Task 5 D: Custom vision

Prerequisites

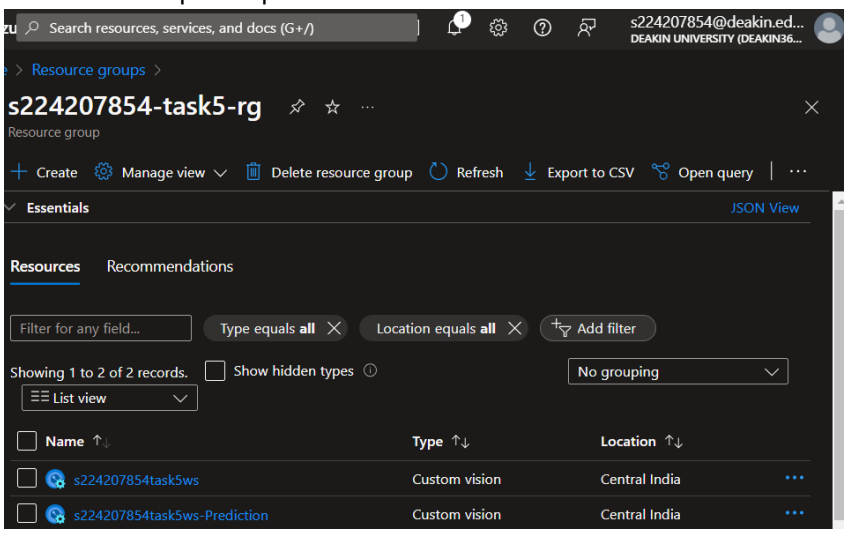
- Install libraries that will be required for this task

```
# OpenCV to perform preprocessing on video files to convert to image files
%pip install opencv-python

# Azure Computervision library:
%pip install azure-cognitiveservices-vision-customvision

# Pillow library
%pip install pillow

# dotenv library
%pip install python-dotenv
```

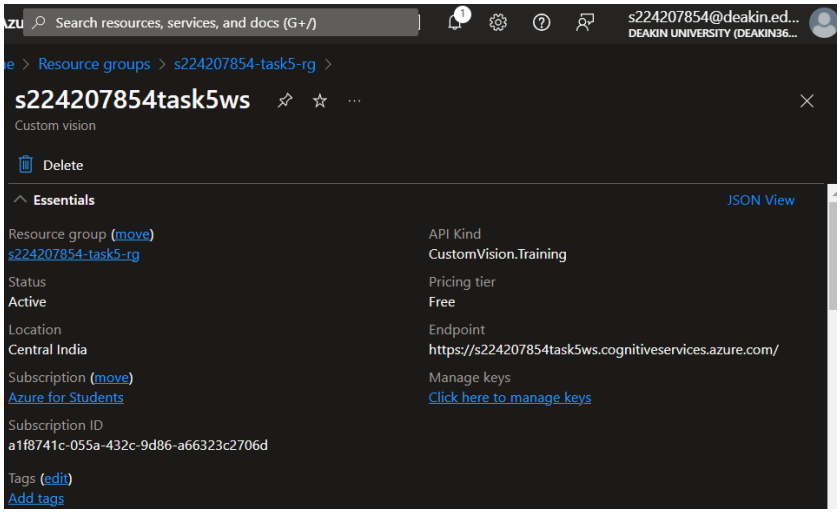
- Import libraries

```
# Importing libraries for Azure Custom Vision
from azure.cognitiveservices.vision.customvision.prediction import CustomVisionPredictionClient
from msrest.authentication import ApiKeyCredentials

# Importing PIL for image manipulation
from PIL import Image, ImageDraw

# Importing os and dotenv for handling secret keys
import os
from dotenv import load_dotenv

import cv2
```

- Setting Azure Custom Vision in Azure portal
    - Resource Group set-up:



    - Creating custom vision service in Azure AI service



Azure Custom Vision Notebook

In our notebook we will be using azure custom vision client to analyze different images and videos as input and then detecting the objects in videos.

Reading the environment variables to authenticate to the Computer Vision resource.

```
# Read the environment variables to authenticate to the Computer Vision resource
load_dotenv()

prediction_key = os.environ.get("VISION_PREDICTION_KEY")
custom_vision_endpoint = os.environ.get("VISION_PREDICTION_ENDPOINT")
custom_vision_project_id = "29d2284c-cc88-4ff4-b09a-b3ba713ff927"
```

We're going to use a tool called OpenCV to change videos into pictures. First, we need to do something called "object detection." This means finding things in the video. Here's how we do it:

- Using OpenCV: This is a tool we'll use to work with videos.
- Converting the Video: We'll use something called cv2 VideoCapture to take the video and turn it into lots of pictures.
- Going through the Frames: We'll look at each picture, one by one, and change it into its own separate picture. Then, we can study each picture on its own.

