use the location where you obtained your subscription keys, and replace the `subscriptionKey` value with your valid subscription key.

```

using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Text;

namespace CSHttpClientSample
{
    static class Program
    {
        // *****
        // *** Update or verify the following values. ***
        // *****

        // Replace the subscriptionKey string value with your valid subscription key.
    }
}

```



```

const string subscriptionKey = "13hc77781f7e4b19b5fcd72a8df7156";

// Replace or verify the region.
//
// You must use the same region in your REST API call as you used to obtain your subscription keys.
// For example, if you obtained your subscription keys from the westus region, replace
// "westcentralus" in the URI below with "westus".
//
// NOTE: Free trial subscription keys are generated in the westcentralus region, so if you are using
// a free trial subscription key, you should not need to change this region.
const string uriBase = "https://westcentralus.api.cognitive.microsoft.com/vision/v1.0/recognizeText";

static void Main()
{
    // Get the path and filename to process from the user.
    Console.WriteLine("Handwriting Recognition:");
    Console.Write("Enter the path to an image with handwritten text you wish to read: ");
    string imagePath = Console.ReadLine();

    // Execute the REST API call.
    ReadHandwrittenText(imagePath);

    Console.WriteLine("\nPlease wait a moment for the results to appear. Then, press Enter to exit...\n");
    Console.ReadLine();
}

/// <summary>
/// Gets the handwritten text from the specified image file by using the Computer Vision REST API.
/// </summary>
/// <param name="imagePath">The image file with handwritten text.</param>
static async void ReadHandwrittenText(string imagePath)
{
    HttpClient client = new HttpClient();

    // Request headers.
    client.DefaultRequestHeaders.Add("Ocp-Apim-Subscription-Key", subscriptionKey);

    // Request parameter. Set "handwriting" to false for printed text.
    string requestParameters = "handwriting=true";

    // Assemble the URI for the REST API Call.
    string uri = uriBase + "?" + requestParameters;

    HttpResponseMessage response = null;

    // This operation requires two REST API calls. One to submit the image for processing,
    // the other to retrieve the text found in the image. This value stores the REST API
    // location to call to retrieve the text.
    string operationLocation = null;

    // Request body. Posts a locally stored JPEG image.
    byte[] byteData = GetImageAsByteArray(imagePath);
    ByteArrayContent content = new ByteArrayContent(byteData);

    // This example uses content type "application/octet-stream".
    // You can also use "application/json" and specify an image URL.
    content.Headers.ContentType = new MediaTypeHeaderValue("application/octet-stream");

    // The first REST call starts the async process to analyze the written text in the image.
    response = await client.PostAsync(uri, content);

    // The response contains the URI to retrieve the result of the process.
    if (response.IsSuccessStatusCode)
        operationLocation = response.Headers.GetValues("Operation-Location").FirstOrDefault();
    else
    {
        // Display the JSON error data.
    }
}

```

```

        Console.WriteLine("\nError:\n");
        Console.WriteLine(JsonPrettyPrint(await response.Content.ReadAsStringAsync()));
        return;
    }

    // The second REST call retrieves the text written in the image.
    //
    // Note: The response may not be immediately available. Handwriting recognition is an
    // async operation that can take a variable amount of time depending on the length
    // of the handwritten text. You may need to wait or retry this operation.
    //
    // This example checks once per second for ten seconds.
    string contentString;
    int i = 0;
    do
    {
        System.Threading.Thread.Sleep(1000);
        response = await client.GetAsync(operationLocation);
        contentString = await response.Content.ReadAsStringAsync();
        ++i;
    }
    while (i < 10 && contentString.IndexOf("\"status\": \"Succeeded\"" ) == -1);

    if (i == 10 && contentString.IndexOf("\"status\": \"Succeeded\"" ) == -1)
    {
        Console.WriteLine("\nTimeout error.\n");
        return;
    }

    // Display the JSON response.
    Console.WriteLine("\nResponse:\n");
    Console.WriteLine(JsonPrettyPrint(contentString));
}

/// <summary>
/// Returns the contents of the specified file as a byte array.
/// </summary>
/// <param name="imageFilePath">The image file to read.</param>
/// <returns>The byte array of the image data.</returns>
static byte[] GetImageAsByteArray(string imageFilePath)
{
    FileStream fileStream = new FileStream(imageFilePath, FileMode.Open, FileAccess.Read);
    BinaryReader binaryReader = new BinaryReader(fileStream);
    return binaryReader.ReadBytes((int)fileStream.Length);
}

/// <summary>
/// Formats the given JSON string by adding line breaks and indents.
/// </summary>
/// <param name="json">The raw JSON string to format.</param>
/// <returns>The formatted JSON string.</returns>
static string JsonPrettyPrint(string json)
{
    if (string.IsNullOrEmpty(json))
        return string.Empty;

    json = json.Replace(Environment.NewLine, "").Replace("\t", "");

    string INDENT_STRING = "  ";
    var indent = 0;
    var quoted = false;
    var sb = new StringBuilder();
    for (var i = 0; i < json.Length; i++)
    {
        var ch = json[i];
        switch (ch)
        {

```

```

        case '{':
        case '[':
            sb.Append(ch);
            if (!quoted)
            {
                sb.AppendLine();
                Enumerable.Range(0, ++indent).ForEach(item => sb.Append(INDENT_STRING));
            }
            break;
        case '}':
        case ']':
            if (!quoted)
            {
                sb.AppendLine();
                Enumerable.Range(0, --indent).ForEach(item => sb.Append(INDENT_STRING));
            }
            sb.Append(ch);
            break;
        case '"':
            sb.Append(ch);
            bool escaped = false;
            var index = i;
            while (index > 0 && json[--index] == '\\')
                escaped = !escaped;
            if (!escaped)
                quoted = !quoted;
            break;
        case ',':
            sb.Append(ch);
            if (!quoted)
            {
                sb.AppendLine();
                Enumerable.Range(0, indent).ForEach(item => sb.Append(INDENT_STRING));
            }
            break;
        case ':':
            sb.Append(ch);
            if (!quoted)
                sb.Append(" ");
            break;
        default:
            sb.Append(ch);
            break;
    }
}
return sb.ToString();
}
}
static class Extensions
{
    public static void ForEach<T>(this IEnumerable<T> ie, Action<T> action)
    {
        foreach (var i in ie)
        {
            action(i);
        }
    }
}
}
}

```

## Handwriting recognition response

A successful response is returned in JSON. Following is an example of a successful response:

```

{
  "status": "Succeeded",
  "recognitionResult": {
    "text": "1234567890"
  }
}

```

```
"lines": [
  {
    "boundingBox": [
      99,
      195,
      1309,
      45,
      1340,
      292,
      130,
      442
    ],
    "text": "when you write them down",
    "words": [
      {
        "boundingBox": [
          152,
          191,
          383,
          154,
          341,
          421,
          110,
          458
        ],
        "text": "when"
      },
      {
        "boundingBox": [
          436,
          145,
          607,
          118,
          565,
          385,
          394,
          412
        ],
        "text": "you"
      },
      {
        "boundingBox": [
          644,
          112,
          873,
          76,
          831,
          343,
          602,
          379
        ],
        "text": "write"
      },
      {
        "boundingBox": [
          895,
          72,
          1092,
          41,
          1050,
          308,
          853,
          339
        ],
        "text": "them"
      },
      {
        "boundingBox": [
          1140,
```

```
33,
1400,
0,
1359,
258,
1098,
300
],
"text": "down"
}
]
},
{
  "boundingBox": [
    142,
    222,
    1252,
    62,
    1269,
    180,
    159,
    340
  ],
  "text": "You remember things better",
  "words": [
    {
      "boundingBox": [
        140,
        223,
        267,
        205,
        288,
        324,
        162,
        342
      ],
      "text": "You"
    },
    {
      "boundingBox": [
        314,
        198,
        740,
        137,
        761,
        256,
        335,
        317
      ],
      "text": "remember"
    },
    {
      "boundingBox": [
        761,
        134,
        1026,
        95,
        1047,
        215,
        782,
        253
      ],
      "text": "things"
    },
    {
      "boundingBox": [
        1046,
        92,
        1285,
```

```

        58,
        1307,
        177,
        1068,
        212
    ],
    "text": "better"
}
],
},
{
    "boundingBox": [
        155,
        405,
        537,
        338,
        557,
        449,
        175,
        516
    ],
    "text": "by hand",
    "words": [
        {
            "boundingBox": [
                146,
                408,
                266,
                387,
                301,
                495,
                181,
                516
            ],
            "text": "by"
        },
        {
            "boundingBox": [
                290,
                383,
                569,
                334,
                604,
                443,
                325,
                491
            ],
            "text": "hand"
        }
    ]
}
]
}
}

```

# Computer Vision Java Quick Starts

12/7/2017 • 15 min to read • [Edit Online](#)

This article provides information and code samples to help you quickly get started using Java and the Computer Vision API to accomplish the following tasks:

- [Analyze an image](#)
- [Use a Domain-Specific Model](#)
- [Intelligently generate a thumbnail](#)
- [Detect and extract printed text from an image](#)
- [Detect and extract handwritten text from an image](#)

## Prerequisites

- Get the Microsoft Computer Vision Android SDK [here](#).
- To use the Computer Vision API, you need a subscription key. You can get free subscription keys [here](#).

## Analyze an image with Computer Vision API using Java

With the [Analyze Image method](#), you can extract visual features based on image content. You can upload an image or specify an image URL and choose which features to return, including:

- A detailed list of tags related to the image content.
- A description of image content in a complete sentence.
- The coordinates, gender, and age of any faces contained in the image.
- The ImageType (clip art or a line drawing).
- The dominant color, the accent color, or whether an image is black & white.
- The category defined in this [taxonomy](#).
- Does the image contain adult or sexually suggestive content?

### Analyze an image Java example request

To run the sample, perform the following steps:

1. Create a new Command Line App.
2. Replace the Main class with the following code (keep any `package` statements).
3. Replace the `subscriptionKey` value with your valid subscription key.
4. Change the `uriBase` value to use the location where you obtained your subscription keys, if necessary.
5. Download these global libraries from the Maven Repository to the `lib` directory in your project:
  - `org.apache.httpcomponents:httpclient:4.2.4`
  - `org.json:json:20170516`
6. Run 'Main'.

```
// This sample uses the Apache HTTP client library(org.apache.httpcomponents:httpclient:4.2.4)
// and the org.json library (org.json:json:20170516).
```

```
import java.net.URI;
import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpPost;
```

```

import org.apache.http.entity.StringEntity;
import org.apache.http.client.utils.URIBuilder;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.util.EntityUtils;
import org.json.JSONObject;

public class Main
{
    // *****
    // *** Update or verify the following values. ***
    // *****

    // Replace the subscriptionKey string value with your valid subscription key.
    public static final String subscriptionKey = "13hc77781f7e4b19b5fcdd72a8df7156";

    // Replace or verify the region.
    //
    // You must use the same region in your REST API call as you used to obtain your subscription keys.
    // For example, if you obtained your subscription keys from the westus region, replace
    // "westcentralus" in the URI below with "westus".
    //
    // NOTE: Free trial subscription keys are generated in the westcentralus region, so if you are using
    // a free trial subscription key, you should not need to change this region.
    public static final String uriBase = "https://westcentralus.api.cognitive.microsoft.com/vision/v1.0/analyze";

    public static void main(String[] args)
    {
        HttpClient httpclient = new DefaultHttpClient();

        try
        {
            URIBuilder builder = new URIBuilder(uriBase);

            // Request parameters. All of them are optional.
            builder.setParameter("visualFeatures", "Categories,Description,Color");
            builder.setParameter("language", "en");

            // Prepare the URI for the REST API call.
            URI uri = builder.build();
            HttpPost request = new HttpPost(uri);

            // Request headers.
            request.setHeader("Content-Type", "application/json");
            request.setHeader("Ocp-Apim-Subscription-Key", subscriptionKey);

            // Request body.
            StringEntity reqEntity = new StringEntity(
{"url":"https://upload.wikimedia.org/wikipedia/commons/1/12/Broadway_and_Times_Square_by_night.jpg\"}");
            request.setEntity(reqEntity);

            // Execute the REST API call and get the response entity.
            HttpResponse response = httpclient.execute(request);
            HttpEntity entity = response.getEntity();

            if (entity != null)
            {
                // Format and display the JSON response.
                String jsonString = EntityUtils.toString(entity);
                JSONObject json = new JSONObject(jsonString);
                System.out.println("REST Response:\n");
                System.out.println(json.toString(2));
            }
        }
        catch (Exception e)
        {
            // Display error message.
            System.out.println(e.getMessage());
        }
    }
}

```



```
    },  
  }  
}
```

## Analyze an image response

A successful response is returned in JSON. The program should produce output similar to the following:

REST Response:

```
{  
  "metadata": {  
    "width": 1826,  
    "format": "Jpeg",  
    "height": 2436  
  },  
  "color": {  
    "dominantColorForeground": "Brown",  
    "isBWImg": false,  
    "accentColor": "B74314",  
    "dominantColorBackground": "Brown",  
    "dominantColors": ["Brown"]  
  },  
  "requestId": "bbffe1a1-4fa3-4a6b-a4d5-a4964c58a811",  
  "description": {  
    "captions": [{  
      "confidence": 0.8241405091548035,  
      "text": "a group of people on a city street filled with traffic at night"  
    }],  
    "tags": [  
      "outdoor",  
      "building",  
      "street",  
      "city",  
      "busy",  
      "people",  
      "filled",  
      "traffic",  
      "many",  
      "table",  
      "car",  
      "group",  
      "walking",  
      "bunch",  
      "crowded",  
      "large",  
      "night",  
      "light",  
      "standing",  
      "man",  
      "tall",  
      "umbrella",  
      "riding",  
      "sign",  
      "crowd"  
    ]  
  },  
  "categories": [{  
    "score": 0.625,  
    "name": "outdoor_street"  
  }]  
}
```

Process finished with exit code 0

# Use a Domain-Specific Model

The Domain-Specific Model is a model trained to identify a specific set of objects in an image. The two domain-specific models that are currently available are celebrities and landmarks. The following example identifies a landmark in an image.

## Landmark Java example request

To run the sample, perform the following steps:

1. Create a new Command Line App.
2. Replace the Main class with the following code (keep any `package` statements).
3. Replace the `subscriptionKey` value with your valid subscription key.
4. Change the `uriBase` value to use the location where you obtained your subscription keys, if necessary.
5. Download these libraries from the Maven Repository to the `lib` directory in your project:
  - `org.apache.httpcomponents:httpclient:4.2.4`
  - `org.json:json:20170516`
6. Run 'Main'.

```
// This sample uses the Apache HTTP client library(org.apache.httpcomponents:httpclient:4.2.4)
// and the org.json library (org.json:json:20170516).

import java.net.URI;
import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.entity.StringEntity;
import org.apache.http.client.utils.URIBuilder;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.util.EntityUtils;
import org.json.JSONObject;

public class Main
{
    // *****
    // *** Update or verify the following values. ***
    // *****

    // Replace the subscriptionKey string value with your valid subscription key.
    public static final String subscriptionKey = "13hc77781f7e4b19b5fcd72a8df7156";

    // Replace or verify the region.
    //
    // You must use the same region in your REST API call as you used to obtain your subscription keys.
    // For example, if you obtained your subscription keys from the westus region, replace
    // "westcentralus" in the URI below with "westus".
    //
    // NOTE: Free trial subscription keys are generated in the westcentralus region, so if you are using
    // a free trial subscription key, you should not need to change this region.
    //
    // Also, if you want to use the celebrities model, change "landmarks" to "celebrities" here and in
    // uriBuilder.setParameter to use the Celebrities model.
    public static final String uriBase = "https://westcentralus.api.cognitive.microsoft.com/vision/v1.0/models/landmarks/analyze";

    public static void main(String[] args)
    {
        HttpClient httpClient = new DefaultHttpClient();

        try
        {
            URIBuilder uriBuilder = new URIBuilder(uriBase);
```

```

// Request parameters.
// To use the Celebrities model, change "landmarks" to "celebrities" here and in uriBase.
uriBuilder.setParameter("model", "landmarks");

// Prepare the URI for the REST API call.
URI uri = uriBuilder.build();
HttpPost request = new HttpPost(uri);

// Request headers.
request.setHeader("Content-Type", "application/json");
request.setHeader("Ocp-Apim-Subscription-Key", subscriptionKey);

// Request body.
StringEntity requestEntity = new StringEntity("{\"url\":\"https://upload.wikimedia.org/wikipedia/commons/2/23/Space_Needle_2011-07-04.jpg\"}");
request.setEntity(requestEntity);

// Execute the REST API call and get the response entity.
HttpResponse response = httpClient.execute(request);
HttpEntity entity = response.getEntity();

if (entity != null)
{
    // Format and display the JSON response.
    String jsonString = EntityUtils.toString(entity);
    JSONObject json = new JSONObject(jsonString);
    System.out.println("REST Response:\n");
    System.out.println(json.toString(2));
}
}
catch (Exception e)
{
    // Display error message.
    System.out.println(e.getMessage());
}
}
}

```

### Landmark example response

A successful response is returned in JSON. The program should produce output similar to the following:

REST Response:

```

{
  "result": { "landmarks": [{
    "confidence": 0.9998178,
    "name": "Space Needle"
  } ] },
  "metadata": {
    "width": 2096,
    "format": "Jpeg",
    "height": 4132
  },
  "requestId": "8551c2b7-fcf9-4932-aff3-87e7f744343f"
}

```

Process finished with exit code 0

## Get a thumbnail with Computer Vision API using Java

Use the [Get Thumbnail method](#) to crop an image based on its region of interest (ROI) to the height and width you desire. The aspect ratio you set for the thumbnail can be different from the aspect ratio of the input image.

## Get a thumbnail Java example request

To run the sample, perform the following steps:

1. Create a new Command Line App.
2. Replace the Main class with the following code (keep any `package` statements).
3. Replace the `subscriptionKey` value with your valid subscription key.
4. Change the `uriBase` value to use the location where you obtained your subscription keys, if necessary.
5. Download these libraries from the Maven Repository to the `lib` directory in your project:
  - `org.apache.httpcomponents:httpclient:4.2.4`
  - `org.json:json:20170516`
6. Run 'Main'.

```
// This sample uses the Apache HTTP client library(org.apache.httpcomponents:httpclient:4.2.4)
// and the org.json library (org.json:json:20170516).

import java.awt.*;
import javax.swing.*;
import java.net.URI;
import java.io.InputStream;
import javax.imageio.ImageIO;
import java.awt.image.BufferedImage;
import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.entity.StringEntity;
import org.apache.http.client.utils.URIBuilder;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.util.EntityUtils;
import org.json.JSONObject;

public class Main
{
    // *****
    // *** Update or verify the following values. ***
    // *****

    // Replace the subscriptionKey string value with your valid subscription key.
    public static final String subscriptionKey = "13hc77781f7e4b19b5fcd72a8df7156";

    // Replace or verify the region.
    //
    // You must use the same region in your REST API call as you used to obtain your subscription keys.
    // For example, if you obtained your subscription keys from the westus region, replace
    // "westcentralus" in the URI below with "westus".
    //
    // NOTE: Free trial subscription keys are generated in the westcentralus region, so if you are using
    // a free trial subscription key, you should not need to change this region.
    public static final String uriBase = "https://westcentralus.api.cognitive.microsoft.com/vision/v1.0/generateThumbnail";

    public static void main(String[] args)
    {
        HttpClient httpClient = new DefaultHttpClient();

        try
        {
            URIBuilder uriBuilder = new URIBuilder(uriBase);

            // Request parameters.
            uriBuilder.setParameter("width", "100");
            uriBuilder.setParameter("height", "150");
            uriBuilder.setParameter("smartCropping", "true");
```

```

// Prepare the URI for the REST API call.
URI uri = uriBuilder.build();
HttpPost request = new HttpPost(uri);

// Request headers.
request.setHeader("Content-Type", "application/json");
request.setHeader("Ocp-Apim-Subscription-Key", subscriptionKey);

// Request body.
StringEntity requestEntity = new StringEntity("
{\"url\": \"https://upload.wikimedia.org/wikipedia/commons/9/94/Bloodhound_Puppy.jpg\"}");
request.setEntity(requestEntity);

// Execute the REST API call and get the response entity.
HttpResponse response = httpClient.execute(request);
HttpEntity entity = response.getEntity();

// Check for success.
if (response.getStatusLine().getStatusCode() == 200)
{
    // Display the thumbnail.
    System.out.println("\nDisplaying thumbnail.\n");
    displayImage(entity.getContent());
}
else
{
    // Format and display the JSON error message.
    String jsonString = EntityUtils.toString(entity);
    JSONObject json = new JSONObject(jsonString);
    System.out.println("Error:\n");
    System.out.println(json.toString(2));
}
}
catch (Exception e)
{
    System.out.println(e.getMessage());
}
}

// Displays the given input stream as an image.
private static void displayImage(InputStream inputStream)
{
    try {
        BufferedImage bufferedImage = ImageIO.read(inputStream);

        ImageIcon imageIcon = new ImageIcon(bufferedImage);

        JLabel jLabel = new JLabel();
        jLabel.setIcon(imageIcon);

        JFrame jFrame = new JFrame();
        jFrame.setLayout(new FlowLayout());
        jFrame.setSize(100, 150);

        jFrame.add(jLabel);
        jFrame.setVisible(true);
        jFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    catch (Exception e) {
        System.out.println(e.getMessage());
    }
}
}

```

## Get a thumbnail response

A successful response contains the thumbnail image binary. If the request fails, the response contains an error code

and a message to help determine what went wrong.

## Optical Character Recognition (OCR) with Computer Vision API using Java

Use the [Optical Character Recognition \(OCR\) method](#) to detect printed text in an image and extract recognized characters into a machine-usable character stream.

