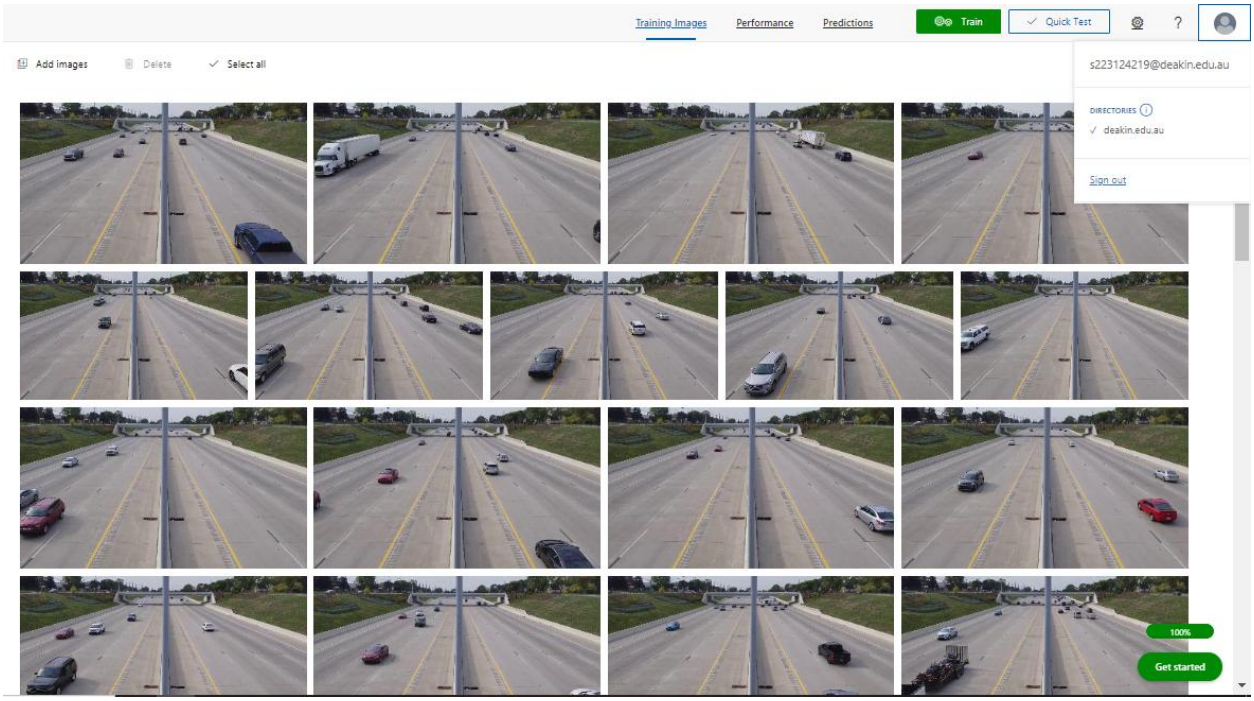
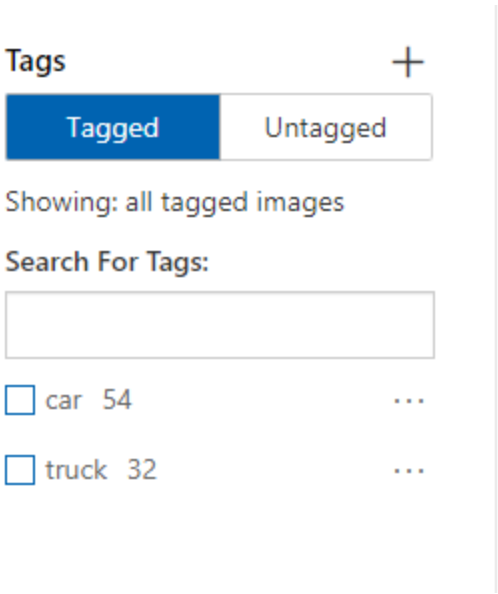
Car Detection Project

