Computer Vision Fundamentals with Google Cloud

Professional Machine Learning Engineer Certification Learning Path navigate_next Computer Vision Fundamentals with Google Cloud navigate_next Introduction to Computer Vision and Pre-built ML Models for Image Classification

Detecting Labels, Faces, and Landmarks in Images with the Cloud Vision API

1 hour Free

Overview

The Cloud Vision API lets you understand the content of an image by encapsulating powerful machine learning models in a simple REST API.

In this lab, you send images to the Vision API and see it detect objects, faces, and landmarks.

Lab objectives

In this lab, you learn how to perform the following tasks:

- Create a Vision API request and call the API with `curl`.
- Use the label, face, and landmark detection methods of the vision API.

Task 0. Setup and requirements

For each lab, you get a new Google Cloud project and set of resources for a fixed time at no cost.

1. Sign in to Qwiklabs using an **incognito window**.
2. Note the lab's access time (for example, `1:15:00`), and make sure you can finish within that time.
   There is no pause feature. You can restart if needed, but you have to start at the beginning.
3. When ready, click **Start lab**.
4. Note your lab credentials (**Username** and **Password**). You will use them to sign in to the Google Cloud Console.
5. Click **Open Google Console**.

6. Click **Use another account** and copy/paste credentials for **this** lab into the prompts.
   If you use other credentials, you'll receive errors or **incur charges**.
7. Accept the terms and skip the recovery resource page.

Note: Do not click **End Lab** unless you have finished the lab or want to restart it. This clears your work and removes the project.

Activate Cloud Shell

Cloud Shell is a virtual machine that contains development tools. It offers a persistent 5-GB home directory and runs on Google Cloud. Cloud Shell provides command-line access to your Google Cloud resources. `gcloud` is the command-line tool for Google Cloud. It comes pre-installed on Cloud Shell and supports tab completion.

1. Click the **Activate Cloud Shell** button ( ) at the top right of the console.
2. Click **Continue**.
   It takes a few moments to provision and connect to the environment. When you are connected, you are also authenticated, and the project is set to your *PROJECT_ID*.

Sample commands

- List the active account name:

gcloud auth list

(Output)

Credentialed accounts: - <myaccount>@<mydomain>.com (active)

(Example output)

Credentialed accounts: - google1623327_student@qwiklabs.net

- List the project ID:

gcloud config list project

(Output)

[core] project = <project_ID>

(Example output)

[core] project = qwiklabs-gcp-44776a13dea667a6 **Note:** Full documentation of **gcloud** is available in the gcloud CLI overview guide.

Task 1. Create an API Key

Since you are using `curl` to send a request to the Vision API, you need to generate an API key to pass in your request URL.

1. To create an API key, navigate to **APIs & Services** > **Credentials** in your Cloud console:
2. Click **CREATE CREDENTIALS** and select **API key**.
3. Next, copy the key you just generated and click **CLOSE**.

Click **Check my progress** to verify the objective. Create an API Key

Now save it to an environment variable to avoid having to insert the value of your API key in each request.

Run the following in Cloud Shell, replacing `<YOUR_API_KEY>` with the key you just copied:

export API_KEY= XYX

Task 2. Upload an Image to a Cloud Storage bucket

Create a Cloud Storage bucket

There are two ways to send an image to the Vision API for image detection: by sending the API a base64 encoded image string, or passing it the URL of a file stored in Cloud Storage. You use a Cloud Storage URL. The first step is to create a Cloud Storage bucket to store your images.

1. Navigate to **Navigation menu** > **Cloud Storage** in the Cloud console for your project, then click **CREATE BUCKET**.
2. Set a unique name (use your project ID because it is unique) and then choose a regional bucket (For example set the region to us-central1).
3. After naming your bucket, click **Choose how to control access to objects**.

