

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

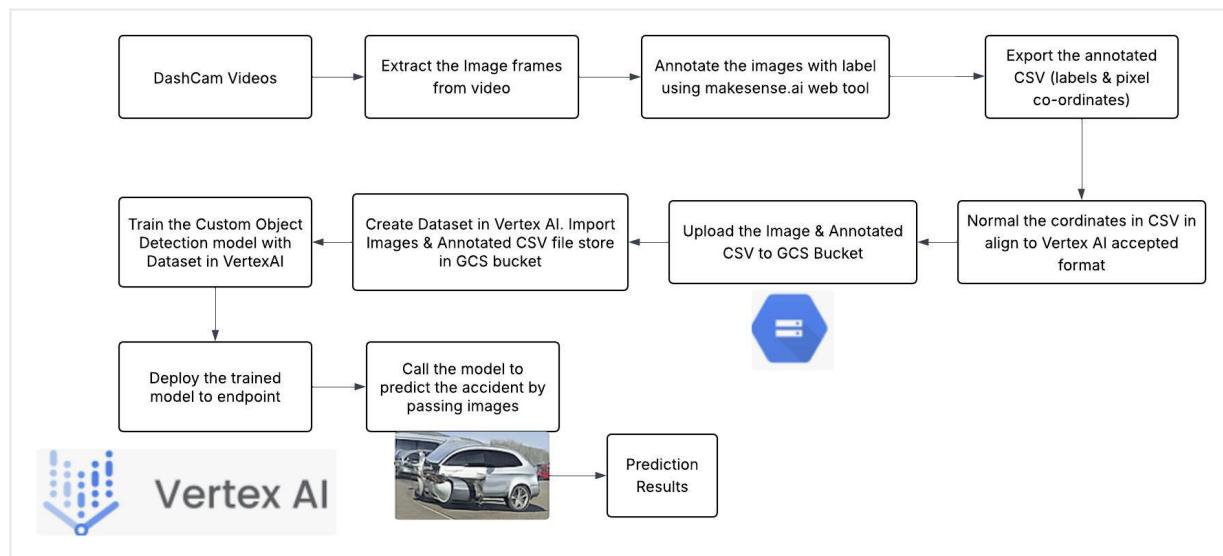
## 1. Introduction

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

```

<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