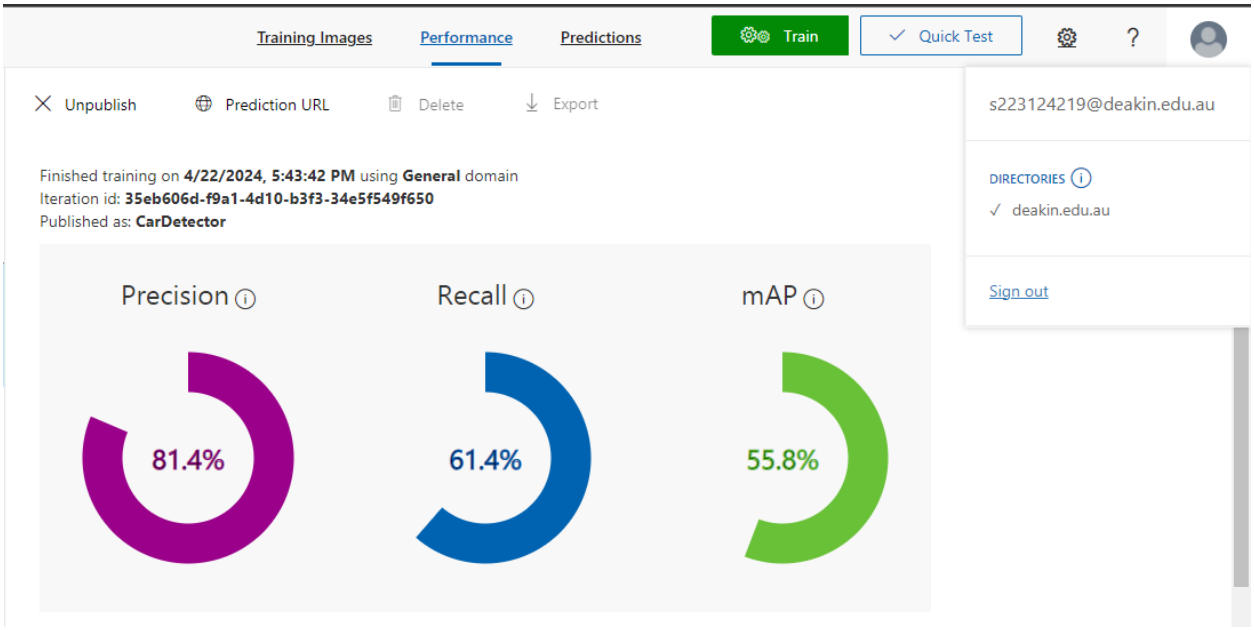
In this report I will be sharing the details of an object detection model created using azure custom vision service. The name of the model is car detector which uses the images of different cars on a highway to detect different cars and trucks.

Model Creation

In order to create an efficient model, I used the custom vision AI service and uploaded the required data of 50 images. I tagged these images so that the model can understand the different types of objects present in the images. Once the tagging was complete I trained the model and achieved a respectable score. Below are the screenshots of the data, tags and model performance.





