SIG788

Engineering AI solutions

Distinction Task 5

Arunkumar Balaraman

S223919051

Contents

Target = Distinction

Object Detection using Azure's Custom Vision API

Introduction

Project uses Microsoft Azure's Custom Vision API to detect and categorize objects in vehicle tracking in a video.

Objective:

The goal is to create a program that can accurately identify objects the objects in video for vehicle tracking using Azure's Custom Vision SDK. The program will highlight detected objects, demonstrating its practical application in real-world scenarios like vehicle tracking analysis.

Approach:

The project involves below steps:

- **Video Selection**: Choose three diverse vehicle videos under different conditions.
- **Image Extraction**: Extract images at one-second intervals from these videos, capturing vehicles from various angles and distances.
- **Azure Project Setup**: Create an Azure Custom Vision project with object detection and multi-label prediction capabilities.
- **Image Tagging**: Accurately tag 120 images with bounding boxes around each vehicle using the parquet bounding box information for the selected video.
- **Model Training**: Train the model using tagged images.
- **Model Evaluation**: Evaluate the model's performance using precision, recall, and mAP metrics.
- **Model Deployment**: Deploy the trained model efficiently, considering the expected frequency and volume of video processing.
- **Model Validation**: Test the model's generalization capabilities and real-world applicability with new videos not included in the training set.

Azure Custom Vision:

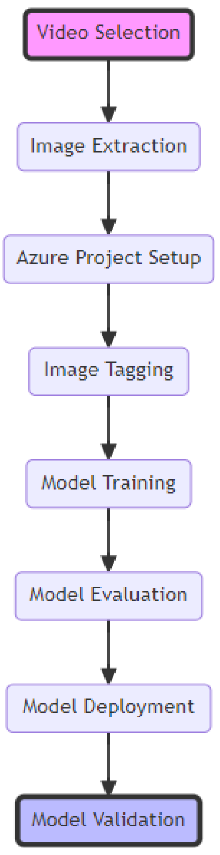
Azure **Custom Vision** is a part of Azure AI Services which allows to build, deploy, and improve own image classifiers.

Key features include:

- **Customization to user Scenario**: can set model to perceive a particular object for a use case.
- **Intuitive Model Creation**: Easily build image identifier model using the simple interface.

- **Flexible Deployment**: Run AI Custom Vision in the cloud or on the edge in containers.
- **Built-in Security**: Rely on enterprise-grade security and privacy for data and any trained models.
- **Achieve Accuracy Without Complexity**: Start training computer vision model by simply uploading and labelling a few images.
- **Accelerate Model Creation**: A user-friendly interface walks through developing and deploying custom computer vision models.
- **Deploy Anywhere:** From the Cloud to the Edge: Run models wherever we need them and according to unique scenario and requirements.

Flowchart:

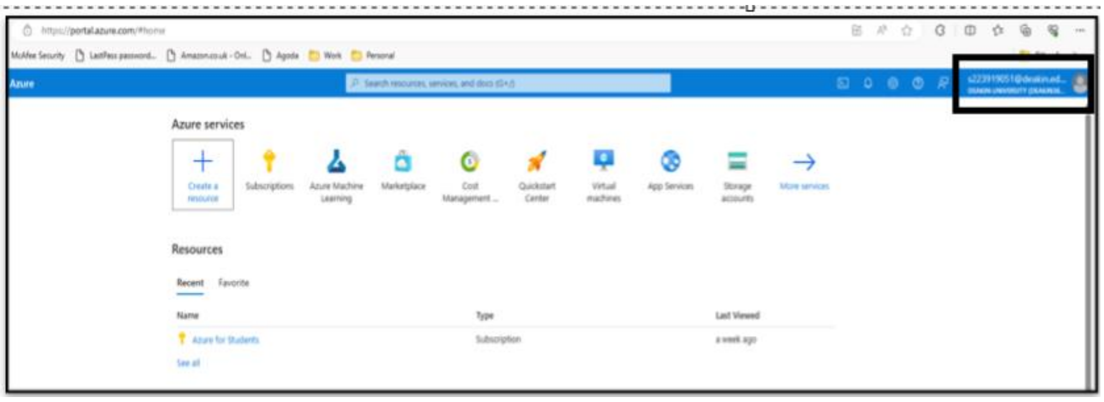


Creating Azure Custom Vision:

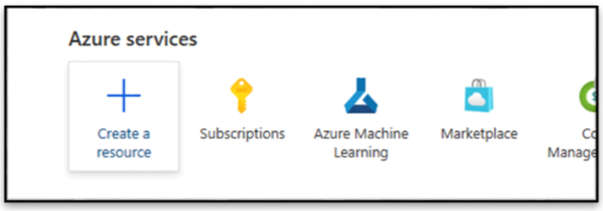
- **Sign in to Azure Portal:** Azure Portal and sign in with Microsoft account Deakin credentials.

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

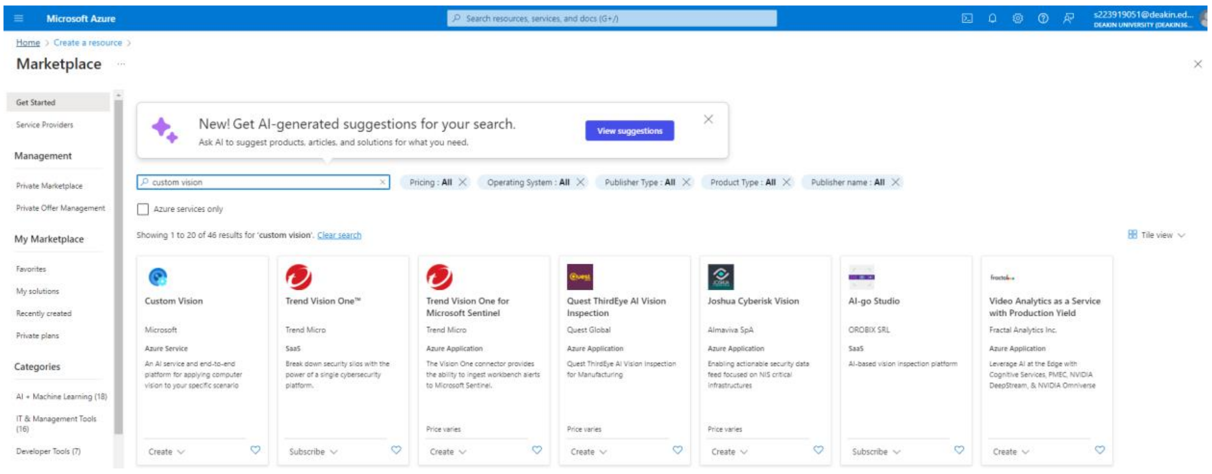
  Landing Page:



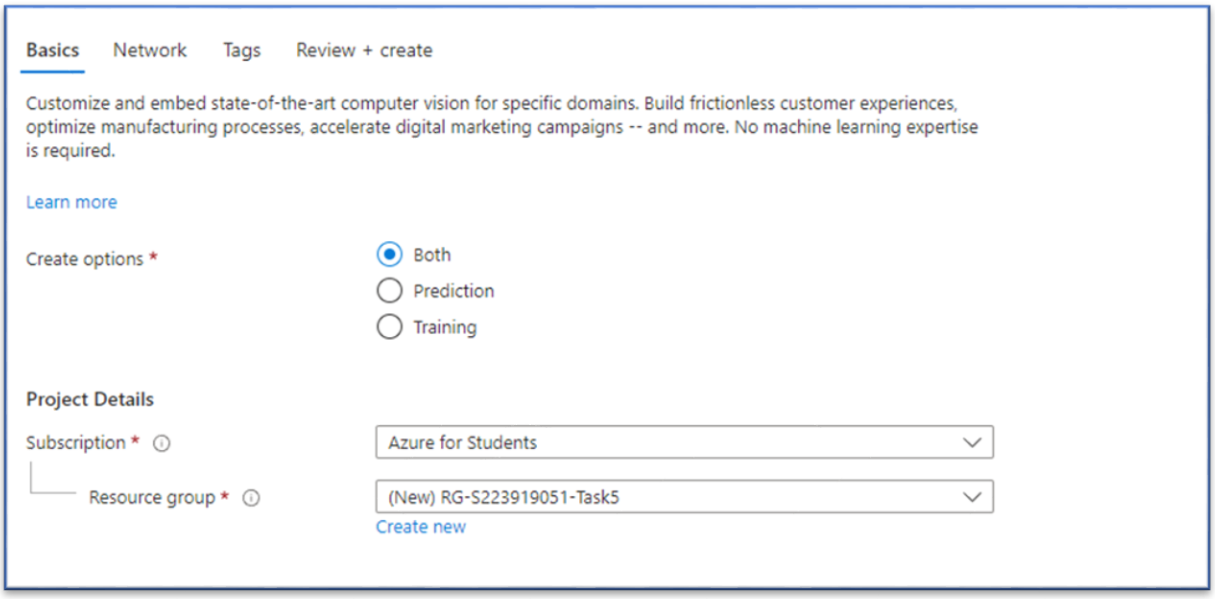
- **Create a New Workspace**:
  - Click on **Create a resource** at the top-left corner.



  - Search for **Custom Vision** & select **Create** from **Custom Vision** to initiate the creation process.



- **Configure the Workspace**:
  - Crate option: Select **Both (Prediction + Training)**
  - Creating a resource group **RG-S223919051-Task5**
  - Instance Detail name **ID-S223919051-Task**
  - Choose an Azure region **Central India** which is close to my location.
  - Pricing tier has been selected with Free Tier
    - Training "**Free F0 (2 Transaction per second, 2 projects)"**
    - Prediction **"Free F0 (2 Transaction per second"**
  - Then click review and create



Customize and embed state-of-the-art computer vision for specific domains. Build frictionless customer experiences, optimize manufacturing processes, accelerate digital marketing campaigns -- and more. No machine learning expertise is required.

Learn more

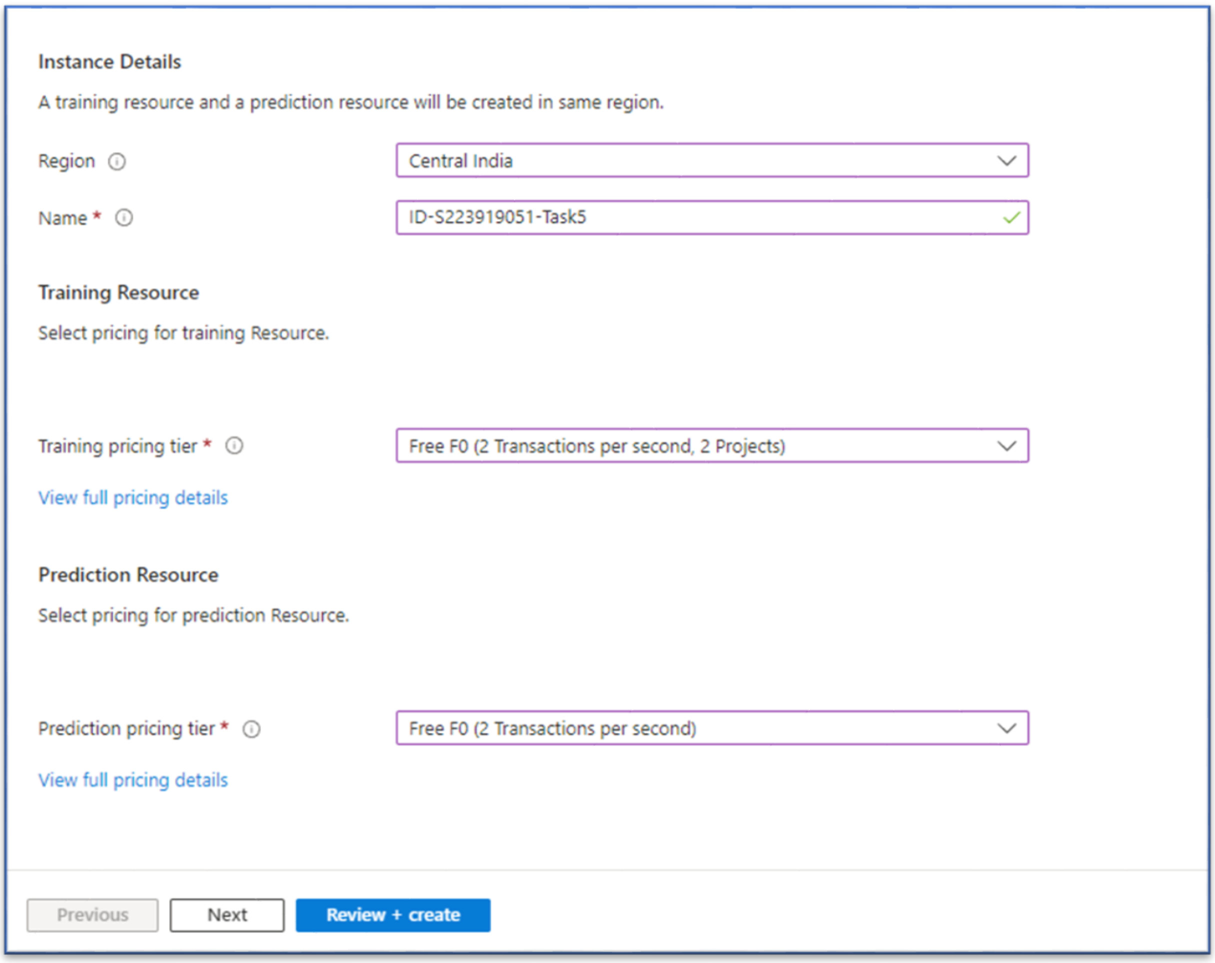
Create options * [✓ Both] Prediction Training

Project Details

Subscription * Azure for Students

Resource group * (New) RG-S223919051-Task5

Create new


Instance Details

A training resource and a prediction resource will be created in same region.

Region: Central India

Name *: ID-S223919051-Task5

Training Resource

Select pricing for training Resource.

Training pricing tier *: Free F0 (2 Transactions per second, 2 Projects)

View full pricing details

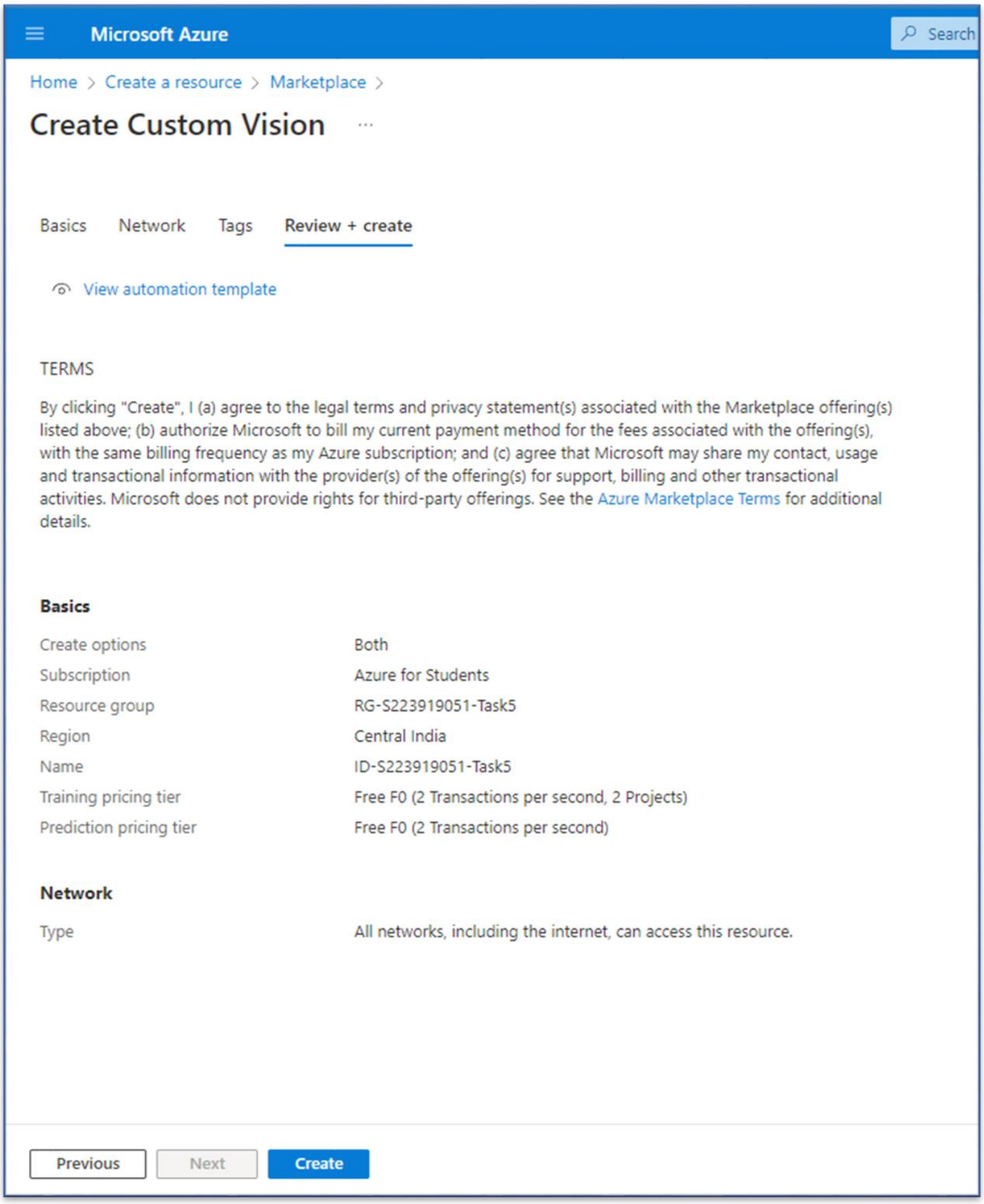
Prediction Resource

Select pricing for prediction Resource.

Prediction pricing tier *: Free F0 (2 Transactions per second)

View full pricing details

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



Create Custom Vision

TERMS

By clicking "Create", I (a) agree to the legal terms and privacy statement(s) associated with the Marketplace offering(s) listed above; (b) authorize Microsoft to bill my current payment method for the fees associated with the offering(s), with the same billing frequency as my Azure subscription; and (c) agree that Microsoft may share my contact, usage and transactional information with the provider(s) of the offering(s) for support, billing and other transactional activities. Microsoft does not provide rights for third-party offerings. See the Azure Marketplace Terms for additional details.

