

# Deakin University

## SIG788- OnTrack Submission

### Task 4.2 D

#### Submitted by

Srileka Vijayakumar

223560038

Attempt #1

Date: 04/09/2023

**Target Grade:** Distinction

#### Task Details

### Task 4.2 D: Computer vision and custom vision

In this task, you need to complete the following part:

1) developing a program using Azure computer vision and custom vision model to detect vehicles (car, bus, bicycle and etc. (up to 3 vehicles)) and track them in a one-minute video. You can use Azure custom vision for this task. The application must be developed using Python language, and Azure custom vision can be used for this task. It is important to use a small dataset of 50-100 images to train the custom vision program to recognise different types of vehicles accurately

For this task you need to complete the following part:

- develop a near real-time vehicle tracking application. Your application should analyse video frames from a one-minute video using azure custom vision. In this assignment, you need to collect a dataset with different types of vehicles and label the images with corresponding vehicle type. Then you need to train your dataset using Azure custom vision and then deploy the model. You need to use the deployed model to track the vehicle in one-minute video. To decrease the cost of API usage, you need to send one frame per second to the API. The selected frames will be sent to the API to detect different vehicles. You will need to provide a demo (video) of how the model is developed and working (Max 3mins)

**Answer:**

To complete this task below are the execution steps followed in this exercise.

**Steps:****Step 1:** Data preparation

- Vehicle images of different vehicle types like car, bus, bike, bicycle etc are selected and categorized. Traffic video which contains different vehicles is also selected on which the model prediction will be run in later steps.

**Step 2:** Azure Cognitive Service Custom vision creation

- In order to execute all the steps below Azure Cognitive Service Custom Vision is created, which provides the keys and urls needed for this exercise.

**Step 3:** Custom vision project creation

- Custom vision project will be created where data will be loaded, tagged and objects detected with rectangle boxes for train set.

**Step 4:** Model training and publish

- Model is trained on the loaded data which has objects detected and labelled correctly. Upon satisfactory performance metrics, the model training iteration is published.

**Step 5:** Model prediction

- Using the generated prediction API, sample images are predicted for object detection and tags to ensure predictions are correct and API is working as expected.

**Step 6:** Traffic video processing and prediction

- Traffic video which has variety of vehicles is processed and frames are extracted as images. On every frame image, prediction API is invoked in order to get the objects detected and tagged correctly. Processed/predicted images are saved for video preparation.

**Step 7:** Predicted video preparation

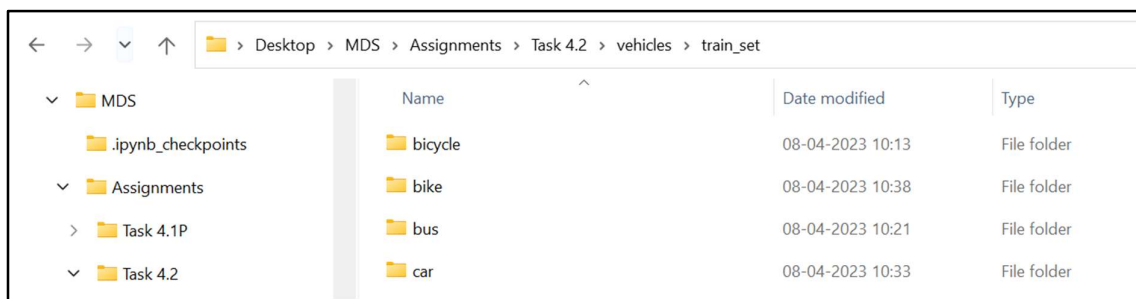
- From the processed/predicted frame images a video is prepared by appending the images one after other

**Step 8:** Sample processed frames from video

- A quick comparison on randomly selected frame images are shown along with the processed images side by side.

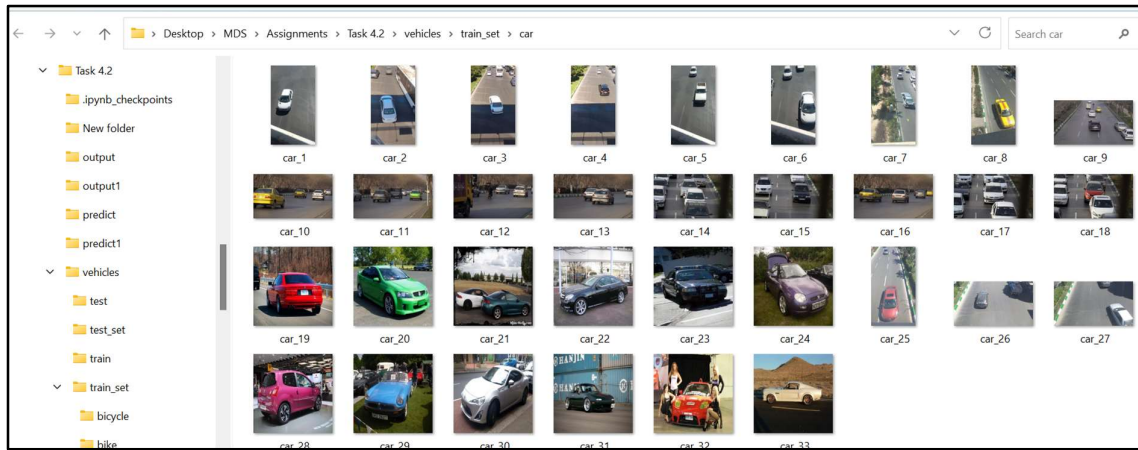
**Step 1:****Data preparation:**

For this task vehicle images are collected for different vehicle types – car, bus, bike, bicycle. Each of these folders contain ~15 images each. [1]



Desktop > MDS > Assignments > Task 4.2 > vehicles > train_set			
	Name	Date modified	Type
▼ MDS			
.ipyb_checkpoints	bicycle	08-04-2023 10:13	File folder
▼ Assignments	bike	08-04-2023 10:38	File folder
> Task 4.1P	bus	08-04-2023 10:21	File folder
▼ Task 4.2	car	08-04-2023 10:33	File folder

Sample images in car folder containing car images



## Step 2:

### Azure Cognitive Service – Custom vision creation

Login to Azure account. The resource group mds-rg is available. Now Cognitive service – custom vision is created as shown below. For this task the created custom vision is task42dcustom. During creation, the custom vision type is given as “Both” so there are 2 entries : “task2dcustom” for training and “task2dcustom-Prediction” for prediction.