### OCR Java example request

To run the sample, perform the following steps:

1. Create a new Command Line App.
2. Replace the Main class with the following code (keep any `package` statements).
3. Replace the `subscriptionKey` value with your valid subscription key.
4. Change the `uriBase` value to use the location where you obtained your subscription keys, if necessary.
5. Download these libraries from the Maven Repository to the `lib` directory in your project:
  - `org.apache.httpcomponents:httpclient:4.2.4`
  - `org.json:json:20170516`
6. Run 'Main'.

```
// This sample uses the Apache HTTP client library(org.apache.httpcomponents:httpclient:4.2.4)
// and the org.json library (org.json:json:20170516).

import java.net.URI;
import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.entity.StringEntity;
import org.apache.http.client.utils.URIBuilder;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.util.EntityUtils;
import org.json.JSONObject;

public class Main
{
    // *****
    // *** Update or verify the following values. ***
    // *****

    // Replace the subscriptionKey string value with your valid subscription key.
    public static final String subscriptionKey = "13hc77781f7e4b19b5fcdd72a8df7156";

    // Replace or verify the region.
    //
    // You must use the same region in your REST API call as you used to obtain your subscription keys.
    // For example, if you obtained your subscription keys from the westus region, replace
    // "westcentralus" in the URI below with "westus".
    //
    // NOTE: Free trial subscription keys are generated in the westcentralus region, so if you are using
    // a free trial subscription key, you should not need to change this region.
    //
    // Also, if you want to use the celebrities model, change "landmarks" to "celebrities" here and in
    // uriBuilder.setParameter to use the Celebrities model.
    public static final String uriBase = "https://westcentralus.api.cognitive.microsoft.com/vision/v1.0/ocr";

    public static void main(String[] args)
    {
        HttpClient httpClient = new DefaultHttpClient();
```

```

try
{
    // NOTE: You must use the same location in your REST call as you used to obtain your subscription keys.
    // For example, if you obtained your subscription keys from westus, replace "westcentralus" in the
    // URL below with "westus".
    UriBuilder uriBuilder = new UriBuilder(uriBase);

    uriBuilder.setParameter("language", "unk");
    uriBuilder.setParameter("detectOrientation ", "true");

    // Request parameters.
    Uri uri = uriBuilder.build();
    HttpPost request = new HttpPost(uri);

    // Request headers.
    request.setHeader("Content-Type", "application/json");
    request.setHeader("Ocp-Apim-Subscription-Key", subscriptionKey);

    // Request body.
    StringEntity requestEntity =
        new StringEntity("
{\"url\": \"https://upload.wikimedia.org/wikipedia/commons/thumb/a/af/Atomist_quote_from_Democritus.png/338px-
Atomist_quote_from_Democritus.png\"}");
    request.setEntity(requestEntity);

    // Execute the REST API call and get the response entity.
    HttpResponse response = httpClient.execute(request);
    HttpEntity entity = response.getEntity();

    if (entity != null)
    {
        // Format and display the JSON response.
        String jsonString = EntityUtils.toString(entity);
        JSONObject json = new JSONObject(jsonString);
        System.out.println("REST Response:\n");
        System.out.println(json.toString(2));
    }
}
catch (Exception e)
{
    // Display error message.
    System.out.println(e.getMessage());
}
}
}

```

## OCR example response

A successful response is returned in JSON. The OCR results include the detected text and bounding boxes for regions, lines, and words.

The program should produce output similar to the following:

REST Response:

```

{
  "orientation": "Up",
  "regions": [{
    "boundingBox": "21,16,304,451",
    "lines": [
      {
        "boundingBox": "28,16,288,41",
        "words": [{
          "boundingBox": "28,16,288,41",
          "text": "NOTHING"
        }]
      }
    ],
  },
  ,
  ,

```

```
{
  "boundingBox": "27,66,283,52",
  "words": [{
    "boundingBox": "27,66,283,52",
    "text": "EXISTS"
  }]
},
{
  "boundingBox": "27,128,292,49",
  "words": [{
    "boundingBox": "27,128,292,49",
    "text": "EXCEPT"
  }]
},
{
  "boundingBox": "24,188,292,54",
  "words": [{
    "boundingBox": "24,188,292,54",
    "text": "ATOMS"
  }]
},
{
  "boundingBox": "22,253,297,32",
  "words": [
    {
      "boundingBox": "22,253,105,32",
      "text": "AND"
    },
    {
      "boundingBox": "144,253,175,32",
      "text": "EMPTY"
    }
  ]
},
{
  "boundingBox": "21,298,304,60",
  "words": [{
    "boundingBox": "21,298,304,60",
    "text": "SPACE."
  }]
},
{
  "boundingBox": "26,387,294,37",
  "words": [
    {
      "boundingBox": "26,387,210,37",
      "text": "Everything"
    },
    {
      "boundingBox": "249,389,71,27",
      "text": "else"
    }
  ]
},
{
  "boundingBox": "127,431,198,36",
  "words": [
    {
      "boundingBox": "127,431,31,29",
      "text": "is"
    },
    {
      "boundingBox": "172,431,153,36",
      "text": "opinion."
    }
  ]
}
]
```



```
"textAngle": 0,  
"language": "en"  
}
```

Process finished with exit code 0

## Text Recognition with Computer Vision API using Java

Use the [RecognizeText method](#) to detect handwritten or printed text in an image and extract recognized characters into a machine-usable character stream.

### Handwriting recognition Java example

To run the sample, perform the following steps:

1. Create a new Command Line App.
2. Replace the Main class with the following code (keep any `package` statements).
3. Replace the `subscriptionKey` value with your valid subscription key.
4. Change the `uriBase` value to use the location where you obtained your subscription keys, if necessary.
5. Download these libraries from the Maven Repository to the `lib` directory in your project:
  - `org.apache.httpcomponents:httpclient:4.2.4`
  - `org.json:json:20170516`
6. Run 'Main'.

```
// This sample uses the Apache HTTP client library(org.apache.httpcomponents:httpclient:4.2.4)  
// and the org.json library (org.json:json:20170516).  
  
import java.net.URI;  
import org.apache.http.HttpEntity;  
import org.apache.http.HttpResponse;  
import org.apache.http.client.HttpClient;  
import org.apache.http.client.methods.HttpGet;  
import org.apache.http.client.methods.HttpPost;  
import org.apache.http.entity.StringEntity;  
import org.apache.http.impl.client.DefaultHttpClient;  
import org.apache.http.util.EntityUtils;  
import org.apache.http.Header;  
import org.json.JSONObject;  
  
public class Main  
{  
    // *****  
    // *** Update or verify the following values. ***  
    // *****  
  
    // Replace the subscriptionKey string value with your valid subscription key.  
    public static final String subscriptionKey = "13hc77781f7e4b19b5fcdd72a8df7156";  
  
    // Replace or verify the region.  
    //  
    // You must use the same region in your REST API call as you used to obtain your subscription keys.  
    // For example, if you obtained your subscription keys from the westus region, replace  
    // "westcentralus" in the URI below with "westus".  
    //  
    // NOTE: Free trial subscription keys are generated in the westcentralus region, so if you are using  
    // a free trial subscription key, you should not need to change this region.  
    //  
    // Also, for printed text, set "handwriting" to false.  
    public static final String uriBase = "https://westcentralus.api.cognitive.microsoft.com/vision/v1.0/recognizeText?handwriting=true";  
  
    public static void main(String[] args)  
    {  
        HttpClient httpClient = new DefaultHttpClient();
```

```

HttpClient textClient = new DefaultHttpClient();
HttpClient resultClient = new DefaultHttpClient();

try
{
    // This operation requires two REST API calls. One to submit the image for processing,
    // the other to retrieve the text found in the image.
    //
    // Begin the REST API call to submit the image for processing.
    URI uri = new URI(uriBase);
    HttpPost textRequest = new HttpPost(uri);

    // Request headers. Another valid content type is "application/octet-stream".
    textRequest.setHeader("Content-Type", "application/json");
    textRequest.setHeader("Ocp-Apim-Subscription-Key", subscriptionKey);

    // Request body.
    StringEntity requestEntity =
        new StringEntity(
{"url\":"https://upload.wikimedia.org/wikipedia/commons/thumb/d/dd/Cursive_Writing_on_Notebook_paper.jpg/800px-
Cursive_Writing_on_Notebook_paper.jpg\"}");
    textRequest.setEntity(requestEntity);

    // Execute the first REST API call to detect the text.
    HttpResponse textResponse = textClient.execute(textRequest);

    // Check for success.
    if (textResponse.getStatusLine().getStatusCode() != 202)
    {
        // Format and display the JSON error message.
        HttpEntity entity = textResponse.getEntity();
        String jsonString = EntityUtils.toString(entity);
        JSONObject json = new JSONObject(jsonString);
        System.out.println("Error:\n");
        System.out.println(json.toString(2));
        return;
    }

    String operationLocation = null;

    // The 'Operation-Location' in the response contains the URI to retrieve the recognized text.
    Header[] responseHeaders = textResponse.getAllHeaders();
    for (Header header : responseHeaders) {
        if (header.getName().equals("Operation-Location"))
        {
            // This string is the URI where you can get the text recognition operation result.
            operationLocation = header.getValue();
            break;
        }
    }

    // NOTE: The response may not be immediately available. Handwriting recognition is an
    // async operation that can take a variable amount of time depending on the length
    // of the text you want to recognize. You may need to wait or retry this operation.

    System.out.println("\nHandwritten text submitted. Waiting 10 seconds to retrieve the recognized text.\n");
    Thread.sleep(10000);

    // Execute the second REST API call and get the response.
    HttpGet resultRequest = new HttpGet(operationLocation);
    resultRequest.setHeader("Ocp-Apim-Subscription-Key", subscriptionKey);

    HttpResponse resultResponse = resultClient.execute(resultRequest);
    HttpEntity responseEntity = resultResponse.getEntity();

    if (responseEntity != null)
    {
        // Format and display the JSON response.
        String jsonString = EntityUtils.toString(responseEntity);
    }
}

```

```

        JSONObject json = new JSONObject(jsonString);
        System.out.println("Text recognition result response: \n");
        System.out.println(json.toString(2));
    }
}
catch (Exception e)
{
    System.out.println(e.getMessage());
}
}
}

```

## Handwriting recognition Java example response

A successful response is returned in JSON. The handwriting recognition results include the detected text and bounding boxes for regions, lines, and words.

The program should produce output similar to the following:

Handwritten text submitted. Waiting 10 seconds to retrieve the recognized text.

Text recognition result response:

```

{
  "status": "Succeeded",
  "recognitionResult": { "lines": [
    {
      "boundingBox": [
        2,
        84,
        783,
        96,
        782,
        154,
        1,
        148
      ],
      "words": [
        {
          "boundingBox": [
            6,
            86,
            92,
            87,
            71,
            151,
            0,
            150
          ],
          "text": "Pack"
        },
        {
          "boundingBox": [
            86,
            87,
            172,
            88,
            150,
            152,
            64,
            151
          ],
          "text": "my"
        },
        {
          "boundingBox": [
            165,
            22
          ],

```

```
88,  
241,  
89,  
219,  
152,  
144,  
152  
],  
"text": "box"  
},  
{  
  "boundingBox": [  
    234,  
    89,  
    343,  
    90,  
    322,  
    154,  
    213,  
    152  
  ],  
  "text": "with"  
},  
{  
  "boundingBox": [  
    347,  
    90,  
    432,  
    91,  
    411,  
    154,  
    325,  
    154  
  ],  
  "text": "five"  
},  
{  
  "boundingBox": [  
    432,  
    91,  
    538,  
    92,  
    516,  
    154,  
    411,  
    154  
  ],  
  "text": "dozen"  
},  
{  
  "boundingBox": [  
    554,  
    92,  
    696,  
    94,  
    675,  
    154,  
    533,  
    154  
  ],  
  "text": "liquor"  
},  
{  
  "boundingBox": [  
    710,  
    94,  
    800,  
    96,  
    800,
```

```
154,
688,
154
],
"text": "jugs"
}
],
"text": "Pack my box with five dozen liquor jugs"
},
{
  "boundingBox": [
    2,
    52,
    65,
    46,
    69,
    89,
    7,
    95
  ],
  "words": [{
    "boundingBox": [
      0,
      62,
      79,
      39,
      94,
      82,
      0,
      105
    ],
    "text": "dog"
  }],
  "text": "dog"
},
{
  "boundingBox": [
    6,
    2,
    771,
    13,
    770,
    75,
    5,
    64
  ],
  "words": [
    {
      "boundingBox": [
        8,
        4,
        92,
        5,
        77,
        71,
        0,
        71
      ],
      "text": "The"
    },
    {
      "boundingBox": [
        89,
        5,
        188,
        5,
        173,
        72,
        74,

```

```
71
],
"text": "quick"
},
{
  "boundingBox": [
    188,
    5,
    323,
    6,
    308,
    73,
    173,
    72
  ],
  "text": "brown"
},
{
  "boundingBox": [
    316,
    6,
    386,
    6,
    371,
    73,
    302,
    73
  ],
  "text": "fox"
},
{
  "boundingBox": [
    396,
    7,
    508,
    7,
    493,
    74,
    381,
    73
  ],
  "text": "jumps"
},
{
  "boundingBox": [
    501,
    7,
    604,
    8,
    589,
    75,
    487,
    74
  ],
  "text": "over"
},
{
  "boundingBox": [
    600,
    8,
    673,
    8,
    658,
    75,
    586,
    75
  ],
  "text": "the"
},
],
```

```
{  
  "boundingBox": [  
    670,  
    8,  
    800,  
    9,  
    787,  
    76,  
    655,  
    75  
  ],  
  "text": "lazy"  
}  
],  
"text": "The quick brown fox jumps over the lazy"  
}  
}]  
}
```

Process finished with exit code 0

# Computer Vision JavaScript Quick Starts

6/27/2017 • 15 min to read • [Edit Online](#)

This article provides information and code samples to help you quickly get started using JavaScript and the Computer Vision API to accomplish the following tasks:

- [Analyze an image](#)
- [Use a Domain-Specific Model](#)
- [Intelligently generate a thumbnail](#)
- [Detect and extract text from an Image](#)

Learn more about obtaining free Subscription Keys [here](#)

Most of these samples use jQuery 1.9.0. For a sample that uses JavaScript without jQuery, see the sample, [Intelligently generate a thumbnail](#).

## Analyze an image with Computer Vision API using JavaScript

With the [Analyze Image method](#), you can extract visual features based on image content. You can upload an image or specify an image URL and choose which features to return, including:

- A detailed list of tags related to the image content.
- A description of image content in a complete sentence.
- The coordinates, gender, and age of any faces contained in the image.
- The ImageType (clipart or a line drawing)
- The dominant color, the accent color, or whether an image is black & white.
- The category defined in this [taxonomy](#).
- Whether the image contains pornographic or sexually suggestive content.

### Analyze an image JavaScript example request

To run the sample, perform the following steps:

1. Copy the following and save it to a file such as `analyze.html`.
2. Replace the `subscriptionKey` value with your valid subscription key.
3. Change the `uriBase` value to use the location where you obtained your subscription keys.
4. Drag-and-drop the file into your browser.
5. Click the `Analyze image` button.

```
<!DOCTYPE html>
<html>
<head>
  <title>Analyze Sample</title>
  <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.9.0/jquery.min.js"></script>
</head>
<body>

<script type="text/javascript">
function processImage() {
  // *****
  // *** Update or verify the following values. ***
  // *****

  // Replace the subscriptionKey string value with your valid subscription key.
```



```

var subscriptionKey = "13hc77781f7e4b19b5fcdd72a8df7156";

// Replace or verify the region.
//
// You must use the same region in your REST API call as you used to obtain your subscription keys.
// For example, if you obtained your subscription keys from the westus region, replace
// "westcentralus" in the URI below with "westus".
//
// NOTE: Free trial subscription keys are generated in the westcentralus region, so if you are using
// a free trial subscription key, you should not need to change this region.
var uriBase = "https://westcentralus.api.cognitive.microsoft.com/vision/v1.0/analyze";

// Request parameters.
var params = {
    "visualFeatures": "Categories,Description,Color",
    "details": "",
    "language": "en",
};

// Display the image.
var sourceImageUrl = document.getElementById("inputImage").value;
document.querySelector("#sourceImage").src = sourceImageUrl;

// Perform the REST API call.
$.ajax({
    url: uriBase + "?" + $.param(params),

    // Request headers.
    beforeSend: function(xhrObj) {
        xhrObj.setRequestHeader("Content-Type", "application/json");
        xhrObj.setRequestHeader("Ocp-Apim-Subscription-Key", subscriptionKey);
    },

    type: "POST",

    // Request body.
    data: '{"url": "' + sourceImageUrl + '"}',
})

.done(function(data) {
    // Show formatted JSON on webpage.
    $("#responseTextArea").val(JSON.stringify(data, null, 2));
})

.fail(function(jqXHR, textStatus, errorThrown) {
    // Display error message.
    var errorString = (errorThrown === "") ? "Error. " : errorThrown + " (" + jqXHR.status + "): ";
    errorString += (jqXHR.responseText === "") ? "" : "jQuery.parseJSON(jqXHR.responseText).message";
    alert(errorString);
});
};
</script>

<h1>Analyze image:</h1>
Enter the URL to an image of a natural or artificial landmark, then click the <strong>Analyze image</strong> button.
<br><br>
Image to analyze: <input type="text" name="inputImage" id="inputImage"
value="http://upload.wikimedia.org/wikipedia/commons/3/3c/Shaki_waterfall.jpg" />
<button onclick="processImage()">Analyze image</button>
<br><br>
<div id="wrapper" style="width:1020px; display:table;">
  <div id="jsonOutput" style="width:600px; display:table-cell;">
    Response:
    <br><br>
    <textarea id="responseTextArea" class="UIInput" style="width:580px; height:400px;"></textarea>
  </div>
  <div id="imageDiv" style="width:420px; display:table-cell;">
    Source image:
    <br><br>

```

```
</div>  
<img id="sourceImage" width="400" />  
</div>  
</div>  
</body>  
</html>
```

### Analyze an image response

A successful response is returned in JSON. Following is an example of a successful response:

```

{
  "categories": [
    {
      "name": "outdoor_water",
      "score": 0.9921875
    }
  ],
  "description": {
    "tags": [
      "nature",
      "water",
      "waterfall",
      "outdoor",
      "rock",
      "mountain",
      "rocky",
      "grass",
      "hill",
      "top",
      "covered",
      "hillside",
      "standing",
      "side",
      "group",
      "walking",
      "white",
      "man",
      "large",
      "snow",
      "grazing",
      "forest",
      "slope",
      "herd",
      "river",
      "giraffe",
      "field"
    ],
    "captions": [
      {
        "text": "a large waterfall over a rocky cliff",
        "confidence": 0.9165146827843689
      }
    ]
  },
  "requestId": "63d3c630-7f3d-43d7-8a97-143012fc53f4",
  "metadata": {
    "width": 1280,
    "height": 959,
    "format": "Jpeg"
  },
  "color": {
    "dominantColorForeground": "Grey",
    "dominantColorBackground": "Green",
    "dominantColors": [
      "Grey",
      "Green"
    ],
    "accentColor": "4D5E2F",
    "isBWImg": false
  }
}

```

## Use a Domain-Specific model

The Domain-Specific Model is a model trained to identify a specific set of objects in an image. The two domain-

specific models that are currently available are celebrities and landmarks. The following example identifies a landmark in an image.

### Landmark JavaScript example request

To run the sample, perform the following steps:

1. Copy the following and save it to a file such as `landmark.html`.
2. Replace the `subscriptionKey` value with your valid subscription key.
3. Change the `uriBase` value to use the location where you obtained your subscription keys.
4. Drag-and-drop the file into your browser.
5. Click the `Analyze image` button.

```
<!DOCTYPE html>
<html>
<head>
  <title>Landmark Sample</title>
  <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.9.0/jquery.min.js"></script>
</head>
<body>

<script type="text/javascript">
  function processImage() {
    // *****
    // *** Update or verify the following values. ***
    // *****

    // Replace the subscriptionKey string value with your valid subscription key.
    var subscriptionKey = "13hc77781f7e4b19b5fcdd72a8df7156";

    // Replace or verify the region.
    //
    // You must use the same region in your REST API call as you used to obtain your subscription keys.
    // For example, if you obtained your subscription keys from the westus region, replace
    // "westcentralus" in the URI below with "westus".
    //
    // NOTE: Free trial subscription keys are generated in the westcentralus region, so if you are using
    // a free trial subscription key, you should not need to change this region.
    //
    // Also, if you want to use the celebrities model, change "landmarks" to "celebrities" here and in
    // uriBuilder.setParameter to use the Celebrities model.
    var uriBase = "https://westcentralus.api.cognitive.microsoft.com/vision/v1.0/models/landmarks/analyze";

    // Request parameters.
    var params = {
      "model": "landmarks", // Use "model": "celebrities" to use the Celebrities model.
    };

    // Display the image.
    var sourceImageUrl = document.getElementById("inputImage").value;
    document.querySelector("#sourceImage").src = sourceImageUrl;

    // Perform the REST API call.
    $.ajax({
      url: uriBase + "?" + $.param(params),

      // Request headers.
      beforeSend: function(xhrObj) {
        xhrObj.setRequestHeader("Content-Type", "application/json");
        xhrObj.setRequestHeader("Ocp-Apim-Subscription-Key", subscriptionKey);
      },

      type: "POST",

      // Request body.
      data: '{"url":"' + sourceImageUrl + '"}'
```

```

        }, { url: "sourceImage" } );
    })

    .done(function(data) {
        // Show formatted JSON on webpage.
        $("#responseTextArea").val(JSON.stringify(data, null, 2));
    })

    .fail(function(jqXHR, textStatus, errorThrown) {
        // Display error message.
        var errorString = (errorThrown === "") ? "Error. " : errorThrown + " (" + jqXHR.status + "): ";
        errorString += (jqXHR.responseText === "") ? "" : jQuery.parseJSON(jqXHR.responseText).message;
        alert(errorString);
    });
};
</script>

<h1>Landmark image:</h1>
Enter the URL to an image of a natural or artificial landmark, then click the <strong>Analyze image</strong> button.
<br><br>
Landscape image to analyze: <input type="text" name="inputImage" id="inputImage"
value="https://upload.wikimedia.org/wikipedia/commons/2/23/Space_Needle_2011-07-04.jpg" />
<button onclick="processImage()">Analyze image</button>
<br><br>
<div id="wrapper" style="width:1020px; display:table;">
  <div id="jsonOutput" style="width:600px; display:table-cell;">
    Response:
    <br><br>
    <textarea id="responseTextArea" class="UIInput" style="width:580px; height:400px;"></textarea>
  </div>
  <div id="imageDiv" style="width:420px; display:table-cell;">
    Source image:
    <br><br>
    <img id="sourceImage" width="400" />
  </div>
</div>
</body>
</html>

```

## Landmark example response

A successful response is returned in JSON. Following is an example of a successful response:

```

{
  "requestId": "fe0d4539-7a21-4fc6-b7af-3bb24beba390",
  "metadata": {
    "width": 2096,
    "height": 4132,
    "format": "Jpeg"
  },
  "result": {
    "landmarks": [
      {
        "name": "Space Needle",
        "confidence": 0.9998178
      }
    ]
  }
}

```

## Get a thumbnail with Computer Vision API using JavaScript

Use the [Get Thumbnail method](#) to crop an image based on its region of interest (ROI) to the height and width you desire, even if the aspect ratio differs from the input image.