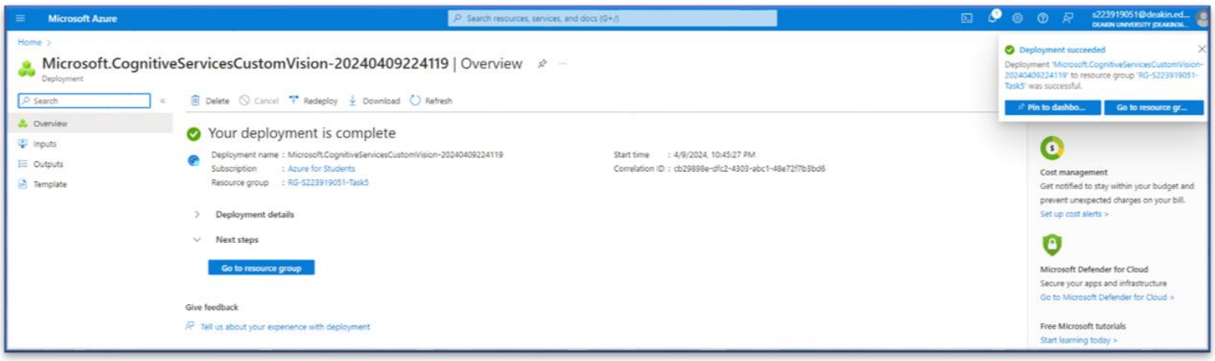
Basics

Create options	Both
Subscription	Azure for Students
Resource group	RG-S223919051-Task5
Region	Central India
Name	ID-S223919051-Task5
Training pricing tier	Free F0 (2 Transactions per second, 2 Projects)
Prediction pricing tier	Free F0 (2 Transactions per second)

Network

Type	All networks, including the internet, can access this resource.

Workspace deployed:



Microsoft.CognitiveServicesCustomVision-20240409224119 | Overview
Deployment

Your deployment is complete

Deployment name : Microsoft.CognitiveServicesCustomVision-20240409224119
Subscription : Azure for Students
Resource group : RG-S223919051-Task5
Start time : 4/9/2024, 10:45:27 PM
Correlation ID : cb29898e-dfc2-4303-abc1-48e72f7b3bd6

Deployment details
Next steps

Give feedback
Tell us about your experience with deployment

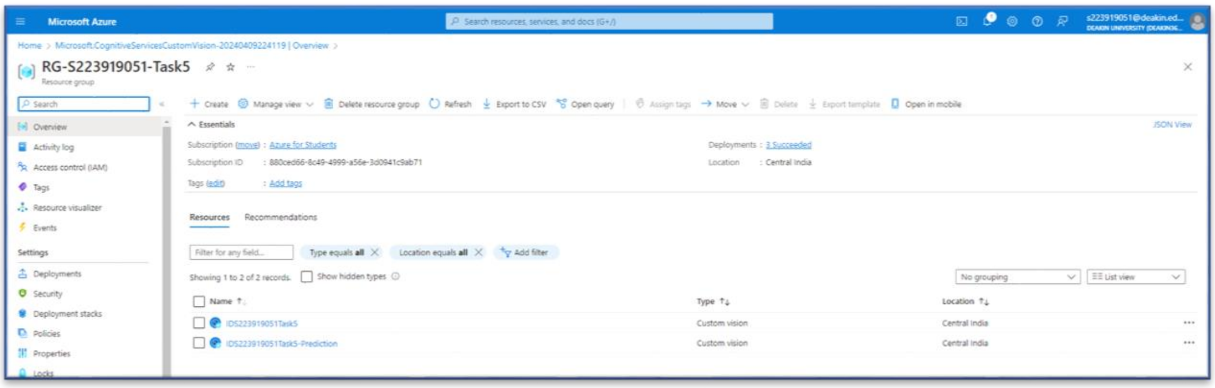
Deployment succeeded
Deployment 'Microsoft.CognitiveServicesCustomVision-20240409224119' to resource group 'RG-S223919051-Task5' was successful.

Cost management
Get notified to stay within your budget and prevent unexpected charges on your bill.
Set up cost alerts >

Microsoft Defender for Cloud
Secure your apps and infrastructure
Go to Microsoft Defender for Cloud >

Free Microsoft tutorials
Start learning today >

Training and Prediction spaces are deployed



RG-S223919051-Task5
Resource group

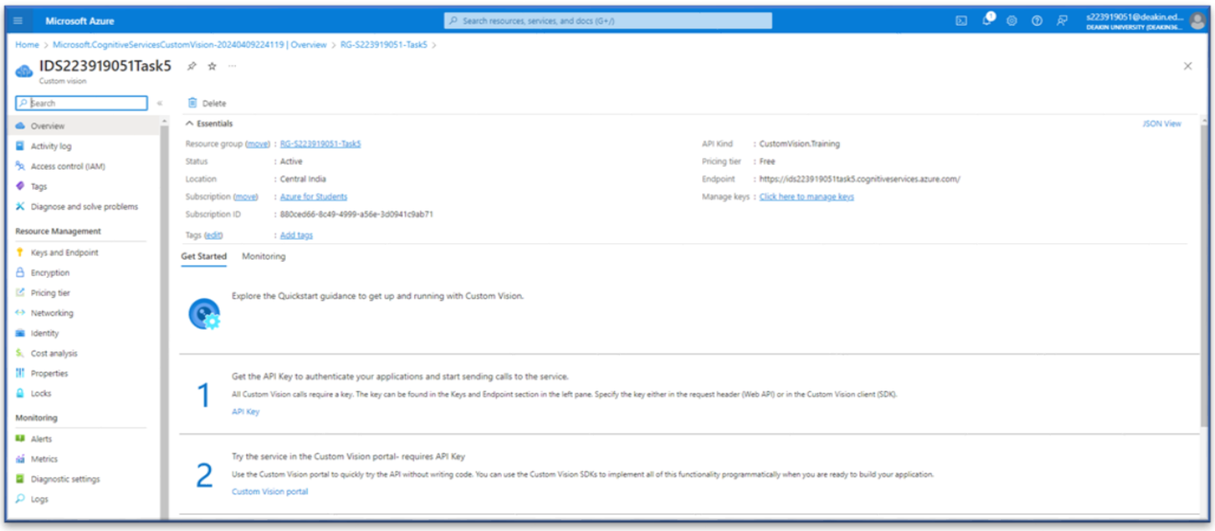
Essentials
Subscription (move) : Azure for Students
Subscription ID : 880ced66-8c49-4999-a56e-3d0941c9ab71
Tags (edit) : Add tags
Deployments : 3 Succeeded
Location : Central India

Resources Recommendations

Showing 1 to 2 of 2 records. Show hidden types

Name	Type	Location
IDS223919051Task5	Custom vision	Central India
IDS223919051Task5-Prediction	Custom vision	Central India

Training Custom Vision:



IDS223919051Task5
Custom vision

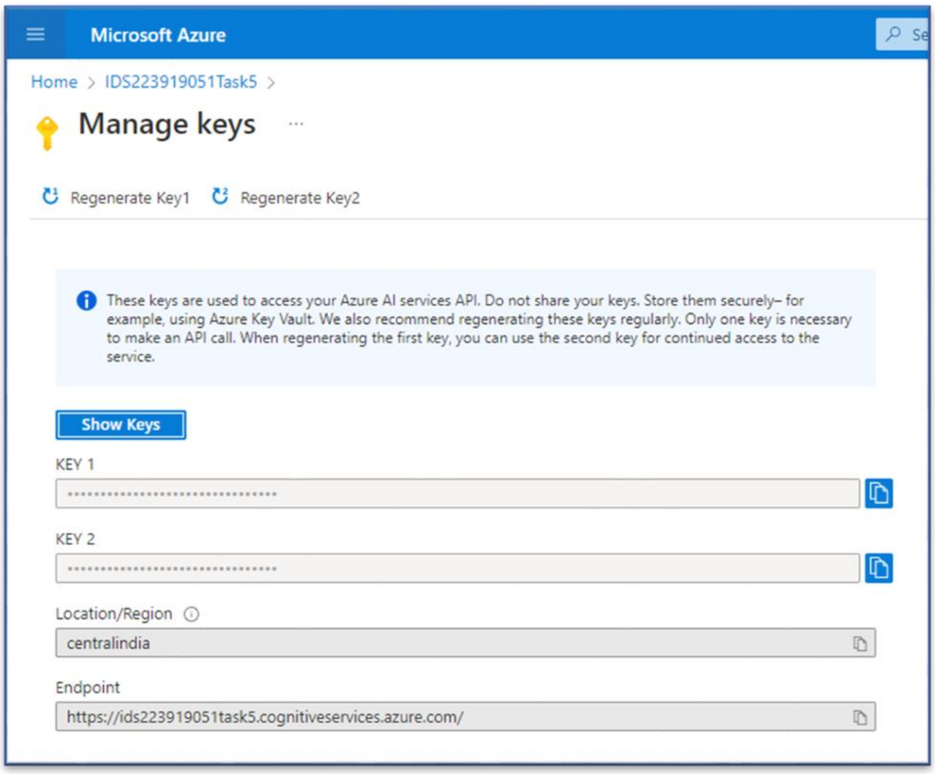
Essentials
Resource group (move) : RG-S223919051-Task5
Status : Active
Location : Central India
Subscription (move) : Azure for Students
Subscription ID : 880ced66-8c49-4999-a56e-3d0941c9ab71
Tags (edit) : Add tags
API Kind : CustomVision.Training
Pricing tier : Free
Endpoint : https://ids223919051task5.cognitiveservices.azure.com/
Manage keys : Click here to manage keys

Explore the Quickstart guidance to get up and running with Custom Vision.

1 Get the API Key to authenticate your applications and start sending calls to the service.
All Custom Vision calls require a key. The key can be found in the Keys and Endpoint section in the left pane. Specify the key either in the request header (Web API) or in the Custom Vision client (SDK).
API Key

2 Try the service in the Custom Vision portal- requires API Key
Use the Custom Vision portal to quickly try the API without writing code. You can use the Custom Vision SDKs to implement all of this functionality programmatically when you are ready to build your application.
Custom Vision portal

1. **Get the API Key to authenticate**



Manage keys

These keys are used to access your Azure AI services API. Do not share your keys. Store them securely– for example, using Azure Key Vault. We also recommend regenerating these keys regularly. Only one key is necessary to make an API call. When regenerating the first key, you can use the second key for continued access to the service.

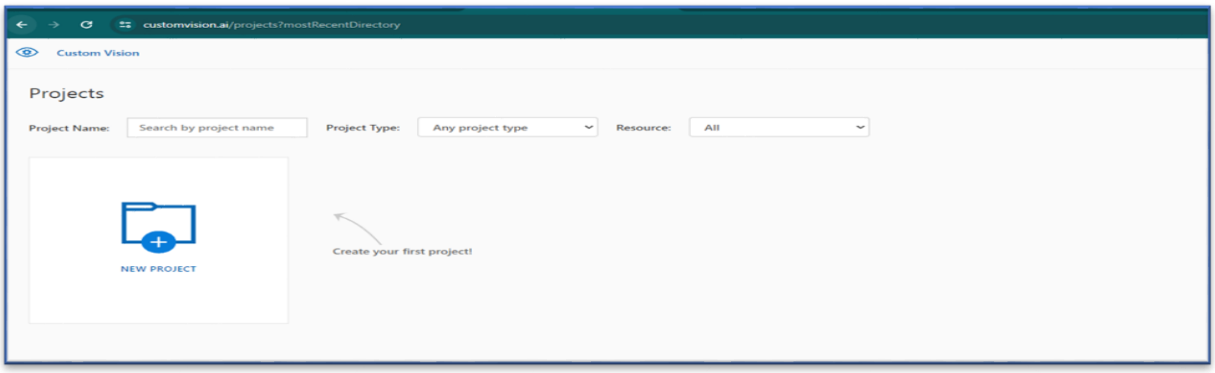
KEY 1: ••••••••••••••••••••••••••••••••

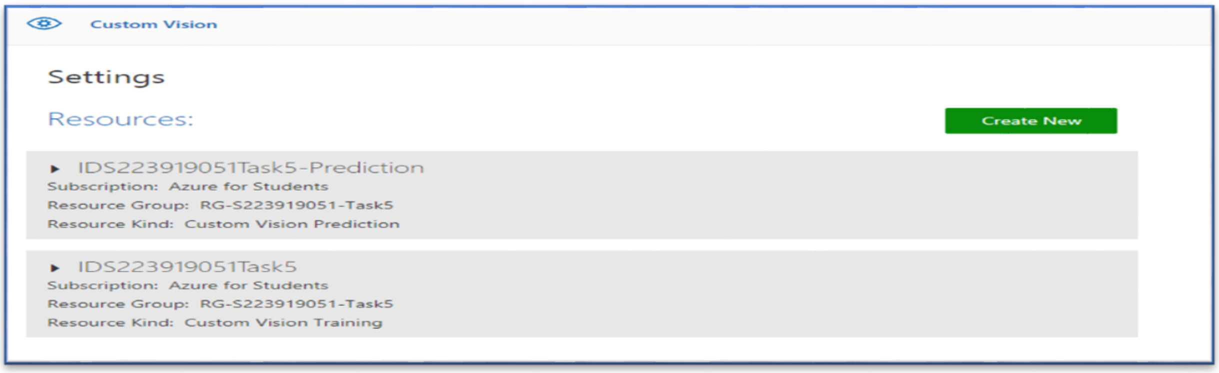
KEY 2: ••••••••••••••••••••••••••••••••

Location/Region: centralindia

Endpoint: https://ids223919051task5.cognitiveservices.azure.com/

2. **Custom Vision Portal**





IDS223919051Task5-Prediction
Subscription: Azure for Students
Resource Group: RG-S223919051-Task5
Resource Kind: Custom Vision Prediction

IDS223919051Task5
Subscription: Azure for Students
Resource Group: RG-S223919051-Task5
Resource Kind: Custom Vision Training

Now let's proceed to create the Python SDK;

Pre-requisites:

Install pre-requisites for Computer Vision in Anaconda prompt.

- pip install azure-cognitiveservices-vision-customvision
- pip install pillow
- pip install python-dotenv

- **azure-cognitiveservices-vision-computervision**: This command installs the Azure Custom Vision library, which allows to interact with the Azure Custom Vision service from Python applications.

- **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.

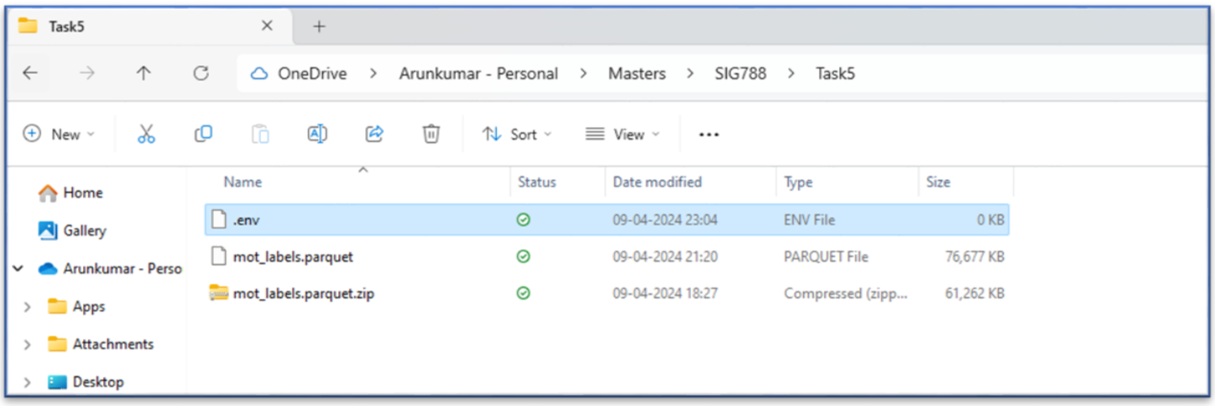
- **python-dotenv**: This package reads key-value pairs from a .env file and sets them as environment variables, enhancing the security and flexibility of application's configuration management.

Configure .env file:

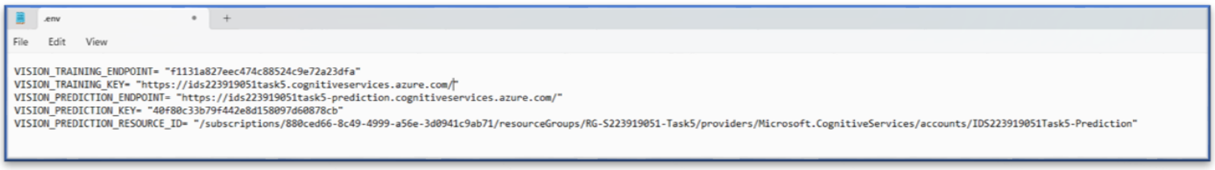
For Azure Computer Vision services, follow these steps to enhance security:

- 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**, **endpoint and resource ID** are not displayed to users, enhancing the security of application. Please refer to the snapshot of the **.env** file.



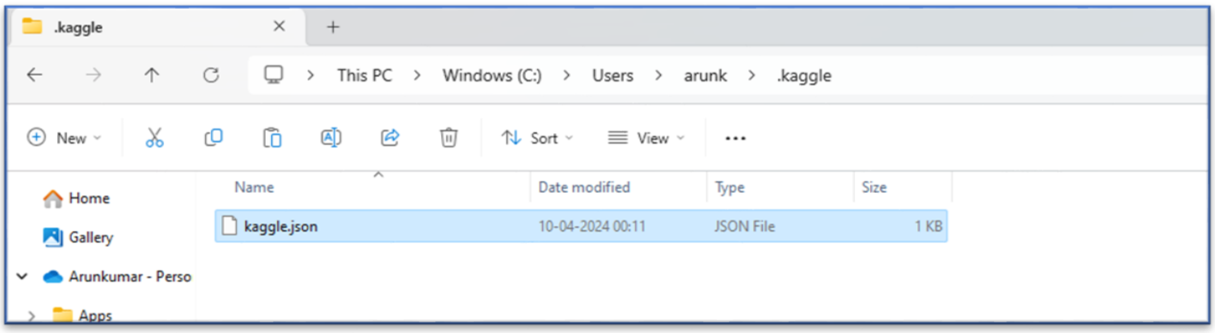
Name	Status	Date modified	Type	Size
.env	✓	09-04-2024 23:04	ENV File	0 KB
mot_labels.parquet	✓	09-04-2024 21:20	PARQUET File	76,677 KB
mot_labels.parquet.zip	✓	09-04-2024 18:27	Compressed (zipp...	61,262 KB



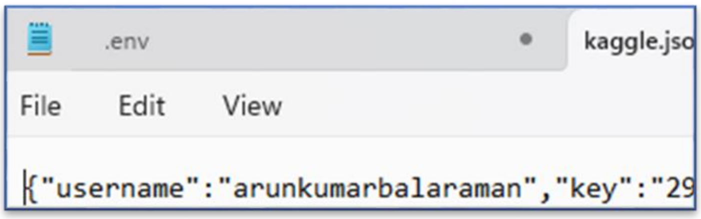
```
VISION_TRAINING_ENDPOINT= "f1131a827eec474c88524c9e72a23dfa"
VISION_TRAINING_KEY= "https://ids223919051task5.cognitiveservices.azure.com/"
VISION_PREDICTION_ENDPOINT= "https://ids223919051task5-prediction.cognitiveservices.azure.com/"
VISION_PREDICTION_KEY= "40f80c33b79f442e8d158097d60878cb"
VISION_PREDICTION_RESOURCE_ID= "/subscriptions/880ced66-8c49-4999-a56e-3d0941c9ab71/resourceGroups/RG-S223919051-Task5/providers/Microsoft.CognitiveServices/accounts/IDS223919051Task5-Prediction"
```

Setting up Kaggle folder:

The Kaggle.json file is used to interact with Datasets to download data, in this case we would be using to download the videos.