Home > Cognitive Services

**Cognitive Services | Custom vision**

Search

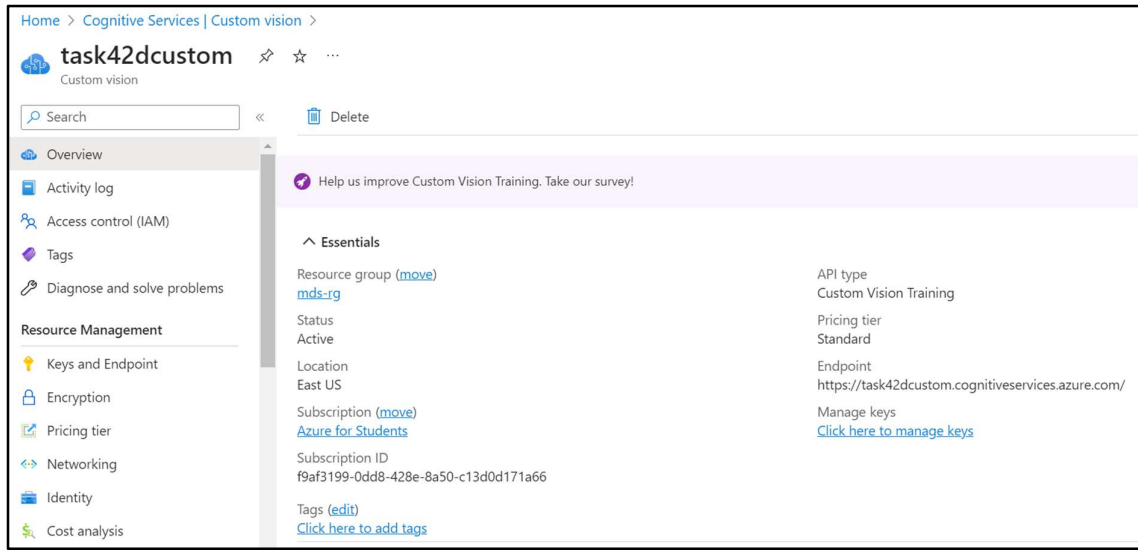
+ Create Manage deleted resources Manage view Refresh Export to CSV Open query Assign tags Delete

Filter for any field... Subscription equals all Type equals all Resource group equals all Add filter More (1)

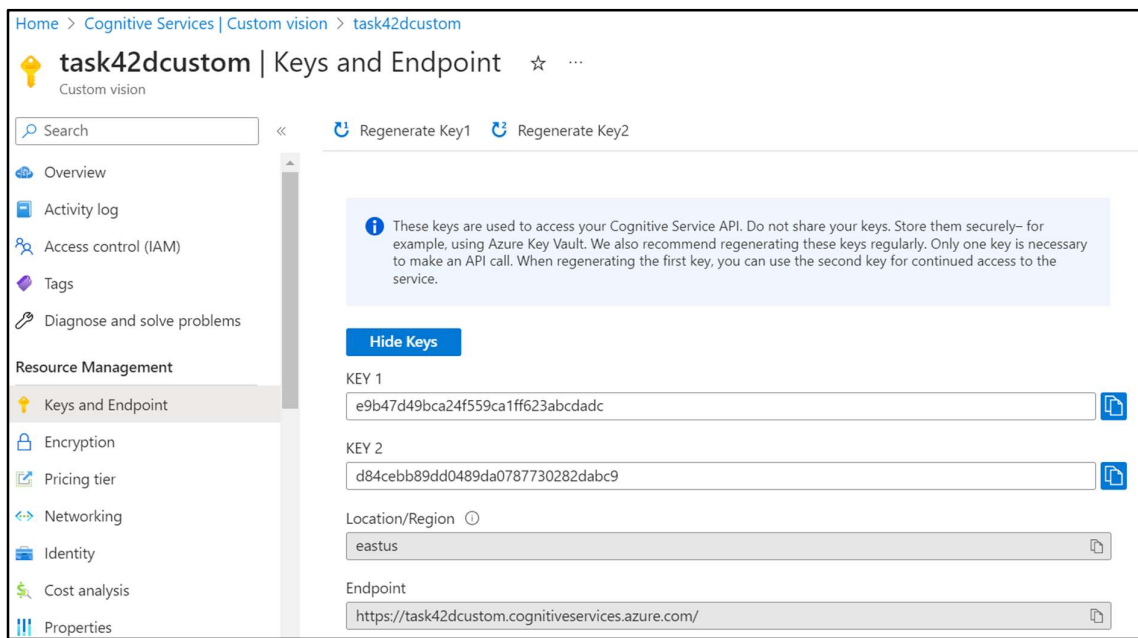
Showing 1 to 4 of 4 records.

Name	Kind	Location	Custom Domain	Pricing tier	Status
<input checked="" type="checkbox"/> task42dcustom	CustomVision.Training	East US	task42dcustom	S0	Succeeded
<input checked="" type="checkbox"/> task42dcustom-Prediction	CustomVision.Prediction	East US	task42dcustom-pr...	S0	Succeeded
<input type="checkbox"/> task42ddetect	CustomVision.Training	East US	task42ddetect	F0	Succeeded
<input type="checkbox"/> task42ddetect-Prediction	CustomVision.Prediction	East US	task42ddetect-pr...	F0	Succeeded