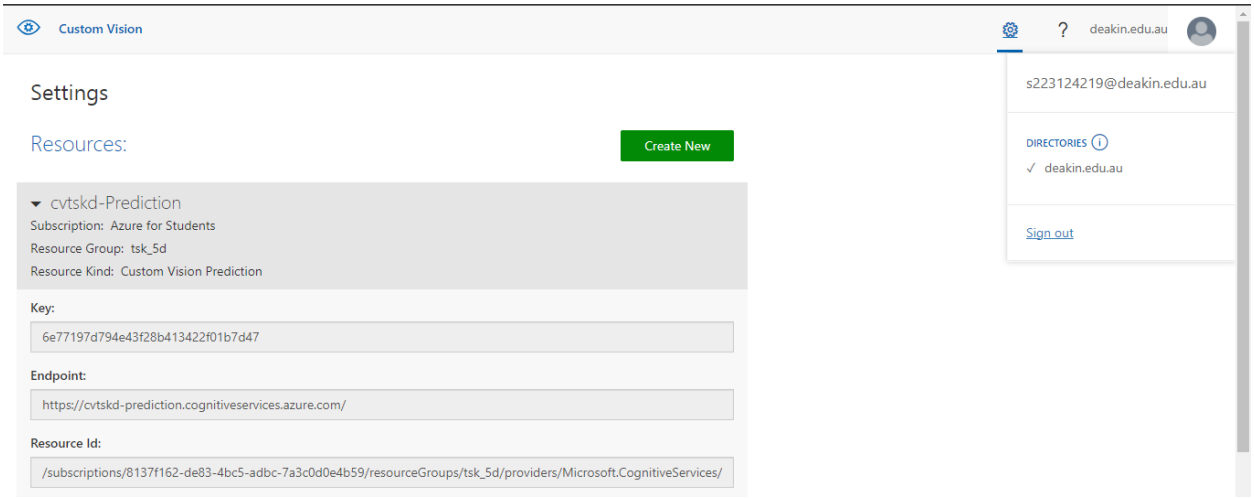


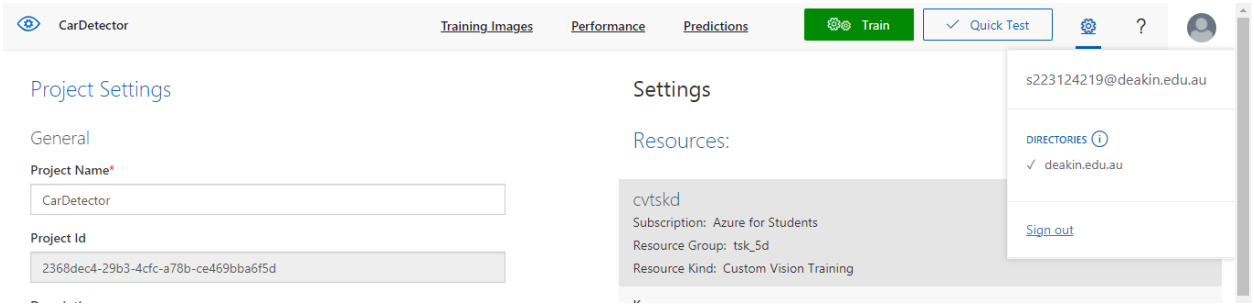
Utilizing the model using python code

Once the model training was completed, I imported all the required libraries and passed in the required credentials to gain access to my azure account.

```
from azure.cognitiveservices.vision.customvision.prediction import CustomVisionPredictionClient
from msrest.authentication import ApiKeyCredentials
import matplotlib.pyplot as plt
from PIL import Image, ImageDraw, ImageFont
import numpy as np
import os
from sort import Sort
import cv2
```

In order to gain access to azure account, I had to pass in the prediction endpoint, prediction key, project id and model name. This allows us to use the model in our code that we have trained using cognitive custom vision services.






```
prediction_endpoint = 'https://cvtskd-prediction.cognitiveservices.azure.com/'
prediction_key = '6e77197d794e43f28b413422f01b7d47'
project_id = '2368dec4-29b3-4cfc-a78b-ce469bba6f5d'
model_name = 'CarDetector'
```

```
# Authenticate a client for the training API
credentials = ApiKeyCredentials(in_headers={"Prediction-key": prediction_key})
prediction_client = CustomVisionPredictionClient(endpoint=prediction_endpoint, credentials=credentials)
```

Credentials variable is used to create authentication credentials for accessing the Custom Vision prediction service. It utilizes the ApiKeyCredentials class from the azure.core.credentials module and passes the prediction key as a header. Prediction_client endpoint variable stores the endpoint URL for accessing the Custom Vision prediction service. The endpoint URL points to the Custom Vision service provided by Azure Cognitive Services.