## Get a thumbnail JavaScript example request

To run the sample, perform the following steps:

1. Copy the following and save it to a file such as `thumbnail.html`.
2. Replace the `subscriptionKey` value with your valid subscription key.
3. Change the `uriBase` value to use the location where you obtained your subscription keys.
4. Drag-and-drop the file into your browser.
5. Click the `Generate thumbnail` button.

```
<!DOCTYPE html>
<html>
<head>
  <title>Thumbnail Sample</title>
</head>
<body>

<script type="text/javascript">
function processImage() {
  // *****
  // *** Update or verify the following values. ***
  // *****

  // Replace the subscriptionKey string value with your valid subscription key.
  var subscriptionKey = "13hc77781f7e4b19b5fcdd72a8df7156";

  // Replace or verify the region.
  //
  // You must use the same region in your REST API call as you used to obtain your subscription keys.
  // For example, if you obtained your subscription keys from the westus region, replace
  // "westcentralus" in the URI below with "westus".
  //
  // NOTE: Free trial subscription keys are generated in the westcentralus region, so if you are using
  // a free trial subscription key, you should not need to change this region.
  var uriBase = "https://westcentralus.api.cognitive.microsoft.com/vision/v1.0/generateThumbnail";

  // Request parameters.
  var params = "?width=100&height=150&smartCropping=true";

  // Display the source image.
  var sourceImageUrl = document.getElementById("inputImage").value;
  document.querySelector("#sourceImage").src = sourceImageUrl;

  // Prepare the REST API call:

  // Create the HTTP Request object.
  var xhr = new XMLHttpRequest();

  // Identify the request as a POST, with the URL and parameters.
  xhr.open("POST", uriBase + params);

  // Add the request headers.
  xhr.setRequestHeader("Content-Type", "application/json");
  xhr.setRequestHeader("Ocp-Apim-Subscription-Key", subscriptionKey);

  // Set the response type to "blob" for the thumbnail image data.
  xhr.responseType = "blob";

  // Process the result of the REST API call.
  xhr.onreadystatechange = function(e) {
    if(xhr.readyState === XMLHttpRequest.DONE) {

      // Thumbnail successfully created.
      if (xhr.status === 200) {
        // Show response headers.
        var s = JSON.stringify(xhr.getAllResponseHeaders(), null, 2);
```

```

document.getElementById("responseTextArea").value = JSON.stringify(xhr.getAllResponseHeaders(), null, 2);

// Show thumbnail image.
var urlCreator = window.URL || window.webkitURL;
var imageUrl = urlCreator.createObjectURL(this.response);
document.querySelector("#thumbnailImage").src = imageUrl;
} else {
// Display the error message. The error message is the response body as a JSON string.
// The code in this code block extracts the JSON string from the blob response.
var reader = new FileReader();

// This event fires after the blob has been read.
reader.addEventListener('loadend', (e) => {
    document.getElementById("responseTextArea").value = JSON.stringify(JSON.parse(e.srcElement.result), null, 2);
});

// Start reading the blob as text.
reader.readAsText(xhr.response);
}
}
}

// Execute the REST API call.
xhr.send({'url': ' ' + sourceImageUrl + '});
};
</script>

<h1>Generate thumbnail image:</h1>
Enter the URL to an image to use in creating a thumbnail image, then click the <strong>Generate thumbnail</strong> button.
<br><br>
Image for thumbnail: <input type="text" name="inputImage" id="inputImage"
value="https://upload.wikimedia.org/wikipedia/commons/thumb/5/56/Shorkie_Poo_Puppy.jpg/1280px-Shorkie_Poo_Puppy.jpg" />
<button onclick="processImage()">Generate thumbnail</button>
<br><br>
<div id="wrapper" style="width:1160px; display:table;">
  <div id="jsonOutput" style="width:600px; display:table-cell;">
    Response:
    <br><br>
    <textarea id="responseTextArea" class="UIInput" style="width:580px; height:400px;"></textarea>
  </div>
  <div id="imageDiv" style="width:420px; display:table-cell;">
    Source image:
    <br><br>
    <img id="sourceImage" width="400" />
  </div>
  <div id="thumbnailDiv" style="width:140px; display:table-cell;">
    Thumbnail:
    <br><br>
    <img id="thumbnailImage" />
  </div>
</div>
</div>
</body>
</html>

```

### Get a thumbnail response

A successful response contains the thumbnail image binary. If the request failed, the response contains an error code and message to help determine what went wrong.

## Optical Character Recognition (OCR) with Computer Vision API using JavaScript

Use the [Optical Character Recognition \(OCR\) method](#) to detect text in an image and extract recognized characters into a machine-usable character stream.

## OCR JavaScript example request

To run the sample, perform the following steps:

1. Copy the following and save it to a file such as `ocr.html`.
2. Replace the `subscriptionKey` value with your valid subscription key.
3. Change the `uriBase` value to use the location where you obtained your subscription keys.
4. Drag-and-drop the file into your browser.
5. Click the `Read image` button.

```
<!DOCTYPE html>
<html>
<head>
  <title>OCR Sample</title>
  <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.9.0/jquery.min.js"></script>
</head>
<body>

<script type="text/javascript">
function processImage() {
  // *****
  // *** Update or verify the following values. ***
  // *****

  // Replace the subscriptionKey string value with your valid subscription key.
  var subscriptionKey = "13hc77781f7e4b19b5fcdd72a8df7156";

  // Replace or verify the region.
  //
  // You must use the same region in your REST API call as you used to obtain your subscription keys.
  // For example, if you obtained your subscription keys from the westus region, replace
  // "westcentralus" in the URI below with "westus".
  //
  // NOTE: Free trial subscription keys are generated in the westcentralus region, so if you are using
  // a free trial subscription key, you should not need to change this region.
  var uriBase = "https://westcentralus.api.cognitive.microsoft.com/vision/v1.0/ocr";

  // Request parameters.
  var params = {
    "language": "unk",
    "detectOrientation": "true",
  };

  // Display the image.
  var sourceImageUrl = document.getElementById("inputImage").value;
  document.querySelector("#sourceImage").src = sourceImageUrl;

  // Perform the REST API call.
  $.ajax({
    url: uriBase + "?" + $.param(params),

    // Request headers.
    beforeSend: function(jqXHR){
      jqXHR.setRequestHeader("Content-Type", "application/json");
      jqXHR.setRequestHeader("Ocp-Apim-Subscription-Key", subscriptionKey);
    },

    type: "POST",

    // Request body.
    data: '{"url": ' + "'" + sourceImageUrl + "'" + '}',
  })

  .done(function(data) {
    // Show formatted JSON on webpage.
    $("#responseTextArea").val(JSON.stringify(data, null, 2));
  });
}
```



```

})

.fail(function(jqXHR, textStatus, errorThrown) {
    // Display error message.
    var errorString = (errorThrown === "") ? "Error. " : errorThrown + " (" + jqXHR.status + "): ";
    errorString += (jqXHR.responseText === "") ? "" : (jQuery.parseJSON(jqXHR.responseText).message) ?
        jQuery.parseJSON(jqXHR.responseText).message : jQuery.parseJSON(jqXHR.responseText).error.message;
    alert(errorString);
});
};
</script>

<h1>Optical Character Recognition (OCR):</h1>
Enter the URL to an image of printed text, then click the <strong>Read image</strong> button.
<br><br>
Image to read: <input type="text" name="inputImage" id="inputImage"
value="https://upload.wikimedia.org/wikipedia/commons/thumb/a/af/Atomist_quote_from_Democritus.png/338px-
Atomist_quote_from_Democritus.png" />
<button onclick="processImage()">Read image</button>
<br><br>
<div id="wrapper" style="width:1020px; display:table;">
    <div id="jsonOutput" style="width:600px; display:table-cell;">
        Response:
        <br><br>
        <textarea id="responseTextArea" class="UIInput" style="width:580px; height:400px;"></textarea>
    </div>
    <div id="imageDiv" style="width:420px; display:table-cell;">
        Source image:
        <br><br>
        <img id="sourceImage" width="400" />
    </div>
</div>
</body>
</html>

```

## OCR example response

A successful response is returned in JSON. The OCR results returned include text, bounding box for regions, lines, and words.

Following is an example of a successful response:

```

{
  "language": "en",
  "textAngle": 0,
  "orientation": "Up",
  "regions": [
    {
      "boundingBox": "21,16,304,451",
      "lines": [
        {
          "boundingBox": "28,16,288,41",
          "words": [
            {
              "boundingBox": "28,16,288,41",
              "text": "NOTHING"
            }
          ]
        }
      ],
    },
    {
      "boundingBox": "27,66,283,52",
      "words": [
        {
          "boundingBox": "27,66,283,52",
          "text": "EXISTS"
        }
      ]
    }
  ]
}

```

```

},
{
  "boundingBox": "27,128,292,49",
  "words": [
    {
      "boundingBox": "27,128,292,49",
      "text": "EXCEPT"
    }
  ]
},
{
  "boundingBox": "24,188,292,54",
  "words": [
    {
      "boundingBox": "24,188,292,54",
      "text": "ATOMS"
    }
  ]
},
{
  "boundingBox": "22,253,297,32",
  "words": [
    {
      "boundingBox": "22,253,105,32",
      "text": "AND"
    },
    {
      "boundingBox": "144,253,175,32",
      "text": "EMPTY"
    }
  ]
},
{
  "boundingBox": "21,298,304,60",
  "words": [
    {
      "boundingBox": "21,298,304,60",
      "text": "SPACE."
    }
  ]
},
{
  "boundingBox": "26,387,294,37",
  "words": [
    {
      "boundingBox": "26,387,210,37",
      "text": "Everything"
    },
    {
      "boundingBox": "249,389,71,27",
      "text": "else"
    }
  ]
},
{
  "boundingBox": "127,431,198,36",
  "words": [
    {
      "boundingBox": "127,431,31,29",
      "text": "is"
    },
    {
      "boundingBox": "172,431,153,36",
      "text": "opinion."
    }
  ]
}
]
}
}

```

```
]
}
```

## Text Recognition with Computer Vision API using JavaScript

Use the [RecognizeText method](#) to detect handwritten or printed text in an image and extract recognized characters into a machine-usable character stream.

### Handwriting recognition Java example

To run the sample, perform the following steps:

1. Copy the following and save it to a file such as `handwriting.html`.
2. Replace the `subscriptionKey` value with your valid subscription key.
3. Change the `uriBase` value to use the location where you obtained your subscription keys.
4. Drag-and-drop the file into your browser.
5. Click the `Read image` button.

```
<!DOCTYPE html>
<html>
<head>
  <title>Handwriting Sample</title>
  <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.9.0/jquery.min.js"></script>
</head>
<body>

<script type="text/javascript">
function processImage() {
  // *****
  // *** Update or verify the following values. ***
  // *****

  // Replace the subscriptionKey string value with your valid subscription key.
  var subscriptionKey = "13hc77781f7e4b19b5fcdd72a8df7156";

  // Replace or verify the region.
  //
  // You must use the same region in your REST API call as you used to obtain your subscription keys.
  // For example, if you obtained your subscription keys from the westus region, replace
  // "westcentralus" in the URI below with "westus".
  //
  // NOTE: Free trial subscription keys are generated in the westcentralus region, so if you are using
  // a free trial subscription key, you should not need to change this region.
  var uriBase = "https://westcentralus.api.cognitive.microsoft.com/vision/v1.0/RecognizeText";

  // Request parameters.
  var params = {
    "handwriting": "true",
  };

  // Display the image.
  var sourceImageUrl = document.getElementById("inputImage").value;
  document.querySelector("#sourceImage").src = sourceImageUrl;

  // This operation requires two REST API calls. One to submit the image for processing,
  // the other to retrieve the text found in the image.
  //
  // Perform the first REST API call to submit the image for processing.
  $.ajax({
    url: uriBase + "?" + $.param(params),

    // Request headers.
    beforeSend: function(jqXHR){
      jqXHR.setRequestHeader("Content-Type" "application/json");
    }
  });
}
```

```

jqXHR.setRequestHeader("Content-Type", "application/json");
jqXHR.setRequestHeader("Ocp-Apim-Subscription-Key", subscriptionKey);
},

type: "POST",

// Request body.
data: '{"url":"' + sourceImageUrl + '"}',
})

.done(function(data, textStatus, jqXHR) {
    // Show progress.
    $("#responseTextArea").val("Handwritten text submitted. Waiting 10 seconds to retrieve the recognized text.");

    // Note: The response may not be immediately available. Handwriting recognition is an
    // async operation that can take a variable amount of time depending on the length
    // of the text you want to recognize. You may need to wait or retry this GET operation.
    //
    // Wait ten seconds before making the second REST API call.
    setTimeout(function () {
        // The "Operation-Location" in the response contains the URI to retrieve the recognized text.
        var operationLocation = jqXHR.getResponseHeader("Operation-Location");

        // Perform the second REST API call and get the response.
        $.ajax({
            url: operationLocation,

            // Request headers.
            beforeSend: function(jqXHR){
                jqXHR.setRequestHeader("Content-Type", "application/json");
                jqXHR.setRequestHeader("Ocp-Apim-Subscription-Key", subscriptionKey);
            },

            type: "GET",
        })

        .done(function(data) {
            // Show formatted JSON on webpage.
            $("#responseTextArea").val(JSON.stringify(data, null, 2));
        })

        .fail(function(jqXHR, textStatus, errorThrown) {
            // Display error message.
            var errorString = (errorThrown === "") ? "Error. " : errorThrown + " (" + jqXHR.status + "): ";
            errorString += (jqXHR.responseText === "") ? "" : (jQuery.parseJSON(jqXHR.responseText).message) ?
                jQuery.parseJSON(jqXHR.responseText).message : jQuery.parseJSON(jqXHR.responseText).error.message;
            alert(errorString);
        });
    }, 10000);
})

.fail(function(jqXHR, textStatus, errorThrown) {
    // Put the JSON description into the text area.
    $("#responseTextArea").val(JSON.stringify(jqXHR, null, 2));

    // Display error message.
    var errorString = (errorThrown === "") ? "Error. " : errorThrown + " (" + jqXHR.status + "): ";
    errorString += (jqXHR.responseText === "") ? "" : (jQuery.parseJSON(jqXHR.responseText).message) ?
        jQuery.parseJSON(jqXHR.responseText).message : jQuery.parseJSON(jqXHR.responseText).error.message;
    alert(errorString);
});
};
</script>
<h1>Read handwritten image image:</h1>
Enter the URL to an image of handwritten text, then click the <strong>Read image</strong> button.
<br><br>
Image to read: <input type="text" name="inputImage" id="inputImage"
value="https://upload.wikimedia.org/wikipedia/commons/thumb/d/dd/Cursive_Writing_on_Notebook_paper.jpg/800px-
Cursive_Writing_on_Notebook_paper.jpg" />
<button onclick="processImage()">Read image</button>

```

```

<button onclick="processImage()" >Read image</button>
<br><br>
<div id="wrapper" style="width:1020px; display:table;">
  <div id="jsonOutput" style="width:600px; display:table-cell;">
    Response:
    <br><br>
    <textarea id="responseTextArea" class="UIInput" style="width:580px; height:400px;"></textarea>
  </div>
  <div id="imageDiv" style="width:420px; display:table-cell;">
    Source image:
    <br><br>
    <img id="sourceImage" width="400" />
  </div>
</div>
</body>
</html>

```

## Handwriting example response

A successful response is returned in JSON. The handwriting results returned include text, bounding box for regions, lines, and words.

Following is an example of a successful response:

```

{
  "status": "Succeeded",
  "recognitionResult": {
    "lines": [
      {
        "boundingBox": [
          2,
          84,
          783,
          96,
          782,
          154,
          1,
          148
        ],
        "text": "Pack my box with five dozen liquor jugs",
        "words": [
          {
            "boundingBox": [
              6,
              86,
              92,
              87,
              71,
              151,
              0,
              150
            ],
            "text": "Pack"
          },
          {
            "boundingBox": [
              86,
              87,
              172,
              88,
              150,
              152,
              64,
              151
            ],
            "text": "my"
          }
        ]
      }
    ]
  }
}

```

```
{
  "boundingBox": [
    165,
    88,
    241,
    89,
    219,
    152,
    144,
    152
  ],
  "text": "box"
},
{
  "boundingBox": [
    234,
    89,
    343,
    90,
    322,
    154,
    213,
    152
  ],
  "text": "with"
},
{
  "boundingBox": [
    347,
    90,
    432,
    91,
    411,
    154,
    325,
    154
  ],
  "text": "five"
},
{
  "boundingBox": [
    432,
    91,
    538,
    92,
    516,
    154,
    411,
    154
  ],
  "text": "dozen"
},
{
  "boundingBox": [
    554,
    92,
    696,
    94,
    675,
    154,
    533,
    154
  ],
  "text": "liquor"
},
{
  "boundingBox": [
    710,
    94,
    800,
    94,
    710,
    154,
    800,
    154
  ],
  "text": "liquor"
}
```

```
      800,
      96,
      800,
      154,
      688,
      154
    ],
    "text": "jugs"
  }
]
},
{
  "boundingBox": [
    2,
    52,
    65,
    46,
    69,
    89,
    7,
    95
  ],
  "text": "dog",
  "words": [
    {
      "boundingBox": [
        0,
        62,
        79,
        39,
        94,
        82,
        0,
        105
      ],
      "text": "dog"
    }
  ]
},
{
  "boundingBox": [
    6,
    2,
    771,
    13,
    770,
    75,
    5,
    64
  ],
  "text": "The quick brown fox jumps over the lazy",
  "words": [
    {
      "boundingBox": [
        8,
        4,
        92,
        5,
        77,
        71,
        0,
        71
      ],
      "text": "The"
    }
  ],
  {
    "boundingBox": [
      89,
      5,
```

```
188,
5,
173,
72,
74,
71
],
"text": "quick"
},
{
  "boundingBox": [
    188,
    5,
    323,
    6,
    308,
    73,
    173,
    72
  ],
  "text": "brown"
},
{
  "boundingBox": [
    316,
    6,
    386,
    6,
    371,
    73,
    302,
    73
  ],
  "text": "fox"
},
{
  "boundingBox": [
    396,
    7,
    508,
    7,
    493,
    74,
    381,
    73
  ],
  "text": "jumps"
},
{
  "boundingBox": [
    501,
    7,
    604,
    8,
    589,
    75,
    487,
    74
  ],
  "text": "over"
},
{
  "boundingBox": [
    600,
    8,
    673,
    8,
    658,
    75,
```



```
586,  
75  
],  
"text": "the"  
},  
{  
  "boundingBox": [  
    670,  
    8,  
    800,  
    9,  
    787,  
    76,  
    655,  
    75  
  ],  
  "text": "lazy"  
}  
]  
}  
]  
}  
}
```

# Computer Vision PHP Quick Starts

6/27/2017 • 6 min to read • [Edit Online](#)

This article provides information and code samples to help you quickly get started using the Computer Vision API with PHP to accomplish the following tasks:

- [Analyze an image](#)
- [Use a Domain-Specific Model](#)
- [Intelligently generate a thumbnail](#)
- [Detect and extract text from an Image](#)

Learn more about obtaining free Subscription Keys [here](#)

## Analyze an Image With Computer Vision API Using PHP

With the [Analyze Image method](#), you can extract visual features based on image content. You can upload an image or specify an image URL and choose which features to return, including:

- The category defined in this [taxonomy](#).
- A detailed list of tags related to the image content.
- A description of image content in a complete sentence.
- The coordinates, gender, and age of any faces contained in the image.
- The ImageType (clipart or a line drawing)
- The dominant color, the accent color, or whether an image is black & white.
- Whether the image contains pornographic or sexually suggestive content.

### Analyze an Image PHP Example Request

Change the REST URL to use the location where you obtained your subscription keys, and replace the "Ocp-Apim-Subscription-Key" value with your valid subscription key.

```

<?php
// This sample uses the Apache HTTP client from HTTP Components (http://hc.apache.org/httpcomponents-client-ga/)
require_once 'HTTP/Request2.php';

// NOTE: You must use the same location in your REST call as you used to obtain your subscription keys.
// For example, if you obtained your subscription keys from westus, replace "westcentralus" in the
// URL below with "westus".
$request = new Http_Request2("https://westcentralus.api.cognitive.microsoft.com/vision/v1.0/analyze");
$url = $request->getUrl();

$headers = array(
    // Request headers
    'Content-Type' => 'application/json',

    // NOTE: Replace the "Ocp-Apim-Subscription-Key" value with a valid subscription key.
    'Ocp-Apim-Subscription-Key' => '13hc77781f7e4b19b5fcd72a8df7156',
);

$request->setHeader($headers);

$parameters = array(
    // Request parameters
    'visualFeatures' => 'Categories',
    'details' => '{string}',
    'language' => 'en',
);

$url->setQueryVariables($parameters);

$request->setMethod(HTTP_Request2::METHOD_POST);

// Request body
$request->setBody("{body}"); // Replace with the body, for example, {"url": "http://www.example.com/images/image.jpg"}

try
{
    $response = $request->send();
    echo $response->getBody();
}
catch (HttpException $ex)
{
    echo $ex;
}

?>

```

## Analyze an Image Response

A successful response is returned in JSON. Following is an example of a successful response:

```

{
  "categories": [
    {
      "name": "abstract_",
      "score": 0.00390625
    },
    {
      "name": "people_",
      "score": 0.83984375,
      "detail": {
        "celebrities": [
          {
            "name": "Satya Nadella",
            "faceRectangle": {
              "left": 597,
              "top": 162,

```

```
        "width": 248,
        "height": 248
    },
    "confidence": 0.999028444
}
]
}
}
],
"adult": {
    "isAdultContent": false,
    "isRacyContent": false,
    "adultScore": 0.0934349000453949,
    "racyScore": 0.068613491952419281
},
"tags": [
    {
        "name": "person",
        "confidence": 0.98979085683822632
    },
    {
        "name": "man",
        "confidence": 0.94493889808654785
    },
    {
        "name": "outdoor",
        "confidence": 0.938492476940155
    },
    {
        "name": "window",
        "confidence": 0.89513939619064331
    }
],
"description": {
    "tags": [
        "person",
        "man",
        "outdoor",
        "window",
        "glasses"
    ],
    "captions": [
        {
            "text": "Satya Nadella sitting on a bench",
            "confidence": 0.48293603002174407
        }
    ]
},
"requestId": "0dbec5ad-a3d3-4f7e-96b4-dfd57efe967d",
"metadata": {
    "width": 1500,
    "height": 1000,
    "format": "Jpeg"
},
"faces": [
    {
        "age": 44,
        "gender": "Male",
        "faceRectangle": {
            "left": 593,
            "top": 160,
            "width": 250,
            "height": 250
        }
    }
],
"color": {
    "dominantColorForeground": "Brown",
    "dominantColorBackground": "Brown",
    "dominantColors": [
```

```
    "Brown",  
    "Black"  
  ],  
  "accentColor": "873B59",  
  "isBWImg": false  
},  
"imageType": {  
  "clipArtType": 0,  
  "lineDrawingType": 0  
}  
}
```

## Use a Domain-Specific Model

The Domain-Specific Model is a model trained to identify a specific set of objects in an image. The two domain-specific models that are currently available are celebrities and landmarks. The following example identifies a landmark in an image.