task42dcustom created as shown below



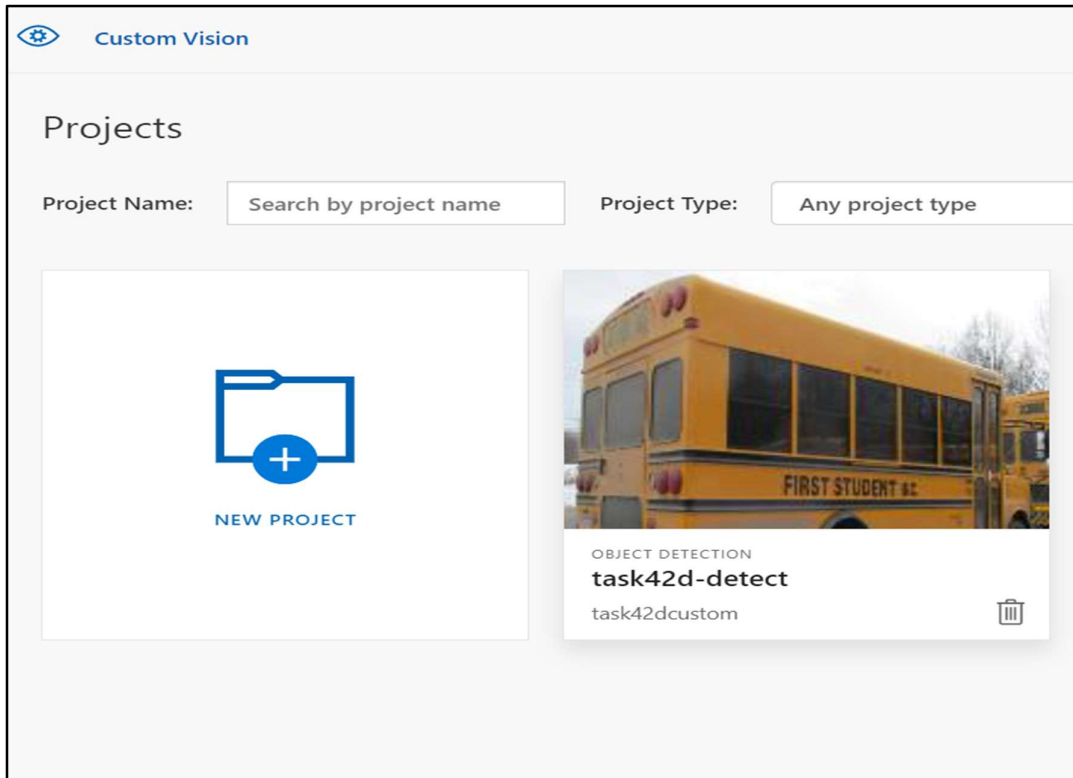
Keys and endpoint url created for task42dcustom



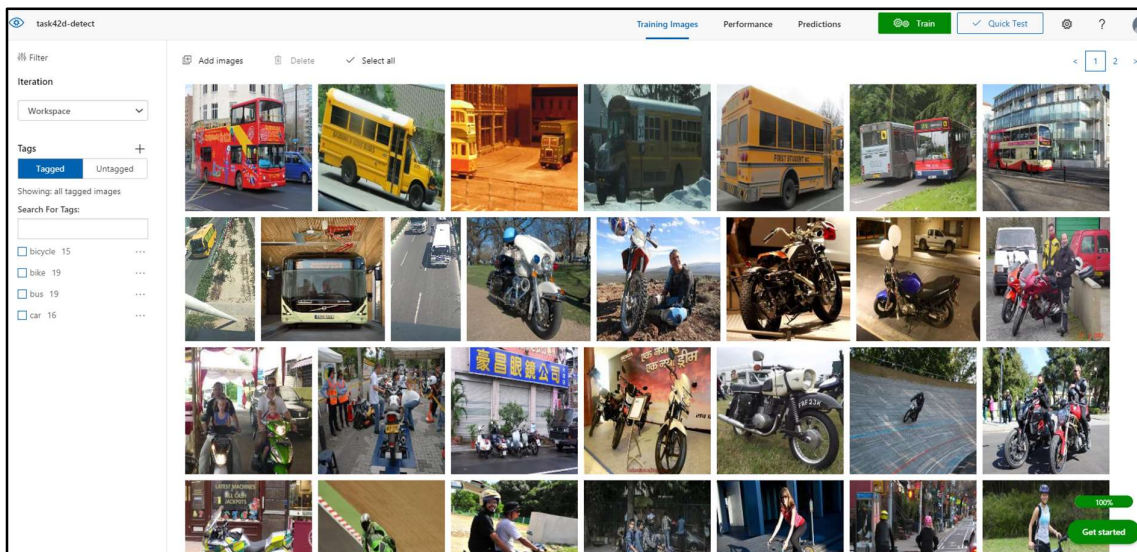
### Step 3:

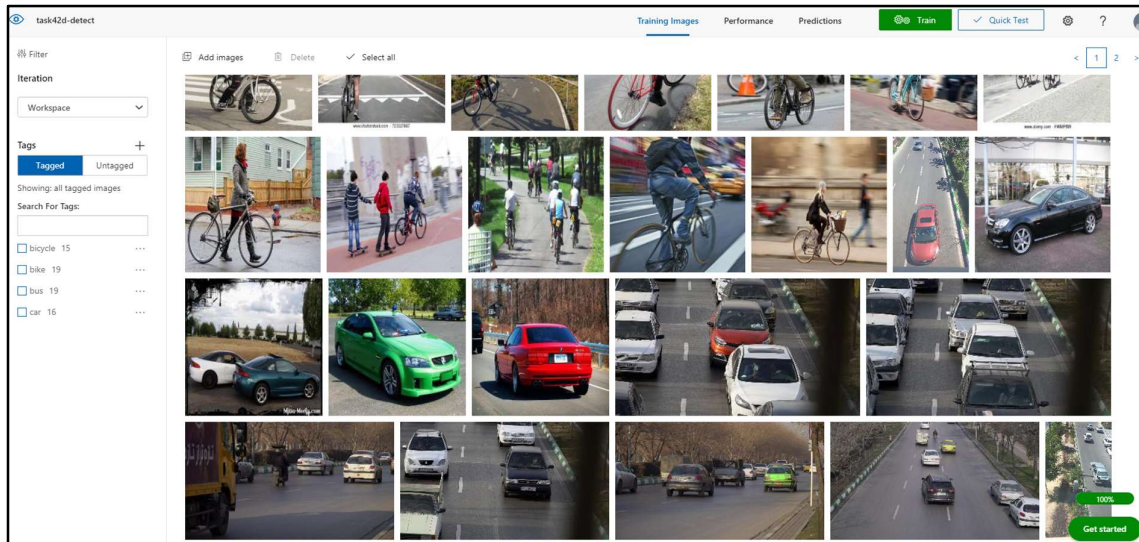
Custom vision project creation:

Go to <https://customvision.ai> and create a new project “task42d-detect”

