2.4 - Detect objects in images

Overview

Object detection is used to locate and identify objects in images. You can use **Azure AI Custom Vision** to train a model to detect specific classes of object in images.

After completing this module, you will be able to:

- Provision Azure resources for Azure AI Custom Vision
- Understand object detection
- Train an object detector
- Consider options for labeling images

Introduction

*Object detection* is a common computer vision problem that requires software to identify the location of specific classes of object in an image. In this module, you will learn how to use the **Azure AI Custom Vision** service to create object detection models.

After completing this module, you'll be able to:

- Describe object detection.
- Train an object detector.
- Describe options for labeling images.

Understand object detection

Object detection is a form of computer vision in which a model is trained to detect the presence and location of one or more classes of object in an image. For example, an AI-enabled checkout system in a grocery store might need to identify the type and location of items being purchased by a customer.
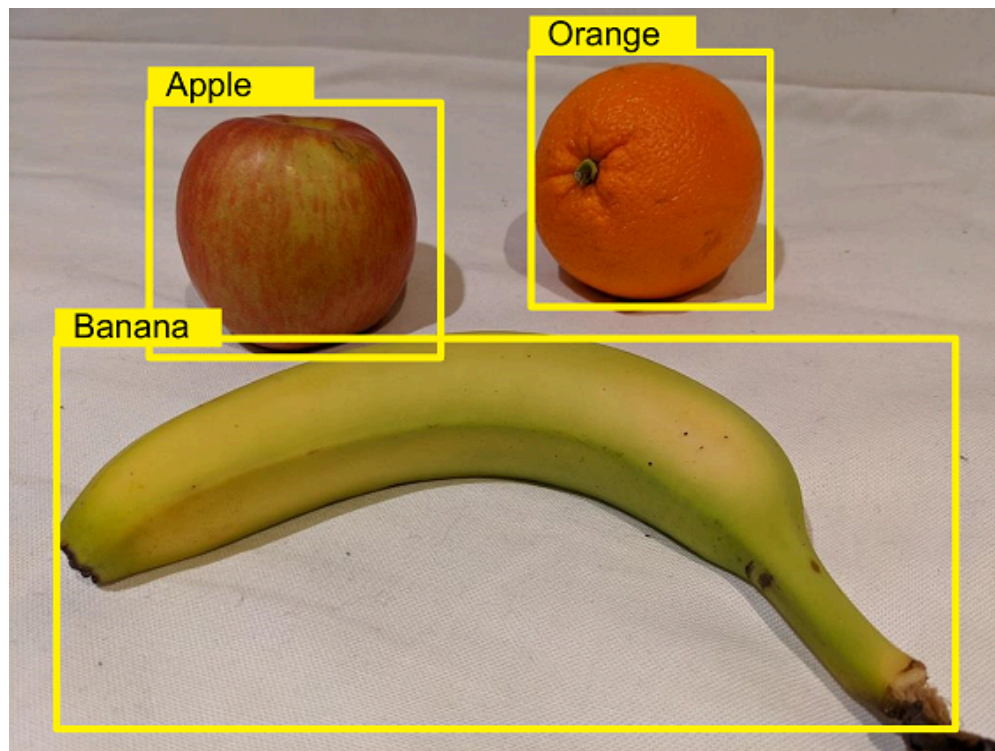
There are two components to an object detection prediction:

- The class label of each object detected in the image. For example, you might ascertain that an image contains one apple and two oranges.
- The location of each object within the image, indicated as coordinates of a *bounding box* that encloses the object.

Use the Azure AI Custom Vision service for object detection

You can use the **Azure AI Custom Vision** service to train an object detection model. To use the Azure AI Custom Vision service, you must provision two kinds of Azure resource:

- A *training* resource used to train your models. This can be:
  - An **Azure AI services multi-service** resource.
  - An **Azure AI Custom Vision (Training)** resource.
- A *prediction* resource, used by client applications to get predictions from your model. This can be:
  - An **Azure AI services multi-service** resource.
  - An **Azure AI Custom Vision (Prediction)** resource.

You can use a **Azure AI services multi-service** resource for both training and prediction, and you can mix-and-match resource types (for example, using an **Azure AI Custom Vision (Training)** resource to train a model that you then publish using an **Azure AI services multi-service** resource). If using a multi-service resource, they key and endpoint for both training and prediction will be the same.

