SIG788

Engineering AI solutions

Pass Task 4

Arunkumar Balaraman
S223919051

Contents

Target = Pass

Object Detection in Images using Azure's Computer Vision API

Introduction

Project uses Microsoft Azure's Computer Vision API to detect and categorize objects in images a key task in many computer vision applications.

Objective:

The goal is to create a program that can accurately identify objects in images using Azure's Computer Vision SDK. The program will highlight detected objects, demonstrating its practical application in real-world scenarios like inventory management and traffic analysis.

Approach:

The project involves several steps:

- **Setup and Authentication**: Establish the development environment and authenticate with Azure's Computer Vision service.
- **Image Processing**: Prepare images for analysis
- **Object Detection**: Use the Python SDK to **detect_objects** method to identify objects in the image.
- **Visualization**: Draw bounding boxes around detected objects for verification.

Azure Computer Vision:

Azure **Computer Vision**, part of Azure **Cognitive Services**, interprets and understands digital images using advanced algorithms and machine learning.

Key features include:

- Object Detection: Identifies and tracks objects in images.
- OCR: Extracts text from images.
- Image Analysis: Extracts visual features like objects and faces.

It's used in various applications like digital asset management and healthcare diagnostics, and is especially useful for tasks that mimic human capabilities, such as identifying objects and people in images.

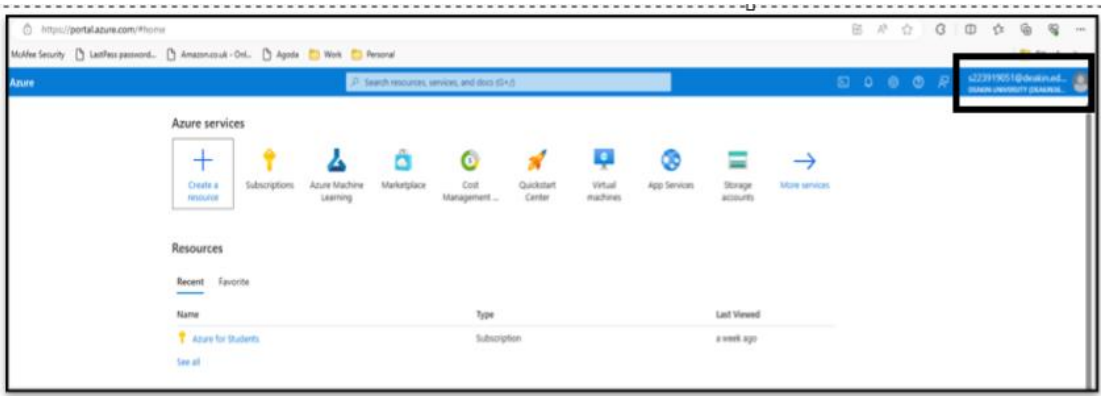
Accessible via API endpoints, it offers ready-to-use models for different use cases. It uses inputs from sensing devices, AI, machine learning, and deep learning to mimic human vision. The applications run on algorithms trained on vast amounts of visual data in the cloud, recognizing patterns to determine the content of other images.

Creating Azure Machine Learning Workspace:

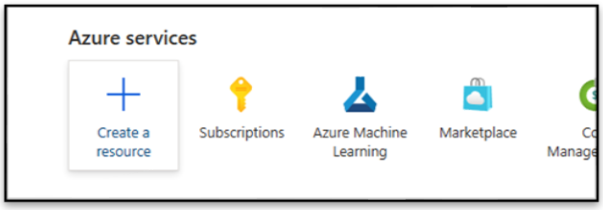
- **Sign in to Azure Portal:** Azure Portal and sign in with Microsoft account Deakin credentials.

  Link: https://portal.azure.com/

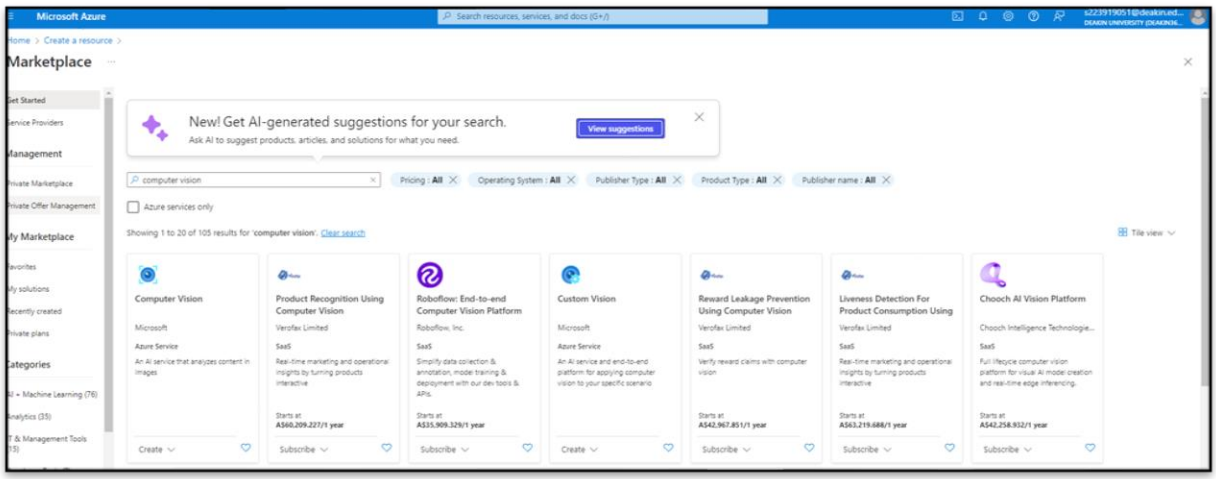
  Landing Page:



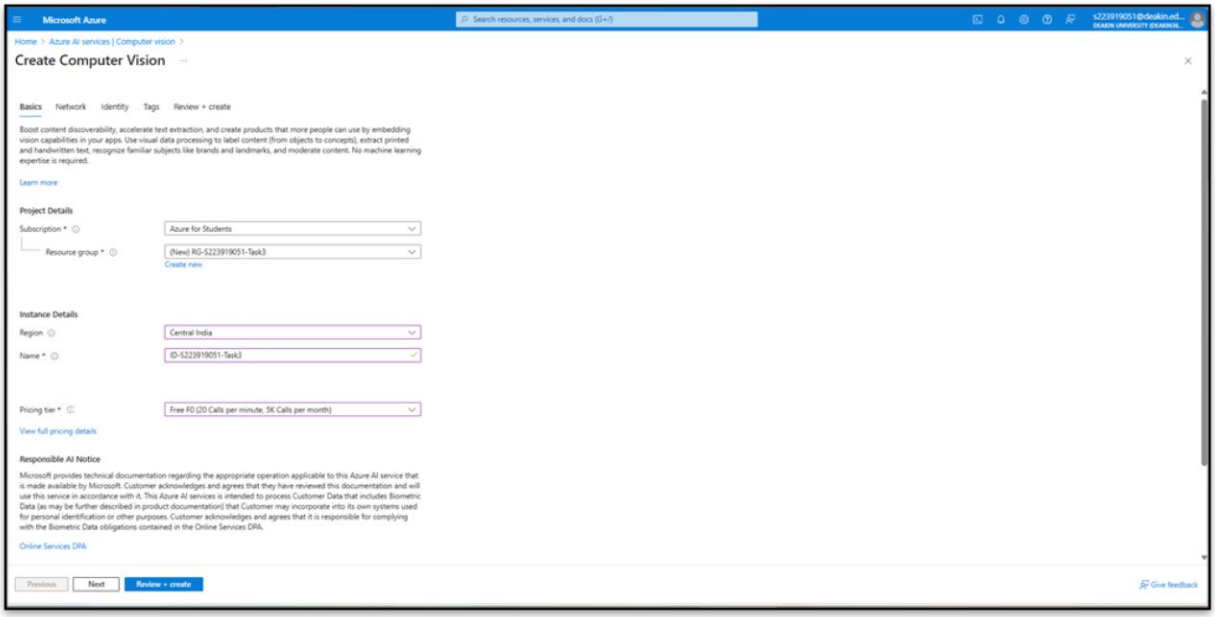
- **Create a New Workspace**:
  - Click on **Create a resource** at the top-left corner.

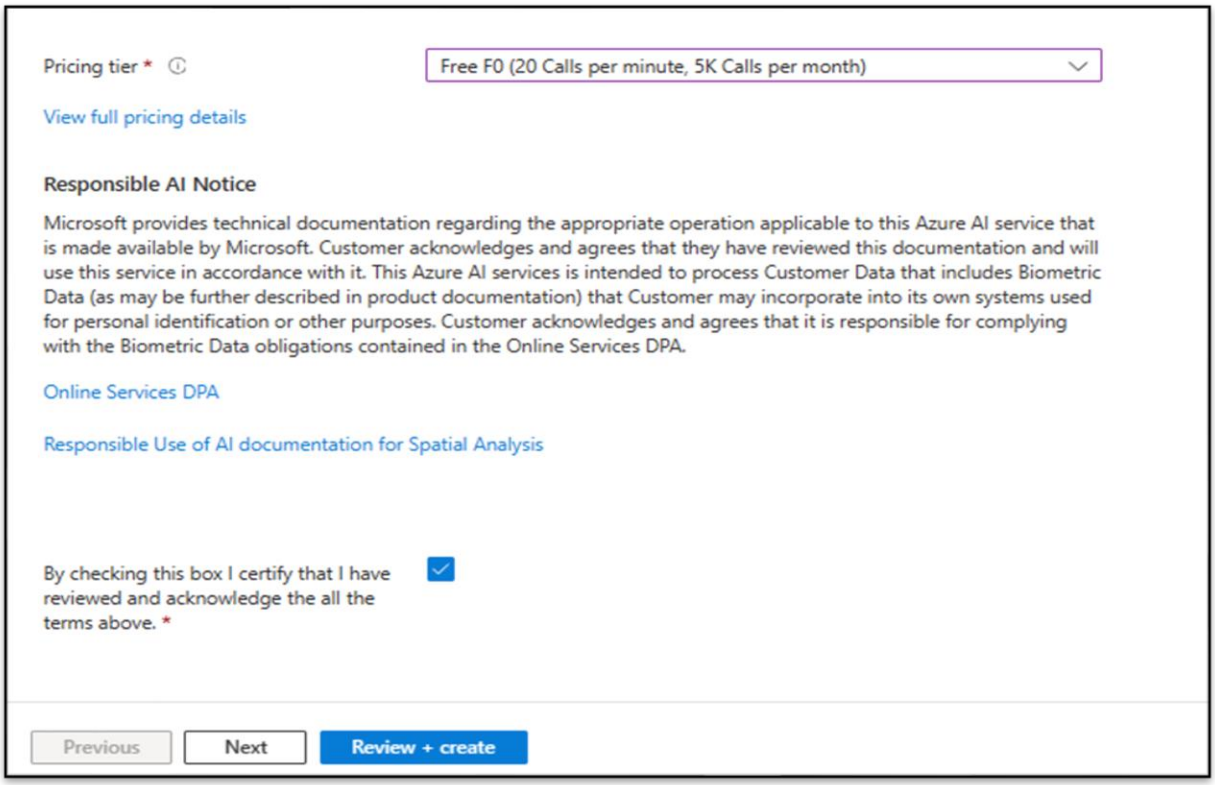


  - Search for **Computer Vision** & select **Create** from Azure Machine Learning to initiate the workspace creation process.