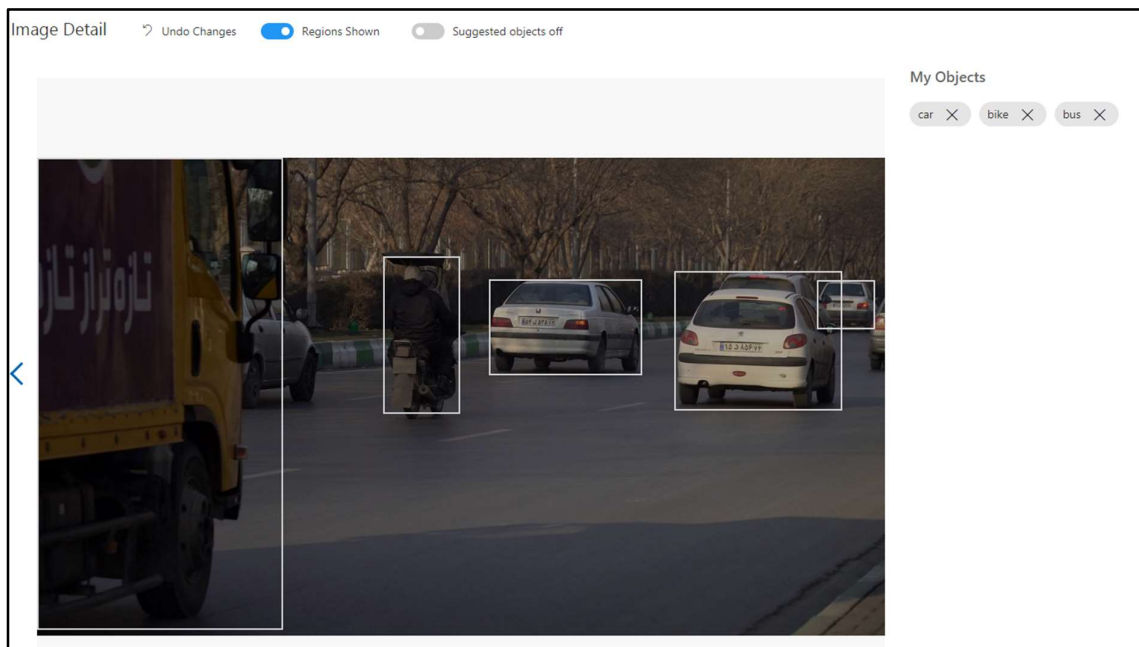


In the created custom vision project all the images prepared in step 1 are uploaded.



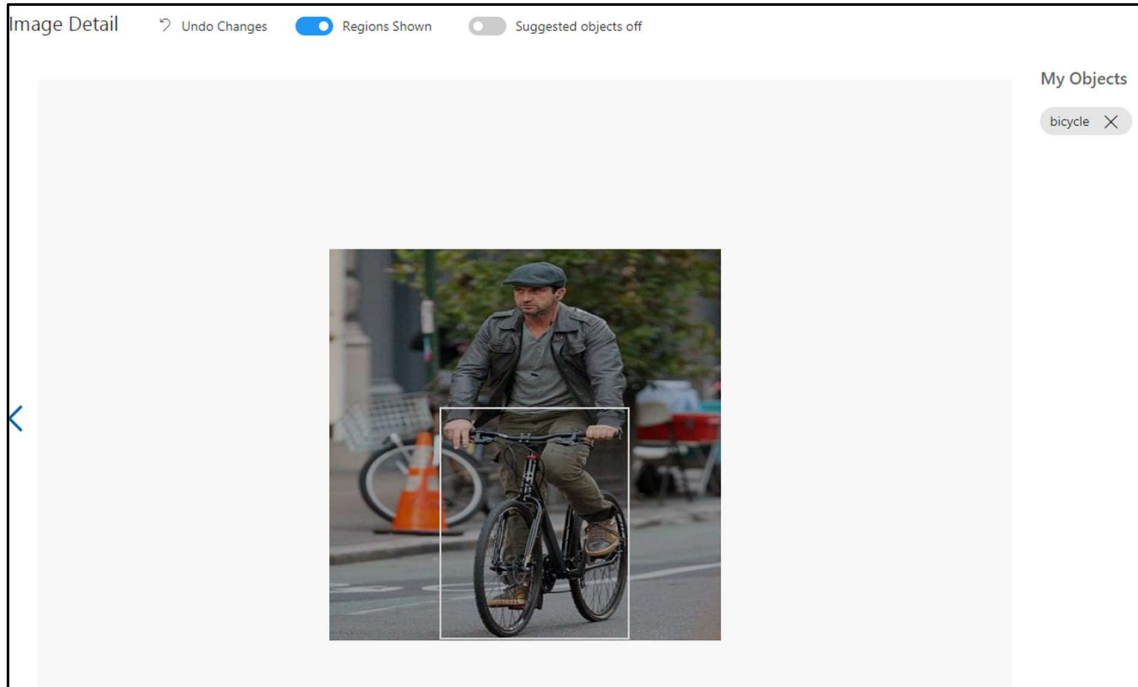


Each image is opened and objects are manually labelled. In the below example, car, bike and bus are manually marked and labelled.





Example showing bicycle image and labelled.

