The key steps of Machine Learning model development: Example of a Machine Learning model using Contact Center AI (CCAI)

Business goal and machine learning solution

The business goal of this demo is to identify potholes on the road and record their geographical location on the map.

The machine learning solution is to train a vision model (object detection/image classification) to automatically label potholes in each (or every a few) video frames.

Combined with GPS metadata embedded in the video footage, each pothole identified will get its geographical location and be visualised on the map.

Data Exploration

Exploratory Data Analysis was carried out in **Vertex AI Workbench** and descriptive statistics of pothole bounding boxes were generated with customised code and visualisation (python library used: seaborn, matplotlib).

Decision summary:

We chose to train both Image Classification and Object Detection models and select by the performance on the independent testset because of the class imbalance issue we discovered from our dataset. And we also implemented an image masking strategy for pothole label distribution correction based on our analysis. Details are explained in the following sections.

Data overview

The dataset we are using consists of 7 videos of road surveys on city roads with a small number of potholes. Images are sampled every 5 frames from the videos. Camera angle is consistent across the dataset. Example images as shown below (Fig.1):

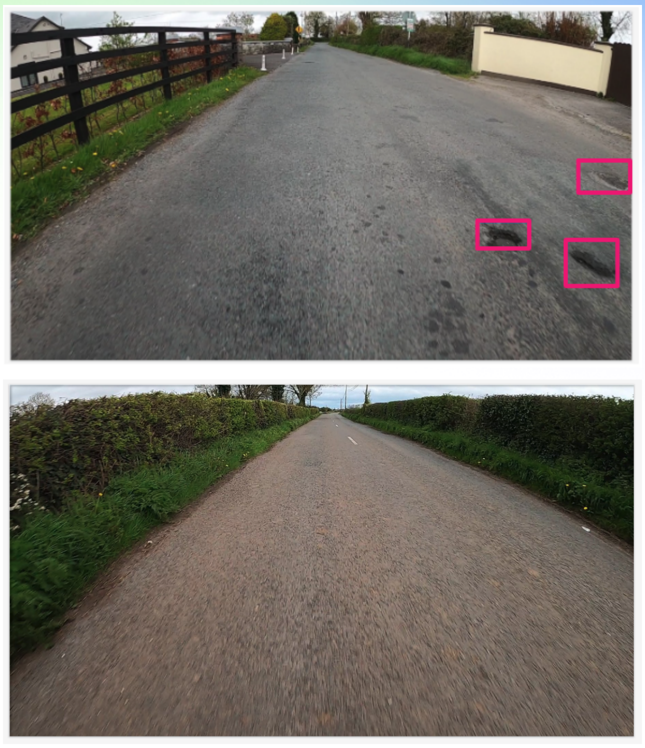


Fig.1 Image with potholes (left) vs image without potholes (right)

Image size is 1920 * 1080 and is consistent across the dataset. The shorter side of the image is only slightly over the resizing threshold (1024) of AutoML image dataset, so no heavy data compression issue is expected. (Reference link).

We select one video as the independent testset, and use the rest as training data.

Pothole occurrence rate

Analysis showed that data is highly imbalanced in terms of pothole-positive video frames vs negative: only 4% of the video frames contain potholes.

Python
```python
# video-image folder list
file_root = 'gcs/footage_2023_clean'
file_path_list = glob.glob(os.path.join(file_root, '*'))

# batch processing: exported dataset -ground truth labels
image_df_list = []
bbox_df_list = []
positive_rate_list = []
num_img_total = 0
num_positive_total = 0

for file_path in file_path_list:
  export_location = glob.glob(os.path.join(file_path, 'export-*'))
```

```
    if export_location:
        label_jsonl = glob.glob(os.path.join(export_location[0],'*','*','data-*.jsonl'))[0]
        image_df, bbox_df = process_df(label_jsonl, file_path)
        positive_rate = len(pd.unique(bbox_df['filename']))/len(image_df)

        image_df_list.append(image_df)
        bbox_df_list.append(bbox_df)
        positive_rate_list.append(positive_rate)

        num_img_total = num_img_total + len(image_df)
        num_positive_total = num_positive_total + len(pd.unique(bbox_df['filename']))

# average positive rate
print('Total number of images: ' + str(num_img_total))
print('Number of images containing potholes: ' + str(num_positive_total))
print('Average pothole-positive rates: ', str(np.mean(np.array(positive_rate_list))))
```

```
Python
### Output ###
Total number of images: 5232
Number of images containing potholes: 233
Average pothole-positive rates:  0.040
```

Although re-sampling methods can help with class imbalance problems, a simple Image Classification model is still very unlikely to perform well because pothole-positive and negative images look almost identical from the camera perspective (Fig.1). Details can be found in the model selection section. On the other hand, 233 images containing potholes is enough for training an AutoML Object Detection model. Pothole detection prediction is easy to convert to image classification prediction as needed. What's more, a pothole detection model provides further opportunities to extract pothole width, depth, severity, etc. information, which are of further business interests. Therefore, we choose to **train both Image Classification and Object Detection models and select by the performance on the independent testset.**

Pothole label distribution

We describe the distribution of Pothole bounding boxes using the following measures:

- Area (bounding box width * height)
- Aspect Ratio (bounding box width / height)
- Location (normalised bounding box centre coordinates)