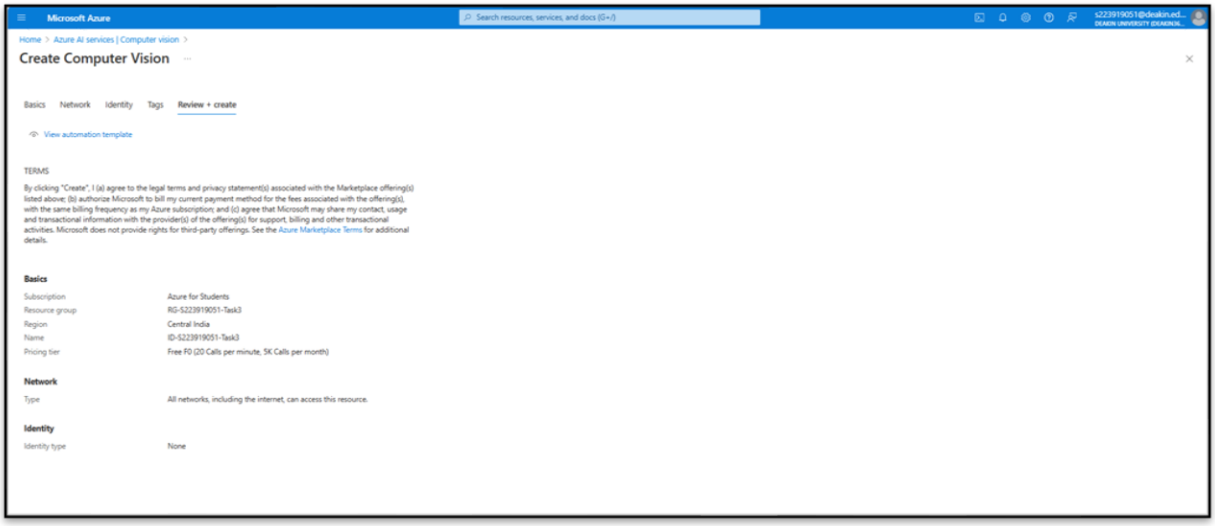
- **Configure the Workspace**:
  - Selecting Deakin Student Azure subscription
  - Creating a resource group **RG-S223919051-Task4**
  - Instance Detail name **ID-S223919051-Task4**
  - Choose an Azure region **Central India** which is close to my location.
  - Pricing tier has been selected with Free Tier "**Free F0 (20 Calls per minute, 5K calls per month)**
  - Also certified terms from Microsoft by checking the button.



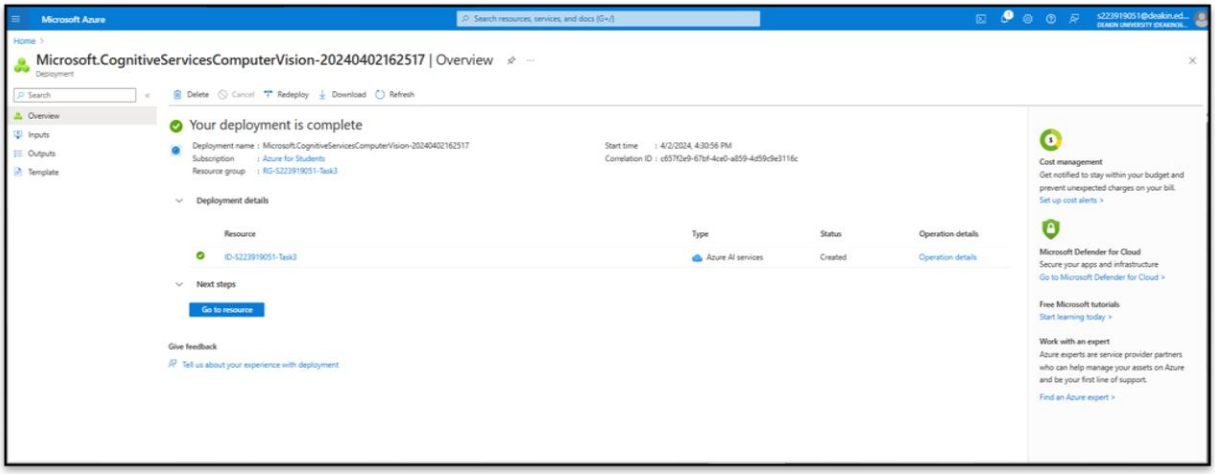


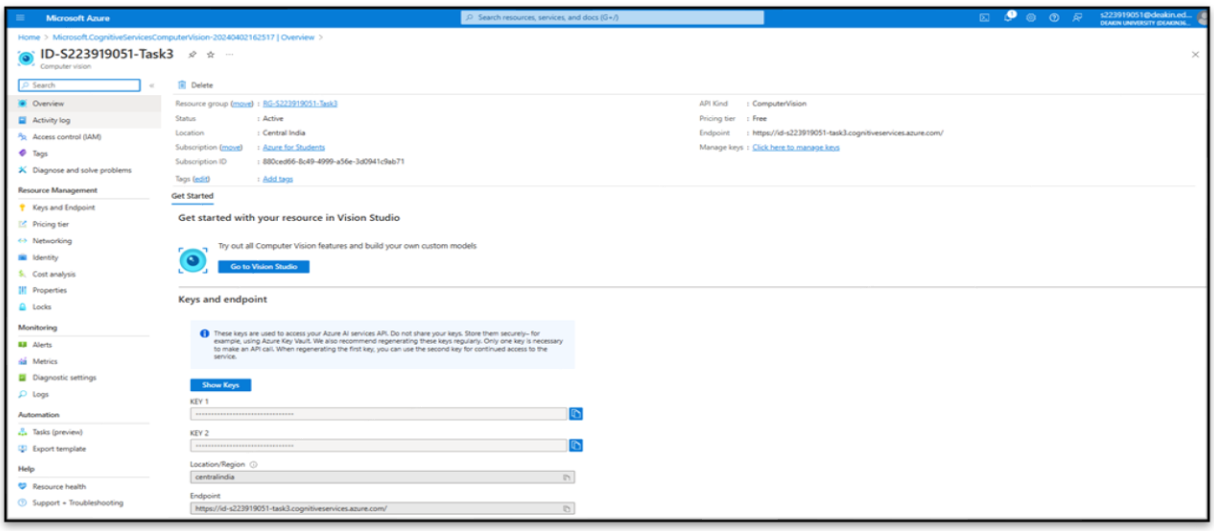
- **Review and Create**: After configuring the details click on **Review + create** button. Once Azure validates configuration. Click on **Create** button to deploy workspace.

Review:



Workspace deployed:





Let's move forward with the Python SDK for Azure Computer Vision.

Pre-requisites:

Install pre-requisites for Computer Vision in Anaconda prompt.

- pip install --upgrade azure-cognitiveservices-vision-computervision
- pip install pillow
- pip install python-dotenv

- **azure-cognitiveservices-vision-computervision**: This Python package interacts with Azure's AI-powered Computer Vision service, providing image analysis capabilities like object identification, face detection, and OCR.

- **pillow**: A PIL fork, Pillow adds image processing capabilities to Python, including opening, manipulating, and saving various image file formats. It's commonly used for tasks like resizing, applying filters, and handling different image formats.

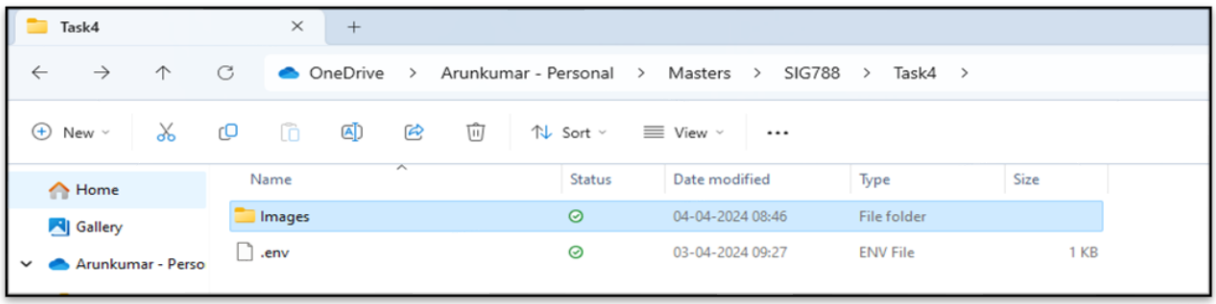
- **python-dotenv**: This package reads key-value pairs from a .env file and sets them as environment variables, enhancing the security and flexibility of application's configuration management.

