GCP REIMAGINE IDEATHON 2023

gcds-oht33765u9-2023

TEAM MEMBERS

Mayank Harlalka
(Analyst – Data Science)
DNA (ANALYTICS)
CHD_SEZ
Mayank.harlalka@infosys.com
+919988003987

SOLUTION : 3 USE CASES

1. Trap Camera Animal Detection
2. Drone Search and Rescue
3. Driver Drowsiness Detection

Filter Filter buckets

Name ↑	Created	Location type	Location	Default storage class	Last modified	Public ac
drone-search-rescue-data	Jul 8, 2023, 3:00:28 PM	Region	europe-west6	Standard	Jul 8, 2023, 3:00:28 PM	Not publ
drowsiness-data	Jul 5, 2023, 11:38:19 PM	Region	us-central1	Standard	Jul 5, 2023, 11:38:19 PM	Not publ
trap-camera-data	Jul 5, 2023, 10:27:33 AM	Region	europe-west9	Standard	Jul 20, 2023, 12:15:55 PM	Not publ

TRAP-CAMERA ANIMAL DETECTION

UseCase Type:

Model Deployment : Cloud
Detection : Cloud
Storage : Cloud
Action Taken : On Cloud (Alerting Concerned Services/ People)

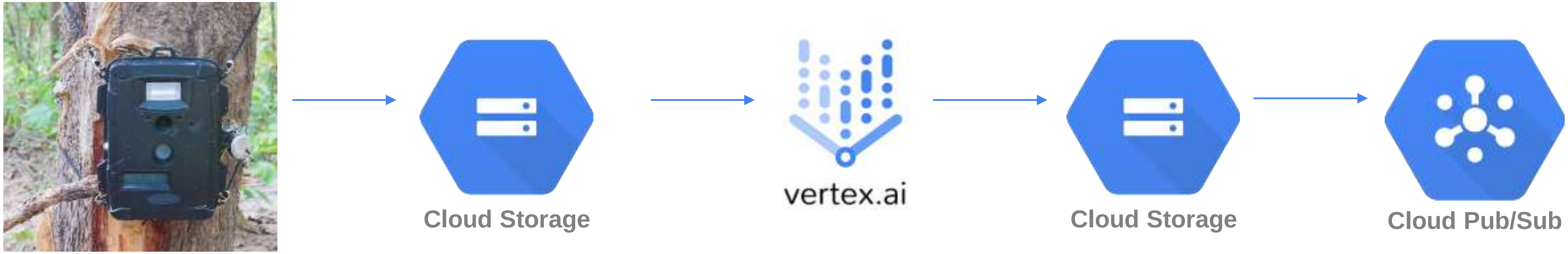
OVERVIEW

Forest Rangers, Wildlife Conservationists etc. setup Trap Cameras in the wild to monitor the activities of animals and to get photographs of animals that tend to be elusive.

Hours are spent manually going through each and every video to across even a tiniest hint of an animal.

Hence we can automate this process of going through the videos and get the animals detected easily.

CURRENT WORKING ARCHITECTURE



Videos from Tap Camera come in batches daily.

e.g. 24 1-hour videos for yesterday

The videos land in a particular folder inside a bucket in Cloud Storage

Vertex AI jupyter notebooks have pre-defined python code, that can be run to start the detection in videos

The code automatically stores the detections in the bucket. The detections are images with bounding boxes around the animal found. Also, the detections are named as VideoName-Timestamp.jpg

As soon as new detections are stored in the bucket, Cloud PubSub will trigger a notification to subscribed web services, which can ingest the topic to send email/SMS to concerned people

EXAMPLE SCENARIO



In our Wildlife Park in Indonesia, we have the following 5 species of concern, which we would like to track and monitor:

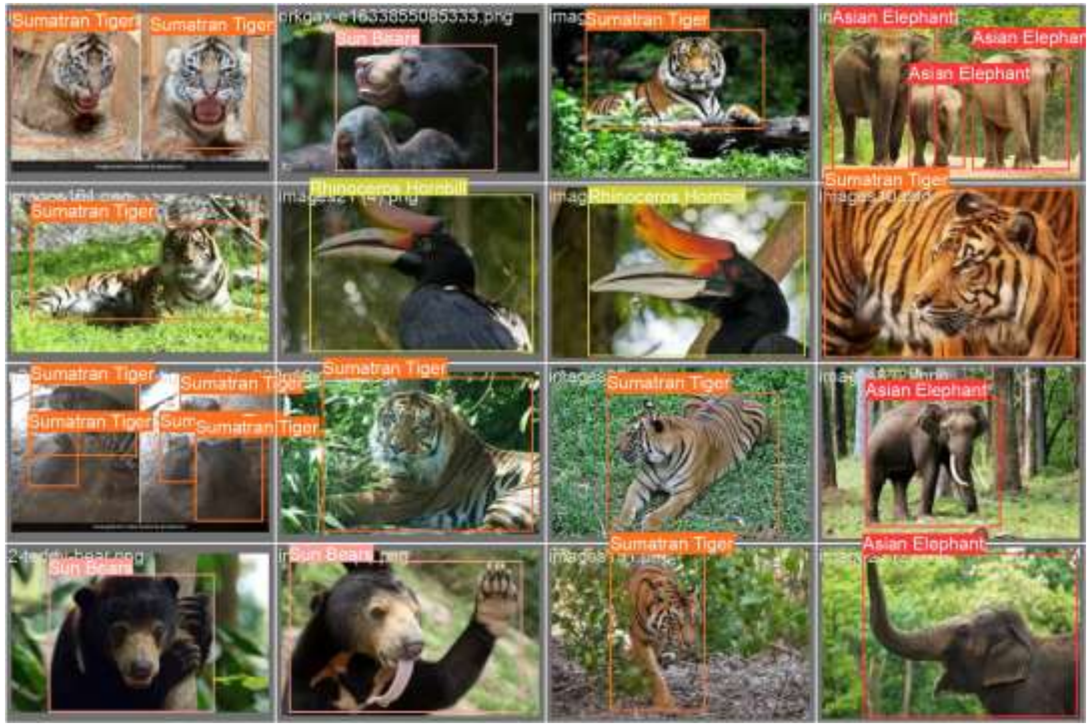
- Sumatran Tiger (Panthera tigris sumatrae)
- Asian Elephant (Elephas maximus)
- Pig Tailed Macaque (Macaca nemestrina)
- Rhinoceros Hornbill (Buceros rhinoceros Linnaeus)
- Sun Bears (Helarctos malayanus)


We would like to get notified about the image capture of any of these in the previous day. The images should contain the detection bounding box across the animal, with its name.

Also, the image file name should contain the video name, and the timestamp of detection in the video

MODEL USED

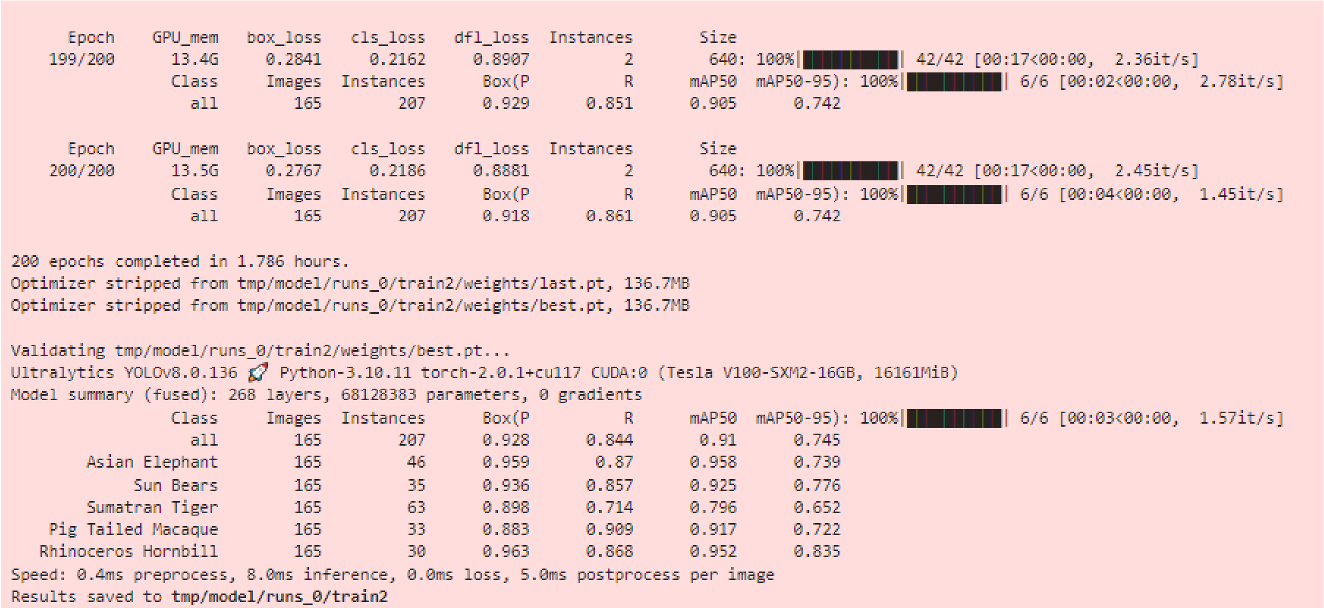
YOLOv8 Object Detection Model (https://docs.ultralytics.com/)

