AI-Powered Accident Detection System Using Java and Google Cloud Vertex AI

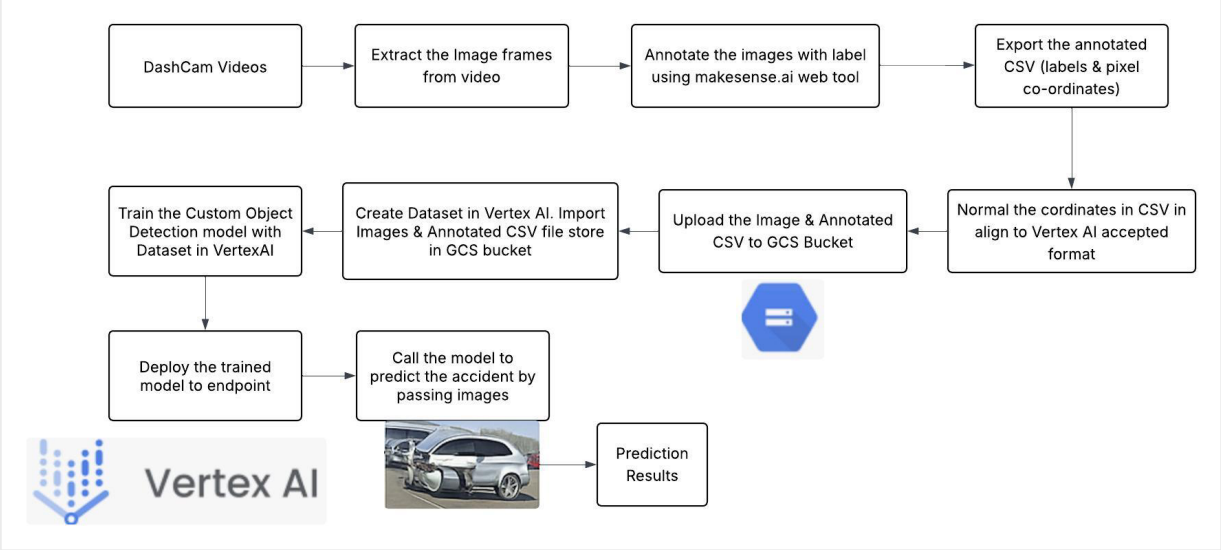
1. Introduction

Road accidents claim over a million lives every year worldwide, with countless more left injured. Timely accident detection can drastically reduce response times for emergency services, potentially saving lives and minimizing damage.

In this article, I'll walk you through how I built an AI-powered accident detection system — entirely using **Java for preprocessing, cloud integration, and prediction calls** — combined with **Google Cloud Vertex AI for training and deploying a custom object detection model.**

This work is not just a technical project; it has real-world societal impact, aligning with public safety priorities and contributing toward innovations that support **national interests in road safety and AI development.**

Architecture Diagram



2. Technology Stack

Here's the stack I used and why:

Component	Purpose
Java	Core programming language for data processing, CSV conversion, cloud uploads, and API integration.
OpenCV (Java Bindings)	Extract frames from dash cam videos.
makesense.ai	Web-based tool for annotating images with bounding boxes.
Google Cloud Storage (GCS)	Stores images and annotation files for Vertex AI.
Vertex AI	Trains, deploys, and serves the accident detection model.
PredictionServiceClient (Java)	Calls the deployed model for predictions directly from a Java application.

Github Link:
https://github.com/lalamanil/AccidentDetectionModelJavaVertexAI

3. Dataset Preparation

3.1 Collecting Dash cam Accident Videos

The first step was collecting dash cam accident videos from publicly available datasets and open-license online sources. These videos were downloaded locally for preprocessing.

3.2 Extracting Frames from Videos

The model works on still images, so I needed to convert video footage into image frames.
Using **OpenCV's Java API**, I wrote a utility class ExtractFramesAsImagesFromVideo.java to:

- Load the input video file.
- Iterate frame-by-frame.
- Save each frame as a .jpg file.

Include below dependency in pom.xml

```xml
<dependency>
    <groupId>org.bytedeco</groupId>
    <artifactId>opencv-platform</artifactId>
    <version>4.11.0-1.5.12</version>
</dependency>
```

Below is the java code to extract frames(Images) for a given Video

```java
package com.video.object.tracking.utility;
/**
 *  @author lalamanil */
import org.bytedeco.opencv.opencv_core.*;
import org.bytedeco.opencv.opencv_videoio.*;
import static org.bytedeco.opencv.global.opencv_imgcodecs.imwrite;

//Utility  to extract image frames from video to annotate dataset for Custom Object tracking model
public class ExtractFramesAsImagesFromVideo {

    public static void getFramesFromVideo(String videoPath, String outputpath) {
        VideoCapture videoCapture = new VideoCapture(videoPath);
        Mat frame = new Mat();
        int frameNumber = 0;
        int imageCount = 0;
        while (videoCapture.read(frame)) {
            // logic to pull every 50th frame. if we put 100, It pulls every 100th frame
            if (frameNumber % 50 == 0) {
                imwrite(outputpath + "frame_" + imageCount + ".jpg", frame);
                System.out.println(outputpath + "frame_" + imageCount + ".jpg");
                imageCount++;
            }
            frameNumber++;
        }
        videoCapture.release();
        videoCapture.close();
        System.out.println("Done!!");
    }

    public static void main(String[] args) {

        // Path to mp4 video
        String inputFilePath = "/Users/lalamanil/voiceanalyzer/CarAccidentTest.mp4";
        // Folder path where Image frames need to be stored
        String outputFilePath = "/Users/lalamanil/voiceanalyzer/CarAccidentTestFrames/";
        getFramesFromVideo(inputFilePath, outputFilePath);
    }

}
```

NOTE: **This process transformed each accident video into hundreds of labeled image frames.**

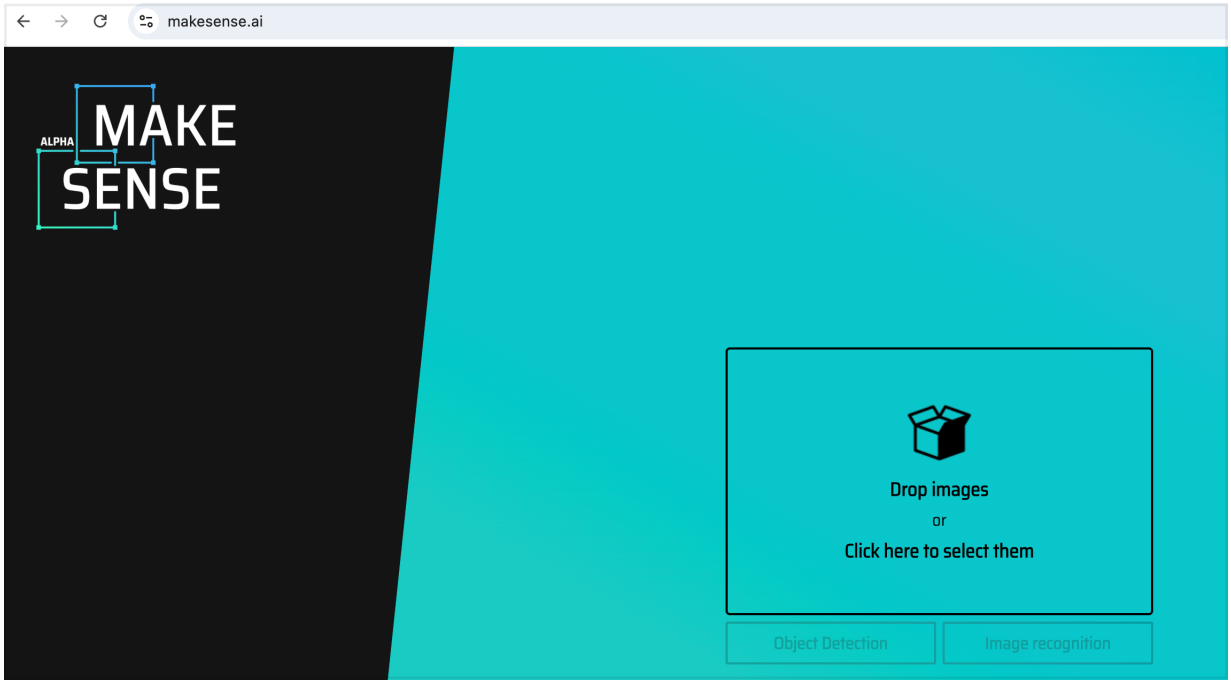
3.3 Annotating Images

With the extracted frames, I used **makesense.ai** to annotate:

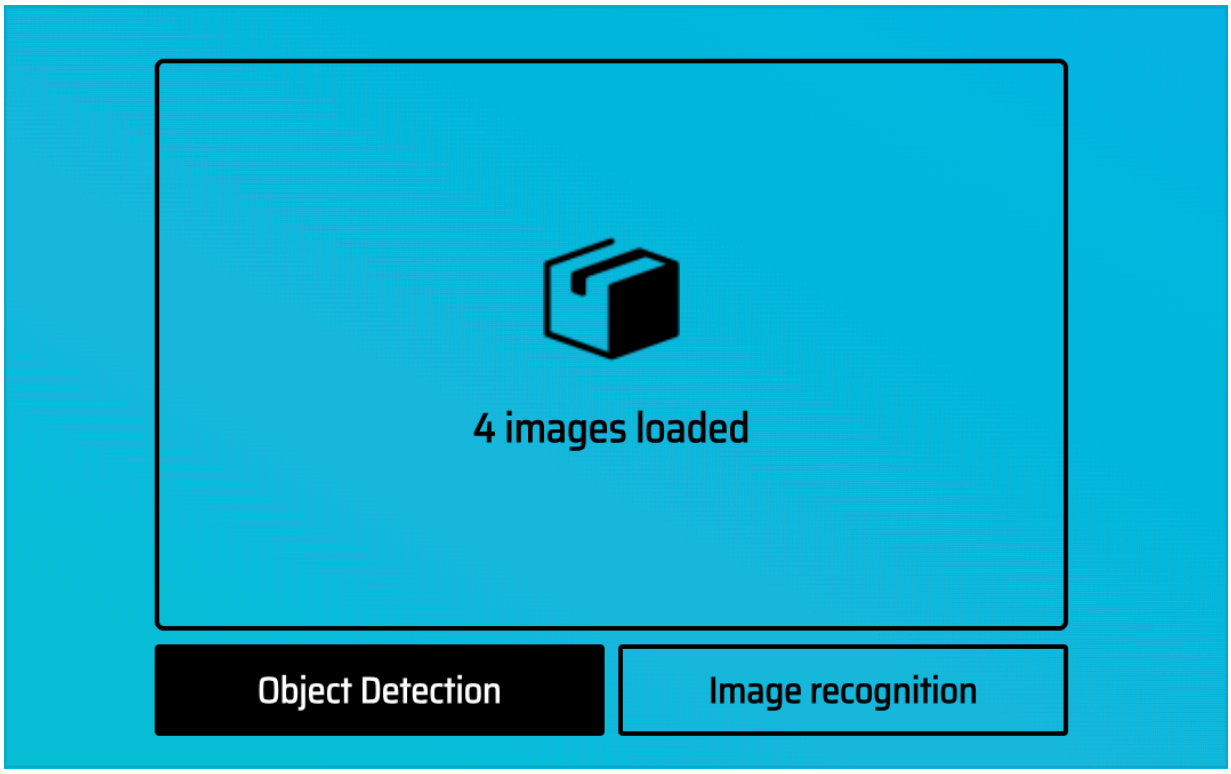
- **Label 1:** accident (frames showing accidents).
- **Label 2:** normal (frames without accidents).