Name	Date modified	Type	Size
kaggle.json	10-04-2024 00:11	JSON File	1 KB



```
{"username":"arunkumarbalaraman","key":"29
```

Selecting 3 Videos for training Images:

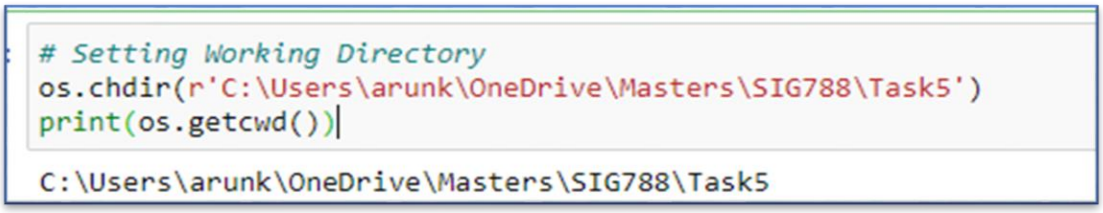
Import the necessary libraries in Python SDK:

```
# to connect to the training resource
from azure.cognitiveservices.vision.customvision.training import
CustomVisionTrainingClient

# to connect to the prediction resource
from azure.cognitiveservices.vision.customvision.prediction import
CustomVisionPredictionClient

# to send the input files in batch; and to identify the object regions; for the training of the
model.
from azure.cognitiveservices.vision.customvision.training.models import
ImageFileCreateBatch, ImageFileCreateEntry, Region

# To authenticate the client
from msrest.authentication import ApiKeyCredentials
import os, uuid

# To read the local environment variables and secret keys
import os, dotenv
from dotenv import load_dotenv, find_dotenv

# To read the dataset
import pandas as pd
```

```
# to view images
from IPython.display import Image as img

# to process data in batches
import itertools

# to draw bounding boxes in local output.
from PIL import Image, ImageDraw
import matplotlib.pyplot as plt
import numpy as np

# Unzip file
import zipfile

# Set os
import os
```

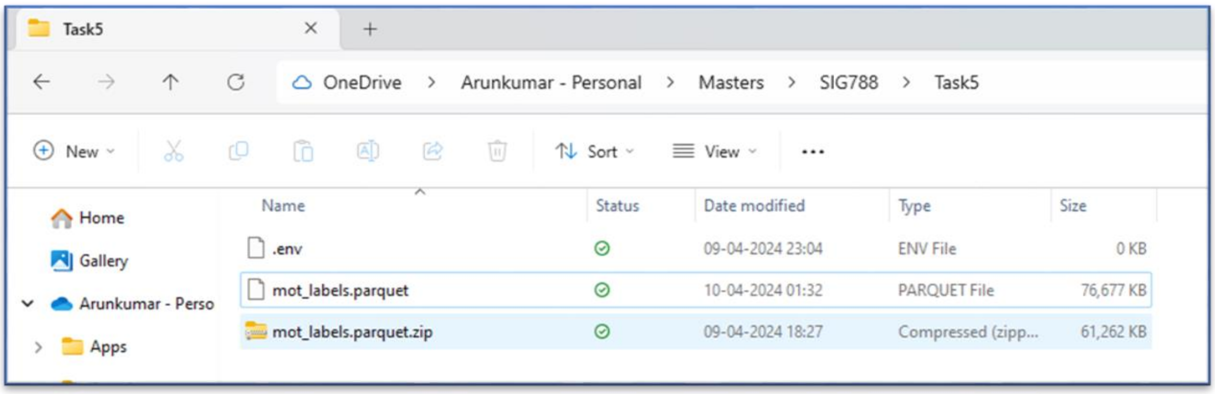
Setting up Working directory:

```
# Setting Working Directory
os.chdir(r'C:\Users\arunk\OneDrive\Masters\SIG788\Task5')
print(os.getcwd())
```



```
# Setting Working Directory
os.chdir(r'C:\Users\arunk\OneDrive\Masters\SIG788\Task5')
print(os.getcwd())

C:\Users\arunk\OneDrive\Masters\SIG788\Task5
```

Downloaded Parquet file from Kaggle:



Name	Status	Date modified	Type	Size
.env	✓	09-04-2024 23:04	ENV File	0 KB
mot_labels.parquet	✓	10-04-2024 01:32	PARQUET File	76,677 KB
mot_labels.parquet.zip	✓	09-04-2024 18:27	Compressed (zipp...	61,262 KB

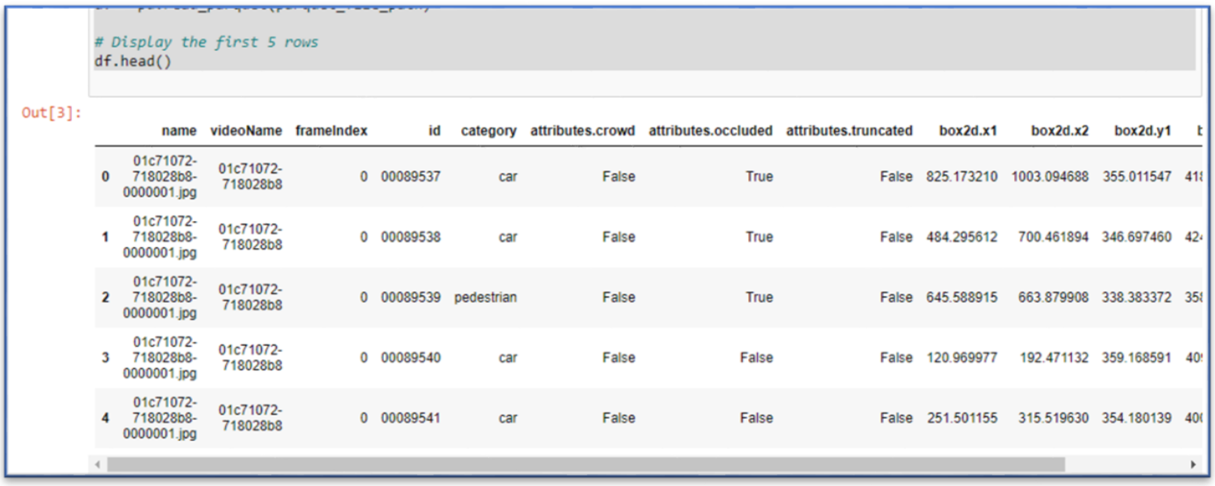
The Parquet file has been downloaded and its been zipped in below code. This file will be used to identify the videos to extract frame and apply bounding boxes information available in this dataset.

Code:

```
# Assign Zip file
zip_file_path = './mot_labels.parquet.zip'
unzipped_folder_path = os.getcwd()
```

```
# Unzipping the .parquet.zip file
with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
    zip_ref.extractall(unzipped_folder_path)

# .parquet file
parquet_file_path = 'mot_labels.parquet'

# Loading the .parquet file
df = pd.read_parquet(parquet_file_path)

# Display the first 5 rows
df.head()
```

Output:



```
# Display the first 5 rows
df.head()
```

Out[3]:

	name	videoName	frameIndex	id	category	attributes.crowd	attributes.occluded	attributes.truncated	box2d.x1	box2d.x2	box2d.y1	b
0	01c71072-718028b8-0000001.jpg	01c71072-718028b8	0	00089537	car	False	True	False	825.173210	1003.094688	355.011547	41
1	01c71072-718028b8-0000001.jpg	01c71072-718028b8	0	00089538	car	False	True	False	484.295612	700.461894	346.697460	42
2	01c71072-718028b8-0000001.jpg	01c71072-718028b8	0	00089539	pedestrian	False	True	False	645.588915	663.879908	338.383372	35
3	01c71072-718028b8-0000001.jpg	01c71072-718028b8	0	00089540	car	False	False	False	120.969977	192.471132	359.168591	40
4	01c71072-718028b8-0000001.jpg	01c71072-718028b8	0	00089541	car	False	False	False	251.501155	315.519630	354.180139	40

Inference on dataset:

The dataset from various videos has been included in Kaggle which would specifically help us in object detection. The dataset includes annotated frames with details about the presence and position of objects like cars, pedestrians, bus, cycle, etc. Each object is assigned a unique ID and category.

Key columns in the dataset:

➢ **Bounding Box Coordinates**: They are

- box2d.x1, box2d.x2 for horizontal positioning and
- box2d.y1, box2d.y2 for vertical positioning helping in precise object localization.

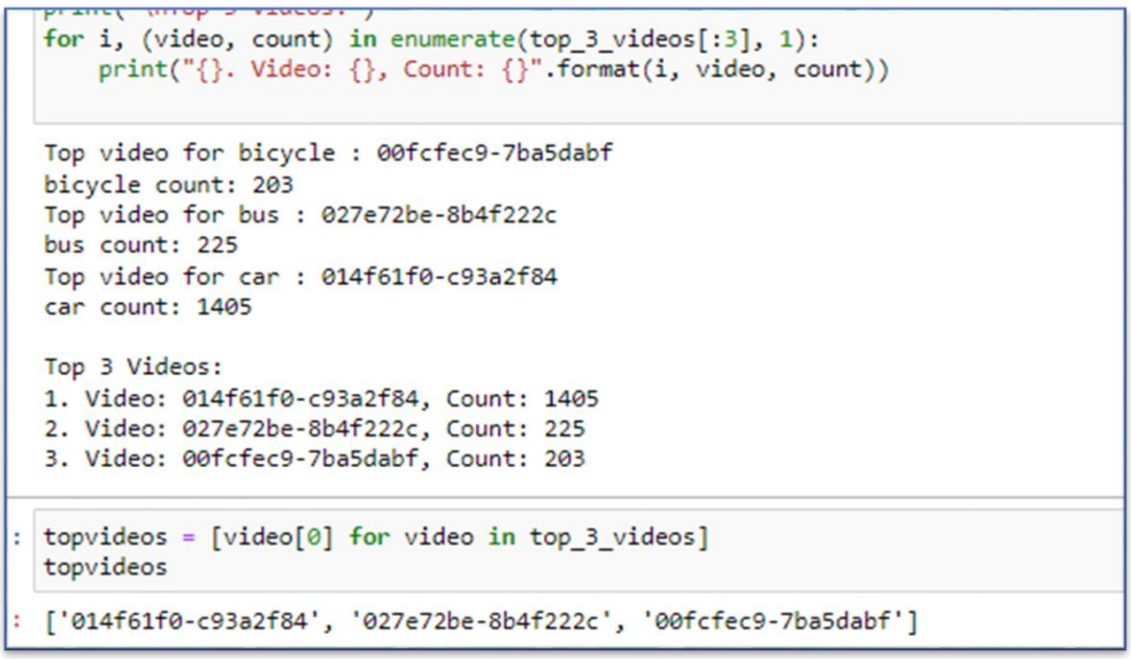
Top 3 Videos

In the dataset, identifying the 3 videos with a more number of categories by performing a group by count on VideoName and category. This helps in selecting the top 3 video when car, bus and bicycle has more occurrence for better training.

Code:

```
# Define a list to store top videos
top_3_videos = []

# Iterate over each category
for category in ['bicycle', 'bus', 'car']:
   # Filter the DataFrame
   filtered_df = df[(df['attributes.crowd'] == False) & (df['attributes.occluded'] == False) &
(df['category'] == category)]

   # Group by 'videoName' and count occurrences of the category
   category_counts = filtered_df.groupby('videoName').size()

   # Find the top video and its count
   top_video = category_counts.idxmax()
   top_count = category_counts.max()

   # Print results
   print("Top video for", category, ":", top_video)
   print(category, "count:", top_count)

   # Add top video to the list
   top_3_videos.append((top_video, top_count))

# Sort top videos based on count in descending order
top_3_videos.sort(key=lambda x: x[1], reverse=True)

# Display the top 3 videos
print("\nTop 3 Videos:")
for i, (video, count) in enumerate(top_3_videos[:3], 1):
   print("{}. Video: {}, Count: {}".format(i, video, count))
```

Output:

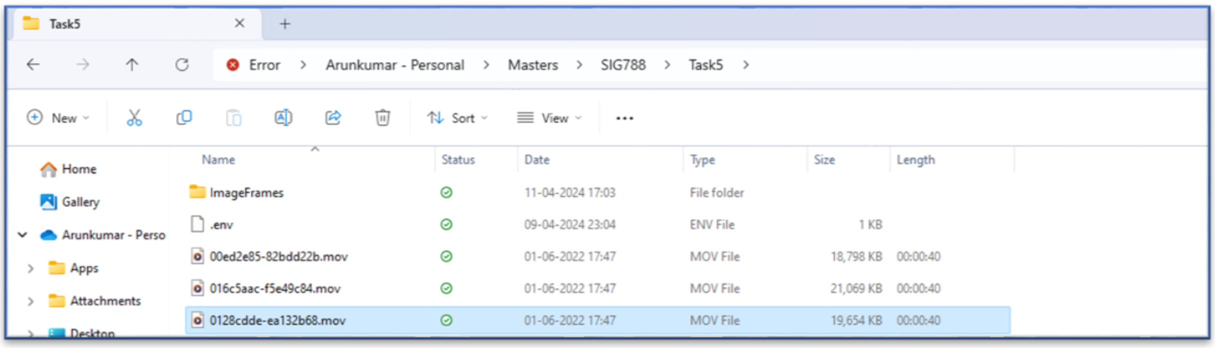


```
for i, (video, count) in enumerate(top_3_videos[:3], 1):
    print("{}. Video: {}, Count: {}".format(i, video, count))

Top video for bicycle : 00fcfec9-7ba5dabf
bicycle count: 203
Top video for bus : 027e72be-8b4f222c
bus count: 225
Top video for car : 014f61f0-c93a2f84
car count: 1405

Top 3 Videos:
1. Video: 014f61f0-c93a2f84, Count: 1405
2. Video: 027e72be-8b4f222c, Count: 225
3. Video: 00fcfec9-7ba5dabf, Count: 203
```

```
topvideos = [video[0] for video in top_3_videos]
topvideos
```

```
['014f61f0-c93a2f84', '027e72be-8b4f222c', '00fcfec9-7ba5dabf']
```

Manually downloaded these videos to extract frame images as I don’t want to download all 1000 images where the file size is 20 GB.

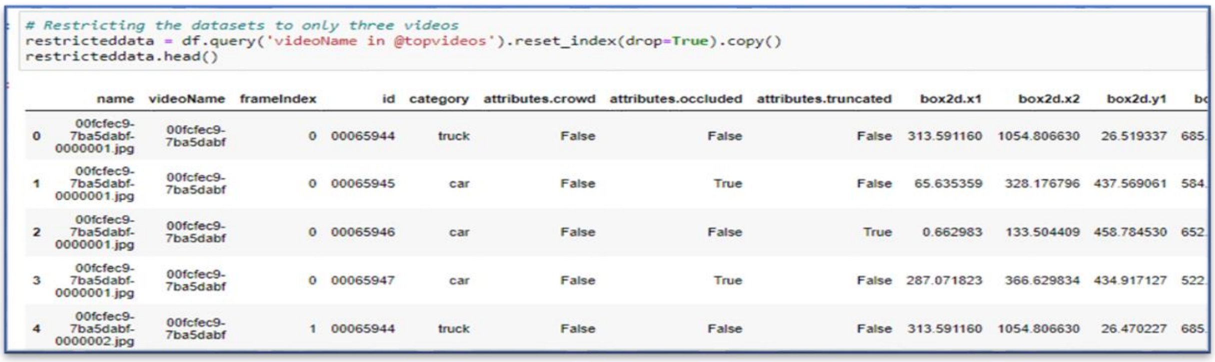


Name	Status	Date	Type	Size	Length
ImageFrames	✓	11-04-2024 17:03	File folder		
.env	✓	09-04-2024 23:04	ENV File	1 KB	
00ed2e85-82bdd22b.mov	✓	01-06-2022 17:47	MOV File	18,798 KB	00:00:40
016c5aac-f5e49c84.mov	✓	01-06-2022 17:47	MOV File	21,069 KB	00:00:40
0128cdde-ea132b68.mov	✓	01-06-2022 17:47	MOV File	19,654 KB	00:00:40

The training frame images would be picked up only from these three videos hence restricting the dataset to information only for these three videos.

```
# Restricting the datasets to only three videos
restricteddata = df.query('videoName in @topvideos').reset_index(drop=True).copy()
restricteddata.head()
```

Output:



```
# Restricting the datasets to only three videos
restricteddata = df.query('videoName in @topvideos').reset_index(drop=True).copy()
restricteddata.head()
```

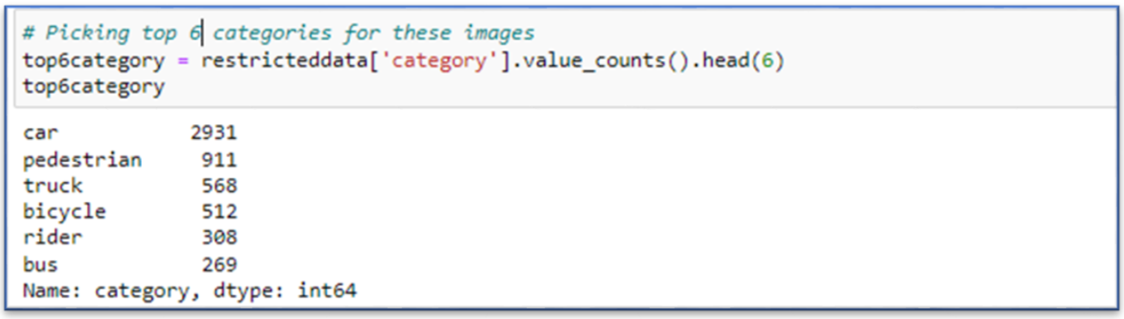
	name	videoName	frameIndex	id	category	attributes.crowd	attributes.occluded	attributes.truncated	box2d.x1	box2d.x2	box2d.y1	bo
0	00fcfec9-7ba5dabf-0000001.jpg	00fcfec9-7ba5dabf	0	00065944	truck	False	False	False	313.591160	1054.806630	26.519337	685.
1	00fcfec9-7ba5dabf-0000001.jpg	00fcfec9-7ba5dabf	0	00065945	car	False	True	False	65.635359	328.176796	437.569061	584.
2	00fcfec9-7ba5dabf-0000001.jpg	00fcfec9-7ba5dabf	0	00065946	car	False	False	True	0.662983	133.504409	458.784530	652.
3	00fcfec9-7ba5dabf-0000001.jpg	00fcfec9-7ba5dabf	0	00065947	car	False	True	False	287.071823	366.629834	434.917127	522.
4	00fcfec9-7ba5dabf-0000002.jpg	00fcfec9-7ba5dabf	1	00065944	truck	False	False	False	313.591160	1054.806630	26.470227	685.