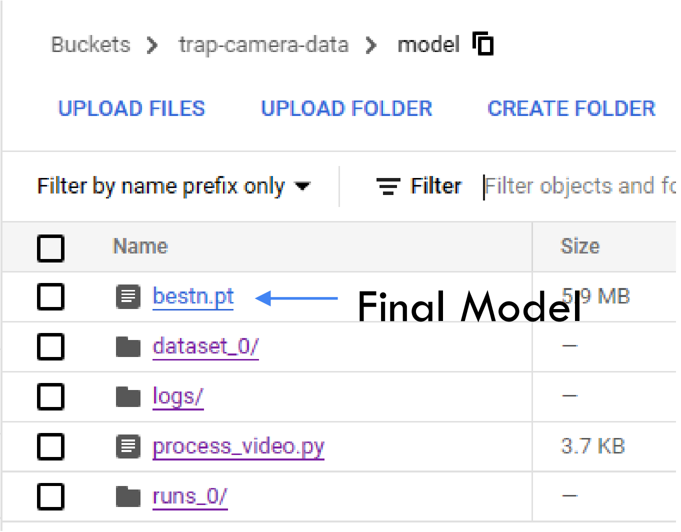


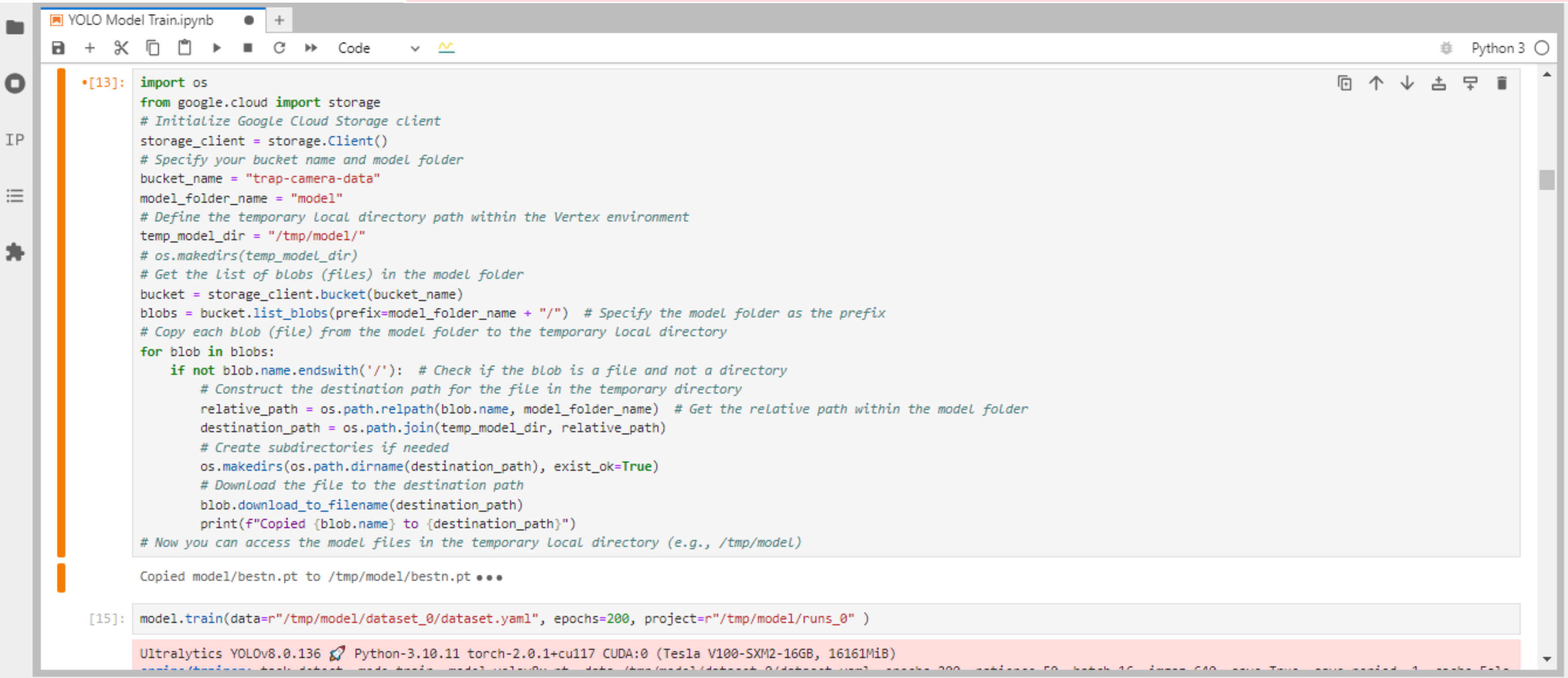
MODEL TRAINING

The model was trained on Vertex AI Notebook, with 800+ images collected and labelled manually across the 5 classes (animals), in the YOLO format.

The dataset was further augmented to include multiple scenarios for light, color, distortions etc.