Configure .env file:

For Azure Computer Vision services, follow these steps to enhance security:

- After deployment, copy Key1 (**subscription_Key**) and the **endpoint**.
- Paste these into a **.env** file in working directory.
- These will be assigned as environment variables and can be read directly in Python code.

This method ensures **subscription_key** and **endpoint** are not displayed to users, enhancing the security of application. Please refer to the snapshot of the **.env** file.



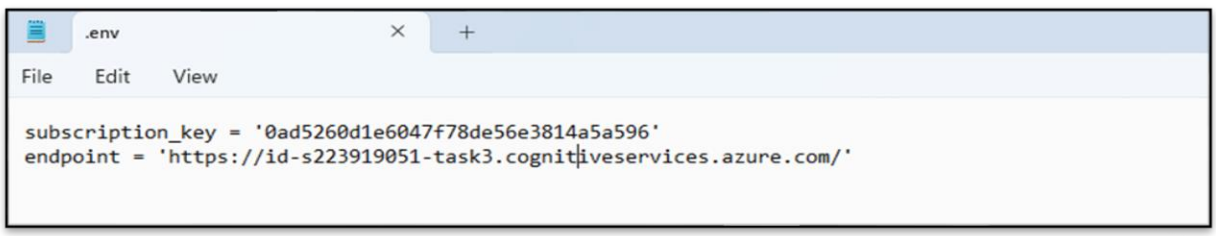


Image Selection for Object Detection:

I've selected a few images from the internet to evaluate the Azure Computer Vision model. Here are the 8 images I've chosen for testing the object detection model.

Billgates with Car:

Flowers with butterfly:

Cattle with Farm:

Ronaldo and Messi playing football:

Bikers in the bike:

Paris Tower with a Girl:

Random people with car:

Celebrities Amitab and Rajni:

Azure's Python SDK:

Import all necessary libraries:

```
# Import the Computer Vision Client from Azure's Cognitive Services for computer vision tasks
from azure.cognitiveservices.vision.computervision import ComputerVisionClient

# Import OperationStatusCodes for tracking the status of operations in the Computer Vision client
from azure.cognitiveservices.vision.computervision.models import OperationStatusCodes

# Import VisualFeatureTypes to specify visual features of interest in an image analysis request
from azure.cognitiveservices.vision.computervision.models import VisualFeatureTypes

# Import CognitiveServicesCredentials for authenticating the Computer Vision client
from msrest.authentication import CognitiveServicesCredentials

# Import os for interacting with the operating system, e.g., reading environment variables
import os

# Import Image from PIL (Python Imaging Library) for image processing tasks
from PIL import Image, ImageDraw, ImageFont

# Import sys for accessing system-specific parameters and functions
import sys

# Load environment variables from a .env file for secure access to Azure subscription key and endpoint
from dotenv import load_dotenv

# Plots
import matplotlib.pyplot as plt
```

Now let's set up set up the working directory.

```
# Setting Working Directory
os.chdir(r'C:\Users\arunk\OneDrive\Masters\SIG788\Task4')
print(os.getcwd())
```

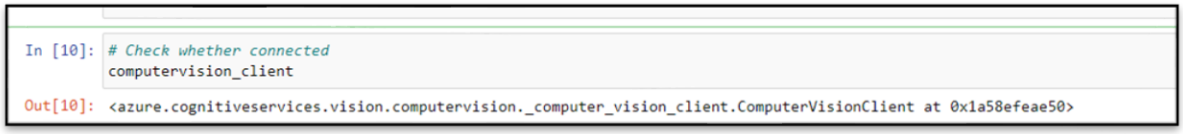
Load the environment variables:

```
# Load variables from .env file to environment variables
load_dotenv()
```

Authenticate using subscription key and endpoint created in Azure.

```
# Retrieve Azure subscription key and endpoint URL from environment variables
subscription_key = os.environ["subscription_key"]
endpoint = os.environ["endpoint"]

# Authenticates using the retrieved subscription key and endpoint, and creates an instance of the Computer Vision Client
computervision_client = ComputerVisionClient(endpoint,
CognitiveServicesCredentials(subscription_key))
```

Checking the Authentication:



Display all the images through code:

```
# Image Directory
image_directory = os.getcwd() + '/Images'

# List all files in the directory
files_in_directory = os.listdir(image_directory)

# Filter for images
image_files = [file for file in files_in_directory if file.lower().endswith(('.png', '.jpg', '.jpeg', '.gif', '.bmp'))]

# Sort the images
image_files.sort()

# matplotlib for 4x2 grid
fig, axes = plt.subplots(4, 2, figsize=(10, 20))

# axes
axes = axes.flatten()

for i, image_file in enumerate(image_files):
    # Open the image
    img = Image.open(os.path.join(image_directory, image_file))

    # Display the image
    axes[i].imshow(img)
    axes[i].axis('off')

# Show the plot
plt.tight_layout()
plt.show()
```

Output:

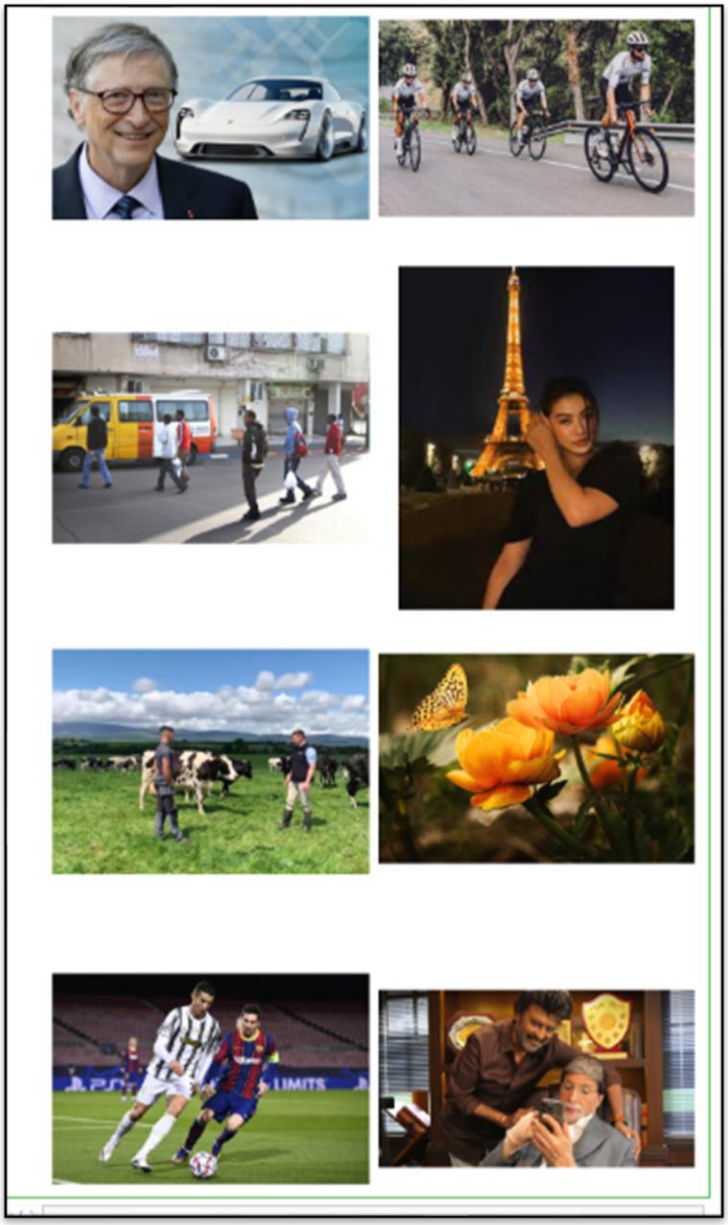


Image Tags using Azure:

Now that we've selected the images, let's move forward with generating tags using Azure Computer Vision.

Azure's Image Tags feature uses AI to generate relevant tags for an image, aiding in categorization and search.

Code:

```
# function to detect and print tags for a given image
def detect_tags(image_path, client):
    # Start of tag detection for the current image
    print(f"\nDetecting tags for: {image_path}")
    # Open the image file in binary-read mode
    with open(image_path, "rb") as image_stream:
        # Use the Azure client to detect tags in the image stream
        tags_result = client.tag_image_in_stream(image_stream)
        # Iterate through detected tags and print them with confidence levels
        for tag in tags_result.tags:
            print(f"Tag: {tag.name} with confidence {tag.confidence * 100:.2f}%")

# Iterate over each image file in the 'image_files' list
for image_file in image_files:
    image = os.path.join(image_directory, image_file)
    # Call 'detect_tags' function for each image, passing the Azure client
    detect_tags(image, computervision_client)
```