Train an object detector

To train an object detection model, you can use the Azure AI Custom Vision portal to upload and label images before training, evaluating, testing, and publishing the model; or you can use the REST API or SDK to write code that performs the training tasks.

The most significant difference between training an *image classification* model and training an *object detection* model is the labeling of the images with tags.

While image classification requires one or more tags that apply to the whole image, object detection requires that each label consists of a tag and a *region* that defines the bounding box for each object in an image. The Azure AI Custom Vision portal provides a graphical interface that you can use to label your training images.
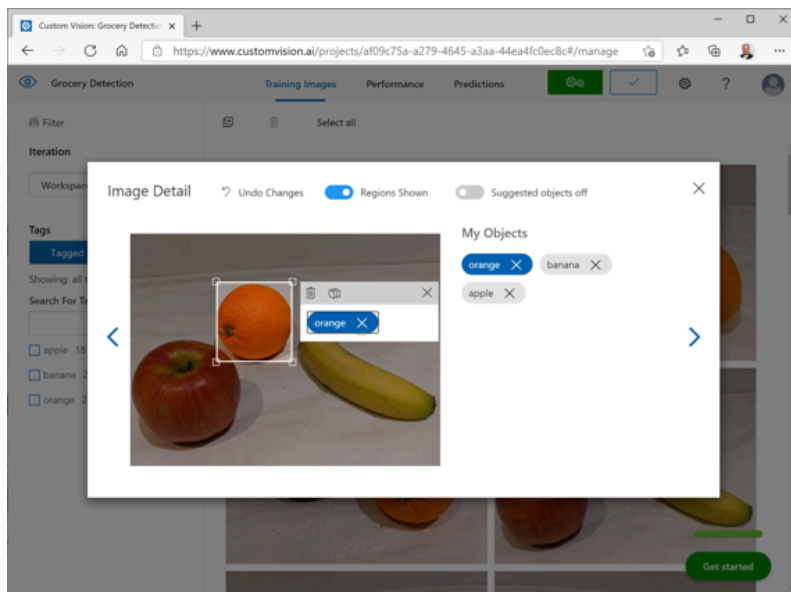
Consider options for labeling images

The easiest option for labeling images for object detection is to use the **interactive interface in the Azure AI Custom Vision portal.**

This interface **automatically suggests regions that contain objects**, to which you can assign tags or adjust by dragging the bounding box to enclose the object you want to label.

Additionally, after tagging an initial batch of images, you can train the model. Subsequent labeling of new images can benefit from the *smart labeler* tool in the portal, which can suggest not only the regions, but the classes of object they contain.

Alternatively, you can use a labeling tool, such as the one provided in Azure Machine Learning Studio or the Microsoft Visual Object Tagging Tool (VOTT), to take advantage of other features, such as assigning image labeling tasks to multiple team members.

Bounding box measurement units

If you choose to use a labeling tool other than the Azure AI Custom Vision portal, you may need to adjust the output to match the measurement units expected by the Azure AI Custom Vision API.

Bounding boxes are defined by four values that represent the left (X) and top (Y) coordinates of the **top-left corner of the bounding box**, and the width and height of the bounding box.

These values are expressed as *proportional* values relative to the source image size. For example, consider this bounding box:

- Left: 0.1
- Top: 0.5
- Width: 0.5
- Height: 0.25

This defines a box in which the left is located 0.1 (one tenth) from the left edge of the image, and the top is 0.5 (half the image height) from the top. **The box is half the width and a quarter of the height of the overall image.**

