An ecosystem for evaluating Forensic Delay Analysis

KLINTERAI

Use Case: A Tensor Database for Construction Sector

Describe the use case

The Use case is described as per the document: "An ecosystem for measuring Forensic Delay Analysis.docx"

Design and review the ML pipeline

➢ **Use case requirements 1**: **Delay notification is sent to the Dashboard**

SRC: CCTV Video Images run through Visual Sentiment Analysis

C: IP Cameras, Relevant IoT Data through Messaging Queue installed in Cabin

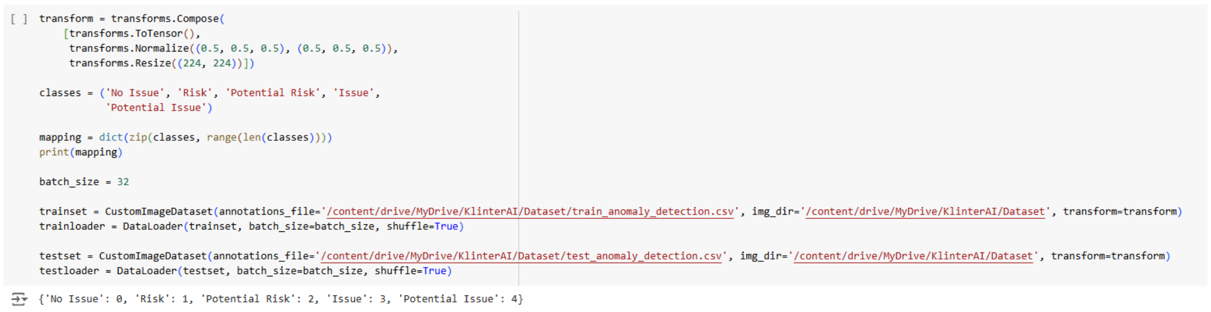
M: Visual Sentiment Analysis Model installed in the Messaging Queue

D: Messaging Queue installed in Cabin connected to Cloud

SINK: The Dashboard which connects to IP Camera(s)

SRC: **Use case requirements 1**

For running Visual Sentiment Analysis, the Dataset is prepared using PyTorch CustomImageDataset. All the 1836 images are loaded onto Google Drive and the Google Drive is force mounted onto Google Colab for Practical Exercise purposes.

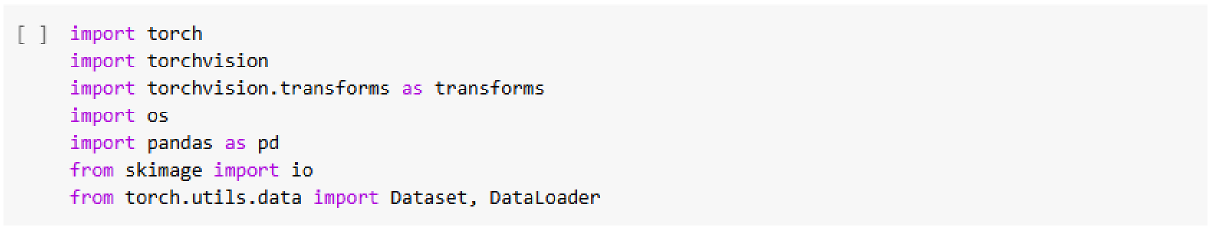


```
[ ] transform = transforms.Compose(
        [transforms.ToTensor(),
         transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
         transforms.Resize((224, 224))])

    classes = ('No Issue', 'Risk', 'Potential Risk', 'Issue',
               'Potential Issue')

    mapping = dict(zip(classes, range(len(classes))))
    print(mapping)

    batch_size = 32

    trainset = CustomImageDataset(annotations_file='/content/drive/MyDrive/KlinterAI/Dataset/train_anomaly_detection.csv', img_dir='/content/drive/MyDrive/KlinterAI/Dataset', transform=transform)
    trainloader = DataLoader(trainset, batch_size=batch_size, shuffle=True)

    testset = CustomImageDataset(annotations_file='/content/drive/MyDrive/KlinterAI/Dataset/test_anomaly_detection.csv', img_dir='/content/drive/MyDrive/KlinterAI/Dataset', transform=transform)
    testloader = DataLoader(testset, batch_size=batch_size, shuffle=True)

    {'No Issue': 0, 'Risk': 1, 'Potential Risk': 2, 'Issue': 3, 'Potential Issue': 4}
```