Output:

```
Detecting tags for: C:\Users\arunk\OneDrive\Masters\SIG788\Task4/Images\Bill-Gates-buys-Porsche-Taycan-small.jpg
Tag: person with confidence 98.93%
Tag: land vehicle with confidence 97.48%
Tag: vehicle with confidence 96.74%
Tag: automotive design with confidence 94.94%
Tag: suit with confidence 92.69%
Tag: luxury vehicle with confidence 92.02%
Tag: wheel with confidence 91.78%
Tag: car with confidence 91.59%
Tag: clothing with confidence 91.33%
Tag: human face with confidence 90.73%
Tag: man with confidence 88.76%
Tag: smile with confidence 84.21%
Tag: outdoor with confidence 71.27%
Tag: automotive with confidence 45.42%

Detecting tags for: C:\Users\arunk\OneDrive\Masters\SIG788\Task4/Images\Cover_187.jpg
Tag: bicycle wheel with confidence 99.55%
Tag: tree with confidence 99.39%
Tag: wheel with confidence 99.33%
Tag: outdoor with confidence 99.30%
Tag: land vehicle with confidence 98.88%
Tag: cycling with confidence 98.78%
Tag: bicycles--equipment and supplies with confidence 98.64%
Tag: sports equipment with confidence 98.60%
```

```
Tag: cycle sport with confidence 98.54%
Tag: vehicle with confidence 98.47%
Tag: helmet with confidence 98.44%
Tag: bicycle frame with confidence 98.35%
Tag: bicycle helmet with confidence 98.15%
Tag: bicycle handlebar with confidence 97.73%
Tag: person with confidence 96.86%
Tag: road bicycle with confidence 96.16%
Tag: bicycle pedal with confidence 96.05%
Tag: bicycle tire with confidence 95.89%
Tag: road bicycle racing with confidence 94.67%
Tag: bicycle fork with confidence 94.38%
Tag: racing bicycle with confidence 93.37%
Tag: bicycle racing with confidence 93.28%
Tag: bicycle with confidence 93.13%
Tag: cyclo-cross with confidence 93.08%
Tag: hybrid bicycle with confidence 92.58%
Tag: cyclo-cross bicycle with confidence 91.40%
Tag: bicycle jersey with confidence 90.34%
Tag: road cycling with confidence 90.10%
Tag: cycling shorts with confidence 89.43%
Tag: bicycle stem with confidence 89.38%
Tag: riding with confidence 88.75%
Tag: mountain bike with confidence 88.52%
Tag: road with confidence 87.52%
Tag: bicycle shoe with confidence 86.44%
Tag: crankset with confidence 84.65%
Tag: groupset with confidence 84.30%
Tag: bike with confidence 81.54%
Tag: people with confidence 66.89%
Tag: bicycling with confidence 57.99%

Detecting tags for: C:\Users\arunk\OneDrive\Masters\SIG788\Task4/Images\F110503FF
NK16.jpg
Tag: outdoor with confidence 99.63%
Tag: footwear with confidence 99.28%
Tag: vehicle with confidence 98.97%
Tag: land vehicle with confidence 98.75%
Tag: clothing with confidence 97.51%
Tag: building with confidence 97.44%
Tag: person with confidence 97.05%
Tag: road with confidence 96.85%
Tag: car with confidence 92.86%
Tag: pedestrian with confidence 90.70%
Tag: van with confidence 90.45%
Tag: man with confidence 89.61%
Tag: wheel with confidence 88.54%
Tag: transport with confidence 87.55%
Tag: people with confidence 85.96%
```

```
Tag: street with confidence 84.57%
Tag: group with confidence 77.28%
Tag: city with confidence 56.39%

Detecting tags for: C:\Users\arunk\OneDrive\Masters\SIG788\Task4/Images\Paristower.
jpeg
Tag: sky with confidence 98.76%
Tag: person with confidence 98.32%
Tag: human face with confidence 95.51%
Tag: outdoor with confidence 93.07%
Tag: building with confidence 90.31%
Tag: clothing with confidence 89.24%
Tag: girl with confidence 87.90%
Tag: tower with confidence 84.08%
Tag: woman with confidence 69.94%
Tag: night with confidence 65.37%

Detecting tags for: C:\Users\arunk\OneDrive\Masters\SIG788\Task4/Images\cattles.jpg
Tag: grass with confidence 99.94%
Tag: sky with confidence 99.61%
Tag: outdoor with confidence 99.54%
Tag: cloud with confidence 97.99%
Tag: cattle with confidence 97.84%
Tag: livestock with confidence 97.10%
Tag: pasture with confidence 95.28%
Tag: dairy cow with confidence 95.00%
Tag: field with confidence 94.16%
Tag: ranch with confidence 94.13%
Tag: cow with confidence 93.88%
Tag: herd with confidence 91.98%
Tag: person with confidence 90.87%
Tag: grassland with confidence 90.85%
Tag: herding with confidence 88.36%
Tag: clothing with confidence 86.89%
Tag: grazing with confidence 86.10%
Tag: meadow with confidence 84.48%
Tag: bovine with confidence 77.81%
Tag: standing with confidence 76.67%
Tag: farm with confidence 71.22%
Tag: group with confidence 67.03%

Detecting tags for: C:\Users\arunk\OneDrive\Masters\SIG788\Task4/Images\flowers-19
830_640.jpg
Tag: butterfly with confidence 98.97%
Tag: moths and butterflies with confidence 98.16%
Tag: insect with confidence 97.50%
Tag: invertebrate with confidence 97.34%
Tag: animal with confidence 92.39%
Tag: outdoor with confidence 90.83%
```

```
Tag: pollinator with confidence 89.47%
Tag: petal with confidence 87.84%
Tag: flower with confidence 82.21%
Tag: orange with confidence 77.50%
Tag: plant with confidence 64.58%

Detecting tags for: C:\Users\arunk\OneDrive\Masters\SIG788\Task4/Images\imago1002
614356h.jpg
Tag: person with confidence 99.44%
Tag: soccer player with confidence 98.33%
Tag: athletic game with confidence 98.03%
Tag: soccer with confidence 98.02%
Tag: grass with confidence 97.56%
Tag: sport with confidence 97.17%
Tag: football with confidence 97.13%
Tag: football player with confidence 96.04%
Tag: sports equipment with confidence 95.40%
Tag: ball game with confidence 94.76%
Tag: soccer ball with confidence 94.46%
Tag: forward with confidence 94.33%
Tag: soccer-specific stadium with confidence 93.41%
Tag: team sport with confidence 92.35%
Tag: ball with confidence 92.00%
Tag: field with confidence 91.19%
Tag: outdoor with confidence 91.15%
Tag: playing with confidence 90.77%
Tag: kick with confidence 90.16%
Tag: stadium with confidence 88.37%
Tag: tournament with confidence 86.24%
Tag: player with confidence 85.87%
Tag: sport venue with confidence 85.25%
Tag: footwear with confidence 84.49%
Tag: championship with confidence 84.10%
Tag: game with confidence 83.67%
Tag: team with confidence 55.14%

Detecting tags for: C:\Users\arunk\OneDrive\Masters\SIG788\Task4/Images\rajini_1698
573891200_1698573891700.jpg
Tag: person with confidence 99.90%
Tag: clothing with confidence 99.71%
Tag: human face with confidence 99.65%
Tag: indoor with confidence 97.79%
Tag: man with confidence 97.13%
Tag: window with confidence 94.93%
Tag: smile with confidence 88.20%
Tag: furniture with confidence 86.51%
Tag: wall with confidence 83.72%
Tag: holding with confidence 75.04%
```

Landmark Detection:

Azure Computer Vision's Landmark Detection identifies landmarks in images using machine learning. It detects natural and man-made landmarks, providing their names and coordinates. Useful in travel and tourism applications, its accuracy depends on image quality and angle. Accessible via API endpoints, it can be integrated into various applications.

Code:

```
# Define a function to detect and report landmarks for a given image
def detect_landmarks(image_path, client):
    # Start of Landmark detection for the current image
    print(f"\nDetecting landmarks for: {image_path}")
    # Open the image file in binary-read mode
    with open(image_path, "rb") as image_stream:
        try:
            # detect landmarks using Azure's analyze_image_by_domain_in_stream method
            landmark_results = client.analyze_image_by_domain_in_stream("landmarks",
image_stream, "en")
            landmarks = landmark_results.result["landmarks"]
            # If landmarks are detected, print it with confidence levels
            if landmarks:
                for landmark in landmarks:
                    print(f"Landmark detected: {landmark['name']} with confidence
{landmark['confidence'] * 100:.2f}%")
            else:
                # if no landmarks
                print("No landmarks detected.")
        except Exception as e:
            # Print an error message if landmark detection was unsuccessful
            print(f"Landmark detection not successful: {e}")

# Iterate over each image file in the 'image_files' list
for image_file in image_files:
    image = os.path.join(image_directory, image_file)
    # Call the 'detect_landmarks' function for each image, passing the Azure client
    detect_landmarks(image, computervision_client)
```

Output:

```
Detecting landmarks for: C:\Users\arunk\OneDrive\Masters\SIG788\Task4/Images\Bill-
Gates-buys-Porsche-Taycan-small.jpg
No landmarks detected.

Detecting landmarks for: C:\Users\arunk\OneDrive\Masters\SIG788\Task4/Images\Cove
r_187.jpg
No landmarks detected.
```

```
Detecting landmarks for: C:\Users\arunk\OneDrive\Masters\SIG788\Task4/Images\F110503FFNK16.jpg
No landmarks detected.

Detecting landmarks for: C:\Users\arunk\OneDrive\Masters\SIG788\Task4/Images\Paris tower.jpeg
No landmarks detected.

Detecting landmarks for: C:\Users\arunk\OneDrive\Masters\SIG788\Task4/Images\cattles.jpg
No landmarks detected.

Detecting landmarks for: C:\Users\arunk\OneDrive\Masters\SIG788\Task4/Images\flowers-19830_640.jpg
No landmarks detected.

Detecting landmarks for: C:\Users\arunk\OneDrive\Masters\SIG788\Task4/Images\imago1002614356h.jpg
No landmarks detected.

Detecting landmarks for: C:\Users\arunk\OneDrive\Masters\SIG788\Task4/Images\rajini_1698573891200_1698573891700.jpg
No landmarks detected.
```

Challenges:

Azure Computer Vision, while powerful can sometimes face challenges in detecting landmarks like the Eiffel Tower especially if the image quality is poor or the landmark is in the background or obscured. Nighttime images can also pose a challenge due to lighting conditions.

In such cases, Azure Custom Vision could be a viable alternative as it allows for more customized training.

Additionally, providing clearer images, particularly of landscapes can improve detection accuracy.

Image description in Azure:

Azure's Image Description provides a human-readable sentence summarizing an image's content.