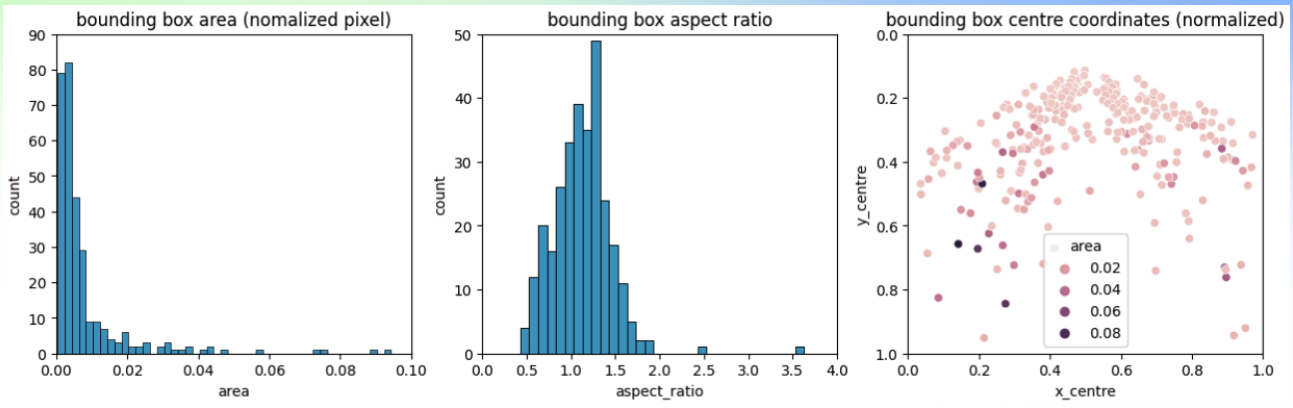


Fig.2 Pothole label distribution

Bounding box Aspect Ratio distribution (Fig.2, middle) is normal while bounding box Area distribution is skewed to the smaller side (Fig.2, left). Smaller objects are generally harder for model training and indicate higher uncertainty in human labelling as well. Moreover, the Location plot (Fig.2, right) indicates that many smaller bounding boxes are further down the road, which should have bigger closer-up views in later video frames. Therefore, we decided to include an **image masking strategy for Area distribution correction** (details see Feature engineering).

Feature engineering

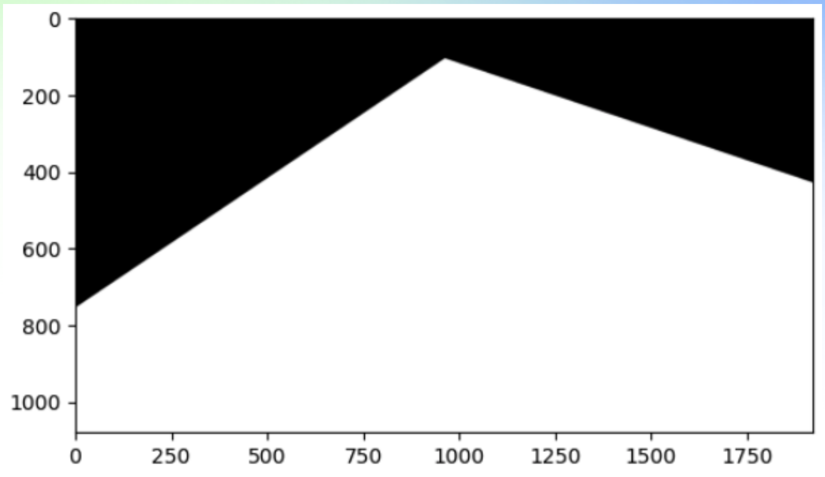
Feature engineering for vision problems

As Convolutional Neural Net (CNN) is an effective feature extractor itself, no specific features need to be prepared like for traditional computer vision models. However, tailored **image and label preprocessing** for our specific problem is crucial for pattern correction and helps to reduce **context-specific "noise"**. In this demo, we developed an image masking strategy according to our dataset.

Road-focus image masking

An image mask as shown below (Fig.3, left) is designed based on the pothole bounding box location distribution (Fig.3, right) and is applied to both training and testing images. This mask helps to block invalid locations for potholes to appear that are otherwise very prone to false positive predictions (Fig.4). Moreover, the mask also blocks locations further down the road that provide smaller-size bounding boxes with higher uncertainty. Any faraway potholes should appear in the unmasked region in a few video frames later. Masking region is customizable for experiment purposes (see code below).

```python
mask = make_mask(l_msk_rate=0.7, t_msk_rate=0.1, r_msk_rate=0.4)
```