### Landmark PHP Example Request

Change the REST URL to use the location where you obtained your subscription keys, and replace the "Ocp-Apim-Subscription-Key" value with your valid subscription key.

```

<html>
<head>
  <title>PHP Sample</title>
</head>
<body>
<?php
// This sample uses PEAR (https://pear.php.net/package/HTTP_Request2/download)
require_once 'HTTP/Request2.php';

// NOTE: You must use the same location in your REST call as you used to obtain your subscription keys.
// For example, if you obtained your subscription keys from westus, replace "westcentralus" in the
// URL below with "westus".
//
// Also, change "landmarks" to "celebrities" in the url to use the Celebrities model.
$request = new Http_Request2('https://westcentralus.api.cognitive.microsoft.com/vision/v1.0/models/landmarks/analyze');
$url = $request->getUrl();

$headers = array(
  // Request headers
  'Content-Type' => 'application/json',

  // NOTE: Replace the "Ocp-Apim-Subscription-Key" value with a valid subscription key.
  'Ocp-Apim-Subscription-Key' => '13hc77781f7e4b19b5fcd72a8df7156',
);

$request->setHeader($headers);

$parameters = array(
  // Request parameters
  'model' => 'landmarks', // Use 'model' => 'celebrities' to use the Celebrities model.
);

$url->setQueryVariables($parameters);

$request->setMethod(HTTP_Request2::METHOD_POST);

// Request body
$body = json_encode(array(
  // Request body parameters
  'url' => 'https://upload.wikimedia.org/wikipedia/commons/2/23/Space_Needle_2011-07-04.jpg',
));
$request->setBody($body);

try
{
  $response = $request->send();
  echo "<pre>" . json_encode(json_decode($response->getBody()), JSON_PRETTY_PRINT) . "</pre>";
}
catch (HttpException $ex)
{
  echo "<pre>" . $ex . "</pre>";
}
?>
</body>
</html>

```

## Landmark Example Response

A successful response is returned in JSON. Following is an example of a successful response:

```
{
  "requestId": "0663b074-8eb3-4fab-a72e-4c31a49bd22e",
  "metadata": {
    "width": 2096,
    "height": 4132,
    "format": "Jpeg"
  },
  "result": {
    "landmarks": [
      {
        "name": "Space Needle",
        "confidence": 0.9998178
      }
    ]
  }
}
```

## Get a Thumbnail with Computer Vision API Using PHP

Use the [Get Thumbnail method](#) to crop an image based on its region of interest (ROI) to the height and width you desire, even if the aspect ratio differs from the input image.

### Get a Thumbnail PHP Example Request

Change the REST URL to use the location where you obtained your subscription keys, and replace the "Ocp-Apim-Subscription-Key" value with your valid subscription key.

```

<?php
// This sample uses the Apache HTTP client from HTTP Components (http://hc.apache.org/httpcomponents-client-ga/)
require_once 'HTTP/Request2.php';

// NOTE: You must use the same location in your REST call as you used to obtain your subscription keys.
// For example, if you obtained your subscription keys from westus, replace "westcentralus" in the
// URL below with "westus".
$request = new Http_Request2('https://westcentralus.api.cognitive.microsoft.com/vision/v1.0/generateThumbnail');
$url = $request->getUrl();

$headers = array(
    // Request headers
    'Content-Type' => 'application/json',

    // NOTE: Replace the "Ocp-Apim-Subscription-Key" value with a valid subscription key.
    'Ocp-Apim-Subscription-Key' => '13hc77781f7e4b19b5fcd72a8df7156',
);

$request->setHeader($headers);

$params = array(
    // Request parameters
    'width' => '{number}', // Replace "{number}" with the desired width of your thumbnail.
    'height' => '{number}', // Replace "{number}" with the desired height of your thumbnail.
    'smartCropping' => 'true',
);

$url->setQueryVariables($params);

$request->setMethod(HTTP_Request2::METHOD_POST);

// Request body
$request->setBody("{body}"); // Replace "{body}" with the body. For example, '{"url": "http://www.example.com/images/image.jpg"}'

try
{
    $response = $request->send();
    echo $response->getBody();
}
catch (HttpException $ex)
{
    echo $ex;
}

?>

```

### Get a Thumbnail Response

A successful response contains the thumbnail image binary. If the request failed, the response contains an error code and a message to help determine what went wrong.

## Optical Character Recognition (OCR) with Computer Vision API Using PHP

Use the [Optical Character Recognition \(OCR\) method](#) to detect text in an image and extract recognized characters into a machine-usable character stream.

### OCR PHP Example Request

Change the REST URL to use the location where you obtained your subscription keys, and replace the "Ocp-Apim-Subscription-Key" value with your valid subscription key.



```

<?php
// This sample uses the Apache HTTP client from HTTP Components (http://hc.apache.org/httpcomponents-client-ga/)
require_once 'HTTP/Request2.php';

$request = new Http_Request2("https://westcentralus.api.cognitive.microsoft.com/vision/v1.0/ocr");
$url = $request->getUrl();

$headers = array(
    // Request headers
    'Content-Type' => 'application/json',

    // NOTE: Replace the "Ocp-Apim-Subscription-Key" value with a valid subscription key.
    'Ocp-Apim-Subscription-Key' => '13hc77781f7e4b19b5fcdd72a8df7156',
);

$request->setHeader($headers);

$params = array(
    // Request parameters
    'language' => 'unk',
    'detectOrientation' => 'true',
);

$url->setQueryVariables($params);

$request->setMethod(HTTP_Request2::METHOD_POST);

// Request body
$request->setBody("{body}"); // Replace "{body}" with the body. For example, '{"url": "http://www.example.com/images/image.jpg"}'

try
{
    $response = $request->send();
    echo $response->getBody();
}
catch (HttpException $ex)
{
    echo $ex;
}

?>

```

## OCR Example Response

Upon success, the OCR results returned include text, bounding box for regions, lines, and words.

```
{
  "language": "en",
  "textAngle": -2.0000000000000338,
  "orientation": "Up",
  "regions": [
    {
      "boundingBox": "462,379,497,258",
      "lines": [
        {
          "boundingBox": "462,379,497,74",
          "words": [
            {
              "boundingBox": "462,379,41,73",
              "text": "A"
            },
            {
              "boundingBox": "523,379,153,73",
              "text": "GOAL"
            },
            {
              "boundingBox": "694,379,265,74",
              "text": "WITHOUT"
            }
          ]
        },
        {
          "boundingBox": "565,471,289,74",
          "words": [
            {
              "boundingBox": "565,471,41,73",
              "text": "A"
            },
            {
              "boundingBox": "626,471,150,73",
              "text": "PLAN"
            },
            {
              "boundingBox": "801,472,53,73",
              "text": "IS"
            }
          ]
        },
        {
          "boundingBox": "519,563,375,74",
          "words": [
            {
              "boundingBox": "519,563,149,74",
              "text": "JUST"
            },
            {
              "boundingBox": "683,564,41,72",
              "text": "A"
            },
            {
              "boundingBox": "741,564,153,73",
              "text": "WISH"
            }
          ]
        }
      ]
    }
  ]
}
```

# Computer Vision Python Quick Starts

2/5/2018 • 9 min to read • [Edit Online](#)

This article provides information and code samples to help you quickly get started using the Computer Vision API with Python to accomplish the following tasks:

- [Analyze an image](#)
- [Use a domain-specific Model](#)
- [Intelligently generate a thumbnail](#)
- [Detect and extract printed text from an image](#)
- [Detect and extract handwritten text from an image](#)

To use the Computer Vision API, you need a subscription key. You can get free subscription keys [here](#).

You can run this example as a Jupyter notebook on [MyBinder](#) by clicking on the launch Binder badge:

launch binder

## Analyze an image with Computer Vision API using Python

With the [Analyze Image method](#), you can extract visual features based on image content. You can upload an image or specify an image URL and choose which features to return, including:

- A detailed list of tags related to the image content.
- A description of image content in a complete sentence.
- The coordinates, gender, and age of any faces contained in the image.
- The ImageType (clip art or a line drawing).
- The dominant color, the accent color, or whether an image is black & white.
- The category defined in this [taxonomy](#).
- Does the image contain adult or sexually suggestive content?

### Analyze an image

To begin analyzing images, replace `subscription_key` with a valid API key that you obtained earlier.

```
subscription_key = "cfa2ac95fcf04101b79b839837876d16"
assert subscription_key
```

Next, ensure that region in `vision_base_url` corresponds to the one where you generated the API key ( `westus` , `westcentralus` , etc.). If you are using a free trial subscription key, you do not need to make any changes here.

```
vision_base_url = "https://westcentralus.api.cognitive.microsoft.com/vision/v1.0/"
```

The image analysis URL looks like the following (see REST API docs [here](#)):

```
https://[location].api.cognitive.microsoft.com/vision/v1.0/analyze[?visualFeatures][&details][&language]
```

```
vision_analyze_url = vision_base_url + "analyze"
```

To begin analyzing an image, set `image_url` to the URL of any image that you want to analyze.

```
image_url = "https://upload.wikimedia.org/wikipedia/commons/thumb/1/12/Broadway_and_Times_Square_by_night.jpg/450px-Broadway_and_Times_Square_by_night.jpg"
```

The following block uses the `requests` library in Python to call out to the Computer Vision `analyze` API and return the results as a JSON object. The API key is passed in via the `headers` dictionary and the types of features to recognize via the `params` dictionary. To see the full list of options that can be used, refer to the [REST API documentation](#) for image analysis.

```
import requests
headers = {'Ocp-Apim-Subscription-Key': subscription_key }
params = {'visualFeatures': 'Categories,Description,Color'}
data = {'url': image_url}
response = requests.post(vision_analyze_url, headers=headers, params=params, json=data)
response.raise_for_status()
analysis = response.json()
```

The `analysis` object contains various fields that describe the image. The most relevant caption for the image can be obtained from the `descriptions` property.

```
image_caption = analysis["description"]["captions"][0]["text"].capitalize()
print(image_caption)
```

A group of people on a city street at night

The following lines of code display the image and overlay it with the inferred caption.

```
%matplotlib inline
from PIL import Image
from io import BytesIO
import matplotlib.pyplot as plt
image = Image.open(BytesIO(requests.get(image_url).content))
plt.imshow(image)
plt.axis("off")
_ = plt.title(image_caption, size="x-large", y=-0.1)
```

## Use a domain-specific model

A [domain-specific model](#) is a model trained to identify a specific set of objects in an image. The two domain-specific models that are currently available are *celebrities* and *landmarks*.

To view the list of domain-specific models supported, you can make the following request against the service.

```
model_url = vision_base_url + "models"
headers = {'Ocp-Apim-Subscription-Key': subscription_key}
models = requests.get(model_url, headers=headers).json()
[model["name"] for model in models["models"]]
```

['celebrities', 'landmarks']

### Landmark identification

To begin using the domain-specific model for landmarks, set `image_url` to point to an image to be analyzed.

```
image_url = "https://upload.wikimedia.org/wikipedia/commons/f/f6/Bunker_Hill_Monument_2005.jpg"
```

The service end point to analyze images for landmarks can be constructed as follows:

```
landmark_analyze_url = vision_base_url + "models/landmarks/analyze"  
print(landmark_analyze_url)
```

```
https://westcentralus.api.cognitive.microsoft.com/vision/v1.0/models/landmarks/analyze
```

The image in `image_url` can now be analyzed for any landmarks. The identified landmark is stored in `landmark_name`.

```
headers = {'Ocp-Apim-Subscription-Key': subscription_key}  
params = {'model': 'landmarks'}  
data = {'url': image_url}  
response = requests.post(landmark_analyze_url, headers=headers, params=params, json=data)  
response.raise_for_status()  
  
analysis = response.json()  
assert analysis["result"]["landmarks"] is not []  
  
landmark_name = analysis["result"]["landmarks"][0]["name"].capitalize()
```

```
image = Image.open(BytesIO(requests.get(image_url).content))  
plt.imshow(image)  
plt.axis("off")  
_ = plt.title(landmark_name, size="x-large", y=-0.1)
```

## Celebrity identification

Along the same lines, the domain-specific model for identifying celebrities can be invoked as shown next. First set `image_url` to point to the image of a celebrity.

```
image_url = "https://upload.wikimedia.org/wikipedia/commons/d/d9/Bill_gates_portrait.jpg"
```

The service end point for detecting celebrity images can be constructed as follows:

```
celebrity_analyze_url = vision_base_url + "models/celebrities/analyze"  
print(celebrity_analyze_url)
```

```
https://westcentralus.api.cognitive.microsoft.com/vision/v1.0/models/celebrities/analyze
```

Next, the image in `image_url` can be analyzed for celebrities

```
headers = {'Ocp-Apim-Subscription-Key': subscription_key}  
params = {'model': 'celebrities'}  
data = {'url': image_url}  
response = requests.post(celebrity_analyze_url, headers=headers, params=params, json=data)  
response.raise_for_status()  
  
analysis = response.json()
```

```
print(analysis)
```

```
{'result': {'celebrities': [{'faceRectangle': {'top': 123, 'left': 156, 'width': 187, 'height': 187}, 'name': 'Bill Gates', 'confidence': 0.9993845224380493}],  
'requestId': 'd3eca546-0112-4574-817e-b6c5f43719bf', 'metadata': {'height': 521, 'width': 550, 'format': 'Jpeg'}}
```

The following lines of code extract the name and bounding box for one of the celebrities found:

```
assert analysis["result"]["celebrities"] is not []  
celebrity_info = analysis["result"]["celebrities"][0]  
celebrity_name = celebrity_info["name"]  
celebrity_face = celebrity_info["faceRectangle"]
```

Next, this information can be overlaid on top of the original image using the following lines of code:

```
from matplotlib.patches import Rectangle  
plt.figure(figsize=(5,5))  
  
image = Image.open(BytesIO(requests.get(image_url).content))  
ax = plt.imshow(image, alpha=0.6)  
origin = (celebrity_face["left"], celebrity_face["top"])  
p = Rectangle(origin, celebrity_face["width"], celebrity_face["height"],  
              fill=False, linewidth=2, color='b')  
ax.axes.add_patch(p)  
plt.text(origin[0], origin[1], celebrity_name, fontsize=20, weight="bold", va="bottom")  
_ = plt.axis("off")
```

## Get a thumbnail with Computer Vision API

Use the [Get Thumbnail method](#) to crop an image based on its region of interest (ROI) to the height and width you desire. The aspect ratio you set for the thumbnail can be different from the aspect ratio of the input image.

To generate the thumbnail for an image, first set `image_url` to point to its location.

```
image_url = "https://upload.wikimedia.org/wikipedia/commons/9/94/Bloodhound_Puppy.jpg"
```

The service end point to generate the thumbnail can be constructed as follows:

```
thumbnail_url = vision_base_url + "generateThumbnail"  
print(thumbnail_url)
```

```
https://westcentralus.api.cognitive.microsoft.com/vision/v1.0/generateThumbnail
```

Next, a 50-by-50 pixel thumbnail for the image can be generated by calling this service endpoint.

```
headers = {'Ocp-Apim-Subscription-Key': subscription_key}  
params = {'width': '50', 'height': '50', 'smartCropping': 'true'}  
data = {'url': image_url}  
response = requests.post(thumbnail_url, headers=headers, params=params, json=data)  
response.raise_for_status()
```

You can verify that the thumbnail is indeed 50-by-50 pixels using the Python Image Library.

```
thumbnail = Image.open(BytesIO(response.content))
print("Thumbnail is {0}-by-{1}".format(*thumbnail.size))
thumbnail
```

## Optical character recognition (OCR) with Computer Vision API

Use the [Optical Character Recognition \(OCR\) method](#) to detect text in an image and extract recognized characters into a machine-usable character stream.

To illustrate the OCR API, set `image_url` to point to the text to be recognized.

```
image_url = "https://upload.wikimedia.org/wikipedia/commons/thumb/a/af/Atomist_quote_from_Democritus.png/338px-Atomist_quote_from_Democritus.png"
```

The service end point for OCR for your region can be constructed as follows:

```
ocr_url = vision_base_url + "ocr"
print(ocr_url)
```

```
https://westcentralus.api.cognitive.microsoft.com/vision/v1.0/ocr
```

Next, you can call into the OCR service to get the text that was recognized along with bounding boxes. In the parameters shown, `"language": "unk"` automatically detects the language in the text and `"detectOrientation": "true"` automatically aligns the image. For more information, see the [REST API documentation](#).

```
headers = {'Ocp-Apim-Subscription-Key': subscription_key}
params = {'language': 'unk', 'detectOrientation': 'true'}
data = {'url': image_url}
response = requests.post(ocr_url, headers=headers, params=params, json=data)
response.raise_for_status()

analysis = response.json()
```

The word bounding boxes and text from the results of analysis can be extracted using the following lines of code:

```
line_infos = [region["lines"] for region in analysis["regions"]]
word_infos = []
for line in line_infos:
    for word_metadata in line:
        for word_info in word_metadata["words"]:
            word_infos.append(word_info)
word_infos
```

```
[{'boundingBox': '28,16,288,41', 'text': 'NOTHING'},
{'boundingBox': '27,66,283,52', 'text': 'EXISTS'},
{'boundingBox': '27,128,292,49', 'text': 'EXCEPT'},
{'boundingBox': '24,188,292,54', 'text': 'ATOMS'},
{'boundingBox': '22,253,105,32', 'text': 'AND'},
{'boundingBox': '144,253,175,32', 'text': 'EMPTY'},
{'boundingBox': '21,298,304,60', 'text': 'SPACE.'},
{'boundingBox': '26,387,210,37', 'text': 'Everything'},
{'boundingBox': '249,389,71,27', 'text': 'else'},
{'boundingBox': '127,431,31,29', 'text': 'is'},
{'boundingBox': '172,431,153,36', 'text': 'opinion.'}]
```

Finally, the recognized text can be overlaid on top of the original image using the `matplotlib` library.

```
plt.figure(figsize=(5,5))

image = Image.open(BytesIO(requests.get(image_url).content))
ax = plt.imshow(image, alpha=0.5)
for word in word_infos:
    bbox = [int(num) for num in word["boundingBox"].split(",")]
    text = word["text"]
    origin = (bbox[0], bbox[1])
    patch = Rectangle(origin, bbox[2], bbox[3], fill=False, linewidth=2, color='y')
    axes.add_patch(patch)
    plt.text(origin[0], origin[1], text, fontsize=20, weight="bold", va="top")
_ = plt.axis("off")
```

## Text recognition with Computer Vision API

Use the [RecognizeText method](#) to detect handwritten or printed text in an image and extract recognized characters into a machine-usable character stream.

Set `image_url` to point to the image to be recognized.

```
image_url = "https://upload.wikimedia.org/wikipedia/commons/thumb/d/dd/Cursive_Writing_on_Notebook_paper.jpg/800px-Cursive_Writing_on_Notebook_paper.jpg"
```

The service end point for the text recognition service can be constructed as follows:

```
text_recognition_url = vision_base_url + "RecognizeText"
print(text_recognition_url)
```

```
https://westcentralus.api.cognitive.microsoft.com/vision/v1.0/RecognizeText
```

The handwritten text recognition service can be used to recognize the text in the image. In the `params` dictionary, set `handwriting` to `false` to recognize only printed text.

```
headers = {'Ocp-Apim-Subscription-Key': subscription_key}
params = {'handwriting': True}
data = {'url': image_url}
response = requests.post(text_recognition_url, headers=headers, params=params, json=data)
response.raise_for_status()
```

The text recognition service does not return the recognized text by itself. Instead, it returns immediately with an "Operation Location" URL in the response header that must be polled to get the result of the operation.



```
operation_url = response.headers["Operation-Location"]
```

After obtaining the `operation_url`, you can query it for the analyzed text. The following lines of code implement a polling loop in order to wait for the operation to complete. Notice that the polling is done via an HTTP `GET` method instead of `POST`.

```
import time

analysis = {}
while not "recognitionResult" in analysis:
    response_final = requests.get(response.headers["Operation-Location"], headers=headers)
    analysis = response_final.json()
    time.sleep(1)
```

Next, the recognized text along with the bounding boxes can be extracted as shown in the following line of code. An important point to note is that the handwritten text recognition API returns bounding boxes as **polygons** instead of **rectangles**. Each polygon  $p$  is defined by its vertices specified using the following convention:

$$p = [x_1, y_1, x_2, y_2, \dots, x_N, y_N]$$

```
polygons = [(line["boundingBox"], line["text"]) for line in analysis["recognitionResult"]["lines"]]
```

Finally, the recognized text can be overlaid on top of the original image using the extracted polygon information. Notice that `matplotlib` requires the vertices to be specified as a list of tuples of the form:

$$p = [(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)]$$

and the post-processing code transforms the polygon data returned by the service into the form required by `matplotlib`.

```
from matplotlib.patches import Polygon

plt.figure(figsize=(15,15))

image = Image.open(BytesIO(requests.get(image_url).content))
ax = plt.imshow(image)
for polygon in polygons:
    vertices = [(polygon[0][i], polygon[0][i+1]) for i in range(0, len(polygon[0]), 2)]
    text = polygon[1]
    patch = Polygon(vertices, closed=True, fill=False, linewidth=2, color='y')
    axes.add_patch(patch)
    plt.text(vertices[0][0], vertices[0][1], text, fontsize=20, va="top")
_ = plt.axis("off")
```

## Analyze an image stored on disk

The Computer Vision REST APIs don't just accept URLs to publically accessible images. They can also be provided the image to be analyzed as part of the HTTP body. For more details of this feature, see the documentation [here](#).