Test

Before applying the object tracking, I decided to first test the model with one of the images to see if it is accurately detecting different objects in one of the images.

In [62]:
```
image_file = 'C:/Users/Hp/Desktop/Engineering AI Sol/5.2d/extracted_frames/frame_0052.jpg'
image = Image.open(image_file)
with open(image_file, mode="rb") as image_data:
    results = prediction_client.detect_image(project_id, model_name, image_data)
```

In [63]:
```
for obj in results.predictions:
    if (obj.probability*100) > 30:
        print("Detected object:", obj.tag_name)
        print("Bounding box:", obj.bounding_box)
        print("Confidence:", obj.probability)
        print()
```

```
Detected object: car
Bounding box: {'additional_properties': {}, 'left': 0.24527396, 'top': 0.36195692, 'width': 0.07298212, 'height': 0.1285204}
Confidence: 0.89217323

Detected object: truck
Bounding box: {'additional_properties': {}, 'left': 0.19364765, 'top': 0.2934665, 'width': 0.061764598, 'height': 0.08511239}
Confidence: 0.8501856

Detected object: car
Bounding box: {'additional_properties': {}, 'left': 0.29169586, 'top': 0.2887986, 'width': 0.042362392, 'height': 0.05960539}
Confidence: 0.64105654

Detected object: truck
Bounding box: {'additional_properties': {}, 'left': 0.2937003, 'top': 0.2830731, 'width': 0.04151711, 'height': 0.060719848}
Confidence: 0.5979977
```

Prediction_client sends a request to the Custom Vision prediction service with the provided project ID, model name, and image data. The service processes the image using the specified model and returns the results, which includes information about the detected objects in the image, such as their location and labels. These results are then stored in the results variable for further processing and analysis. In the second block of code the results variable is used and the detected objects information is displayed.

Object detection

To detect the objects I wrote a function that is responsible for detecting objects in a frame using Azure Custom Vision.

The function detect_object, takes a single argument, frame, which represents a single image. The frame is converted to bytes using OpenCV', which encodes the frame as a JPEG image. The resulting encoded image is converted to bytes. This converts the image data into a format that can be sent over the network. The converted image bytes are used as input to call the Custom Vision API to detect objects. The predicition_client.detect_image() function is invoked with the required parameters. The results of the object detection are stored in the results.

The function then proceeds to extract the bounding box information for the detected objects from the results variable. It initializes two empty lists, detections and tag_names, to store the bounding box coordinates and tag names respectively for each detected object. It iterates over each prediction in the results. If the confidence probability of a prediction is above 40%, the bounding box coordinates are calculated using the position and dimensions of the bounding box in the frame. These coordinates are cast to integers. The calculated bounding box coordinates and the tag name are appended to the respective lists.

Finally, the function returns a NumPy array containing the detection and tag_names list. The bounding box coordinates and tag names for each detected object are returned as arrays, allowing for further processing and visualization.

Below is the screen shot of the function detect_object.

In [67]:
```python
#itteration 2
# Function to detect objects in a frame using Azure Custom Vision
def detect_objects(frame):
    # Convert frame to bytes
    _, img_encoded = cv2.imencode('.jpg', frame)
    image_bytes = img_encoded.tobytes()

    # Call the Custom Vision API to detect objects
    results = prediction_client.detect_image(project_id, model_name, image_bytes)

    # Extract bounding box information for detected objects
    detections = []
    tag_names = []  # List to store tag names corresponding to each detected object
    for prediction in results.predictions:
        if prediction.probability > 0.4:  # Only include predictions with a confidence above 40%
            left = int(prediction.bounding_box.left * frame.shape[1])
            top = int(prediction.bounding_box.top * frame.shape[0])
            right = int(left + prediction.bounding_box.width * frame.shape[1])
            bottom = int(top + prediction.bounding_box.height * frame.shape[0])
            detections.append([left, top, right, bottom])
            tag_names.append(prediction.tag_name)

    return np.array(detections), tag_names
```

Object Tracking

In order to track the objects, I decided to make a function that is responsible for tracking objects using the SORT (Simple Online and Realtime Tracking) algorithm. The function, track_objects(), take two arguments, detections and tracking. Detection represents the detected objects in a frame, typically in the form of bounding box coordinates and tracker is an object that implements the SORT algorithm for object tracking.