The following image shows labeling information in JSON format for objects in an image.
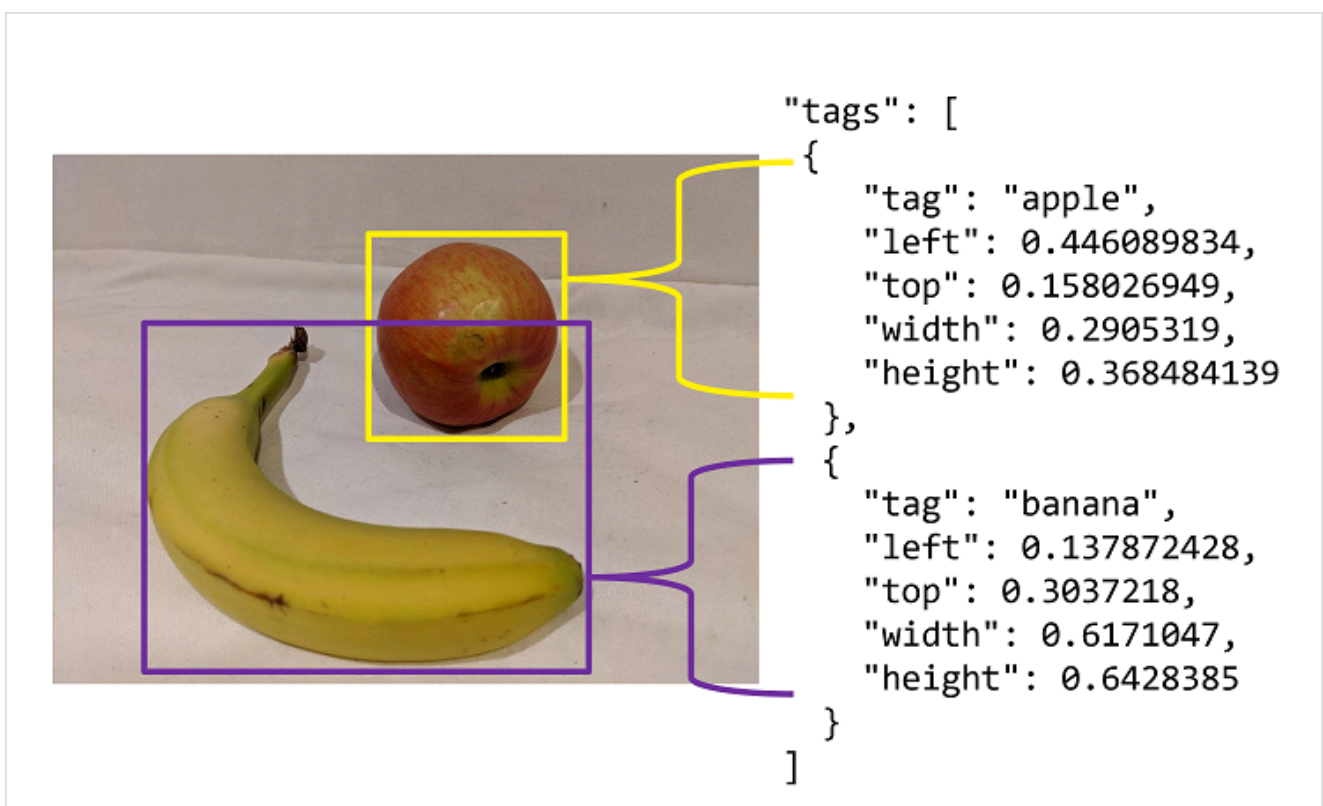
Exercise - Detect objects in images with Azure AI Custom Vision

Create Custom Vision resources

If you already have **Custom Vision** resources for training and prediction in your Azure subscription, you can use them or an existing multi-service account in this exercise. If not, use the following instructions to create them.

> **Note**: If you use a multi-service account, the key and endpoint will be the same for both your training and prediction.

1. In a new browser tab, open the Azure portal at `https://portal.azure.com`, and sign in using the Microsoft account associated with your Azure subscription.
2. Select the **+ Create a resource** button, search for *custom vision*, and create a **Custom Vision** resource with the following settings:
   - **Create options**: Both
   - **Subscription**: *Your Azure subscription*
   - **Resource group**: *Choose or create a resource group (if you are using a restricted subscription, you may not have permission to create a new resource group - use the one provided)*
   - **Region**: *Choose any available region*
   - **Name**: *Enter a unique name*
   - **Training pricing tier**: F0
   - **Prediction pricing tier**: F0

   > **Note**: If you already have an F0 custom vision service in your subscription, select **S0** for this one.

3. Wait for the resources to be created, and then view the deployment details and note that two Custom Vision resources are provisioned; one for training, and another for prediction (evident by the **-Prediction** suffix). You can view these by navigating to the resource group where you created them.

   > **Important**: Each resource has its own *endpoint* and *keys*, which are used to manage access from your code. To train an image classification model, your code must use the *training* resource (with its endpoint and key); and to use the trained model to predict image classes, your code must use the *prediction* resource (with its endpoint and key).