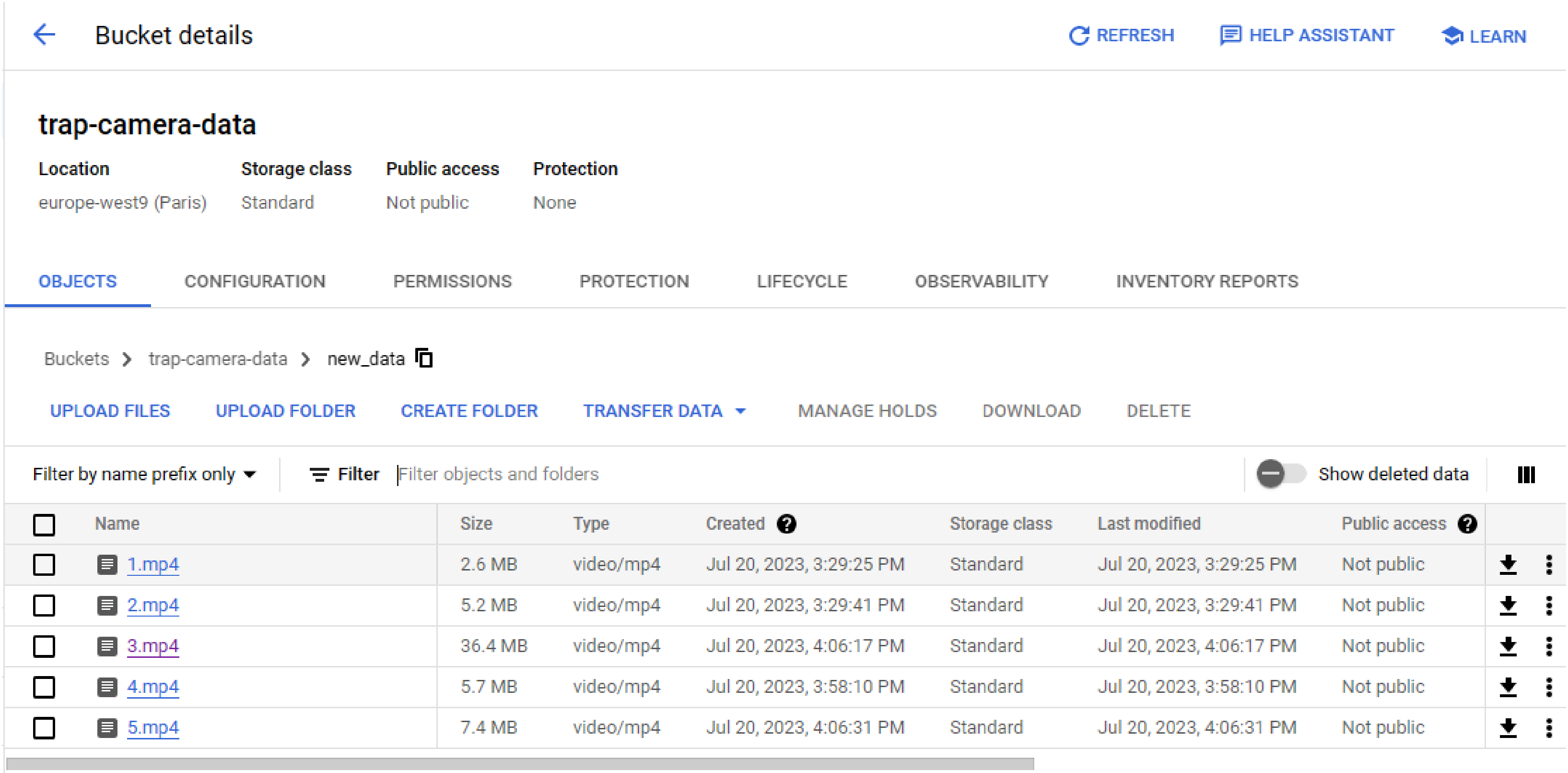






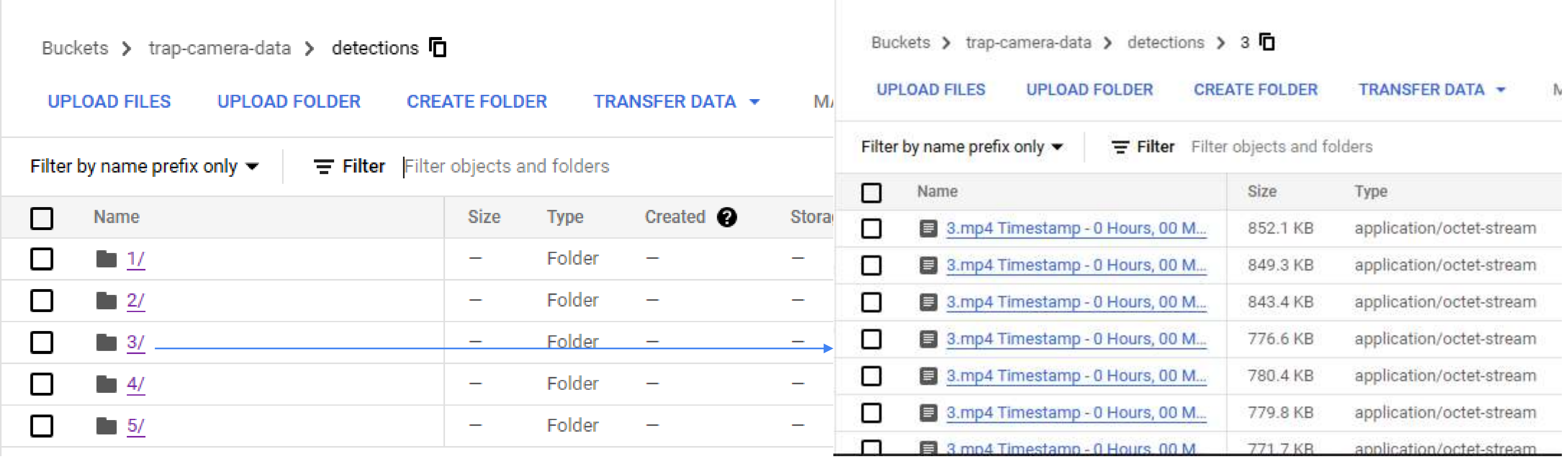
SAMPLE OPERATION – STEP 1

As an example, 5 videos were placed in a specific folder:



SAMPLE OPERATION – STEP 2

After the model was run in the Vertex AI notebook, it created the detection images in separate folder for each video, with the image name as VideoName-Timestamp.jpg