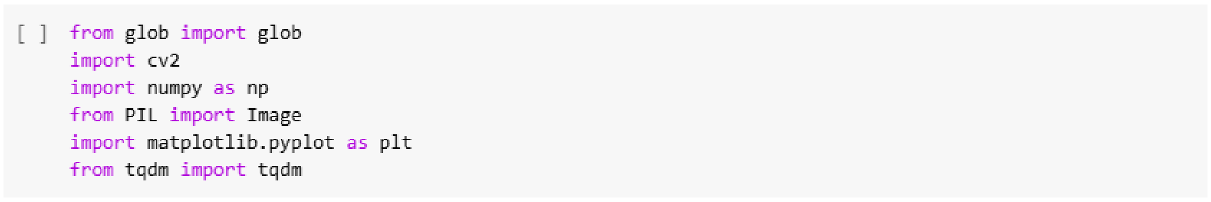
Train / Test Images are split into two annotations file(s): train_anomaly_detection.csv and test_anomaly_detection.csv. A batch size of 32 images is used to load the train / test set. For Image Classification, the classes (categories) used are:

• No Issue	0
• Risk	1
• Potential Risk	2
• Issue	3

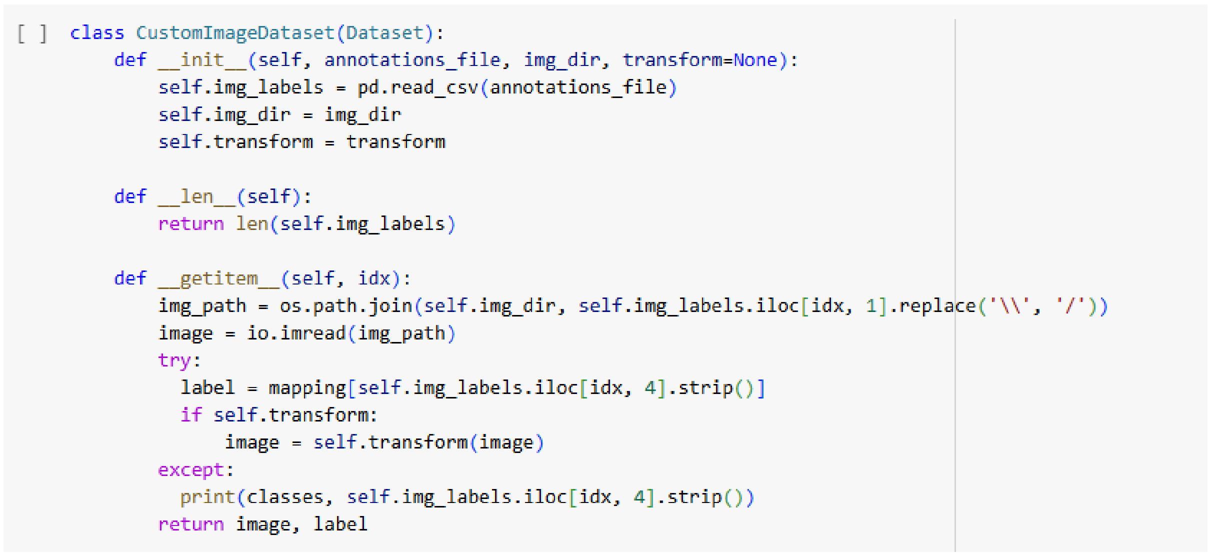
- Potential Issue 4



```
[ ]  import torch
     import torchvision
     import torchvision.transforms as transforms
     import os
     import pandas as pd
     from skimage import io
     from torch.utils.data import Dataset, DataLoader
```