Create a Custom Vision project

To train an object detection model, you need to create a Custom Vision project based on your training resource. To do this, you'll use the Custom Vision portal.

1. In a new browser tab, open the Custom Vision portal at `https://customvision.ai`, and sign in using the Microsoft account associated with your Azure subscription.
2. Create a new project with the following settings:
   - **Name**: Detect Fruit
   - **Description**: Object detection for fruit.
   - **Resource**: *The Custom Vision resource you created previously*
   - **Project Types**: Object Detection
   - **Domains**: General
3. Wait for the project to be created and opened in the browser.

Add and tag images

To train an object detection model, you need to upload images that contain the classes you want the model to identify, and tag them to indicate bounding boxes for each object instance.

1. In Visual Studio Code, view the training images in the **03-object-detection/training-images** folder where you cloned the repository. This folder contains images of fruit.
2. In the Custom Vision portal, in your object detection project, select **Add images** and upload all of the images in the extracted folder.
3. After the images have been uploaded, select the first one to open it.

4. **Hold the mouse over any object in the image until an automatically detected region is displayed like the image below.** Then select the object, and if necessary resize the region to surround it.
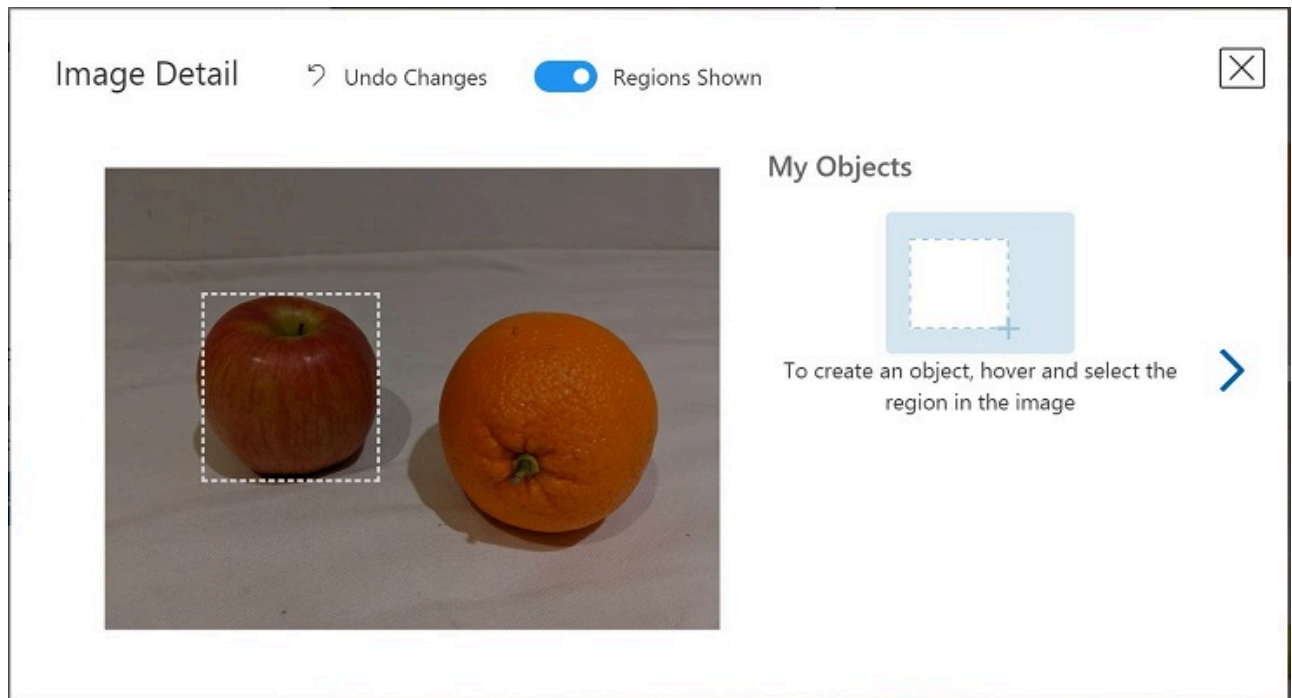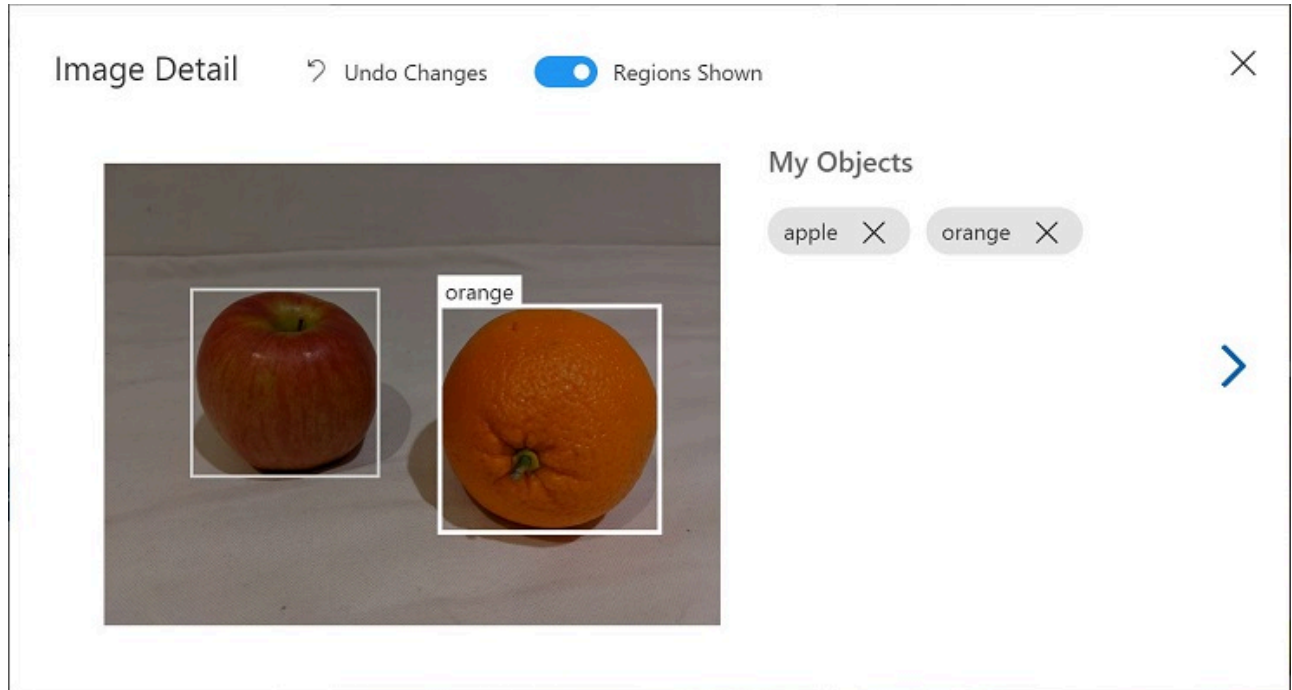