Category – Tagging

Now let’s restrict the data further into top 6 category for performing training from these videos.

```
# Picking top 6 categories for these images
Top6category = restricteddata['category'].value_counts().head()
Top6category
```

Output



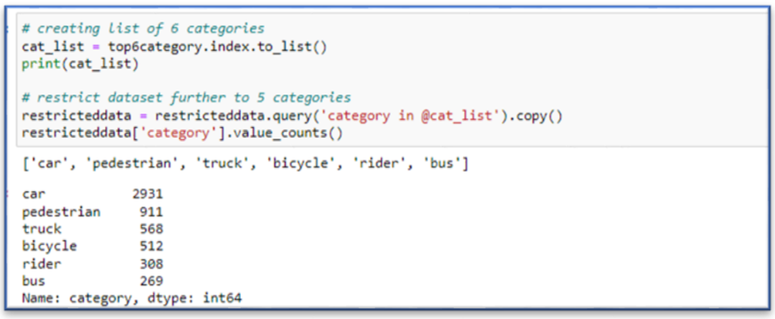
```
# Picking top 6 categories for these images
top6category = restricteddata['category'].value_counts().head(6)
top6category

car           2931
pedestrian     911
truck          568
bicycle        512
rider          308
bus            269
Name: category, dtype: int64
```

The top 6 categories are Car, pedestrian, bicycle, truck, bus & rider are most frequent in these videos and tagging the images with these categories.

Deleting other categories from the dataset.

```
# creating list of 6 categories
cat_list = top6category.index.to_list()
print(cat_list)

# restrict dataset further to 5 categories
restricteddata = restricteddata.query('category in @cat_list').copy()
restricteddata['category'].value_counts()
```

Output:



```
# creating list of 6 categories
cat_list = top6category.index.to_list()
print(cat_list)

# restrict dataset further to 5 categories
restricteddata = restricteddata.query('category in @cat_list').copy()
restricteddata['category'].value_counts()

['car', 'pedestrian', 'truck', 'bicycle', 'rider', 'bus']

car           2931
pedestrian     911
truck          568
bicycle        512
rider          308
bus            269
Name: category, dtype: int64
```

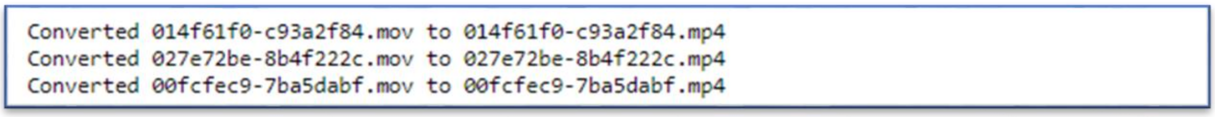
Video conversion mov to mp4:

Converting .mov to .mp4 for Azure Custom Vision ensures consistency and avoids potential errors during model training or inference. The .mp4 format is widely supported and playable on various devices and platforms, enhancing the versatility and accessibility of the model.

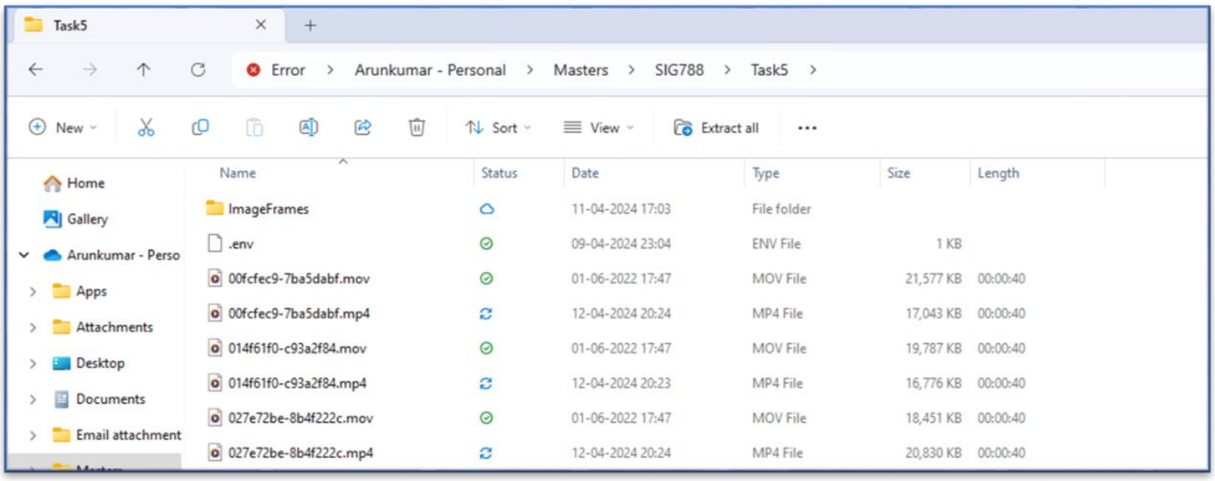
Code:

```
# Loop the top 3 videos
for video_file in topvideos:
  (
     ffmpeg
     .input(video_file + '.mov')
     .output(video_file + '.mp4')
     .global_args('-y')  # overwrite .mp4 if available
     .run()
  )
  # Print success
  print(f"Converted {video_file + '.mov'} to {video_file + '.mp4'}")
```

Output:

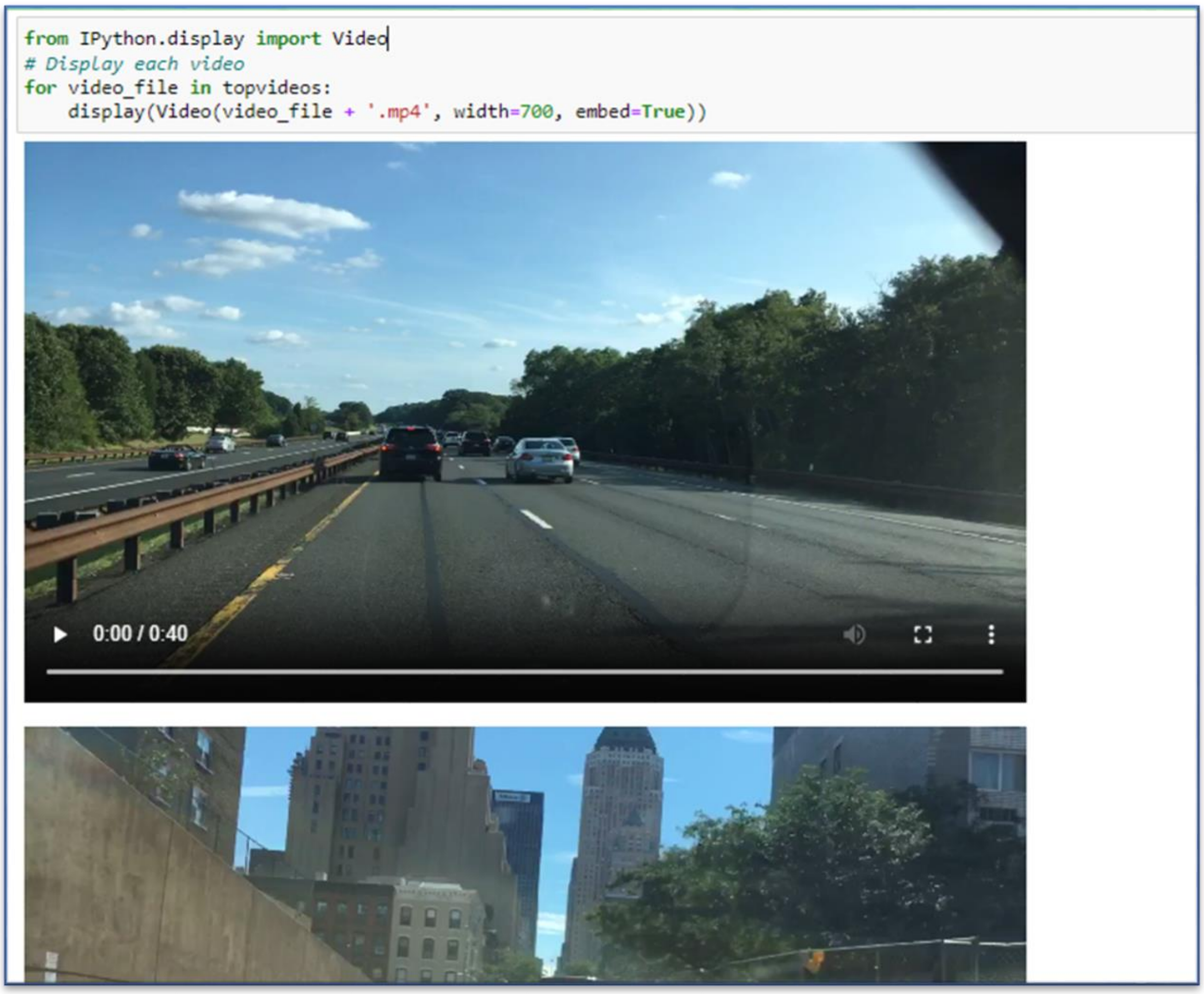


```
Converted 014f61f0-c93a2f84.mov to 014f61f0-c93a2f84.mp4
Converted 027e72be-8b4f222c.mov to 027e72be-8b4f222c.mp4
Converted 00fcfec9-7ba5dabf.mov to 00fcfec9-7ba5dabf.mp4
```



Name	Status	Date	Type	Size	Length
ImageFrames		11-04-2024 17:03	File folder		
.env		09-04-2024 23:04	ENV File	1 KB	
00fcfec9-7ba5dabf.mov		01-06-2022 17:47	MOV File	21,577 KB	00:00:40
00fcfec9-7ba5dabf.mp4		12-04-2024 20:24	MP4 File	17,043 KB	00:00:40
014f61f0-c93a2f84.mov		01-06-2022 17:47	MOV File	19,787 KB	00:00:40
014f61f0-c93a2f84.mp4		12-04-2024 20:23	MP4 File	16,776 KB	00:00:40
027e72be-8b4f222c.mov		01-06-2022 17:47	MOV File	18,451 KB	00:00:40
027e72be-8b4f222c.mp4		12-04-2024 20:24	MP4 File	20,830 KB	00:00:40

Successfully converted the .mov into .mp4 files and now let’s proceed to display the .mp4 format using **Ipython.display**



```
from IPython.display import Video
# Display each video
for video_file in topvideos:
    display(Video(video_file + '.mp4', width=700, embed=True))
```

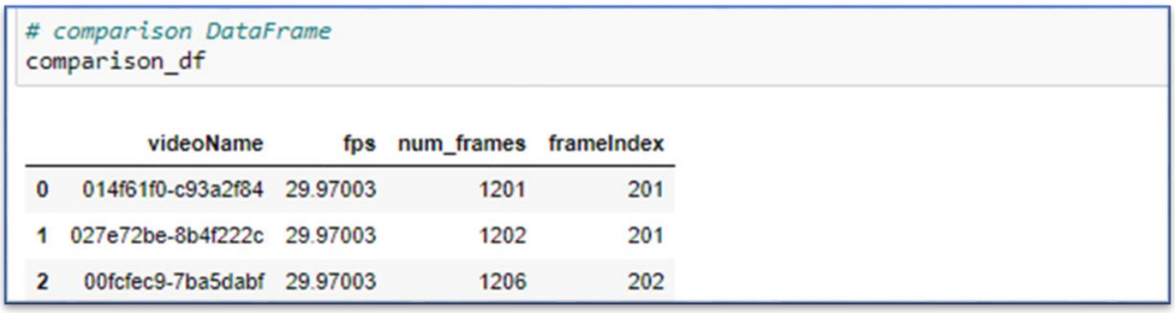
Frame image calculation and selection:

Let’s calculate the frame in video against frame in the datasets.

```
# Initialize a list
```

```
video_frames_info = []

# Loop through each video to get frame counts
for video in topvideos:
    cap = cv2.VideoCapture(video + '.mp4')
    if not cap.isOpened():
        print(f"Failed to open video: {video + '.mp4'}")
        num_frames = 0
        fps = 0
    else:
        fps = cap.get(cv2.CAP_PROP_FPS)
        num_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
    cap.release()

    video_frames_info.append({
        'videoName': video,
        'fps': fps,
        'num_frames': num_frames
    })

# Convert to DataFrame
video_frames_df = pd.DataFrame(video_frames_info)

# Group by 'videoName' to get the max frame index for each video
max_frame_indices = restricteddata[['frameIndex',
'videoName']].groupby('videoName').max().reset_index()

# Merge the two DataFrames on 'videoName' to compare
comparison_df = pd.merge(video_frames_df, max_frame_indices, how='left',
on='videoName', suffixes=('_actual', '_dataset'))

# comparison DataFrame
comparison_df
```

Output:



```
# comparison DataFrame
comparison_df
```

	videoName	fps	num_frames	frameIndex
0	014f61f0-c93a2f84	29.97003	1201	201
1	027e72be-8b4f222c	29.97003	1202	201
2	00fcfec9-7ba5dabf	29.97003	1206	202

Inference:

Compared three videos with varying **frame rates** and **total frames**.

- All three videos has a frame rate of 30 FPS. The total number of frames varies significantly among the videos due to differences in video length and frame rate.

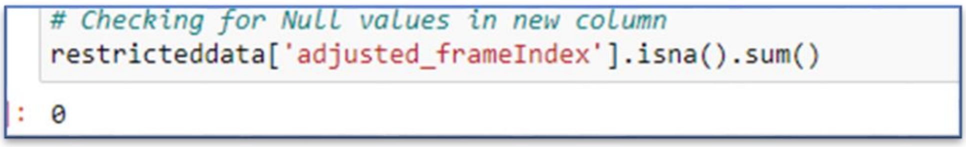
The **maximum frame index** in dataset is **notably lower** than the actual number of frames for each video, indicating that the dataset **might not encompass** the full length of the videos. This suggests that only a subset of each video's frames is used, possibly due to relevance or **dataset incompleteness**.

To **overcome** this **limitation**, an integer value obtained by **dividing** the **actual video frames** by the **dataset's frame index** is used to capture frame images at specific points.

Code:

```
# Loop for each video
for index, row in comparison_df.iterrows():
   video_name = row['videoName']  # Video Name
   num_frames = row['num_frames'] # Num of frames in Video
   max_frame_index = row['frameIndex'] # Max frames in dataset

   # Calculate the adjustment factor for each video
   adjustment_factor = num_frames / max_frame_index

   # Update frameIndex in video_labels where videoName matches
   restricteddata.loc[restricteddata['videoName'] == video_name, 'adjusted_frameIndex'] = (
      restricteddata.loc[restricteddata['videoName'] == video_name, 'frameIndex'] *
adjustment_factor
   ).round().astype(int)

# Checking for Null values in new column
restricteddata['adjusted_frameIndex'].isna().sum()
```

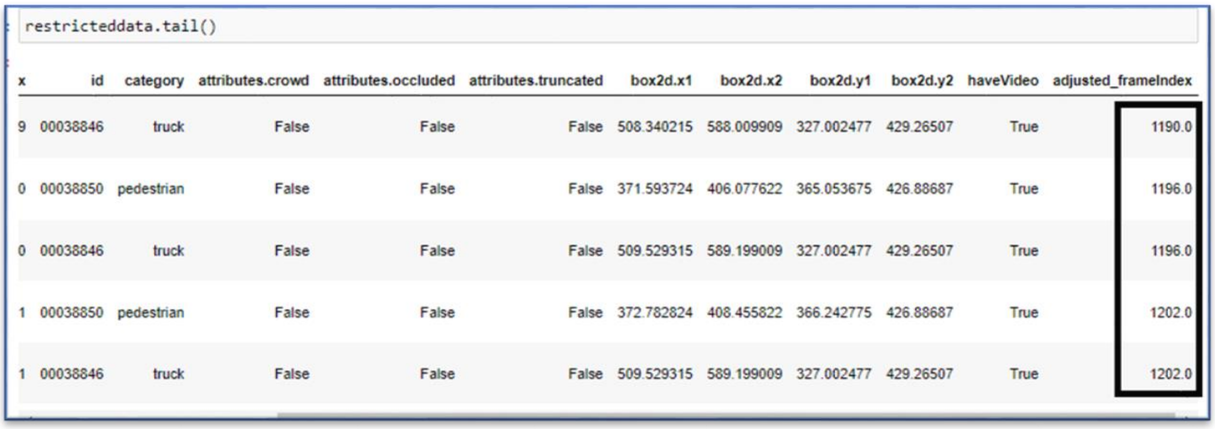
Output:



```
# Checking for Null values in new column
restricteddata['adjusted_frameIndex'].isna().sum()
```

```
: 0
```

Basic check the new column adjusted_frameIndex to match the video



```
restricteddata.tail()
```

x	id	category	attributes.crowd	attributes.occluded	attributes.truncated	box2d.x1	box2d.x2	box2d.y1	box2d.y2	haveVideo	adjusted_frameIndex
9	00038846	truck	False	False	False	508.340215	588.009909	327.002477	429.26507	True	1190.0
0	00038850	pedestrian	False	False	False	371.593724	406.077622	365.053675	426.88687	True	1196.0
0	00038846	truck	False	False	False	509.529315	589.199009	327.002477	429.26507	True	1196.0
1	00038850	pedestrian	False	False	False	372.782824	408.455822	366.242775	426.88687	True	1202.0
1	00038846	truck	False	False	False	509.529315	589.199009	327.002477	429.26507	True	1202.0

As we now successfully created the adjusted_frameIndex, let’s proceed to create the frame images and save it a folder to load the images into Azure custom vision for model training.