The code in this section uses this feature to analyze a sample image on disk. The primary difference between passing in an image URL vs. image data is that the header to the request must contain an entry of the form:

```
{"Content-Type": "application/octet-stream"}
```

and the binary image data must be passed in via the `data` parameter to `requests.post` as opposed to the `json`

parameter.

First, download a sample image from the [Computer Vision API](#) page to the local file system and make `image_path` point to it.

```
%%bash
mkdir -p images
curl -Ls https://aka.ms/csnb-house-yard -o images/house_yard.jpg
```

```
image_path = "images/house_yard.jpg"
```

Then, read it into a byte array and send it to the Vision service to be analyzed.

```
image_data = open(image_path, "rb").read()
headers = {'Ocp-Apim-Subscription-Key': subscription_key,
           "Content-Type": "application/octet-stream" }
params = {'visualFeatures': 'Categories,Description,Color'}
response = requests.post(vision_analyze_url,
                         headers=headers,
                         params=params,
                         data=image_data)

response.raise_for_status()

analysis = response.json()
image_caption = analysis["description"][0]["captions"][0]["text"].capitalize()
image_caption
```

```
'A large lawn in front of a house'
```

As before, the caption can be easily overlaid on the image. Notice that since the image is already available locally, the process is slightly shorter.

```
image = Image.open(image_path)
plt.imshow(image)
plt.axis("off")
_ = plt.title(image_caption, size="x-large", y=-0.1)
```

# Computer Vision Ruby Quick Starts

6/27/2017 • 4 min to read • [Edit Online](#)

This article provides information and code samples to help you quickly get started using the Computer Vision API with Ruby to accomplish the following tasks:

- [Analyze an image](#)
- [Intelligently generate a thumbnail](#)
- [Detect and extract text from an Image](#)

Learn more about obtaining free Subscription Keys [here](#)

## Analyze an Image With Computer Vision API Using Ruby

With the [Analyze Image method](#), you can extract visual features based on image content. You can upload an image or specify an image URL and choose which features to return, including:

- The category defined in this [taxonomy](#).
- A detailed list of tags related to the image content.
- A description of image content in a complete sentence.
- The coordinates, gender, and age of any faces contained in the image.
- The ImageType (clipart or a line drawing)
- The dominant color, the accent color, or whether an image is black & white.
- Whether the image contains pornographic or sexually suggestive content.

### Analyze an Image Ruby Example Request

Change the REST URL to use the location where you obtained your subscription keys, replace the "Ocp-Apim-Subscription-Key" value with your valid subscription key, and add a URL to a photograph of a celebrity to the `body` variable.

```

require 'net/http'

# NOTE: You must use the same location in your REST call as you used to obtain your subscription keys.
# For example, if you obtained your subscription keys from westus, replace "westcentralus" in the
# URL below with "westus".
uri = URI('https://westcentralus.api.cognitive.microsoft.com/vision/v1.0/analyze')
uri.query = URI.encode_www_form({
  # Request parameters
  'visualFeatures' => 'Categories',
  'details' => '{string}',
  'language' => 'en'
})

request = Net::HTTP::Post.new(uri.request_uri)
# Request headers
request['Content-Type'] = 'application/json'
# NOTE: Replace the "Ocp-Apim-Subscription-Key" value with a valid subscription key.
request['Ocp-Apim-Subscription-Key'] = '{subscription key}'
# Replace with the body, for example, "{\"url\": \"http://www.example.com/images/image.jpg\"}"
request.body = "{body}"

response = Net::HTTP.start(uri.host, uri.port, :use_ssl => uri.scheme == 'https') do |http|
  http.request(request)
end

puts response.body

```

## Analyze an Image Response

A successful response is returned in JSON. Following is an example of a successful response:

```

{
  "categories": [
    {
      "name": "abstract_",
      "score": 0.00390625
    },
    {
      "name": "people_",
      "score": 0.83984375,
      "detail": {
        "celebrities": [
          {
            "name": "Satya Nadella",
            "faceRectangle": {
              "left": 597,
              "top": 162,
              "width": 248,
              "height": 248
            }
          },
          "confidence": 0.999028444
        ]
      }
    }
  ],
  "adult": {
    "isAdultContent": false,
    "isRacyContent": false,
    "adultScore": 0.0934349000453949,
    "racyScore": 0.068613491952419281
  },
  "tags": [
    {
      "name": "person",
      "confidence": 0.98979085683822632
    },
    {
      "name": "satya nadella",
      "confidence": 0.98979085683822632
    }
  ]
}

```

```

    },
    {
      "name": "man",
      "confidence": 0.94493889808654785
    },
    {
      "name": "outdoor",
      "confidence": 0.938492476940155
    },
    {
      "name": "window",
      "confidence": 0.89513939619064331
    }
  ],
  "description": {
    "tags": [
      "person",
      "man",
      "outdoor",
      "window",
      "glasses"
    ],
    "captions": [
      {
        "text": "Satya Nadella sitting on a bench",
        "confidence": 0.48293603002174407
      }
    ]
  },
  "requestId": "0dbec5ad-a3d3-4f7e-96b4-dfd57efe967d",
  "metadata": {
    "width": 1500,
    "height": 1000,
    "format": "Jpeg"
  },
  "faces": [
    {
      "age": 44,
      "gender": "Male",
      "faceRectangle": {
        "left": 593,
        "top": 160,
        "width": 250,
        "height": 250
      }
    }
  ],
  "color": {
    "dominantColorForeground": "Brown",
    "dominantColorBackground": "Brown",
    "dominantColors": [
      "Brown",
      "Black"
    ],
    "accentColor": "873B59",
    "isBWMng": false
  },
  "imageType": {
    "clipArtType": 0,
    "lineDrawingType": 0
  }
}

```

## Get a Thumbnail with Computer Vision API Using Ruby

Use the [Get Thumbnail method](#) to crop an image based on its region of interest (ROI) to the height and width you

desire, even if the aspect ratio differs from the input image.

### Get a Thumbnail Ruby Example Request

Change the REST URL to use the location where you obtained your subscription keys, replace the "Ocp-Apim-Subscription-Key" value with your valid subscription key, and add a URL to a photograph of a celebrity to the `body` variable.

```
require 'net/http'

# NOTE: You must use the same location in your REST call as you used to obtain your subscription keys.
# For example, if you obtained your subscription keys from westus, replace "westcentralus" in the
# URL below with "westus".
uri = URI("https://westcentralus.api.cognitive.microsoft.com/vision/v1.0/generateThumbnail")
uri.query = URI.encode_www_form({
  # Request parameters
  'width' => '{number}',
  'height' => '{number}',
  'smartCropping' => 'true'
})

request = Net::HTTP::Post.new(uri.request_uri)
# Request headers
request['Content-Type'] = 'application/json'
# NOTE: Replace the "Ocp-Apim-Subscription-Key" value with a valid subscription key.
request['Ocp-Apim-Subscription-Key'] = '{subscription key}'
# Replace with the body, for example, "{ \"url\": \"http://www.example.com/images/image.jpg\" }"
request.body = "{body}"

response = Net::HTTP.start(uri.host, uri.port, :use_ssl => uri.scheme == 'https') do |http|
  http.request(request)
end

puts response.body
```

### Get a Thumbnail Response

A successful response contains the thumbnail image binary. If the request failed, the response contains an error code and a message to help determine what went wrong.

## Optical Character Recognition (OCR) with Computer Vision API Using Ruby

Use the [Optical Character Recognition \(OCR\) method](#) to detect text in an image and extract recognized characters into a machine-usable character stream.

### OCR Ruby Example Request

Change the REST URL to use the location where you obtained your subscription keys, replace the "Ocp-Apim-Subscription-Key" value with your valid subscription key, and add a URL to a photograph of a celebrity to the `body` variable.

```

require 'net/http'

# NOTE: You must use the same location in your REST call as you used to obtain your subscription keys.
# For example, if you obtained your subscription keys from westus, replace "westcentralus" in the
# URL below with "westus".
uri = URI('https://westcentralus.api.cognitive.microsoft.com/vision/v1.0/ocr')
uri.query = URI.encode_www_form({
  # Request parameters
  'language' => 'unk',
  'detectOrientation' => 'true'
})

request = Net::HTTP::Post.new(uri.request_uri)
# Request headers
request['Content-Type'] = 'application/json'
# NOTE: Replace the "Ocp-Apim-Subscription-Key" value with a valid subscription key.
request['Ocp-Apim-Subscription-Key'] = '{subscription key}'
# Replace with the body, for example, '{"url": "http://www.example.com/images/image.jpg"}'
request.body = '{"body}'

response = Net::HTTP.start(uri.host, uri.port, :use_ssl => uri.scheme == 'https') do |http|
  http.request(request)
end

puts response.body

```

## OCR Example Response

Upon success, the OCR results returned include text, bounding box for regions, lines, and words.

```
{
  "language": "en",
  "textAngle": -2.0000000000000338,
  "orientation": "Up",
  "regions": [
    {
      "boundingBox": "462,379,497,258",
      "lines": [
        {
          "boundingBox": "462,379,497,74",
          "words": [
            {
              "boundingBox": "462,379,41,73",
              "text": "A"
            },
            {
              "boundingBox": "523,379,153,73",
              "text": "GOAL"
            },
            {
              "boundingBox": "694,379,265,74",
              "text": "WITHOUT"
            }
          ]
        },
        {
          "boundingBox": "565,471,289,74",
          "words": [
            {
              "boundingBox": "565,471,41,73",
              "text": "A"
            },
            {
              "boundingBox": "626,471,150,73",
              "text": "PLAN"
            },
            {
              "boundingBox": "801,472,53,73",
              "text": "IS"
            }
          ]
        },
        {
          "boundingBox": "519,563,375,74",
          "words": [
            {
              "boundingBox": "519,563,149,74",
              "text": "JUST"
            },
            {
              "boundingBox": "683,564,41,72",
              "text": "A"
            },
            {
              "boundingBox": "741,564,153,73",
              "text": "WISH"
            }
          ]
        }
      ]
    }
  ]
}
```



# Computer Vision API C# Tutorial

11/30/2017 • 4 min to read • [Edit Online](#)

Explore a basic Windows application that uses Computer Vision API to perform optical character recognition (OCR), create smart-cropped thumbnails, plus detect, categorize, tag and describe visual features, including faces, in an image. The below example lets you submit an image URL or a locally stored file. You can use this open source example as a template for building your own app for Windows using the Vision API and WPF (Windows Presentation Foundation), a part of .NET Framework.

## Prerequisites

### Platform requirements

The below example has been developed for the .NET Framework using [Visual Studio 2015, Community Edition](#).

### Subscribe to Computer Vision API and get a subscription key

Before creating the example, you must subscribe to Computer Vision API which is part of the Microsoft Cognitive Services (formerly Project Oxford). For subscription and key management details, see [Subscriptions](#). Both the primary and secondary key can be used in this tutorial.

#### NOTE

The tutorial is designed to use subscription keys in the **westcentralus** region. The subscription keys generated in the Computer Vision free trial use the **westcentralus** region, so they work correctly. If you generated your subscription keys using your Azure account through <https://azure.microsoft.com/>, you must specify the **westcentralus** region. Keys generated outside the **westcentralus** region will not work.

### Get the client library and example

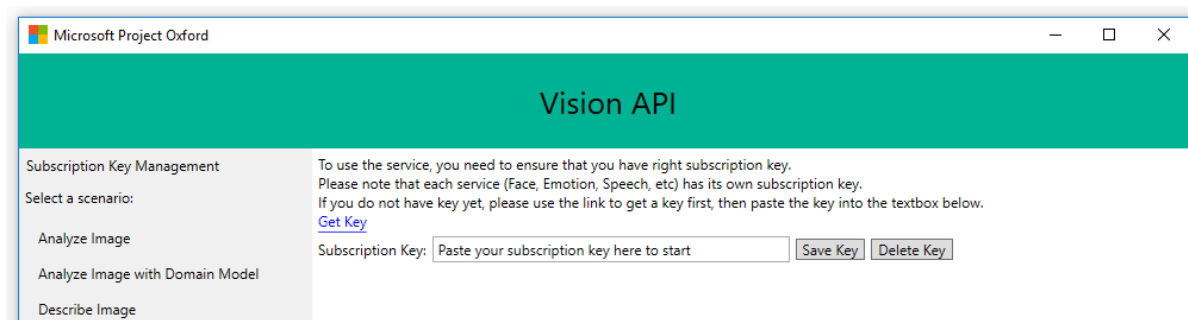
You may clone the Computer Vision API client library and example application to your computer via [SDK](#). Don't download it as a ZIP.

In your GitHub Desktop, open Sample-WPF\VisionAPI-WPF-Samples.sln.

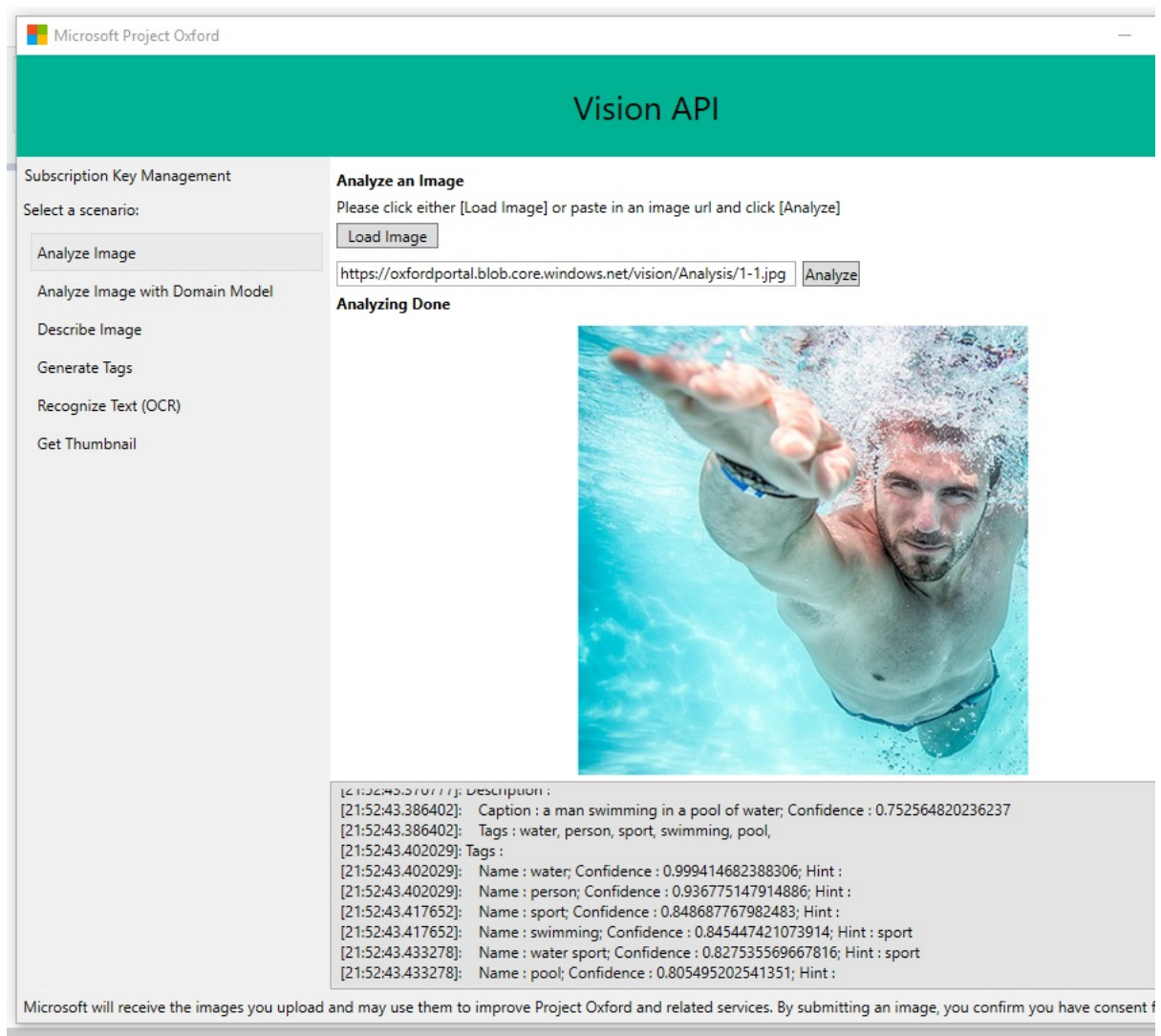
1. Press Ctrl+Shift+B, or click Build on the ribbon menu, then select Build Solution.

2. After the build is complete, press **F5** or click **Start** on the ribbon menu to run the example.

3. Locate the Computer Vision API user interface window with the text edit box reading "Paste your subscription key here to start". You can choose to persist your subscription key on your PC or laptop by clicking the "Save Key" button. When you want to delete the subscription key from the system, click "Delete Key" to remove it from your PC or laptop.



4. Under "Select Scenario" click to use one of the six scenarios, then follow the instructions on the screen. Microsoft receives the images you upload and may use them to improve Computer Vision API and related services. By submitting an image, you confirm that you have followed our [Developer Code of Conduct](#).



4. There are example images to be used with this example application. You can find these images on the Face API Windows Github repo, in the [Data folder](#). Please note the use of these images is licensed under agreement [LICENSE-IMAGE](#).

Now that you have a running application, let us review how this example app integrates with Cognitive Services technology. This will make it easier to either continue building onto this app or develop your own app using Microsoft Computer Vision API.

This example app makes use of the Computer Vision API Client Library, a thin C# client wrapper for the Microsoft Computer Vision API. When you built the example app as described above, you got the Client Library from a NuGet package. You can review the Client Library source code in the folder titled "**Client Library**" under **Vision, Windows, Client Library**, which is part of the downloaded file repository mentioned above in Prerequisites.

You can also find out how to use the Client Library code in Solution Explorer: Under **VisionAPI-WPF\_Samples**, expand **AnalyzePage.xaml** to locate **AnalyzePage.xaml.cs**, which is used for submitting an image to the image analysis endpoint. Double-click the .xaml.cs files to have them open in new windows in Visual Studio.

Reviewing how the Vision Client Library gets used in our example app, let's look at two code snippets from **AnalyzePage.xaml.cs**. The file contains code comments indicating "KEY SAMPLE CODE STARTS HERE" and "KEY SAMPLE CODE ENDS HERE" to help you locate the code snippets reproduced below.

The analyze endpoint is able to work with either an image URL or binary image data (in form of an octet stream) as input. First, you find a using directive, which lets you use the Vision Client Library.

```
// -----
// KEY SAMPLE CODE STARTS HERE
// Use the following namespace for VisionServiceClient
// -----
using Microsoft.ProjectOxford.Vision;
using Microsoft.ProjectOxford.Vision.Contract;
// -----
// KEY SAMPLE CODE ENDS HERE
// -----
```

**UploadAndAnalyzeImage(...)** This code snippet shows how to use the Client Library to submit your subscription key and a locally stored image to the analyze endpoint of the Computer Vision API service.

```
private async Task<AnalysisResult> UploadAndAnalyzeImage(string imagePath)
{
    // -----
    // KEY SAMPLE CODE STARTS HERE
    // -----
    //
    // Create Project Oxford Computer Vision API Service client
    //
    VisionServiceClient visionServiceClient = new VisionServiceClient(SubscriptionKey);
    Log("VisionServiceClient is created");

    using (Stream imageFileStream = File.OpenRead(imageFilePath))
    {
        //
        // Analyze the image for all visual features
        //
        Log("Calling VisionServiceClient.AnalyzeImageAsync(...)");
        VisualFeature[] visualFeatures = new VisualFeature[] { VisualFeature.Adult, VisualFeature.Categories, VisualFeature.Color,
        VisualFeature.Description, VisualFeature.Faces, VisualFeature.ImageType, VisualFeature.Tags };
        AnalysisResult analysisResult = await visionServiceClient.AnalyzeImageAsync(imageFileStream, visualFeatures);
        return analysisResult;
    }

    // -----
    // KEY SAMPLE CODE ENDS HERE
    // -----
}
```

**AnalyzeUrl(...)** This code snippet shows how to use the Client Library to submit your subscription key and a photo URL to the analyze endpoint of the Computer Vision API service.

```

private async Task<AnalysisResult> AnalyzeUrl(string imageUrl)
{
    // -----
    // KEY SAMPLE CODE STARTS HERE
    // -----

    //
    // Create Project Oxford Computer Vision API Service client
    //
    VisionServiceClient visionServiceClient = new VisionServiceClient(SubscriptionKey);
    Log("VisionServiceClient is created");

    //
    // Analyze the url for all visual features
    //
    Log("Calling VisionServiceClient.AnalyzeImageAsync(...)");
    VisualFeature[] visualFeatures = new VisualFeature[] { VisualFeature.Adult, VisualFeature.Categories, VisualFeature.Color,
    VisualFeature.Description, VisualFeature.Faces, VisualFeature.ImageType, VisualFeature.Tags };
    AnalysisResult analysisResult = await visionServiceClient.AnalyzeImageAsync(imageUrl, visualFeatures);
    return analysisResult;
}
// -----
// KEY SAMPLE CODE ENDS HERE
// -----

```

**Other pages and endpoints** How to interact with the other endpoints exposed by the Computer Vision API service can be seen by looking at the other pages in the sample; for instance, the OCR endpoint is shown as part of the code contained in OCRPage.xaml.cs

• [Get started with Face API](#)

# Computer Vision API Java tutorial

9/25/2017 • 22 min to read • [Edit Online](#)

This tutorial shows the features of the Microsoft Cognitive Services Computer Vision REST API.