```

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

```

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

### 3.3 Annotating Images

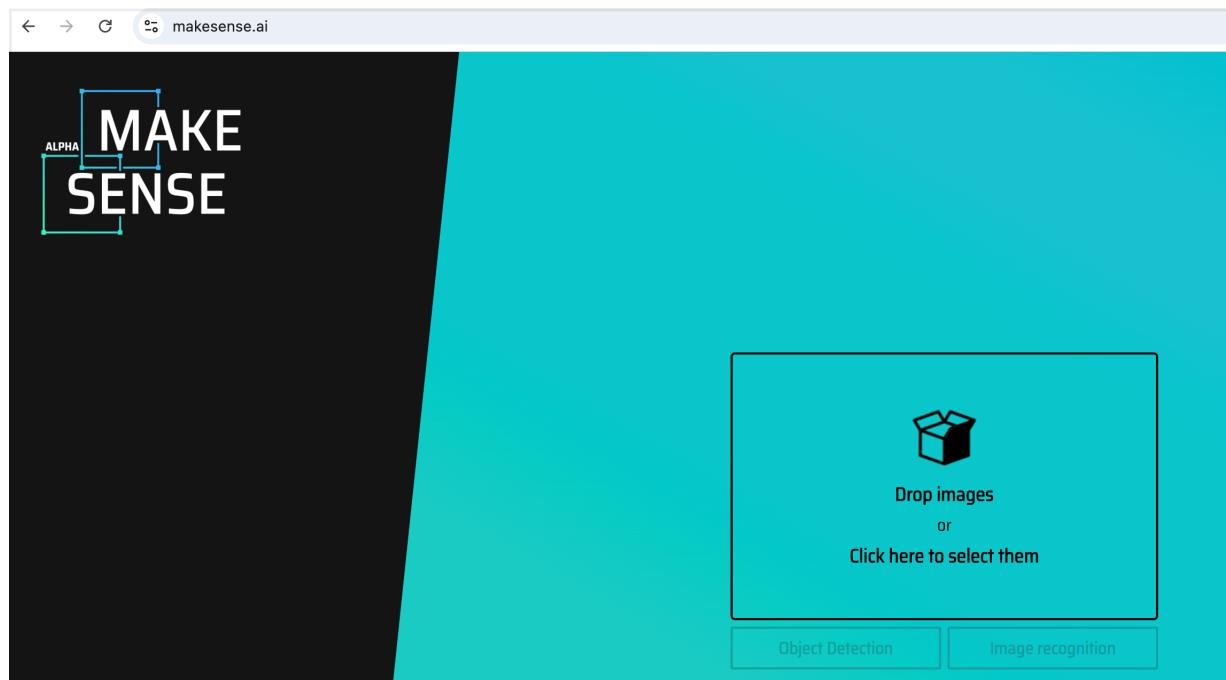
With the extracted frames, I used **makesense.ai** to annotate:

✓ **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.**



4 images loaded

Object Detection

Image recognition

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

## Create labels



Before you start, you can create a list of labels you plan to