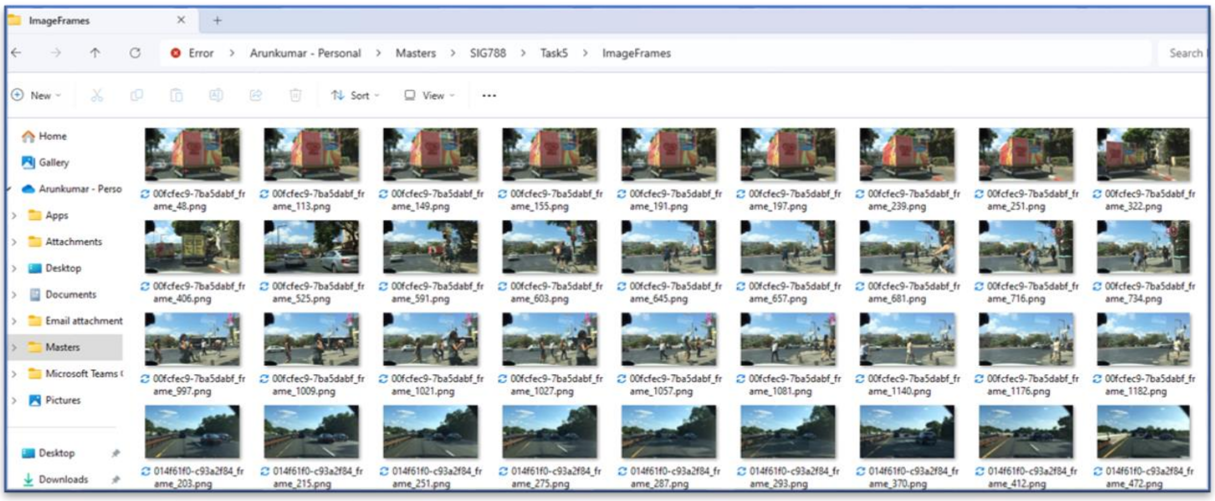
Create Frame Image:

We will utilize the adjusted frame index from the dataset for each video to generate images, which will be stored in a designated folder. Subsequently, employed Python SDK to verify the precision of the bounding boxes. The images, along with the bounding box data, will then be uploaded to the Azure Custom Vision project for further analysis.

Selecting 33 random frames to save image in output folder for each Video Name.

```
# 33 random frames per video
def select_random_frames(frames, count=33):
   if len(frames) > count:
      return np.random.choice(frames, size=count, replace=False)
   return frames

unique_frames_per_video =
restricteddata.groupby('videoName')['adjusted_frameIndex'].unique().apply(lambda x:
select_random_frames(np.sort(x)))

# Function to read the frame index from dataframe and save image
def save_specific_frame(video_path, frame_number, output_filename):
   cap = cv2.VideoCapture(video_path)
   cap.set(cv2.CAP_PROP_POS_FRAMES, frame_number)
   success, frame = cap.read()
   if success:
      cv2.imwrite(output_filename, frame)
   cap.release()

# Image directory
image_directory = 'ImageFrames'
if not os.path.exists(image_directory):
   os.makedirs(image_directory)

# Loop for each item
for video_name, frames in unique_frames_per_video.items():
   video_path = video_name + '.mp4'
   for frame_number in frames:
      output_filename = os.path.join(image_directory,
f"{video_name}_frame_{int(frame_number)}.png")
      save_specific_frame(video_path, frame_number, output_filename)

# List unique_frames_per_video
selected_frames = [(video_name, frame) for video_name, frames in
unique_frames_per_video.items() for frame in frames]

# Frame selection
filtered_dataset = restricteddata[restricteddata.apply(lambda row: (row['videoName'],
row['adjusted_frameIndex']) in selected_frames, axis=1)].reset_index(drop=True)
```

99 images are saved in the below path to load the same into Azure Custom Vision project.



ImageFrames
Error > Arunkumar - Personal > Masters > SIG788 > Task5 > ImageFrames
Search
New
Sort
View
Home
Gallery
Arunkumar - Perso
Apps
Attachments
Desktop
Documents
Email attachment
Masters
Microsoft Teams
Pictures
Desktop
Downloads
00fcfec9-7ba5dabf_frame_48.png
00fcfec9-7ba5dabf_frame_113.png
00fcfec9-7ba5dabf_frame_149.png
00fcfec9-7ba5dabf_frame_155.png
00fcfec9-7ba5dabf_frame_191.png
00fcfec9-7ba5dabf_frame_197.png
00fcfec9-7ba5dabf_frame_239.png
00fcfec9-7ba5dabf_frame_251.png
00fcfec9-7ba5dabf_frame_322.png
00fcfec9-7ba5dabf_frame_406.png
00fcfec9-7ba5dabf_frame_525.png
00fcfec9-7ba5dabf_frame_591.png
00fcfec9-7ba5dabf_frame_603.png
00fcfec9-7ba5dabf_frame_645.png
00fcfec9-7ba5dabf_frame_657.png
00fcfec9-7ba5dabf_frame_681.png
00fcfec9-7ba5dabf_frame_716.png
00fcfec9-7ba5dabf_frame_734.png
00fcfec9-7ba5dabf_frame_997.png
00fcfec9-7ba5dabf_frame_1009.png
00fcfec9-7ba5dabf_frame_1021.png
00fcfec9-7ba5dabf_frame_1027.png
00fcfec9-7ba5dabf_frame_1057.png
00fcfec9-7ba5dabf_frame_1081.png
00fcfec9-7ba5dabf_frame_1140.png
00fcfec9-7ba5dabf_frame_1176.png
00fcfec9-7ba5dabf_frame_1182.png
014f61f0-c93a2f84_frame_203.png
014f61f0-c93a2f84_frame_215.png
014f61f0-c93a2f84_frame_251.png
014f61f0-c93a2f84_frame_275.png
014f61f0-c93a2f84_frame_287.png
014f61f0-c93a2f84_frame_293.png
014f61f0-c93a2f84_frame_370.png
014f61f0-c93a2f84_frame_412.png
014f61f0-c93a2f84_frame_472.png

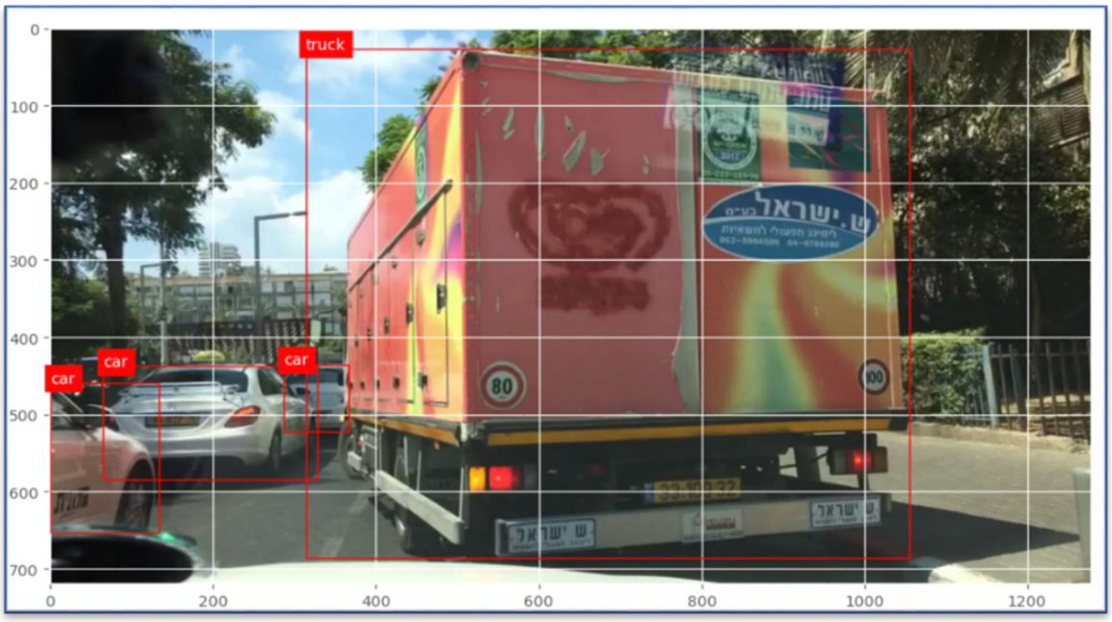
As we now saved the images, let's display 2 images for each video name to ensure the bounding boxes are correctly aligned to train the images in Custom vision project.

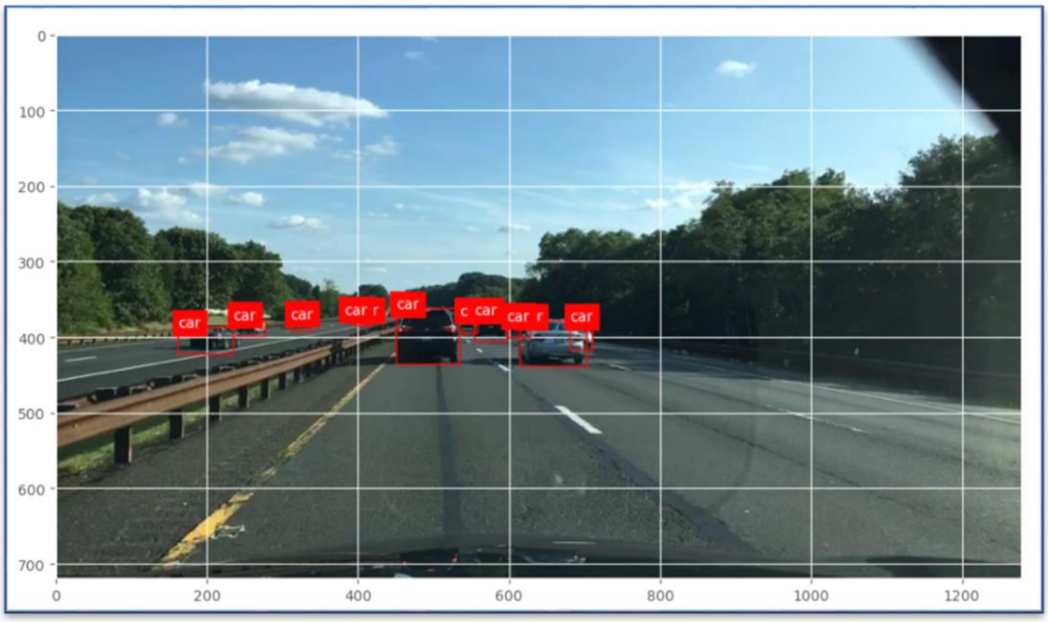
```python
# function to diplay the saved images with bounding boxes
def display_image_with_annotations(image_path, annotations):
    """
    Displays an image with bounding boxes and category labels.

    Parameters:
    - image_path: Path to the image file.
    - annotations: DataFrame containing bounding box and category information.
    """
    # Load the image
    img = Image.open(image_path)

    # Create a figure
    fig, ax = plt.subplots(1, figsize=(12, 8))
    ax.imshow(img)

    # Draw bounding boxes and labels
    for _, row in annotations.iterrows():
        x1, y1, x2, y2 = row['box2d.x1'], row['box2d.y1'], row['box2d.x2'], row['box2d.y2']
        width, height = x2 - x1, y2 - y1
        rect = patches.Rectangle((x1, y1), width, height, linewidth=1, edgecolor='r',
facecolor='none')
        ax.add_patch(rect)
        ax.text(x1, y1, row['category'], color='white', backgroundcolor='red')

    plt.show()
```

```python
# Loop to run to display 2 images per videofile
for video_name in filtered_dataset['videoName'].unique():
    # Filter the dataset for this videoName
    video_data = filtered_dataset[filtered_dataset['videoName'] == video_name]
```

```
# Get two unique frames
unique_frames = video_data['adjusted_frameIndex'].unique()[:2]

for frame in unique_frames:
    # Filter annotations for selected frame
    frame_annotations = video_data[video_data['adjusted_frameIndex'] == frame]

    # Image filename
    image_filename = f"{video_name}_frame_{int(frame)}.png"
    image_path = os.path.join(image_directory, image_filename)

    # Display the image with annotations
    display_image_with_annotations(image_path, frame_annotations)
```

Output:





As we now validated the bounding boxes are correctly assigned, now let’s loaded the images into Azure custom vision project.

Azure Custom Vision Project Image Load:

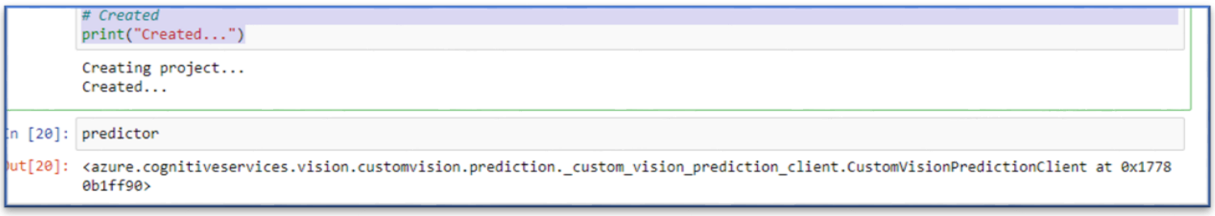
As we have validated the images with bounding boxes, we will start to set up the environment variables for the key, subscription and endpoint for both train and prediction.

```
# Read the environment file to access secret keys
load_dotenv()

# Replace with valid values
VISION_TRAINING_ENDPOINT = os.environ["VISION_TRAINING_ENDPOINT"]
training_key = os.environ["VISION_TRAINING_KEY"]
prediction_key = os.environ["VISION_PREDICTION_KEY"]
prediction_resource_id = os.environ["VISION_PREDICTION_RESOURCE_ID"]
VISION_PREDICTION_ENDPOINT= os.environ["VISION_PREDICTION_ENDPOINT"]
```

Project Creation:

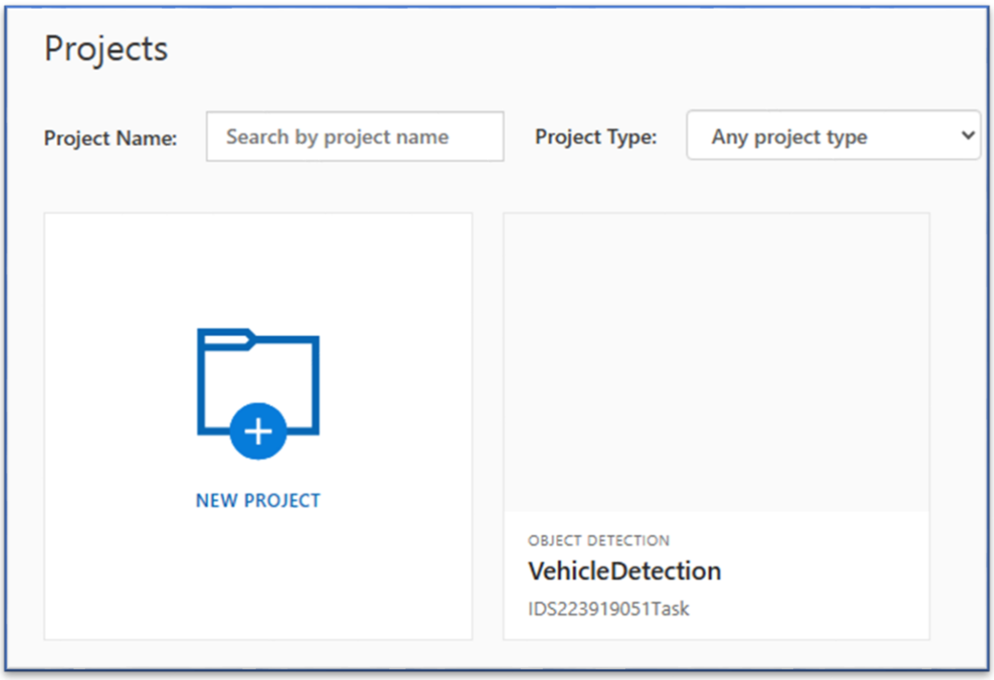
```
# create variables for training resource
credentials = ApiKeyCredentials(in_headers={"Training-key": training_key})
trainer = CustomVisionTrainingClient(VISION_TRAINING_ENDPOINT, credentials)

# create variables for prediction resource
prediction_credentials = ApiKeyCredentials(in_headers={"Prediction-Key": prediction_key})
predictor = CustomVisionPredictionClient(VISION_PREDICTION_ENDPOINT,
prediction_credentials)

publish_iteration_name = "VehicleDetection"

# Find the object detection domain
obj_detection_domain = next(domain for domain in trainer.get_domains() if
                    domain.type == "ObjectDetection" and domain.name == "General")

# Create a new project
print("Creating project...")
# Use uuid to avoid project name collisions.
project =
trainer.create_project(publish_iteration_name,domain_id=obj_detection_domain.id)
# Created
print("Created...")
```

Output:



```
# Created
print("Created...")

Creating project...
Created...

In [20]: predictor

Out[20]: <azure.cognitiveservices.vision.customvision.prediction._custom_vision_prediction_client.CustomVisionPredictionClient at 0x1778
0b1ff90>
```

The project Vehicle detection has been successfully created in Azure custom vision platform using Python SDK.



Projects

Project Name: Search by project name Project Type: Any project type

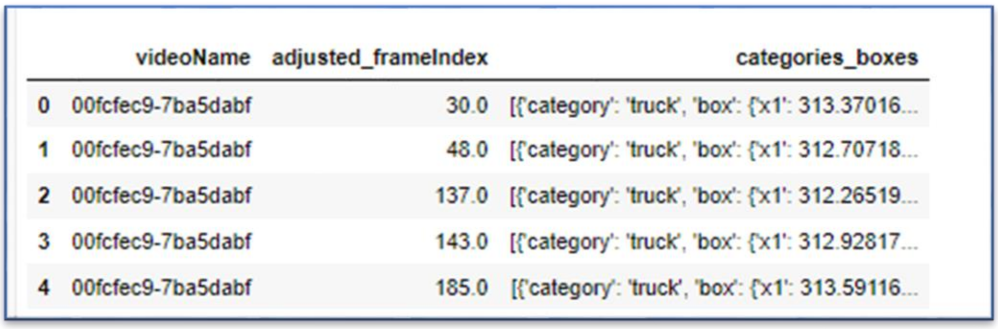
NEW PROJECT

OBJECT DETECTION
VehicleDetection
IDS223919051Task

Let's now condense the data into a single row per image. Since an image contains multiple objects represented as rows, we'll create dictionary records for categories and bounding box information to consolidate the data.

```
# Function to aggregate categories and corresponding boxes together
def aggregate_categories_and_boxes(group):
    categories_boxes = []
    for _, row in group.iterrows():
        categories_boxes.append({
            'category': row['category'],
            'box': {'x1': row['box2d.x1'], 'y1': row['box2d.y1'], 'x2': row['box2d.x2'], 'y2':
row['box2d.y2']}
        })
    return categories_boxes

# Group by 'videoName' and 'adjusted_frameIndex', then apply the aggregation function
aggregated_data = filtered_dataset.groupby(['videoName',
'adjusted_frameIndex']).apply(aggregate_categories_and_boxes).reset_index(name='catego
ries_boxes')

# aggregated data head
aggregated_data.head()
```

output:



	videoName	adjusted_frameIndex	categories_boxes
0	00fcfec9-7ba5dabf	30.0	[{'category': 'truck', 'box': {'x1': 313.37016...
1	00fcfec9-7ba5dabf	48.0	[{'category': 'truck', 'box': {'x1': 312.70718...
2	00fcfec9-7ba5dabf	137.0	[{'category': 'truck', 'box': {'x1': 312.26519...
3	00fcfec9-7ba5dabf	143.0	[{'category': 'truck', 'box': {'x1': 312.92817...
4	00fcfec9-7ba5dabf	185.0	[{'category': 'truck', 'box': {'x1': 313.59116...