Annotations were created by **drawing bounding boxes** around accident scenes and exporting the result as a CSV file.

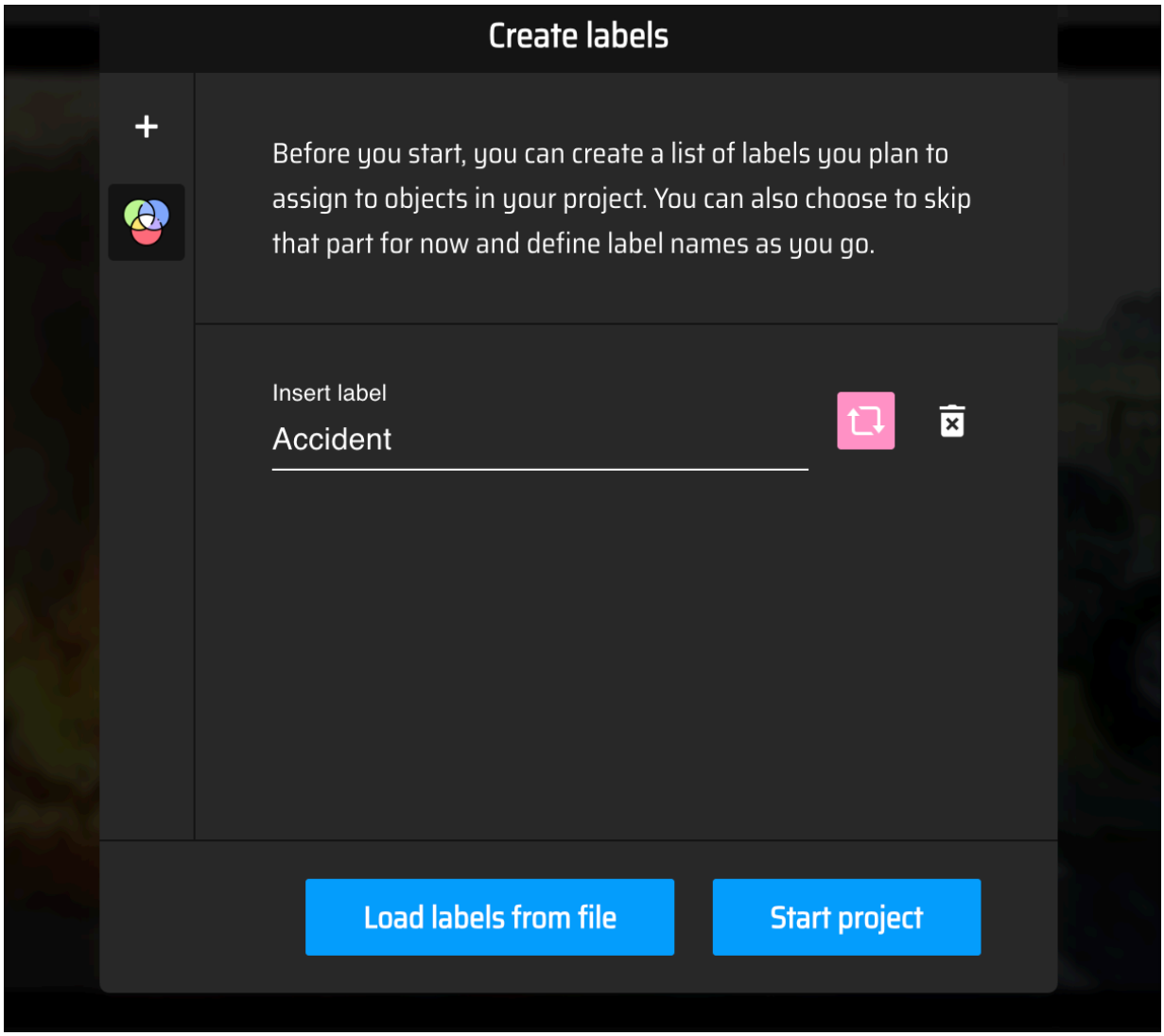
Below are the some sample screen shots, How to draw bounding boxes around images using web tool **https://www.makesense.ai/**



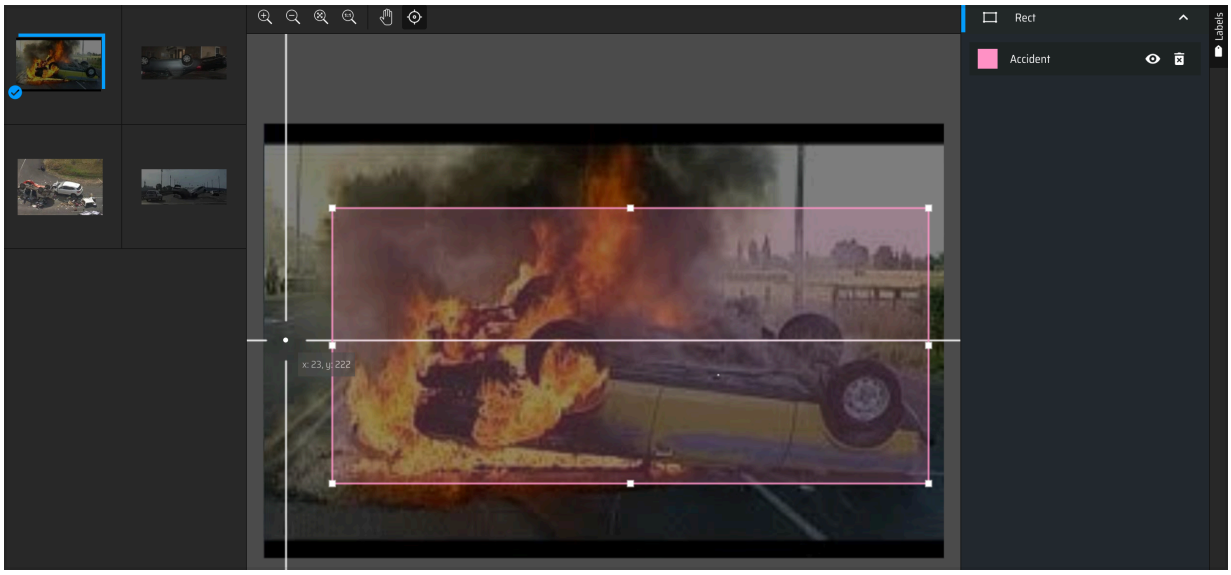
Load Image frames from local machine. Here I have loaded 4 images for demo and selected Object Detection.



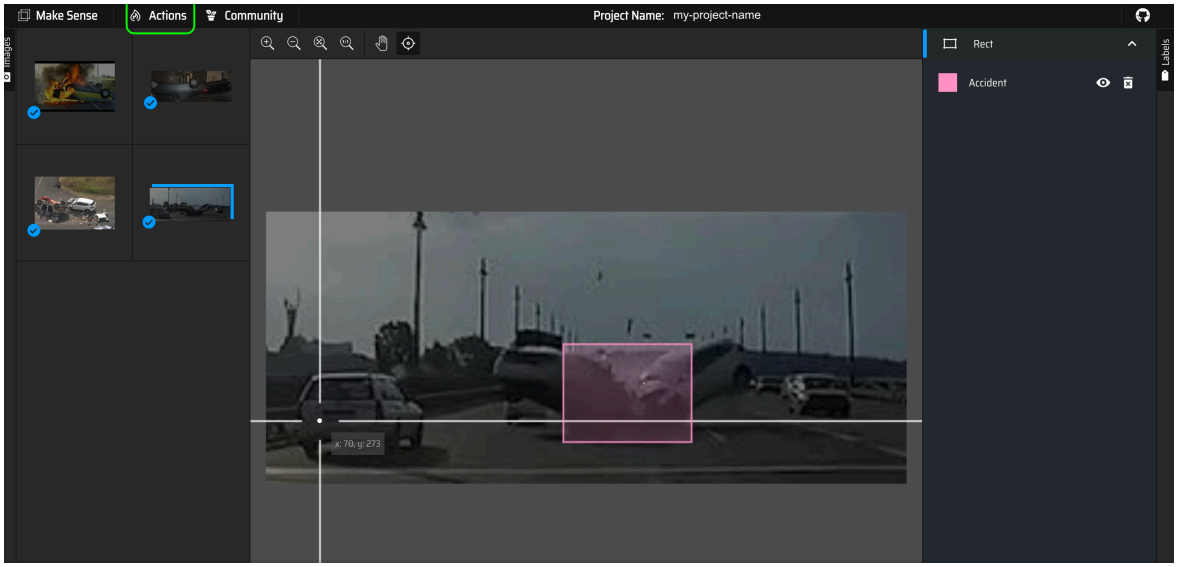
Create a Labels to assign to objects and click on Start project.



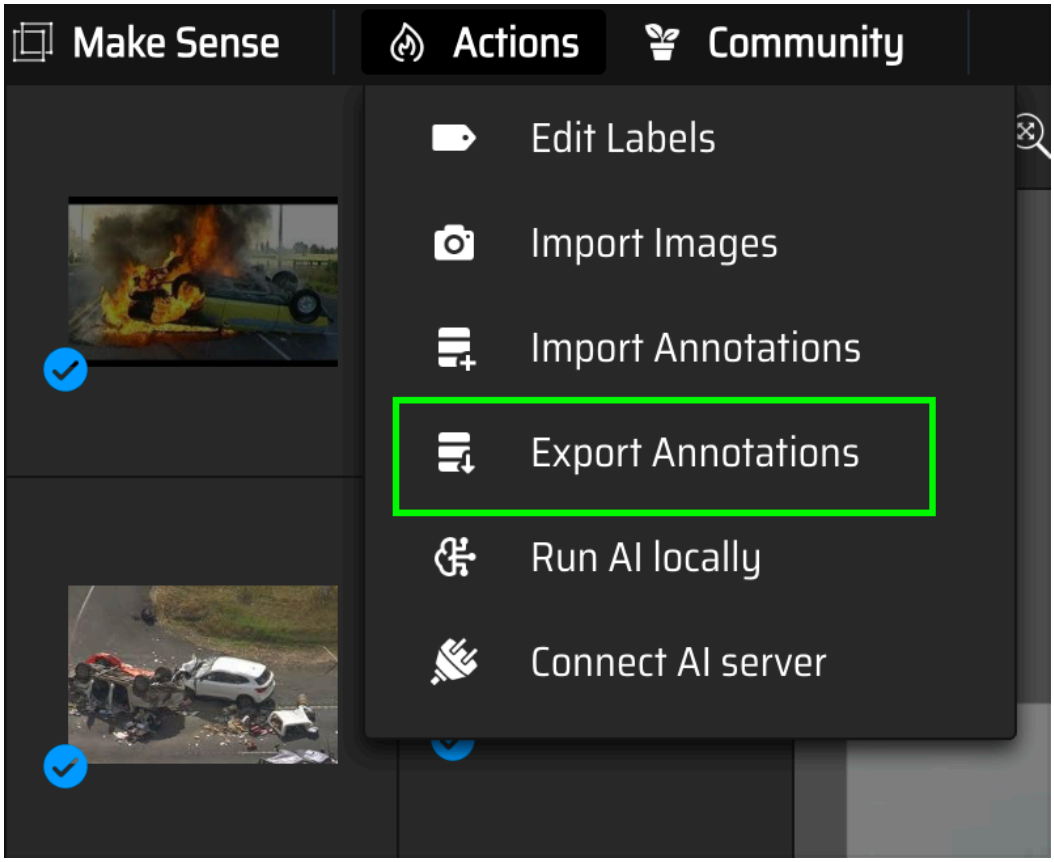
Now draw bounding boxes around accident scenes