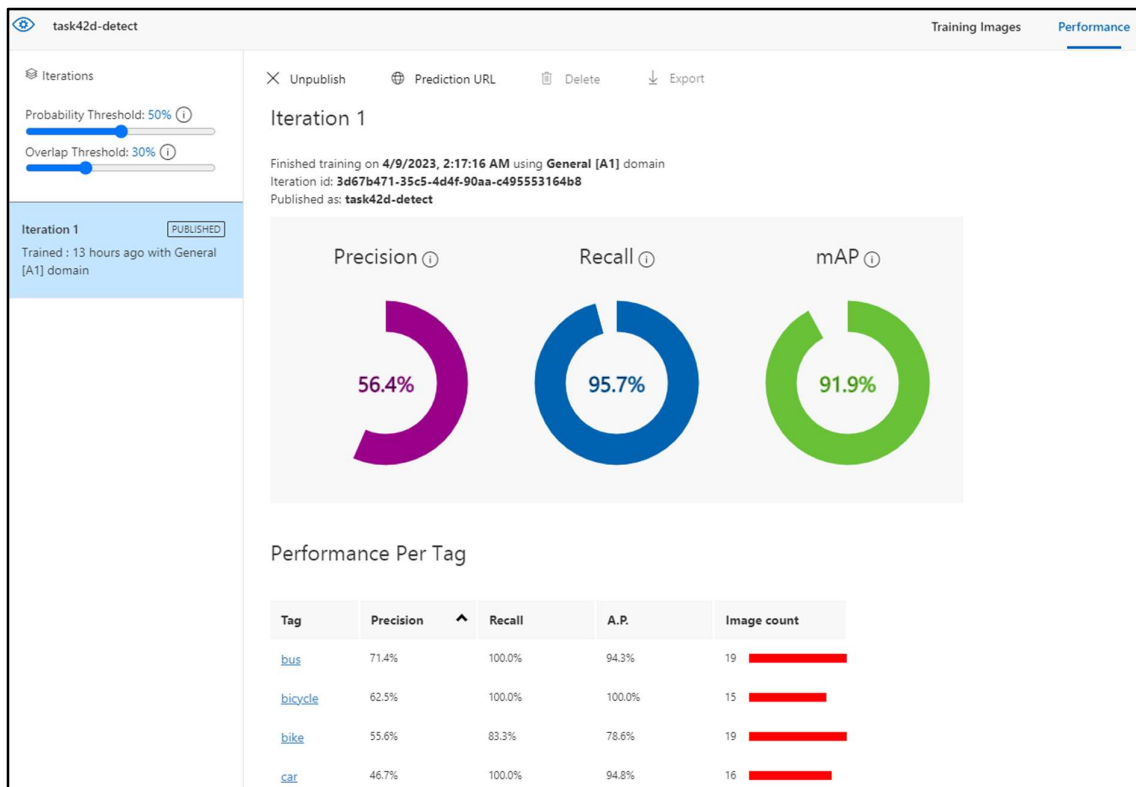


#### **Step 4:**

##### Model training and publish

Since the coordinates for the objects in the images are not already available, it was manually marked and labelled in the console as shown in previous steps. Model training is also done in the console. If the object coordinates are available as dictionary the model can be trained using python SDK on jupyter notebook.

After all the images are labelled, the model is trained. The below screen shot shows the performance metrics for all 4 categories. This is the first iteration and the same has been publish for future use.



The Prediction API is generated as shown below. The same will be used in Python SDK for prediction

task42d-detect

Training Images Performance Predictions Train Quick Test

Iterations

Probability Threshold: 50%  
Overlap Threshold: 30%

Iteration 1 **PUBLISHED**  
Trained : 13 hours ago with General [A1] domain

Unpublish Prediction URL Delete Export

How to use the Prediction API

If you have an image URL:

```
https://task42ddetect-prediction.cognitiveservices.azure.com/customvision/v3.0/Prediction/c9865fe4-d553-4bee-8daa-2606f4dcef5d/detect/iterations/task42d-detect/url
```

Set **Prediction-Key** Header to : 3ea938c591054fe28bab5059e8f4129d  
Set **Content-Type** Header to : application/json  
Set Body to : {"url": "https://example.com/image.png"}

If you have an image file:

```
https://task42ddetect-prediction.cognitiveservices.azure.com/customvision/v3.0/Prediction/c9865fe4-d553-4bee-8daa-2606f4dcef5d/detect/iterations/task42d-detect/url
```

Set **Prediction-Key** Header to : 3ea938c591054fe28bab5059e8f4129d  
Set **Content-Type** Header to : application/octet-stream  
Set Body to : <image file>

Got it!

Prediction API – Image URL:

<https://task42ddetect-prediction.cognitiveservices.azure.com/customvision/v3.0/Prediction/c9865fe4-d553-4bee-8daa-2606f4dcef5d/detect/iterations/task42d-detect/url>



Set **Prediction-Key** Header to : 3ea938c591054fe28bab5059e8f4129d

Set **Content-Type** Header to : application/json

Set Body to : {"Url": "https://example.com/image.png"}

Prediction API for image files:

<https://task42ddetect->

[prediction.cognitiveservices.azure.com/customvision/v3.0/Prediction/c9865fe4-d553-4bee-8daa-2606f4dcef5d/detect/iterations/task42d-detect/image](https://prediction.cognitiveservices.azure.com/customvision/v3.0/Prediction/c9865fe4-d553-4bee-8daa-2606f4dcef5d/detect/iterations/task42d-detect/image)

Set **Prediction-Key** Header to : 3ea938c591054fe28bab5059e8f4129d

Set **Content-Type** Header to : application/octet-stream

Set Body to : <image file>

### **Step 5:**

#### **Model prediction:**

Using the trained model, the vehicles in traffic video mp4 file will be predicted as detailed below.

Importing all needed software packages:

```
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
```

To see if the trained model predicts well, and prediction API works as expected, a sample image is taken and objects are detected using the prediction API above.

```
plt.figure(figsize=(10, 8))
base_image_location="C:/Users/srile/Desktop/MDS/Assignments/Task 4.2/vehicles/test_set"

path_to_file=os.path.join(base_image_location, "imtest1.jpeg")
img_data = cv2.imread(path_to_file)

plt.imshow(cv2.cvtColor(img_data,cv2.COLOR_BGR2RGB))

<matplotlib.image.AxesImage at 0x1bf3409dd60>
```