Uncheck **Enforce public access prevention on this bucket** and select the **Fine-grained** circle:

- **Choose how to control access to objects**

  Prevent public access

  Restrict data from being publicly accessible via the internet. Will prevent this bucket from being used for web hosting. Learn more

  ☐ Enforce public access prevention on this bucket

  Access control

  ○ Uniform
  Ensure uniform access to all objects in the bucket by using only bucket-level permissions (IAM). This option becomes permanent after 90 days. Learn more

  ◉ Fine-grained
  Specify access to individual objects by using object-level permissions (ACLs) in addition to your bucket-level permissions (IAM). Learn more

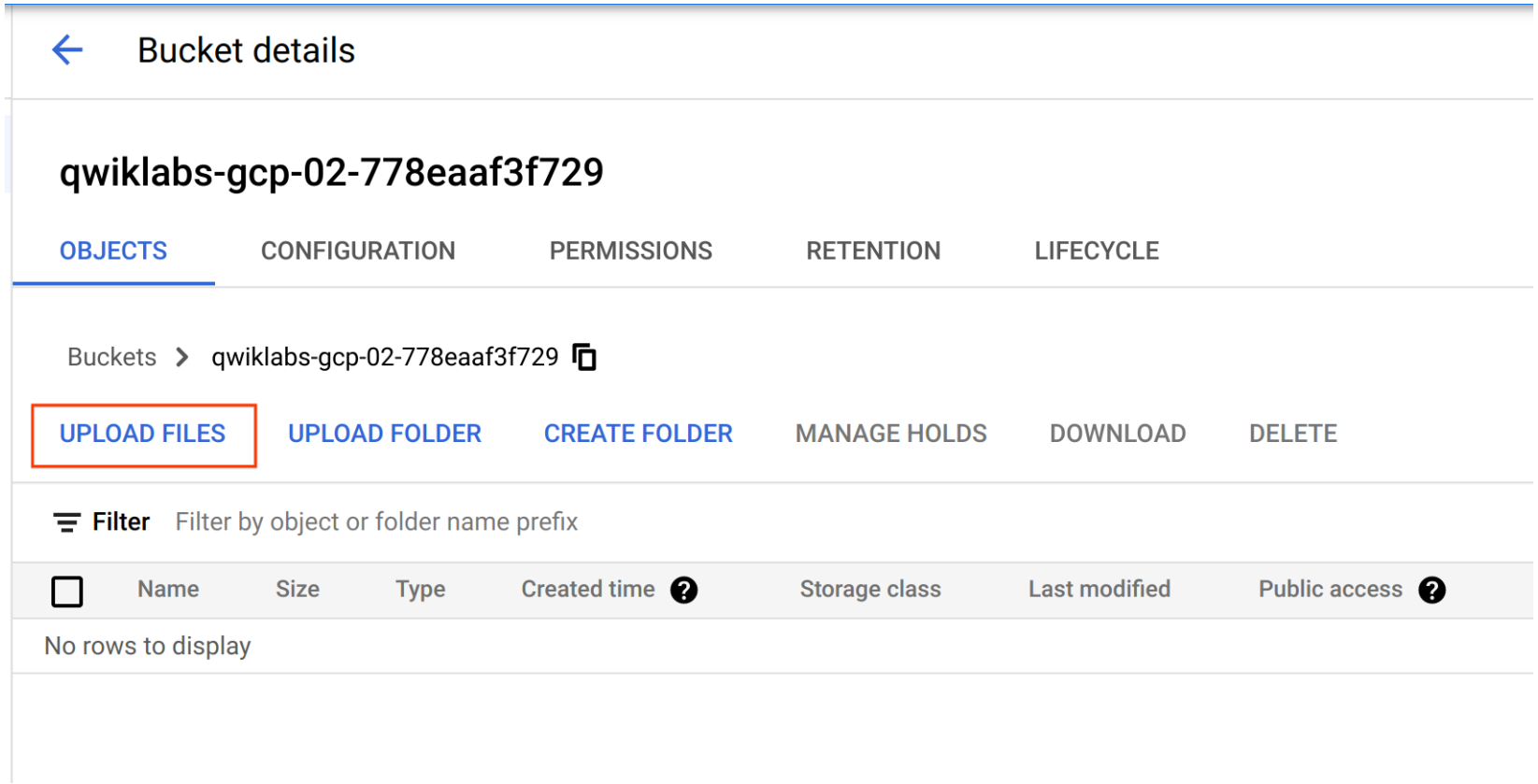
  CONTINUE

4. All other settings for your bucket can remain as the default setting. Click **CREATE**.

Upload an image to your bucket

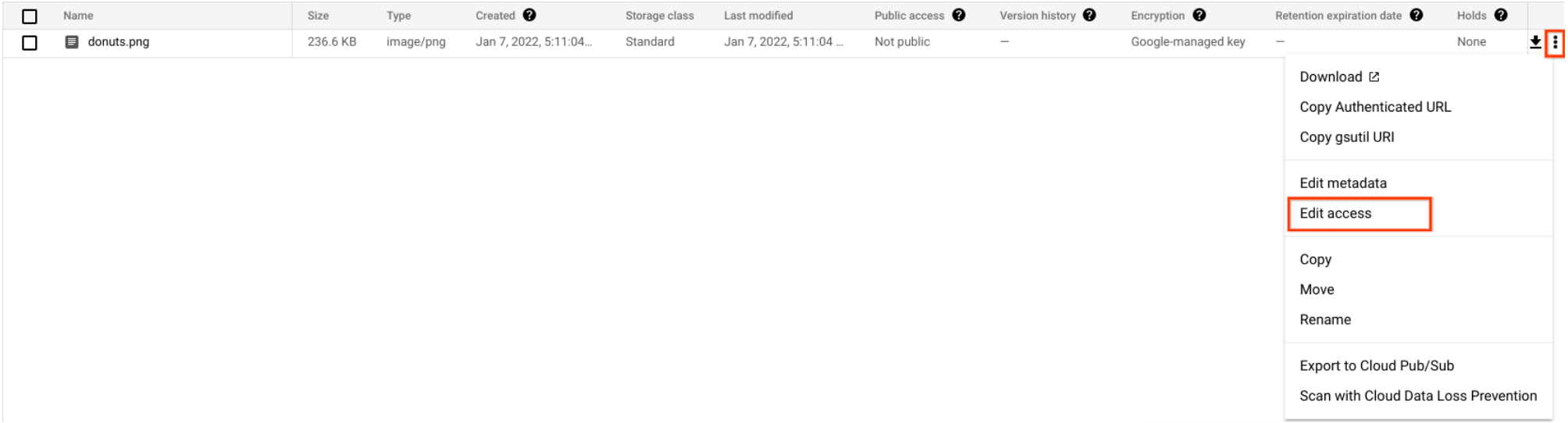
1. Right click on the following image of donuts, then click **Save image as** and save it to your computer as **donuts.png**.

2. Go to the bucket you just created and click **UPLOAD FILES**. Then select **donuts.png**.



You should see the file in your bucket.