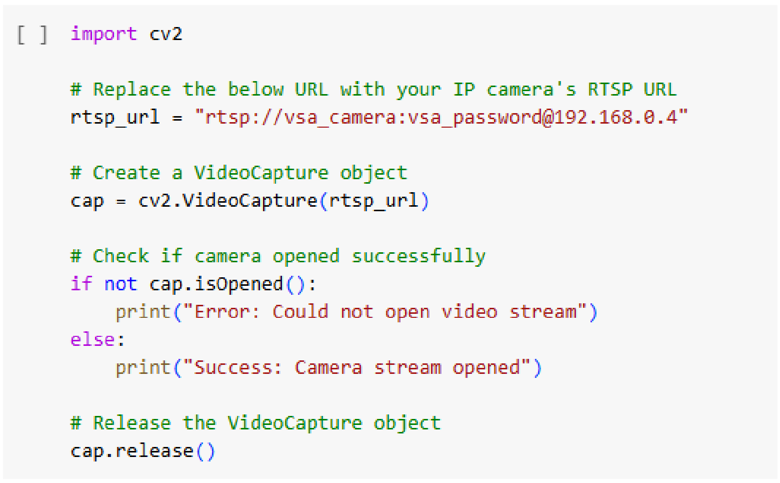
```
[ ]  from glob import glob
     import cv2
     import numpy as np
     from PIL import Image
     import matplotlib.pyplot as plt
     from tqdm import tqdm
```



```
[ ]  class CustomImageDataset(Dataset):
         def __init__(self, annotations_file, img_dir, transform=None):
             self.img_labels = pd.read_csv(annotations_file)
             self.img_dir = img_dir
             self.transform = transform

         def __len__(self):
             return len(self.img_labels)

         def __getitem__(self, idx):
             img_path = os.path.join(self.img_dir, self.img_labels.iloc[idx, 1].replace('\\', '/'))
             image = io.imread(img_path)
             try:
               label = mapping[self.img_labels.iloc[idx, 4].strip()]
               if self.transform:
                   image = self.transform(image)
             except:
               print(classes, self.img_labels.iloc[idx, 4].strip())
             return image, label
```

<u>C (Collector)</u>: <u>Use case requirements 1</u>

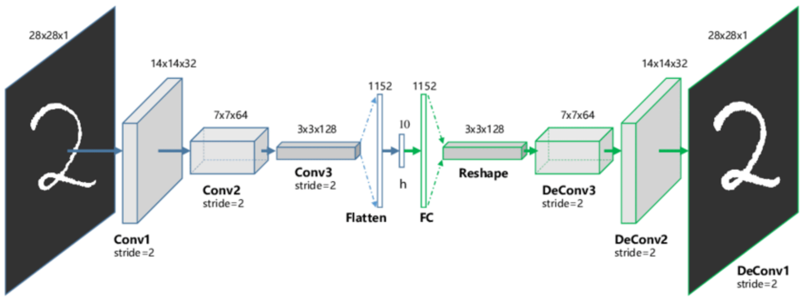
The MQTT Broker installed inside Cabin, with a Raspberry Pi-Like Device, connects to all the IP Cameras through RTSP (Real-Time Streaming Protocol). The MQTT Broker accepts other IoT Devices as well through Wireless technology. The Boilerplate code for accessing IP Cameras through OpenCV software is provided below:

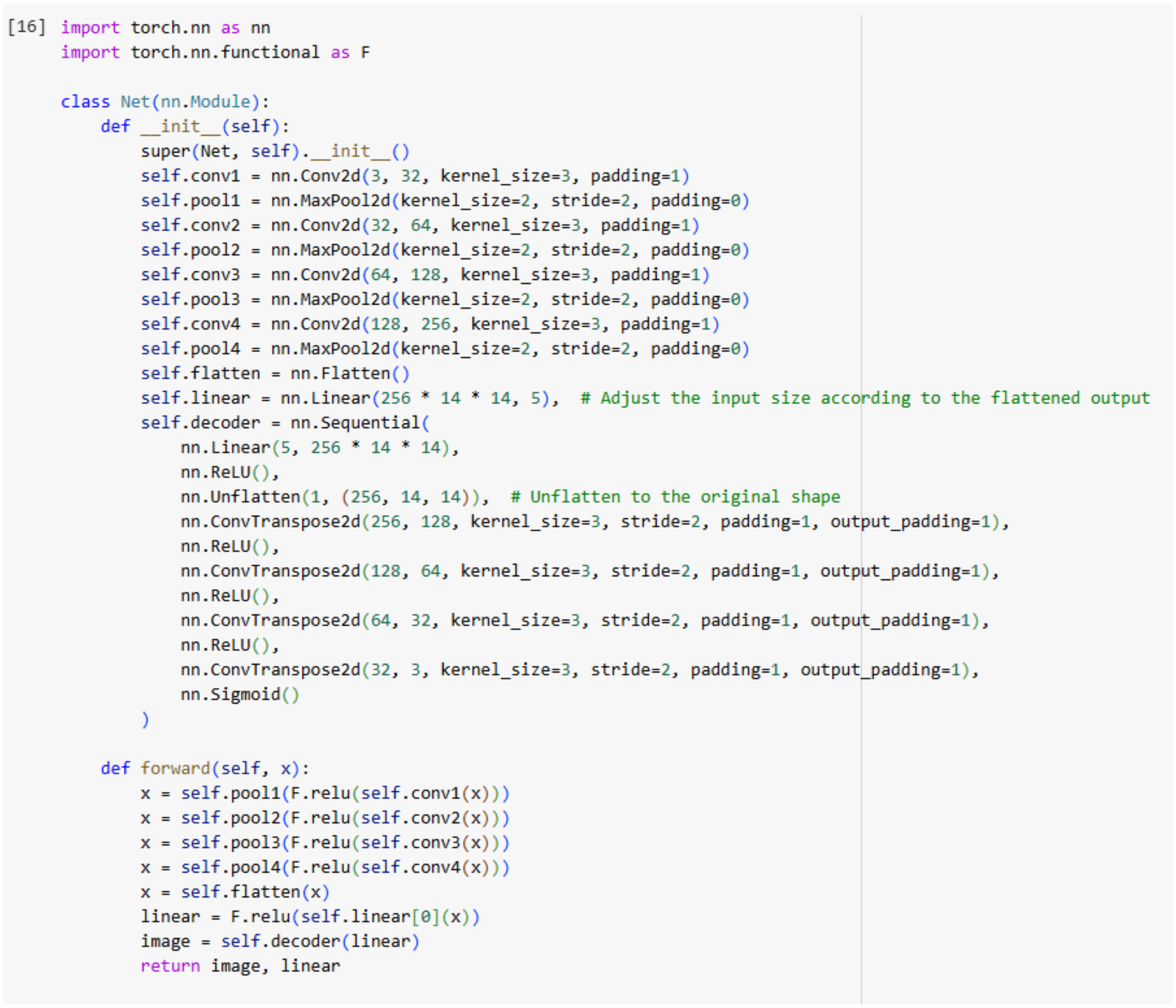


```
[ ]  import cv2

     # Replace the below URL with your IP camera's RTSP URL
     rtsp_url = "rtsp://vsa_camera:vsa_password@192.168.0.4"

     # Create a VideoCapture object
     cap = cv2.VideoCapture(rtsp_url)

     # Check if camera opened successfully
     if not cap.isOpened():
         print("Error: Could not open video stream")
     else:
         print("Success: Camera stream opened")

     # Release the VideoCapture object
     cap.release()
```

<u>M (Model)</u>: <u>Use case requirements 1</u>

The Visual Sentiment Analysis (VSA) Model is installed inside the MQTT Broker and Messaging Queue operated under Raspberry Pi-Like Device. The Visual Sentiment Analysis (VSA) Model is an Autoencoder, a Convolutional Autoencoder with 5 features in the hidden layer specifically designed for image classification.