For the above image, API request has been posted as shown below and results are printed.

```
import cv2
import requests

__file__ = "C:/Users/srile/Desktop/MDS/Assignments/Task 4.2/vehicles/"
Predict_ENDPOINT = 'https://task42ddetect-prediction.cognitiveservices.azure.com/customvision/v3.0/Prediction/c9865fe4-d553-4bee-
prediction_key = '3ea938c591054fe28bab5059e8f4129d'
base_image_location = os.path.join(os.path.dirname(__file__), "test_set")

# Make request and process response
headers = dict()
headers['Prediction-Key'] = prediction_key
headers['Content-Type'] = 'application/octet-stream'

def predict_api_call(path_to_file):
    with open(path_to_file, "rb") as image_contents:
        response = requests.request('post', Predict_ENDPOINT, data=image_contents, headers=headers)

    if response.status_code == 200 or response.status_code == 201:

        if 'content-length' in response.headers and int(response.headers['content-length']) == 0:
            result = None
        elif 'content-type' in response.headers and isinstance(response.headers['content-type'], str):
            if 'application/json' in response.headers['content-type'].lower():
                result = response.json() if response.content else None
            elif 'image' in response.headers['content-type'].lower():
                result = response.content
        else:
            result = None
        img_data = cv2.imread(path_to_file)
        return img_data, result

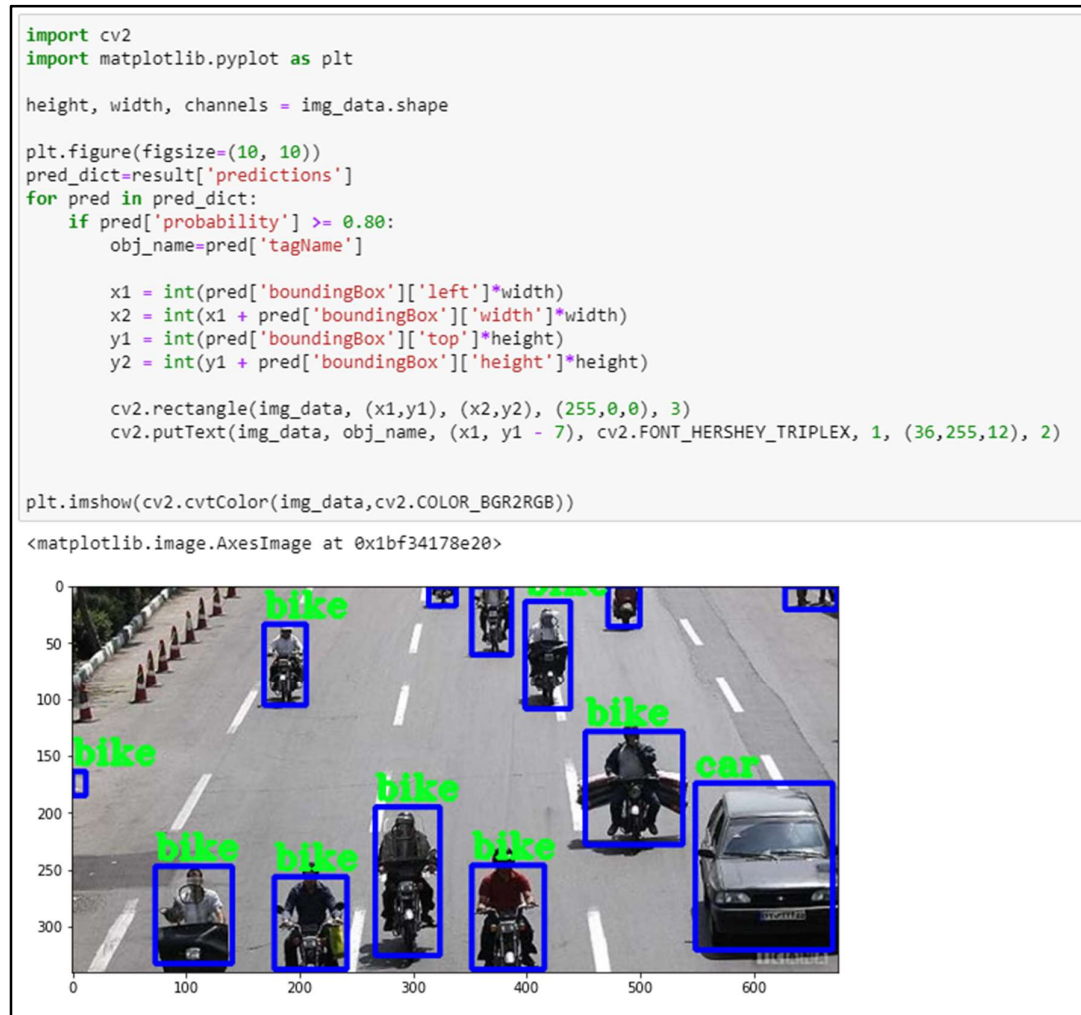
path_to_file = os.path.join(base_image_location, "imtest1.jpeg")
img_data, result = predict_api_call(path_to_file)

print(result)
```

Since there are many objects in the image, the returned result was very huge and only a part of it is shown here. Few bounding boxes are highlighted here.

```
{'id': '0f9c2287-ee48-4cf9-9c43-9cb0e8181c02', 'project': 'c9865fe4-d553-4bee-8daa-2606f4dcef5d', 'iteration': '3d67b471-35c5-4
d4f-90aa-c49553164b8', 'created': '2023-04-09T07:49:14.632Z', 'predictions': [{'probability': 0.017866684, 'tagId': '6cca2f4a-
217f-403b-a694-5b4fb6f948d4', 'tagName': 'bicycle', 'boundingBox': {'left': 0.26158795, 'top': 0.6784838, 'width': 0.101061225,
'height': 0.32151622}}, {'probability': 0.016559212, 'tagId': '6cca2f4a-217f-403b-a694-5b4fb6f948d4', 'tagName': 'bicycle', 'bo
undingBox': {'left': 0.5237889, 'top': 0.638536, 'width': 0.0942812, 'height': 0.36146402}}, {'probability': 0.014902815, 'tagI
d': '6cca2f4a-217f-403b-a694-5b4fb6f948d4', 'tagName': 'bicycle', 'boundingBox': {'left': 0.10871672, 'top': 0.66316485, 'widt
h': 0.09928384, 'height': 0.33683515}}, {'probability': 0.9989323, 'tagId': '6cceb4cb-1c3c-4f09-be09-7b28ce490e4b', 'tagName':
'bike', 'boundingBox': {'left': 0.39601392, 'top': 0.5756524, 'width': 0.084501445, 'height': 0.38287944}}, {'probability': 0.9
9856097, 'tagId': '6cceb4cb-1c3c-4f09-be09-7b28ce490e4b', 'tagName': 'bike', 'boundingBox': {'left': 0.7004357, 'top': 0.002907
3607, 'width': 0.041697502, 'height': 0.10689042}}, {'probability': 0.99816257, 'tagId': '6cceb4cb-1c3c-4f09-be09-7b28ce490e4
b', 'tagName': 'bike', 'boundingBox': {'left': 0.5918549, 'top': 0.0430259, 'width': 0.05994135, 'height': 0.28080067}}, {'prob
ability': 0.9977324, 'tagId': '6cceb4cb-1c3c-4f09-be09-7b28ce490e4b', 'tagName': 'bike', 'boundingBox': {'left': 0.2648895, 'to
p': 0.75319296, 'width': 0.09574357, 'height': 0.24000728}}, {'probability': 0.99696535, 'tagId': '6cceb4cb-1c3c-4f09-be09-7b28
ce490e4b', 'tagName': 'bike', 'boundingBox': {'left': 0.6701348, 'top': 0.37989244, 'width': 0.12804526, 'height': 0.2937512}},
{'probability': 0.9968341, 'tagId': '6cceb4cb-1c3c-4f09-be09-7b28ce490e4b', 'tagName': 'bike', 'boundingBox': {'left': 0.521932
54, 'top': 0.7250895, 'width': 0.09553951, 'height': 0.26929152}}, {'probability': 0.995834, 'tagId': '6cceb4cb-1c3c-4f09-be09-
7b28ce490e4b', 'tagName': 'bike', 'boundingBox': {'left': 0.5216773, 'top': 0.0047403374, 'width': 0.052119017, 'height': 0.178
25711}}, {'probability': 0.99313265, 'tagId': '6cceb4cb-1c3c-4f09-be09-7b28ce490e4b', 'tagName': 'bike', 'boundingBox': {'lef
t': 0.25162718, 'top': 0.10095411, 'width': 0.055263937, 'height': 0.21343812}}, {'probability': 0.9901579, 'tagId': '6cceb4cb-
1c3c-4f09-be09-7b28ce490e4b', 'tagName': 'bike', 'boundingBox': {'left': 0.10890307, 'top': 0.7279777, 'width': 0.10124006, 'he
ight': 0.25288248}}, {'probability': 0.98529124, 'tagId': '6cceb4cb-1c3c-4f09-be09-7b28ce490e4b', 'tagName': 'bike', 'boundingB
ox': {'left': 0.46588323, 'top': 0.0033258174, 'width': 0.03634277, 'height': 0.05283951}}, {'probability': 0.9644027, 'tagId':
'6cceb4cb-1c3c-4f09-be09-7b28ce490e4b', 'tagName': 'bike', 'boundingBox': {'left': 0.93093747, 'top': 0.0034114148, 'width': 0.
06906253, 'height': 0.059095}}, {'probability': 0.8894729, 'tagId': '6cceb4cb-1c3c-4f09-be09-7b28ce490e4b', 'tagName': 'bike',
'boundingBox': {'left': 0.0, 'top': 0.48354664, 'width': 0.018800061, 'height': 0.06394401}}, {'probability': 0.5262505, 'tagI
d': '6cceb4cb-1c3c-4f09-be09-7b28ce490e4b', 'tagName': 'bike', 'boundingBox': {'left': 0.0, 'top': 0.42993534, 'width': 0.01823
7783, 'height': 0.13443065}}, {'probability': 0.49651518, 'tagId': '6cceb4cb-1c3c-4f09-be09-7b28ce490e4b', 'tagName': 'bike',
'boundingBox': {'left': 0.4532227, 'top': 0.0005111769, 'width': 0.013584644, 'height': 0.040825937}}, {'probability': 0.36276
5, 'tagId': '6cceb4cb-1c3c-4f09-be09-7b28ce490e4b', 'tagName': 'bike', 'boundingBox': {'left': 0.9617134, 'top': 0.00220263, 'w
idth': 0.0329175, 'height': 0.08749496}}, {'probability': 0.2688136, 'tagId': '6cceb4cb-1c3c-4f09-be09-7b28ce490e4b', 'tagNam
e': 'bike', 'boundingBox': {'left': 0.5203966, 'top': 0.010884519, 'width': 0.055795074, 'height': 0.0589421}}, {'probability':
0.114361934, 'tagId': '6cceb4cb-1c3c-4f09-be09-7b28ce490e4b', 'tagName': 'bike', 'boundingBox': {'left': 0.45826247, 'top': 0.0
014174754, 'width': 0.020183504, 'height': 0.047882795}}, {'probability': 0.1092585, 'tagId': '6cceb4cb-1c3c-4f09-be09-7b28ce49
```