assign to objects in your project. You can also choose to skip that part for now and define label names as you go.

Insert label

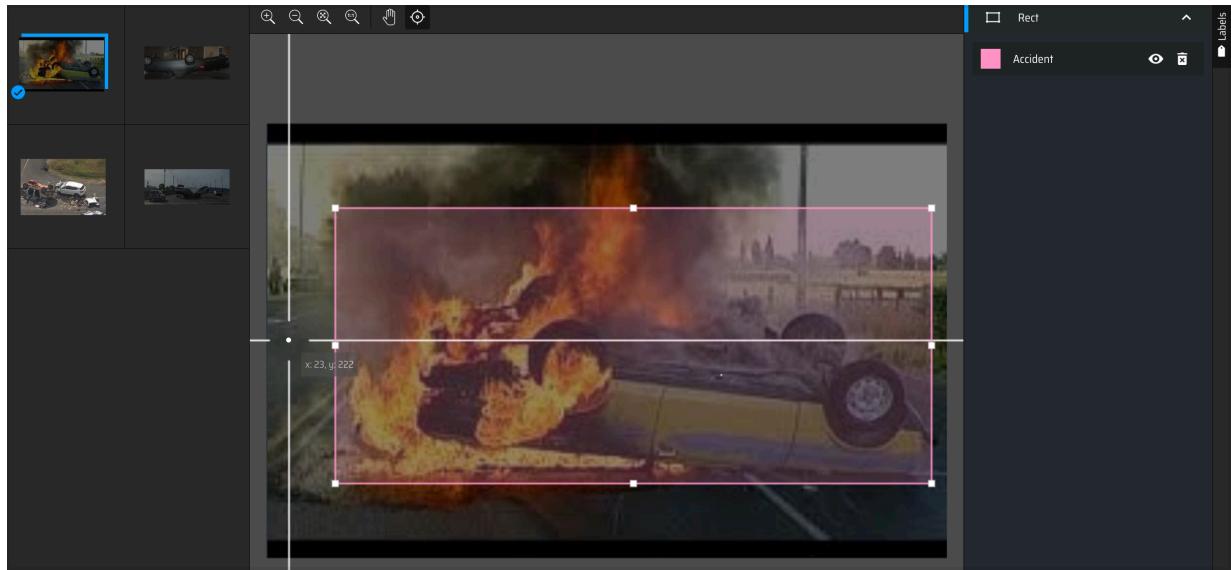
Accident



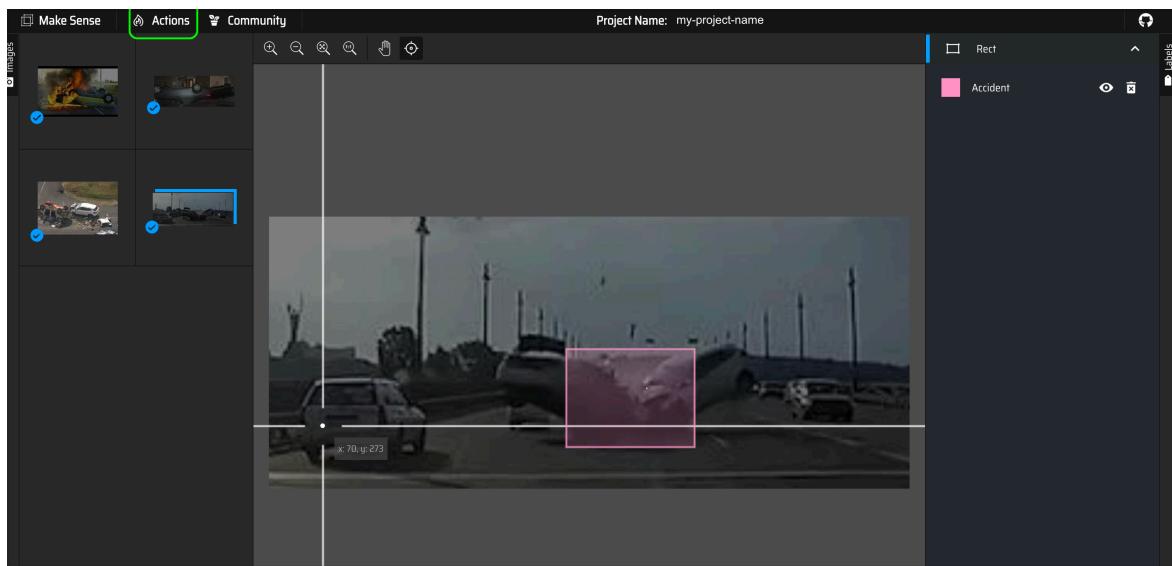
[Load labels from file](#)

[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.

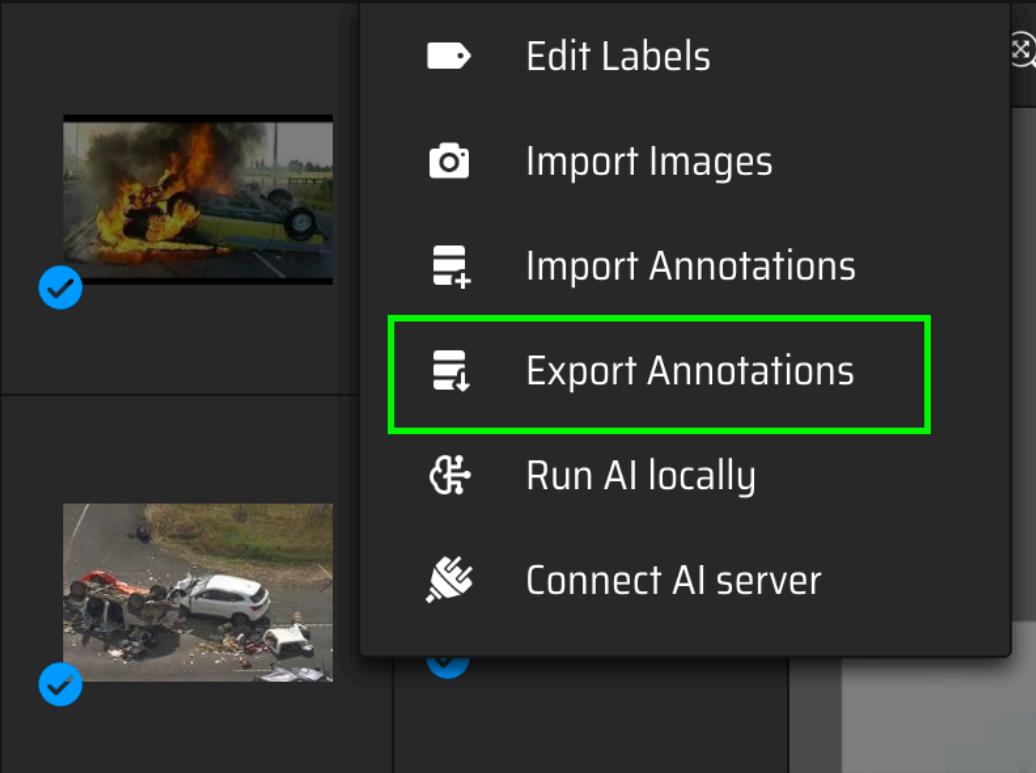


Select Export Annotations

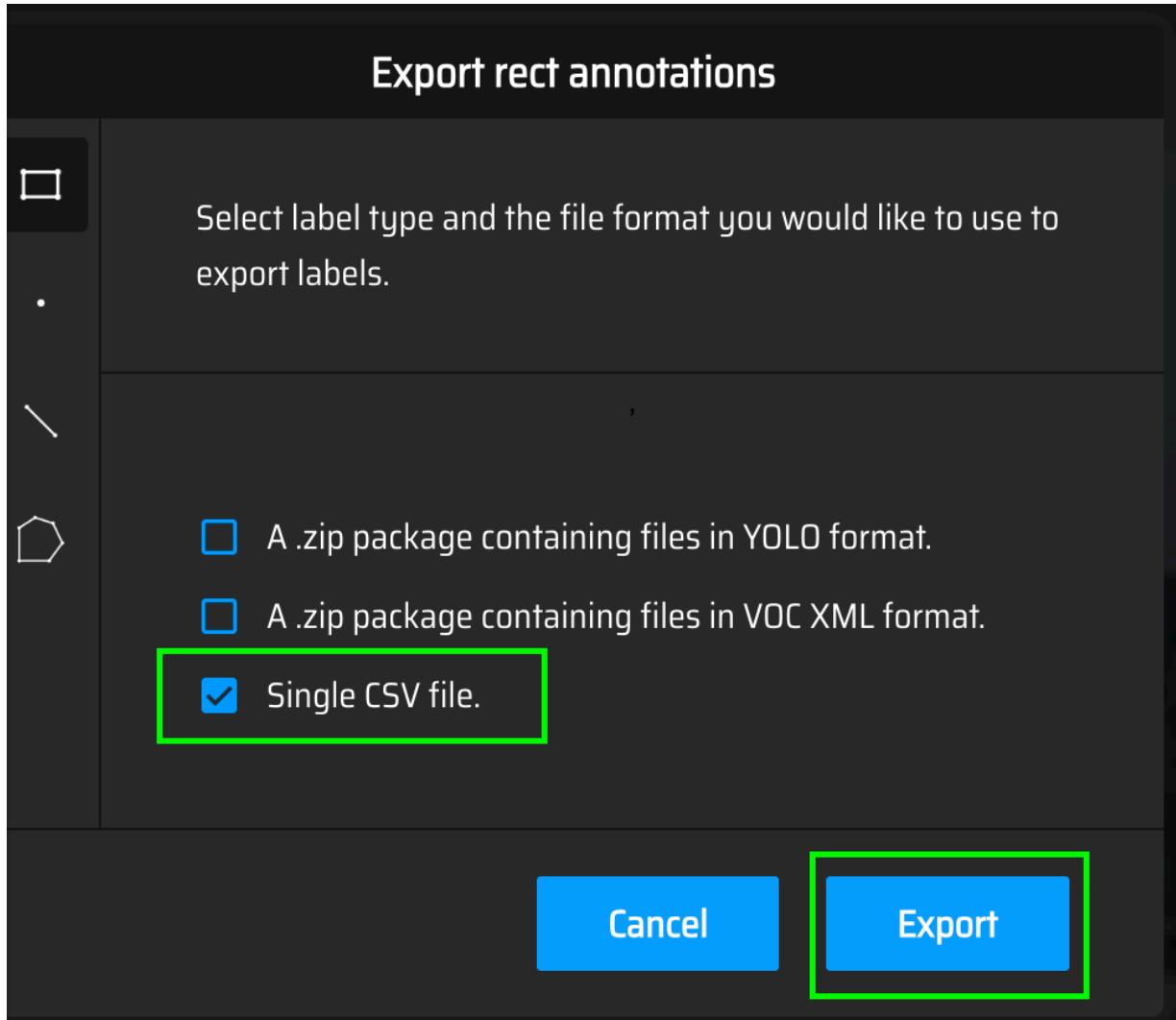
 Make Sense

 Actions

 Community



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.