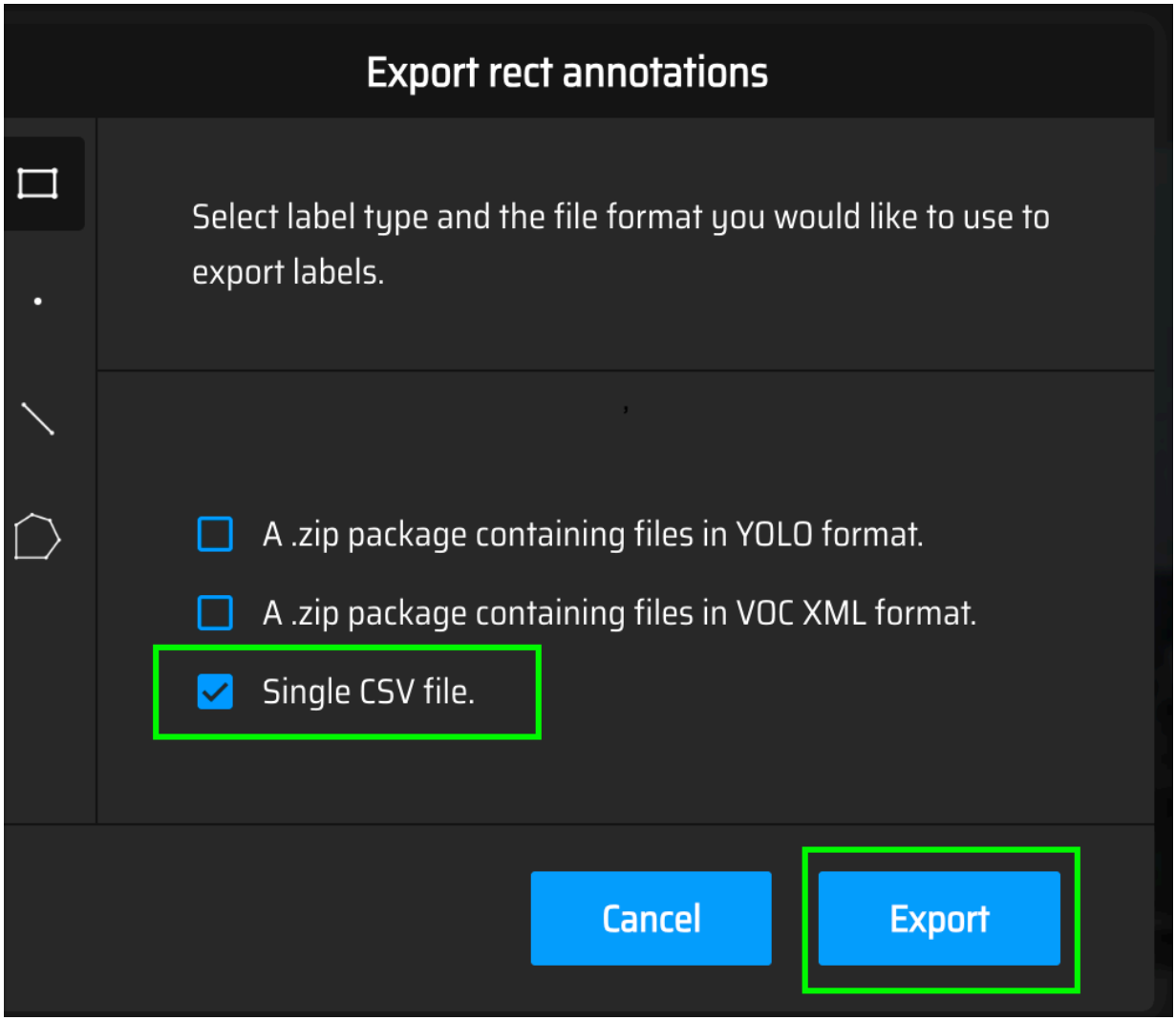
Once bounding boxes were drawn over accident scene. Click on Action to download annotated CSV file for all images.



Select Export Annotations



Select Single CSV file check box



CSV file will be downloaded to your local machine.

label_name	bbox_x	bbox_y	bbox_width	bbox_height	image_name	image_width	image_height
Accident	71	86	613	283	10_flip70.jpg	700	446
Accident	2	106	995	191	20_flip70.jpg	1004	402
Accident	8	71	135	63	3_flip.jpg	275	183
Accident	388	172	167	128	43_flip70.jpg	836	355

NOTE: However, the exported CSV had pixel-based coordinates, while Vertex AI requires normalized coordinates.

4. Converting Annotations to Vertex AI Format

Problem:

- makesense.ai output → (x_min, y_min, width, height) in pixels.
- Vertex AI expects → normalized coordinates between 0 and 1.

Solution:

I wrote **ConvertMakeSenseToVertexAIAutoMLCSV.java to**

- Read the makesense CSV file.
- Normalize coordinates:

```
double xmin = x / imageWidth;
double ymin = y / imageHeight;
double xmax = (x + width) / imageWidth;
double ymax = (y + height) / imageHeight;
```

- Save the converted file in Vertex AI–compatible format.

```java
/**
 * @author lalamanil
 *
 */
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStreamReader;

public class ConvertMakeSenseToVertexAIAutoMLCSV {

    public static void saveCSV(String data, String outputFilePath) {
        BufferedWriter bw = null;
        try {
            bw = new BufferedWriter(new FileWriter(new File(outputFilePath)));
            bw.write(data);

            System.out.println("Created VertexAI AutoML compatible CSV annotated file at:" + outputFilePath);
        } catch (IOException e) {
            // TODO: handle exception
            e.printStackTrace();
        } finally {
            if (null != bw) {

                try {
                    bw.close();
                } catch (IOException e) {
                    // TODO: handle exception
                    e.printStackTrace();
                }

            }
        }

    }
```

```java
public static void mapMakeSenceCSVToAutoMlCSV(String makeSenseFilePath, String outputFilePath) {

    BufferedReader br = null;

    try {
        br = new BufferedReader(new InputStreamReader(new FileInputStream(makeSenseFilePath)));

        String line = null;

        boolean isFirst = Boolean.TRUE;

        StringBuilder builder = new StringBuilder();

        while ((line = br.readLine()) != null) {

            if (isFirst) {

                builder.append("SET,IMAGE_PATH,LABEL,X_MIN,Y_MIN,X_MAX,Y_MIN,X_MAX,Y_MAX,X_MIN,Y_MAX");
                builder.append("\r\n");
                isFirst = Boolean.FALSE;
                continue;
            }

            String[] elements = line.split(",");

            if (elements.length != 8) {
                continue;
            }

            String label = elements[0];

            double x = Double.parseDouble(elements[1]);
            double y = Double.parseDouble(elements[2]);
            double width = Double.parseDouble(elements[3]);
            double height = Double.parseDouble(elements[4]);
            String imageName = elements[5];

            double imageWidth = Double.parseDouble(elements[6]);
            double imageHeight = Double.parseDouble(elements[7]);
```

```java
            // converting pixel value to normalized values (0 to 1)
            double xmin = x / imageWidth;
            double ymin = y / imageHeight;
            double xmax = (x + width) / imageWidth;
            double ymax = (y + height) / imageHeight;

            String entry = String.format("UNASSIGNED,%s,%s,%.6f,%.6f,%s,%s,%.6f,%.6f,%s,%s", imageName, label, xmin,
                    ymin, "", "", xmax, ymax, "", "");
            builder.append(entry);
            builder.append("\r\n");

        }

        // System.out.println(builder.toString());
        saveCSV(builder.toString(), outputFilePath);

    } catch (FileNotFoundException e) {
        // TODO: handle exception
        e.printStackTrace();
    } catch (IOException e) {
        // TODO: handle exception
        e.printStackTrace();
    } finally {

        if (null != br) {

            try {
                br.close();
            } catch (IOException e) {
                // TODO: handle exception
                e.printStackTrace();
            }

        }
    }

}
```

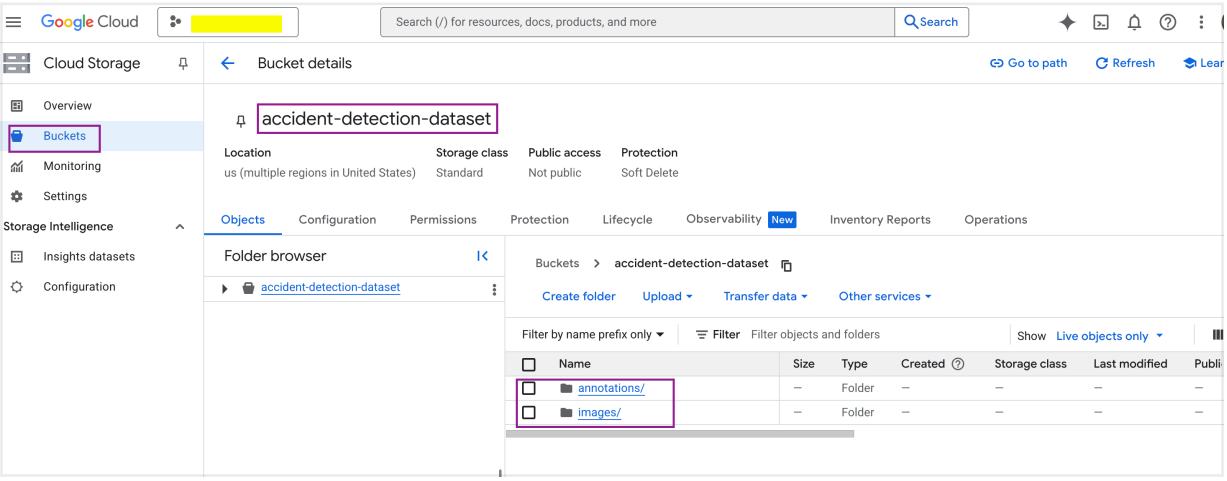
```
    public static void main(String[] args) {

        String makeSenseFile = "/Users/lalamanil/voiceanalyzer/HighwayTraffic/labels_n_2025-08-04-09-17-15.csv";
        String outputFile = "/Users/lalamanil/voiceanalyzer/accidentDataSet/custom_dataset/RenamedImages/LabelImages/changeTest.csv";

        mapMakeSenceCSVToAutoMlCSV(makeSenseFile, outputFile);

    }
}
```

Below is the Vertex AI compatible Annotated CSV file generated from above scrip.