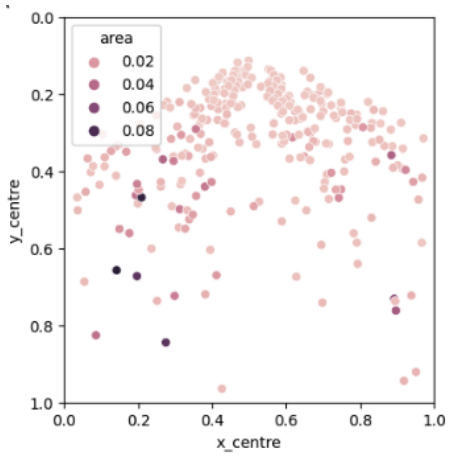


Fig.3 Image Masking Strategy

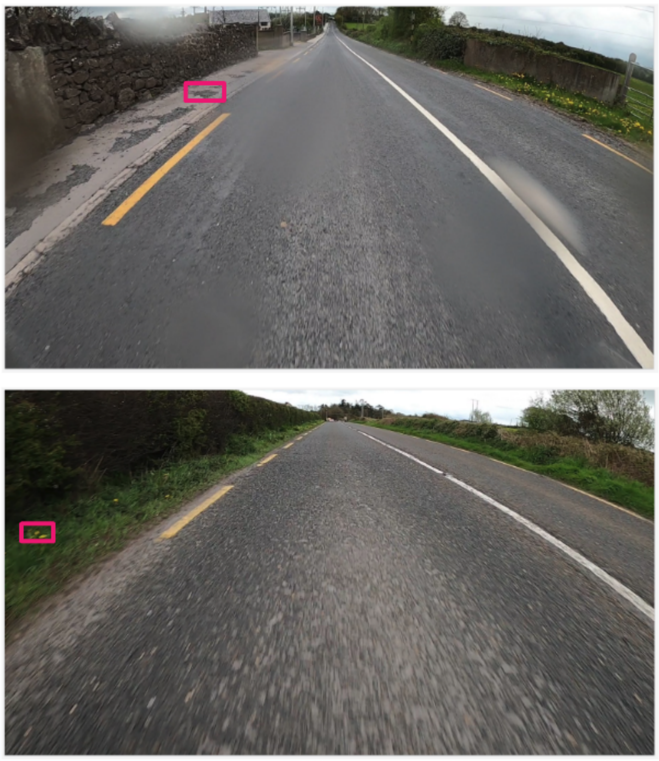


Fig. 4 False positive predictions without masking

Data distribution after masking

Given this very small dataset we have, we adjusted the image mask to keep at least 100 potholes for training. 2 videos with very few/uncertain potholes were excluded from the dataset. As a result, our dataset was reduced to 4578 images with 94 pothole-positive images, occurrence rate 2%.

Small-size pothole labels were reduced after masking (Fig.5).

```
Python
### Output ###
Total number of images: 4578
Number of images containing potholes: 94
Average pothole-positive rates:  0.021
```

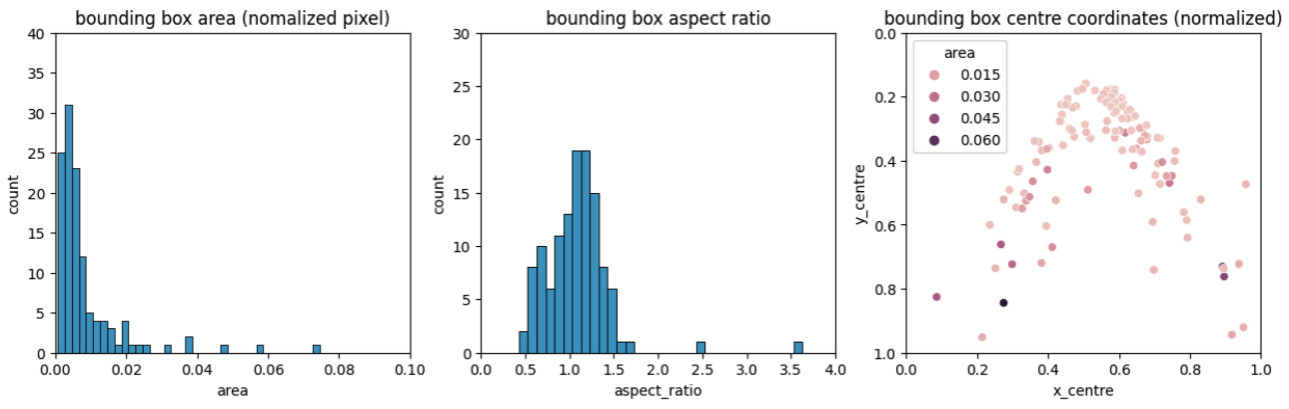


Fig.5 Pothole label distribution after masking

Preprocessing and the data pipeline

The pipeline consists of 4 parts (fig.6):

First, road survey videos are captured by a GoPro device which records GPS information at the same time. Next, videos are dropped into the **GSC bucket** for raw data, which triggers the data preprocessing.

During data preprocessing, videos are first sampled every 5 frames (adjustable) into image series and masked (masking ROI adjustable) to only retain the road surface region. Then each image is sent through a pre-trained Deep Neural Network (DNN) model for Personal Identifying Information (PII) redaction. Specifically, car number plates and pedestrians are detected and blurred. The resulting images are saved in the GCS bucket for clean data.

Finished preprocessing will trigger a batch prediction run on our trained pothole detection model in **Vertex AI** and the results will be written in a **BigQuery** table for the dashboard to pick up.

Finally, the pothole prediction results along with their GPS information will be visualised in **Looker Studio** for the end user for use cases such as pothole repair planning (fig.7).