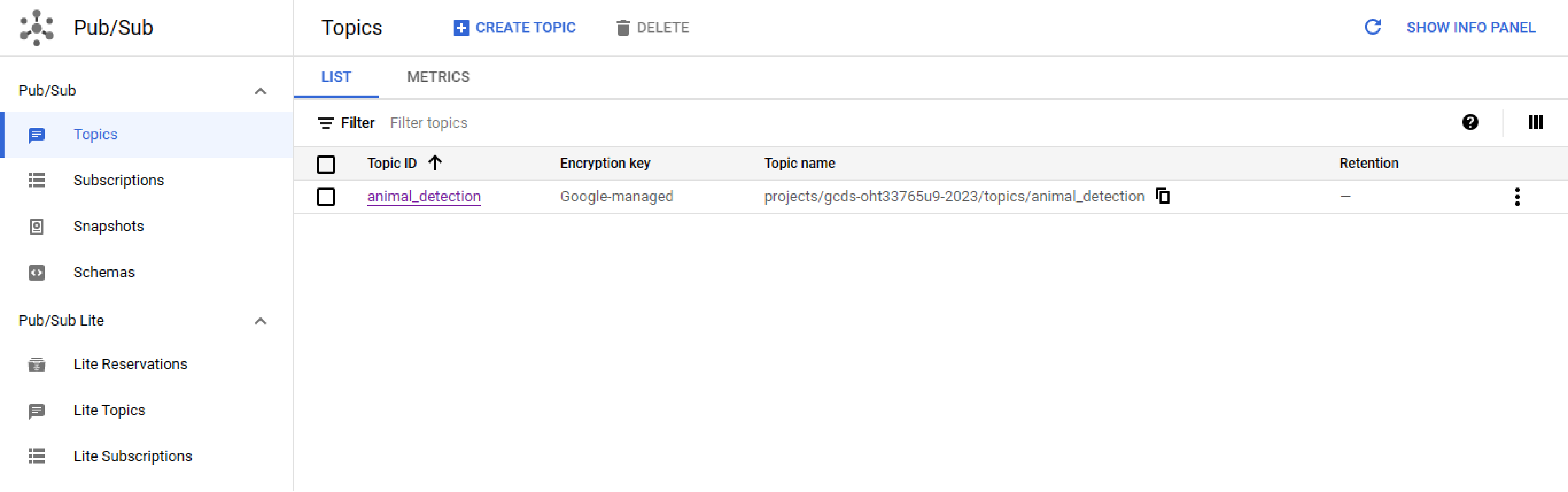


DETECTION CODE USED

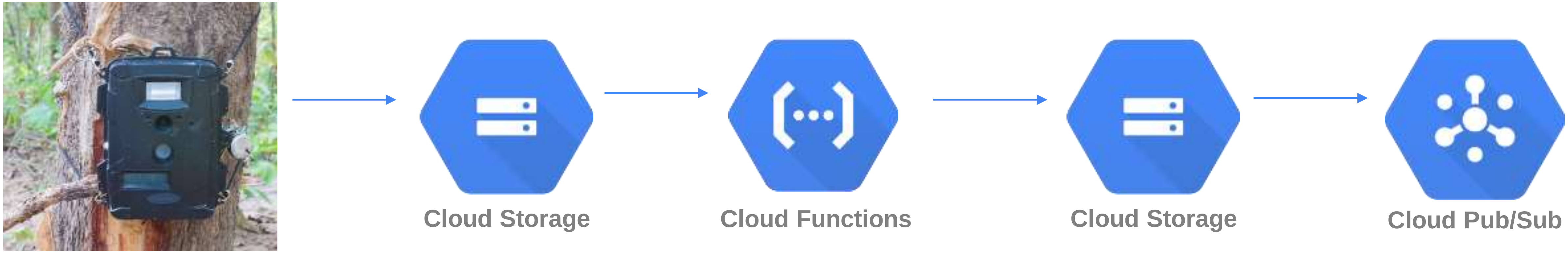
```
for blob in blobs:
    print(blob)
    if blob.name.endswith(".mp4"):
        video_name = os.path.basename(blob.name)
        video_path = f"gs://{new_data_bucket_name}/{blob.name}"
        # Download the video from Google Cloud Storage to the temporary local directory
        temp_video_path = os.path.join(temp_frames_dir, video_name)
        blob.download_to_filename(temp_video_path)
        # Create a folder to store detections for the current video
        video_detections_folder_name = os.path.splitext(video_name)[0]
        video_detections_folder = os.path.join(temp_frames_dir, video_detections_folder_name)
        os.makedirs(video_detections_folder, exist_ok=True)
        # Open the video from the temporary local directory
        cap = cv2.VideoCapture(temp_video_path)
        # Get the video's frames per second (fps) and set the downsampling rate
        fps = cap.get(cv2.CAP_PROP_FPS)
        downsampling_rate = max(int(fps * min_time_difference), 1)  seconds
        frame_counter = 0
        # Calculate the fps_multiplier correctly
        fps_multiplier = 1.0 / fps
        while cap.isOpened():
            ret, curr_frame = cap.read()
            if not ret:
                break
            # Only process the frame if it is within the downsampling rate
            if frame_counter % downsampling_rate == 0:
                result = model(curr_frame)
                for res in result:
                    boxes = res.boxes.cpu().numpy()
                    for i, box in enumerate(boxes):
                        if box.conf[0] > 0.7:
                            res_plotted = res[0].plot(conf=False)
                            current_frame_timestamp = cap.get(cv2.CAP_PROP_POS_MSEC) * fps_multiplier
                            timedelta_obj = str(datetime.timedelta(seconds=round(current_frame_timestamp /
1000,1)))

                            x = timedelta_obj.split(':')
                            hh, mm, ss = x[0], x[1], x[2]
                            filename = f"{video_name} Timestamp - {hh} Hours, {mm} Minutes, {ss} Seconds.jpg"
                            output_filename = os.path.join(video_detections_folder, filename)
                            cv2.imwrite(output_filename, res_plotted)
                            print("Saved frame:", output_filename)
            # Increment the frame counter
            frame_counter += 1
        cap.release()
```

RESULTS (DETECTION)

RESULT (SENT TO PUBSUB TOPIC)



SUGGESTED CHANGE 1 – AUTOMATING DETECTION RUN THROUGH SERVERLESS CLOUD FUNCTIONS



Videos from Tap Camera come in batches daily.

e.g. 24 1-hour videos for yesterday

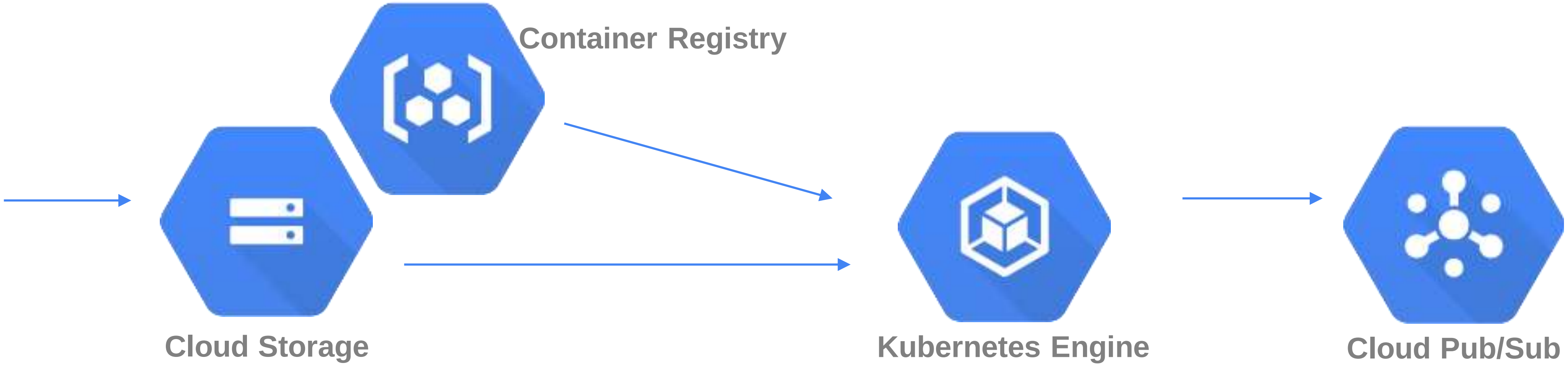
The videos land in a particular folder inside a bucket in Cloud Storage

As soon as a new video lands in the bucket, using object finalization trigger, Cloud Functions can kick in and run a python code to pass the videos through an object detection model.

The code automatically stores the detections in the bucket. The detections are images with bounding boxes around the animal found. Also, the detections are named as VideoName-Timestamp.jpg

As soon as new detections are stored in the bucket, Cloud PubSub will trigger a notification to subscribed web services, which can ingest the topic to send email/SMS to concerned people

SUGGESTED CHANGE 2 – AUTOMATING DETECTION RUN THROUGH KUBERNETES CONTAINERIZATION



Videos from Tap Camera come in batches daily.

e.g. 24 1-hour videos for yesterday

The videos land in a particular folder inside a bucket in Cloud Storage

The Kubernetes Engine can host the model as an endpoint on a webservice, using which the videos can processed upon.
The model along with all the dependencies will be present in the container registry as a docker container.
Then the endpoint can send the detections directly to pubsub.

As soon as new detections are stored in the bucket, Cloud PubSub will trigger a notification to subscribed web services, which can ingest the topic to send email/SMS to concerned people

DRONE SEARCH & RESCUE

UseCase Type:

Model Deployment : Edge Device (Drone Camera)
Detection : Edge Device
Storage : Cloud
Action Taken : On Cloud (Alerting Concerned Services/ People)

OVERVIEW

Drones can be used to detect people and animals requiring urgent help from immediate physical harm in remote areas or hard-to-monitor places.

As soon as a detection is made, the concerned authorities can be notified and alerted to take immediate actions.

Moreover, an accurate location can be sent along with the detection.

CURRENT WORKING ARCHITECTURE

Drones are deployed with cameras in concerned areas.

The ML model is stored on the camera itself. Services like Cloud IoT Core can be used.

Cloud IoT Core

Detections are made on the device itself in real-time.
The detections contain the image of the person, along with the label of the danger present.
Additionally, Google Maps API is used to enrich the data with location.
Optionally, Android devices can be used for easier deployment capabilities.

Cloud Pub/Sub

In case of detections, immediate notification is sent to PubSub topic.
A webservice subscribed to that topic can send an email/ SOS/ SMS alert to the concerned authorities

Cloud Storage

Data is stored on Cloud Storage for backup and further processing.

EXAMPLE SCENARIO

In our beach park, we would like to keep track of majorly two types of accidents:

1. Drowning People,
2. Capsized Boats

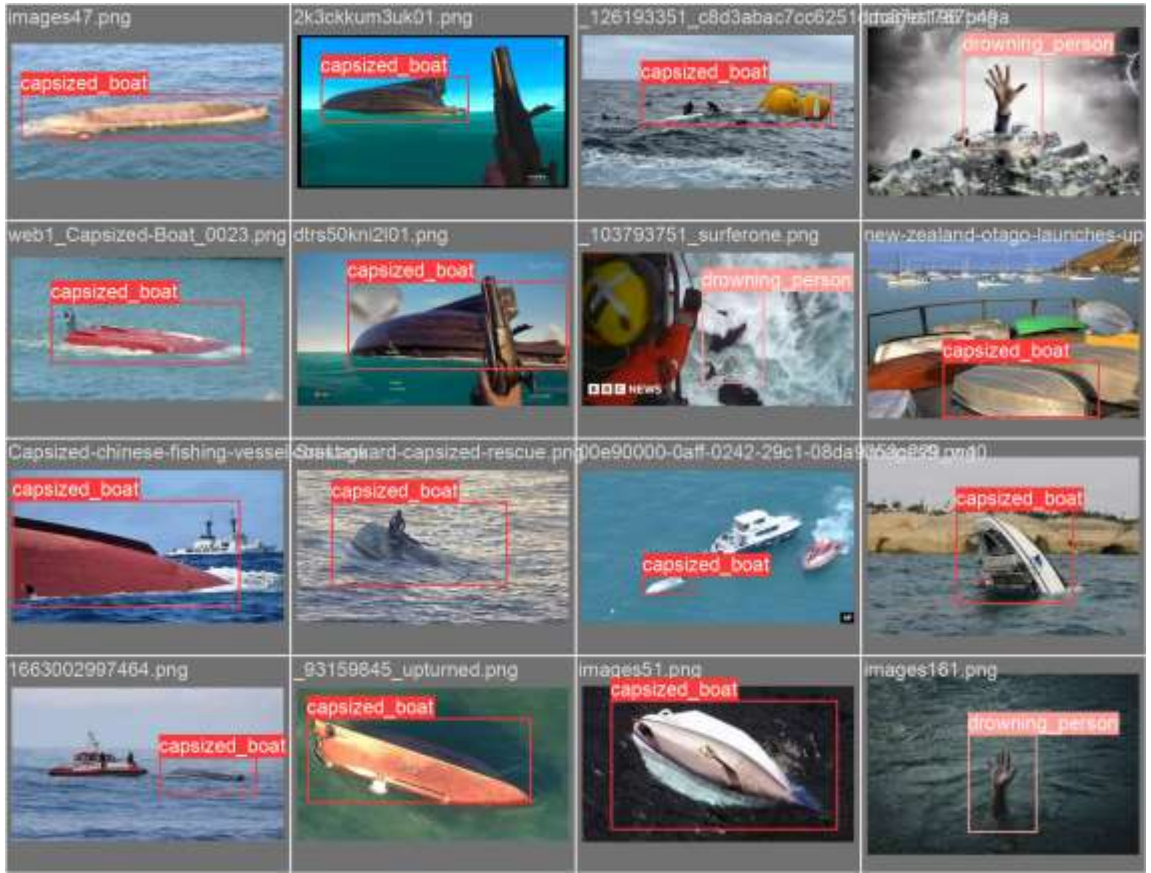
Since the life guards are not able to monitor far away waters, we would like to take the help of a drone continuously monitoring the waters.

The drone should alert us immediately in case of the above two dangers, following which we can send a rapid response team.

The detection should contain an image of the human(s), with the danger, and also the location of the event.

MODEL USED

YOLOv8 Object Detection Model (https://docs.ultralytics.com/)



MODEL TRAINING

The model was trained on Vertex AI Notebook & Google Colab, with 800+ images collected and labelled manually across the 2 classes in the YOLO format.

The dataset was further augmented to include multiple scenarios for light, color, distortions etc.

```python
from ultralytics import YOLO
import locale
locale.getpreferredencoding = lambda: "UTF-8"
from google.colab.patches import cv2_imshow
import tensorflow as tf
#Avoid OOM errors by setting GPU Memory Consumption Growth
gpus = tf.config.experimental.list_physical_devices('GPU')
for gpu in gpus:
    tf.config.experimental.set_memory_growth(gpu, True)
tf.config.list_physical_devices('GPU')
from google.colab import drive
drive.mount('/content/drive')
```

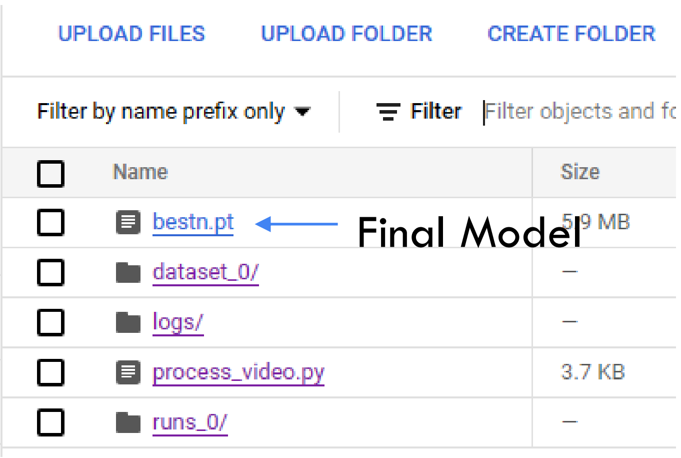
```
Mounted at /content/drive
```

```python
model = YOLO("yolov8m.pt")
```

```
Downloading https://github.com/ultralytics/assets/releases/download/v0.0.0/yolov8m.pt to yolov8m.pt...
100%|██████████| 49.7M/49.7M [00:00<00:00, 56.7MB/s]
```

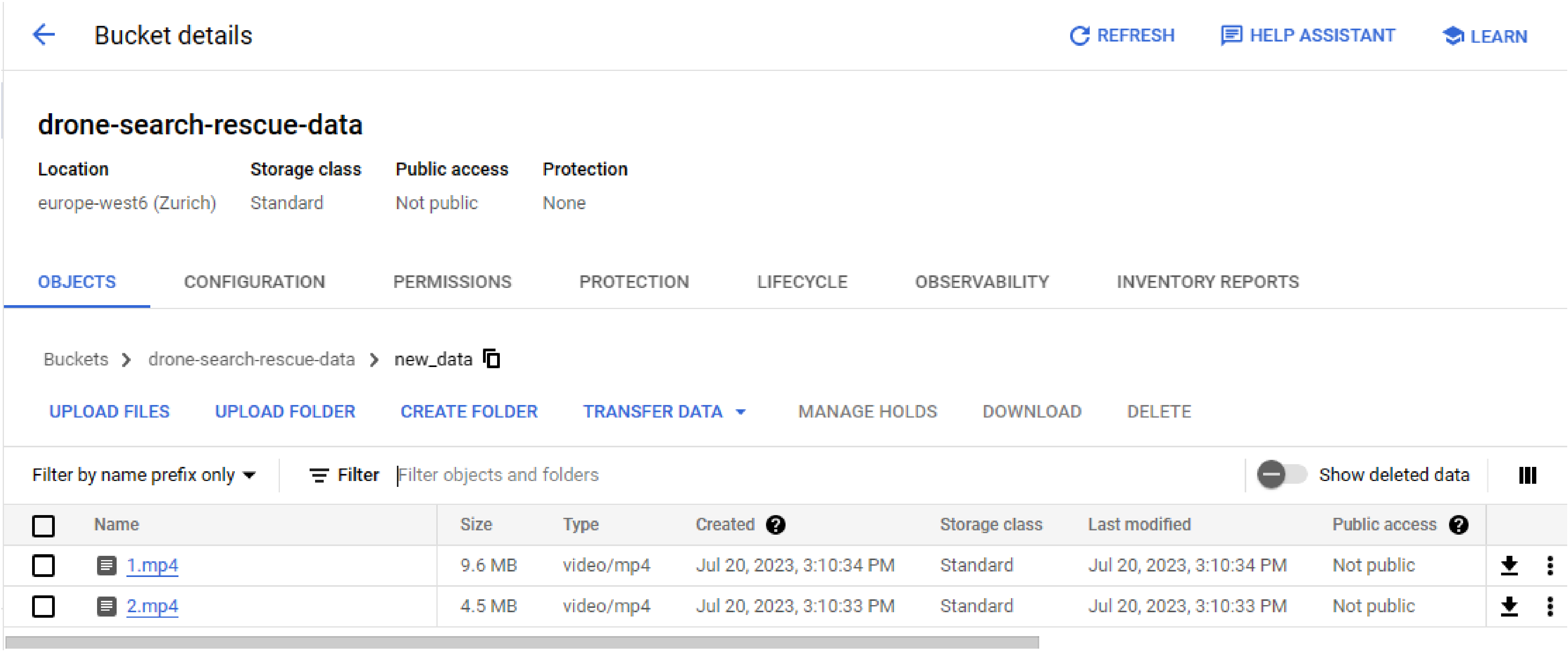
```python
model.train(data=r"/content/drive/MyDrive/Machine Learning/Drone Search and Rescue YOLO/dataset/dataset.yaml", epochs=100, project=
```

```
      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     99/100     7.25G     0.8303     0.6203      1.239          3        640: 100%|██████████| 11/11 [00:05<00:00,  1.99it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 2/2 [00:00<00:00,  2.59it/s]
                   all         41         41      0.884      0.685      0.816      0.391

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
    100/100     7.22G     0.8804     0.6307      1.301          3        640: 100%|██████████| 11/11 [00:05<00:00,  2.00it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 2/2 [00:02<00:00,  1.19s/it]
                   all         41         41      0.877      0.685       0.82      0.388

100 epochs completed in 0.292 hours.
Optimizer stripped from /content/drive/MyDrive/Machine Learning/Drone Search and Rescue YOLO/dataset/train3/weights/last.pt, 52.0MB
Optimizer stripped from /content/drive/MyDrive/Machine Learning/Drone Search and Rescue YOLO/dataset/train3/weights/best.pt, 52.0MB

Validating /content/drive/MyDrive/Machine Learning/Drone Search and Rescue YOLO/dataset/train3/weights/best.pt...
Ultralytics YOLOv8.0.132 🚀 Python-3.10.12 torch-2.0.1+cu118 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 218 layers, 25840918 parameters, 0 gradients
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 2/2 [00:01<00:00,  1.87it/s]
                   all         41         41      0.948      0.697      0.802      0.431
          capsized_boat         41         35      0.896      0.737      0.909      0.448
        drowning_person         41          6          1      0.656      0.696      0.414
Speed: 5.9ms preprocess, 8.9ms inference, 0.0ms loss, 1.5ms postprocess per image
Results saved to /content/drive/MyDrive/Machine Learning/Drone Search and Rescue YOLO/dataset/train3
```