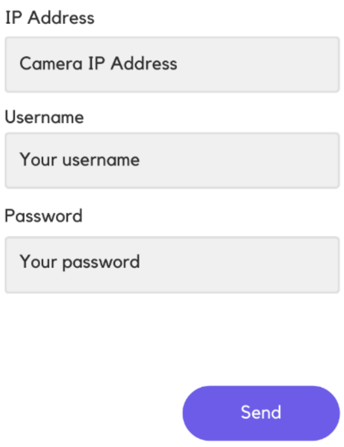


```
[16] import torch.nn as nn
     import torch.nn.functional as F

     class Net(nn.Module):
         def __init__(self):
             super(Net, self).__init__()
             self.conv1 = nn.Conv2d(3, 32, kernel_size=3, padding=1)
             self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)
             self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
             self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)
             self.conv3 = nn.Conv2d(64, 128, kernel_size=3, padding=1)
             self.pool3 = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)
             self.conv4 = nn.Conv2d(128, 256, kernel_size=3, padding=1)
             self.pool4 = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)
             self.flatten = nn.Flatten()
             self.linear = nn.Linear(256 * 14 * 14, 5),  # Adjust the input size according to the flattened output
             self.decoder = nn.Sequential(
                 nn.Linear(5, 256 * 14 * 14),
                 nn.ReLU(),
                 nn.Unflatten(1, (256, 14, 14)),  # Unflatten to the original shape
                 nn.ConvTranspose2d(256, 128, kernel_size=3, stride=2, padding=1, output_padding=1),
                 nn.ReLU(),
                 nn.ConvTranspose2d(128, 64, kernel_size=3, stride=2, padding=1, output_padding=1),
                 nn.ReLU(),
                 nn.ConvTranspose2d(64, 32, kernel_size=3, stride=2, padding=1, output_padding=1),
                 nn.ReLU(),
                 nn.ConvTranspose2d(32, 3, kernel_size=3, stride=2, padding=1, output_padding=1),
                 nn.Sigmoid()
             )

         def forward(self, x):
             x = self.pool1(F.relu(self.conv1(x)))
             x = self.pool2(F.relu(self.conv2(x)))
             x = self.pool3(F.relu(self.conv3(x)))
             x = self.pool4(F.relu(self.conv4(x)))
             x = self.flatten(x)
             linear = F.relu(self.linear[0](x))
             image = self.decoder(linear)
             return image, linear
```

The Model is designed using PyTorch Autoencoder and integrated into the Raspberry Pi-Like Device used as Messaging Queue, MQTT Broker and OpenCV Inference Box from the feed of IP Cameras.