SET	IMAGE_PATH	LABEL	X_MIN	Y_MIN	X_MAX	Y_MIN	X_MAX	Y_MAX	X_MIN	Y_MAX
UNASSIGNED	10_flip70.jpg	Accident	0.101429	0.192825			0.977143	0.827354		
UNASSIGNED	20_flip70.jpg	Accident	0.001992	0.263682			0.993028	0.738806		
UNASSIGNED	3_flip.jpg	Accident	0.029091	0.387978			0.520000	0.732240		
UNASSIGNED	43_flip70.jpg	Accident	0.464115	0.484507			0.663876	0.845070		

5. Uploading Data to Google Cloud Storage

With images and annotations ready, I created a **Google Cloud Storage bucket**:



Uploaded images and normalized annotation CSV to respective folder with in the bucket

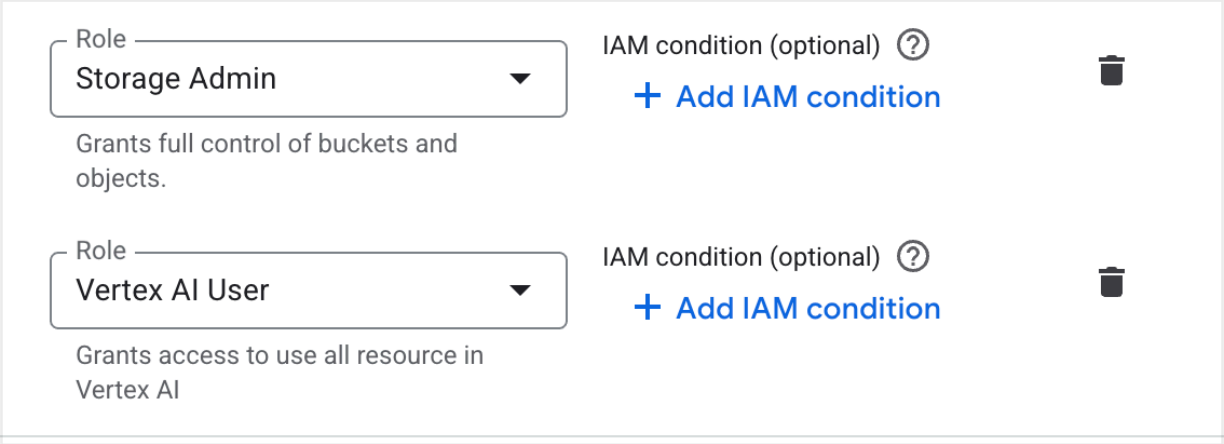
- ✅ **images/ → all .jpg frames.**
- ✅ **annotations/ → normalized annotation CSV.**

I have written GCSStorageUtility.java to upload both images and CSV to GCS

via Java client libraries.

Prerequisite:

Create a service account in Google cloud console and provide below roles



Add below dependency to pom.xml

```
    <dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>com.google.cloud</groupId>
                <artifactId>libraries-bom</artifactId>
                <version>26.58.0</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
        </dependencies>
    </dependencyManagement>

    <dependencies>
        <dependency>
            <groupId>com.google.cloud</groupId>
            <artifactId>google-cloud-storage</artifactId>
        </dependency>
```

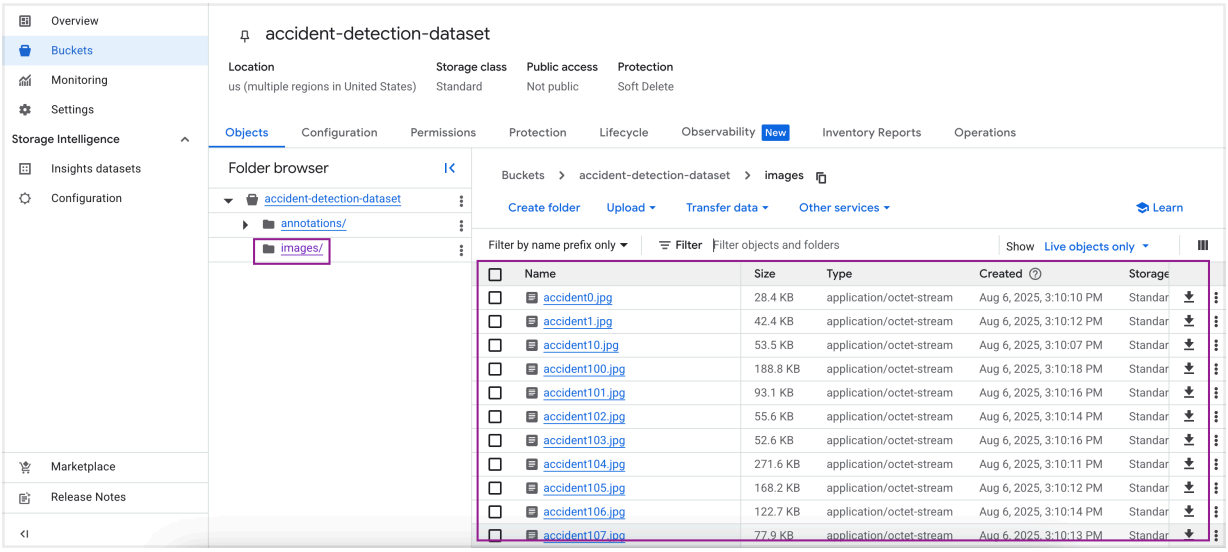
```java
package com.video.object.tracking.utility;
/*
 * @author lalamanil
 */
import java.io.File;
import java.io.IOException;
import java.io.InputStream;
import java.nio.file.Files;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.TimeUnit;
import com.google.auth.oauth2.GoogleCredentials;
import com.google.cloud.storage.Blob;
import com.google.cloud.storage.BlobId;
import com.google.cloud.storage.BlobInfo;
import com.google.cloud.storage.Storage;
import com.google.cloud.storage.StorageOptions;

public class GCSStorageUtility {

    private static Storage storage;
    //Bucket name created in GCP console
    private static String bucketName = "accident-detection-dataset";
    //Folder where images will be stored with in the bucket
    private static String imageprefix = "images/";
    //Folder where Annotated CSV file will be stored with in the bucket
    private static String annotatedCSVPrefix = "annotations/";
    //Local ImageFolder path where images are located
    private static String imageFolderPath = "/Users/lalamanil/voiceanalyzer/accidentDataSet/custom_dataset/RenamedImages";
```

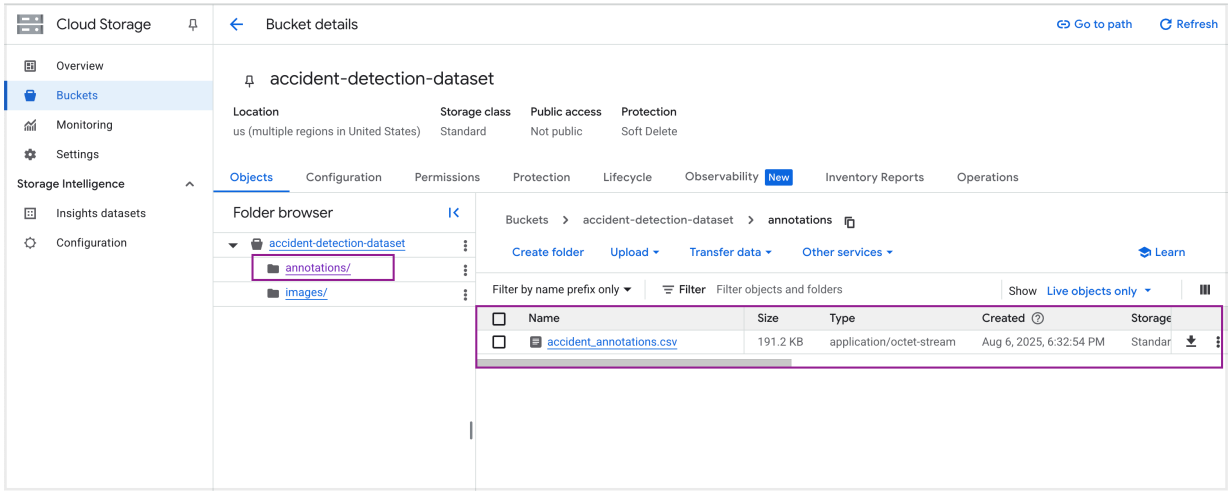
```java
    //Local annotated CSV file path where annotated CSV file is located
    private static String annotationCSVFilePath = "/Users/lalamanil/voiceanalyzer/accidentDataSet/custom_dataset/RenamedImages/LabelImages/gcsauto

    static {
        try {
            //reading service account
            InputStream inputStream = GCSStorageUtility.class.getClassLoader()
                    .getResourceAsStream("ServiceAccount.json");
            GoogleCredentials googleCredentials = GoogleCredentials.fromStream(inputStream)
                    .createScoped("https://www.googleapis.com/auth/cloud-platform");
            storage = StorageOptions.newBuilder().setCredentials(googleCredentials).build().getService();
        } catch (IOException e) {
            // TODO: handle exception
            e.printStackTrace();
        }
    }

    // uploading images (dataset) to google cloud storage bucket
    public static void uploadImageFiles() {
        File imageFolder = new File(imageFolderPath);
        File[] files = imageFolder.listFiles((dir, name) -> name.toLowerCase().endsWith(".jpg"));
        ExecutorService executorService = Executors.newFixedThreadPool(10);
        for (File file : files) {
            executorService.submit(() -> {
                try {
                    BlobId blobId = BlobId.of(bucketName, imageprefix + file.getName());
                    BlobInfo blobInfo = BlobInfo.newBuilder(blobId).build();
                    Blob savedBlob = storage.create(blobInfo, Files.readAllBytes(file.toPath()));
```

```
                System.out.println(savedBlob.getBlobId() + " " + file.getName());
            } catch (IOException e) {
                // TODO: handle exception
                e.printStackTrace();
            }
        });
    }
    executorService.shutdown();
    try {
        executorService.awaitTermination(1, TimeUnit.HOURS);
    } catch (InterruptedException e) {
        // TODO: handle exception
        e.printStackTrace();
    }
    System.out.println("✅ All uploads finished.");

}

// uploading Annotated csv file to GCS bucket
public static void uploadAnnotatedCSVFile() {
    File file = new File(annotationCSVFilePath);
    BlobId blobId = BlobId.of(bucketName, annotatedCSVPrefix + "accident_annotations.csv");
    BlobInfo blobInfo = BlobInfo.newBuilder(blobId).build();
    try {
        Blob blob = storage.create(blobInfo, Files.readAllBytes(file.toPath()));
        System.out.println(blob.getBlobId() + ":" + file.getName());
        System.out.println("✅  upload successful.");
    } catch (IOException e) {
```

```
            // TODO: handle exception
            e.printStackTrace();
        }
    }

    public static void main(String[] args) {
        uploadImageFiles();
        //uploadAnnotatedCSVFile();
    }

}
```