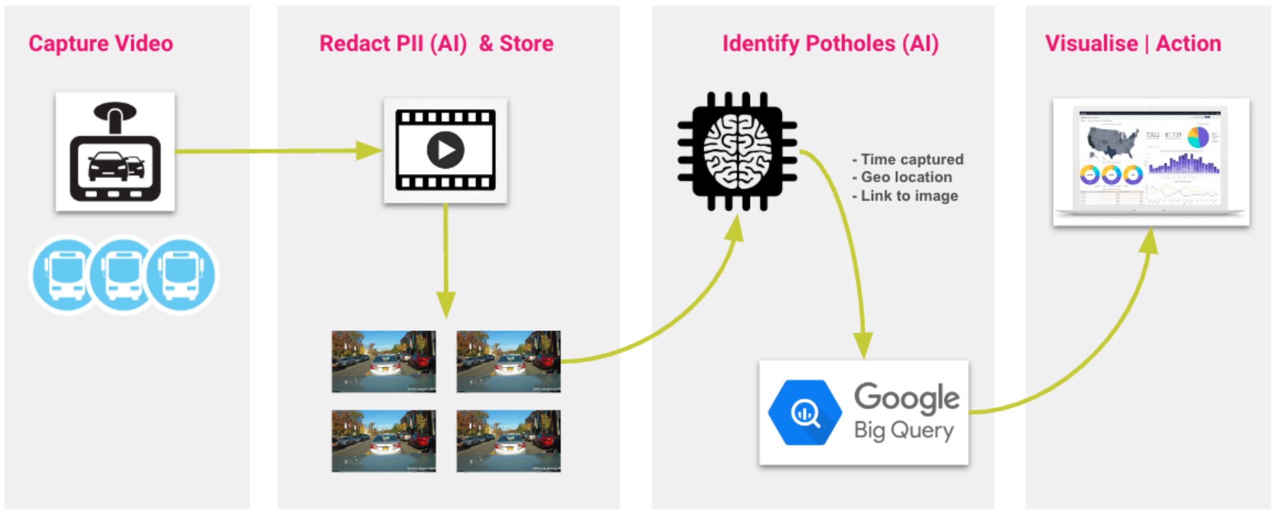
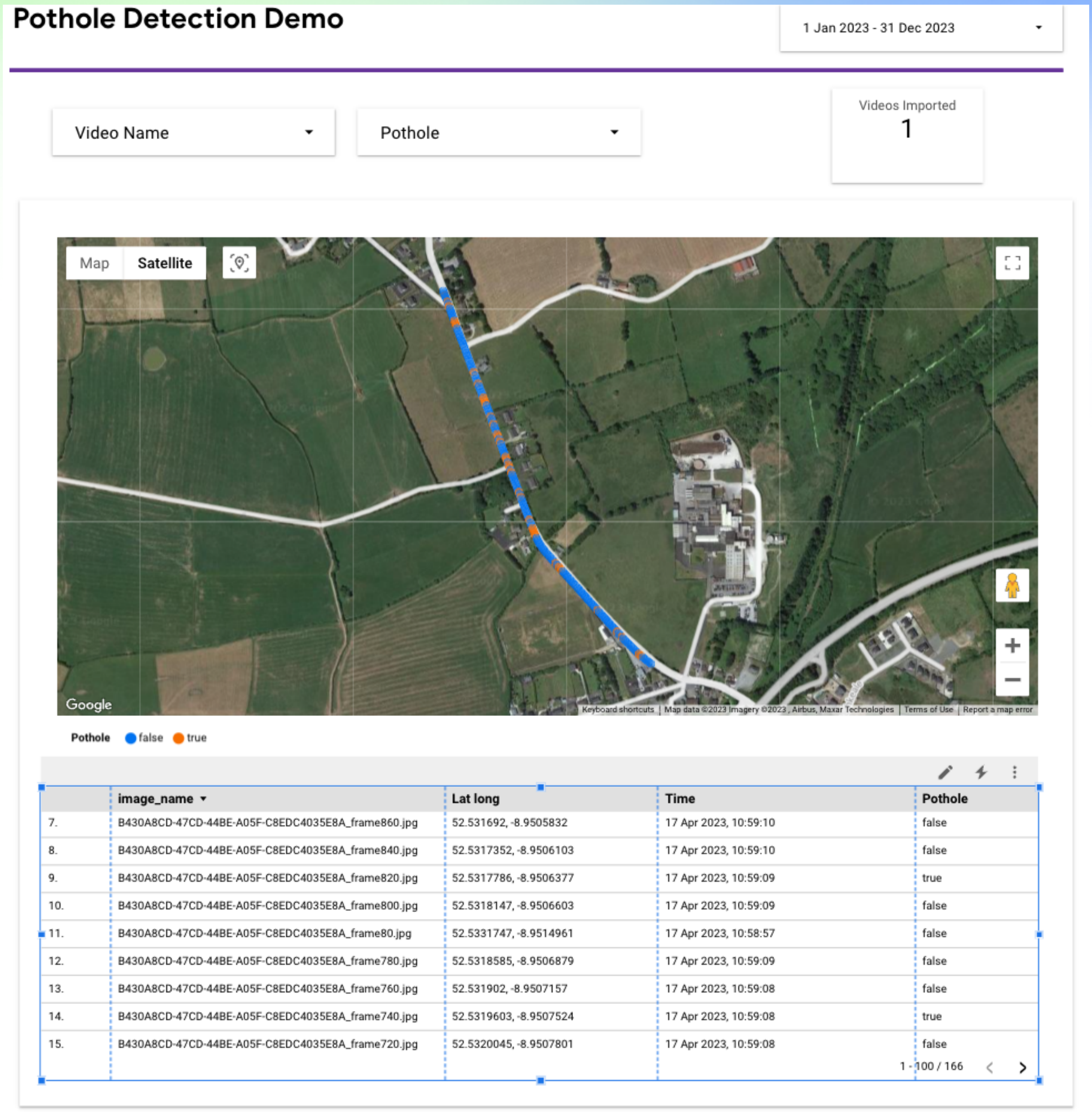


Fig.6 Data pipeline



	image_name ▾	Lat long	Time	Pothole
7.	B430A8CD-47CD-44BE-A05F-C8EDC4035E8A_frame860.jpg	52.531692, -8.9505832	17 Apr 2023, 10:59:10	false
8.	B430A8CD-47CD-44BE-A05F-C8EDC4035E8A_frame840.jpg	52.5317352, -8.9506103	17 Apr 2023, 10:59:10	false
9.	B430A8CD-47CD-44BE-A05F-C8EDC4035E8A_frame820.jpg	52.5317786, -8.9506377	17 Apr 2023, 10:59:09	true
10.	B430A8CD-47CD-44BE-A05F-C8EDC4035E8A_frame800.jpg	52.5318147, -8.9506603	17 Apr 2023, 10:59:09	false
11.	B430A8CD-47CD-44BE-A05F-C8EDC4035E8A_frame80.jpg	52.5331747, -8.9514961	17 Apr 2023, 10:58:57	false
12.	B430A8CD-47CD-44BE-A05F-C8EDC4035E8A_frame780.jpg	52.5318585, -8.9506879	17 Apr 2023, 10:59:09	false
13.	B430A8CD-47CD-44BE-A05F-C8EDC4035E8A_frame760.jpg	52.531902, -8.9507157	17 Apr 2023, 10:59:08	false
14.	B430A8CD-47CD-44BE-A05F-C8EDC4035E8A_frame740.jpg	52.5319603, -8.9507524	17 Apr 2023, 10:59:08	true
15.	B430A8CD-47CD-44BE-A05F-C8EDC4035E8A_frame720.jpg	52.5320045, -8.9507801	17 Apr 2023, 10:59:08	false