Inside the function, the tracker_update() method is called with the detections as its argument. This method updates the tracker with the newly detected objects and returns the tracked objects. The tracked objects are stored in the variable tracked_objects, which is then returned.

In [68]:
```python
# Function to track objects using SORT algorithm
def track_objects(detections, tracker):
    tracked_objects = tracker.update(detections)
    return tracked_objects
```

Object Counts

This function helps in visualizing the count of the objects per frame. It gets the object_count and frame which are displayed on to the frame using cv2.putText method.

In [69]:
```
# Function to visualize object counts
def visualize_object_counts(frame, object_count, tag_names):
    # Write object count and tag names on the frame
    cv2.putText(frame, f"Object Count: {object_count}", (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 0), 2)
```

Working of main()

The main function orchestrates the process of detecting, tracking, and visualizing objects within a video stream. It begins by initializing a video capture object, which opens the specified video file for frame-by-frame reading. Concurrently, an instance of the SORT (Simple Online and Realtime Tracking) algorithm is created to facilitate object tracking. Additionally, a VideoWriter object is established to write the processed frames into an output video file. Within the main loop, each frame of the input video is iterated over, incrementing the frame count and breaking the loop if no further frames are available. During each iteration, objects within the frame are detected using the Azure Custom Vision API, followed by tracking these objects using the SORT algorithm. Subsequently, the number of objects detected in each frame is counted, and visualized on the frame. Additionally, bounding boxes and unique IDs are drawn around the tracked objects to enhance visibility, these unique ids help identify and track the object in each frame. Finally, the processed frame is written into the output video file, and the frame is displayed. The program can be terminated by pressing the 'q' key, at which point the video capture and output objects are released, and all OpenCV windows are closed.

Below is the code for calling all the functionalities using main.

n [*]:
```
# Main function
def main():

    # Initialize video capture
    video_capture = cv2.VideoCapture('Highway Traffic.mp4')

    # Initialize SORT tracker
    tracker = Sort()

    # Define the codec and create VideoWriter object
    fourcc = cv2.VideoWriter_fourcc(*'XVID')
    out = cv2.VideoWriter('output_video.avi', fourcc, 20.0, (int(video_capture.get(3)), int(video_capture.get(4))))

    # Loop through frames
    frame_count = 0
    while video_capture.isOpened():
        ret, frame = video_capture.read()
        if not ret:
            break

        # Detect objects in the frame
        detections, tag_names = detect_objects(frame)

        # Track objects using SORT
        tracked_objects = track_objects(detections, tracker)

        # Count number of objects per frame
        object_count = len(tracked_objects)
```

```python
# Visualize object counts and tag names on the frame
visualize_object_counts(frame, object_count, tag_names)

# Draw bounding boxes and IDs for tracked objects
for obj in tracked_objects:
    left, top, right, bottom, track_id = obj
    # Ensure bounding box coordinates are integers
    left = int(left)
    top = int(top)
    right = int(right)
    bottom = int(bottom)
    cv2.rectangle(frame, (left, top), (right, bottom), (0, 255, 0), 2)
    cv2.putText(frame, f"ID: {track_id}", (left, top - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)

# Write the frame into the output video
out.write(frame)

# Display the frame
cv2.imshow('Video', frame)
```

```python
        # Break the loop if 'q' key is pressed
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    # Release the video capture object
    video_capture.release()
    out.release()
    cv2.destroyAllWindows()

if __name__ == "__main__":
    main()
```

Below is the screenshot of the detected objects, their tracking ID and the object count of one of the frames.