Quickstart: Create an object detection project with the Custom Vision client library

Article • 07/18/2023

Get started with the Custom Vision client library for Python. Follow these steps to install the package and try out the example code for building an object detection model. You'll create a project, add tags, train the project, and use the project's prediction endpoint URL to programmatically test it. Use this example as a template for building your own image recognition app.

Note

If you want to build and train an object detection model *without* writing code, see the **browser-based guidance** instead.

Use the Custom Vision client library for Python to:

- Create a new Custom Vision project
- Add tags to the project
- Upload and tag images
- Train the project
- Publish the current iteration
- Test the prediction endpoint

Reference documentation | Library source code | Package (PyPI) | Samples

Prerequisites

- Azure subscription - Create one for free
- Python 3.x
  - Your Python installation should include pip . You can check if you have pip installed by running `pip --version` on the command line. Get pip by installing the latest version of Python.

- Once you have your Azure subscription, create a Custom Vision resource in the Azure portal to create a training and prediction resource.
    - You can use the free pricing tier (`F0`) to try the service, and upgrade later to a paid tier for production.

Create environment variables

In this example, you'll write your credentials to environment variables on the local machine running the application.

Go to the Azure portal. If the Custom Vision resources you created in the **Prerequisites** section deployed successfully, select the **Go to Resource** button under **Next Steps**. You can find your keys and endpoints in the resources**' key and endpoint** pages, under **resource management**. You'll need to get the keys for both your training and prediction resources, along with the API endpoints.

You can find the prediction resource ID on the prediction resource's **Properties** tab in the Azure portal, listed as **Resource ID**.

Tip

You also use **https://www.customvision.ai/** to get these values. After you sign in, select the **Settings** icon at the top right. On the **Setting** pages, you can view all the keys, resource ID, and endpoints.

Caution

Don't include the key directly in your code, and never post it publicly. See the Azure AI services **security** article for more authentication options like **Azure Key Vault**.

To set the environment variables, open a console window and follow the instructions for your operating system and development environment.

1. To set the `VISION_TRAINING KEY` environment variable, replace `your-training-key` with one of the keys for your training resource.
2. To set the `VISION_TRAINING_ENDPOINT` environment variable, replace `your-training-endpoint` with the endpoint for your training resource.

3. To set the `VISION_PREDICTION_KEY` environment variable, replace `your-prediction-key` with one of the keys for your prediction resource.
4. To set the `VISION_PREDICTION_ENDPOINT` environment variable, replace `your-prediction-endpoint` with the endpoint for your prediction resource.
5. To set the `VISION_PREDICTION_RESOURCE_ID` environment variable, replace `your-resource-id` with the resource ID for your prediction resource.

Windows

Console

```
setx VISION_TRAINING_KEY your-training-key
```

Console

```
setx VISION_TRAINING_ENDPOINT your-training-endpoint
```

Console

```
setx VISION_PREDICTION_KEY your-prediction-key
```

Console

```
setx VISION_PREDICTION_ENDPOINT your-prediction-endpoint
```

Console

```
setx VISION_PREDICTION_RESOURCE_ID your-resource-id
```

After you add the environment variables, you may need to restart any running programs that will read the environment variables, including the console window.

Setting up

Install the client library

To write an image analysis app with Custom Vision for Python, you'll need the Custom Vision client library. After installing Python, run the following command in PowerShell or a console window:

PowerShell

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

Create a new Python application

Create a new Python file and import the following libraries.

Python

```
from azure.cognitiveservices.vision.customvision.training import
CustomVisionTrainingClient
from azure.cognitiveservices.vision.customvision.prediction import
CustomVisionPredictionClient
from azure.cognitiveservices.vision.customvision.training.models import
ImageFileCreateBatch, ImageFileCreateEntry, Region
from msrest.authentication import ApiKeyCredentials
import os, time, uuid
```

Tip

Want to view the whole quickstart code file at once? You can find it on **GitHub** , which contains the code examples in this quickstart.

Create variables for your resource's Azure endpoint and keys.

Python

```
# Replace with valid values
ENDPOINT = os.environ["VISION_TRAINING_ENDPOINT"]
training_key = os.environ["VISION_TRAINING_KEY"]
prediction_key = os.environ["VISION_PREDICTION_KEY"]
prediction_resource_id = os.environ["VISION_PREDICTION_RESOURCE_ID"]
```

Object model

Expand table

Name	Description
CustomVisionTrainingClient	This class handles the creation, training, and publishing of your models.
CustomVisionPredictionClient	This class handles the querying of your models for object detection predictions.
ImagePrediction	This class defines a single object prediction on a single image. It includes properties for the object ID and name, the bounding box location of the object, and a confidence score.