Code:

```
# function to get and print a description for a given image
def get_image_description(image_path, client):
    # Start of getting a description for the current image
    print(f"\nGetting description for: {image_path}")
    # Open the image file in binary-read mode
    with open(image_path, "rb") as image_stream:
        # Use the Azure client to get a description of the image
```

```
        description_results = client.describe_image_in_stream(image_stream)
        # If descriptions (captions) are found, print it with their confidence levels
        if description_results.captions:
            for caption in description_results.captions:
                print(f"Description: '{caption.text}' with confidence {caption.confidence * 100:.2f}%")
        else:
            # If no description
            print("No description available.")

# Iterate over each image file in the 'image_files' list
for image_file in image_files:
    image = os.path.join(image_directory, image_file)
    # Call the 'get_image_description' function for each image, passing the Azure client
    get_image_description(image, computervision_client)
```

Output:

```
Getting description for: C:\Users\arunk\OneDrive\Masters\SIG788\Task4/Images\Bill-Gates-buys-Porsche-Taycan-small.jpg
Description: 'a man in a suit and tie' with confidence 42.45%

Getting description for: C:\Users\arunk\OneDrive\Masters\SIG788\Task4/Images\Cover_187.jpg
Description: 'a group of men riding bikes' with confidence 51.66%

Getting description for: C:\Users\arunk\OneDrive\Masters\SIG788\Task4/Images\F110503FFNK16.jpg
Description: 'a group of people walking on a street' with confidence 45.76%

Getting description for: C:\Users\arunk\OneDrive\Masters\SIG788\Task4/Images\Paristower.jpeg
Description: 'a person posing in front of the eiffel tower at night' with confidence 69.60%

Getting description for: C:\Users\arunk\OneDrive\Masters\SIG788\Task4/Images\cattles.jpg
Description: 'men standing in a field with cows' with confidence 45.88%

Getting description for: C:\Users\arunk\OneDrive\Masters\SIG788\Task4/Images\flowers-19830_640.jpg
Description: 'a butterfly on a flower' with confidence 64.16%

Getting description for: C:\Users\arunk\OneDrive\Masters\SIG788\Task4/Images\imago1002614356h.jpg
Description: 'a couple of football players compete over a football ball' with confidence 56.66%
```

```
Getting description for: C:\Users\arunk\OneDrive\Masters\SIG788\Task4/Images\rajini_1698573891200_1698573891700.jpg
Description: 'a man taking a picture of a man sitting on a chair' with confidence 42.99 %
```

Celebrity Detection:

Azure's Celebrity Detection identifies famous individuals in images using machine learning, providing their names and confidence scores.

Code:

```
# function to detect celebrities within an image
def detect_celebrities(image_path, client):
    # Start of celebrity detection for the current image
    print(f"\nDetecting celebrities in: {image_path}")
    try:
        # Open the image file in binary-read mode
        with open(image_path, "rb") as image_stream:
            # Attempt to detect celebrities using Azure's analyze_image_by_domain_in_stream method
            celeb_results = client.analyze_image_by_domain_in_stream("celebrities", image_stream, "en")
            celebrities = celeb_results.result["celebrities"]
            # If celebrities are found
            if celebrities:
                print("Celebrities detected:")
                for celeb in celebrities:
                    print(f"- {celeb['name']} ({celeb['confidence'] * 100:.2f}%)")
            else:
                # If no celebrities were detected
                print("No celebrities detected.")
    except Exception as e:
        # Error
        print("An error occurred while trying to detect celebrities in the image. The Celebrity Recognition feature is not supported and requires registration. To apply for access, visit https://aka.ms/celebrityrecognition. This feature is available only to Microsoft managed customers and partners. If conducting academic research or independent testing for public benefit, contact limacctr@microsoft.com with project details.")
        print(f"Error: {e}")

# Iterate over each image file in the 'image_files' list
for image_file in image_files:
    image = os.path.join(image_directory, image_file)
    # Call the 'detect_celebrities' function for each image, passing the Azure client
    detect_celebrities(image, computervision_client)
```

Output:

Detecting celebrities in: C:\Users\arunk\OneDrive\Masters\SIG788\Task4/Images\Bill-Gates-buys-Porsche-Taycan-small.jpg
An error occurred while trying to detect celebrities in the image. The Celebrity Recognition feature is not supported and requires registration. To apply for access, visit https://aka.ms/celebrityrecognition. This feature is available only to Microsoft managed customers and partners. If conducting academic research or independent testing for public benefit, contact limacctr@microsoft.com with project details.
Error: (InvalidRequest) Feature is not supported. Please apply for access at https://aka.ms/celebrityrecognition

Detecting celebrities in: C:\Users\arunk\OneDrive\Masters\SIG788\Task4/Images\Cover_187.jpg
An error occurred while trying to detect celebrities in the image. The Celebrity Recognition feature is not supported and requires registration. To apply for access, visit https://aka.ms/celebrityrecognition. This feature is available only to Microsoft managed customers and partners. If conducting academic research or independent testing for public benefit, contact limacctr@microsoft.com with project details.
Error: (InvalidRequest) Feature is not supported. Please apply for access at https://aka.ms/celebrityrecognition

Detecting celebrities in: C:\Users\arunk\OneDrive\Masters\SIG788\Task4/Images\F110503FFNK16.jpg
An error occurred while trying to detect celebrities in the image. The Celebrity Recognition feature is not supported and requires registration. To apply for access, visit https://aka.ms/celebrityrecognition. This feature is available only to Microsoft managed customers and partners. If conducting academic research or independent testing for public benefit, contact limacctr@microsoft.com with project details.
Error: (InvalidRequest) Feature is not supported. Please apply for access at https://aka.ms/celebrityrecognition

Detecting celebrities in: C:\Users\arunk\OneDrive\Masters\SIG788\Task4/Images\Paristower.jpeg
An error occurred while trying to detect celebrities in the image. The Celebrity Recognition feature is not supported and requires registration. To apply for access, visit https://aka.ms/celebrityrecognition. This feature is available only to Microsoft managed customers and partners. If conducting academic research or independent testing for public benefit, contact limacctr@microsoft.com with project details.
Error: (InvalidRequest) Feature is not supported. Please apply for access at https://aka.ms/celebrityrecognition

Detecting celebrities in: C:\Users\arunk\OneDrive\Masters\SIG788\Task4/Images\cattles.jpg
An error occurred while trying to detect celebrities in the image. The Celebrity Recognition feature is not supported and requires registration. To apply for access, visit https://aka.ms/celebrityrecognition. This feature is available only to Microsoft managed customers and partners. If conducting academic research or independent testing for public benefit, contact limacctr@microsoft.com with project details.

Error: (InvalidRequest) Feature is not supported. Please apply for access at https://aka.ms/celebrityrecognition

Detecting celebrities in: C:\Users\arunk\OneDrive\Masters\SIG788\Task4/Images\flowers-19830_640.jpg
An error occurred while trying to detect celebrities in the image. The Celebrity Recognition feature is not supported and requires registration. To apply for access, visit https://aka.ms/celebrityrecognition. This feature is available only to Microsoft managed customers and partners. If conducting academic research or independent testing for public benefit, contact limacctr@microsoft.com with project details.
Error: (InvalidRequest) Feature is not supported. Please apply for access at https://aka.ms/celebrityrecognition

Detecting celebrities in: C:\Users\arunk\OneDrive\Masters\SIG788\Task4/Images\imago1002614356h.jpg
An error occurred while trying to detect celebrities in the image. The Celebrity Recognition feature is not supported and requires registration. To apply for access, visit https://aka.ms/celebrityrecognition. This feature is available only to Microsoft managed customers and partners. If conducting academic research or independent testing for public benefit, contact limacctr@microsoft.com with project details.
Error: (InvalidRequest) Feature is not supported. Please apply for access at https://aka.ms/celebrityrecognition

Detecting celebrities in: C:\Users\arunk\OneDrive\Masters\SIG788\Task4/Images\rajini_1698573891200_1698573891700.jpg
An error occurred while trying to detect celebrities in the image. The Celebrity Recognition feature is not supported and requires registration. To apply for access, visit https://aka.ms/celebrityrecognition. This feature is available only to Microsoft managed customers and partners. If conducting academic research or independent testing for public benefit, contact limacctr@microsoft.com with project details.
Error: (InvalidRequest) Feature is not supported. Please apply for access at https://aka.ms/celebrityrecognition

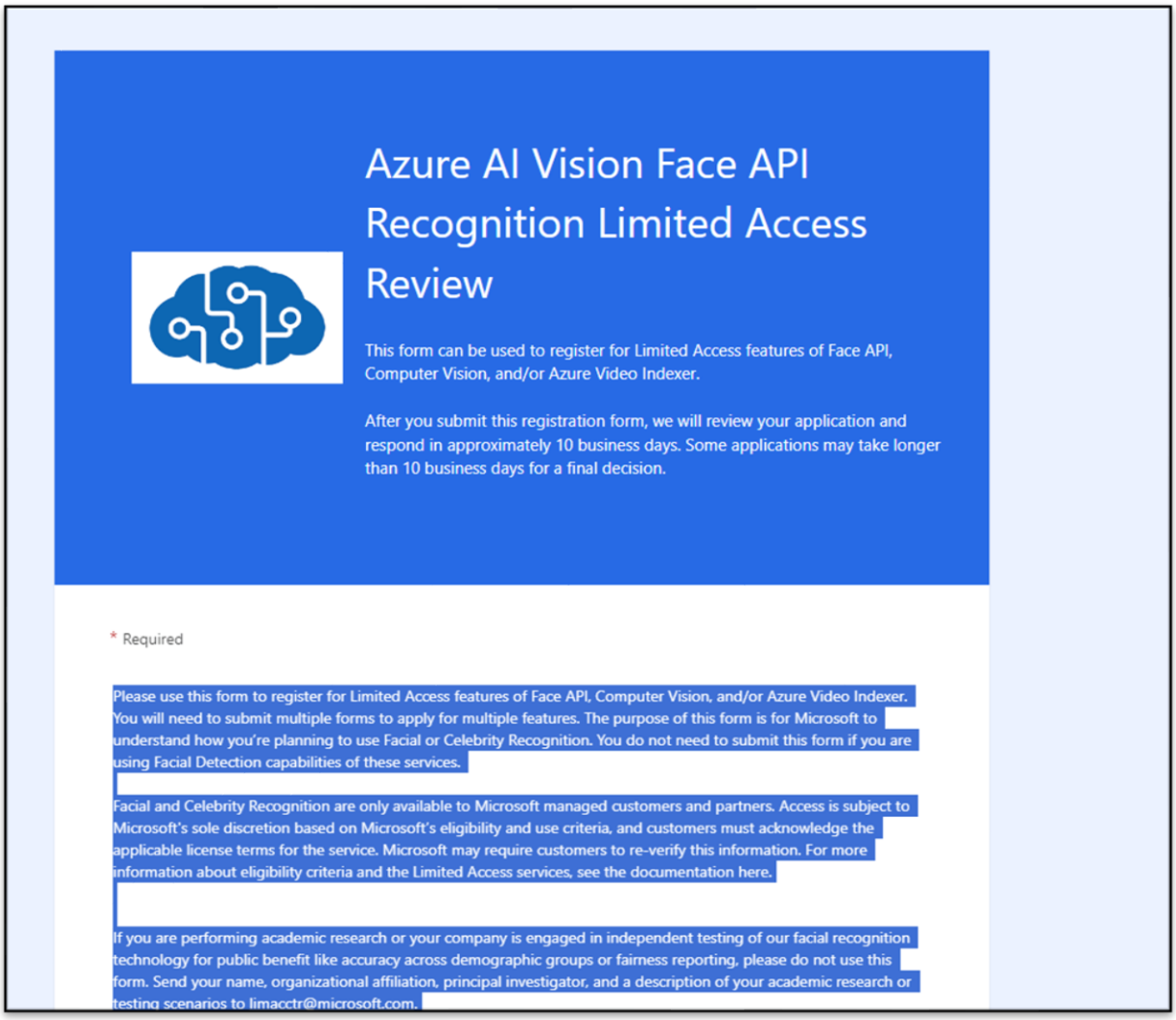
Challenges:

Azure's Celebrity Detection feature is a part of Azure Computer Vision that identifies famous individuals in images. However, as per Microsoft's policy, this feature is available only to Azure Managed users. To gain access, I need to register and apply for access, which can take up to 10 business days. I've already applied for this access and represented the code above. The output with the error handler indicates the need for this registration process.

"*An error occurred while trying to detect celebrities in the image. The Celebrity Recognition feature is not supported and requires registration. To apply for access, visit https://aka.ms/celebrityrecognition. This feature is available only to Microsoft managed customers and partners. If conducting academic research or independent testing for public benefit, contact limacctr@microsoft.com with project details."*

I've enrolled myself to delve deeper into these services for enhanced understanding. Alternatively, Azure Custom Vision could also be utilized for similar tasks.

Below is the screen shot of the registration:



Object Detection in Azure:

Azure's Object Detection feature is a part of Azure Computer Vision that identifies and locates objects within an image, providing the bounding box coordinates for each detected object. This functionality is useful for understanding the relationships between objects in an image and determining if there are multiple instances of the same object.

In addition to Azure's built-in capabilities, I've developed a custom function that leverages Azure Computer Vision to detect objects in images. This function not only identifies the objects but also saves the images with rectangles marking the detected objects. The processed images are stored in the **imagesout** folder in working directory.

Code:

```
# function to detect objects in an image, draw bounding boxes around them
def detect_objects_and_save(image_path, client, output_folder):
    # Start of object detection for a specific image
    print(f"\nDetecting objects in: {image_path}")
```

```
    # Open the image for analysis and drawing
    img = Image.open(image_path)
    draw = ImageDraw.Draw(img)
    font = ImageFont.truetype("arial.ttf", size=20)
    # Open the image file in binary-read mode
    with open(image_path, "rb") as image_stream:
        objects_result = client.detect_objects_in_stream(image_stream)
        # Check if any objects were detected
        if objects_result.objects:
            print("Objects detected:")
            for object in objects_result.objects:
                # For each detected object, draw a bounding box on the image
                box = object.rectangle
                draw.rectangle([box.x, box.y, box.x + box.w, box.y + box.h], outline="red",
width=3)
                # Text to display (object name and confidence)
                text = f"{object.object_property} ({object.confidence * 100:.1f}%)"
                # Draw the text in bold black inside the bounding box
                draw.text((box.x, box.y), text, fill="red", font=font, )
                # Print detected object details
                print(f"- {object.object_property} with confidence {object.confidence *
100:.2f}% at {box}")
            # Path for saving the modified image
            output_path = os.path.join(output_folder, os.path.basename(image_path))
            img.save(output_path)
            # Print image path
            print(f"Image with objects outlined saved to: {output_path}")
        else:
            # Print no object detected in image
            print("No objects detected.")

# Output folder path
output_folder = os.getcwd() + '/imageout'

# Iterate through a list of image files
for image_file in image_files:
    image = os.path.join(image_directory, image_file)
    # Detect objects in the image, draw bounding boxes
    detect_objects_and_save(image, computervision_client, output_folder)
```

Output:

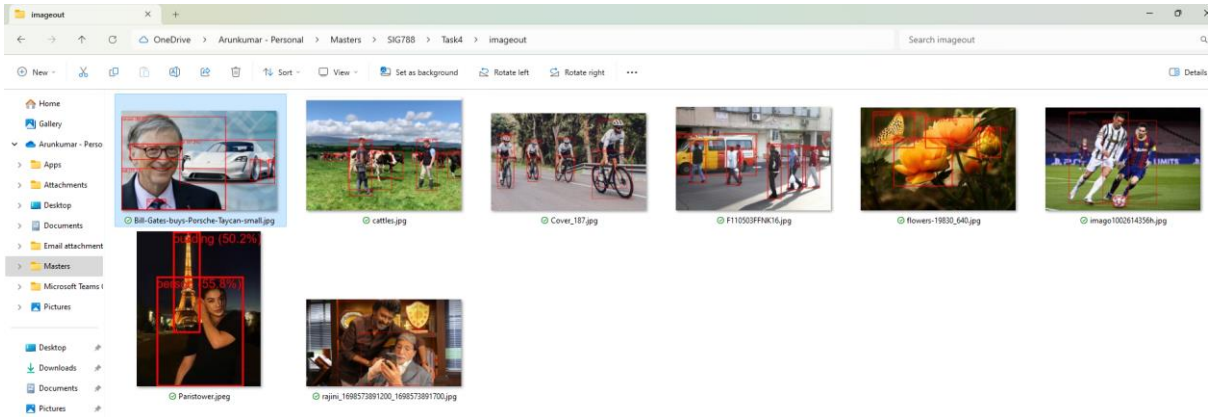
```
Detecting objects in: C:\Users\arunk\OneDrive\Masters\SIG788\Task4/Images\Bill-Gate
s-buys-Porsche-Taycan-small.jpg
Objects detected:
- Tire with confidence 65.40% at {'additional_properties': {}, 'x': 637, 'y': 241, 'w': 53, 'h':
105}
- Tire with confidence 52.30% at {'additional_properties': {}, 'x': 729, 'y': 242, 'w': 32, 'h':
96}
```

```
- tie with confidence 67.50% at {'additional_properties': {}, 'x': 126, 'y': 439, 'w': 94, 'h': 57}
- Glasses with confidence 69.40% at {'additional_properties': {}, 'x': 43, 'y': 165, 'w': 260, 'h': 79}
- car with confidence 87.20% at {'additional_properties': {}, 'x': 288, 'y': 146, 'w': 473, 'h': 216}
- suit with confidence 75.70% at {'additional_properties': {}, 'x': 0, 'y': 287, 'w': 516, 'h': 206}
- person with confidence 88.60% at {'additional_properties': {}, 'x': 0, 'y': 31, 'w': 517, 'h': 465}
Image with objects outlined saved to: C:\Users\arunk\OneDrive\Masters\SIG788\Task4/imageout\Bill-Gates-buys-Porsche-Taycan-small.jpg

Detecting objects in: C:\Users\arunk\OneDrive\Masters\SIG788\Task4/Images\Cover_187.jpg
Objects detected:
- helmet with confidence 52.40% at {'additional_properties': {}, 'x': 107, 'y': 191, 'w': 78, 'h': 78}
- bicycle with confidence 60.50% at {'additional_properties': {}, 'x': 343, 'y': 402, 'w': 93, 'h': 233}
- person with confidence 86.80% at {'additional_properties': {}, 'x': 607, 'y': 226, 'w': 180, 'h': 326}
- person with confidence 87.20% at {'additional_properties': {}, 'x': 995, 'y': 60, 'w': 272, 'h': 635}
- person with confidence 85.90% at {'additional_properties': {}, 'x': 53, 'y': 202, 'w': 164, 'h': 442}
- person with confidence 81.90% at {'additional_properties': {}, 'x': 333, 'y': 251, 'w': 130, 'h': 343}
- bicycle with confidence 78.10% at {'additional_properties': {}, 'x': 65, 'y': 403, 'w': 145, 'h': 299}
- bicycle with confidence 83.40% at {'additional_properties': {}, 'x': 598, 'y': 388, 'w': 177, 'h': 243}
- bicycle with confidence 87.10% at {'additional_properties': {}, 'x': 940, 'y': 345, 'w': 370, 'h': 447}
Image with objects outlined saved to: C:\Users\arunk\OneDrive\Masters\SIG788\Task4/imageout\Cover_187.jpg

Detecting objects in: C:\Users\arunk\OneDrive\Masters\SIG788\Task4/Images\F110503FFNK16.jpg
Objects detected:
- person with confidence 72.90% at {'additional_properties': {}, 'x': 66, 'y': 157, 'w': 76, 'h': 199}
- person with confidence 81.00% at {'additional_properties': {}, 'x': 237, 'y': 185, 'w': 59, 'h': 180}
- person with confidence 83.30% at {'additional_properties': {}, 'x': 428, 'y': 173, 'w': 65, 'h': 257}
- person with confidence 81.80% at {'additional_properties': {}, 'x': 524, 'y': 177, 'w': 70, 'h': 210}
```

```
- person with confidence 80.60% at {'additional_properties': {}, 'x': 605, 'y': 185, 'w': 68, '
h': 195}
- car with confidence 53.20% at {'additional_properties': {}, 'x': 0, 'y': 126, 'w': 376, 'h': 19
7}
Image with objects outlined saved to: C:\Users\arunk\OneDrive\Masters\SIG788\Task4/
imageout\F110503FFNK16.jpg

Detecting objects in: C:\Users\arunk\OneDrive\Masters\SIG788\Task4/Images\Paristow
er.jpeg
Objects detected:
- building with confidence 50.20% at {'additional_properties': {}, 'x': 59, 'y': 0, 'w': 44, 'h':
164}
- person with confidence 55.80% at {'additional_properties': {}, 'x': 32, 'y': 73, 'w': 141, 'h'
: 178}
Image with objects outlined saved to: C:\Users\arunk\OneDrive\Masters\SIG788\Task4/
imageout\Paristower.jpeg

Detecting objects in: C:\Users\arunk\OneDrive\Masters\SIG788\Task4/Images\cattles.jp
g
Objects detected:
- cow with confidence 54.50% at {'additional_properties': {}, 'x': 113, 'y': 375, 'w': 64, 'h':
42}
- cow with confidence 51.30% at {'additional_properties': {}, 'x': 192, 'y': 371, 'w': 74, 'h':
49}
- cow with confidence 58.30% at {'additional_properties': {}, 'x': 698, 'y': 366, 'w': 57, 'h':
73}
- mammal with confidence 52.10% at {'additional_properties': {}, 'x': 860, 'y': 363, 'w': 50
, 'h': 104}
- cow with confidence 72.40% at {'additional_properties': {}, 'x': 934, 'y': 354, 'w': 90, 'h':
187}
- person with confidence 58.30% at {'additional_properties': {}, 'x': 286, 'y': 279, 'w': 149,
'h': 374}
- cow with confidence 61.90% at {'additional_properties': {}, 'x': 277, 'y': 334, 'w': 288, 'h'
: 276}
- cow with confidence 64.20% at {'additional_properties': {}, 'x': 417, 'y': 357, 'w': 223, 'h'
: 181}
- person with confidence 81.60% at {'additional_properties': {}, 'x': 729, 'y': 274, 'w': 136,
'h': 344}
Image with objects outlined saved to: C:\Users\arunk\OneDrive\Masters\SIG788\Task4/
imageout\cattles.jpg

Detecting objects in: C:\Users\arunk\OneDrive\Masters\SIG788\Task4/Images\flowers-1
9830_640.jpg
Objects detected:
- lycaenid butterfly with confidence 68.60% at {'additional_properties': {}, 'x': 51, 'y': 17, '
w': 171, 'h': 150}
- flower with confidence 61.80% at {'additional_properties': {}, 'x': 267, 'y': 40, 'w': 226, '
h': 129}
```

```
- flower with confidence 53.50% at {'additional_properties': {}, 'x': 479, 'y': 80, 'w': 100, '
h': 128}
- flower with confidence 53.80% at {'additional_properties': {}, 'x': 137, 'y': 126, 'w': 242,
'h': 200}
Image with objects outlined saved to: C:\Users\arunk\OneDrive\Masters\SIG788\Task4/
imageout\flowers-19830_640.jpg

Detecting objects in: C:\Users\arunk\OneDrive\Masters\SIG788\Task4/Images\imago10
02614356h.jpg
Objects detected:
- soccer ball with confidence 79.10% at {'additional_properties': {}, 'x': 519, 'y': 686, 'w':
103, 'h': 88}
- person with confidence 91.80% at {'additional_properties': {}, 'x': 182, 'y': 34, 'w': 463, '
h': 710}
- person with confidence 94.10% at {'additional_properties': {}, 'x': 447, 'y': 101, 'w': 415,
'h': 651}
Image with objects outlined saved to: C:\Users\arunk\OneDrive\Masters\SIG788\Task4/
imageout\imago1002614356h.jpg

Detecting objects in: C:\Users\arunk\OneDrive\Masters\SIG788\Task4/Images\rajini_16
98573891200_1698573891700.jpg
Objects detected:
- person with confidence 87.10% at {'additional_properties': {}, 'x': 325, 'y': 11, 'w': 824, '
h': 877}
- person with confidence 80.10% at {'additional_properties': {}, 'x': 566, 'y': 340, 'w': 753,
'h': 554}
Image with objects outlined saved to: C:\Users\arunk\OneDrive\Masters\SIG788\Task4/
imageout\rajini_1698573891200_1698573891700.jpg
```