Images are uploaded to GCS bucket under images/



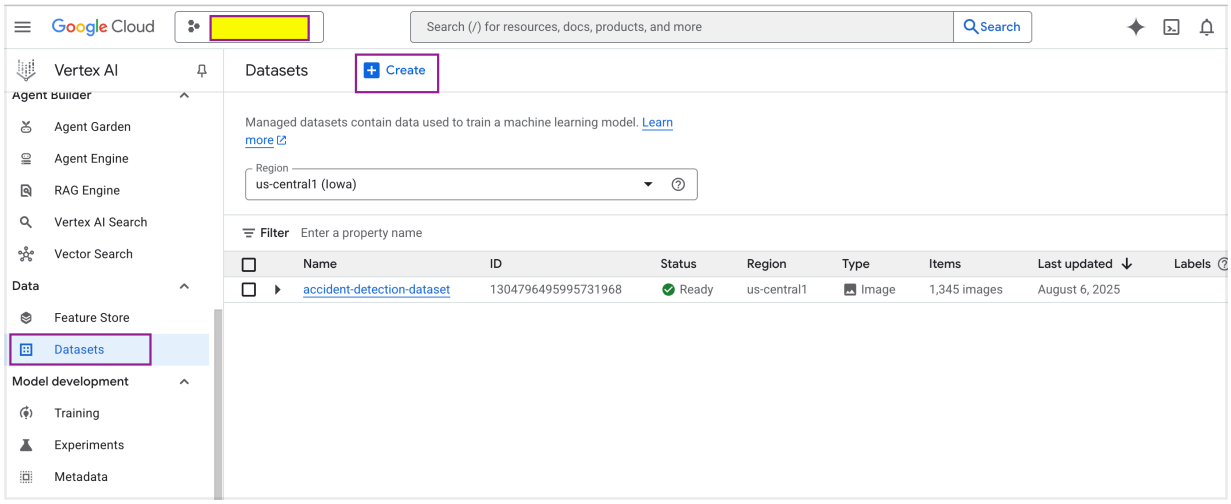


Make sure to prefix image path with GCS image folder path as show in below screen shot before uploading annotated CSV to GCS bucket

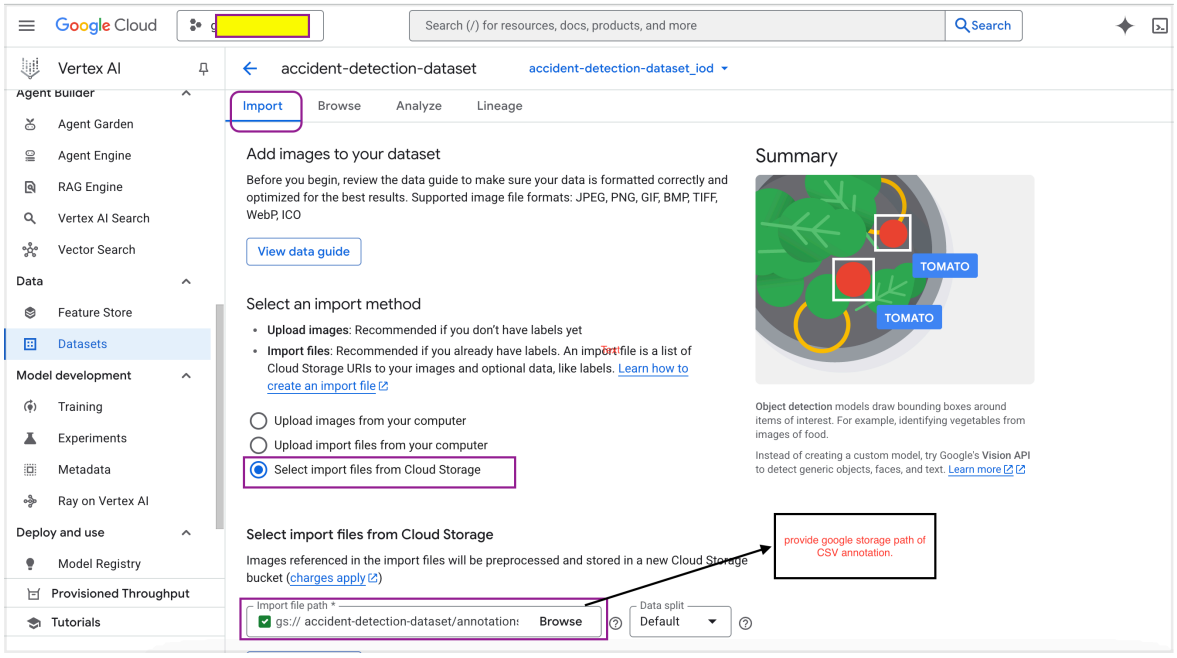
UNASSIGNED	gs://accident-detection-dataset/images/accident0.jpg	Accident	0.074286	0.260090			0.985714	0.872197
UNASSIGNED	gs://accident-detection-dataset/images/accident1.jpg	Accident	0.438429	0.011516			0.749469	0.571977
UNASSIGNED	gs://accident-detection-dataset/images/accident2.jpg	Accident	0.050193	0.298969			0.660232	0.881443
UNASSIGNED	gs://accident-detection-dataset/images/accident3.jpg	Accident	0.424710	0.216495			0.942085	0.871134
UNASSIGNED	gs://accident-detection-dataset/images/accident4.jpg	Accident	0.007722	0.103093			0.799228	1.000000
UNASSIGNED	gs://accident-detection-dataset/images/accident5.jpg	Accident	0.172131	0.402321			0.873770	0.773694
UNASSIGNED	gs://accident-detection-dataset/images/accident6.jpg	Accident	0.178063	0.199620			0.858974	0.716730
UNASSIGNED	gs://accident-detection-dataset/images/accident7.jpg	Accident	0.191218	0.269598			0.872521	0.795411
UNASSIGNED	gs://accident-detection-dataset/images/accident8.jpg	Accident	0.128936	0.299213			0.511244	0.702756
UNASSIGNED	gs://accident-detection-dataset/images/accident9.jpg	Accident	0.610127	0.195358			0.832911	0.508704
UNASSIGNED	gs://accident-detection-dataset/images/accident9.jpg	Accident	0.013924	0.375242			0.483544	0.918762

6. Importing Data into Vertex AI

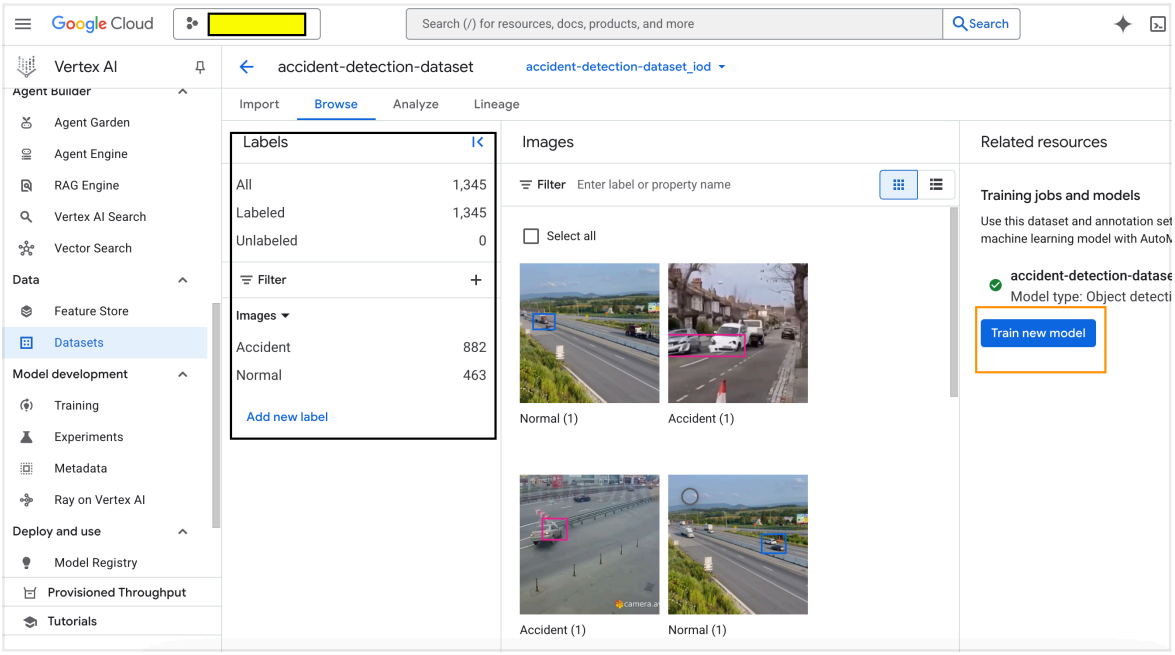
✅ **Vertex AI → Datasets** in the Google Cloud Console. Create the Dataset



Import the data (Images) stored in Google Cloud Storage via Annotate CSV file store in GCS bucket



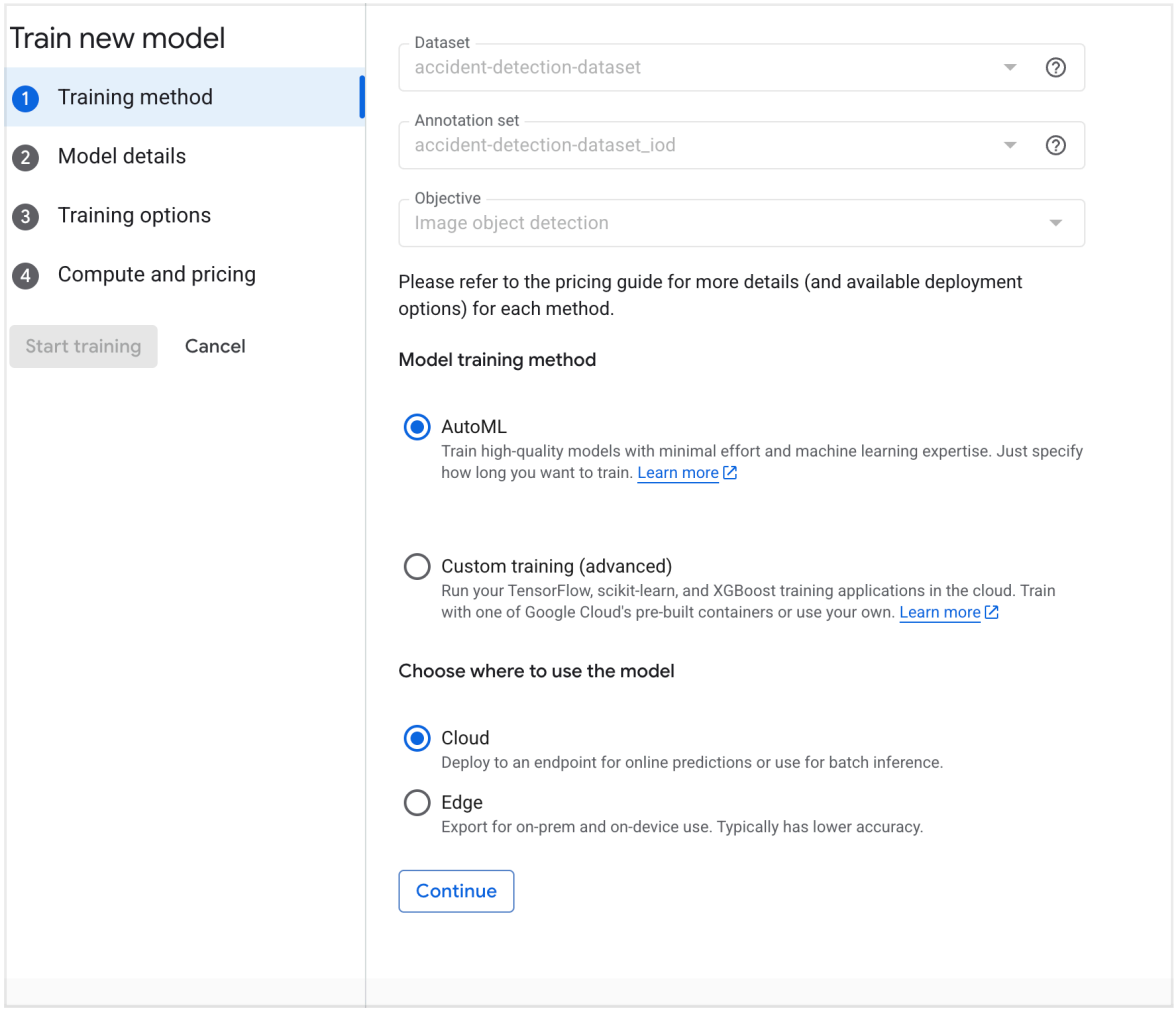
Vertex AI automatically linked each annotation with its image. Verified the images and bounding boxes were correctly displayed.