Code examples

These code snippets show you how to do the following with the Custom Vision client library for Python:

- Authenticate the client
- Create a new Custom Vision project
- Add tags to the project
- Upload and tag images
- Train the project
- Publish the current iteration
- Test the prediction endpoint

Authenticate the client

Instantiate a training and prediction client with your endpoint and keys. Create **ApiKeyServiceClientCredentials** objects with your keys, and use them with your endpoint to create a CustomVisionTrainingClient and CustomVisionPredictionClient object.

Python

```python
credentials = ApiKeyCredentials(in_headers={"Training-key": training_key})
trainer = CustomVisionTrainingClient(ENDPOINT, credentials)
prediction_credentials = ApiKeyCredentials(in_headers={"Prediction-key":
prediction_key})
predictor = CustomVisionPredictionClient(ENDPOINT, prediction_credentials)
```

Create a new Custom Vision project

Add the following code to your script to create a new Custom Vision service project.

See the create_project method to specify other options when you create your project (explained in the Build a detector web portal guide).

Python

```python
publish_iteration_name = "detectModel"

# Find the object detection domain
obj_detection_domain = next(domain for domain in trainer.get_domains() if
domain.type == "ObjectDetection" and domain.name == "General")

# Create a new project
print ("Creating project...")
# Use uuid to avoid project name collisions.
project = trainer.create_project(str(uuid.uuid4()),
domain_id=obj_detection_domain.id)
```

Add tags to the project

To create object tags in your project, add the following code:

Python

```python
# Make two tags in the new project
fork_tag = trainer.create_tag(project.id, "fork")
scissors_tag = trainer.create_tag(project.id, "scissors")
```

Upload and tag images

First, download the sample images for this project. Save the contents of the sample Images folder to your local device.

When you tag images in object detection projects, you need to specify the region of each tagged object using normalized coordinates. The following code associates each of the sample images with its tagged region. The regions specify the bounding box in normalized coordinates, and the coordinates are given in the order: left, top, width, height.

Python

```python
fork_image_regions = {
    "fork_1": [ 0.145833328, 0.3509314, 0.5894608, 0.238562092 ],
    "fork_2": [ 0.294117659, 0.216944471, 0.534313738, 0.5980392 ],
    "fork_3": [ 0.09191177, 0.0682516545, 0.757352948, 0.6143791 ],
    "fork_4": [ 0.254901975, 0.185898721, 0.5232843, 0.594771266 ],
    "fork_5": [ 0.2365196, 0.128709182, 0.5845588, 0.71405226 ],
    "fork_6": [ 0.115196079, 0.133611143, 0.676470637, 0.6993464 ],
    "fork_7": [ 0.164215669, 0.31008172, 0.767156839, 0.410130739 ],
    "fork_8": [ 0.118872553, 0.318251669, 0.817401946, 0.225490168 ],
    "fork_9": [ 0.18259804, 0.2136765, 0.6335784, 0.643790841 ],
    "fork_10": [ 0.05269608, 0.282303959, 0.8088235, 0.452614367 ],
    "fork_11": [ 0.05759804, 0.0894935, 0.9007353, 0.3251634 ],
    "fork_12": [ 0.3345588, 0.07315363, 0.375, 0.9150327 ],
    "fork_13": [ 0.269607842, 0.194068655, 0.4093137, 0.6732026 ],
    "fork_14": [ 0.143382356, 0.218578458, 0.7977941, 0.295751631 ],
    "fork_15": [ 0.19240196, 0.0633497, 0.5710784, 0.8398692 ],
    "fork_16": [ 0.140931368, 0.480016381, 0.6838235, 0.240196079 ],
    "fork_17": [ 0.305147052, 0.2512582, 0.4791667, 0.5408496 ],
    "fork_18": [ 0.234068632, 0.445702642, 0.6127451, 0.344771236 ],
    "fork_19": [ 0.219362751, 0.141781077, 0.5919118, 0.6683006 ],
    "fork_20": [ 0.180147052, 0.239820287, 0.6887255, 0.235294119 ]
}

scissors_image_regions = {
    "scissors_1": [ 0.4007353, 0.194068655, 0.259803921, 0.6617647 ],
    "scissors_2": [ 0.426470578, 0.185898721, 0.172794119, 0.5539216 ],
    "scissors_3": [ 0.289215684, 0.259428144, 0.403186262, 0.421568632 ],
    "scissors_4": [ 0.343137264, 0.105833367, 0.332107842, 0.8055556 ],
    "scissors_5": [ 0.3125, 0.09766343, 0.435049027, 0.71405226 ],
    "scissors_6": [ 0.379901975, 0.24308826, 0.32107842, 0.5718954 ],
    "scissors_7": [ 0.341911763, 0.20714055, 0.3137255, 0.6356209 ],
    "scissors_8": [ 0.231617644, 0.08459154, 0.504901946, 0.8480392 ],
    "scissors_9": [ 0.170343131, 0.332957536, 0.767156839, 0.403594762 ],
    "scissors_10": [ 0.204656869, 0.120539248, 0.5245098, 0.743464053 ],
    "scissors_11": [ 0.05514706, 0.159754932, 0.799019635, 0.730392158 ],
    "scissors_12": [ 0.265931368, 0.169558853, 0.5061275, 0.606209159 ],
    "scissors_13": [ 0.241421565, 0.184264734, 0.448529422, 0.6830065 ],
    "scissors_14": [ 0.05759804, 0.05027781, 0.75, 0.882352948 ],
    "scissors_15": [ 0.191176474, 0.169558853, 0.6936275, 0.6748366 ],
    "scissors_16": [ 0.1004902, 0.279036, 0.6911765, 0.477124184 ],
    "scissors_17": [ 0.2720588, 0.131977156, 0.4987745, 0.6911765 ],
    "scissors_18": [ 0.180147052, 0.112369314, 0.6262255, 0.6666667 ],
    "scissors_19": [ 0.333333343, 0.0274019931, 0.443627447, 0.852941155 ],
    "scissors_20": [ 0.158088237, 0.04047389, 0.6691176, 0.843137264 ]
}
```