Explore a Java Swing application that uses the Computer Vision REST API to perform optical character recognition (OCR), create smart-cropped thumbnails, plus detect, categorize, tag, and describe visual features, including faces, in an image. This example lets you submit an image URL for analysis or processing. You can use this open source example as a template for building your own app in Java to use the Computer Vision REST API.

This tutorial will cover how to use Computer Vision to:

- Analyze an image
- Identify a natural or artificial landmark in an image
- Identify a celebrity in an image
- Create a quality thumbnail from an image
- Read printed text in an image
- Read handwritten text in an image

The Java Swing form application has already been written, but has no functionality. In this tutorial, you add the code specific to the Computer Vision REST API to complete the application's functionality.

## Prerequisites

### Platform requirements

This tutorial has been developed using the NetBeans IDE. Specifically, the **Java SE** version of NetBeans, which you can [download here](#).

### Subscribe to Computer Vision API and get a subscription key

Before creating the example, you must subscribe to Computer Vision API which is part of the Microsoft Cognitive Services. For subscription and key management details, see [Subscriptions](#). Both the primary and secondary keys are valid to use in this tutorial.

## Download the tutorial project

1. Go to the [Cognitive Services Java Computer Vision Tutorial](#) repository.
2. Click the **Clone or download** button.
3. Click **Download ZIP** to download a .zip file of the tutorial project.

There is no need to extract the contents of the .zip file because NetBeans imports the project from the .zip file.

## Import the tutorial project

Import the **cognitive-services-java-computer-vision-tutorial-master.zip** file into NetBeans.

1. In NetBeans, click **File > Import Project > From ZIP...**. The **Import Project(s) from ZIP** dialog box appears.
2. In the **ZIP File:** field, click the **Browse** button to locate the **cognitive-services-java-computer-vision-tutorial-master.zip** file, then click **Open**.
3. Click **Import** from the **Import Project(s) from ZIP** dialog box.
4. In the **Projects** panel, expand **ComputerVision > Source Packages > <default package>**. Some versions of

NetBeans use **src** instead of **Source Packages** > **<default package>**. In that case, expand **src**.

5. Double-click **MainFrame.java** to load the file into the NetBeans editor. The **Design** tab of the **MainFrame.java** file appears.
6. Click the **Source** tab to view the Java source code.

## Build and run the tutorial project

1. Press **F6** to build and run the tutorial application.

In the tutorial application, click a tab to bring up the pane for that feature. The buttons have empty methods, so they do nothing.

At the bottom of the window are the fields **Subscription Key** and **Subscription Region**. These fields must be filled with a valid subscription key and the correct region for that subscription key. To obtain a subscription key, see [Subscriptions](#). If you obtained your subscription key from the free trial at that link, then the default **westcentralus** is the correct region for your subscription keys.

2. Exit the tutorial application.

## Add the tutorial code

The Java Swing application is set up with six tabs. Each tab demonstrates a different function of Computer Vision (analyze, OCR, etc). The six tutorial sections do not have interdependencies, so you can add one section, all six sections, or only a section or two. And you can add the sections in any order.

Let's get started.

## Analyze an image

The Analyze feature of Computer Vision analyzes an image for more than 2,000 recognizable objects, living beings, scenery, and actions. Once the analysis is complete, Analyze returns a JSON object that describes the image with descriptive tags, color analysis, captions, and more.

To complete the Analyze feature of the tutorial application, perform the following steps:

### Analyze step 1: Add the event handler code for the form button

The **analyzeImageButtonActionPerformed** event handler method clears the form, displays the image specified in the URL, then calls the **AnalyzeImage** method to analyze the image. When **AnalyzeImage** returns, the method displays the formatted JSON response in the **Response** text area, extracts the first caption from the **JSONObject**, and displays the caption and the confidence level that the caption is correct.

Copy and paste the following code into the **analyzeImageButtonActionPerformed** method.

#### NOTE

NetBeans won't let you paste to the method definition line ( `private void` ) or to the closing curly brace of that method. To copy the code, copy the lines between the method definition and the closing curly brace, and paste them over the contents of the method.

```

private void analyzeImageButtonActionPerformed(java.awt.event.ActionEvent evt) {
    URL analyzeImageUrl;

    // Clear out the previous image, response, and caption, if any.
    analyzeImage.setIcon(new ImageIcon());
    analyzeCaptionLabel.setText("");
    analyzeResponseTextArea.setText("");

    // Display the image specified in the text box.
    try {
        analyzeImageUrl = new URL(analyzeImageUriTextBox.getText());
        BufferedImage bImage = ImageIO.read(analyzeImageUrl);
        scaleAndShowImage(bImage, analyzeImage);
    } catch (IOException e) {
        analyzeResponseTextArea.setText("Error loading Analyze image: " + e.getMessage());
        return;
    }

    // Analyze the image.
    JSONObject jsonObj = AnalyzeImage(analyzeImageUrl.toString());

    // A return of null indicates failure.
    if (jsonObj == null) {
        return;
    }

    // Format and display the JSON response.
    analyzeResponseTextArea.setText(jsonObj.toString(2));

    // Extract the text and confidence from the first caption in the description object.
    if (jsonObj.has("description") && jsonObj.getJSONObject("description").has("captions")) {

        JSONObject jsonCaption = jsonObj.getJSONObject("description").getJSONArray("captions").getJSONObject(0);

        if (jsonCaption.has("text") && jsonCaption.has("confidence")) {

            analyzeCaptionLabel.setText("Caption: " + jsonCaption.getString("text") +
                " (confidence: " + jsonCaption.getDouble("confidence") + ").");
        }
    }
}

```

## Analyze step 2: Add the wrapper for the REST API call

The **AnalyzeImage** method wraps the REST API call to analyze an image. The method returns a **JSONObject** describing the image, or **null** if there was an error.

Copy and paste the **AnalyzeImage** method to just underneath the **analyzeImageButtonActionPerformed** method.

```

/**
 * Encapsulates the Microsoft Cognitive Services REST API call to analyze an image.
 * @param imageUrl: The string URL of the image to analyze.
 * @return: A JSONObject describing the image, or null if a runtime error occurs.
 */
private JSONObject AnalyzeImage(String imageUrl) {
    try (CloseableHttpClient httpclient = HttpClientBuilder.create().build())
    {
        // Create the URI to access the REST API call for Analyze Image.
        String uriString = uriBasePreRegion +
            String.valueOf(subscriptionRegionComboBox.getSelectedItem()) +
            uriBasePostRegion + uriBaseAnalyze;
        URIBuilder builder = new URIBuilder(uriString);

        // Request parameters. All of them are optional.
        builder.setParameter("visualFeatures", "Categories,Description,Color,Adult");
        builder.setParameter("language", "en");

        // Prepare the URI for the REST API call.
        URI uri = builder.build();
        HttpPost request = new HttpPost(uri);

        // Request headers.
        request.setHeader("Content-Type", "application/json");
        request.setHeader("Ocp-Apim-Subscription-Key", subscriptionKeyTextField.getText());

        // Request body.
        StringEntity reqEntity = new StringEntity("{\"url\":\"" + imageUrl + "\"}");
        request.setEntity(reqEntity);

        // Execute the REST API call and get the response entity.
        HttpResponse response = httpclient.execute(request);
        HttpEntity entity = response.getEntity();

        // If we got a response, parse it and display it.
        if (entity != null)
        {
            // Return the JSONObject.
            String jsonString = EntityUtils.toString(entity);
            return new JSONObject(jsonString);
        } else {
            // No response. Return null.
            return null;
        }
    }
    catch (Exception e)
    {
        // Display error message.
        System.out.println(e.getMessage());
        return null;
    }
}

```

### Analyze step 3: Run the application

Press **F6** to run the application. Put your subscription key into the **Subscription Key** field and verify that you are using the correct region in **Subscription Region**. Enter a URL to an image to analyze, then click the **Analyze Image** button to analyze an image and see the result.

## Recognize a landmark

The Landmark feature of Computer Vision analyzes an image for natural and artificial landmarks, such as mountains or famous buildings. Once the analysis is complete, Landmark returns a JSON object that identifies the landmarks found in the image.



To complete the Landmark feature of the tutorial application, perform the following steps:

### Landmark step 1: Add the event handler code for the form button

The **landmarkImageButtonActionPerformed** event handler method clears the form, displays the image specified in the URL, then calls the **LandmarkImage** method to analyze the image. When **LandmarkImage** returns, the method displays the formatted JSON response in the **Response** text area, then extracts the first landmark name from the **JSONObject** and displays it on the window along with the confidence level that the landmark was identified correctly.

Copy and paste the following code into the **landmarkImageButtonActionPerformed** method.

#### NOTE

NetBeans won't let you paste to the method definition line ( `private void` ) or to the closing curly brace of that method. To copy the code, copy the lines between the method definition and the closing curly brace, and paste them over the contents of the method.

```
private void landmarkImageButtonActionPerformed(java.awt.event.ActionEvent evt) {
    URL landmarkImageUrl;

    // Clear out the previous image, response, and caption, if any.
    landmarkImage.setIcon(new ImageIcon());
    landmarkCaptionLabel.setText("");
    landmarkResponseTextArea.setText("");

    // Display the image specified in the text box
    try {
        landmarkImageUrl = new URL(landmarkImageUriTextBox.getText());
        BufferedImage bImage = ImageIO.read(landmarkImageUrl);
        scaleAndShowImage(bImage, landmarkImage);
    } catch (IOException e) {
        landmarkResponseTextArea.setText("Error loading Landmark image: " + e.getMessage());
        return;
    }

    // Identify the landmark in the image.
    JSONObject jsonObj = LandmarkImage(landmarkImageUrl.toString());

    // A return of null indicates failure.
    if (jsonObj == null) {
        return;
    }

    // Format and display the JSON response.
    landmarkResponseTextArea.setText(jsonObj.toString(2));

    // Extract the text and confidence from the first caption in the description object.
    if (jsonObj.has("result") && jsonObj.getJSONObject("result").has("landmarks")) {
        JSONObject jsonCaption = jsonObj.getJSONObject("result").getJSONArray("landmarks").getJSONObject(0);

        if (jsonCaption.has("name") && jsonCaption.has("confidence")) {
            landmarkCaptionLabel.setText("Caption: " + jsonCaption.getString("name") +
                " (confidence: " + jsonCaption.getDouble("confidence") + ").");
        }
    }
}
```

### Landmark step 2: Add the wrapper for the REST API call

The **LandmarkImage** method wraps the REST API call to analyze an image. The method returns a **JSONObject**

describing the landmarks found in the image, or **null** if there was an error.

Copy and paste the **LandmarkImage** method to just underneath the **landmarkImageButtonActionPerformed** method.

```
/**
 * Encapsulates the Microsoft Cognitive Services REST API call to identify a landmark in an image.
 * @param imageUrl: The string URL of the image to process.
 * @return: A JSONObject describing the image, or null if a runtime error occurs.
 */
private JSONObject LandmarkImage(String imageUrl) {
    try (CloseableHttpClient httpclient = HttpClientBuilder.create().build())
    {
        // Create the URI to access the REST API call to identify a Landmark in an image.
        String uriString = uriBasePreRegion +
            String.valueOf(subscriptionRegionComboBox.getSelectedItem()) +
            uriBasePostRegion + uriBaseLandmark;
        URIBuilder builder = new URIBuilder(uriString);

        // Request parameters. All of them are optional.
        builder.setParameter("visualFeatures", "Categories,Description,Color");
        builder.setParameter("language", "en");

        // Prepare the URI for the REST API call.
        URI uri = builder.build();
        HttpPost request = new HttpPost(uri);

        // Request headers.
        request.setHeader("Content-Type", "application/json");
        request.setHeader("Ocp-Apim-Subscription-Key", subscriptionKeyTextField.getText());

        // Request body.
        StringEntity reqEntity = new StringEntity("{\"url\":\"" + imageUrl + "\"}");
        request.setEntity(reqEntity);

        // Execute the REST API call and get the response entity.
        HttpResponse response = httpclient.execute(request);
        HttpEntity entity = response.getEntity();

        // If we got a response, parse it and display it.
        if (entity != null)
        {
            // Return the JSONObject.
            String jsonString = EntityUtils.toString(entity);
            return new JSONObject(jsonString);
        } else {
            // No response. Return null.
            return null;
        }
    }
    catch (Exception e)
    {
        // Display error message.
        System.out.println(e.getMessage());
        return null;
    }
}
```

### Landmark step 3: Run the application

Press **F6** to run the application. Put your subscription key into the **Subscription Key** field and verify that you are using the correct region in **Subscription Region**. Click the **Landmark** tab, enter a URL to an image of a landmark, then click the **Analyze Image** button to analyze an image and see the result.

## Recognize celebrities

The Celebrities feature of Computer Vision analyzes an image for famous people. Once the analysis is complete, Celebrities returns a JSON object that identifies the Celebrities found in the image.

To complete the Celebrities feature of the tutorial application, perform the following steps:

### Celebrities step 1: Add the event handler code for the form button

The **celebritiesImageButtonActionPerformed** event handler method clears the form, displays the image specified in the URL, then calls the **CelebritiesImage** method to analyze the image. When **CelebritiesImage** returns, the method displays the formatted JSON response in the **Response** text area, then extracts the first celebrity name from the **JSONObject** and displays the name on the window along with the confidence level that the celebrity was identified correctly.

Copy and paste the following code into the **celebritiesImageButtonActionPerformed** method.

#### NOTE

NetBeans won't let you paste to the method definition line ( `private void` ) or to the closing curly brace of that method. To copy the code, copy the lines between the method definition and the closing curly brace, and paste them over the contents of the method.

```
private void celebritiesImageButtonActionPerformed(java.awt.event.ActionEvent evt) {
    URL celebritiesImageUrl;

    // Clear out the previous image, response, and caption, if any.
    celebritiesImage.setIcon(new ImageIcon());
    celebritiesCaptionLabel.setText("");
    celebritiesResponseTextArea.setText("");

    // Display the image specified in the text box.
    try {
        celebritiesImageUrl = new URL(celebritiesImageUriTextBox.getText());
        BufferedImage bImage = ImageIO.read(celebritiesImageUrl);
        scaleAndShowImage(bImage, celebritiesImage);
    } catch (IOException e) {
        celebritiesResponseTextArea.setText("Error loading Celebrity image: " + e.getMessage());
        return;
    }

    // Identify the celebrities in the image.
    JSONObject jsonObj = CelebritiesImage(celebritiesImageUrl.toString());

    // A return of null indicates failure.
    if (jsonObj == null) {
        return;
    }

    // Format and display the JSON response.
    celebritiesResponseTextArea.setText(jsonObj.toString(2));

    // Extract the text and confidence from the first caption in the description object.
    if (jsonObj.has("result") && jsonObj.getJSONObject("result").has("celebrities")) {

        JSONObject jsonCaption = jsonObj.getJSONObject("result").getJSONArray("celebrities").getJSONObject(0);

        if (jsonCaption.has("name") && jsonCaption.has("confidence")) {

            celebritiesCaptionLabel.setText("Caption: " + jsonCaption.getString("name") +
                " (confidence: " + jsonCaption.getDouble("confidence") + ").");
        }
    }
}
```

## Celebrities step 2: Add the wrapper for the REST API call

The **CelebritiesImage** method wraps the REST API call to analyze an image. The method returns a **JSONObject** describing the celebrities found in the image, or **null** if there was an error.

Copy and paste the **CelebritiesImage** method to just underneath the **celebritiesImageButtonActionPerformed** method.

```
/**
 * Encapsulates the Microsoft Cognitive Services REST API call to identify celebrities in an image.
 * @param imageUrl: The string URL of the image to process.
 * @return: A JSONObject describing the image, or null if a runtime error occurs.
 */
private JSONObject CelebritiesImage(String imageUrl) {
    try (CloseableHttpClient httpClient = HttpClientBuilder.create().build())
    {
        // Create the URI to access the REST API call to identify celebrities in an image.
        String uriString = uriBasePreRegion +
            String.valueOf(subscriptionRegionComboBox.getSelectedItem()) +
            uriBasePostRegion + uriBaseCelebrities;
        URIBuilder builder = new URIBuilder(uriString);

        // Request parameters. All of them are optional.
        builder.setParameter("visualFeatures", "Categories,Description,Color");
        builder.setParameter("language", "en");

        // Prepare the URI for the REST API call.
        URI uri = builder.build();
        HttpPost request = new HttpPost(uri);

        // Request headers.
        request.setHeader("Content-Type", "application/json");
        request.setHeader("Ocp-Apim-Subscription-Key", subscriptionKeyTextField.getText());

        // Request body.
        StringEntity reqEntity = new StringEntity("{\"url\":\"" + imageUrl + "\"}");
        request.setEntity(reqEntity);

        // Execute the REST API call and get the response entity.
        HttpResponse response = httpClient.execute(request);
        HttpEntity entity = response.getEntity();

        // If we got a response, parse it and display it.
        if (entity != null)
        {
            // Return the JSONObject.
            String jsonString = EntityUtils.toString(entity);
            return new JSONObject(jsonString);
        } else {
            // No response. Return null.
            return null;
        }
    }
    catch (Exception e)
    {
        // Display error message.
        System.out.println(e.getMessage());
        return null;
    }
}
```

## Celebrities step 3: Run the application

Press **F6** to run the application. Put your subscription key into the **Subscription Key** field and verify that you are using the correct region in **Subscription Region**. Click the **Celebrities** tab, enter a URL to an image of a celebrity, then click the **Analyze Image** button to analyze an image and see the result.

# Intelligently generate a thumbnail

The Thumbnail feature of Computer Vision generates a thumbnail from an image. By using the **Smart Crop** feature, the Thumbnail feature will identify the area of interest in an image and center the thumbnail on this area, to generate more aesthetically pleasing thumbnail images.

To complete the Thumbnail feature of the tutorial application, perform the following steps:

## Thumbnail step 1: Add the event handler code for the form button

The **thumbnailImageButtonActionPerformed** event handler method clears the form, displays the image specified in the URL, then calls the **getThumbnailImage** method to create the thumbnail. When **getThumbnailImage** returns, the method displays the generated thumbnail.

Copy and paste the following code into the **thumbnailImageButtonActionPerformed** method.

### NOTE

NetBeans won't let you paste to the method definition line ( `private void` ) or to the closing curly brace of that method. To copy the code, copy the lines between the method definition and the closing curly brace, and paste them over the contents of the method.

```
private void thumbnailImageButtonActionPerformed(java.awt.event.ActionEvent evt) {
    URL thumbnailImageUrl;
    JSONObject jsonError[] = new JSONObject[1];

    // Clear out the previous image, response, and thumbnail, if any.
    thumbnailSourceImage.setIcon(new ImageIcon());
    thumbnailResponseTextArea.setText("");
    thumbnailImage.setIcon(new ImageIcon());

    // Display the image specified in the text box.
    try {
        thumbnailImageUrl = new URL(thumbnailImageUriTextBox.getText());
        BufferedImage bImage = ImageIO.read(thumbnailImageUrl);
        scaleAndShowImage(bImage, thumbnailSourceImage);
    } catch(IOException e) {
        thumbnailResponseTextArea.setText("Error loading image to thumbnail: " + e.getMessage());
        return;
    }

    // Get the thumbnail for the image.
    BufferedImage thumbnail = getThumbnailImage(thumbnailImageUrl.toString(), jsonError);

    // A non-null value indicates error.
    if (jsonError[0] != null) {
        // Format and display the JSON error.
        thumbnailResponseTextArea.setText(jsonError[0].toString(2));
        return;
    }

    // Display the thumbnail.
    if (thumbnail != null) {
        scaleAndShowImage(thumbnail, thumbnailImage);
    }
}
```

## Thumbnail step 2: Add the wrapper for the REST API call

The **getThumbnailImage** method wraps the REST API call to analyze an image. The method returns a **BufferedImage** that contains the thumbnail, or **null** if there was an error. The error message will be returned in the first element of the **jsonError** string array.

Copy and paste the following **getThumbnailImage** method to just underneath the **thumbnailImageButtonActionPerformed** method.

```
/**
 * Encapsulates the Microsoft Cognitive Services REST API call to create a thumbnail for an image.
 * @param imageUrl: The string URL of the image to process.
 * @return: A BufferedImage containing the thumbnail, or null if a runtime error occurs. In the case
 * of an error, the error message will be returned in the first element of the jsonError string array.
 */
private BufferedImage getThumbnailImage(String imageUrl, JSONObject[] jsonError) {
    try (CloseableHttpClient httpClient = HttpClientBuilder.create().build())
    {
        // Create the URI to access the REST API call to identify celebrities in an image.
        String uriString = uriBasePreRegion +
            String.valueOf(subscriptionRegionComboBox.getSelectedIndex()) +
            uriBasePostRegion + uriBaseThumbnail;
        URIBuilder uriBuilder = new URIBuilder(uriString);

        // Request parameters.
        uriBuilder.setParameter("width", "100");
        uriBuilder.setParameter("height", "150");
        uriBuilder.setParameter("smartCropping", "true");

        // Prepare the URI for the REST API call.
        URI uri = uriBuilder.build();
        HttpPost request = new HttpPost(uri);

        // Request headers.
        request.setHeader("Content-Type", "application/json");
        request.setHeader("Ocp-Apim-Subscription-Key", subscriptionKeyTextField.getText());

        // Request body.
        StringEntity requestEntity = new StringEntity("{\"url\":\"" + imageUrl + "\"}");
        request.setEntity(requestEntity);

        // Execute the REST API call and get the response entity.
        HttpResponse response = httpClient.execute(request);
        HttpEntity entity = response.getEntity();

        // Check for success.
        if (response.getStatusLine().getStatusCode() == 200)
        {
            // Return the thumbnail.
            return ImageIO.read(entity.getContent());
        }
        else
        {
            // Format and display the JSON error message.
            String jsonString = EntityUtils.toString(entity);
            jsonError[0] = new JSONObject(jsonString);
            return null;
        }
    }
    catch (Exception e)
    {
        String errorMessage = e.getMessage();
        System.out.println(errorMessage);
        jsonError[0] = new JSONObject(errorMessage);
        return null;
    }
}
```

### Thumbnail step 3: Run the application

Press **F6** to run the application. Put your subscription key into the **Subscription Key** field and verify that you are using the correct region in **Subscription Region**. Click the **Thumbnail** tab, enter a URL to an image, then click the

**Generate Thumbnail** button to analyze an image and see the result.

## Read printed text (OCR)

The Optical Character Recognition (OCR) feature of Computer Vision analyzes an image of printed text. After the analysis is complete, OCR returns a JSON object that contains the text and the location of the text in the image.