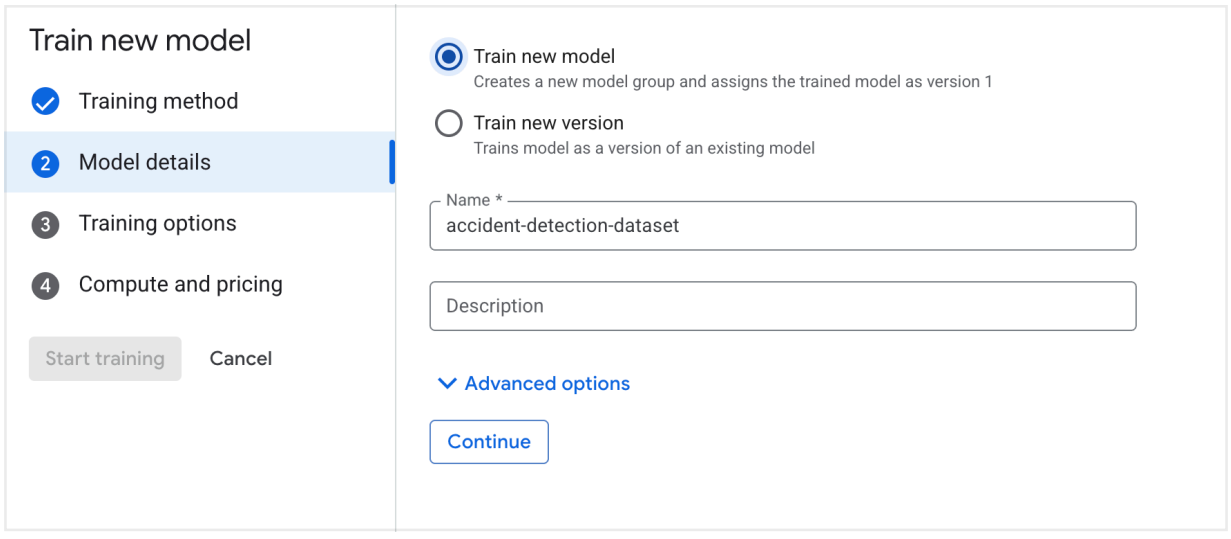


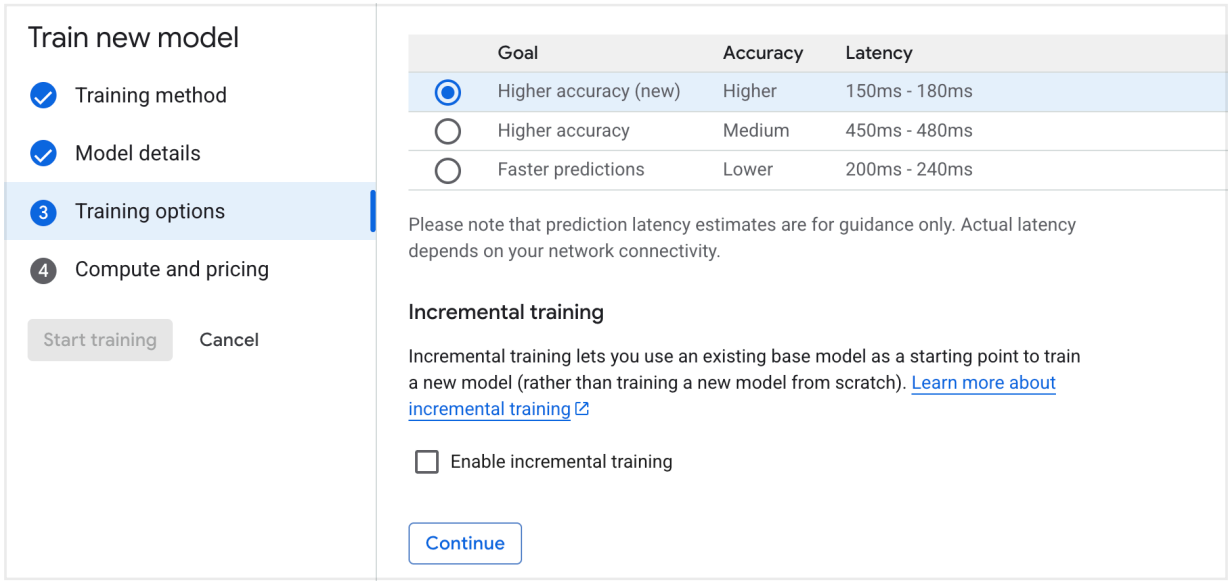
7. Training the Model

Trained a **Custom Object Detection** model in Vertex AI:

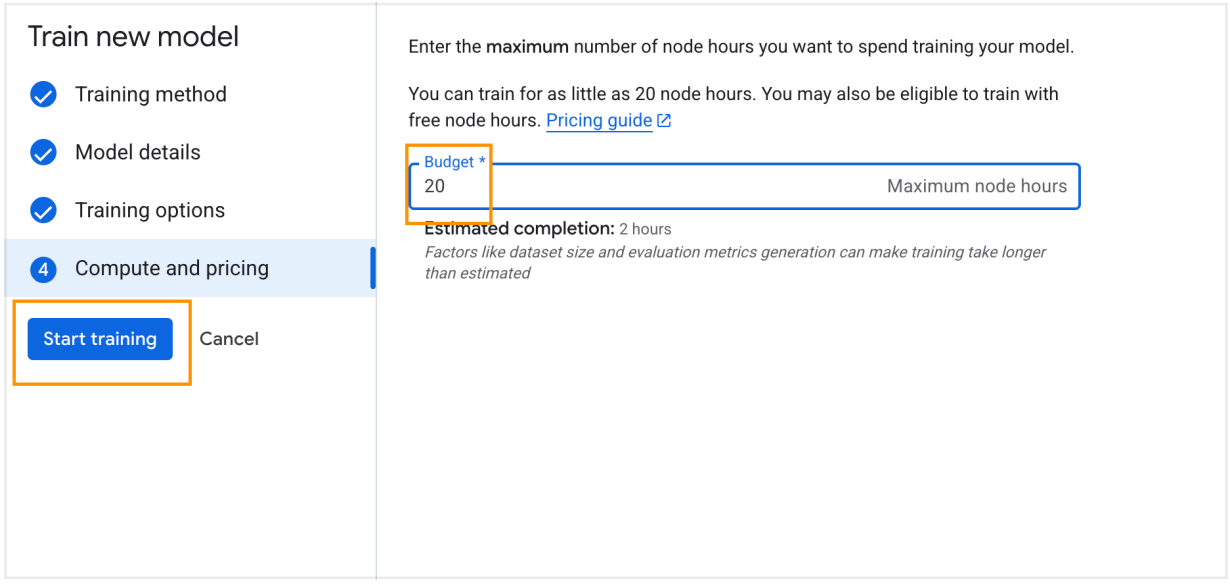
✅ Selected dataset.


Train new model
1 Training method
2 Model details
3 Training options
4 Compute and pricing
Start training
Cancel
Dataset
accident-detection-dataset
Annotation set
accident-detection-dataset_iod
Objective
Image object detection
Please refer to the pricing guide for more details (and available deployment options) for each method.
Model training method
AutoML
Train high-quality models with minimal effort and machine learning expertise. Just specify how long you want to train. Learn more
Custom training (advanced)
Run your TensorFlow, scikit-learn, and XGBoost training applications in the cloud. Train with one of Google Cloud's pre-built containers or use your own. Learn more
Choose where to use the model
Cloud
Deploy to an endpoint for online predictions or use for batch inference.
Edge
Export for on-prem and on-device use. Typically has lower accuracy.
Continue

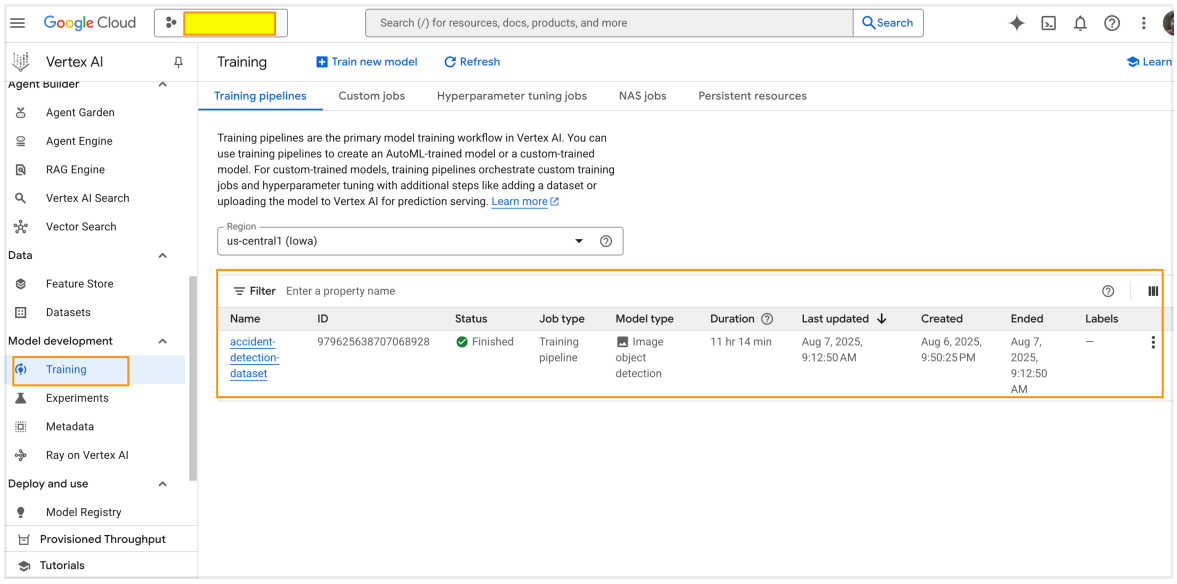

Train new model
Training method
2 Model details
3 Training options
4 Compute and pricing
Start training
Cancel
Train new model
Creates a new model group and assigns the trained model as version 1
Train new version
Trains model as a version of an existing model
Name *
accident-detection-dataset
Description
Advanced options
Continue



Chose the training budget in node hours based on dataset size.