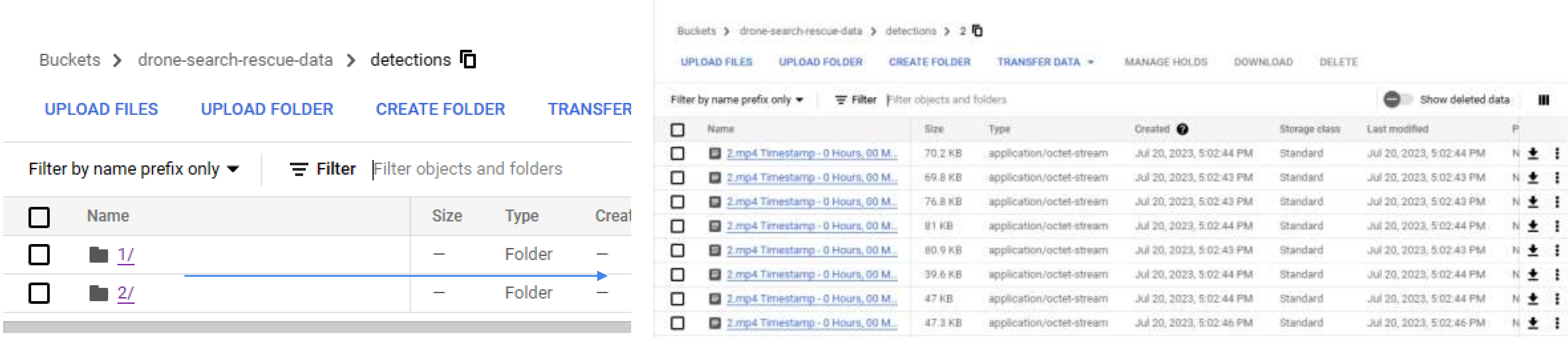
SAMPLE OPERATION – STEP 1

As an example, 2 videos were placed in a specific folder:



SAMPLE OPERATION – STEP 2

After the model was run in the Vertex AI notebook, it created the detection images in separate folder for each video, with the image name as VideoName-Timestamp.jpg



DETECTION CODE USED

```
for blob in blobs:
    print(blob)
    if blob.name.endswith(".mp4"):
        video_name = os.path.basename(blob.name)
        video_path = f"gs://{new_data_bucket_name}/{blob.name}"
        # Download the video from Google Cloud Storage to the temporary local directory
        temp_video_path = os.path.join(temp_frames_dir, video_name)
        blob.download_to_filename(temp_video_path)
        # Create a folder to store detections for the current video
        video_detections_folder_name = os.path.splitext(video_name)[0]
        video_detections_folder = os.path.join(temp_frames_dir, video_detections_folder_name)
        os.makedirs(video_detections_folder, exist_ok=True)
        # Open the video from the temporary local directory
        cap = cv2.VideoCapture(temp_video_path)
        # Get the video's frames per second (fps) and set the downsampling rate
        fps = cap.get(cv2.CAP_PROP_FPS)
        downsampling_rate = max(int(fps * min_time_difference), 1)  seconds
        frame_counter = 0
        # Calculate the fps_multiplier correctly
        fps_multiplier = 1.0 / fps
        while cap.isOpened():
            ret, curr_frame = cap.read()
            if not ret:
                break
            # Only process the frame if it is within the downsampling rate
            if frame_counter % downsampling_rate == 0:
                result = model(curr_frame)
                for res in result:
                    boxes = res.boxes.cpu().numpy()
                    for i, box in enumerate(boxes):
                        if box.conf[0] > 0.7:
                            res_plotted = res[0].plot(conf=False)
                            current_frame_timestamp = cap.get(cv2.CAP_PROP_POS_MSEC) * fps_multiplier
                            timedelta_obj = str(datetime.timedelta(seconds=round(current_frame_timestamp /
1000,1)))
                            x = timedelta_obj.split(':')
                            hh, mm, ss = x[0], x[1], x[2]
                            filename = f"{video_name} Timestamp - {hh} Hours, {mm} Minutes, {ss} Seconds.jpg"
                            output_filename = os.path.join(video_detections_folder, filename)
                            cv2.imwrite(output_filename, res_plotted)
                            print("Saved frame:", output_filename)
            # Increment the frame counter
            frame_counter += 1
        cap.release()
```

RESULTS (DETECTION)

SUGGESTED CHANGES

- Training the model with better data i.e. top view data taken from a drone, with classes simulated in real time.
- Training for multiple classes i.e. other dangers e.g. sharks
- Training for multiple angles i.e. looking down from drone, looking straight etc.
- Deployment of Android Devices on the Drone using Raspberry Pi, instead of expensive drone cameras.
- Drones can also carry first aid kit and small supplies, that can be dropped to people who need them in the interim the rescue people come.

DRIVER DROWSINESS DETECTION

UseCase Type:

Model Deployment : Edge Device (Vehicle Dash Camera)

Detection : Edge Device

Storage : Edge Device, Cloud

Action Taken : On Device (Raising Car Alarm, Stopping Car)

OVERVIEW

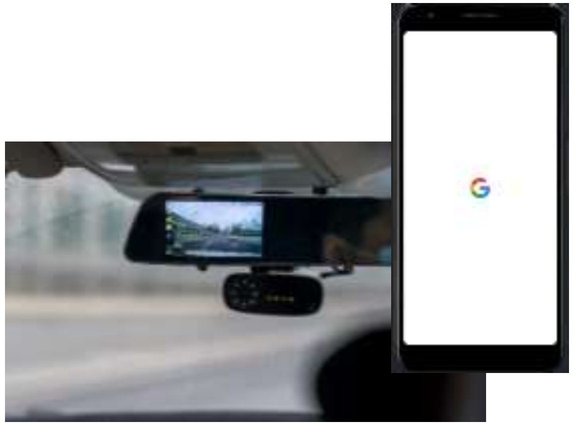
Dashboard cameras (Dashcams) are getting increasingly popular. They can be both inward facing (looking inside the car), as well as outward facing (looking outside the car, typically in front of the car).

They are used to monitor and store the actions of passengers, drivers, pedestrians, traffic etc by the fleet owners of cabs, trucks etc.

A major concern for all the parties involved is the drowsiness of the driver, which can cause fatal accidents.

Though the drowsiness cannot be prevented, measures can be taken such that as soon as a driver feels drowsy, he can be awoken by an instant alarm in the vehicle, or maybe even the vehicle can be brought to a halt safely.

CURRENT WORKING ARCHITECTURE



Dashcams are deployed inside the car, aimed at driver's face.
The ML model is stored on the camera itself.
Services like Cloud IoT Core can be used.
Optionally, Android devices can be used for easier deployment capabilities.

Cloud IoT Core

Detections are made on the device itself in real-time.
The detections contain the image of the person, along with the label of awake or drowsy.
Additionally, Google Maps API is used to enrich the data with location.

Cloud Storage

Data is stored on Cloud Storage for backup and further processing.

Cloud Pub/Sub

In case of detections, immediate notification is sent to PubSub topic.
A webservice subscribed to that topic can send an email/ SOS/ SMS alert to the concerned authorities notifying the location of the driver and vehicle.