To complete the OCR feature of the tutorial application, perform the following steps:

### OCR step 1: Add the event handler code for the form button

The **ocrImageButtonActionPerformed** event handler method clears the form, displays the image specified in the URL, then calls the **OcrImage** method to analyze the image. When **OcrImage** returns, the method displays the detected text as formatted JSON in the **Response** text area.

Copy and paste the following code into the **ocrImageButtonActionPerformed** method.

#### NOTE

NetBeans won't let you paste to the method definition line (`private void`) or to the closing curly brace of that method. To copy the code, copy the lines between the method definition and the closing curly brace, and paste them over the contents of the method.

```
private void ocrImageButtonActionPerformed(java.awt.event.ActionEvent evt) {
    URL ocrImageUrl;

    // Clear out the previous image, response, and caption, if any.
    ocrImage.setIcon(new ImageIcon());
    ocrResponseTextArea.setText("");

    // Display the image specified in the text box.
    try {
        ocrImageUrl = new URL(ocrImageUriTextBox.getText());
        BufferedImage bImage = ImageIO.read(ocrImageUrl);
        scaleAndShowImage(bImage, ocrImage);
    } catch (IOException e) {
        ocrResponseTextArea.setText("Error loading OCR image: " + e.getMessage());
        return;
    }

    // Read the text in the image.
    JSONObject jsonObj = OcrImage(ocrImageUrl.toString());

    // A return of null indicates failure.
    if (jsonObj == null) {
        return;
    }

    // Format and display the JSON response.
    ocrResponseTextArea.setText(jsonObj.toString(2));
}
```

### OCR step 2: Add the wrapper for the REST API call

The **OcrImage** method wraps the REST API call to analyze an image. The method returns a **JSONObject** of the JSON data returned from the call, or **null** if there was an error.

Copy and paste the following **OcrImage** method to just underneath the **ocrImageButtonActionPerformed** method.

```

/**
 * Encapsulates the Microsoft Cognitive Services REST API call to read text in an image.
 * @param imageUrl: The string URL of the image to process.
 * @return: A JSONObject describing the image, or null if a runtime error occurs.
 */
private JSONObject OcrImage(String imageUrl) {
    try (CloseableHttpClient httpClient = HttpClientBuilder.create().build())
    {
        // Create the URI to access the REST API call to read text in an image.
        String uriString = uriBasePreRegion +
            String.valueOf(subscriptionRegionComboBox.getSelectedItem()) +
            uriBasePostRegion + uriBaseOcr;
        URIBuilder uriBuilder = new URIBuilder(uriString);

        // Request parameters.
        uriBuilder.setParameter("language", "unk");
        uriBuilder.setParameter("detectOrientation ", "true");

        // Prepare the URI for the REST API call.
        URI uri = uriBuilder.build();
        HttpPost request = new HttpPost(uri);

        // Request headers.
        request.setHeader("Content-Type", "application/json");
        request.setHeader("Ocp-Apim-Subscription-Key", subscriptionKeyTextField.getText());

        // Request body.
        StringEntity reqEntity = new StringEntity("{\"url\":\"" + imageUrl + "\"}");
        request.setEntity(reqEntity);

        // Execute the REST API call and get the response entity.
        HttpResponse response = httpClient.execute(request);
        HttpEntity entity = response.getEntity();

        // If we got a response, parse it and display it.
        if (entity != null)
        {
            // Return the JSONObject.
            String jsonString = EntityUtils.toString(entity);
            return new JSONObject(jsonString);
        } else {
            // No response. Return null.
            return null;
        }
    }
    catch (Exception e)
    {
        // Display error message.
        System.out.println(e.getMessage());
        return null;
    }
}

```

### OCR step 3: Run the application

Press **F6** to run the application. Put your subscription key into the **Subscription Key** field and verify that you are using the correct region in **Subscription Region**. Click the **OCR** tab, enter a URL to an image of printed text, then click the **Read Image** button to analyze an image and see the result.

## Read handwritten text (handwriting recognition)

The Handwriting Recognition feature of Computer Vision analyzes an image of handwritten text. After the analysis is complete, Handwriting Recognition returns a JSON object that contains the text and the location of the text in the image.



To complete the Handwriting Recognition feature of the tutorial application, perform the following steps:

### Handwriting Recognition step 1: Add the event handler code for the form button

The **handwritingImageButtonActionPerformed** event handler method clears the form, displays the image specified in the URL, then calls the **HandwritingImage** method to analyze the image. When **HandwritingImage** returns, the method displays the detected text as formatted JSON in the **Response** text area.

Copy and paste the following code into the **handwritingImageButtonActionPerformed** method.

#### NOTE

NetBeans won't let you paste to the method definition line ( `private void` ) or to the closing curly brace of that method. To copy the code, copy the lines between the method definition and the closing curly brace, and paste them over the contents of the method.

```
private void handwritingImageButtonActionPerformed(java.awt.event.ActionEvent evt) {
    URL handwritingImageUrl;

    // Clear out the previous image, response, and caption, if any.
    handwritingImage.setIcon(new ImageIcon());
    handwritingResponseTextArea.setText("");

    // Display the image specified in the text box
    try {
        handwritingImageUrl = new URL(handwritingImageUriTextBox.getText());
        BufferedImage bImage = ImageIO.read(handwritingImageUrl);
        scaleAndShowImage(bImage, handwritingImage);
    } catch (IOException e) {
        handwritingResponseTextArea.setText("Error loading Handwriting image: " + e.getMessage());
        return;
    }

    // Read the text in the image.
    JSONObject jsonObj = HandwritingImage(handwritingImageUrl.toString());

    // A return of null indicates failure.
    if (jsonObj == null) {
        return;
    }

    // Format and display the JSON response.
    handwritingResponseTextArea.setText(jsonObj.toString(2));
}
```

### Handwriting Recognition step 2: Add the wrapper for the REST API call

The **HandwritingImage** method wraps the two REST API calls needed to analyze an image. Because handwriting recognition is a time consuming process, a two step process is used. The first call submits the image for processing; the second call retrieves the detected text when the processing is complete.

After the text is retrieved, the **HandwritingImage** method returns a **JSONObject** describing the text and the locations of the text, or **null** if there was an error.

Copy and paste the following **HandwritingImage** method to just underneath the **handwritingImageButtonActionPerformed** method.

```
/**
 * Encapsulates the Microsoft Cognitive Services REST API call to read handwritten text in an image.
 * @param imageUrl: The string URL of the image to process.
 * @return: A JSONObject describing the image, or null if a runtime error occurs.
 */
```

```

private JSONObject HandwritingImage(String imageUrl) {
    try (CloseableHttpClient textClient = HttpClientBuilder.create().build();
        CloseableHttpClient resultClient = HttpClientBuilder.create().build())
    {
        // Create the URI to access the REST API call to read text in an image.
        String uriString = uriBasePreRegion +
            String.valueOf(subscriptionRegionComboBox.getSelectedItem()) +
            uriBasePostRegion + uriBaseHandwriting;
        URIBuilder uriBuilder = new URIBuilder(uriString);

        // Request parameters.
        uriBuilder.setParameter("handwriting", "true");

        // Prepare the URI for the REST API call.
        URI uri = uriBuilder.build();
        HttpPost request = new HttpPost(uri);

        // Request headers.
        request.setHeader("Content-Type", "application/json");
        request.setHeader("Ocp-Apim-Subscription-Key", subscriptionKeyTextField.getText());

        // Request body.
        StringEntity reqEntity = new StringEntity("{\"url\":\"" + imageUrl + "\"}");
        request.setEntity(reqEntity);

        // Execute the REST API call and get the response.
        HttpResponse textResponse = textClient.execute(request);

        // Check for success.
        if (textResponse.getStatusLine().getStatusCode() != 202) {
            // An error occurred. Return the JSON error message.
            HttpEntity entity = textResponse.getEntity();
            String jsonString = EntityUtils.toString(entity);
            return new JSONObject(jsonString);
        }

        String operationLocation = null;

        // The 'Operation-Location' in the response contains the URI to retrieve the recognized text.
        Header[] responseHeaders = textResponse.getAllHeaders();
        for (Header header : responseHeaders) {
            if (header.getName().equals("Operation-Location"))
            {
                // This string is the URI where you can get the text recognition operation result.
                operationLocation = header.getValue();
                break;
            }
        }

        // NOTE: The response may not be immediately available. Handwriting recognition is an
        // async operation that can take a variable amount of time depending on the length
        // of the text you want to recognize. You may need to wait or retry this operation.
        //
        // This example checks once per second for ten seconds.

        JSONObject responseObj = null;
        int i = 0;
        do {
            // Wait one second.
            Thread.sleep(1000);

            // Check to see if the operation completed.
            HttpGet resultRequest = new HttpGet(operationLocation);
            resultRequest.setHeader("Ocp-Apim-Subscription-Key", subscriptionKeyTextField.getText());
            HttpResponse resultResponse = resultClient.execute(resultRequest);
            HttpEntity responseEntity = resultResponse.getEntity();
            if (responseEntity != null)
            {
                // Get the JSON response.
            }
        } while (i < 10);
    }
}

```

```

        String jsonString = EntityUtils.toString(responseEntity);
        responseObj = new JSONObject(jsonString);
    }
}
while (i < 10 && responseObj != null &&
        !responseObj.getString("status").equalsIgnoreCase("Succeeded"));

// If the operation completed, return the JSON object.
if (responseObj != null) {
    return responseObj;
} else {
    // Return null for timeout error.
    System.out.println("Timeout error.");
    return null;
}
}
catch (Exception e)
{
    // Display error message.
    System.out.println(e.getMessage());
    return null;
}
}

```

### Handwriting Recognition step 3: Run the application

To run the application, press **F6**. Put your subscription key into the **Subscription Key** field and verify that you are using the correct region in **Subscription Region**. Click the **Read Handwritten Text** tab, enter a URL to an image of handwritten text, then click the **Read Image** button to analyze an image and see the result.

## Next steps

- [Computer Vision API C# Tutorial](#)
- [Computer Vision API Python Tutorial](#)

# Computer Vision API JavaScript tutorial

9/20/2017 • 18 min to read • [Edit Online](#)

This tutorial shows the features of the Microsoft Cognitive Services Computer Vision REST API.

Explore a JavaScript application that uses the Computer Vision REST API to perform optical character recognition (OCR), create smart-cropped thumbnails, plus detect, categorize, tag, and describe visual features, including faces, in an image. This example lets you submit an image URL for analysis or processing. You can use this open source example as a template for building your own app in JavaScript to use the Computer Vision REST API.

The JavaScript form application has already been written, but has no Computer Vision functionality. In this tutorial, you add the code specific to the Computer Vision REST API to complete the application's functionality.

## Prerequisites

### Platform requirements

This tutorial has been developed using a simple text editor.

### Subscribe to Computer Vision API and get a subscription key

Before creating the example, you must subscribe to Computer Vision API which is part of the Microsoft Cognitive Services. For subscription and key management details, see [Subscriptions](#). Both the primary and secondary keys are valid to use in this tutorial.

## Download the tutorial project

Clone the [Cognitive Services JavaScript Computer Vision Tutorial](#), or download the .zip file and extract it to an empty directory.

If you would prefer to use the finished tutorial with all tutorial code added, you can use the files in the **Completed** folder.

## Add the tutorial code

The JavaScript application is set up with six .html files, one for each feature. Each file demonstrates a different function of Computer Vision (analyze, OCR, etc). The six tutorial sections do not have interdependencies, so you can add the tutorial code to one file, all six files, or only a couple of files. And you can add the tutorial code to the files in any order.

Let's get started.

## Analyze an image

The Analyze feature of Computer Vision analyzes an image for more than 2,000 recognizable objects, living beings, scenery, and actions. Once the analysis is complete, Analyze returns a JSON object that describes the image with descriptive tags, color analysis, captions, and more.

To complete the Analyze feature of the tutorial application, perform the following steps:

### Analyze step 1: Add the event handler code for the form button

Open the **analyze.html** file in a text editor and locate the **analyzeButtonClick** function near the bottom of the file.

The **analyzeButtonClick** event handler function clears the form, displays the image specified in the URL, then calls

the **AnalyzeImage** function to analyze the image.

Copy and paste the following code into the **analyzeButtonClick** function.

```
function analyzeButtonClick() {  
  
    // Clear the display fields.  
    $("#sourceImage").attr("src", "");  
    $("#responseTextArea").val("");  
    $("#captionSpan").text("");  
  
    // Display the image.  
    var sourceImageUrl = $("#inputImage").val();  
    $("#sourceImage").attr("src", sourceImageUrl);  
  
    AnalyzeImage(sourceImageUrl, $("#responseTextArea"), $("#captionSpan"));  
}
```

### Analyze step 2: Add the wrapper for the REST API call

The **AnalyzeImage** function wraps the REST API call to analyze an image. Upon a successful return, the formatted JSON analysis will be displayed in the specified textarea, and the caption will be displayed in the specified span.

Copy and paste the **AnalyzeImage** function code to just underneath the **analyzeButtonClick** function.

```

/* Analyze the image at the specified URL by using Microsoft Cognitive Services Analyze Image API.
 * @param {string} sourceImageUrl - The URL to the image to analyze.
 * @param {<textarea> element} responseTextArea - The text area to display the JSON string returned
 * from the REST API call, or to display the error message if there was
 * an error.
 * @param {<span> element} captionSpan - The span to display the image caption.
 */
function AnalyzeImage(sourceImageUrl, responseTextArea, captionSpan) {
    // Request parameters.
    var params = {
        "visualFeatures": "Categories,Description,Color",
        "details": "",
        "language": "en",
    };

    // Perform the REST API call.
    $.ajax({
        url: common.uriBasePreRegion +
            $("#subscriptionRegionSelect").val() +
            common.uriBasePostRegion +
            common.uriBaseAnalyze +
            "?" +
            $.param(params),

        // Request headers.
        beforeSend: function(jqXHR){
            jqXHR.setRequestHeader("Content-Type", "application/json");
            jqXHR.setRequestHeader("Ocp-Apim-Subscription-Key",
                encodeURIComponent($("#subscriptionKeyInput").val()));
        },

        type: "POST",

        // Request body.
        data: '{"url": "' + sourceImageUrl + '"}',
    })

    .done(function(data) {
        // Show formatted JSON on webpage.
        responseTextArea.val(JSON.stringify(data, null, 2));

        // Extract and display the caption and confidence from the first caption in the description object.
        if (data.description && data.description.captions) {
            var caption = data.description.captions[0];

            if (caption.text && caption.confidence) {
                captionSpan.text("Caption: " + caption.text +
                    " (confidence: " + caption.confidence + ").");
            }
        }
    })

    .fail(function(jqXHR, textStatus, errorThrown) {
        // Prepare the error string.
        var errorString = (errorThrown === "") ? "Error. " : errorThrown + " (" + jqXHR.status + "): ";
        errorString += (jqXHR.responseText === "") ? "" : (jQuery.parseJSON(jqXHR.responseText).message) ?
            jQuery.parseJSON(jqXHR.responseText).message : jQuery.parseJSON(jqXHR.responseText).error.message;

        // Put the error JSON in the response textarea.
        responseTextArea.val(JSON.stringify(jqXHR, null, 2));

        // Show the error message.
        alert(errorString);
    });
}

```

### Analyze step 3: Run the application

Save the **analyze.html** file and open it in a Web browser. Put your subscription key into the **Subscription Key** field and verify that you are using the correct region in **Subscription Region**. Enter a URL to an image to analyze, then click the **Analyze Image** button to analyze an image and see the result.

## Recognize a landmark

The Landmark feature of Computer Vision analyzes an image for natural and artificial landmarks, such as mountains or famous buildings. Once the analysis is complete, Landmark returns a JSON object that identifies the landmarks found in the image.

To complete the Landmark feature of the tutorial application, perform the following steps:

### Landmark step 1: Add the event handler code for the form button

Open the **landmark.html** file in a text editor and locate the **landmarkButtonClick** function near the bottom of the file.

The **landmarkButtonClick** event handler function clears the form, displays the image specified in the URL, then calls the **IdentifyLandmarks** function to analyze the image.

Copy and paste the following code into the **landmarkButtonClick** function.

```
function landmarkButtonClick() {  
  
    // Clear the display fields.  
    $("#sourceImage").attr("src", "");  
    $("#responseTextArea").val("");  
    $("#captionSpan").text("");  
  
    // Display the image.  
    var sourceImageUrl = $("#inputImage").val();  
    $("#sourceImage").attr("src", sourceImageUrl);  
  
    IdentifyLandmarks(sourceImageUrl, $("#responseTextArea"), $("#captionSpan"));  
}
```

### Landmark step 2: Add the wrapper for the REST API call

The **IdentifyLandmarks** function wraps the REST API call to analyze an image. Upon a successful return, the formatted JSON analysis will be displayed in the specified textarea, and the caption will be displayed in the specified span.

Copy and paste the **IdentifyLandmarks** function code to just underneath the **landmarkButtonClick** function.

```

/* Identify landmarks in the image at the specified URL by using Microsoft Cognitive Services
 * Landmarks API.
 * @param {string} sourceImageUrl - The URL to the image to analyze for landmarks.
 * @param {<textarea> element} responseTextArea - The text area to display the JSON string returned
 * from the REST API call, or to display the error message if there was
 * an error.
 * @param {<span> element} captionSpan - The span to display the image caption.
 */
function IdentifyLandmarks(sourceImageUrl, responseTextArea, captionSpan) {
    // Request parameters.
    var params = {
        "model": "landmarks"
    };

    // Perform the REST API call.
    $.ajax({
        url: common.uriBasePreRegion +
            $("#subscriptionRegionSelect").val() +
            common.uriBasePostRegion +
            common.uriBaseLandmark +
            "?" +
            $.param(params),

        // Request headers.
        beforeSend: function(jqXHR) {
            jqXHR.setRequestHeader("Content-Type", "application/json");
            jqXHR.setRequestHeader("Ocp-Apim-Subscription-Key",
                encodeURIComponent($("#subscriptionKeyInput").val()));
        },

        type: "POST",

        // Request body.
        data: '{"url": "' + sourceImageUrl + '"}',
    })

    .done(function(data) {
        // Show formatted JSON on webpage.
        responseTextArea.val(JSON.stringify(data, null, 2));

        // Extract and display the caption and confidence from the first caption in the description object.
        if (data.result && data.result.landmarks) {
            var landmark = data.result.landmarks[0];

            if (landmark.name && landmark.confidence) {
                captionSpan.text("Landmark: " + landmark.name +
                    " (confidence: " + landmark.confidence + ").");
            }
        }
    })

    .fail(function(jqXHR, textStatus, errorThrown) {
        // Prepare the error string.
        var errorString = (errorThrown === "") ? "Error. " : errorThrown + " (" + jqXHR.status + "): ";
        errorString += (jqXHR.responseText === "") ? "" : (jQuery.parseJSON(jqXHR.responseText).message) ?
            jQuery.parseJSON(jqXHR.responseText).message : jQuery.parseJSON(jqXHR.responseText).error.message;

        // Put the error JSON in the response text area.
        responseTextArea.val(JSON.stringify(jqXHR, null, 2));

        // Show the error message.
        alert(errorString);
    });
}

```

### Landmark step 3: Run the application



Save the **landmark.html** file and open it in a Web browser. Put your subscription key into the **Subscription Key** field and verify that you are using the correct region in **Subscription Region**. Enter a URL to an image to analyze, then click the **Analyze Image** button to analyze an image and see the result.

## Recognize celebrities

The Celebrities feature of Computer Vision analyzes an image for famous people. Once the analysis is complete, Celebrities returns a JSON object that identifies the Celebrities found in the image.

To complete the Celebrities feature of the tutorial application, perform the following steps:

### Celebrities step 1: Add the event handler code for the form button

Open the **celebrities.html** file in a text editor and locate the **celebritiesButtonClick** function near the bottom of the file.

The **celebritiesButtonClick** event handler function clears the form, displays the image specified in the URL, then calls the **IdentifyCelebrities** function to analyze the image.

Copy and paste the following code into the **celebritiesButtonClick** function.

```
function celebritiesButtonClick() {  
  
    // Clear the display fields.  
    $("#sourceImage").attr("src", "");  
    $("#responseTextArea").val("");  
    $("#captionSpan").text("");  
  
    // Display the image.  
    var sourceImageUrl = $("#inputImage").val();  
    $("#sourceImage").attr("src", sourceImageUrl);  
  
    IdentifyCelebrities(sourceImageUrl, $("#responseTextArea"), $("#captionSpan"));  
}
```

### Celebrities step 2: Add the wrapper for the REST API call

```

/* Identify celebrities in the image at the specified URL by using Microsoft Cognitive Services
 * Celebrities API.
 * @param {string} sourceImageUrl - The URL to the image to analyze for celebrities.
 * @param {<textarea> element} responseTextArea - The text area to display the JSON string returned
 * from the REST API call, or to display the error message if there was
 * an error.
 * @param {<span> element} captionSpan - The span to display the image caption.
 */
function IdentifyCelebrities(sourceImageUrl, responseTextArea, captionSpan) {
    // Request parameters.
    var params = {
        "model": "celebrities"
    };

    // Perform the REST API call.
    $.ajax({
        url: common.uriBasePreRegion +
            $("#subscriptionRegionSelect").val() +
            common.uriBasePostRegion +
            common.uriBaseCelebrities +
            "?" +
            $.param(params),

        // Request headers.
        beforeSend: function(jqXHR) {
            jqXHR.setRequestHeader("Content-Type", "application/json");
            jqXHR.setRequestHeader("Ocp-Apim-Subscription-Key",
                encodeURIComponent($("#subscriptionKeyInput").val()));
        },

        type: "POST",

        // Request body.
        data: '{"url": ' + "'" + sourceImageUrl + "'",
    });

    .done(function(data) {
        // Show formatted JSON on webpage.
        responseTextArea.val(JSON.stringify(data, null, 2));

        // Extract and display the caption and confidence from the first caption in the description object.
        if (data.result && data.result.celebrities) {
            var celebrity = data.result.celebrities[0];

            if (celebrity.name && celebrity.confidence) {
                captionSpan.text("Celebrity name: " + celebrity.name +
                    " (confidence: " + celebrity.confidence + ").");
            }
        }
    });

    .fail(function(jqXHR, textStatus, errorThrown) {
        // Prepare the error string.
        var errorString = (errorThrown === "") ? "Error. " : errorThrown + " (" + jqXHR.status + "): ";
        errorString += (jqXHR.responseText === "") ? "" : (jQuery.parseJSON(jqXHR.responseText).message) ?
            jQuery.parseJSON(jqXHR.responseText).message : jQuery.parseJSON(jqXHR.responseText).error.message;

        // Put the error JSON in the response text area.
        responseTextArea.val(JSON.stringify(jqXHR, null, 2));

        // Show the error message.
        alert(errorString);
    });
}

```

### Celebrities step 3: Run the application

Save the **celebrities.html** file and open it in a Web browser. Put your subscription key into the **Subscription Key** field and verify that you are using the correct region in **Subscription Region**. Enter a URL to an image to analyze, then click the **Analyze Image** button to analyze an image and see the result.