```
/*
 * @author lalamani
 *
 */
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 outputPath) {
        BufferedWriter bw = null;
        try {
            bw = new BufferedWriter(new FileWriter(new File(outputPath)));
            bw.write(data);

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

Terminal

```

    public static void mapMakeSenceCSVToAutoMlCSV(String makeSenseFilePath, String outputPath) {
        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]);

                // 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(), outputPath);
        } 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/voiceanalyizer/HighwayTraffic/labels_n_2025-08-04-09-17-15.csv";
    String outputFile = "/Users/lalamanil/voiceanalyizer/accidentDataSet/custom_dataset/RenamedImages/LabelImages/changeTest.csv";
    mapMakeSenseCSVToAutoMLCSV(makeSenseFile, outputFile);
}
}

```

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

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**:

The screenshot shows the Google Cloud Storage Bucket details page for the 'accident-detection-dataset' bucket. The left sidebar shows 'Buckets' is selected. The main pane displays the bucket's configuration, including location (us), storage class (Standard), public access (Not public), and protection (Soft Delete). Below this, the 'Objects' tab is active, showing a list of objects. Two folders are listed: 'annotations/' and 'images/'. Both are marked as 'Folder' type and have a size of '-'. The 'Live objects only' filter is applied.

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

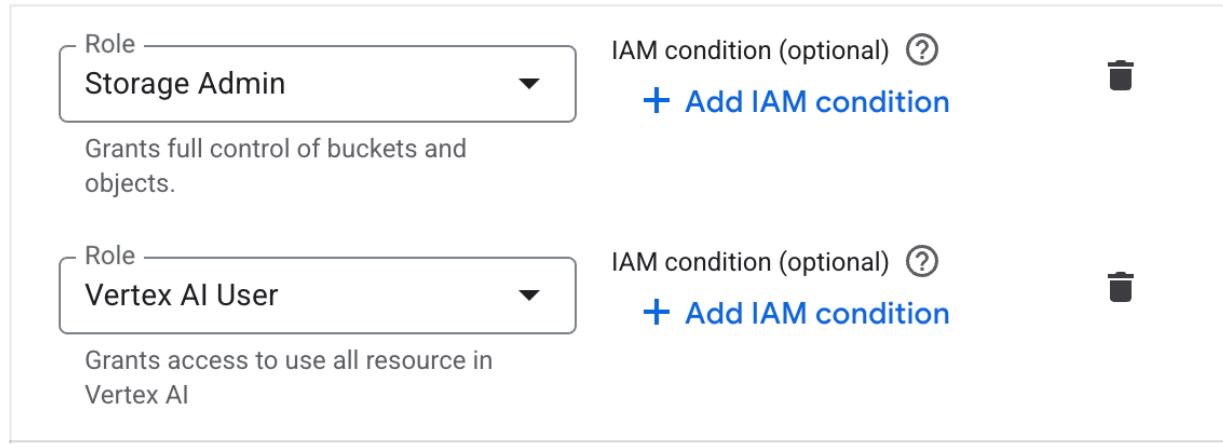
- ✓ images/ → all .jpg frames.
- ✓ annotations/ → normalized annotation CSV.

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



The screenshot shows two roles defined in Google Cloud IAM. Each role has a dropdown menu showing its name, a detailed description below it, an optional IAM condition section, and a trash can icon for deletion.

Role	IAM condition (optional)	Action
Storage Admin	+ Add IAM condition	trash can
Vertex AI User	+ Add IAM condition	trash can