Normalization:

We performed normalization of the x1, x2, y1, & y2 to ensure:

- **Coordinate System Compatibility:** Normalizing bounding box coordinates to a standard range ensures compatibility across different systems and platforms, and meets the expectations of object detection models.
- **Scale Invariance:** Normalization ensures consistent object treatment across varying image resolutions, enhancing model robustness by focusing on the object's relative space in the image.
- **Flexibility and Generalization:** Normalization enables flexible model operation across different deployment environments and improves model generalization, enhancing performance on real-world tasks.
- **Efficient Training:** Normalization simplifies the learning process by reducing coordinate value ranges and maintains stable gradients during training, preventing issues like exploding or vanishing gradients.

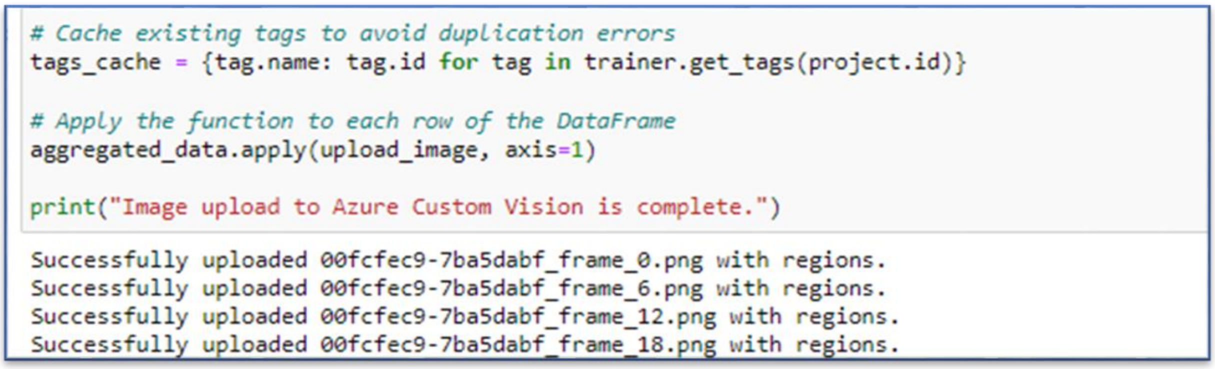
```
# Normalize the x1, x2, y1, y2
def normalize_box(box, image_width=1280, image_height=720):

  return {
    "left": box['x1'] / image_width,
    "top": box['y1'] / image_height,
    "width": (box['x2'] - box['x1']) / image_width,
    "height": (box['y2'] - box['y1']) / image_height,
  }
```

Upload Images:

```
def upload_image(row):
  image_filename = f"{row['videoName']}_frame_{int(row['adjusted_frameIndex'])}.png"
  image_path = os.path.join('ImageFrames', image_filename)

  if not os.path.exists(image_path):
    print(f"File not found: {image_path}")
    return

  regions = []
  for cb in row['categories_boxes']:
    category = cb['category']
    box = cb['box']

    if category not in tags_cache:
      # Create tag if not in cache and update the cache
      created_tag = trainer.create_tag(project.id, category)
      tags_cache[category] = created_tag.id

    tag_id = tags_cache[category]
    normalized_box = normalize_box(box)

    regions.append(Region(tag_id=tag_id, left=normalized_box['left'],
top=normalized_box['top'], width=normalized_box['width'],
height=normalized_box['height']))

  with open(image_path, "rb") as image_contents:
    tagged_images_with_regions = [ImageFileCreateEntry(name=image_filename,
contents=image_contents.read(), regions=regions)]

    upload_result = trainer.create_images_from_files(project.id,
ImageFileCreateBatch(images=tagged_images_with_regions))

    if not upload_result.is_batch_successful:
      print(f"Failed to upload image: {image_filename}")
      for image in upload_result.images:
        print("Image status: ", image.status)
    else:
      print(f"Successfully uploaded {image_filename} with regions.")
```

```
# Cache existing tags to avoid duplication errors
tags_cache = {tag.name: tag.id for tag in trainer.get_tags(project.id)}

# Apply the function to each row of the DataFrame
aggregated_data.apply(upload_image, axis=1)

print("Image upload to Azure Custom Vision is complete.")
```

Output:



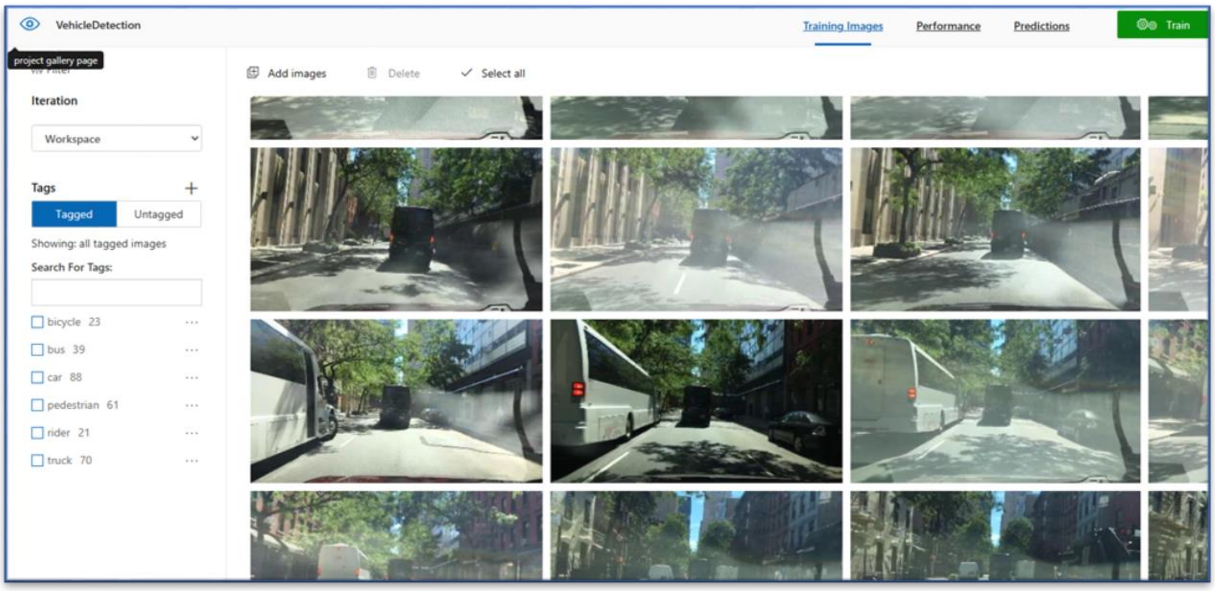
```
# Cache existing tags to avoid duplication errors
tags_cache = {tag.name: tag.id for tag in trainer.get_tags(project.id)}

# Apply the function to each row of the DataFrame
aggregated_data.apply(upload_image, axis=1)

print("Image upload to Azure Custom Vision is complete.")
```

```
Successfully uploaded 00fcfec9-7ba5dabf_frame_0.png with regions.
Successfully uploaded 00fcfec9-7ba5dabf_frame_6.png with regions.
Successfully uploaded 00fcfec9-7ba5dabf_frame_12.png with regions.
Successfully uploaded 00fcfec9-7ba5dabf_frame_18.png with regions.
```

Inference:

Loaded all the 99 images with its region and tag name information for the Azure custom vision project for the training of the model with the 6 categories

1. Bicycle
2. Car
3. Pedestrian
4. Truck
5. Bus
6. rider



Validating images in Portal:

Image 1 with region and tags.

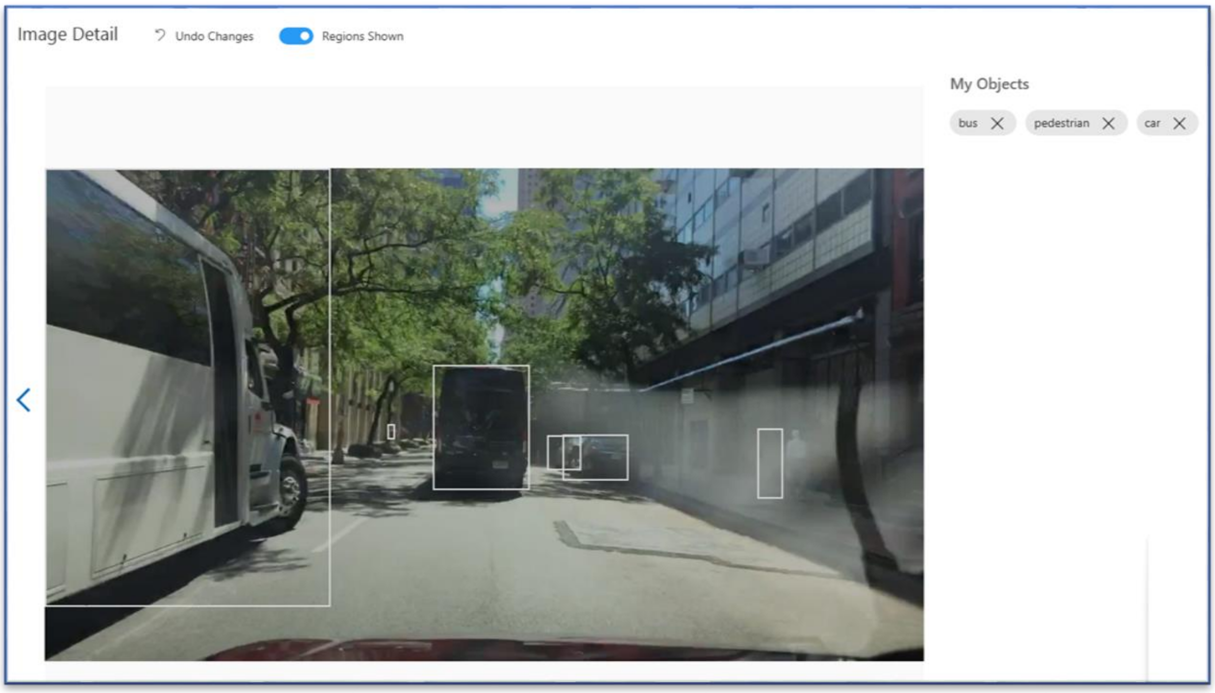
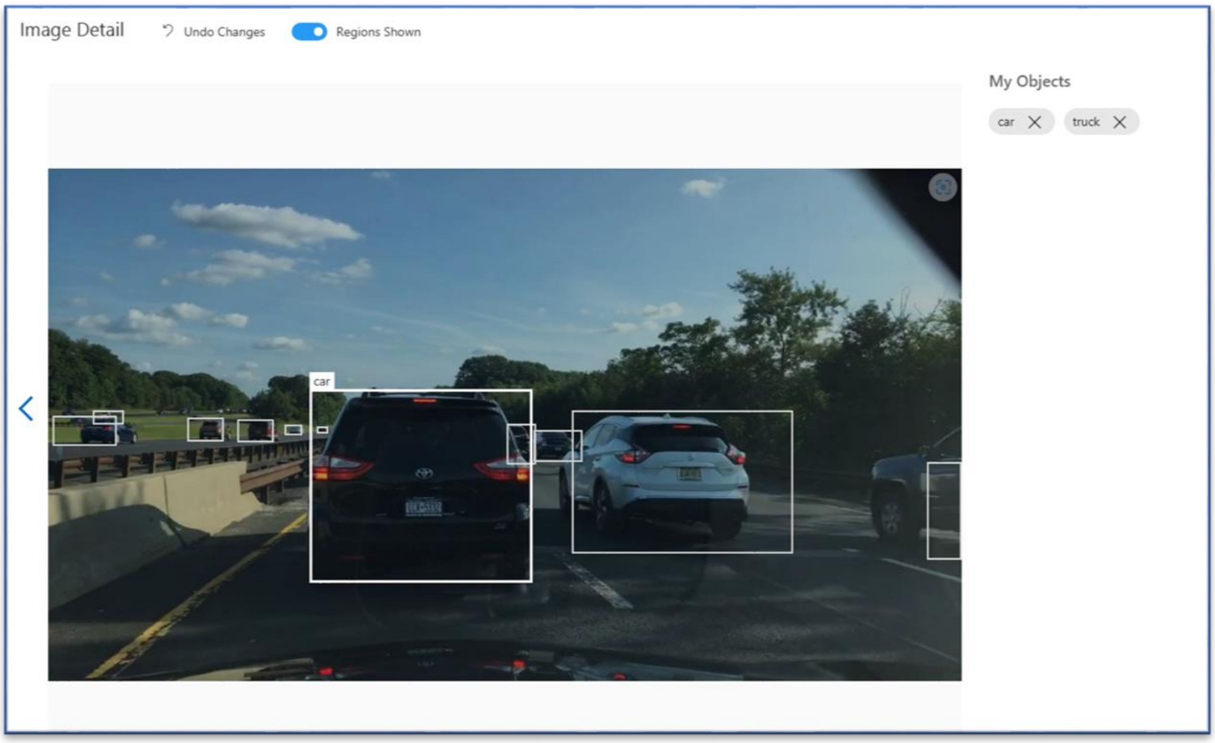


Image2 with region and tags.

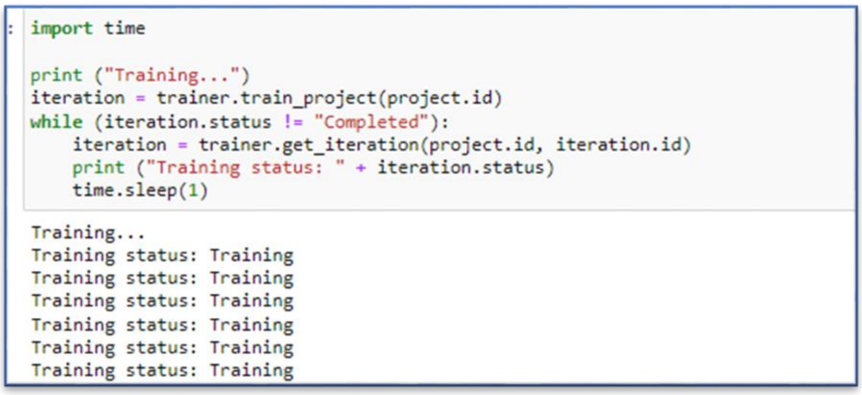


As we now successfully loaded the image along with region and tag name to Azure custom vision project, let's proceed with training.

Training:

```
import time

print ("Training...")
iteration = trainer.train_project(project.id)
while (iteration.status != "Completed"):
    iteration = trainer.get_iteration(project.id, iteration.id)
    print ("Training status: " + iteration.status)
    time.sleep(1)
```

Output:



```
import time

print ("Training...")
iteration = trainer.train_project(project.id)
while (iteration.status != "Completed"):
    iteration = trainer.get_iteration(project.id, iteration.id)
    print ("Training status: " + iteration.status)
    time.sleep(1)

Training...
Training status: Training
Training status: Training
Training status: Training
Training status: Training
Training status: Training
Training status: Training
```

- **Initiate Training**: The `trainer.train_project(project.id)` command starts the model training using the uploaded images and tags.
- **Polling for Status**: A while loop checks the training status continuously. The `trainer.get_iteration(project.id, iteration.id)` retrieves the current training iteration and prints the status.
- **Waiting for Completion**: The `time.sleep(1)` command pauses the script for a second before checking the status again, preventing excessive server requests.
- **Completion**: Once the iteration status is 'Completed', the loop exits, indicating the end of training. The model is then ready for testing or predictions.

Performance:

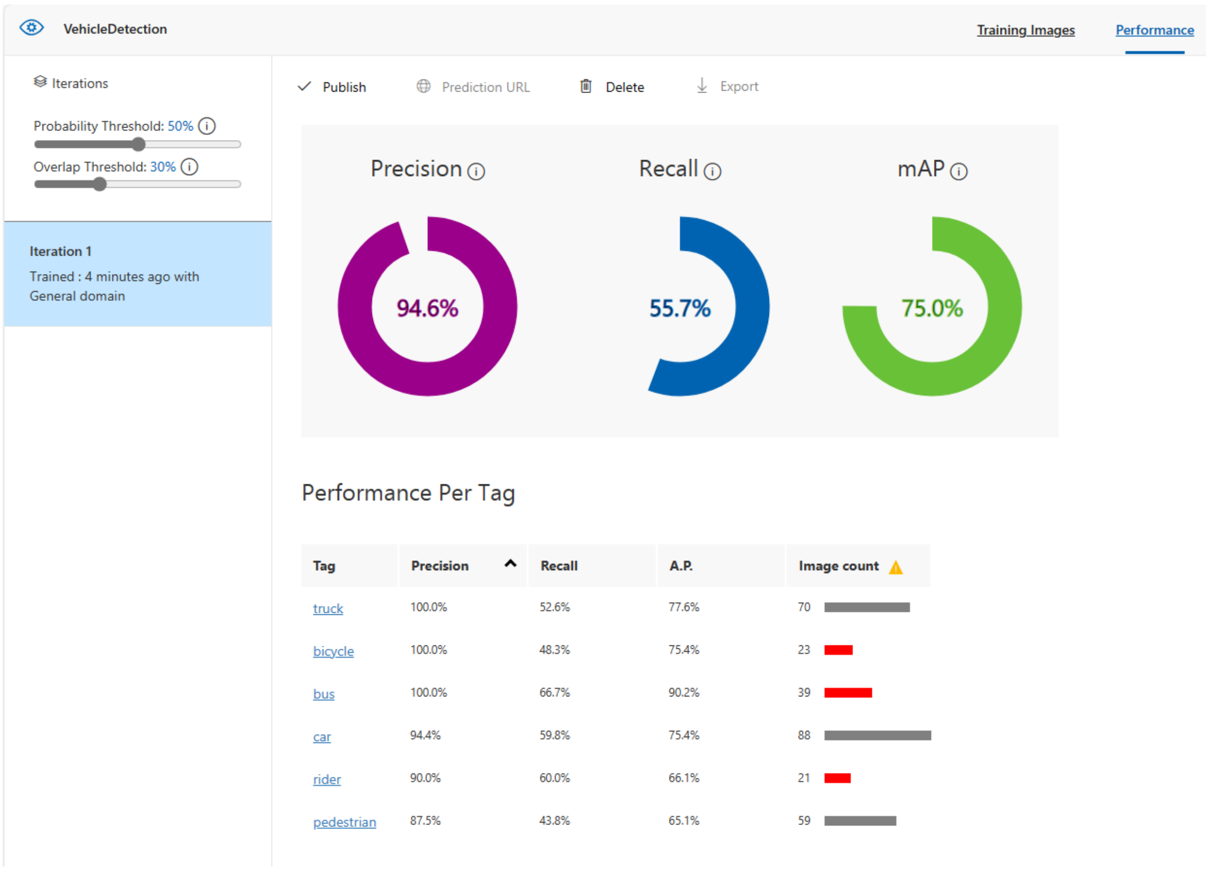
Below are the performance metrics in which the model would be evaluated.