Cloud SDK

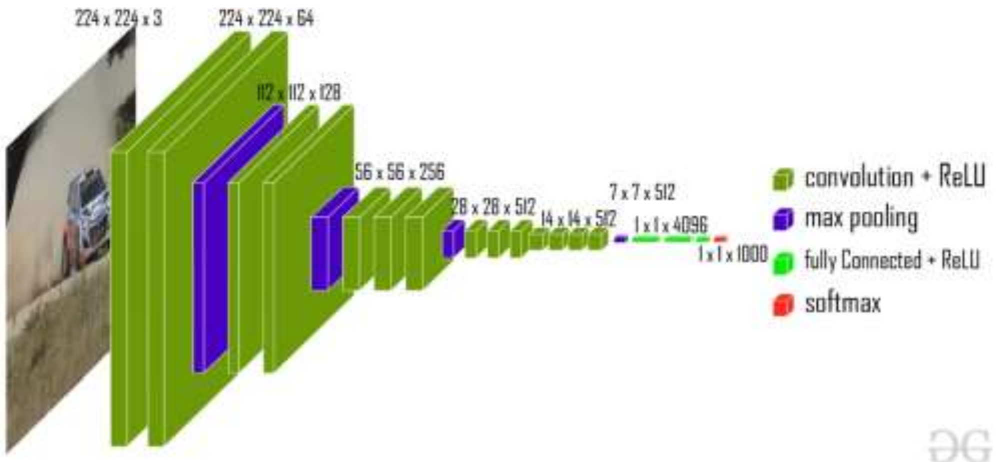
The camera/ android device used in dashcam integrated with the software system of the vehicle.
When a drowsiness threshold is reached, it can start an alarm or maybe even stop the car.

EXAMPLE SCENARIO

- We run a fleet of shipping trucks.
- We need to constantly monitor our drivers for signs of drowsiness.
- If a driver is drowsy for more than 5 seconds, an alarm should ring in the car.
- The alarm volume should keep on rising if the driver is still drowsy.
- If the said driver is drowsy for more than 10 seconds, stop the vehicle.
- Also send us a notification with a location, in case a vehicle is stopped

MODEL USED

VGG16 Classification Model (https://keras.io/api/applications/vgg/)



MODEL TRAINING

The model was trained on local machine, with 4000+ images collected and labelled across the 2 classes (drowsy & awake) in a Kaggle repository. (https://www.kaggle.com/datasets/prasadvpatil/mrl-dataset)

The repository contains only the data for eyes, which is a good starting point for training a drowsiness model, as drowsiness corresponds to longer periods of closed eyes.

The dataset was further augmented to include multiple scenarios for light, color, distortions etc.

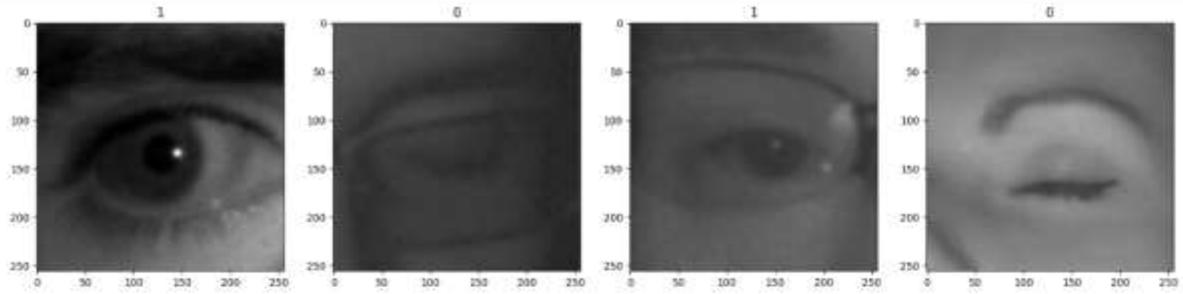
```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatten, Dropout
```

```python
model = Sequential()
```

```python
#
model.add(Conv2D(16, (3,3), 1, activation='relu', input_shape=(256,256,3)))
model.add(MaxPooling2D())

model.add(Conv2D(32, (3,3), 1, activation='relu'))
model.add(MaxPooling2D())

model.add(Conv2D(16, (3,3), 1, activation='relu'))
model.add(MaxPooling2D())

model.add(Flatten())

model.add(Dense(256, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

```python
model.compile('adam', loss=tf.losses.BinaryCrossentropy(), metrics=['accuracy'])
```

```python
fig, ax = plt.subplots(ncols=4, figsize=(20,20))
for idx, img in enumerate(batch[0][:4]):
    ax[idx].imshow(img.astype(int))
    ax[idx].title.set_text(batch[1][idx])
```




```python
new_model = load_model(os.path.join('models','cnnimageclassifier.h5'))
```

```python
yhatnew = new_model.predict(np.expand_dims(resize/255, 0))
```

```
1/1 [==============================] - 0s 130ms/step
```

```python
if yhatnew > 0.5:
    print(f'Predicted class is Awake')
else:
    print(f'Predicted class is Drowsy')
```

SAMPLE OPERATION

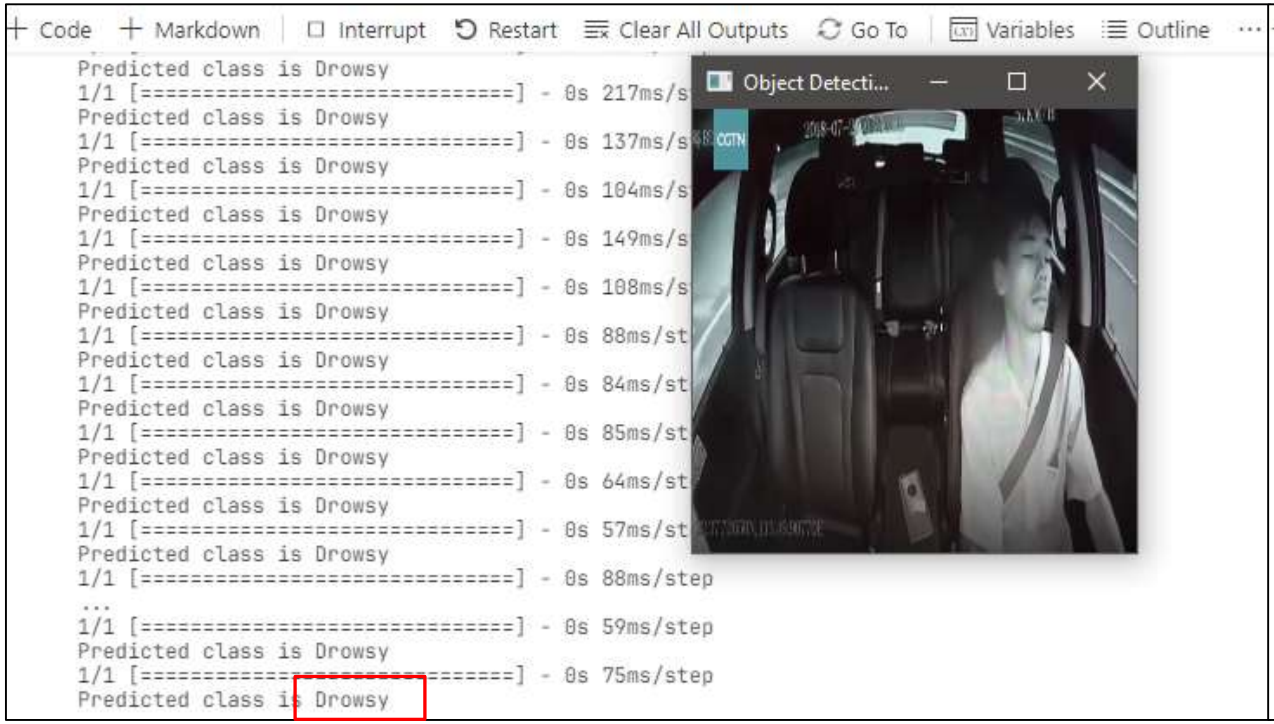
As an example, 1 video was placed in a specific folder.

The video shows a man driving a truck while being drowsy.

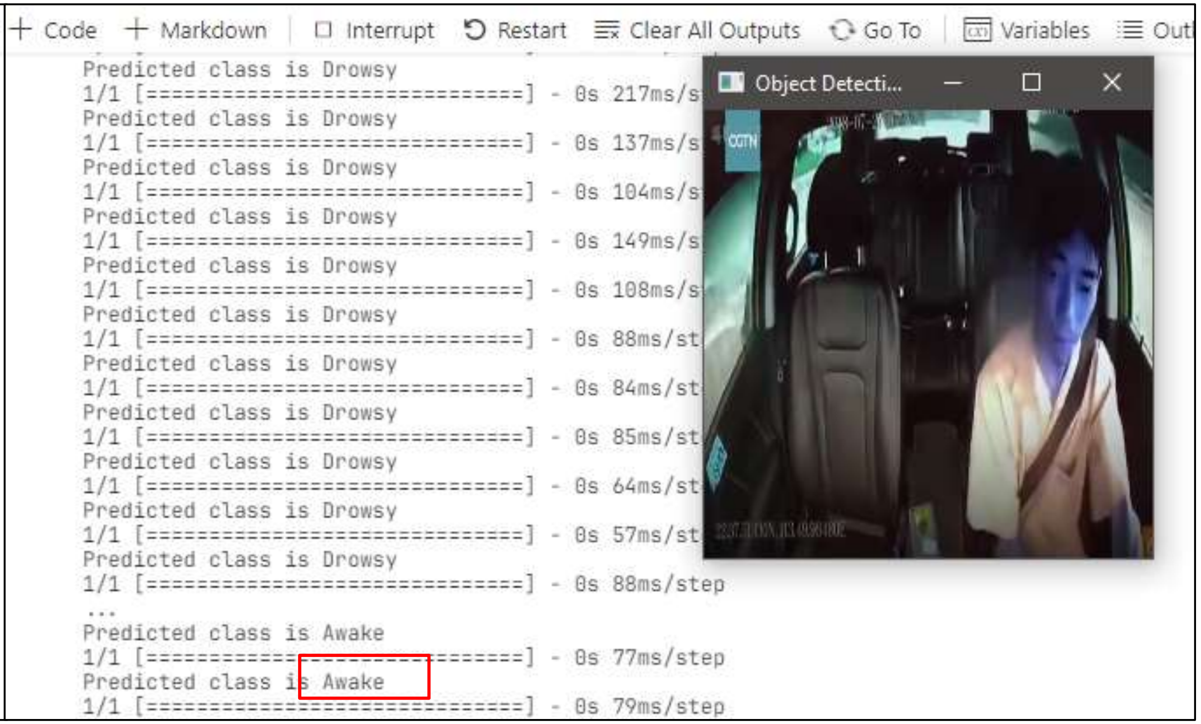
The video was passed through the model as shown in the code.