Alternatively, you can simply drag around the object to create a region.

5. When the region surrounds the object, add a new tag with the appropriate object type (*apple*, *banana*, or *orange*) as shown here:
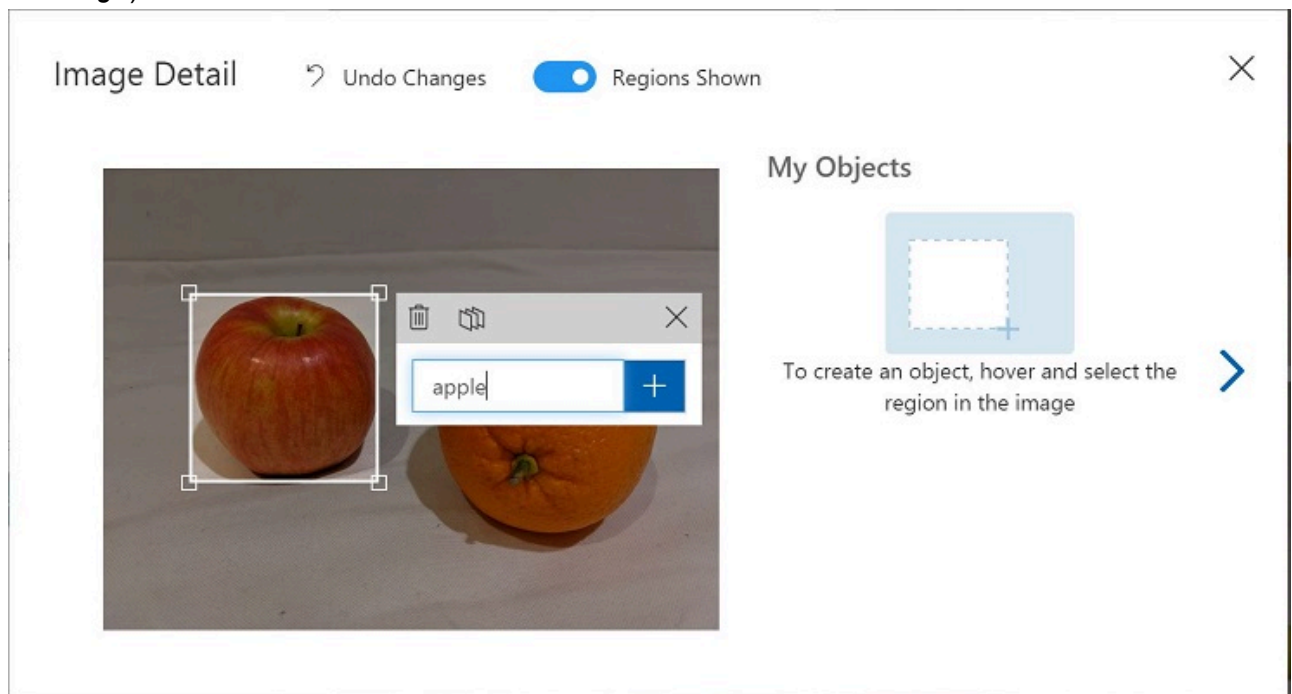
6. Select and tag each other object in the image, resizing the regions and adding new tags as required.