1 - 100 / 166

Fig.7 Looker Studio dashboard of pothole detected sites

Code snippet for preprocessing:

```
Python
def main(event, context):
  file_path = event['attributes']['objectId']
  if event['attributes']['eventType'] == 'OBJECT_FINALIZE':
    if file_path.lower().endswith('.mp4') and file_path.split('/')[0] == 'Raw_videos':
          storage_client = storage.Client()
          bigquery_client = bigquery.Client()

          bucket_id =  event['attributes']['bucketId']
          protopath,bucket, proto_name = download_blob(storage_client,
                              bucket_id,
                            "Config_files/MobileNetSSD_deploy.prototxt")
```

```
            modelpath,bucket, model_name = download_blob(storage_client,
                                bucket_id,
                               "Config_files/MobileNetSSD_deploy.caffemodel")

            detector = cv2.dnn.readNetFromCaffe(prototxt=protopath, caffeModel=modelpath)


            CLASSES = ["background", "aeroplane", "bicycle", "bird", "boat",
                   "bottle", "bus", "car", "cat", "chair", "cow", "diningtable",
                   "dog", "horse", "motorbike", "person", "pottedplant", "sheep",
                   "sofa", "train", "tvmonitor"]


            file_location, bucket, video_name = download_blob(storage_client,
                                   bucket_id,
                                event['attributes']['objectId'])

            blurred_bucket = storage_client.get_bucket(bucket_id)
            blurred_bucket_id = "gcs_rapids_clean"
            blurred_bucket = storage_client.get_bucket(blurred_bucket_id)

            # Blurring
            blur_pii(file_location, video_name, detector,CLASSES,storage_client,bigquery_client,
    blurred_bucket)
        else:
            print('Object: {} is not a valid trigger'.format(event['attributes']['objectId']))
      else:
          print('Invalid event type: {}'.format(event['attributes']['eventType']))

    # Blurring pii
    def blur_pii(file_location, video_name,detector,CLASSES,storage_client,bigquery_client,
    blurred_bucket):

    # end of snippet
```

Machine learning model design(s) and selection

Partners must describe either of the following:

Which AutoML product was chosen for demo #3

Evidence must describe (in the whitepaper) selection criteria implemented, as well as the specific machine learning model algorithms that were selected for training or evaluation purposes (as appropriate). Code snippets detailing the incorporation of the pre-trained machine learning APIs or the AutoML product into the machine learning model solution for demo #3 must be enumerated.

Model selection criteria

We choose to experiment with both the **AutoML Image Classification** and **Object Detection models**, and select the one better suited for our problem. Selection criteria is based on the performance on the independent testset, specifically, the **Precision**, **Recall** and **F1 score** for identifying pothole-positive images. **F1 score** is used as the main selection matrix.

For the Object Detection model, we convert the bounding box level prediction into image level prediction by thresholding bounding box confidence score. If the highest bounding box confidence score in one image passes the threshold, that image is predicted as pothole-positive (see code below).

Python
```python
# Thresholding bounding box
pred_positive = []
for i, row in df_pred_all.iterrows():
  if row['prediction.confidences'][0] > threshold:
    filename = os.path.split(row['instance.content'])[-1]
    filename = filename[:10]+'.jpg'
    pred_positive.append(filename)

# Calculate metrics
TP = []
FP = []
FN = []
for img in pred_positive:
  if img in GTP:
    TP.append(img)
  else:
    FP.append(img)
for img in GTP:
  if img not in pred_positive:
    FN.append(img)

precision = len(TP)/len(pred_positive)
recall = len(TP)/len(GTP)
f1 = 2*precision*recall/(precision+recall)
```

Object Detection model out-performs Image Classification model

The independent testset information is summarised in the table below:

Video name	Video length	Images sampled	Pothole-positive images
D6908F35-8B03-417F-AE0E-81EE1263C4CD	0:39	467	12

Precision, recall, and f1 score of the Image Classification model as well as the Object Detection model are listed in the table below:

Machine learning model algorithms	Precision	Recall	F1-score
Object Detection	0.58	0.58	0.58
Image Classification	0.06	1.0	0.12

The Image Classification model failed for two reasons. First, although during training the pothole-negative class was down-sampled to the same number of images as the pothole-positive class, the independent testset was still highly imbalanced as a representative example of the production data. Therefore, the model was heavily biased to the false-positive side, resulting in more than 400 images with prediction confidence above 0.9. In contrast, the Object Detection model achieved a reasonable F1 score (0.58), though false positive was also the issue to be improved. Detailed evaluation of the Object Detection model see section.

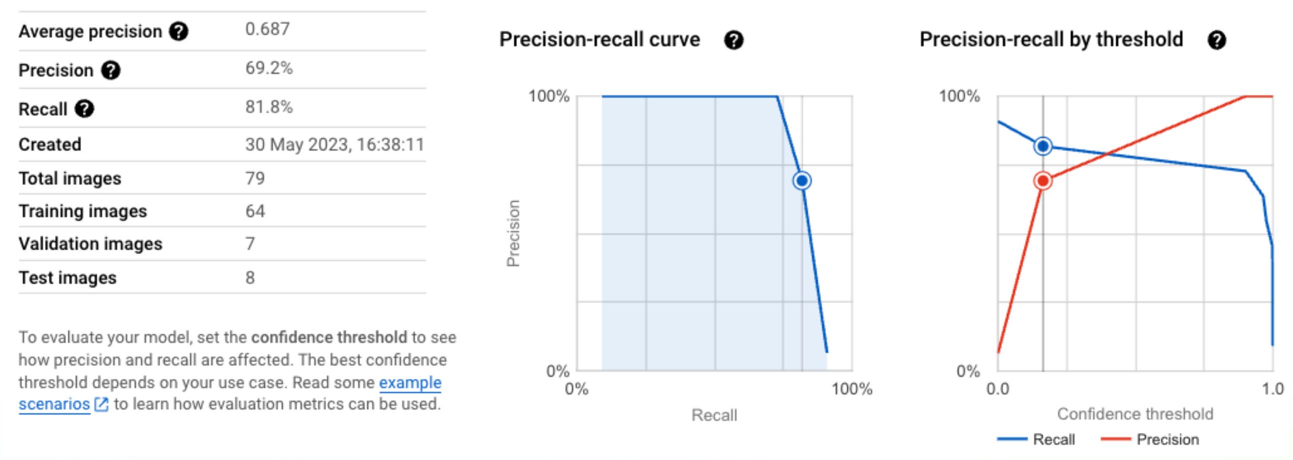
Machine learning model training and development