Output Images in Folder with Bounding box:



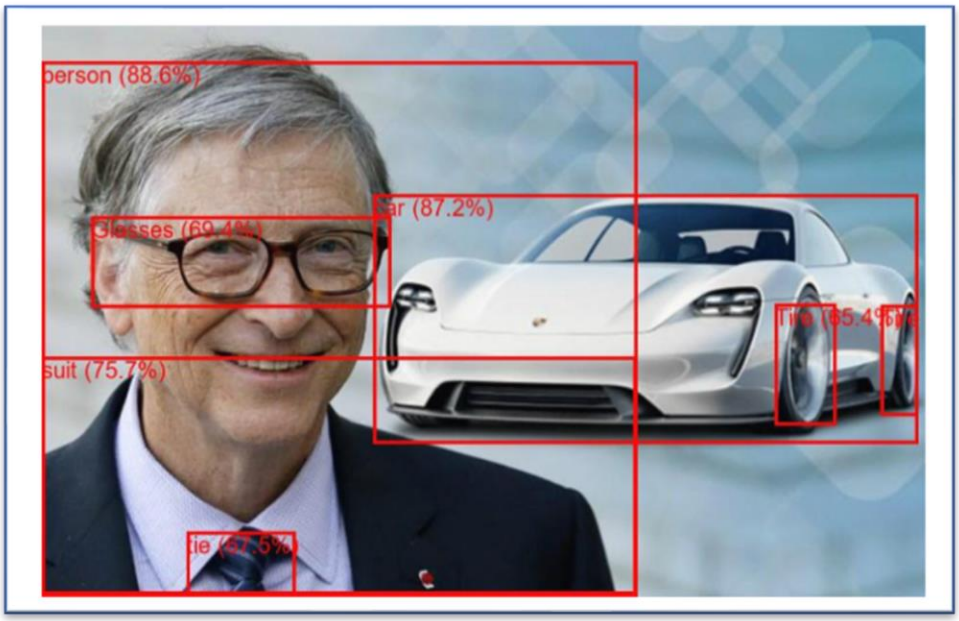
Printing the images with bounding Box:

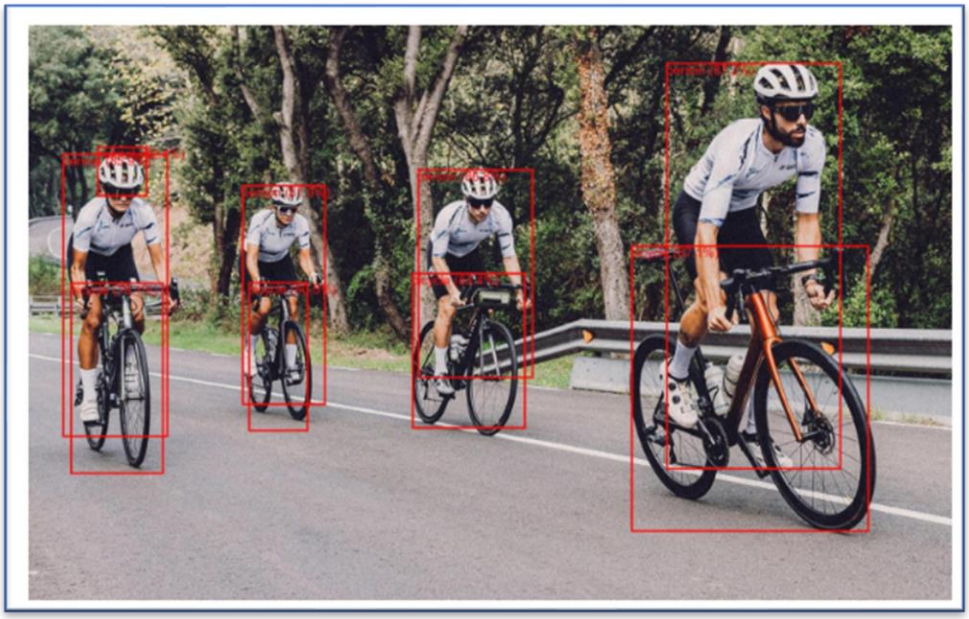
Code:

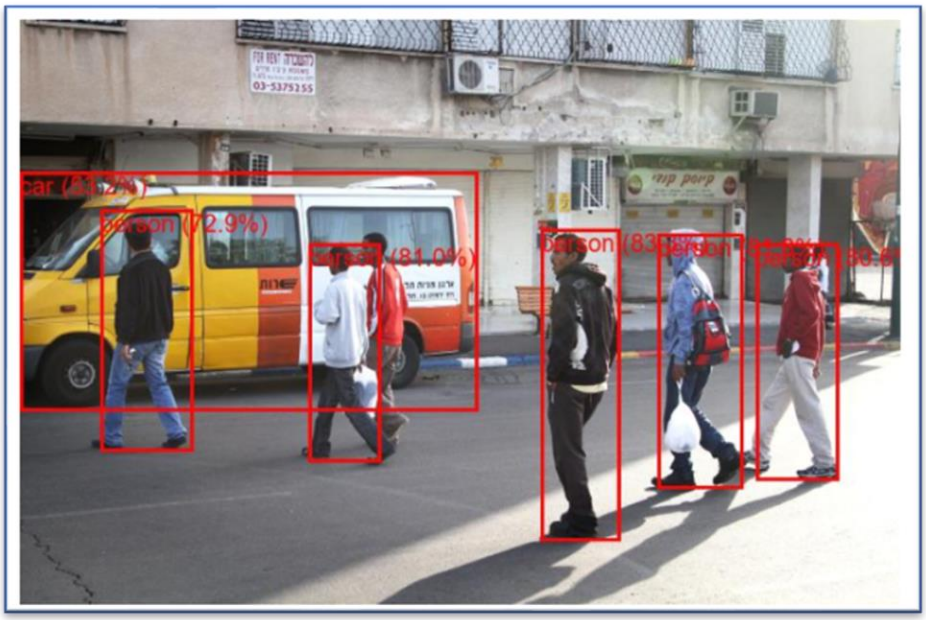
```
# Image Directory
image_directory = os.getcwd() + '/imageout'
```

