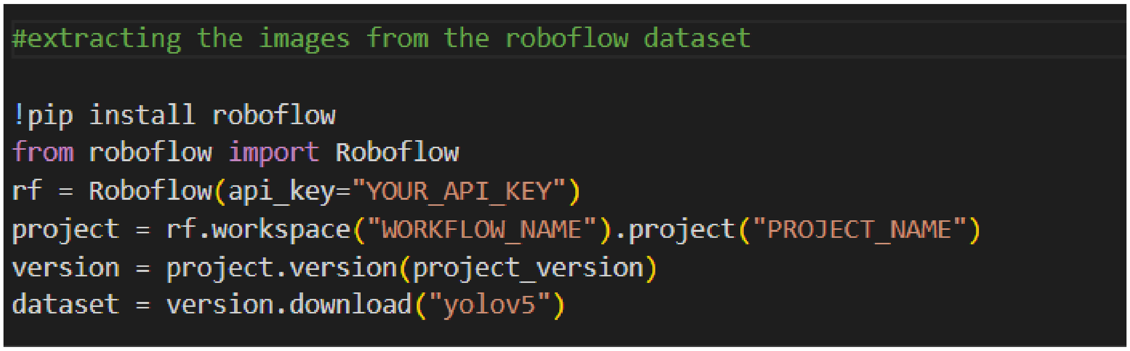
Project Code

[Since our project had to use four models due to space and GPU constraints, the codes given below only contain placeholders for the paths and not the actual paths which are in our project]

Image Extraction:

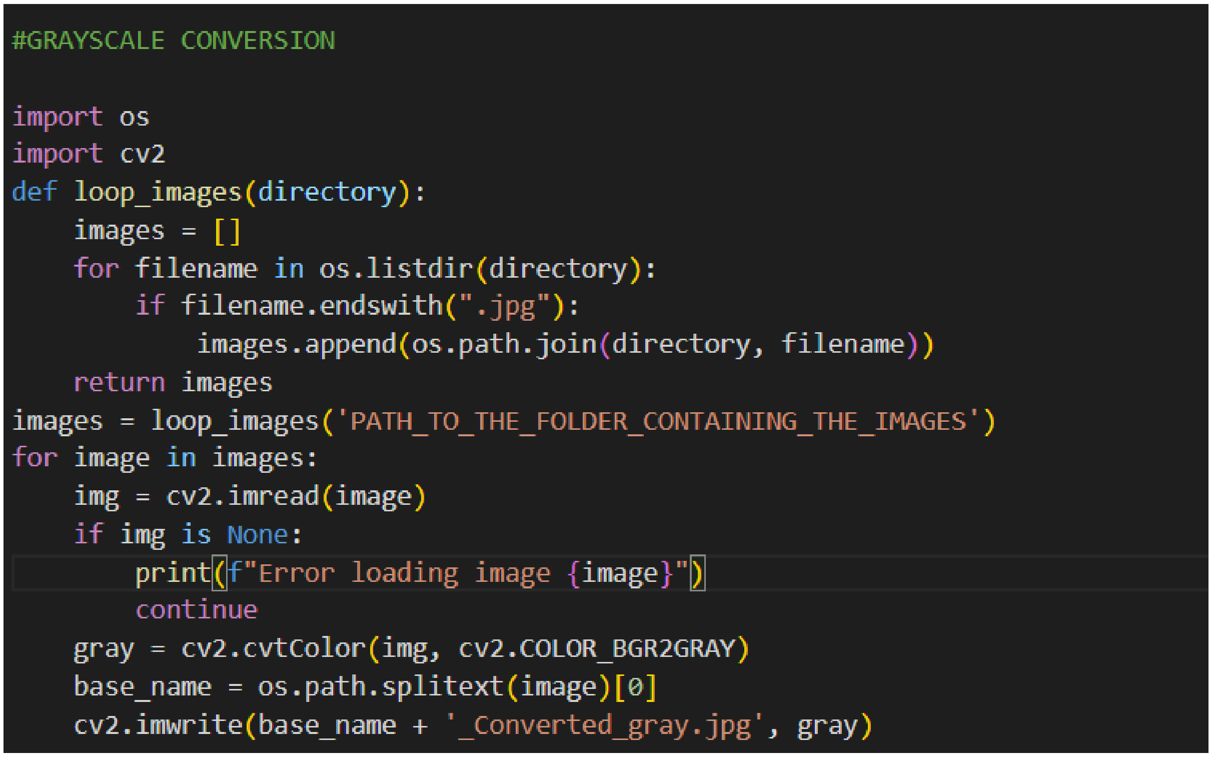
Replace the API Key, workflow, project name and version as per the roboflow generated information:



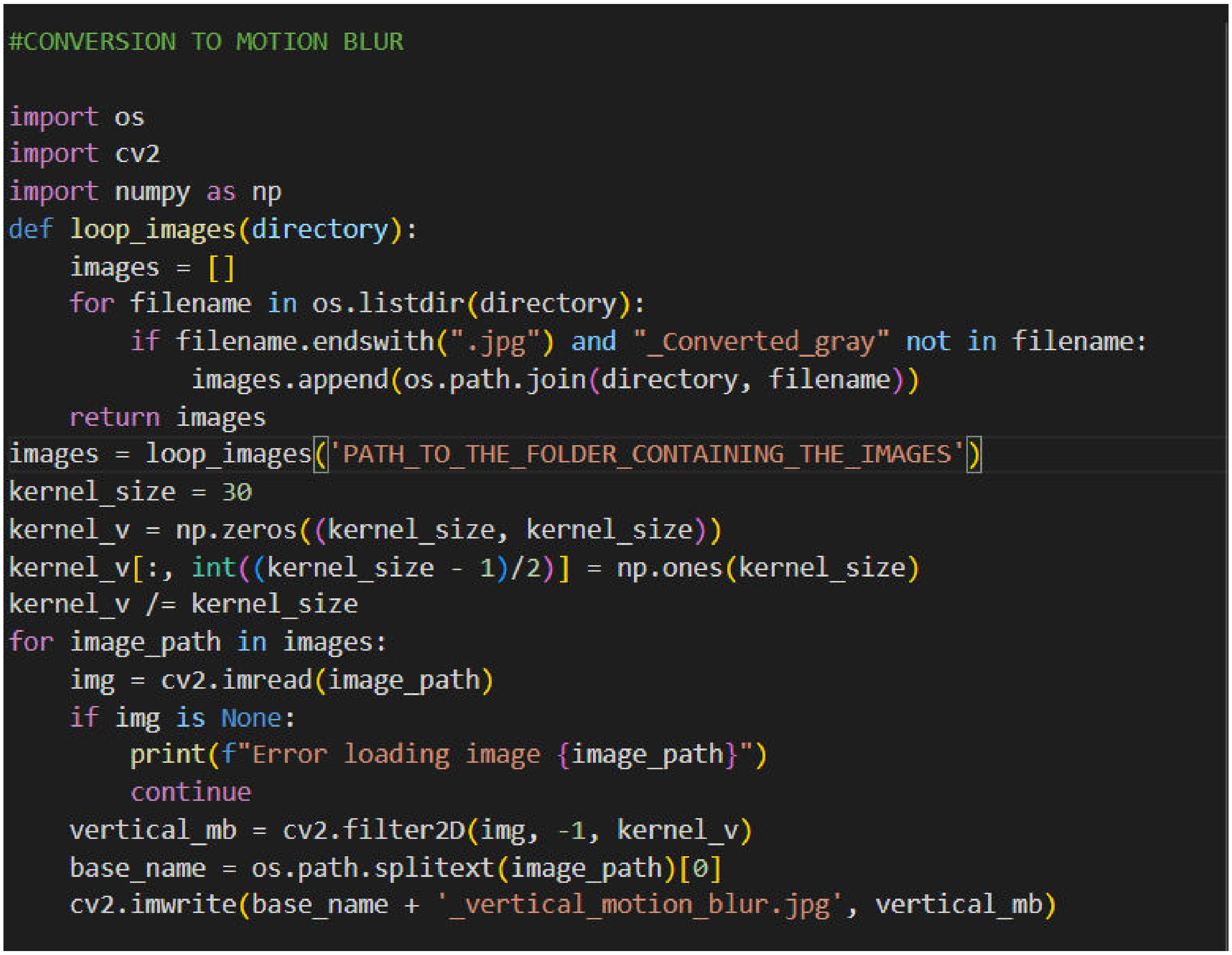
```
#extracting the images from the roboflow dataset

!pip install roboflow
from roboflow import Roboflow
rf = Roboflow(api_key="YOUR_API_KEY")
project = rf.workspace("WORKFLOW_NAME").project("PROJECT_NAME")
version = project.version(project_version)
dataset = version.download("yolov5")
```