This code uses two modules called OpenCV and OS. It's made to take frames out of video files. The function called "save_frames_from_video" does a few things. First, it opens a video file. Then, it makes a new folder where it will put the frames. After that, it counts all the frames in the video. Next, it goes through each frame. It saves every 100th frame as a picture in the new folder.

```python
# Function to save frames from a video
def save_frames_from_video(video_path):
    # Open the video file
    cap = cv2.VideoCapture(video_path)
    if not cap.isOpened():
        print(f"Error: Couldn't open video file {video_path}")
        return

    # Create output directory if it doesn't exist
    output_dir = os.path.splitext(os.path.basename(video_path))[0] + "_frames"
    os.makedirs(output_dir, exist_ok=True)

    # Get total number of frames in the video
    n_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))

    # Iterate through each frame
    for frame in range(n_frames):
        ret, img = cap.read()
        if not ret:
            print(f"Error: Failed to read frame {frame} from {video_path}")
            break
        if frame % 100 == 0:
            # Save frame as PNG image
            frame_path = os.path.join(output_dir, f"frame_{frame:06d}.png")
            cv2.imwrite(frame_path, img)

    cap.release()


# Directory containing input video files
input_directory = "videos/"

file_counter = 0
for filename in os.listdir(input_directory):
    if filename.endswith(".mov") or filename.endswith(".mp4"):
        print(file_counter)
        video_path = os.path.join(input_directory, filename)
        save_frames_from_video(video_path)
        print(f"Frames extracted from {filename}.")
        file_counter += 1

print("All videos processed.")
```

2m 12.3s

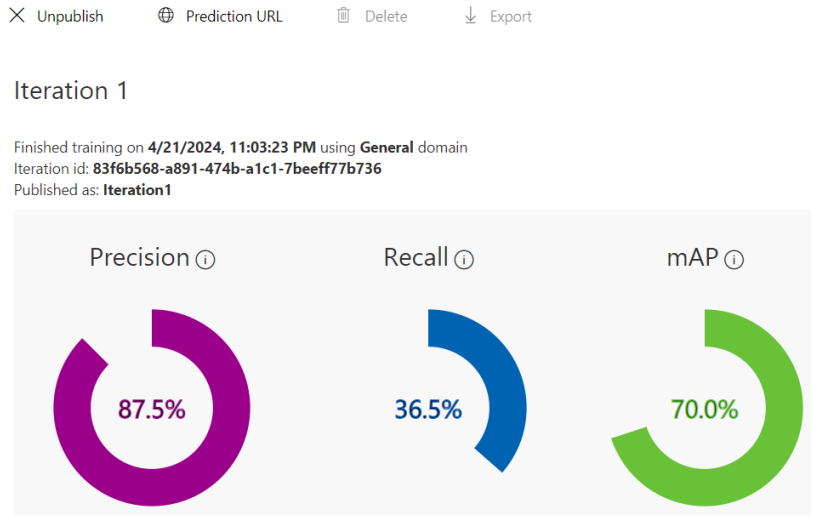
Output:

```
0
Frames extracted from 0000f77c-6257be58.mov.
1
Frames extracted from 0000f77c-62c2a288.mov.
2
Frames extracted from 0000f77c-cb820c98.mov.
3
Frames extracted from 0001542f-5ce3cf52.mov.
4
Frames extracted from 0001542f-7c670be8.mov.
5
Frames extracted from 0001542f-ec815219.mov.
6
Frames extracted from 0004974f-05e1c285.mov.
7
Frames extracted from 00054602-3bf57337.mov.
8
Frames extracted from 00067cfb-5443fe39.mov.
9
Frames extracted from 00067cfb-5adfaaa7.mov.
10
Frames extracted from 00067cfb-caba8a02.mov.
11
Frames extracted from 00067cfb-e535423e.mov.
12
Frames extracted from 00067cfb-f1b91e3c.mov.
13
Frames extracted from 0008a165-c48f4b3e.mov.
All videos processed.
```

Uploading images for Model Training

First, after getting pictures from the video, I put them into a project on Azure Custom Vision to teach a computer model.

- Tagging: I looked at each picture and put labels on them like "car", "person", "big vehicle" and more.
- Teaching the Model: I uploaded the labeled pictures to Azure Custom Vision. The computer uses these to learn how to find and name things in the pictures. This helps it to understand what's in the video frames better.



We're doing well in recognizing objects, with an accuracy of 87.5%. But we're not so good at finding all the objects, with recall rate 36.5% of them being detected. Overall, our performance is good, with a mean average precision(mAP) of 70%.