ⓘ **Note**

If you don't have a click-and-drag utility to mark the coordinates of regions, you can use the web UI at **Customvision.ai** . In this example, the coordinates are already provided.

Then, use this map of associations to upload each sample image with its region coordinates (you can upload up to 64 images in a single batch). Add the following code.

Python

```python
base_image_location = os.path.join (os.path.dirname(__file__), "Images")

# Go through the data table above and create the images
print ("Adding images...")
tagged_images_with_regions = []

for file_name in fork_image_regions.keys():
    x,y,w,h = fork_image_regions[file_name]
    regions = [ Region(tag_id=fork_tag.id, left=x,top=y,width=w,height=h) ]

    with open(os.path.join (base_image_location, "fork", file_name + ".jpg"),
mode="rb") as image_contents:
        tagged_images_with_regions.append(ImageFileCreateEntry(name=file_name,
contents=image_contents.read(), regions=regions))

for file_name in scissors_image_regions.keys():
    x,y,w,h = scissors_image_regions[file_name]
    regions = [ Region(tag_id=scissors_tag.id, left=x,top=y,width=w,height=h) ]

    with open(os.path.join (base_image_location, "scissors", file_name +
".jpg"), mode="rb") as image_contents:
        tagged_images_with_regions.append(ImageFileCreateEntry(name=file_name,
contents=image_contents.read(), regions=regions))

upload_result = trainer.create_images_from_files(project.id,
ImageFileCreateBatch(images=tagged_images_with_regions))
if not upload_result.is_batch_successful:
    print("Image batch upload failed.")
    for image in upload_result.images:
        print("Image status: ", image.status)
    exit(-1)
```

Note

You'll need to change the path to the images based on where you downloaded the Azure AI services Python SDK Samples repo earlier.

Train the project

This code creates the first iteration of the prediction model.

Python

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

Tip

Train with selected tags

You can optionally train on only a subset of your applied tags. You may want to do this if you haven't applied enough of certain tags yet, but you do have enough of others. In the **train_project** call, set the optional parameter *selected_tags* to a list of the ID strings of the tags you want to use. The model will train to only recognize the tags on that list.

Publish the current iteration

An iteration is not available in the prediction endpoint until it is published. The following code makes the current iteration of the model available for querying.

Python

```python
# The iteration is now trained. Publish it to the project endpoint
trainer.publish_iteration(project.id, iteration.id, publish_iteration_name,
prediction_resource_id)
print ("Done!")
```

Test the prediction endpoint

To send an image to the prediction endpoint and retrieve the prediction, add the following code to the end of the file:

Python

```python
# Now there is a trained endpoint that can be used to make a prediction

# Open the sample image and get back the prediction results.
with open(os.path.join (base_image_location, "test", "test_image.jpg"),
mode="rb") as test_data:
    results = predictor.detect_image(project.id, publish_iteration_name,
test_data)

# Display the results.
for prediction in results.predictions:
    print("\t" + prediction.tag_name + ": {0:.2f}% bbox.left = {1:.2f},
bbox.top = {2:.2f}, bbox.width = {3:.2f}, bbox.height =
{4:.2f}".format(prediction.probability * 100, prediction.bounding_box.left,
prediction.bounding_box.top, prediction.bounding_box.width,
prediction.bounding_box.height))
```

Run the application

Run *CustomVisionQuickstart.py*.