- **Intersection over Union (IoU)**: Measures the overlap between the predicted bounding box and the ground truth box to assess localization accuracy.

- **Precision:** The ratio of true positives to the sum of true and false positives, indicating the accuracy of positive predictions.

- **Recall**: The ratio of true positives to the sum of true positives and false negatives, reflecting the model's ability to correctly identify all positives.

- **Mean Average Precision (mAP)**: The average of the Average Precision (AP) for each class, providing a score for the model's overall detection accuracy. It uses the ground-truth bounding box and the detected box for calculation.



Probability Threshold: 50%
Overlap Threshold: 30%

Iteration 1
Trained : 4 minutes ago with General domain

Precision 94.6%
Recall 55.7%
mAP 75.0%

Performance Per Tag

Tag	Precision	Recall	A.P.	Image count
truck	100.0%	52.6%	77.6%	70
bicycle	100.0%	48.3%	75.4%	23
bus	100.0%	66.7%	90.2%	39
car	94.4%	59.8%	75.4%	88
rider	90.0%	60.0%	66.1%	21
pedestrian	87.5%	43.8%	65.1%	59

Model Performance:

Thresholds:

- Probability Threshold (50%): The model classifies detections as positive if the confidence level is at least 50%.

- Overlap Threshold (30%): Predicted bounding boxes are considered true positives if they overlap at least 30% with the ground truth.

Performance Metrics:

- Precision (94.6%): The model's predictions are 94.6% accurate.

- Recall (55.7%): The model identifies 55.7% of all actual tags.

- mAP (75.0%): The model has a good balance of precision and recall across different classes.

Performance Per Tag:

- Truck: Precision is 100%, recall is 52.6%, AP is 77.6%, and image count is 70.

- Bicycle: Precision is 100%, recall is 48.3%, AP is 75.4%, and image count is 23 (red bar).

- Bus: Precision is 100%, recall is 66.7%, AP is 90.2%, and image count is 39 (red bar).

- Car: Precision is 94.4%, recall is 59.8%, AP is 75.4%, and image count is 88.

- Rider: Precision is 90.0%, recall is 60.0%, AP is 66.1%, and image count is 21 (red bar).

- Pedestrian: Precision is 87.5%, recall is 43.8%, AP is 65.1%, and image count is 59.

Image Count Imbalance:

- The red bars for 'bicycle', 'bus', and 'rider' indicates an imbalanced or insufficient dataset for these classes.

Conclusion:

The model exhibits high precision across all tags, but the recall rates are significantly lower, indicating missed true positives. This is especially evident for 'bicycle', 'bus', and 'rider' tags, which have fewer images in the dataset. The overall mAP score of 75.0% suggests a moderate balance of precision and recall. However, there's substantial room for improvement while adding more images while working on live projects.

Limitation:

The model's performance is constrained by the limited number of images due to the use of a free tier. To avoid exhausting resources on a single project and to preserve them for upcoming projects, the quantity of images is kept minimal. This limitation could potentially affect the model's learning ability, especially for imbalanced classes.

Improvement Strategies:

Improvements in recall could be achieved through data augmentation, model tuning, and adding more training data, particularly for underrepresented classes.

Testing a Sample Image:

Tested the image using browse file

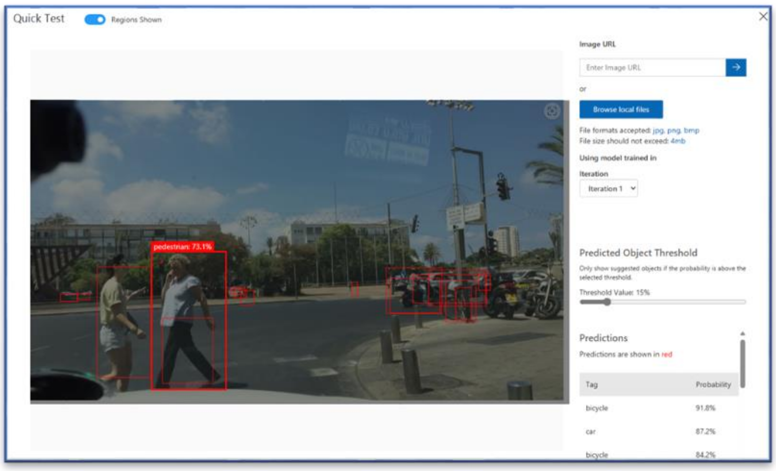


Image URL

Enter Image URL

or

File formats accepted: jpg, png, bmp
File size should not exceed: 4mb

Using model trained in

Iteration

Iteration 1

Predicted Object Threshold

Only show suggested objects if the probability is above the selected threshold.

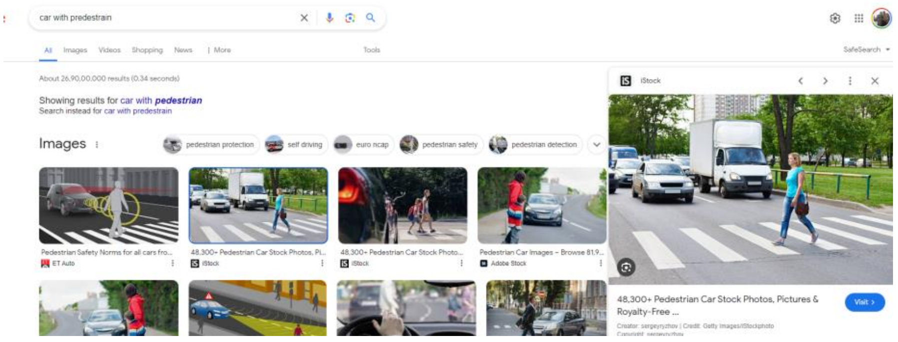
Threshold Value: 15%

Predictions

Predictions are shown in red

Tag	Probability
bicycle	91.8%
car	87.2%
bicycle	84.2%

Testing a image using web url image


car with predestrain
All Images Videos Shopping News More Tools SafeSearch
About 26,90,00,000 results (0.34 seconds)
Showing results for car with pedestrian
Search instead for car with predestrain
Images
pedestrian protection
self driving
euro ncap
pedestrian safety
pedestrian detection
Pedestrian Safety Norms for all cars fro...
ET Auto
48,300+ Pedestrian Car Stock Photos, Pi...
iStock
48,300+ Pedestrian Car Stock Photo...
iStock
Pedestrian Car Images – Browse 81,9...
Adobe Stock
iStock
48,300+ Pedestrian Car Stock Photos, Pictures & Royalty-Free ...
Visit
Creator: sergeyryzhov | Credit: Getty Images/iStockphoto

Testing:

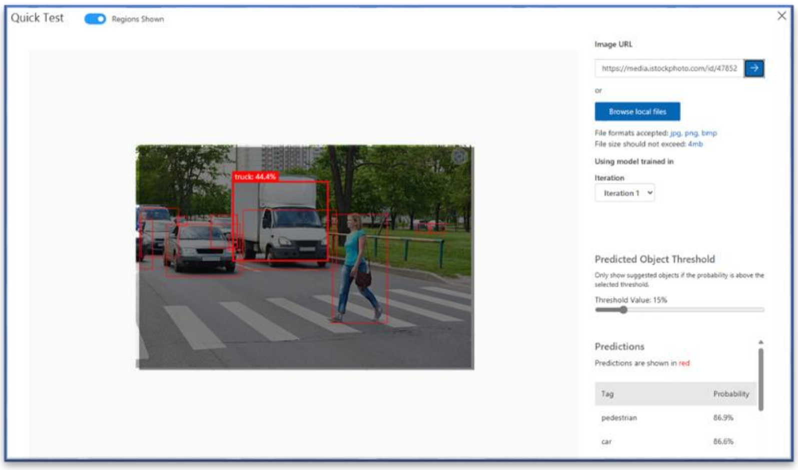


Image URL

https://media.istockphoto.com/id/47852

or

File formats accepted: jpg, png, bmp
File size should not exceed: 4mb

Using model trained in

Iteration

Iteration 1

Predicted Object Threshold

Only show suggested objects if the probability is above the selected threshold.

Threshold Value: 15%

Predictions

Predictions are shown in red

Tag	Probability
pedestrian	86.9%
car	86.6%

Test Results:

Truck Detection: The model correctly identifies a truck with a confidence of 44.4%. The bounding box accurately encompasses the vehicle.

Pedestrian Detection: The model identifies a pedestrian with a high confidence of 86.9%, assuming the bounding box accurately encloses a person.

Car Detection: The model identifies a car with a high confidence of 86.6%, provided the bounding box correctly identifies a car in the image.

Quick Test Insights:

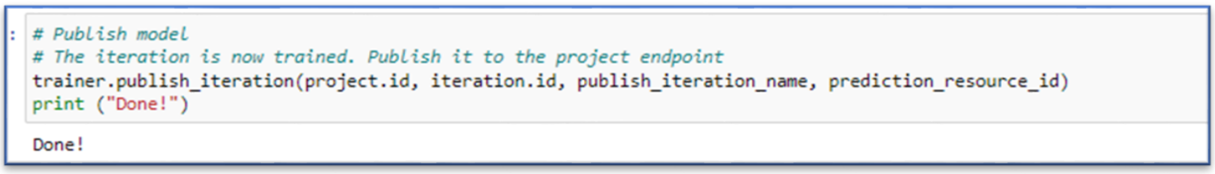
These tests have been conducted on the random web URL image provides an immediate sense of the model's performance on individual instances. However, they may not fully reflect the model's accuracy in a production environment as they are based on single instances rather than a diverse set of test data. Let's publish the model and test the same with a video.

Publish Model:

Code:

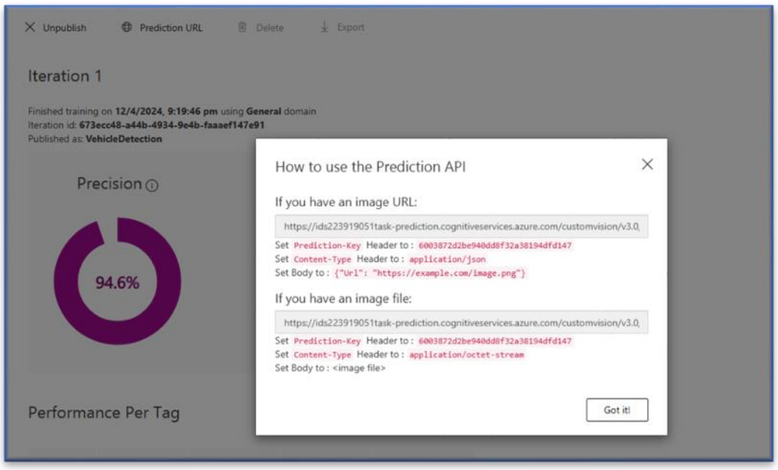
```
# Publish model
# The iteration is now trained. Publish it to the project endpoint
trainer.publish_iteration(project.id, iteration.id, publish_iteration_name,
prediction_resource_id)
print ("Done!")
```

Output:



```
# Publish model
# The iteration is now trained. Publish it to the project endpoint
trainer.publish_iteration(project.id, iteration.id, publish_iteration_name, prediction_resource_id)
print ("Done!")

Done!
```

Prediction & Testing:



Iteration 1

Finished training on **12/4/2024, 9:19:46 pm** using **General** domain
Iteration id: **673ecc48-a44b-4934-9e4b-faaaef147e91**
Published as: **VehicleDetection**

How to use the Prediction API

If you have an image URL:

```
https://ids223919051task-prediction.cognitiveservices.azure.com/customvision/v3.0,
Set Prediction-Key Header to : 6003872d2be940dd8f32a38194dfd147
Set Content-Type Header to : application/json
Set Body to : {"Url": "https://example.com/image.png"}
```

If you have an image file:

```
https://ids223919051task-prediction.cognitiveservices.azure.com/customvision/v3.0,
Set Prediction-Key Header to : 6003872d2be940dd8f32a38194dfd147
Set Content-Type Header to : application/octet-stream
Set Body to : <image file>
```

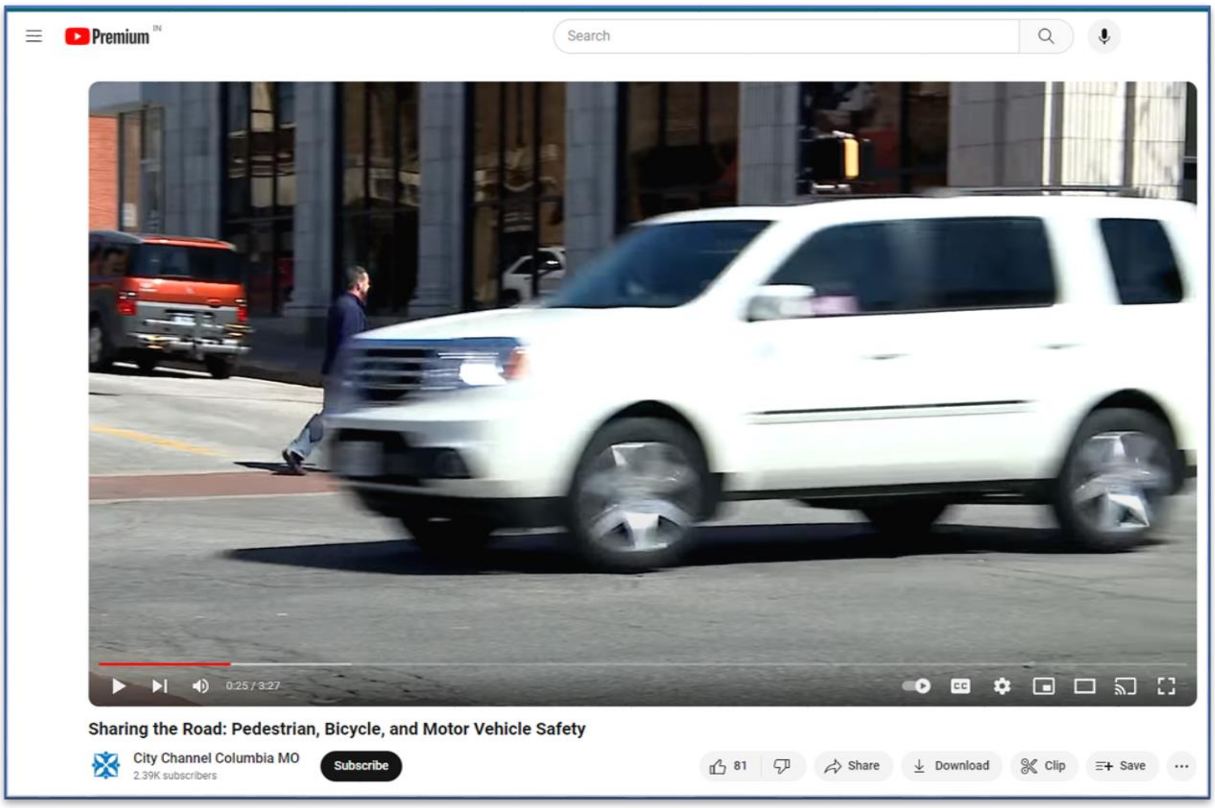
As now we have published the model and created the prediction URL, now let's create the Python SDK to test the prediction URL for the image from local machine by sending a 1 minute video file selected from youtube.

Youtube Video:

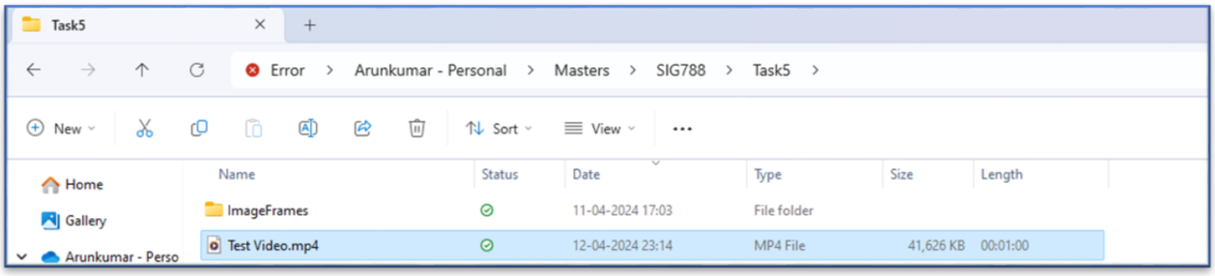
I have selected the below three minutes video for testing the model prediction.

URL: https://www.youtube.com/watch?v=jj8CfZDPeBs

Video Selected:



The video has been downloaded using the link https://ssyoutube.com/en169ZG/youtube-video-downloader to the local machine and then used video trimmer from the link https://online-video-cutter.com/#google_vignette to trim the video to 1 minute and saved the video as **Test Video.mp4** in the working directory.



Name	Status	Date	Type	Size	Length
ImageFrames	✓	11-04-2024 17:03	File folder		
Test Video.mp4	✓	12-04-2024 23:14	MP4 File	41,626 KB	00:01:00

Then the test video.mp4 has been sent to the model for the prediction & detecting the object detection.