Data sampling

The independent testset is selected to best represent the production data distribution. The rest of the data is used for training. During training, each image is treated as an independent sample (ie., the same pothole in adjacent video frames can be treated as potholes from different viewpoints). Thus, the sampling method during training was set to default (random).

AutoML training

Training was triggered from UI as well as via API for experiment convenience (code listed below). Training result see fig.8.



Average precision	0.687
Precision	69.2%
Recall	81.8%
Created	30 May 2023, 16:38:11
Total images	79
Training images	64
Validation images	7
Test images	8

To evaluate your model, set the **confidence threshold** to see how precision and recall are affected. The best confidence threshold depends on your use case. Read some example scenarios to learn how evaluation metrics can be used.

Fig.8 Model details

```
Python
def create_training_pipeline_image_object_detection_sample(
    display_name,
    dataset_id,
    model_display_name,
    project = 'rapids-dev',
    location = 'europe-west4',
    api_endpoint = 'europe-west4-aiplatform.googleapis.com',
):
    # The AI Platform services require regional API endpoints.
    client_options = {"api_endpoint": api_endpoint}
    # Initialize client that will be used to create and send requests.
    # This client only needs to be created once, and can be reused for multiple requests.
    client = aiplatform.gapic.PipelineServiceClient(client_options=client_options)
    training_task_inputs = trainingjob.definition.AutoMlImageObjectDetectionInputs(
        model_type="CLOUD_HIGH_ACCURACY_1",
        budget_milli_node_hours=20000,
        disable_early_stopping=False,
    ).to_value()

    training_pipeline = {
        "display_name": display_name,
        "training_task_definition":
"gs://google-cloud-aiplatform/schema/trainingjob/definition/automl_image_object_detection_1.0.0.y
aml",
        "training_task_inputs": training_task_inputs,
        "input_data_config": {"dataset_id": dataset_id},
        "model_to_upload": {"display_name": model_display_name},
    }
    parent = f"projects/{project}/locations/{location}"
    response = client.create_training_pipeline(
        parent=parent, training_pipeline=training_pipeline
    )
    print("response:", response)

# Run training
display_name = 'training_0518'
dataset_id = '1056784605920624640'
model_display_name = 'ds_ex_B43'
create_training_pipeline_image_object_detection_sample(
    display_name,
    dataset_id,
    model_display_name
)
```

Evaluation metric

The model evaluation metric is the same with the selection criteria described above. From the business goal perspective, the question for the current ML solution to address is to identify potholes in video clips and report their GPS locations. GPS metadata was captured and stored per video frame, thus the evaluation metrics are calculated at image-level. Although the Object Detection model outputs prediction at individual pothole level, it is too fine-grained for the business goal. Moreover, due to the shape variety of potholes and viewpoint shifting, traditional IoU (intersection over union) metric could easily reject a prediction that is actually correct in the context (Fig.9). Thus, image-level evaluation is considered a better option.

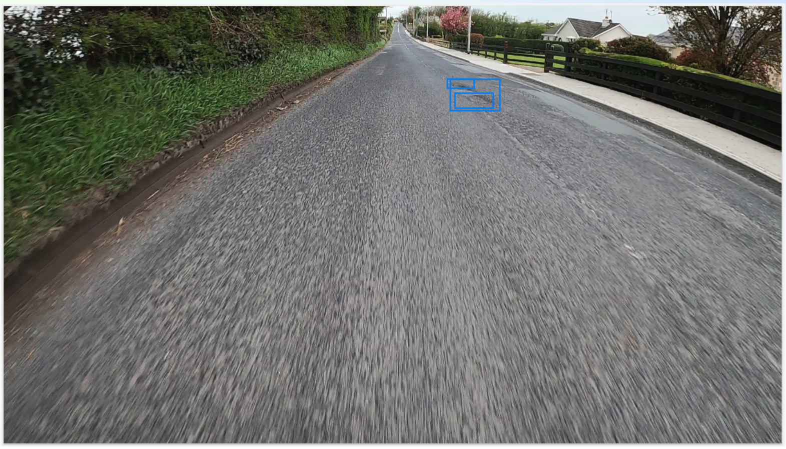


Fig.9 Image-level evaluation.
(Single large bounding box: model prediction; 2 small bounding box: human labels)

Machine learning model evaluation

Partners must describe how the machine learning model, whether implemented using pre-trained machine learning APIs or via AutoML, performs on an independent test dataset.

Evidence must include records/data (in the whitepaper) of how the machine learning model developed and selected to address the business question performed on an independent test dataset that reflects the distribution of data that the machine learning model is expected to encounter in a production environment. In addition, code snippets on model testing need to be enumerated.

Independent testset reflecting production environment

As stated in the data section, all our videos are taken from the actual production environment. We selected a video best representing the overall data distribution, and used the rest as training data so that our testset is also matching the training data (Fig.10).