3. Now you need to make this image publicly available. Click on the 3 dots for your image and select **Edit access**.

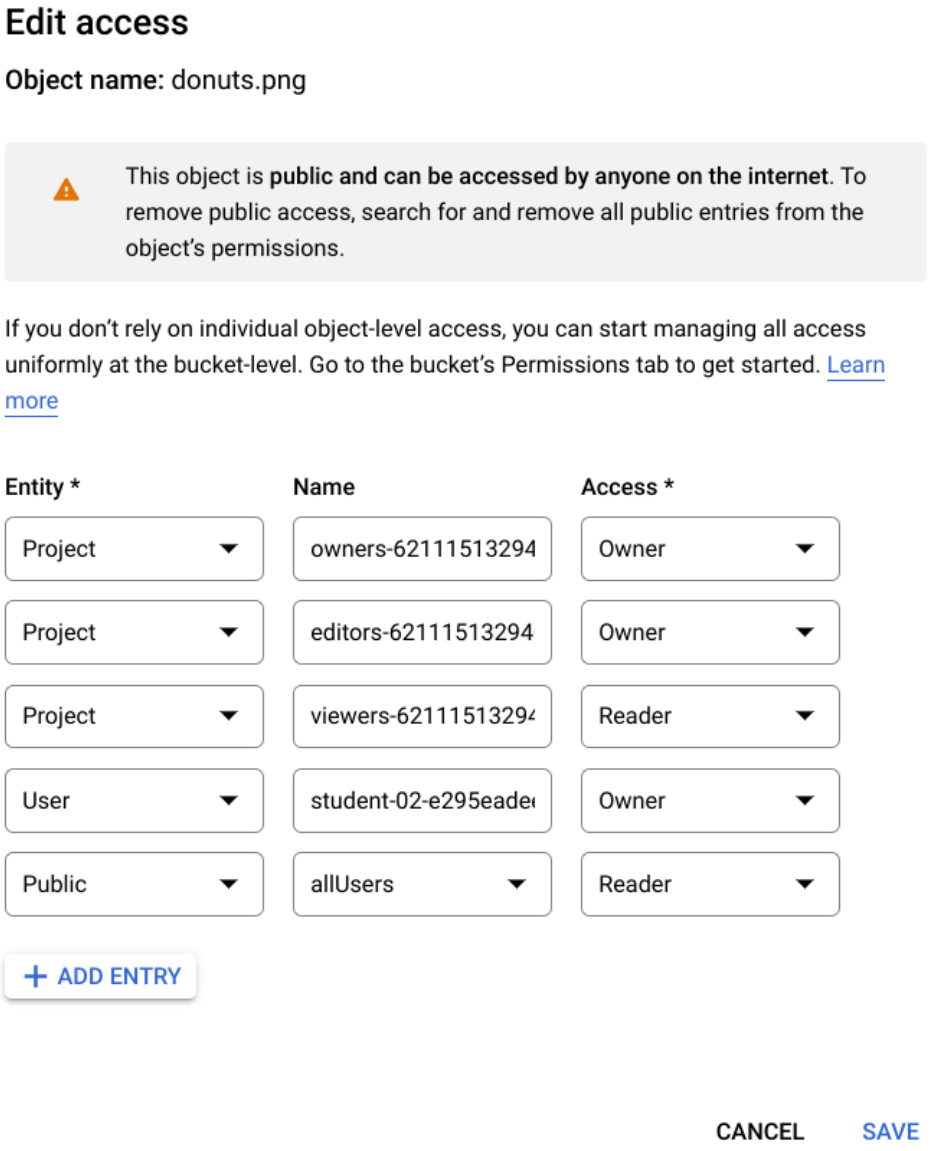


4. Click **Add entry** then enter the following:

Entity: Public

Name: allUsers

Access: Reader



5. Then click **SAVE**.

Now that you have the file in your bucket, you're ready to create a Vision API request, passing it the URL of this donuts picture.

Click **Check my progress** to verify the objective. Upload an Image to a Cloud Storage bucket

Task 3. Create your Vision API request

Now you create a `request.json` file in the Cloud Shell environment.

Create a JSON request file for submitting the base64 encoded form for processing:

Note

Replace `my-bucket-name` with the name of your storage bucket.

echo ' { "requests": [{ "image": { "source": {"gcsImageUri":"gs://qwiklabs-gcp-03-8xxxxxx3/donuts.png" } }, "features": [{ "type": "LABEL_DETECTION", "maxResults": 10 }] }] } ' > request.json

Task 4. Label Detection

The first Cloud Vision API feature you try out is label detection. This method returns a list of labels (words) of what's in your image.

Call the Vision API with `curl`:

curl -s -X POST -H "Content-Type: application/json" --data-binary @request.json https://vision.googleapis.com/v1/images:annotate?key={AIzxxxxxx5yq4YRmLlxxxxxxxxxxxxxxxxxmU}

Your response should look something like the following:

{ "responses": [{ "labelAnnotations": [{ "mid": "/m/01dk8s", "description": "Powdered sugar", "score": 0.9861496, "topicality": 0.9861496 }, { "mid": "/m/01wydv", "description": "Beignet", "score": 0.9565117, "topicality": 0.9565117 }, { "mid": "/m/02wbm", "description": "Food", "score": 0.9424965, "topicality": 0.9424965 }, { "mid": "/m/0hnyx", "description": "Pastry", "score": 0.8173416, "topicality": 0.8173416 }, { "mid": "/m/02q08p0", "description": "Dish", "score": 0.8076026, "topicality": 0.8076026 }, { "mid": "/m/01ykh", "description": "Cuisine", "score": 0.79036003, "topicality": 0.79036003 }, { "mid": "/m/03nsjgy", "description": "Kourabiedes", "score": 0.77726763, "topicality": 0.77726763 }, { "mid": "/m/06gd3r", "description": "Angel wings", "score": 0.73792106, "topicality": 0.73792106 }, { "mid": "/m/06x4c", "description": "Sugar", "score": 0.71921736, "topicality": 0.71921736 }, { "mid": "/m/01zl9v", "description": "Zeppole", "score": 0.7111677, "topicality": 0.7111677 }] }] }

The API was able to identify the specific type of donuts these are, powdered sugar. Cool! For each label the Vision API found, it returns a:

- `description` with the name of the item.
- `score`, a number from 0 - 1 indicating how confident it is that the description matches what's in the image.
- `mid` value that maps to the item's `mid` in Google's Knowledge Graph. You can use the `mid` when calling the Knowledge Graph API to get more information on the item.