**Storage Admin**  
Grants full control of buckets and objects.

**Vertex AI User**  
Grants access to use all resource in Vertex AI

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>
```

```

1 package com.video.object.tracking.utility;
2 /*
3  * @author lalamani.l
4 */
5 import java.io.File;
6 import java.io.IOException;
7 import java.io.InputStream;
8 import java.nio.file.Files;
9 import java.util.concurrent.ExecutorService;
10 import java.util.concurrent.Executors;
11 import java.util.concurrent.TimeUnit;
12 import com.google.auth.oauth2.GoogleCredentials;
13 import com.google.cloud.storage.Blob;
14 import com.google.cloud.storage.BlobId;
15 import com.google.cloud.storage.BlobInfo;
16 import com.google.cloud.storage.Storage;
17 import com.google.cloud.storage.StorageOptions;
18
19 public class GCSStorageUtility {
20
21     private static Storage storage;
22     //Bucket name created in GCP console
23     private static String bucketName = "accident-detection-dataset";
24     //Folder where images will be stored with in the bucket
25     private static String imageprefix = "images/";
26     //Folder where Annotated CSV file will be stored with in the bucket
27     private static String annotatedCSVPrefix = "annotations/";
28     //Local ImageFolder path where images are located
29     private static String imageFolderPath = "/Users/lalamani/voiceanalyzer/accidentDataSet/custom_dataset/RenamedImages";
30
31     //Local annotated CSV file path where annotated CSV file is located
32     private static String annotationCSVFilePath = "/Users/lalamani/voiceanalyzer/accidentDataSet/custom_dataset/RenamedImages/LabelImages/gcsauto";
33
34     static {
35         try {
36             //reading service account
37             InputStream inputStream = GCSStorageUtility.class.getClassLoader()
38                 .getResourceAsStream("ServiceAccount.json");
39             GoogleCredentials googleCredentials = GoogleCredentials.fromStream(inputStream)
40                 .createScoped("https://www.googleapis.com/auth/cloud-platform");
41             storage = StorageOptions.newBuilder().setCredentials(googleCredentials).build().getService();
42         } catch (IOException e) {
43             // TODO: handle exception
44             e.printStackTrace();
45         }
46     }
47
48     // uploading images (dataset) to google cloud storage bucket
49     public static void uploadImageFiles() {
50         File imageFolder = new File(imageFolderPath);
51         File[] files = imageFolder.listFiles((dir, name) -> name.toLowerCase().endsWith(".jpg"));
52         ExecutorService executorService = Executors.newFixedThreadPool(10);
53         for (File file : files) {
54             executorService.submit(() -> {
55                 try {
56                     BlobId blobId = BlobId.of(bucketName, imageprefix + file.getName());
57                     BlobInfo blobInfo = BlobInfo.newBuilder(blobId).build();
58                     Blob savedBlob = storage.create(blobInfo, Files.readAllBytes(file.toPath()));
59                 }
60             });
61         }
62     }

```

```

18         System.out.println(savedBlob.getBlobId() + ":" + file.getName());
19     } catch (IOException e) {
20         // TODO: handle exception
21         e.printStackTrace();
22     }
23 }
24 executorService.shutdown();
25 try {
26     executorService.awaitTermination(1, TimeUnit.HOURS);
27 } catch (InterruptedException e) {
28     // TODO: handle exception
29     e.printStackTrace();
30 }
31 System.out.println("✅ All uploads finished.");
32
33
34 // uploading Annotated csv file to GCS bucket
35 public static void uploadAnnotatedCSVFile() {
36     File file = new File(annotationCSVFilePath);
37     BlobId blobId = BlobId.of(bucketName, annotatedCSVPrefix + "accident_annotations.csv");
38     BlobInfo blobInfo = BlobInfo.newBuilder(blobId).build();
39     try {
40         Blob blob = storage.create(blobInfo, Files.readAllBytes(file.toPath()));
41         System.out.println(blob.getBlobId() + ":" + file.getName());
42         System.out.println("✅ upload successful.");
43     } catch (IOException e) {
44
45
46         // TODO: handle exception
47         e.printStackTrace();
48     }
49 }
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85

```