Converting Test Video to Image Frame

I again defined the function save_frames_from_video that extracts frames from a video file in the prediction/ directory and saves them as PNG images inside a directory with suffix _frames.

```python
image_directory_for_prediction = "00091078-59817bb0_frames"
output_dir = os.path.join(image_directory_for_prediction, "predictions")
os.makedirs(output_dir, exist_ok=True)

threshold = 0.4  # Adjust as needed


# Function to draw bounding boxes on image
def draw_boxes(image, predictions):
    draw = ImageDraw.Draw(image)
    for prediction in predictions:
        if prediction.probability > threshold:
            left = prediction.bounding_box.left * image.width
            top = prediction.bounding_box.top * image.height
            right = left + (prediction.bounding_box.width * image.width)
            bottom = top + (prediction.bounding_box.height * image.height)
            draw.rectangle([left, top, right, bottom], outline="red")
            draw.text((left, top), prediction.tag_name, fill="red")


# Iterate over images and perform predictions
for filename in os.listdir(image_directory_for_prediction):
    if filename.endswith(".jpg") or filename.endswith(".png"):
        image_path = os.path.join(image_directory_for_prediction, filename)
        with open(image_path, "rb") as image_contents:
            print(image_path)
            predictions = predictor.detect_image(
                custom_vision_project_id, published_model_name, image_contents.read()
            )

        # Load image
        image = Image.open(image_path).convert("RGB")

        # Draw bounding boxes on image
        draw_boxes(image, predictions.predictions)

        # Save predicted image with bounding boxes
        predicted_image_path = os.path.join(output_dir, filename)
        image.save(predicted_image_path)

print("All predictions saved successfully.")
```

22m 18.5s

Predicting Object Tags Using Azure Custom Vision SDK

In this section, I utilize the Azure Custom Vision SDK to predict object tags within images extracted from the a test video. The provided code provides the necessary credentials and parameters for prediction which includes the prediction key, endpoint, model name, and threshold for confidence

```python
# Function to save frames from a video
def save_frames_from_video(video_path):
    # Open the video file
    cap = cv2.VideoCapture(video_path)
    if not cap.isOpened():
        print(f"Error: Couldn't open video file {video_path}")
        return

    # Create output directory if it doesn't exist
    output_dir = os.path.splitext(os.path.basename(video_path))[0] + "_frames"
    print(f"output_dir = {output_dir}")
    os.makedirs(output_dir, exist_ok=True)

    # Get total number of frames in the video
    n_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
    print(f"No. of Frames in the video = {n_frames}")

    # Iterate through each frame
    for frame in range(n_frames):
        ret, img = cap.read()
        if not ret:
            print(f"Error: Failed to read frame {frame} from {video_path}")
            break
        #         if frame % 300 == 0:
        # Save frame as PNG image
        frame_path = os.path.join(output_dir, f"frame_{frame:06d}.png")
        cv2.imwrite(frame_path, img)

    cap.release()


# Directory containing input video files
input_directory = "prediction/"

for filename in os.listdir(input_directory):
    if filename.endswith(".mov") or filename.endswith(".mp4"):
        video_path = os.path.join(input_directory, filename)
        save_frames_from_video(video_path)
        print(f"Frames extracted from {filename}.")

print("All videos processed.")
```

✓ 1m 24.1s

Converting the Video back to MP4 with Bounding Box

```python
# Directory containing predicted images
predicted_dir = "00091078-59817bb0_frames/predictions"

# Output video file path
output_video_path = (
    "predicted_video.mp4"  # You can change the extension to .mov if needed
)

# Get the list of predicted image filenames
image_files = [
    f for f in os.listdir(predicted_dir) if f.endswith(".jpg") or f.endswith(".png")
]
image_files.sort()  # Ensure the images are sorted properly

# Get the dimensions of the first image to initialize the video writer
first_image = cv2.imread(os.path.join(predicted_dir, image_files[0]))
height, width, _ = first_image.shape

# Define the codec and create VideoWriter object
fourcc = cv2.VideoWriter_fourcc(*"mp4v")  # Change 'mp4v' to 'avc1' for .mov format
fps = 30  # Adjust as needed
out = cv2.VideoWriter(output_video_path, fourcc, fps, (width, height))

# Iterate through predicted images and add them to the video
for filename in image_files:
    image_path = os.path.join(predicted_dir, filename)
    frame = cv2.imread(image_path)
    out.write(frame)

# Release the VideoWriter and close all OpenCV windows
out.release()
cv2.destroyAllWindows()

print(f"Video saved as {output_video_path}")
```

47.5s