7. Use the **>** link on the right to go to the next image, and tag its objects. Then just keep working through the entire image collection, tagging each apple, banana, and orange.

8. When you have finished tagging the last image, close the **Image Detail** editor. On the **Training Images** page, under **Tags**, select **Tagged** to see all of your tagged images:
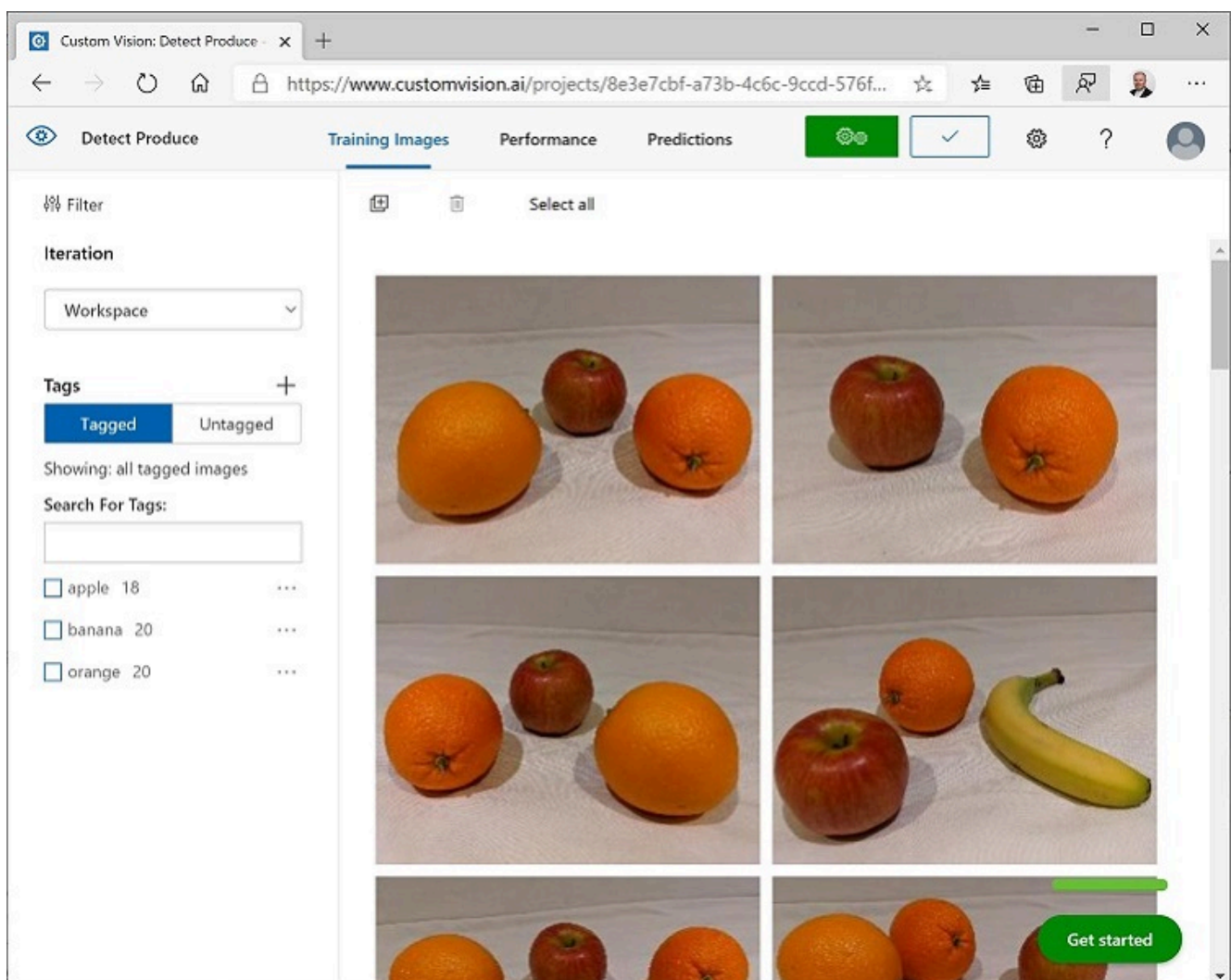
Use the Training API to upload images

