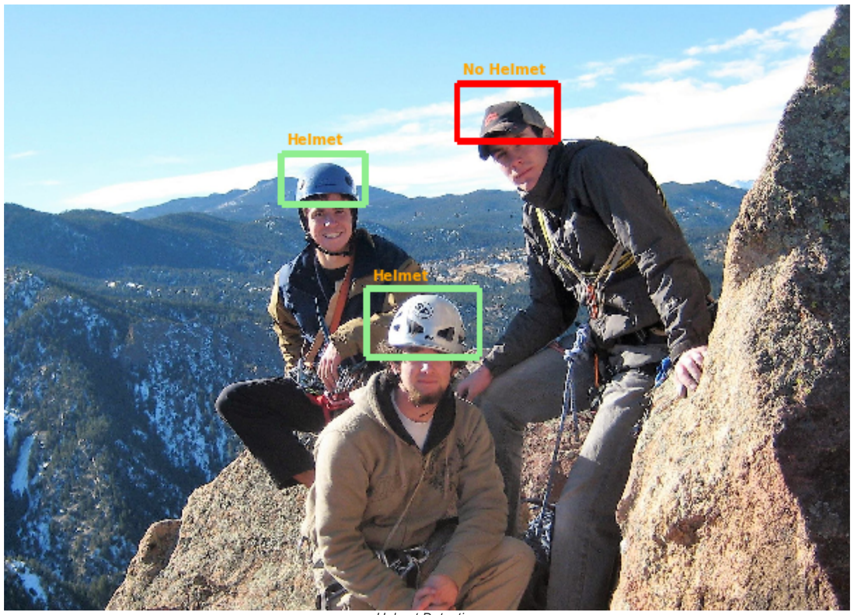
Challenge 6 - An Object Lesson in Safety

Background

Adventure Works has partnered with a local guide service to help them collect images of mountaineers and climbers in an effort to encourage the use of helmets for safety. Adventure Works wants a solution that can analyze an image and locate people's heads that are protected by a helmet and heads *not* wearing helmets; like this:



Helmet Detection

Object detection adds a layer of complexity and usefulness on top of classification. It predicts whether something is present, and where in the image it is located. This is important if the model needs to identify multiple instances of classes in a given image, and indicate their location with a *bounding box*.

The Custom Vision service enables you to create a cloud-based model for object detection. This requires that you train the model using your own images, indicating the location of the objects you want your model to detect by specifying the class and bounding box pixel coordinates for each object in the training images.

Prerequisites

- An environment for sharing code and working in Jupyter.
- A **Custom Vision** cognitive service account. If you don't already have one, create one here.
- A new set of *safety* images with which to train and test your object detection model. A list of image URLs for training and testing can be found here. You can run the following code in a Jupyter notebook cell to download the images:

```
import os
import shutil
import requests
from io import BytesIO
from PIL import Image

# Create an empty folder
folder = 'safety_images'
if os.path.exists(folder):
    shutil.rmtree(folder)
os.makedirs(folder)

# Get the list of image URLs
!curl https://challenge.blob.core.windows.net/challengefiles/summit_post_urls_selected.txt -o urls.txt
urls = open("urls.txt", "r")

# Download each image
for url in urls.readlines():
    url = url.rstrip()
    filename = url.split('/')[-1]
    response = requests.get(url)
    img = Image.open(BytesIO(response.content))
    saveAs = os.path.join(folder, filename)
    print("writing " + saveAs)
    img.save(saveAs, 'JPEG')
print("Images downloaded to", folder)
```

Challenge

1. Use the Custom Vision service to create an object detection model that identifies the location of heads protected by helmets, and heads *not* protected by helmets. You should train this model using a subset of the images.
2. Call a prediction endpoint for your model using Python code in a Jupyter Notebook to detect the location of protected and unprotected heads in some test images.
3. Visualize the test images, indicating the protected and unprotected heads using appropriately annotated bounding boxes.

Hints

- The Custom Vision service has an easy to use user interface for interactively uploading images, tagging objects in them, and training the model.

- Alternatively, you can use the Python SDK for the Custom Vision service to write code that creates your project, uploads images with their class and tag information, and trains your model. This provides a more repeatable solution.
- If you plan to write Python code to create your model, you will need to generate tags for the classes and bounding boxes in your images. The Visual Object Tagging Tool (VoTT) is a useful utility for this task.