The same image is plotted again with the results predicted above. i.e tags are added as text, rectangle is drawn using the bounding box coordinates, etc The results are applied on the image only if the probability score is > 80% to ensure objects are labelled and highlighted correctly.



As can be seen above, the model has predicted correctly all the objects and tagged them accurately. The bounding box coordinates also fit on the object precisely. Hence the model really works well and also the prediction API call is successful.

### Step 6:

#### Traffic video processing and prediction

Below are the 2 videos that are selected for this task. All below steps are explained using example1 for clarity. In the final output step, the predicted videos of both the examples are attached. [1]

Example video 1:



Example video 2:



The video file is read and split into frames. The frames are extracted for every 0.1 second. Due to resource cost and performance issues, a short video was used for this task and around 101 images were collected from the video frames.

All the extracted images are written to “/output1/” folder and image names will be “imagex.jpg” where x is 1 through 101.

```
import cv2
vidcap = cv2.VideoCapture('road_traffic.mp4')

image_list=[]
def getFrame(sec):

    vidcap.set(cv2.CAP_PROP_POS_MSEC,sec*1000)
    hasFrames,image = vidcap.read()

    if hasFrames:
        cv2.imwrite("./output1/image"+str(count)+".jpg", image) # save frame as JPG file
        image_list.append(image)
    return hasFrames

sec = 0
frameRate = 0.1 #//it will capture image in each 0.1 second
count=1
success = getFrame(sec)
while success:
    count = count + 1
    sec = sec + frameRate
    sec = round(sec, 2)
    success = getFrame(sec)

len(image_list)

101
```

All the images extracted in /output1/ folder are then processed one by one. For every image predict\_api\_call was invoked. This API call will detect all the objects in the image and return with tag names, prediction probability score and bounding box coordinates.

```
base_image_location="C:/Users/srile/Desktop/MDS/Assignments/Task 4.2/output1/"

cnt=0
for image_num in range(1,len(image_list)+1):
    path_to_file=os.path.join(base_image_location, "image{}.jpg".format(image_num))
    img_data, result = predict_api_call(path_to_file)

    write_pred_image(img_data,result,image_num)
    cnt+=1

print("Totally {} images were written in prediction folder".format(cnt))

Totally 101 images were written in prediction folder
```

To apply the API call results i.e tag name, rectangle using bounding box coordinates etc, a function write\_pred\_image was defined as below. This function will apply the results on the image if the probability score is > 80% to ensure the objects are correctly tagged and rectangle



was drawn precisely. After applying the results this function will also save the image in “/predict1/” folder with image names as “imagex.jpg” where x will be 1 through 101.

```
def write_pred_image(img_data,result,n):
    height, width, channels = img_data.shape

    pred_dict=result['predictions']
    for pred in pred_dict:
        if pred['probability'] >= 0.80:
            obj_name=pred['tagName']

            x1 = int(pred['boundingBox']['left']*width)
            x2 = int(x1 + pred['boundingBox']['width']*width)
            y1 = int(pred['boundingBox']['top']*height)
            y2 = int(y1 + pred['boundingBox']['height']*height)

            cv2.rectangle(img_data, (x1,y1), (x2,y2), (255,0,0), 3)
            cv2.putText(img_data, obj_name, (x1, y1 - 7), cv2.FONT_HERSHEY_TRIPLEX, 1, (36,255,12), 2)

    cv2.imwrite("./predict1/image"+str(n)+".jpg", img_data)
```

As shown in previous screen shot, this function has written totally 101 predicted images to “/predict1/” folder.