You can use the UI in the Custom Vision portal to tag your images, but many AI development teams use other tools that generate files containing information about tags and object regions in images. In scenarios like this, you can use the Custom Vision training API to upload tagged images to the project.

> **Note**: In this exercise, you can choose to use the API from either the **C#** or **Python** SDK. In the steps below, perform the actions appropriate for your preferred language.

1. Click the *settings* (⚙) icon at the top right of the **Training Images** page in the Custom Vision portal to view the project settings.
2. Under **General** (on the left), note the **Project Id** that uniquely identifies this project.
3. On the right, under **Resources** note that the key and endpoint are shown. These are the details for the *training* resource (you can also obtain this information by viewing the resource in the Azure portal).
4. In Visual Studio Code, under the **03-object-detection** folder, expand the **C-Sharp** or **Python** folder depending on your language preference.
5. Right-click the **train-detector** folder and open an integrated terminal. Then install the Custom Vision Training package by running the appropriate command for your language preference:

C#

```
dotnet add package Microsoft.Azure.CognitiveServices.Vision.CustomVision.Training --
version 2.0.0
```

Python

```
pip install azure-cognitiveservices-vision-customvision==3.1.1
```

1. View the contents of the **train-detector** folder, and note that it contains a file for configuration settings:

   - **C#**: appsettings.json
   - **Python**: .env

   Open the configuration file and update the configuration values it contains to reflect the endpoint and key for your Custom Vision *training* resource, and the project ID for the object detection project you created previously. Save your changes.

2. In the **train-detector** folder, open **tagged-images.json** and examine the JSON it contains. The JSON defines a list of images, each containing one or more tagged regions. Each tagged region includes a tag name, and the top and left coordinates and width and height dimensions of the bounding box containing the tagged object.

   > **Note**: The coordinates and dimensions in this file indicate relative points on the image. **For example, a *height* value of 0.7 indicates a box that is 70% of the height of the image.** Some tagging tools generate other formats of file in which the coordinate and dimension values represent pixels, inches, or other units of measurements.

3. Note that the **train-detector** folder contains a subfolder in which the image files referenced in the JSON file are stored.
4. Note that the **train-detector** folder contains a code file for the client application:

   - **C#**: `Program.cs`

- **Python**: train-detector.py

Open the code file and review the code it contains, noting the following details:

- Namespaces from the package you installed are imported
- The **Main** function retrieves the configuration settings, and uses the key and endpoint to create an authenticated **CustomVisionTrainingClient**, which is then used with the project ID to create a **Project** reference to your project.
- The **Upload_Images** function extracts the tagged region information from the JSON file and uses it to create a batch of images with regions, which it then uploads to the project.

5. Return the integrated terminal for the **train-detector** folder, and enter the following command to run the program:

   C#

   ```
   dotnet run
   ```

   Python

   ```
   python train-detector.py
   ```

6. Wait for the program to end. Then return to your browser and view the **Training Images** page for your project in the Custom Vision portal (refreshing the browser if necessary).
7. Verify that some new tagged images have been added to the project.