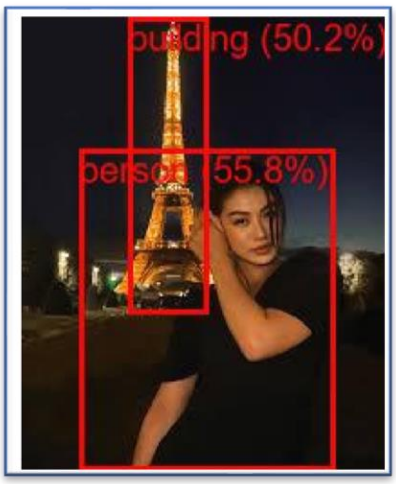
```
# List all files in the directory
files_in_directory = os.listdir(image_directory)

# Filter for images
image_files = [file for file in files_in_directory if file.lower().endswith(('.png', '.jpg',
'.jpeg', '.gif', '.bmp'))]

# Sort the images
image_files.sort()

# Iterate through the sorted image files
for image_file in image_files:
    # Open the image
    img = Image.open(os.path.join(image_directory, image_file))

    # Create a new figure
    plt.figure(figsize=(10, 10))  # Adjust the figure size as needed

    # Display the image
    plt.imshow(img)
    plt.axis('off')  # Turn off the axis

    # Show the plot
    plt.show()
```

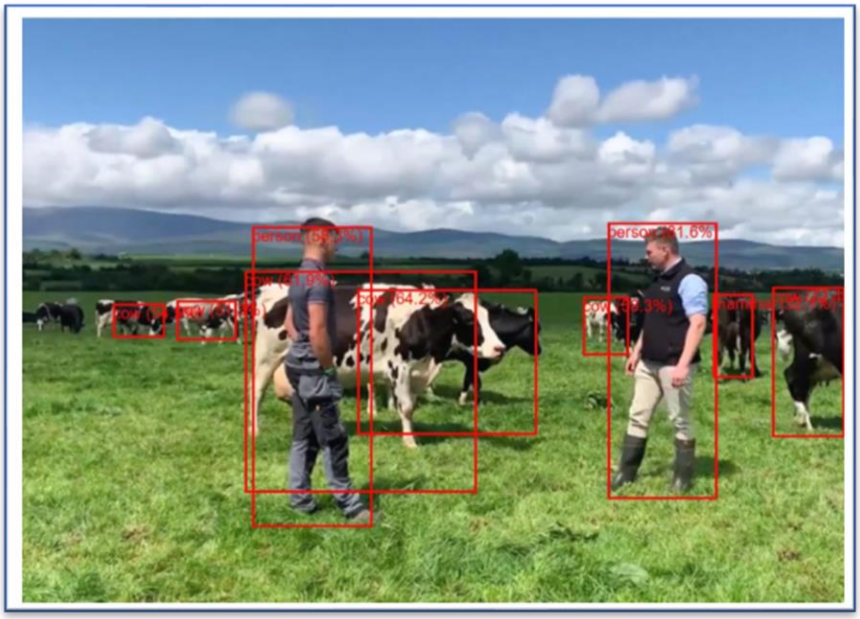
Output:

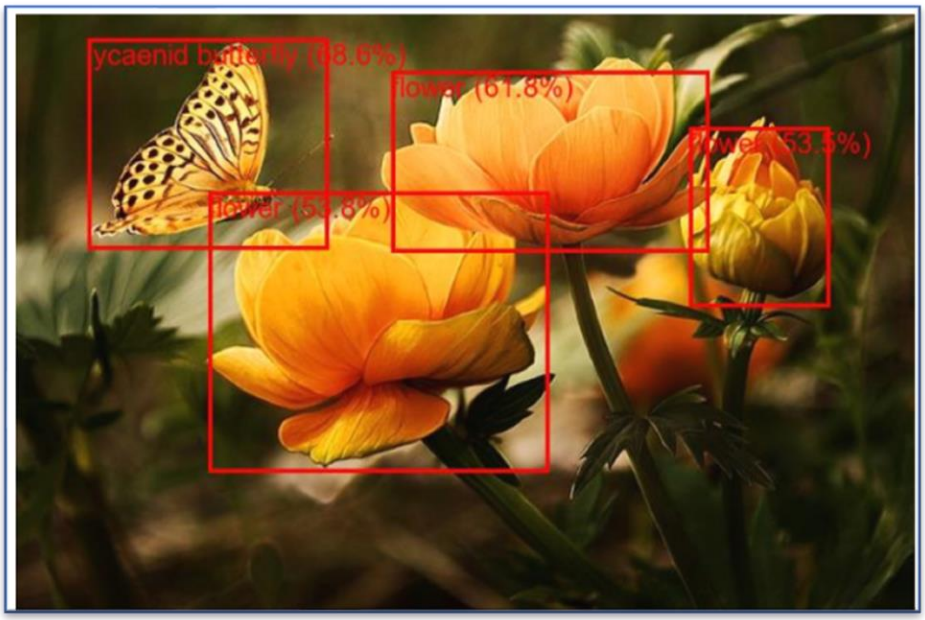


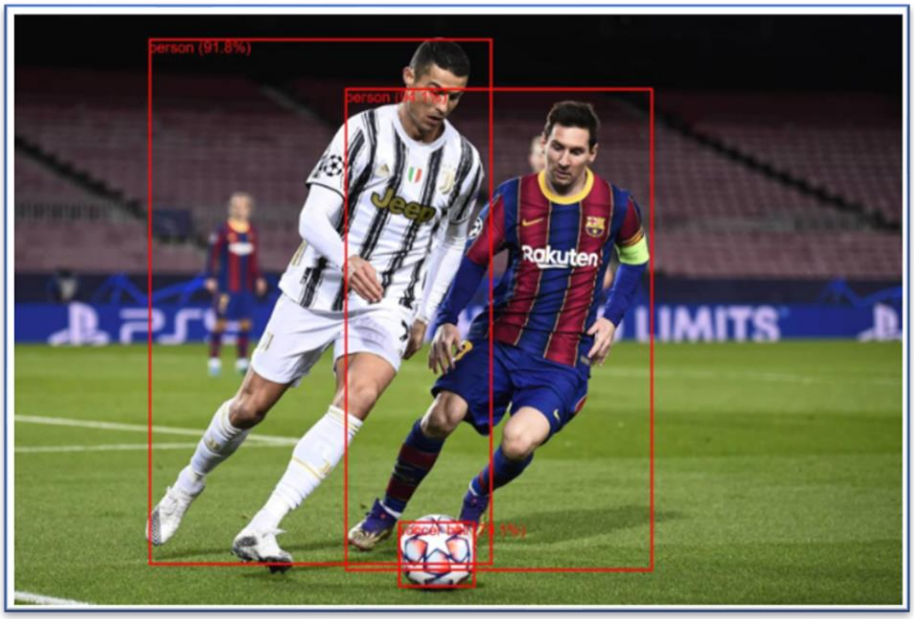





building (50.2%)
person (55.8%)




flower (61.8%)



Result:

Each image has been saved in the imageout folder with the detected object along with object name and its confidence from the model.

- Bill-Gates-buys-Porsche-Taycan-small.jpg
    - Detected: Tire (65.4%), Tire (52.3%), Tie (67.5%), Glasses (69.4%), Car (87.2%), Suit (75.7%), Person (88.6%).
- Cover_187.jpg
    - Detected: Helmet (52.4%), Bicycle (60.5%, 78.1%, 83.4%, 87.1%), Person (86.8%, 87.2%, 85.9%, 81.9%).

- F110503FFNK16.jpg
    - Detected: Person (72.9%, 81%, 83.3%, 81.8%, 80.6%), Car (53.2%).
- Paristower.jpeg
    - Detected: Building (50.2%), Person (55.8%).
- cattles.jpg
    - Detected: Cow (54.5%, 51.3%, 58.3%, 72.4%, 61.9%, 64.2%), Mammal (52.1%), Person (58.3%, 81.6%).
- flowers-19830_640.jpg
    - Detected: Lycaenid Butterfly (68.6%), Flower (61.8%, 53.5%, 53.8%).
- imago1002614356h.jpg
    - Detected: Soccer Ball (79.1%), Person (91.8%, 94.1%).
- rajini_1698573891200_1698573891700.jpg
    - Detected: Person (87.1%, 80.1%).

Each image has detected objects, by identifying various objects with their confidence levels and locations. The images are also saved in **imageout** with bounding boxes.

Next Steps:

- **Analysis and Improvement**: Evaluate object detection accuracy considering factors like lighting, resolution, and complexity & explore options of integrating Azure's Custom Vision for improved detection.
- **Expand Dataset**: Broaden list of images to test the API's robustness across multiple contexts.
- **Optimize Performance**: Monitor API performance and adjust request rate from free tier to Standard 1 tier as needed.
- **Enhance User Interface**: Develop an intuitive interface for image upload and result display using web application.

References:

- Microsoft Learn (no.date.) "**Quickstart: Image Analysis**" Available at: https://learn.microsoft.com/en-us/azure/ai-services/computer-vision/quickstarts-sdk/image-analysis-client-library?tabs=windows%2Cvisual-studio&pivots=programming-language-python

- Microsoft Learn (no.date.) "**Call the Image Analysis 3.2 API**" Available at: https://learn.microsoft.com/en-us/azure/ai-services/computer-vision/how-to/call-analyze-image?tabs=python

- Great Learning (2024) "**Case study on Computer Vision & Custom Vision using Python SDK**" Available at: https://olympus.mygreatlearning.com/courses/109553?module_id=747540

- Alam Smith (Sep 23, 2021) "**Azure Computer Vision using Python**" Available at: https://www.youtube.com/watch?v=3Zfcn1tsSwE&t=142s

- Geeksofgeeks (no.date.) "**Python PIL | ImageDraw.Draw.text()**" Available at: https://www.geeksforgeeks.org/python-pil-imagedraw-draw-text/