## Intelligently generate a thumbnail

The Thumbnail feature of Computer Vision generates a thumbnail from an image. By using the **Smart Crop** feature, the Thumbnail feature will identify the area of interest in an image and center the thumbnail on this area, to generate more aesthetically pleasing thumbnail images.

To complete the Thumbnail feature of the tutorial application, perform the following steps:

### Thumbnail step 1: Add the event handler code for the form button

Open the **thumbnail.html** file in a text editor and locate the **thumbnailButtonClick** function near the bottom of the file.

The **thumbnailButtonClick** event handler function clears the form, displays the image specified in the URL, then calls the **getThumbnail** function twice to create two thumbnails, one smart cropped and one without smart crop.

Copy and paste the following code into the **thumbnailButtonClick** function.

```
function thumbnailButtonClick() {

    // Clear the display fields.
    document.getElementById("sourceImage").src = "#";
    document.getElementById("thumbnailImageSmartCrop").src = "#";
    document.getElementById("thumbnailImageNonSmartCrop").src = "#";
    document.getElementById("responseTextArea").value = "";
    document.getElementById("captionSpan").text = "";

    // Display the image.
    var sourceImageUrl = document.getElementById("inputImage").value;
    document.getElementById("sourceImage").src = sourceImageUrl;

    // Get a smart cropped thumbnail.
    getThumbnail(sourceImageUrl, true, document.getElementById("thumbnailImageSmartCrop"),
        document.getElementById("responseTextArea"));

    // Get a non-smart-cropped thumbnail.
    getThumbnail(sourceImageUrl, false, document.getElementById("thumbnailImageNonSmartCrop"),
        document.getElementById("responseTextArea"));
}
```

### Thumbnail step 2: Add the wrapper for the REST API call

The **getThumbnail** function wraps the REST API call to analyze an image. Upon a successful return, the thumbnail will be displayed in the specified img element.

Copy and paste the following **getThumbnail** function to just underneath the **thumbnailButtonClick** function.

```
/* Get a thumbnail of the image at the specified URL by using Microsoft Cognitive Services
 * Thumbnail API.
 * @param {string} sourceImageUrl URL to image.
 * @param {boolean} smartCropping Set to true to use the smart cropping feature which crops to the
 * more interesting area of an image; false to crop for the center
 * of the image.
 * @param {<img> element} imageElement The img element in the DOM which will display the thumbnail image.
 * @param {<textarea> element} responseTextArea - The text area to display the Response Headers returned
 * from the REST API call, or to display the error message if there was
 * an error.
 */
function getThumbnail(sourceImageUrl, smartCropping, imageElement, responseTextArea) {
```

```

// Create the HTTP Request object.
var xhr = new XMLHttpRequest();

// Request parameters.
var params = "width=100&height=150&smartCropping=" + smartCropping.toString();

// Build the full URI.
var fullUri = common.uriBasePreRegion +
    document.getElementById("subscriptionRegionSelect").value +
    common.uriBasePostRegion +
    common.uriBaseThumbnail +
    "?" +
    params;

// Identify the request as a POST, with the URI and parameters.
xhr.open("POST", fullUri);

// Add the request headers.
xhr.setRequestHeader("Content-Type", "application/json");
xhr.setRequestHeader("Ocp-Apim-Subscription-Key",
    encodeURIComponent(document.getElementById("subscriptionKeyInput").value));

// Set the response type to "blob" for the thumbnail image data.
xhr.responseType = "blob";

// Process the result of the REST API call.
xhr.onreadystatechange = function(e) {
    if(xhr.readyState === XMLHttpRequest.DONE) {

        // Thumbnail successfully created.
        if(xhr.status === 200) {
            // Show response headers.
            var s = JSON.stringify(xhr.getAllResponseHeaders(), null, 2);
            responseTextArea.value = JSON.stringify(xhr.getAllResponseHeaders(), null, 2);

            // Show thumbnail image.
            var urlCreator = window.URL || window.webkitURL;
            var imageUrl = urlCreator.createObjectURL(this.response);
            imageElement.src = imageUrl;
        } else {
            // Display the error message. The error message is the response body as a JSON string.
            // The code in this code block extracts the JSON string from the blob response.
            var reader = new FileReader();

            // This event fires after the blob has been read.
            reader.addEventListener('loadend', (e) => {
                responseTextArea.value = JSON.stringify(JSON.parse(e.srcElement.result), null, 2);
            });

            // Start reading the blob as text.
            reader.readAsText(xhr.response);
        }
    }
}

// Execute the REST API call.
xhr.send(JSON.stringify({url: ' ' + sourceImageUrl + ' '}));
}

```

### Thumbnail step 3: Run the application

Save the **thumbnail.html** file and open it in a Web browser. Put your subscription key into the **Subscription Key** field and verify that you are using the correct region in **Subscription Region**. Enter a URL to an image to analyze, then click the **Generate Thumbnails** button to analyze an image and see the result.

## Read printed text (OCR)

The Optical Character Recognition (OCR) feature of Computer Vision analyzes an image of printed text. After the analysis is complete, OCR returns a JSON object that contains the text and the location of the text in the image.

To complete the OCR feature of the tutorial application, perform the following steps:

### OCR step 1: Add the event handler code for the form button

Open the **ocr.html** file in a text editor and locate the **ocrButtonClick** function near the bottom of the file.

The **ocrButtonClick** event handler function clears the form, displays the image specified in the URL, then calls the **ReadOcrImage** function to analyze the image.

Copy and paste the following code into the **ocrButtonClick** function.

```
function ocrButtonClick() {  
  
    // Clear the display fields.  
    $("#sourceImage").attr("src", "");  
    $("#responseTextArea").val("");  
    $("#captionSpan").text("");  
  
    // Display the image.  
    var sourceImageUrl = $("#inputImage").val();  
    $("#sourceImage").attr("src", sourceImageUrl);  
  
    ReadOcrImage(sourceImageUrl, $("#responseTextArea"));  
}
```

### OCR step 2: Add the wrapper for the REST API call

The **ReadOcrImage** function wraps the REST API call to analyze an image. Upon a successful return, the formatted JSON describing the text and the location of the text will be displayed in the specified textarea.

Copy and paste the following **ReadOcrImage** function to just underneath the **ocrButtonClick** function.

```

/* Recognize and read printed text in an image at the specified URL by using Microsoft Cognitive
 * Services OCR API.
 * @param {string} sourceImageUrl - The URL to the image to analyze for printed text.
 * @param {<textarea> element} responseTextArea - The text area to display the JSON string returned
 * from the REST API call, or to display the error message if there was
 * an error.
 */
function ReadOcrImage(sourceImageUrl, responseTextArea) {
    // Request parameters.
    var params = {
        "language": "unk",
        "detectOrientation": "true",
    };

    // Perform the REST API call.
    $.ajax({
        url: common.uriBasePreRegion +
            $("#subscriptionRegionSelect").val() +
            common.uriBasePostRegion +
            common.uriBaseOcr +
            "?" +
            $.param(params),

        // Request headers.
        beforeSend: function(jqXHR) {
            jqXHR.setRequestHeader("Content-Type", "application/json");
            jqXHR.setRequestHeader("Ocp-Apim-Subscription-Key",
                encodeURIComponent($("#subscriptionKeyInput").val()));
        },

        type: "POST",

        // Request body.
        data: '{"url": ' + "'" + sourceImageUrl + "'" + '}',
    })

    .done(function(data) {
        // Show formatted JSON on webpage.
        responseTextArea.val(JSON.stringify(data, null, 2));
    })

    .fail(function(jqXHR, textStatus, errorThrown) {
        // Put the JSON description into the text area.
        responseTextArea.val(JSON.stringify(jqXHR, null, 2));

        // Display error message.
        var errorString = (errorThrown === "") ? "Error. " : errorThrown + " (" + jqXHR.status + "): ";
        errorString += (jqXHR.responseText === "") ? "" : (jQuery.parseJSON(jqXHR.responseText).message) ?
            jQuery.parseJSON(jqXHR.responseText).message : jQuery.parseJSON(jqXHR.responseText).error.message;
        alert(errorString);
    });
}

```

### OCR step 3: Run the application

Save the **ocr.html** file and open it in a Web browser. Put your subscription key into the **Subscription Key** field and verify that you are using the correct region in **Subscription Region**. Enter a URL to an image of text to read, then click the **Read Image** button to analyze an image and see the result.

## Read handwritten text (Handwriting Recognition)

The Handwriting Recognition feature of Computer Vision analyzes an image of handwritten text. After the analysis is complete, Handwriting Recognition returns a JSON object that contains the text and the location of the text in the image.

To complete the Handwriting Recognition feature of the tutorial application, perform the following steps:

### Handwriting Recognition step 1: Add the event handler code for the form button

Open the **handwriting.html** file in a text editor and locate the **handwritingButtonClick** function near the bottom of the file.

The **handwritingButtonClick** event handler function clears the form, displays the image specified in the URL, then calls the **ReadHandwrittenImage** function to analyze the image.

Copy and paste the following code into the **handwritingButtonClick** function.

```
function handwritingButtonClick() {  
  
    // Clear the display fields.  
    $("#sourceImage").attr("src", "");  
    $("#responseTextArea").val("");  
  
    // Display the image.  
    var sourceImageUrl = $("#inputImage").val();  
    $("#sourceImage").attr("src", sourceImageUrl);  
  
    ReadHandwrittenImage(sourceImageUrl, $("#responseTextArea"));  
}
```

### Handwriting Recognition step 2: Add the wrapper for the REST API call

The **ReadHandwrittenImage** function wraps the two REST API calls needed to analyze an image. Because Handwriting Recognition is a time consuming process, a two step process is used. The first call submits the image for processing; the second call retrieves the detected text when the processing is complete.

After the text is retrieved, the formatted JSON describing the text and the location of the text will be displayed in the specified textarea.

Copy and paste the following **ReadHandwrittenImage** function to just underneath the **handwritingButtonClick** function.

```
/* Recognize and read text from an image of handwriting at the specified URL by using Microsoft  
 * Cognitive Services Recognize Handwritten Text API.  
 * @param {string} sourceImageUrl - The URL to the image to analyze for handwriting.  
 * @param {<textarea> element} responseTextArea - The text area to display the JSON string returned  
 * from the REST API call, or to display the error message if there was  
 * an error.  
 */  
function ReadHandwrittenImage(sourceImageUrl, responseTextArea) {  
    // Request parameters.  
    var params = {  
        "handwriting": "true",  
    };  
  
    // This operation requires two REST API calls. One to submit the image for processing,  
    // the other to retrieve the text found in the image.  
    //  
    // Perform the first REST API call to submit the image for processing.  
    $.ajax({  
        url: common.uriBasePreRegion +  
            $("#subscriptionRegionSelect").val() +  
            common.uriBasePostRegion +  
            common.uriBaseHandwriting +  
            "?" +  
            $.param(params),  
  
        // Request headers.  
        beforeSend: function(jqXHR) {  
            jqXHR.setRequestHeader("Content-Type", "application/json");  
        }  
    });  
}
```

```

jqXHR.setRequestHeader("Content-Type", "application/json");
jqXHR.setRequestHeader("Ocp-Apim-Subscription-Key",
    encodeURIComponent($("#subscriptionKeyInput").val()));
},

type: "POST",

// Request body.
data: '{"url": ' + "" + sourceImageUrl + ""}',
})

.done(function(data, textStatus, jqXHR) {
    // Show progress.
    responseTextArea.val("Handwritten image submitted.");

    // Note: The response may not be immediately available. Handwriting Recognition is an
    // async operation that can take a variable amount of time depending on the length
    // of the text you want to recognize. You may need to wait or retry this GET operation.
    //
    // Try once per second for up to ten seconds to receive the result.
    var tries = 10;
    var waitTime = 100;
    var taskCompleted = false;

    var timeoutID = setInterval(function () {
        // Limit the number of calls.
        if (--tries <= 0) {
            window.clearTimeout(timeoutID);
            responseTextArea.val("The response was not available in the time allowed.");
            return;
        }

        // The "Operation-Location" in the response contains the URI to retrieve the recognized text.
        var operationLocation = jqXHR.getResponseHeader("Operation-Location");

        // Perform the second REST API call and get the response.
        $.ajax({
            url: operationLocation,

            // Request headers.
            beforeSend: function(jqXHR){
                jqXHR.setRequestHeader("Content-Type", "application/json");
                jqXHR.setRequestHeader("Ocp-Apim-Subscription-Key",
                    encodeURIComponent($("#subscriptionKeyInput").val()));
            },

            type: "GET",
        })

        .done(function(data) {
            // If the result is not yet available, return.
            if (data.status && (data.status === "NotStarted" || data.status === "Running")) {
                return;
            }

            // Show formatted JSON on webpage.
            responseTextArea.val(JSON.stringify(data, null, 2));

            // Indicate the task is complete and clear the timer.
            taskCompleted = true;
            window.clearTimeout(timeoutID);
        })

        .fail(function(jqXHR, textStatus, errorThrown) {
            // Indicate the task is complete and clear the timer.
            taskCompleted = true;
            window.clearTimeout(timeoutID);

            // Display error message.
            var errorString = (errorThrown === "") ? "Error: " + errorThrown : " (" + jqXHR.status + ": " +

```



```

        var errorString = (errorThrown === undefined) ? "Error: " + errorThrown + " (" + jqXHR.status + ") : ";
        errorString += (jqXHR.responseText === "") ? "" : (jQuery.parseJSON(jqXHR.responseText).message) ?
            jQuery.parseJSON(jqXHR.responseText).message : jQuery.parseJSON(jqXHR.responseText).error.message;
        alert(errorString);
    });
    }, waitTime);
})

.fail(function(jqXHR, textStatus, errorThrown) {
    // Put the JSON description into the text area.
    responseTextArea.val(JSON.stringify(jqXHR, null, 2));

    // Display error message.
    var errorString = (errorThrown === undefined) ? "Error: " + errorThrown + " (" + jqXHR.status + ") : ";
    errorString += (jqXHR.responseText === "") ? "" : (jQuery.parseJSON(jqXHR.responseText).message) ?
        jQuery.parseJSON(jqXHR.responseText).message : jQuery.parseJSON(jqXHR.responseText).error.message;
    alert(errorString);
});
}

```

### Handwriting Recognition step 3: Run the application

Save the **handwriting.html** file and open it in a Web browser. Put your subscription key into the **Subscription Key** field and verify that you are using the correct region in **Subscription Region**. Enter a URL to an image of text to read, then click the **Read Image** button to analyze an image and see the result.

## Next steps

- [Computer Vision API C# Tutorial](#)
- [Computer Vision API Python Tutorial](#)

# Computer Vision API Python Tutorial

6/27/2017 • 1 min to read • [Edit Online](#)

This tutorial shows you how to use the Computer Vision API in Python and how to visualize your results using some popular libraries. Use Jupyter to run the tutorial. To learn how to get started with interactive Jupyter notebooks, refer to: [Jupyter Documentation](#).

## Opening the Tutorial Notebook in Jupyter

1. Navigate to the [tutorial notebook in GitHub](#).
2. Click on the green button to clone or download the tutorial.
3. Open a command prompt and go to the folder *Cognitive-Vision-Python-master\Jupyter Notebook*.
4. Run the command **jupyter notebook** from the command prompt. This will start Jupyter.
5. In the Jupyter window, click on *Computer Vision API Example.ipynb* to open the tutorial notebook

## Running the Tutorial

To use this notebook, you will need a subscription key for the Computer Vision API. Visit the [Subscription page](#) to sign up. On the "Sign in" page, use your Microsoft account to sign in and you will be able to subscribe and get free keys. After completing the sign-up process, paste your key into the variables section of the notebook (reproduced below). Either the primary or the secondary key works. Make sure to enclose the key in quotes to make it a string.

```
# Variables

_url = 'https://westcentralus.api.cognitive.microsoft.com/vision/v1/analyses'
_key = None #Here you have to paste your primary key
_maxNumRetries = 10
```

# 86-Categories Taxonomy

6/27/2017 • 1 min to read • [Edit Online](#)

abstract\_

abstract\_net

abstract\_nonphoto

abstract\_rect

abstract\_shape

abstract\_texture

animal\_

animal\_bird

animal\_cat

animal\_dog

animal\_horse

animal\_panda

building\_

building\_arch

building\_brickwall

building\_church

building\_corner

building\_doorwindows

building\_pillar

building\_stair

building\_street

dark\_

drink\_

drink\_can

dark\_fire

dark\_fireworks

sky\_object

food\_

food\_bread

food\_fastfood

food\_grilled  
food\_pizza  
indoor\_  
indoor\_churchwindow  
indoor\_court  
indoor\_doorwindows  
indoor\_marketstore  
indoor\_room  
indoor\_venue  
dark\_light  
others\_  
outdoor\_  
outdoor\_city  
outdoor\_field  
outdoor\_grass  
outdoor\_house  
outdoor\_mountain  
outdoor\_oceanbeach  
outdoor\_playground  
outdoor\_railway  
outdoor\_road  
outdoor\_sportsfield  
outdoor\_stonerock  
outdoor\_street  
outdoor\_water  
outdoor\_waterside  
people\_  
people\_baby  
people\_crowd  
people\_group  
people\_hand  
people\_many  
people\_portrait

people\_show

people\_tattoo

people\_young

plant\_

plant\_branch

plant\_flower

plant\_leaves

plant\_tree

object\_screen

object\_sculpture

sky\_cloud

sky\_sun

people\_swimming

outdoor\_pool

text\_

text\_mag

text\_map

text\_menu

text\_sign

trans\_bicycle

trans\_bus

trans\_car

trans\_trainstation

# Computer Vision API Frequently Asked Questions

7/10/2017 • 2 min to read • [Edit Online](#)

If you can't find answers to your questions in this FAQ, try asking the Computer Vision API community on [StackOverflow](#) or contact [Help and Support on UserVoice](#)

**Question:** *Can I train Computer Vision API to use custom tags? For example, I would like to feed in pictures of cat breeds to 'train' the AI, then receive the breed value on an AI request.*

**Answer:** This function is currently not available. However, our engineers are working to bring this functionality to Computer Vision.

**Question:** *Can Computer Vision be used locally without an internet connection?*

**Answer:** We currently do not offer an on-premises or local solution.

**Question:** *Which languages are supported with Computer Vision?*

**Answer:** Supported languages include:

		SUPPORTED LANGUAGES		
Danish (da-DK)	Dutch (nl-NL)	English	Finnish (fi-FI)	French (fr-FR)
German (de-DE)	Greek (el-GR)	Hungarian (hu-HU)	Italian (it-IT)	Japanese (ja-JP)
Korean (ko-KR)	Norwegian (nb-NO)	Polish (pl-PL)	Portuguese (pt-BR) (pt-PT)	Russian (ru-RU)
Spanish (es-ES)	Swedish (sv-SV)	Turkish (tr-TU)		

**Question:** *Can Computer Vision be used to read license plates?*

**Answer:** The Vision API offers good text-detection with OCR, but it is not currently optimized for license plates. We are constantly trying to improve our services and have added OCR for auto license plate recognition to our list of feature requests.

**Question:** *Which languages are supported for handwriting recognition?*

**Answer:** Currently, only English is supported.

**Question:** *What types of writing surfaces are supported for handwriting recognition?*

**Answer:** The technology works with different kinds of surfaces, including whiteboards, white paper, and yellow sticky notes.

**Question:** *How long does the handwriting recognition operation take?*

**Answer:** The amount of time that it takes depends on the length of the text. For longer texts, it can take up to several seconds. Therefore, after the Recognize Handwritten Text operation completes, you may need to wait before you can retrieve the results using the Get Handwritten Text Operation Result operation.

**Question:** *How does the handwriting recognition technology handle text that was inserted using a caret in the middle of a line?*

**Answer:** Such text is returned as a separate line by the handwriting recognition operation.

---

**Question:** *How does the handwriting recognition technology handle crossed-out words or lines?*

**Answer:** If the words are crossed out with multiple lines to render them unrecognizable, the handwriting recognition operation doesn't pick them up. However, if the words are crossed out using a single line, that crossing is treated as noise, and the words still get picked up by the handwriting recognition operation.

---

**Question:** *What text orientations are supported for the handwriting recognition technology?*

**Answer:** Text oriented at angles of up to around 30 degrees to 40 degrees may get picked up by the handwriting recognition operation.

---

# The Research Behind Computer Vision API

6/27/2017 • 1 min to read • [Edit Online](#)

Hao Fang, Saurabh Gupta, Forrest Iandola, Rupesh Srivastava, Li Deng, Piotr Dollar, Jianfeng Gao, Xiaodong He, Margaret Mitchell, John Platt, Lawrence Zitnick, and Geoffrey Zweig, *From Captions to Visual Concepts and Back*, CVPR, June 2015 (**Won 1st Prize at the COCO Image Captioning Challenge 2015**)

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. *Deep Residual Learning for Image Recognition*. arXiv (**Won both ImageNet and MS COCO competitions 2015**)

Yandong Guo, Lei Zhang, Yuxiao Hu, Xiaodong He, Jianfeng Gao, *MS-Celeb-1M: Challenge of Recognizing One Million Celebrities in the Real World*, IS&T International Symposium on Electronic Imaging, 2016

Xiao Zhang, Lei Zhang, Xin-Jing Wang, Heung-Yeung Shum, *Finding Celebrities in Billions of Web Images*, accepted by IEEE Transaction on Multimedia, 2012