Train and test a model

Now that you've tagged the images in your project, you're ready to train a model. You

1. In the Custom Vision project, click **Train** to train an object detection model using the tagged images. Select the **Quick Training** option.
2. Wait for training to complete (it might take ten minutes or so), and then review the *Precision*, *Recall*, and *mAP* performance metrics - these measure the prediction accuracy of the classification model, and should all be high.
3. At the top right of the page, click **Quick Test**, and then in the **Image URL** box, enter `https://aka.ms/apple-orange` and view the prediction that is generated. Then close the **Quick Test** window.

Publish the object detection model

Now you're ready to publish your trained model so that it can be used from a client application.

1. In the Custom Vision portal, on the **Performance** page, click ✓ **Publish** to publish the trained model with the following settings:
   - **Model name**: fruit-detector
   - **Prediction Resource**: *The **prediction** resource you created previously which ends with "-Prediction" (not the training resource)*.
2. At the top left of the **Project Settings** page, click the *Projects Gallery* (👁) icon to return to the Custom Vision portal home page, where your project is now listed.
3. On the Custom Vision portal home page, at the top right, click the *settings* (⚙) icon to view the settings for your Custom Vision service. Then, under **Resources**, find your *prediction* resource which ends with

"-Prediction" (not the training resource) to determine its **Key** and **Endpoint** values (you can also obtain this information by viewing the resource in the Azure portal).

Use the image classifier from a client application

Now that you've published the image classification model, you can use it from a client application. Once again, you can choose to use **C#** or **Python**.

1. In Visual Studio Code, browse to the **03-object-detection** folder and in the folder for your preferred language (**C-Sharp** or **Python**), expand the **test-detector** folder.
2. Right-click the **test-detector** folder and open an integrated terminal. Then enter the following SDK-specific command to install the Custom Vision Prediction package:

C#

```
dotnet add package Microsoft.Azure.CognitiveServices.Vision.CustomVision.Prediction --
version 2.0.0
```

Python

```
pip install azure-cognitiveservices-vision-customvision==3.1.1
```

> **Note**: The Python SDK package includes both training and prediction packages, and may already be installed.

1. Open the configuration file for your client application (*appsettings.json* for C# or *.env* for Python) and update the configuration values it contains to reflect the endpoint and key for your Custom Vision *prediction* resource, the project ID for the object detection project, and the name of your published model (which should be *fruit-detector*). Save your changes.
2. Open the code file for your client application (*Program.cs* for C#, test-detector.py for Python) and review the code it contains, noting the following details:
   - Namespaces from the package you installed are imported
   - The **Main** function retrieves the configuration settings, and uses the key and endpoint to create an authenticated **CustomVisionPredictionClient**.
   - The prediction client object is used to get object detection predictions for the **produce.jpg** image, specifying the project ID and model name in the request. The predicted tagged regions are then drawn on the image, and the result is saved as **output.jpg**.
3. Return to the integrated terminal for the **test-detector** folder, and enter the following command to run the program:

C#

```
dotnet run
```

Python

```
python test-detector.py
```

1. After the program has completed, view the resulting **output.jpg** file to see the detected objects in the image.

Knowledge Check

1. What does an object detection model predict? *

- ⦿ The location and class of specific classes of object in an image.

  ✔ **That's correct. Object detection is used to identify bounding boxes containing specific classes of object in an image.**

- ◯ The class of the main subject of an image.

- ◯ The file type of an image.

2. What must you do before taking advantage of the smart labeler tool when creating an object detection model? *

- ◯ Create a JSON file containing bounding box coordinates.

- ⦿ Tag some images with objects of each class and train an initial object detection model.

  ✔ **That's correct. To take advantage of the smart labeler, tag some images and train an initial model. Subsequently, the portal will suggest tags for new images.**

- ◯ Train an image classification (multilabel) model.

Summary

In this module, you have learned how to use the **Azure AI Custom Vision** service to create object detection models.

Now that you've completed this module, you can:

- Describe object detection.
- Train an object detector.
- Describe options for labeling images.

To find out more about the Azure AI Custom Vision service, see the Azure AI Custom Vision documentation.

✍ Compiled by Kenneth Leung (2025)