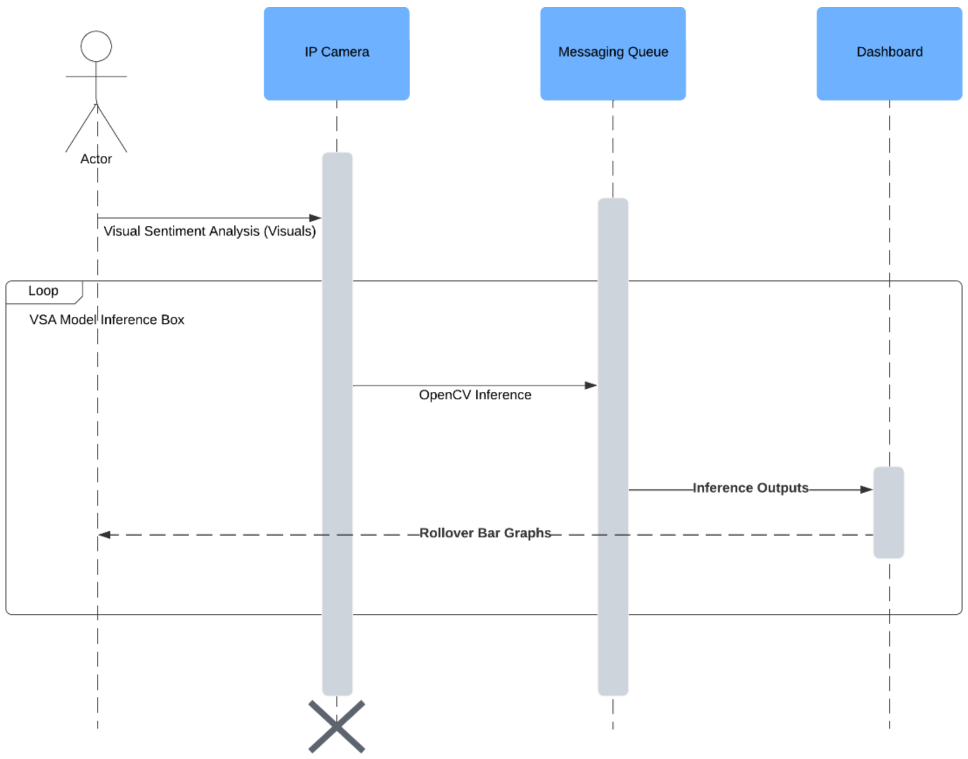
D (Distributor): Use case requirements 1

The Raspberry Pi-Like Device is the Distributor here, sending information aggregated from the IP Cameras to the Cloud Dashboard. The Cloud is connected to the Distributor in a 2-way linking. The RTSP Protocol is instantiated inside the OpenCV Inference Box or the Device using a web form attached to the Cloud Dashboard that send information to the Device attached to the Construction Site Cabin. A Cabin is mandatory for all projects in Construction Industry as per Health & Safety Standards, hence this Inference Box is part of the Cabin.


IP Address
Camera IP Address
Username
Your username
Password
Your password

SINK (Dashboard): Use case requirements 1

The Dashboard is the xApp from ITU.3061 Standards. The Dashboard connects to the Messaging Queue using a Cloud Database readable from the Messaging Queue.



Design and review the ML pipeline

- ➢ **Use case requirements 2: Site Engineer addresses the Risk via Delay Notification Received at Mobile Device**

SRC: CCTV Video Images run through Visual Sentiment Analysis

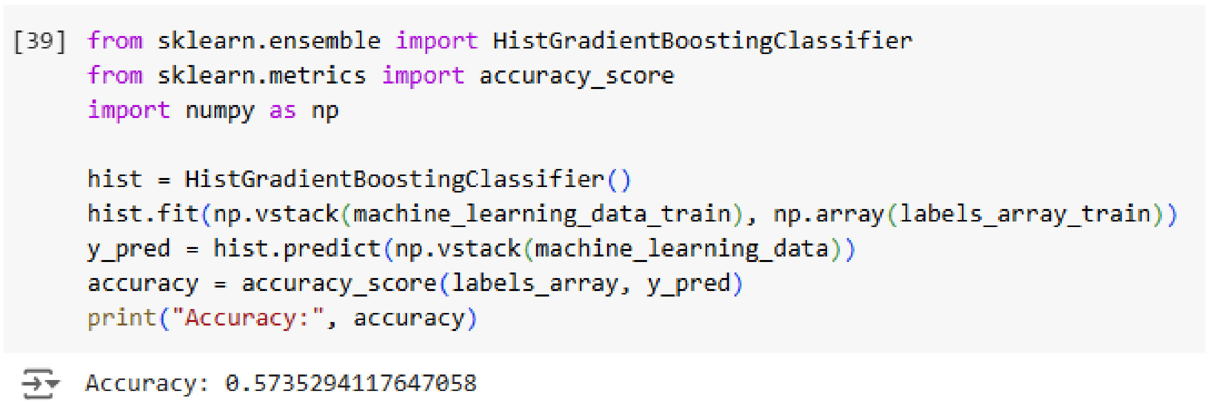
M: Visual Sentiment Analysis Model installed in Cameras or the Messaging Queue