Image Augmentation:

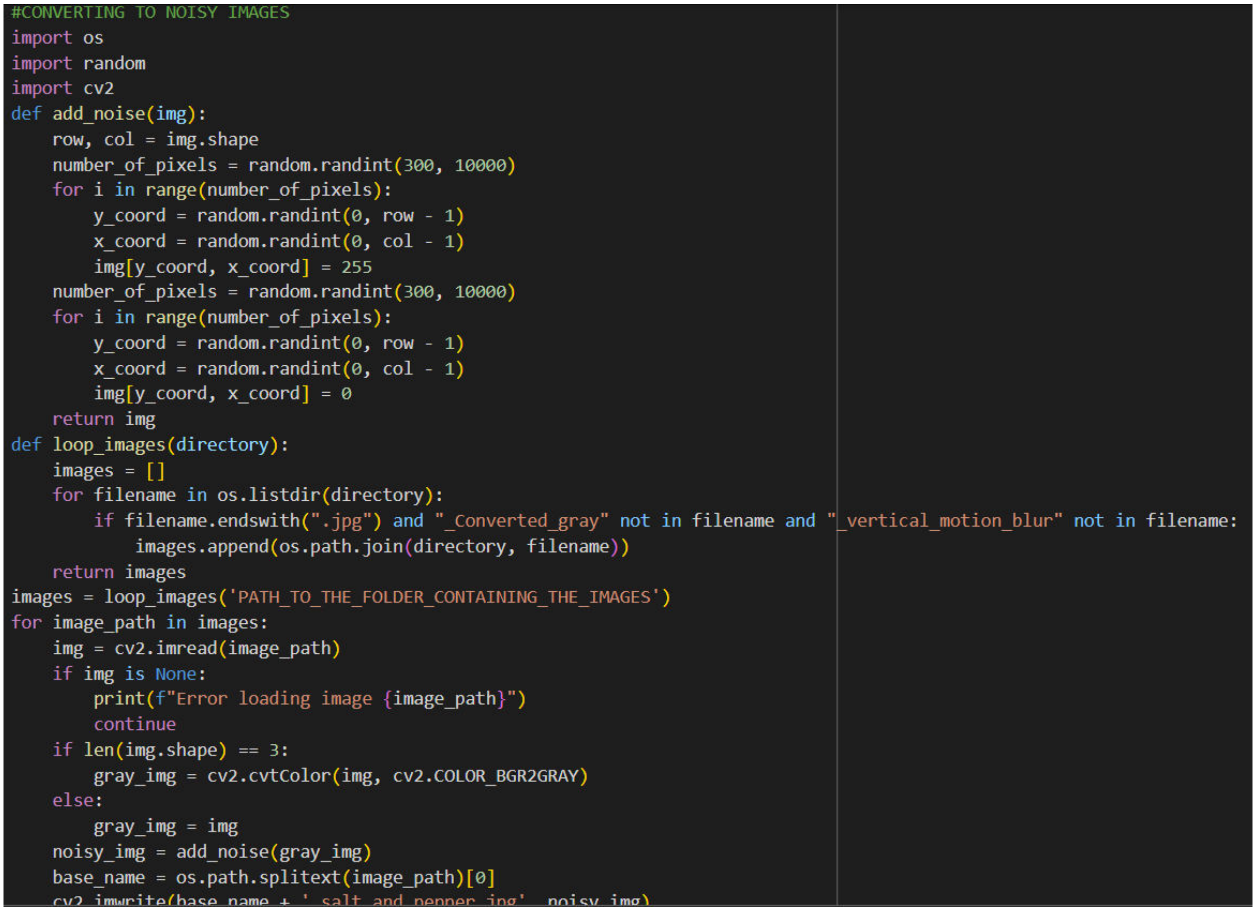
Replace the path for the folder containing the dataset images for each augmentation