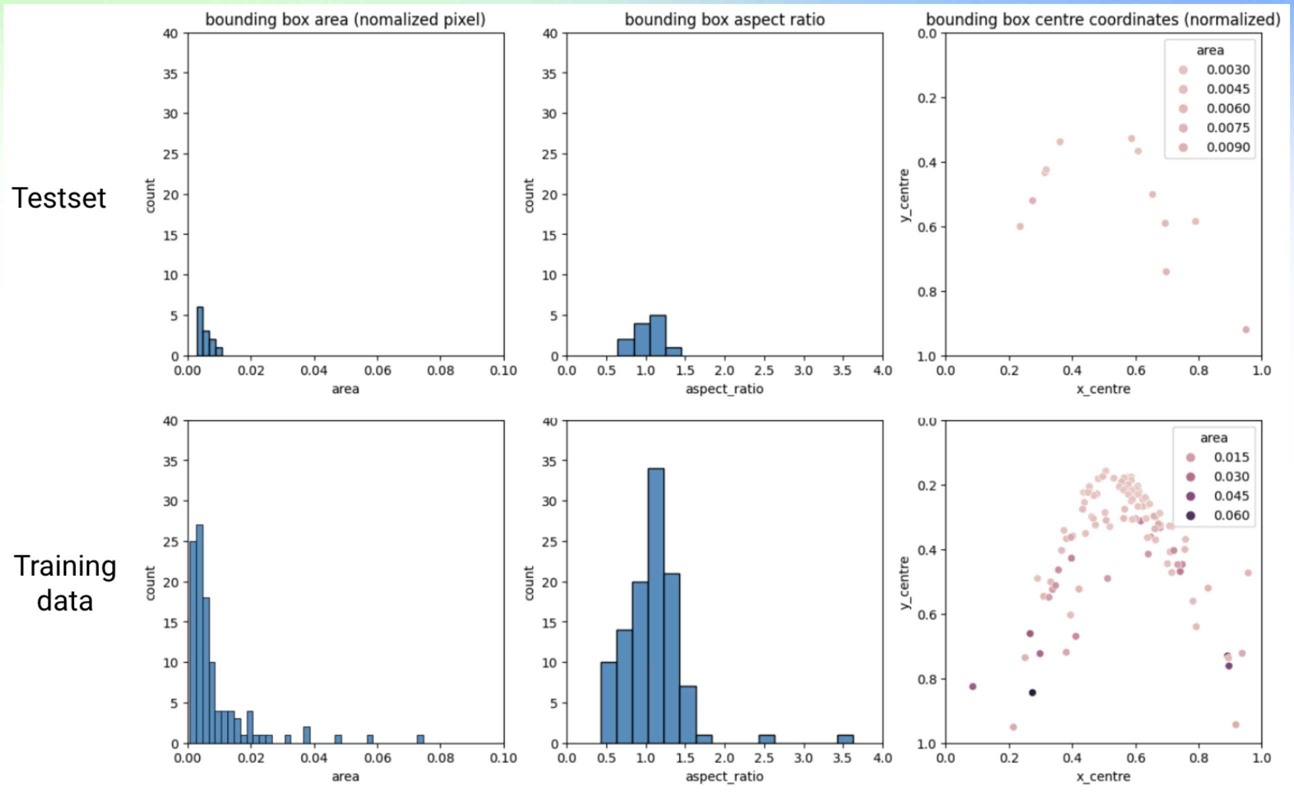


Fig.10 Pothole label distribution: Testset vs Training data

Evaluation on testset

The model achieved 0.58 F1 score on the independent testset, with precision and recall both at 0.58 as well. Video information and metrics are summarised in the tables below. Testing code is listed below.

Given the small size of the training data we have (79 images with 110 pothole labels), the model performed reasonably well.

The major issue with the model is false positive predictions. Currently the model is easy to pick up objects with kind of similar patterns with a pothole as positive (e.g. darker stain, patch of weed, tree shadow, cat's-eye, manhole-cover, etc., see fig.11). These could potentially be improved by adding more training data, especially including negative samples as well. Current false negative predictions (fig.12) are mostly caused by the high confidence threshold set to eliminate as many false positives as possible. Thus reducing false positives should be the future direction. True positives see fig.13.

Video name	Video length	Images sampled	Pothole-positive images
D6908F35-8B03-417F-AE0E-81EE1263C4CD	0:39	467	12

Machine learning model algorithms	Precision	Recall	F1-score
Object Detection	0.58	0.58	0.58

```
Python
threshold = 0.975

print('Number of images tested: ' + str(len(df_pred_all)))
print('Ground truth: ' + str(len(GTP)))

# thresholding bbox: collect img
pred_positive = []
for i, row in df_pred_all.iterrows():
   if row['prediction.confidences'][0] > threshold:
      filename = os.path.split(row['instance.content'])[-1]
      filename = filename[:10]+'.jpg'
      pred_positive.append(filename)

# Calculate metrics
TP = []
FP = []
FN = []
for img in pred_positive:
   if img in GTP:
      TP.append(img)
   else:
      FP.append(img)
for img in GTP:
   if img not in pred_positive:
      FN.append(img)

precision = len(TP)/len(pred_positive)
recall = len(TP)/len(GTP)
f1 = 2*precision*recall/(precision+recall)

print('Detection threshold: ' + str(threshold) + ': ' + str(len(pred_positive)) + ' pothole-positive
images.')
print('Prediction Precision: %.2f' % precision)
print('Prediction Recall: %.2f' % recall)
print('F1 score: %.2f' % f1)
print('')
```

```
Python
# Output

Number of images tested: 467
Ground truth: 12
Detection threshold: 0.975: 12 pothole-positive images.
Prediction Precision: 0.58
Prediction Recall: 0.58
```