P: A Proper Rollover Metric Policy attached to the Model that signifies a Risk from the Delay in the Presence of an Activity

D: Messaging Queue installed in Cabin

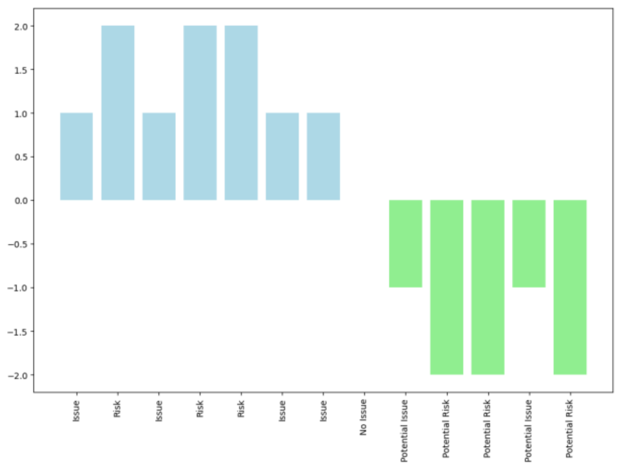
SINK: The EquipAny Mobile App

P (Policy): Use case requirements 2

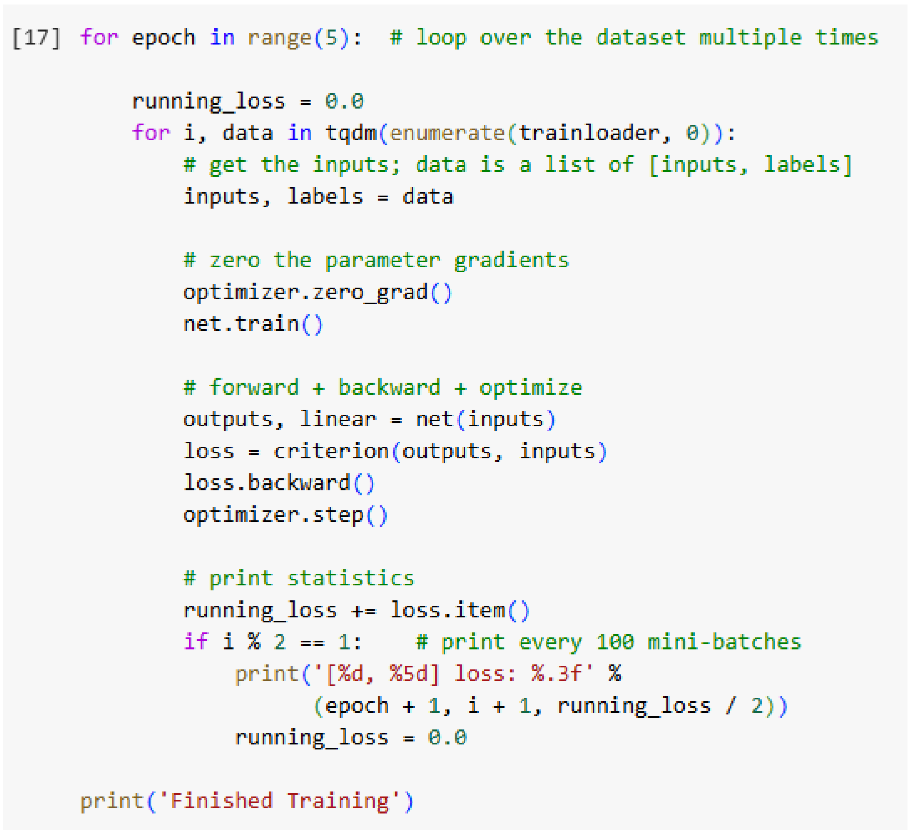


```
[39] from sklearn.ensemble import HistGradientBoostingClassifier
     from sklearn.metrics import accuracy_score
     import numpy as np

     hist = HistGradientBoostingClassifier()
     hist.fit(np.vstack(machine_learning_data_train), np.array(labels_array_train))
     y_pred = hist.predict(np.vstack(machine_learning_data))
     accuracy = accuracy_score(labels_array, y_pred)
     print("Accuracy:", accuracy)

     Accuracy: 0.5735294117647058
```