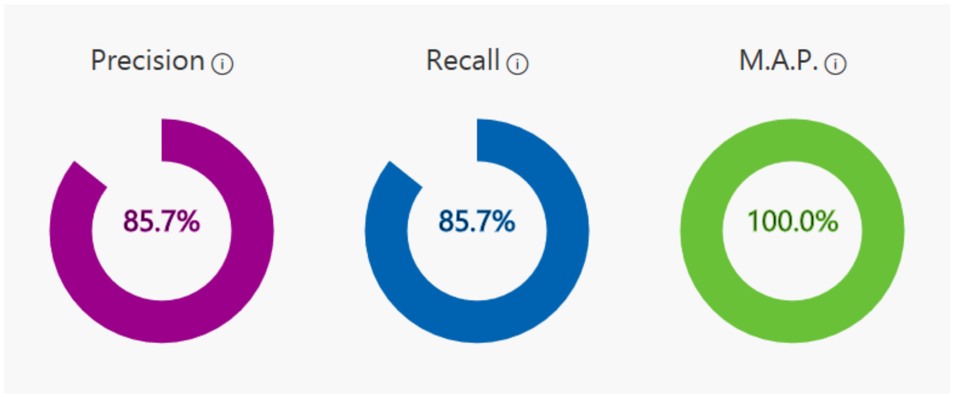
The **References** section includes some links to helpful resources.

Success Criteria

Object detection models are generally evaluated using the following metrics:

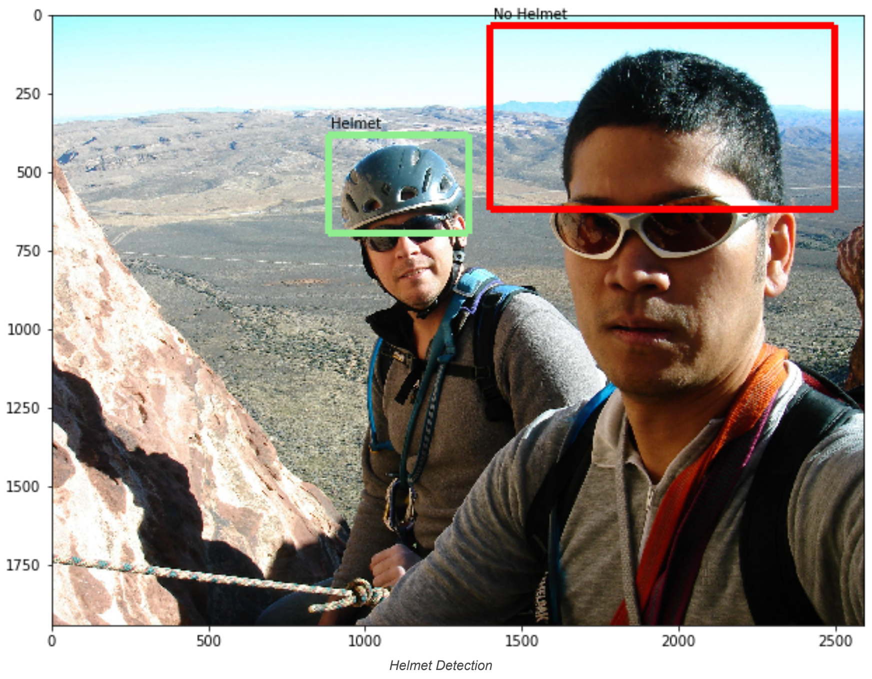
- **Precision**: The probability that predictions generated by your model for a given class are correct.
- **Recall**: The percentage of class instances your model predicts correctly.
- **Mean Average Precision (MAP)**: The overall performance for all classes.

Your model must achieve a MAP of **60%** or higher, as shown in the Custom Vision portal:



Custom Vision Metrics

Additionally, you must write code that uses the model to get predictions for the classes and locations of objects in a test image, and plots the image overlaid with annotated bounding boxes for the predicted classes, like this:



References

- Custom Vision portal
- Custom Vision documentation
- Custom Vision object detection tutorial
- Visual Object Tagging Tool (VoTT)