Task 5. Web Detection

In addition to getting labels on what's in your image, the Vision API can also search the Internet for additional details on your image. Through the API's webDetection method, you get a lot of interesting data back:

- A list of entities found in your image, based on content from pages with similar images
- URLs of exact and partial matching images found across the web, along with the URLs of those pages
- URLs of similar images, like doing a reverse image search

To try out web detection, use the same image of beignets and change one line in the `request.json` file (you can also venture out into the unknown and use an entirely different image).

1. Under the features list, change **type** from `LABEL_DETECTION` to `WEB_DETECTION`. The `request.json` should now look like this:

{ "requests": [{ "image": { "source": { "gcsImageUri": "gs://qwiklabs-gcp-04-axxxxxxxxxxx0-bucket/donuts.png" } }, "features": [{ "type": "WEB_DETECTION", "maxResults": 10 }] }] }

Save the file.

2. To send it to the Vision API, use the same `curl` command as before (just press the up arrow in Cloud Shell):

```
curl -s -X POST -H "Content-Type: application/json" --data-binary @request.json
https://vision.googleapis.com/v1/images:annotate?key={AIzaSxxxxxxxxxWB00MgbiQJ4m}
```

Dive into the response, starting with `webEntities`. Here are some of the entities this image returned:

{ "responses": [{ "webDetection": { "webEntities": [{ "entityId": "/m/0z5n", "score": 0.8868, "description": "API" }, { "entityId": "/m/07kg1sq", "score": 0.3139, "description": "Computer Vision" }, { "entityId": "/m/0105pbj4", "score": 0.2713, "description": "Cloud Vision" }, { "entityId": "/m/01hyh_", "score": 0.2594, "description": "Machine learning" }, ...]

This image has been used in many presentations on Cloud ML APIs, which is why the API found the entities "Machine learning" and "Cloud Vision".

If you inpsect the URLs under `fullMatchingImages`, `partialMatchingImages`, and `pagesWithMatchingImages`, you notice that many of the URLs point to this lab site (super meta!).

Say you wanted to find other images of beignets, but not the exact same images. That's where the `visuallySimilarImages` part of the API response comes in handy. Here are a few of the visually similar images it found:

"visuallySimilarImages": [{ "url": "https://media.istockphoto.com/photos/cafe-du-monde-picture-id1063530570?k=6&m=1063530570&s=612x612&w=0&h=b74EYAjlfxMw8G-G_6BW-6ltP9Y2UFQ3TjZopN-pigI=" }, { "url": "https://s3-media2.fl.yelpcdn.com/bphoto/oid0KchdCqlSqZzpznCEoA/o.jpg" }, { "url": "https://s3-media1.fl.yelpcdn.com/bphoto/mgAhrlLFvXe0IkT5UMOUlw/348s.jpg" }, ...]

You can navigate to those URLs to see the similar images:

And now you probably really want a powdered sugar beignet (sorry)! This is similar to searching by an image on Google Images.

With Cloud Vision you can access this functionality with an easy to use REST API and integrate it into your applications.

Task 6. Face Detection

Next explore the face detection methods of the Vision API.

- The face detection method returns data on faces found in an image, including the emotions of the faces and their location in the image.

Upload a new image

To use this method, you upload a new image with faces to the Cloud Storage bucket.

1. Right click on the following image, then click **Save image as** and save it to your computer as **selfie.png**.


ATM

2. Now upload it to your Cloud Storage bucket the same way you did before, and make it public.

Click **Check my progress** to verify the objective. Upload an image for Face Detection to your bucket

Update request file

1. Next, update your `request.json` file with the following, which includes the URL of the new image, and uses face and landmark detection instead of label detection. Be sure to replace **my-bucket-name** with the name of your Cloud Storage bucket:

{ "requests": [{ "image": { "source": { "gcsImageUri": "gs://my-bucket-name/selfie.png" } }, "features": [{ "type": "FACE_DETECTION" }, { "type": "LANDMARK_DETECTION" }] }] }

2. **Save** the file.