```

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

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

}

```

## Images are uploaded to GCS bucket under images/

The screenshot shows the Google Cloud Storage console interface. On the left, there's a sidebar with navigation links like Overview, Buckets, Monitoring, Settings, Storage Intelligence, Insights datasets, Configuration, Marketplace, and Release Notes. The 'Buckets' link is currently selected. In the main area, the 'accident-detection-dataset' bucket is selected. The 'Objects' tab is active. The 'images/' folder is expanded, showing 10 files: accident0.jpg, accident1.jpg, accident10.jpg, accident100.jpg, accident101.jpg, accident102.jpg, accident103.jpg, accident104.jpg, accident105.jpg, and accident106.jpg, all of which were uploaded on Aug 6, 2025.

Name	Size	Type	Created	Storage
accident0.jpg	28.4 KB	application/octet-stream	Aug 6, 2025, 3:10:10 PM	Standard
accident1.jpg	42.4 KB	application/octet-stream	Aug 6, 2025, 3:10:12 PM	Standard
accident10.jpg	53.5 KB	application/octet-stream	Aug 6, 2025, 3:10:07 PM	Standard
accident100.jpg	188.8 KB	application/octet-stream	Aug 6, 2025, 3:10:18 PM	Standard
accident101.jpg	93.1 KB	application/octet-stream	Aug 6, 2025, 3:10:16 PM	Standard
accident102.jpg	55.6 KB	application/octet-stream	Aug 6, 2025, 3:10:14 PM	Standard
accident103.jpg	52.6 KB	application/octet-stream	Aug 6, 2025, 3:10:16 PM	Standard
accident104.jpg	271.6 KB	application/octet-stream	Aug 6, 2025, 3:10:11 PM	Standard
accident105.jpg	168.2 KB	application/octet-stream	Aug 6, 2025, 3:10:12 PM	Standard
accident106.jpg	122.7 KB	application/octet-stream	Aug 6, 2025, 3:10:14 PM	Standard
accident107.jpg	77.9 KB	application/octet-stream	Aug 6, 2025, 3:10:13 PM	Standard

The screenshot shows the Google Cloud Storage interface. On the left, there's a sidebar with 'Overview', 'Buckets' (which is selected), 'Monitoring', 'Settings', 'Storage Intelligence', 'Insights datasets', and 'Configuration'. The main area is titled 'Bucket details' for 'accident-detection-dataset'. It shows the location as 'us (multiple regions in United States)', storage class as 'Standard', public access as 'Not public', and protection as 'Soft Delete'. Below this, there are tabs for 'Objects', 'Configuration', 'Permissions', 'Protection', 'Lifecycle', 'Observability', 'Inventory Reports', and 'Operations'. Under 'Objects', there's a 'Folder browser' section with a breadcrumb trail: Buckets > accident-detection-dataset > annotations. It includes buttons for 'Create folder', 'Upload', 'Transfer data', and 'Other services'. A filter bar allows filtering by name prefix and type, with options to 'Show Live objects only'. The main list shows two objects: 'accident\_annotations.csv' (191.2 KB, application/octet-stream) and 'accident\_annotations.csv' (191.2 KB, application/octet-stream). The first object is highlighted.

**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/accident0.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

The screenshot shows the Google Cloud Vertex AI Datasets page. On the left, there's a sidebar with sections like Agent Builder, Data, Model development, and Deploy and use. Under Data, 'Datasets' is selected and highlighted with a purple box. In the main area, a table lists datasets. One row is selected, showing details: Name (accident-detection-dataset), ID (1304796495995731968), Status (Ready), Region (us-central1), Type (Image), Items (1,345 images), Last updated (August 6, 2025). A 'Create' button is at the top right of the table.

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

The screenshot shows the 'Import' page for the 'accident-detection-dataset'. The sidebar on the left has 'Datasets' selected. The main area shows an 'Import' tab selected. It includes a summary image showing two red tomatoes with bounding boxes labeled 'TOMATO'. Below the image is a 'Summary' section with text about object detection models. A callout box points to the image with the text 'provide google storage path of CSV annotation.' The 'Select import method' section shows 'Select import files from Cloud Storage' selected. The 'Select import files from Cloud Storage' section includes a 'Import file path' field containing 'gs://accident-detection-dataset/annotation' and a 'Browse' button. A 'Data split' dropdown is set to 'Default'.

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

Labels

All	1,345
Labeled	1,345
Unlabeled	0

Images

Filter	Enter label or property name
Select all	<input type="checkbox"/>
Images ▾	
Accident	882
Normal	463

Related resources

Training jobs and models

accident-detection-dataset Model type: Object detection

Train new model

## 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**

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**

**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**

**Train new model**

- Training method
- Model details
- 3 Training options
- 4 Compute and pricing

Start training
Cancel

---

Goal	Accuracy	Latency
<input checked="" type="radio"/> Higher accuracy (new)	Higher	150ms - 180ms
<input type="radio"/> Higher accuracy	Medium	450ms - 480ms
<input type="radio"/> Faster predictions	Lower	200ms - 240ms

Please note that prediction latency estimates are for guidance only. Actual latency depends on your network connectivity.

#### Incremental training

Incremental training lets you use an existing base model as a starting point to train a new model (rather than training a new model from scratch). [Learn more about incremental training](#)

Enable incremental training

Continue

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

**Train new model**

- Training method
- Model details
- Training options
- 4** Compute and pricing

Enter the maximum number of node hours you want to spend training your model.

You can train for as little as 20 node hours. You may also be eligible to train with free node hours. [Pricing guide](#)

Budget \*

20

Maximum node hours

**Estimated completion:** 2 hours

*Factors like dataset size and evaluation metrics generation can make training take longer than estimated*

**Start training** Cancel

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

The screenshot shows the Google Cloud Vertex AI interface. On the left, a sidebar navigation menu includes sections for Agent builder, Data, Model development (with 'Training' selected), and Deploy and use. The main content area is titled 'Training pipelines' and displays a table of training jobs. One job is highlighted: 'accident-detection-dataset' (ID: 979625638707068928), which is 'Finished' (Status: green checkmark), a 'Training pipeline' (Job type), and 'Image object detection' (Model type). The table also includes columns for Duration, Last updated, Created, Ended, and Labels.