F1 score: 0.58

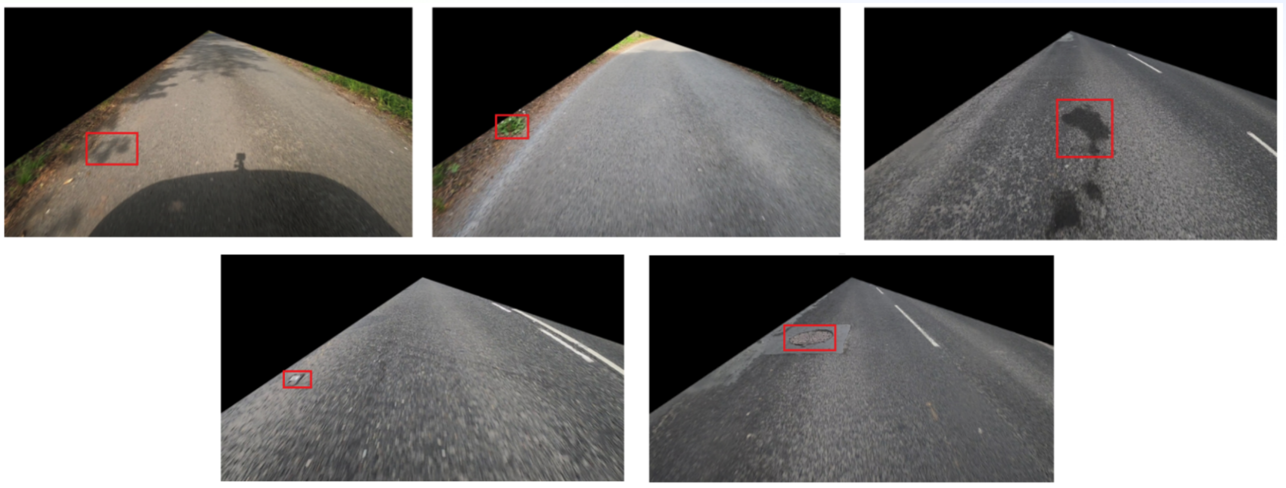


Fig.11 False positive predictions

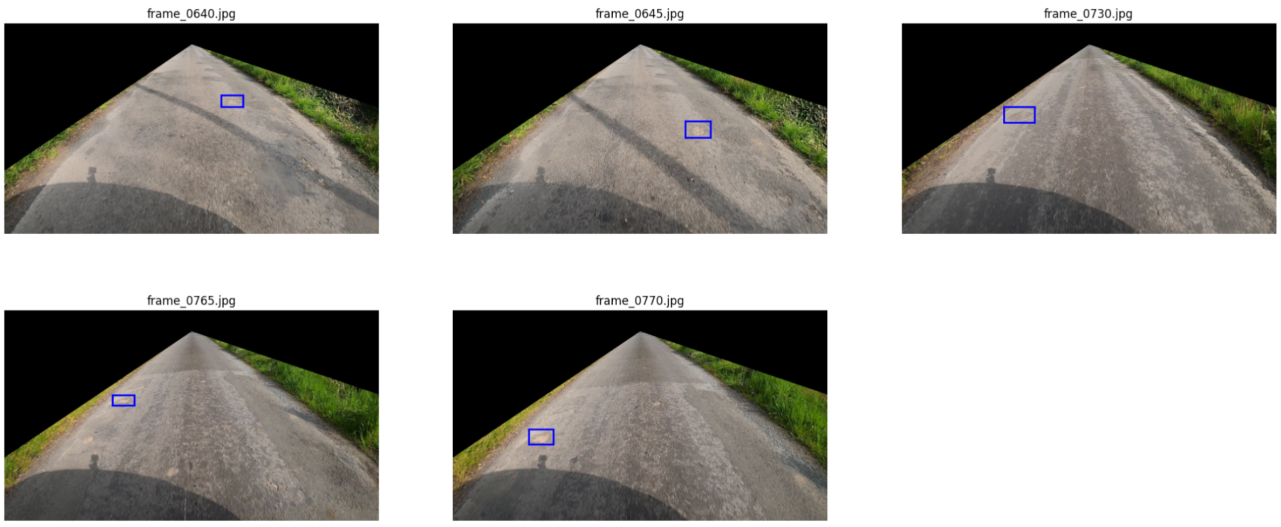


Fig.12 False negative predictions

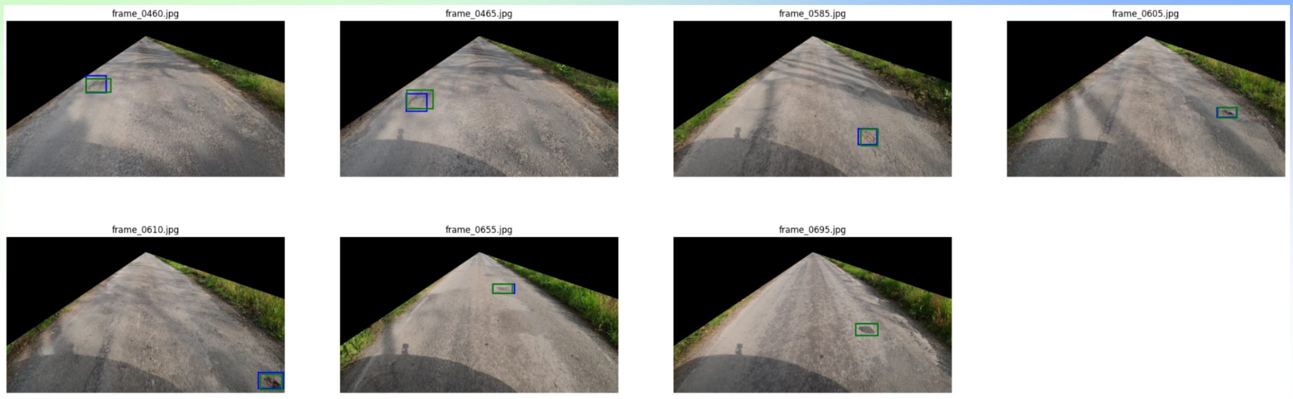


Fig.13 True positive predictions