```
from tensorflow.keras.models import load_model
new_model = 
load_model(r'C:\Users\HP\Desktop\drowsinessdetection.h5')
def detect_objects_in_video(model, video_path):
    # Open the video file
    cap = cv2.VideoCapture(video_path)
    while True:
        ret, frame = cap.read()
        if not ret:
            break
        # Preprocess the frame (adjust according to your model's 
input requirements)
        frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        frame = cv2.resize(frame, (256, 256))  # Adjust input 
dimensions
        frame = frame / 255.0  # Normalize pixel values
        # Make predictions using the model
        prediction = model.predict(np.expand_dims(frame, axis=0))
        if prediction > 0.5:
            print(f'Predicted class is Awake')
        else:
            print(f'Predicted class is Drowsy')

        # Process the predictions and draw bounding boxes
        # (You need to customize this part based on your model's 
output and the objects you want to detect)

        # Display the frame with detections
        cv2.imshow('Object Detection', frame)
        # Exit on pressing 'q'
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
    # Release video capture and close the OpenCV window
    cap.release()
    cv2.destroyAllWindows()
# Replace 'video_path' with the path to your video file
video_path = r'C:\Users\HP\Desktop\1.mp4'
detect_objects_in_video(new_model, video_path)
```

RESULTS (DETECTION)

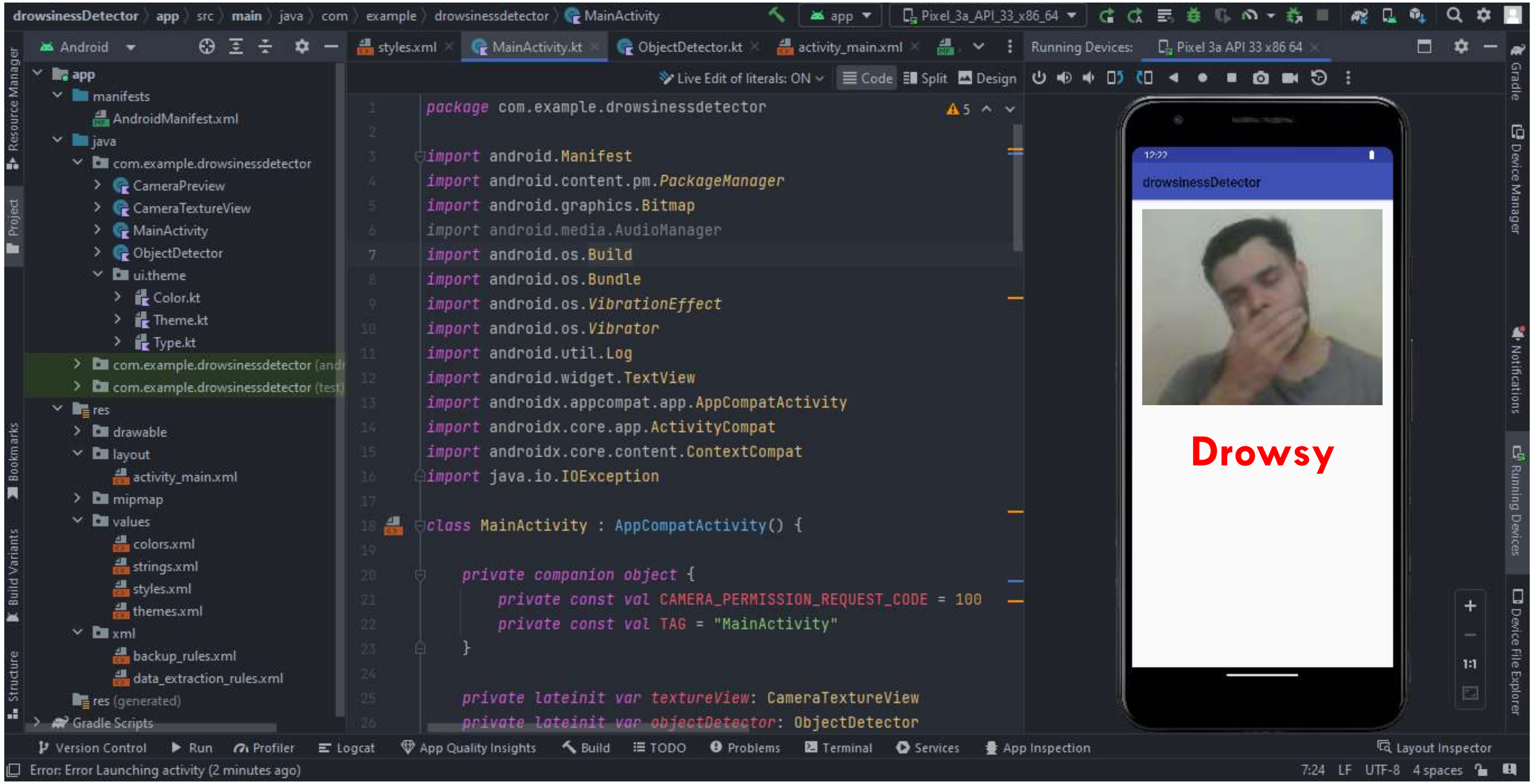


Drowsy



Awake

ANDROID APP AS A SUBSTITUTE FOR DASHCAM



The Android App takes in the video feed from the front camera, and applies to it a TFLite version of the model stored in the application package.

The model is run continuously on each of the frames, and classifies it as drowsy or awake.

If the drowsy class is maintained for more than 5 seconds, the phone vibrates.

SUGGESTED CHANGES

- Refined Android App
- An ML model trained on an exhaustive set of data for drowsiness, which includes yawning, hand movement, etc.
- Integration of the Android App with vehicle's electrical system.
- Deployment of the package on interior facing dashcams focused on the driver's face.