## 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.

The screenshot shows the Google Cloud Vertex AI interface, specifically the 'Endpoints' section. The sidebar shows 'Endpoints' is selected. The main area displays the details of an endpoint named 'accident-detection-endpoint'. Under 'Deployed models', there is a table showing one model deployment: 'accident-detection-dataset (Version 1)' (Model ID: 1445192585990635520, Status: green checkmark, Ready). The table includes columns for Model, ID, Status, Deployment resource pool, Most recent alerts, Monitoring, Traffic split, and Con. A blue button labeled 'Deploy another model' is visible below the table. At the bottom, there is a chart interval selector with options from 1 hour to 30 days, and a section for 'Predictions/second'.

Used the Vertex AI Console to upload test images.

The screenshot shows the Vertex AI interface for the 'accident-detection-dataset' version 1. The left sidebar includes sections for Vertex AI Search, Vector Search, Data (Feature Store, Datasets), Model development (Training, Experiments, Metadata), and Ray on Vertex AI. Under Deploy and use, the Model Registry is selected. The main content area has tabs for Evaluate, Deploy & test (selected), Batch infer, Version details, and Lineage. The Deploy your model section shows an endpoint named 'accident-detection-endpoint' with status Active, 1 model, and deployment resource pool us-central1. The Test your model section displays a preview of a racing track image with a bounding box around a car, and a results table showing two accident detections: Accident 1 (0.884) and Accident 2 (0.077). A 'Terminal' button is also visible.

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

This screenshot is identical to the one above, showing the Vertex AI interface for the 'accident-detection-dataset' version 1. It displays the same deployed endpoint and real-time prediction results for accident detection on a racing track image.

## 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>
```

```
1 package com.video.object.tracking.utility;
2
3 /**
4  * @Author Anil Lalam
5 */
6
7 import java.io.File;
8 import java.io.IOException;
9 import java.io.InputStream;
10 import java.nio.file.Files;
11 import java.util.Base64;
12 import java.util.List;
13 import com.google.api.gax.core.FixedCredentialsProvider;
14 import com.google.auth.oauth2.GoogleCredentials;
15 import com.google.cloud.aiplatform.v1.EndpointName;
16 import com.google.cloud.aiplatform.v1.PredictRequest;
17 import com.google.cloud.aiplatform.v1.PredictResponse;
18 import com.google.cloud.aiplatform.v1.PredictionServiceClient;
19 import com.google.cloud.aiplatform.v1.PredictionServiceSettings;
20 import com.google.protobuf.Struct;
21 import com.google.protobuf.Value;
22
23 public class AccidentDetectionPredictionModel {
24
25     private static PredictionServiceClient predictionServiceClient;
26
27     static {
28         InputStream inputStream = AccidentDetectionPredictionModel.class.getClassLoader()
29             .getResourceAsStream("ServiceAccount.json");
30         if (null != inputStream) {
31             try {
32                 GoogleCredentials googleCredentials = GoogleCredentials.fromStream(inputStream);
33                 PredictionServiceSettings predictionServiceSettings = PredictionServiceSettings.newBuilder()
34                     .setCredentialsProvider(FixedCredentialsProvider.create(googleCredentials)).build();
35                 predictionServiceClient = PredictionServiceClient.create(predictionServiceSettings);
36
37             } catch (IOException e) {
38                 // TODO: handle exception
39                 e.printStackTrace();
39 }
```

```

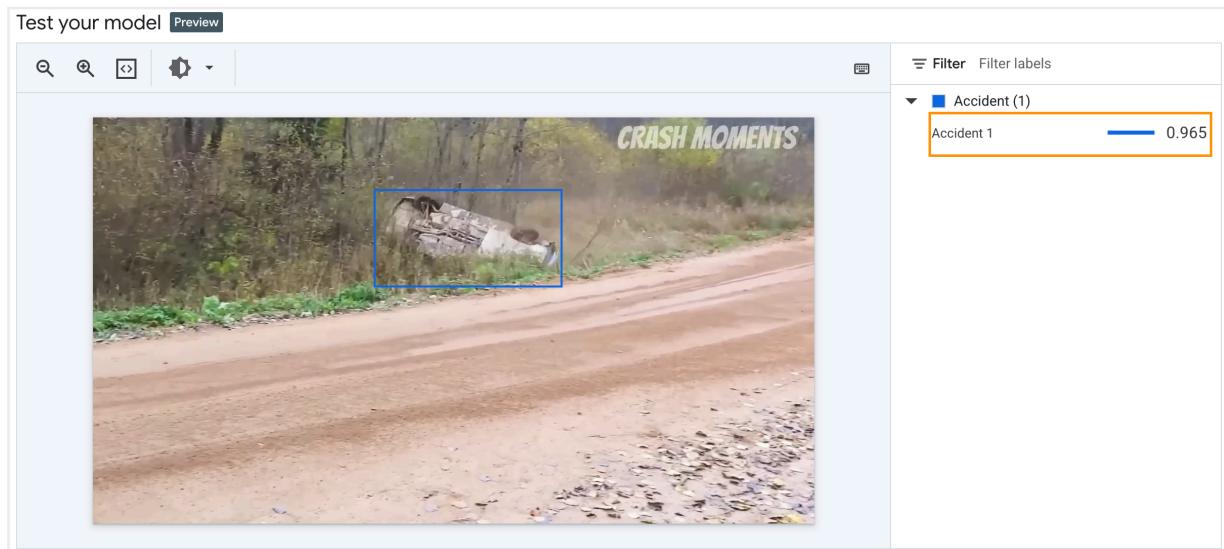
40         }
41     } else {
42         System.out.println("InputStream is null. Please check service Account in src/main/resources");
43     }
44 }
45
46 public static void predictImage(String projectId, String location, String endpointId, String localImagePath) {
47     if (null != predictionServiceClient) {
48         try {
49             // read image as bytes
50             byte[] content = Files.readAllBytes(new File(localImagePath).toPath());
51             String base64Image = Base64.getEncoder().encodeToString(content);
52             // Build instance
53             Struct instance = Struct.newBuilder()
54                 .putFields("content", Value.newBuilder().setStringValue(base64Image).build()).build();
55
56             EndpointName endpointName = EndpointName.of(projectId, location, endpointId);
57             PredictRequest request = PredictRequest.newBuilder().setEndpoint(endpointName.toString())
58                 .addInstances(Value.newBuilder().setStructValue(instance).build()).build();
59             PredictResponse predictResponse = predictionServiceClient.predict(request);
60             System.out.println("Prediction results:");
61             if (null != predictResponse) {
62                 List<Value> predictionList = predictResponse.getPredictionsList();
63                 if (null != predictionList && !predictionList.isEmpty()) {
64                     for (Value prediction : predictionList) {
65                         Struct predictionStruct = prediction.getStructValue();
66                         // labels
67                         List<Value> labels = predictionStruct.getFieldsOrThrow("displayNames").getListValue()
68                             .getValuesList();
69                         // confidence scores
70                         List<Value> scores = predictionStruct.getFieldsOrThrow("confidences").getListValue()
71                             .getValuesList();
72                         // Bounding boxes
73                         List<Value> bboxes = predictionStruct.getFieldsOrThrow("bboxes").getListValue()
74                             .getValuesList();
75
76
77                     for (int i = 0; i < labels.size(); i++) {
78                         String label = labels.get(i).getStringValue();
79                         double score = scores.get(i).getNumberValue();
80                         List<Value> box = bboxes.get(i).getListValue().getValuesList();
81                         double yMin = box.get(0).getNumberValue();
82                         double xMin = box.get(1).getNumberValue();
83                         double yMax = box.get(2).getNumberValue();
84                         double xMax = box.get(3).getNumberValue();
85                         System.out.printf(
86                             "Object: %s, Confidence: %.2f, BBox[yMin=%f, xMin=%f, yMax=%f, xMax=%f]",
87                             label, score, yMin, xMin, yMax, xMax);
88                         System.out.println();
89                     }
90
91                 }
92             } else {
93                 System.out.println("predictionList is null or empty");
94             }
95         } else {
96             System.out.println("predictResponse is null or empty");
97         }
98     }
99
100    }
101
102    } catch (IOException e) {
103        // TODO: handle exception
104        e.printStackTrace();
105    }
106
107    } else {
108        System.out.println("predictionServiceClient is null or empty");
109    }
110
111
112 }
113