Code:

Function to extract the frame per second.

```
# function to extract frame per second
def extract_frames(video_path, frame_interval):

    # Initialize video capture
    cap = cv2.VideoCapture(video_path)

    # Get the frames per second (fps) of the video
    fps = cap.get(cv2.CAP_PROP_FPS)

    # Initialize frame count
    frame_count = 0

    # Loop through video frames
    while cap.isOpened():
        ret, frame = cap.read()  # Read a frame from the video
        if not ret:
            break

        # Check if the current frame number for interval
        if frame_count % int(fps * frame_interval) == 0:
            yield frame

        frame_count += 1  # Increment the frame count

    # Release the video capture object to free resources
    cap.release()
```

Function to send frame to Azure custom vision model using the prediction key and prediction url for the image.

```
# Function to send frame to Azure prediction Api & return Json response
def predict_frame(image, url, prediction_key):

    headers = {
        'Content-Type': 'application/octet-stream',  # Indicates that the body contains binary
data.
        'Prediction-Key': prediction_key          # Authentication key required by the API.
    }
    # Sends the POST request to the API.
    response = requests.post(url, headers=headers, data=image)

    # Parses and returns the response as JSON.
    return response.json()
```

Function to draw the bounding boxes and labelling for the predicted image with the given threshold.

```
# function to draw bounding boxes and label to the frame with the probalility threshold
```

```
def draw_predictions(frame, predictions, probability_threshold=0.001):

    # loop through the predicitons
    for prediction in predictions['predictions']:
        probability = prediction['probability']

        # Only draw predictions with probability
        if probability > probability_threshold:

            # Get the bounding box for the prediction
            bbox = prediction['boundingBox']

            # Get the label for the prediction
            tag = prediction['tagName']

            # Calculate the pixel coordinates of the bounding box
            left = int(bbox['left'] * frame.shape[1])
            top = int(bbox['top'] * frame.shape[0])
            width = int(bbox['width'] * frame.shape[1])
            height = int(bbox['height'] * frame.shape[0])

            # Draw a rectangle around the detected object
            cv2.rectangle(frame, (left, top), (left + width, top + height), (0, 255, 0), 2)

            # Draw the label and probability for bounding box
            label = f"{tag}: {probability:.2f}"
            cv2.putText(frame, label, (left, top - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255,
0), 2)

    return frame
```

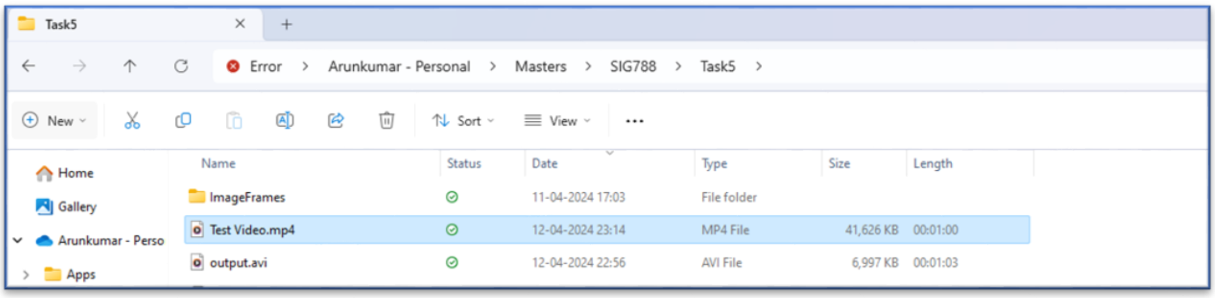
Now let’s pass the test video into the model prediction to object detect in the video **one frame per second** to reduce the usage cost in Azure.

```
# Test Video file
video_file = 'Test Video.mp4'

# To read load the prediction URL
load_dotenv()

#Read the predition URL from environment variables
prediction_url = os.environ["Vision_PREDICTION_URL"]

# Initialize video capture to get video properties
cap = cv2.VideoCapture(video_file)
ret, frame = cap.read()
height, width = frame.shape[:2]
cap.release()

# Initialize VideoWriter to save output
```

```
fourcc = cv2.VideoWriter_fourcc(*'XVID')
out = cv2.VideoWriter('output.avi', fourcc, 1.0, (width, height))

# Extract and predict using one frame per second
for frame in extract_frames(video_file, frame_interval=1.0):  # Extract one frame per second
    _, buffer = cv2.imencode('.jpg', frame)
    response = predict_frame(buffer.tobytes(), prediction_url, prediction_key)
    frame_with_predictions = draw_predictions(frame, response, 0.5)  # Apply thresholds of
50%
    out.write(frame_with_predictions)

# Release the VideoWriter
out.release()
```

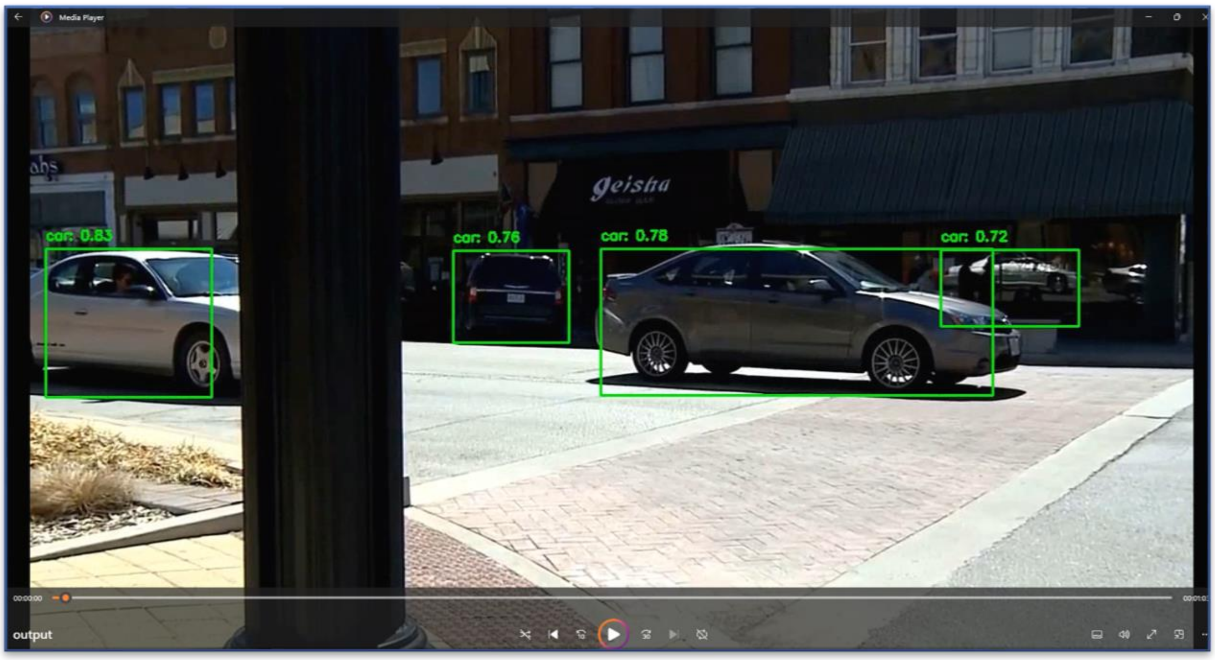
Output:

The output.avi has been created and saved in the working directory.


Task5

Error > Arunkumar - Personal > Masters > SIG788 > Task5 >

Name	Status	Date	Type	Size	Length
ImageFrames	✓	11-04-2024 17:03	File folder		
Test Video.mp4	✓	12-04-2024 23:14	MP4 File	41,626 KB	00:01:00
output.avi	✓	12-04-2024 22:56	AVI File	6,997 KB	00:01:03



Conclusion:

The project successfully automated for object detection using videos by extracting frames from YouTube videos and predicting using an Azure Custom Vision model.

Limitations:

- **Limited Training Data**: The model was trained with only **99 images**, limiting its ability to generalize across varied real-world scenarios.

- **Model Overfitting**: There's a risk of overfitting due to the limited training data.

- **Frame Rate Dependency**: The current setup uses only **one frame per second** will not be adequate for all scenarios.

Next Steps:

- **Increase Training Data**: Enhance the training dataset with more diverse images to improve model accuracy.

- **Enhance Model Training**: Experiment with different architectures and hyperparameters, and consider data augmentation techniques.

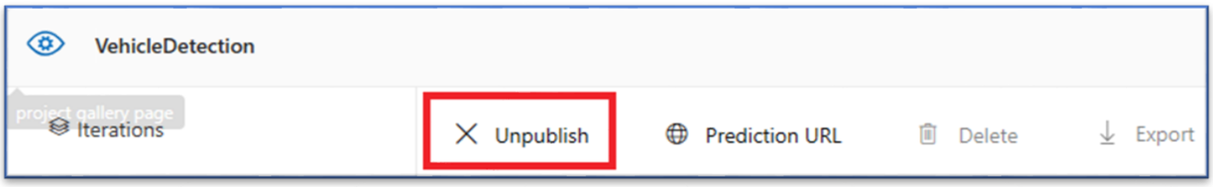
- **Implement Continuous Learning**: Establish a mechanism for continuous learning from new data, including incorrect predictions.

- **Refine Thresholds and Evaluation Metrics**: Adjust the probability threshold and implement additional metrics like F1-score for a comprehensive understanding of the model's performance.

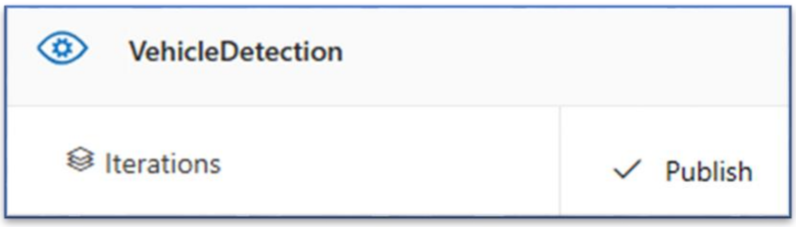
Clean up Activity:

As we now completed the project, let's delete the resource and custom vision project.

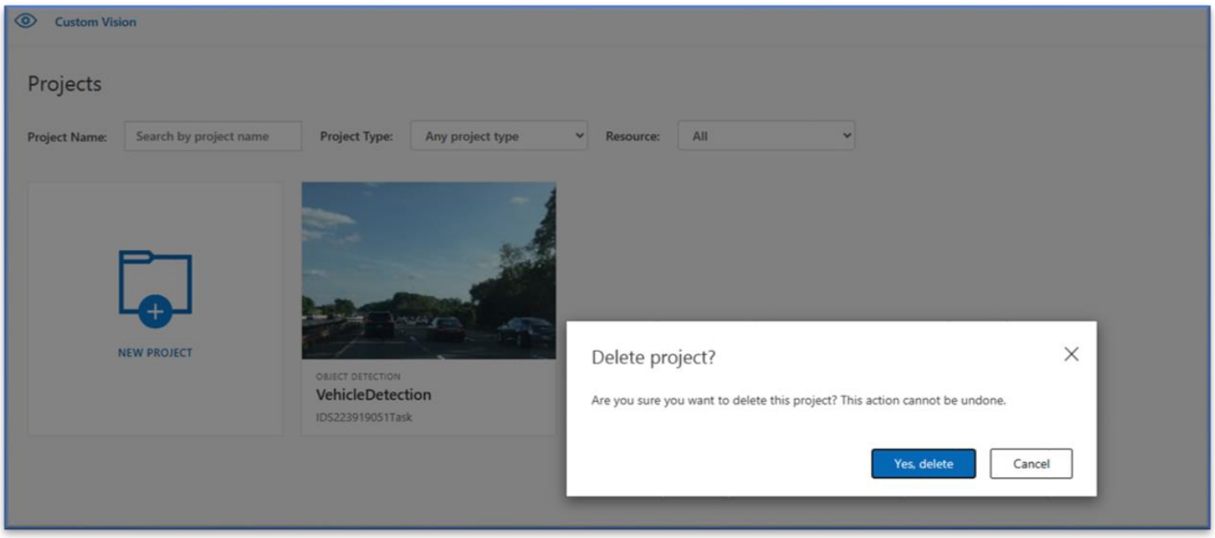
Unpublish the Model:



Unpublished:



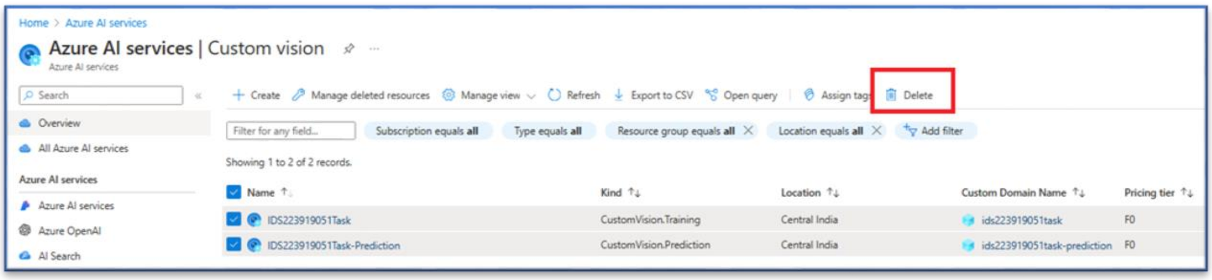
Deleting the project:



Delete project?

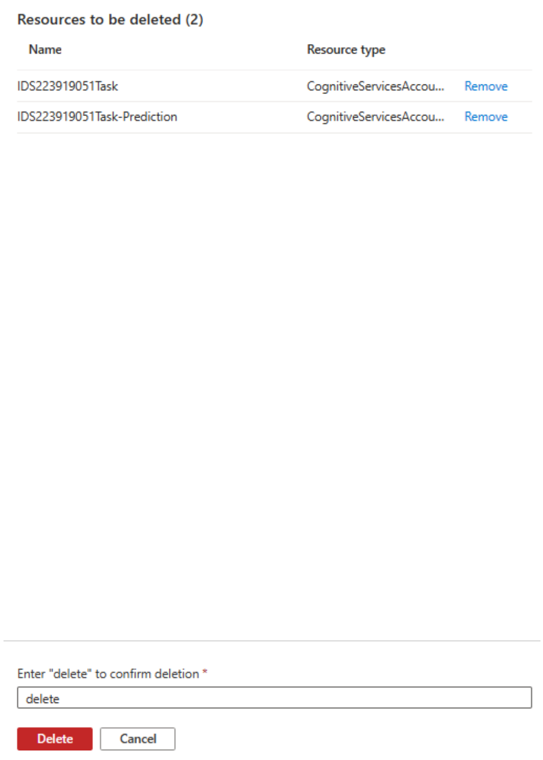
Are you sure you want to delete this project? This action cannot be undone.

Deleted the project, now we will get into Azure portal to delete the resources and resources group.



Showing 1 to 2 of 2 records.

Name	Kind	Location	Custom Domain Name	Pricing tier
IDS223919051Task	CustomVision.Training	Central India	ids223919051task	F0
IDS223919051Task-Prediction	CustomVision.Prediction	Central India	ids223919051task-prediction	F0



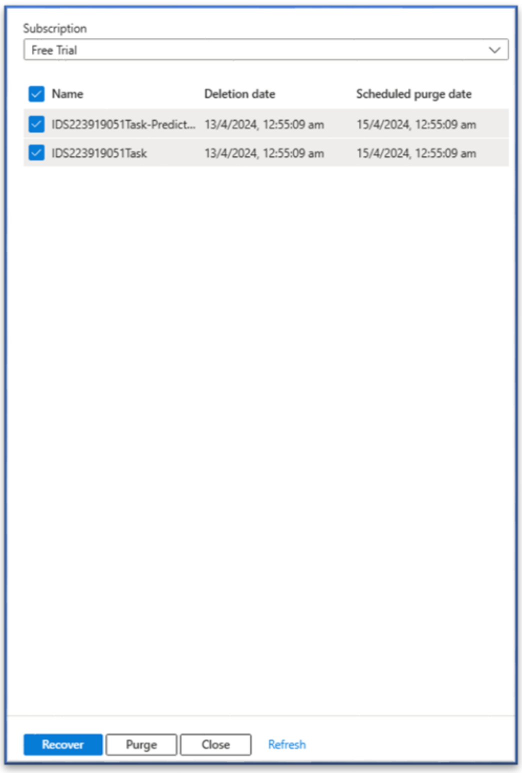
Resources to be deleted (2)

Name	Resource type	
IDS223919051Task	CognitiveServicesAccou...	Remove
IDS223919051Task-Prediction	CognitiveServicesAccou...	Remove

Enter "delete" to confirm deletion *

delete

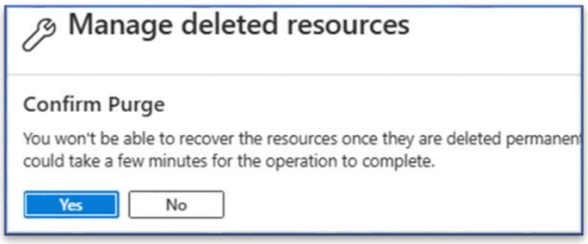
Now click on managed deleted resources and purge the deleted resources permanently



Subscription

Free Trial

✓ Name	Deletion date	Scheduled purge date
✓ IDS223919051Task-Predict...	13/4/2024, 12:55:09 am	15/4/2024, 12:55:09 am
✓ IDS223919051Task	13/4/2024, 12:55:09 am	15/4/2024, 12:55:09 am

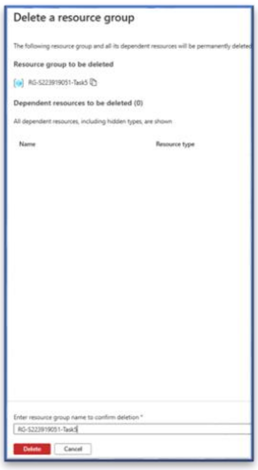


Manage deleted resources

Confirm Purge

You won't be able to recover the resources once they are deleted permanen
could take a few minutes for the operation to complete.

Now click on resource groups and delete resource group



Delete a resource group

The following resource group and all its dependent resources will be permanently deleted

Resource group to be deleted

RG-S223919051-Task5

Dependent resources to be deleted (0)

All dependent resources, including hidden types, are shown

Name	Resource type

Enter resource group name to confirm deletion *

RG-S223919051-Task5

After finishing the work, I have ensured to delete all the resources and resource group to ensure no extra billing for unused resources.

References:

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

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

- Microsoft Learn (no.date.) " **Custom Vision documentation**" Available at: https://learn.microsoft.com/en-us/azure/ai-services/custom-vision-service/

- Microsoft Learn (no.date.) " **Quickstart: Create an image classification project with the Custom Vision client library or REST API**" Available at: https://learn.microsoft.com/en-us/azure/ai-services/custom-vision-service/quickstarts/image-classification?tabs=windows%2Cvisual-studio&pivots=programming-language-python

- Microsoft Learn (no.date.) " **Quickstart: Create an object detection project with the Custom Vision client library**" Available at: https://learn.microsoft.com/en-us/azure/ai-services/custom-vision-service/quickstarts/object-detection?tabs=windows%2Cvisual-studio&pivots=programming-language-python

- Furturelearn (no.date) "**Extracting video frames using OpenCV**" available at: https://www.futurelearn.com/info/courses/introduction-to-image-analysis-for-plant-phenotyping/0/steps/305359

- City Channel Columbia MO (Apr 10, 2015) "**Sharing the Road: Pedestrian, Bicycle, and Motor Vehicle Safety**" Used as test image and Available at: https://www.youtube.com/watch?v=jj8CfZDPeBs

- ssyoutube (no.date) "**Video Download from youtune**" available at: https://ssyoutube.com/en169ZG/youtube-video-downloader

- onlinevideocutter (no.date) "**Video Trimmer**" available at: https://online-video-cutter.com/#google_vignette