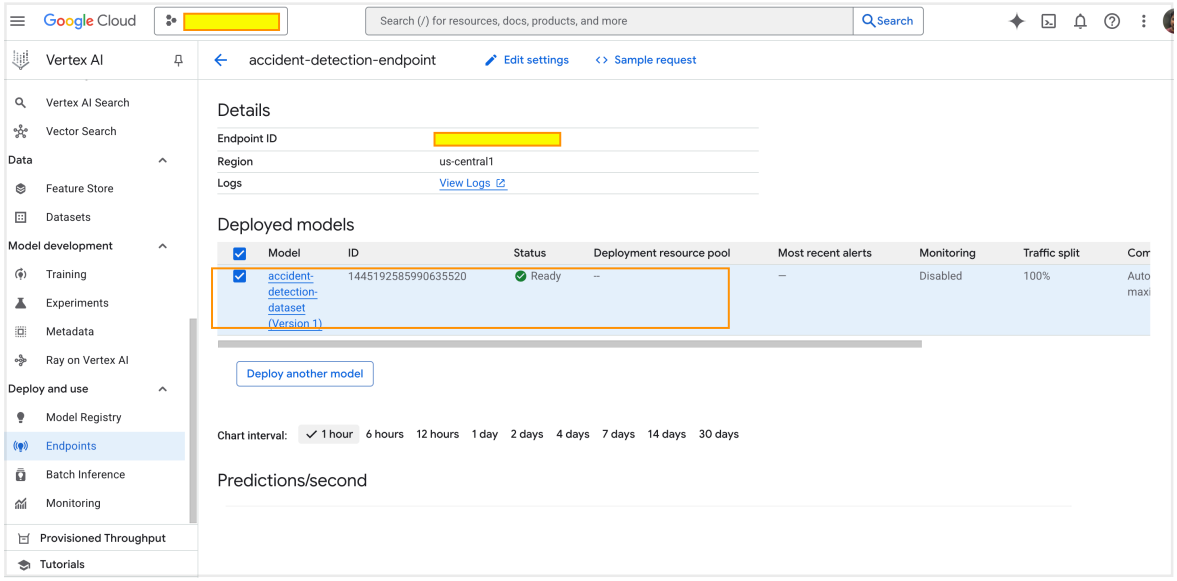


Training completed successfully, producing an evaluated model with precision and recall metrics.

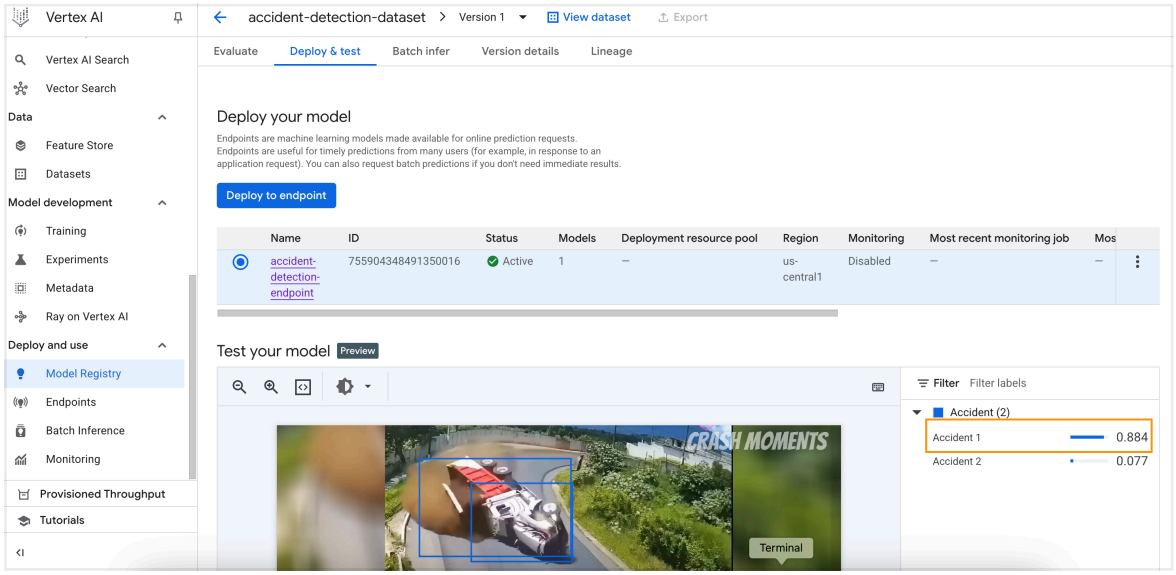


8. Deploying the Model

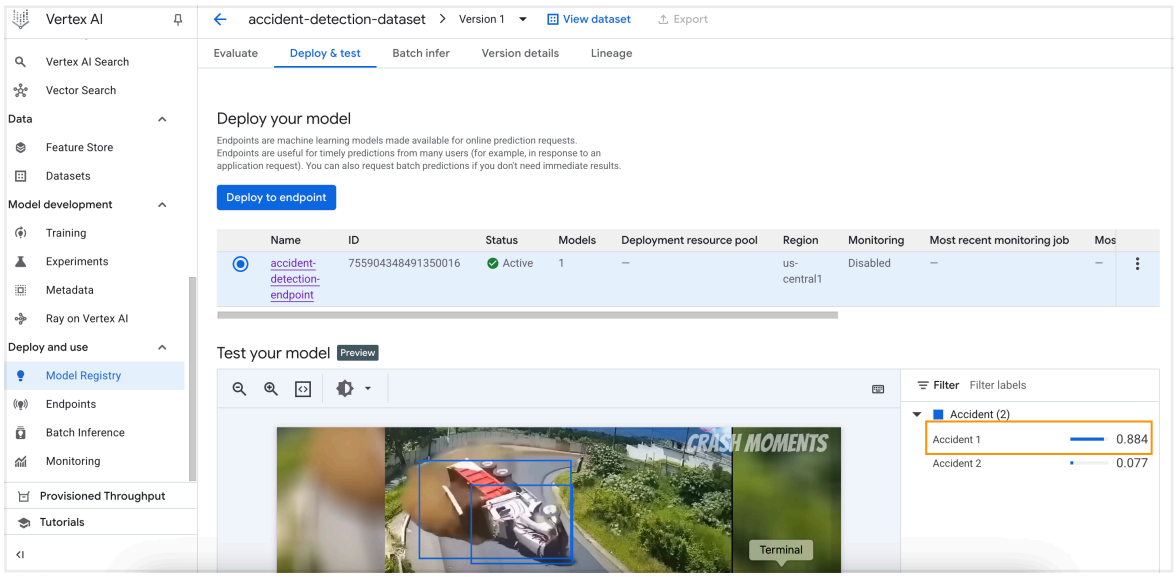
- Once trained, Deployed the model to an **endpoint**: **Configured 1 active node for real-time predictions.** It will generate Endpoint ID.



- **Used the Vertex AI Console to upload test images.**



✅ **Received bounding boxes, labels, and confidence scores in real time.**



9. Integrating Predictions into Java

Using **PredictionServiceClient**, Integrated the deployed model into a Java application:

- ✅ Passed the image bytes to the Vertex AI endpoint.
- ✅ Parsed the prediction results.
- ✅ Extracted:
  Bounding box coordinates.
  Confidence score

Detected label.

Add below dependency to pom.xml

```
<dependency>
    <groupId>com.google.cloud</groupId>
    <artifactId>google-cloud-aiplatform</artifactId>
</dependency>
```

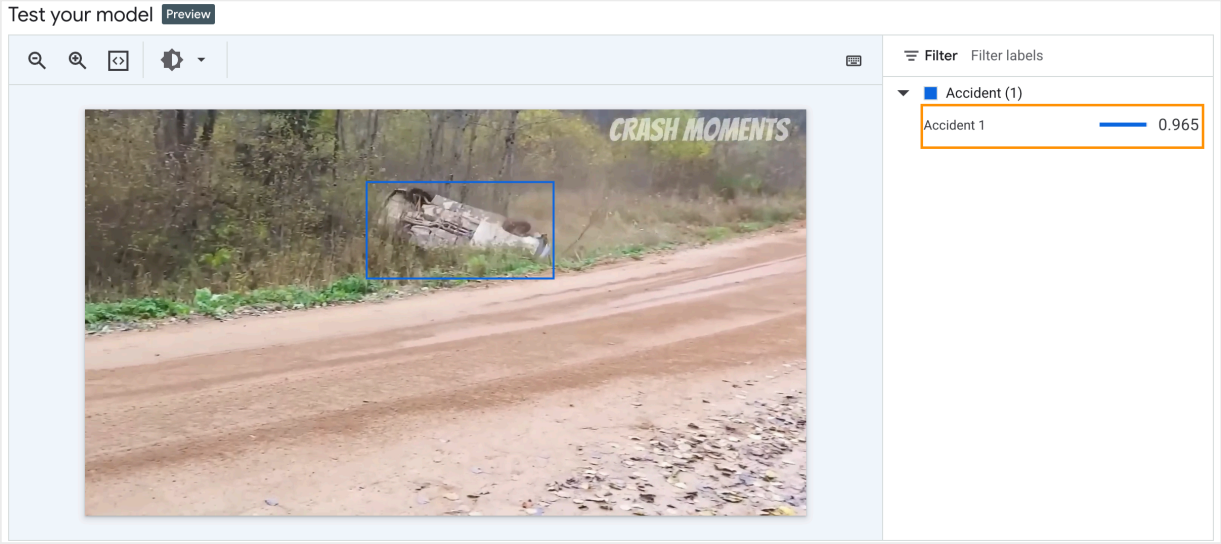
```
package com.video.object.tracking.utility;

/*
 * @Author Anil Lalam
 */

import java.io.File;
import java.io.IOException;
import java.io.InputStream;
import java.nio.file.Files;
import java.util.Base64;
import java.util.List;
import com.google.api.gax.core.FixedCredentialsProvider;
import com.google.auth.oauth2.GoogleCredentials;
import com.google.cloud.aiplatform.v1.EndpointName;
import com.google.cloud.aiplatform.v1.PredictRequest;
import com.google.cloud.aiplatform.v1.PredictResponse;
import com.google.cloud.aiplatform.v1.PredictionServiceClient;
import com.google.cloud.aiplatform.v1.PredictionServiceSettings;
import com.google.protobuf.Struct;
import com.google.protobuf.Value;

public class AccidentDetectionPredictionModel {

    private static PredictionServiceClient predictionServiceClient;

    static {
        InputStream inputStream = AccidentDetectionPredictionModel.class.getClassLoader()
                .getResourceAsStream("ServiceAccount.json");
        if (null != inputStream) {
            try {
                GoogleCredentials googleCredentials = GoogleCredentials.fromStream(inputStream);
                PredictionServiceSettings predictionServiceSettings = PredictionServiceSettings.newBuilder()
                        .setCredentialsProvider(FixedCredentialsProvider.create(googleCredentials)).build();
                predictionServiceClient = PredictionServiceClient.create(predictionServiceSettings);

            } catch (IOException e) {
                // TODO: handle exception
                e.printStackTrace();
```