```

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

🔍 🔎 ⌂ ⚙️



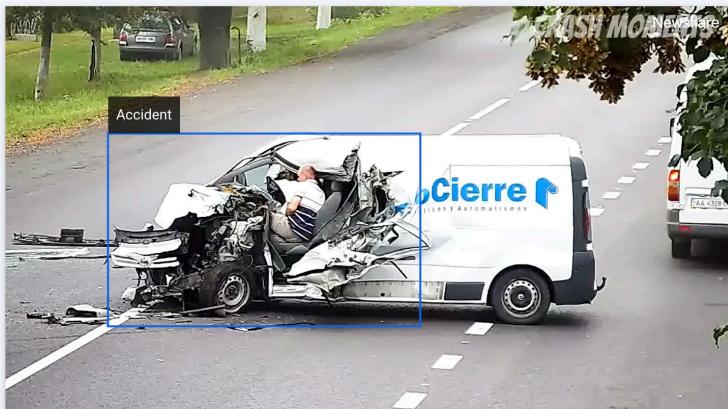
Filter Filter labels

▼ Accident (3)

- |            |       |
|------------|-------|
| Accident 1 | 0.963 |
| Accident 2 | 0.962 |
| Accident 3 | 0.064 |

Test your model [Preview](#)

🔍 🔎 ⌂ ⚙️

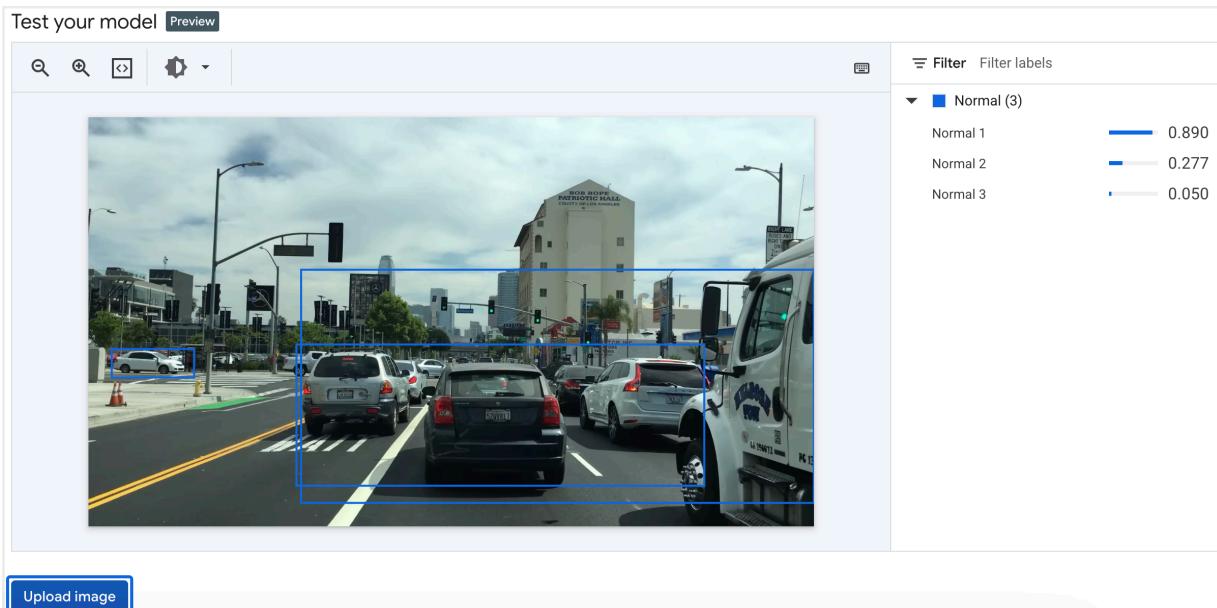


Filter Filter labels

▼ Accident (1)

- |            |       |
|------------|-------|
| Accident 1 | 0.987 |
|------------|-------|

[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**.