The output of the Model node in the ML Pipeline is subjected to Risk analysis. Half of the outputs correspond to Risk and Issue and the other half correspond to Potential Risk and Potential Issue. In-effect this is an Anomaly Detection task, which will fetch even more higher accuracy than image classification or sentiment analysis task. The right Policy is to apply for a Rollover metric to reduce the number of frames coming out of the OpenCV Inference Box Video Frames. The Rollover Metric that must be used is Effective Rollover Sum to perform that reduction.



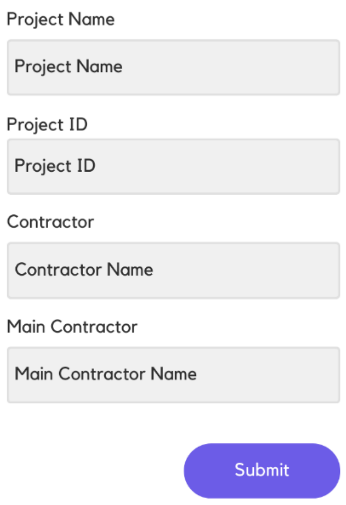
The training loop of the ML Pipeline is provided below. It uses a loss of MSELoss (Mean Squared Error Loss) and uses the feature representation layer to train with labels and produce a minimum accuracy of 0.57.



```
[17] for epoch in range(5):  # loop over the dataset multiple times

         running_loss = 0.0
         for i, data in tqdm(enumerate(trainloader, 0)):
             # get the inputs; data is a list of [inputs, labels]
             inputs, labels = data

             # zero the parameter gradients
             optimizer.zero_grad()
             net.train()

             # forward + backward + optimize
             outputs, linear = net(inputs)
             loss = criterion(outputs, inputs)
             loss.backward()
             optimizer.step()

             # print statistics
             running_loss += loss.item()
             if i % 2 == 1:    # print every 100 mini-batches
                 print('[%d, %5d] loss: %.3f' %
                       (epoch + 1, i + 1, running_loss / 2))
                 running_loss = 0.0

     print('Finished Training')
```

<u>SINK (EquipAny)</u>: <u>Use case requirements 2</u>

The EquipAny mobile app is the rApp of the ITU.3061 Standards. EquipAny is used on demand and does not listen to the Real-Time output or warnings of the Visual Sentiment Analysis (VSA) Model. The EquipAny Mobile App Screen is shown below: to reflect on identifying a Generic Issue as per the Screenshot. EquipAny Mobile App asks the Project Name, Project ID, Contractor, and Main Contractor as the initial set of attributes.



Project Name	Project Name
Project ID	Project ID
Contractor	Contractor Name
Main Contractor	Main Contractor Name

Build the ML Pipeline

The ML Pipeline is built at Github Repository: KlinterAI with link:

https://github.com/KlinterAI/klinterai-website

Demo

Please refer to the Code Walk through Video for the Demo in this section.

https://Klinterai-tenant.vercel.app

https://Klinterai.carrd.co

The above two sites are KlinterAI showcase site and crowdfunding site respectively.