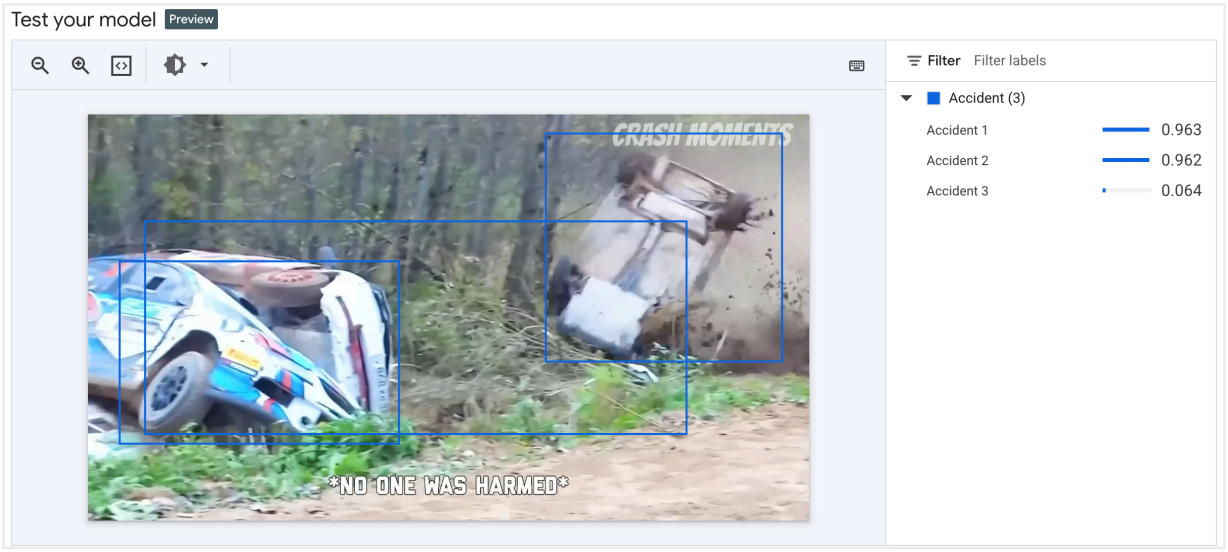
```
            }

        } else {
            System.out.println("Inputstream is null. Please check service Account in src/main/resources");
        }

    }

    public static void predictImage(String projectId, String location, String endpointId, String localImagePath) {
        if (null != predictionServiceClient) {
            try {
                // read image as bytes
                byte[] content = Files.readAllBytes(new File(localImagePath).toPath());
                String base64Image = Base64.getEncoder().encodeToString(content);
                // Build instance
                Struct instance = Struct.newBuilder()
                        .putFields("content", Value.newBuilder().setStringValue(base64Image).build()).build();

                EndpointName endpointName = EndpointName.of(projectId, location, endpointId);
                PredictRequest request = PredictRequest.newBuilder().setEndpoint(endpointName.toString())
                        .addInstances(Value.newBuilder().setStructValue(instance).build()).build();
                PredictResponse predictResponse = predictionServiceClient.predict(request);
                System.out.println("Prediction results:");
                if (null != predictResponse) {
                    List<Value> predictionList = predictResponse.getPredictionsList();
                    if (null != predictionList && !predictionList.isEmpty()) {
                        for (Value prediction : predictionList) {
                            Struct predictionStruct = prediction.getStructValue();
                            // labels
                            List<Value> labels = predictionStruct.getFieldsOrThrow("displayNames").getListValue()
                                    .getValuesList();
                            // confidence scores
                            List<Value> scores = predictionStruct.getFieldsOrThrow("confidences").getListValue()
                                    .getValuesList();
                            // Bounding boxes
                            List<Value> bboxes = predictionStruct.getFieldsOrThrow("bboxes").getListValue()
                                    .getValuesList();

                            for (int i = 0; i < labels.size(); i++) {
                                String label = labels.get(i).getStringValue();
                                double score = scores.get(i).getNumberValue();
                                List<Value> box = bboxes.get(i).getListValue().getValuesList();
                                double yMin = box.get(0).getNumberValue();
                                double xMin = box.get(1).getNumberValue();
                                double yMax = box.get(2).getNumberValue();
                                double xMax = box.get(3).getNumberValue();
                                System.out.printf(
                                        "Object: %s, Confidence: %.2f,BBox[yMin=%.2f, xMin=%.2f, yMax=%.2f,xMax=%.2f]",
                                        label, score, yMin, xMin, yMax, xMax);
                                System.out.println();
                            }

                        }

                    } else {
                        System.out.println("predictionList is null or empty");
                    }
                } else {

                    System.out.println("predictResponse is null or empty");

                }

            } catch (IOException e) {
                // TODO: handle exception
                e.printStackTrace();
            }

        } else {
            System.out.println("predictionServiceClient is null or empty");
        }

    }

```

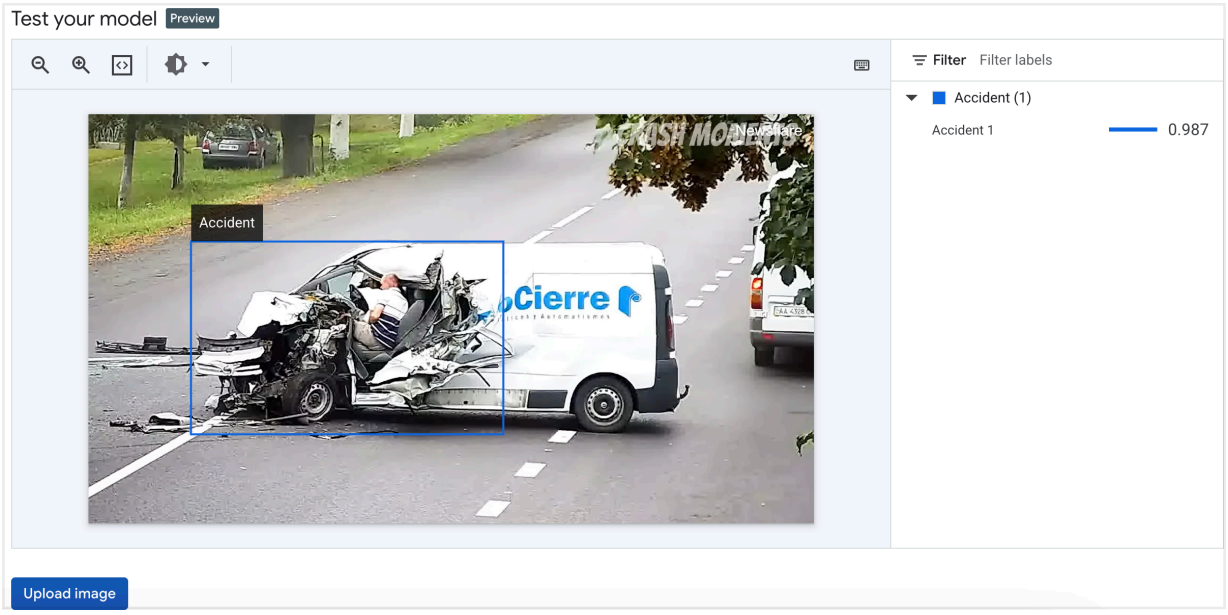
```
114     public static void main(String[] args) {
115
116         String projectId = [redacted];
117         String location = "us-central1";
118         String endpointId = '[redacted]
119         String locationPath = "/Users/lalamanil/voiceanalyzer/TestAccidentFrames/frame_28.jpg";
120
121         predictImage(projectId, location, endpointId, locationPath);
122
123     }
124
125 }
126
```

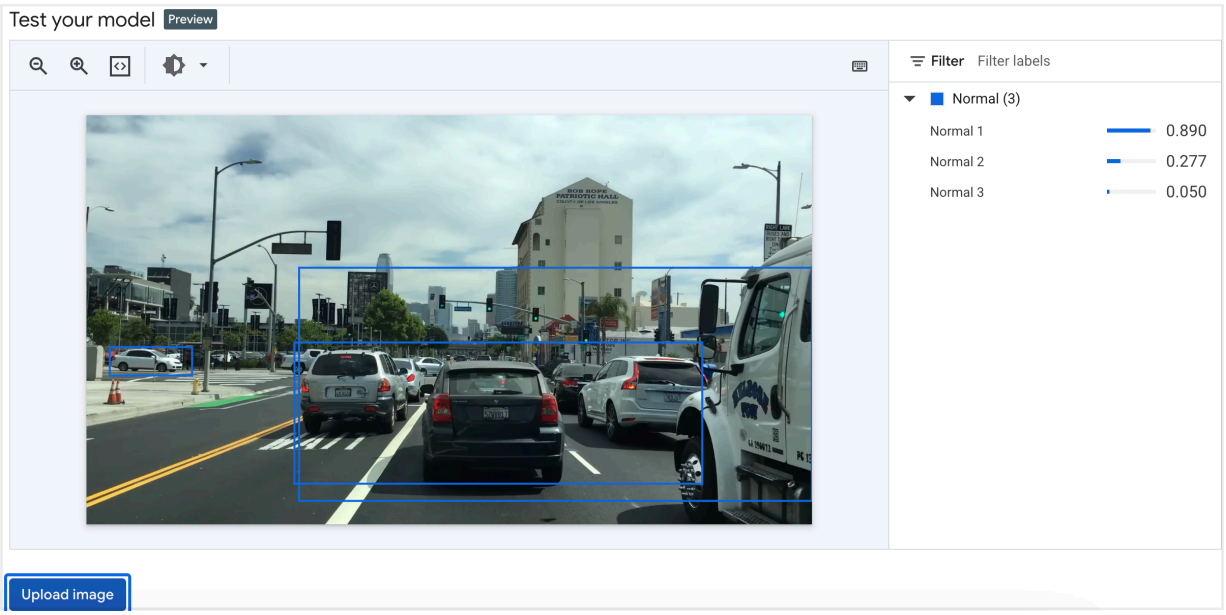
10. Results

- The model successfully detected accident frames with high confidence.
- Bounding boxes matched expected accident locations.
- Below are example before/after images with predicted bounding boxes.




Test your model
Preview
Filter
Filter labels
Accident (3)
Accident 1
0.963
Accident 2
0.962
Accident 3
0.064
CRASH MOMENTS
NO ONE WAS HARMED


Test your model
Preview
Filter
Filter labels
Accident (1)
Accident 1
0.987
Accident
Cierre
Upload image



11. Societal Impact and NIW Alignment

Road accidents are a pressing public safety concern. AI-powered detection systems like this can:

- Enable **faster emergency response**.
- Support **traffic monitoring** in smart cities.
- Help in **forensic analysis** of accident causes.

This project demonstrates **technical expertise** in AI and cloud systems while delivering **societal benefits** — a combination directly aligned with the **National Interest Waiver's goals** of fostering innovation that benefits the United States.

12. Future Enhancements

- Integrate **real-time streaming** from live dashcams.
- Expand dataset diversity for improved robustness.
- Use **GCP Pub/Sub** for automated accident alerts.

🟡 Deploy on **edge devices** for in-vehicle detection.

13. Conclusion

From **raw dashcam footage to a deployed AI model**, this project shows how **Java and Google Cloud Vertex AI can be combined** to deliver an intelligent accident detection system.

It’s a clear example of applying **AI for good** — advancing both **technological capability** and **public safety goals**.