### **Step 7:**

#### **Predicted video preparation**

From all the predicted images which is present in “/predict1/” folder, where it has all images with objects tagged and rectangles drawn on them using bounding box coordinates, a new video is prepared by appending all the images one after the other. The below code shows this preparation of new video.

```
import cv2
import numpy as np
import glob

img_folder='C:/Users/srile/Desktop/MDS/Assignments/Task 4.2/predict1'
images_ = [img for img in os.listdir(img_folder)
            if img.endswith(".jpg") or
            img.endswith(".jpeg") or
            img.endswith(".png")]
frame = cv2.imread(os.path.join(img_folder, images_[0]))

height, width, layers = frame.shape
video_name='road_traffic_out.avi'
video = cv2.VideoWriter(video_name, 0, 1, (width, height))

# Appending the images to the video one by one
for image in images_:
    video.write(cv2.imread(os.path.join(img_folder, image)))

# Deallocating memories taken for window creation
cv2.destroyAllWindows()
video.release() # releasing the video generated
```

The new predicted video is 'road\_traffic\_out.avi' and it is attached here.



road\_traffic\_out.avi

For example 2, the predicted video is attached here,



test\_video\_out.avi

### **Step 8:**

#### **Sample processed frames from video**

The below code shows the original image extracted from the video and the corresponding processed image, where all the prediction results are applied. Randomly 5 frames are selected and corresponding processed image is shown side-by-side for quick reference.

```
def show_images(n):

    orig_folder='C:/Users/srile/Desktop/MDS/Assignments/Task 4.2/output1'
    pred_folder='C:/Users/srile/Desktop/MDS/Assignments/Task 4.2/predict1'

    plt.figure(figsize=(20, 20))

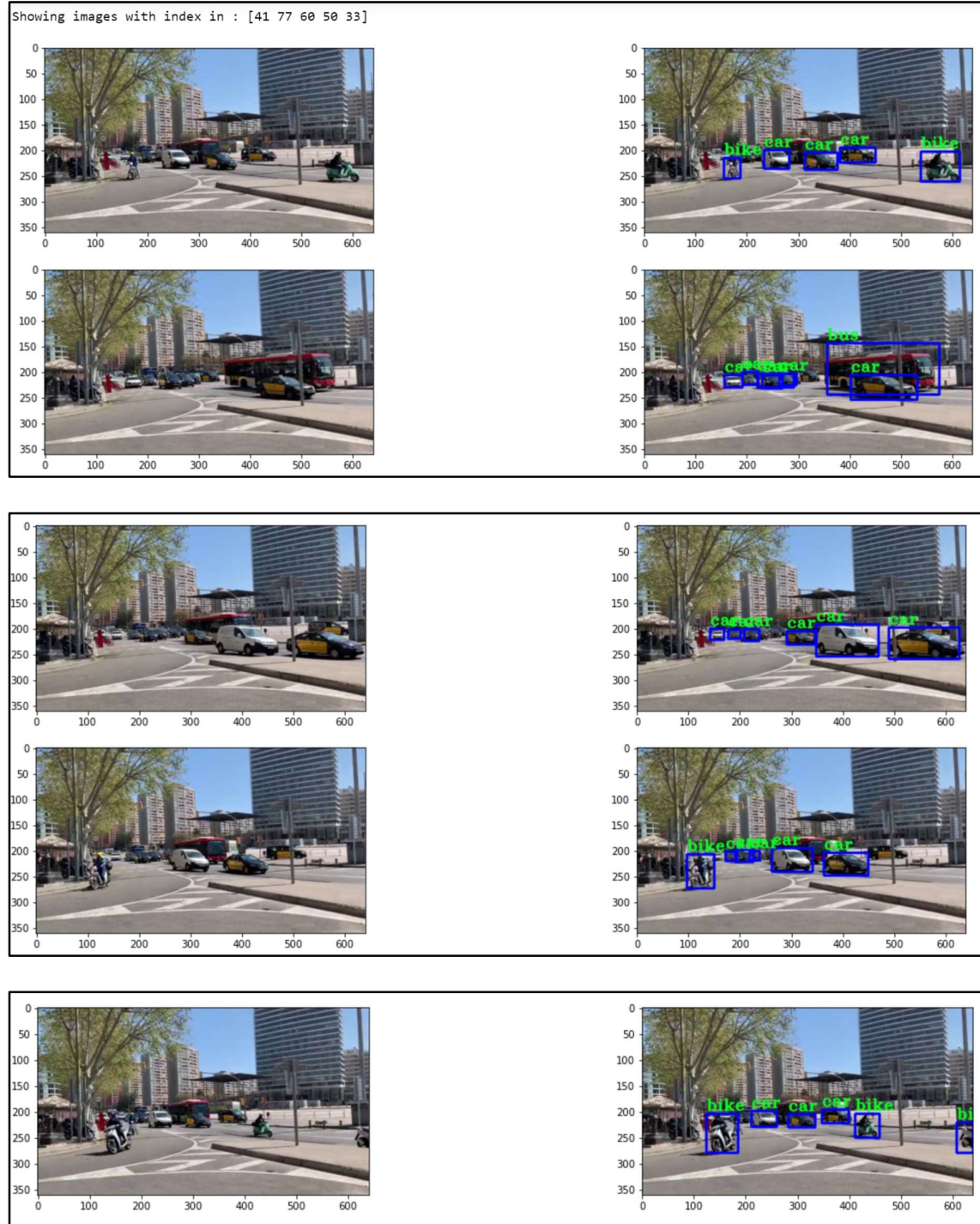
    rand = np.random.randint(0, 101, n) # Generating n random numbers from total number of images
    print("Showing images with index in : {}".format(rand))

    for cnt,j in enumerate(rand):
        orig_data = cv2.imread(os.path.join(orig_folder, "image{}.jpg".format(j)))
        pred_data = cv2.imread(os.path.join(pred_folder, "image{}.jpg".format(j)))
        plt.subplot(n, 2, (cnt*2)+1)
        plt.imshow(cv2.cvtColor(orig_data,cv2.COLOR_BGR2RGB))
        plt.subplot(n, 2, (cnt*2)+2)
        plt.imshow(cv2.cvtColor(pred_data,cv2.COLOR_BGR2RGB))

    show_images(5)
```



5 random sample frames and the corresponding processed images:



As shown above the different vehicles are detected and tagged correctly with bounding box coordinates. From the above picture car, bike and bus are detected correctly.

**References:**

[1] *Github id MaryamBoneh* [online] Available at <https://github.com/MaryamBoneh/Vehicle-Detection/tree/main/Dataset> - Vehicle datasets train images, test images, videos

- <https://b2n.ir/vehicleDataset>

[2] *Microsoft manual 01/07/2023* [Online] Available at <https://learn.microsoft.com/en-us/azure/cognitive-services/custom-vision-service/use-prediction-api>

[3] *Great learning material* Available at [https://olympus.mygreatlearning.com/courses/91262/files/8006730?module\\_item\\_id=4026267](https://olympus.mygreatlearning.com/courses/91262/files/8006730?module_item_id=4026267)