PowerShell

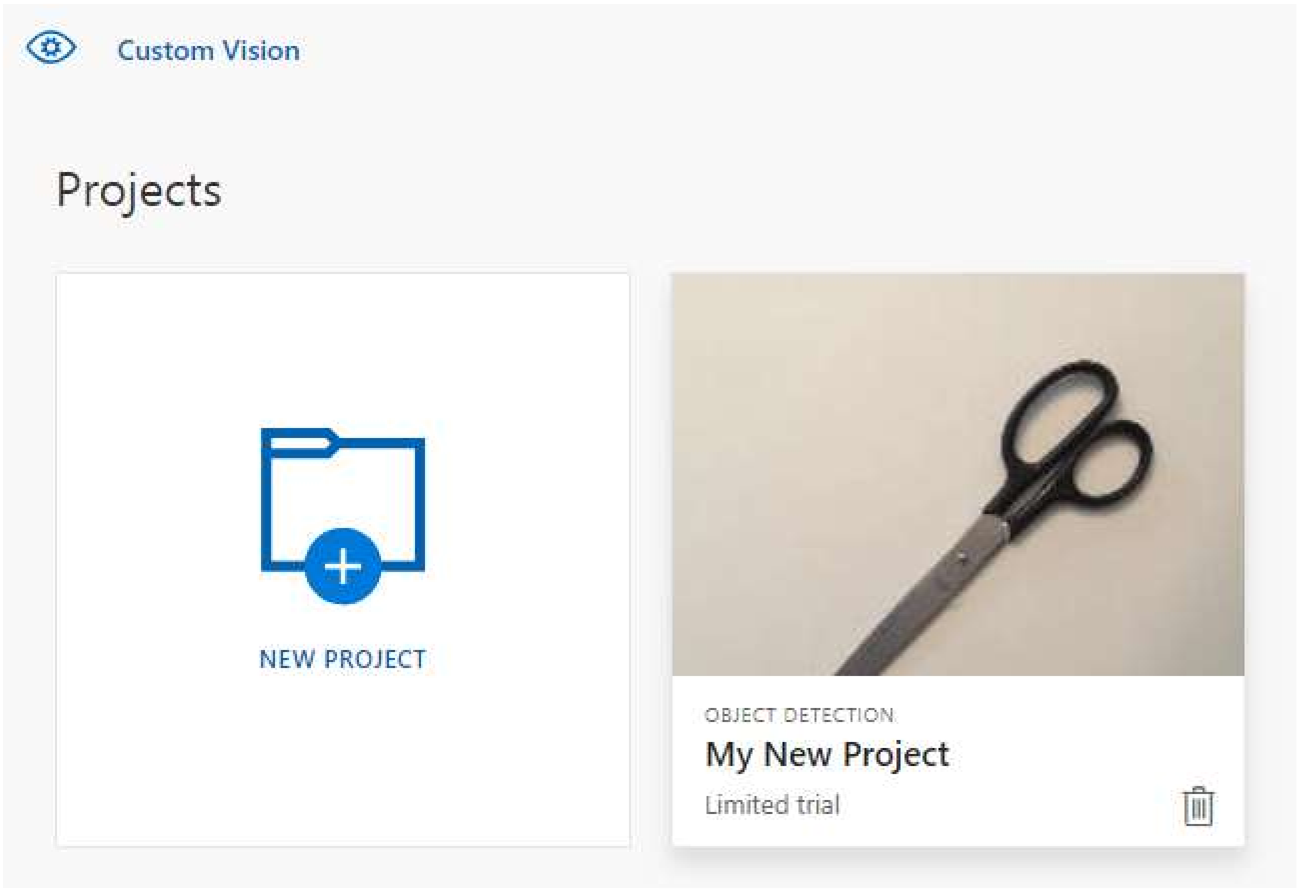
```powershell
python CustomVisionQuickstart.py
```

The output of the application should appear in the console. You can then verify that the test image (found in **<base_image_location>/images/Test**) is tagged appropriately and that the region of detection is correct. You can also go back to the Custom Vision website and see the current state of your newly created project.

Clean up resources

If you wish to implement your own object detection project (or try an image classification project instead), you may want to delete the fork/scissors detection project from this example. A free subscription allows for two Custom Vision projects.

On the Custom Vision website, navigate to **Projects** and select the trash can under My New Project.



Next steps

Now you've done every step of the object detection process in code. This sample executes a single training iteration, but often you'll need to train and test your model multiple times in order to make it more accurate. The following guide deals with image classification, but its principles are similar to object detection.

Test and retrain a model

- What is Custom Vision?
- The source code for this sample can be found on GitHub
- SDK reference documentation