Task 7. Call the Vision API and parse the response

Now you're ready to call the Vision API using the same `curl` command you used above:

curl -s -X POST -H "Content-Type: application/json" --data-binary @request.json https://vision.googleapis.com/v1/images:annotate?key=${API_KEY}

Take a look at the `faceAnnotations` object in the response. You notice the API returns an object for each face found in the image - in this case, three. Here's a clipped version of the response:

{ "faceAnnotations": [{ "boundingPoly": { "vertices": [{ "x": 669, "y": 324 }, ...] }, "fdBoundingPoly": { ... }, "landmarks": [{ "type": "LEFT_EYE", "position": { "x": 692.05646, "y": 372.95868, "z": -0.00025268539 } }, ...], "rollAngle": 0.21619819, "panAngle": -23.027969, "tiltAngle": -1.5531756, "detectionConfidence": 0.72354823, "landmarkingConfidence": 0.20047489, "joyLikelihood": "LIKELY", "sorrowLikelihood": "VERY_UNLIKELY", "angerLikelihood": "VERY_UNLIKELY", "surpriseLikelihood": "VERY_UNLIKELY", "underExposedLikelihood": "VERY_UNLIKELY", "blurredLikelihood": "VERY_UNLIKELY", "headwearLikelihood": "VERY_LIKELY" } ... } }

- `boundingPoly` gives you the x,y coordinates around the face in the image.
- `fdBoundingPoly` is a smaller box than `boundingPoly`, focusing on the skin part of the face.
- `landmarks` is an array of objects for each facial feature, some you may not have even known about. This tells us the type of landmark, along with the 3D position of that feature (x,y,z coordinates) where the z coordinate is the depth. The remaining values gives you more details on the face, including the likelihood of joy, sorrow, anger, and surprise.

The response you're reading is for the person standing furthest back in the image - you can see he's making kind of a silly face which explains the `joyLikelihood` of `LIKELY`.

Task 8. Landmark Annotation

- Landmark detection can identify common (and obscure) landmarks. It returns the name of the landmark, its latitude and longitude coordinates, and the location of where the landmark was identified in an image.

Upload a new image

To use this method, you upload a new image with faces to the Cloud Storage bucket.

1. Right click on the following image, then click **Save image as** and save it to your computer as **city.png**.

2. Now upload it to your Cloud Storage bucket the same way you did before, and make it public.

Click **Check my progress** to verify the objective. Upload an image for Landmark Annotation to your bucket

Update request file

1. Next, update your `request.json` file with the following, which includes the URL of the new image, and uses landmark detection. Be sure to replace **my-bucket-name** with the name of your Cloud Storage bucket:

{ "requests": [{ "image": { "source": { "gcsImageUri": "gs://my-bucket-name/city.png" } }, "features": [{ "type": "LANDMARK_DETECTION", "maxResults": 10 }] }] }

2. **Save** the file.

Task 9. Call the Vision API and parse the response

Now you're ready to call the Vision API using the same `curl` command you used above:

curl -s -X POST -H "Content-Type: application/json" --data-binary @request.json https://vision.googleapis.com/v1/images:annotate?key=${API_KEY}

Next, look at the `landmarkAnnotations` part of the response:

"landmarkAnnotations": [{ "mid": "/m/041cp3", "description": "Boston", "score": 0.788803, "boundingPoly": { "vertices": [{ "y": 576 }, { "x": 1942, "y": 576 }, { "x": 1942, "y": 1224 }, { "y": 1224 }] }, ...

Here, the Vision API was able to tell that this picture was taken in Boston, and gives you a map of the exact location. The values in this response should look similar to the `labelAnnotations` response above:

- the `mid` of the landmark
- it's name (`description`)
- a confidence `score`
- The `boundingPoly` shows the region in the image where the landmark was identified.
- The `locations` key tells us the latitude longitude coordinates of the picture.

Task 10. Explore other Vision API methods

You've looked at the Vision API's label, face, and landmark detection methods, but there are three others you haven't explored. Dive into the docs to learn about the other three:

- **Logo detection**: identify common logos and their location in an image.
- **Safe search detection**: determine whether or not an image contains explicit content. This is useful for any application with user-generated content. You can filter images based on four factors: adult, medical, violent, and spoof content.
- **Text detection**: run OCR to extract text from images. This method can even identify the language of text present in an image.

Congratulations!

You've learned how to analyze images with the Vision API. In this example you passed the API the Cloud Storage URL of your image. Alternatively, you can pass a base64 encoded string of your image.

End your lab

When you have completed your lab, click **End Lab**. Qwiklabs removes the resources you've used and cleans the account for you.

You will be given an opportunity to rate the lab experience. Select the applicable number of stars, type a comment, and then click **Submit**.

The number of stars indicates the following:

- 1 star = Very dissatisfied
- 2 stars = Dissatisfied
- 3 stars = Neutral
- 4 stars = Satisfied
- 5 stars = Very satisfied

You can close the dialog box if you don't want to provide feedback.

For feedback, suggestions, or corrections, please use the **Support** tab.