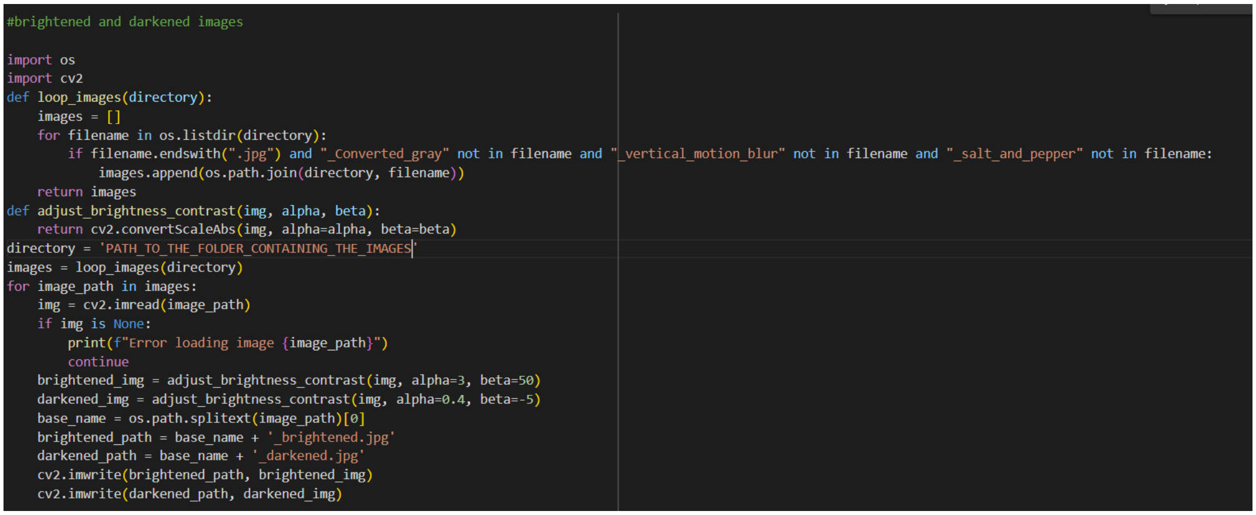
```
#GRAYSCALE CONVERSION

import os
import cv2
def loop_images(directory):
    images = []
    for filename in os.listdir(directory):
        if filename.endswith(".jpg"):
            images.append(os.path.join(directory, filename))
    return images
images = loop_images('PATH_TO_THE_FOLDER_CONTAINING_THE_IMAGES')
for image in images:
    img = cv2.imread(image)
    if img is None:
        print(f"Error loading image {image}")
        continue
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    base_name = os.path.splitext(image)[0]
    cv2.imwrite(base_name + '_Converted_gray.jpg', gray)
```



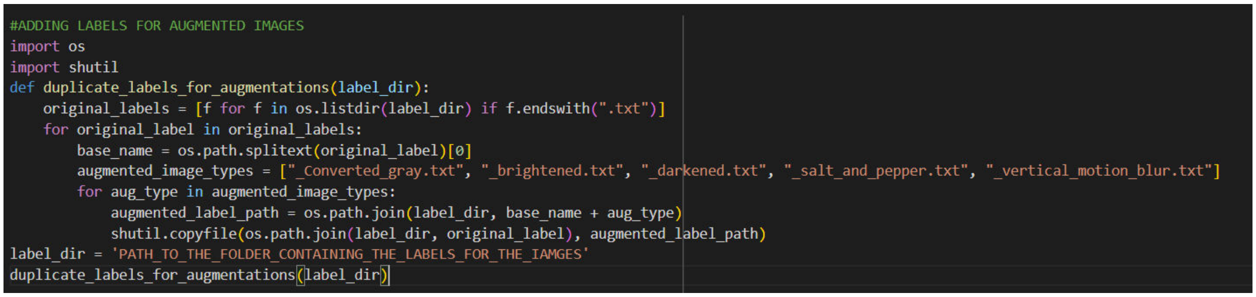
```
#CONVERSION TO MOTION BLUR

import os
import cv2
import numpy as np
def loop_images(directory):
    images = []
    for filename in os.listdir(directory):
        if filename.endswith(".jpg") and "_Converted_gray" not in filename:
            images.append(os.path.join(directory, filename))
    return images
images = loop_images('PATH_TO_THE_FOLDER_CONTAINING_THE_IMAGES')
kernel_size = 30
kernel_v = np.zeros((kernel_size, kernel_size))
kernel_v[:, int((kernel_size - 1)/2)] = np.ones(kernel_size)
kernel_v /= kernel_size
for image_path in images:
    img = cv2.imread(image_path)
    if img is None:
        print(f"Error loading image {image_path}")
        continue
    vertical_mb = cv2.filter2D(img, -1, kernel_v)
    base_name = os.path.splitext(image_path)[0]
    cv2.imwrite(base_name + '_vertical_motion_blur.jpg', vertical_mb)
```



```
#CONVERTING TO NOISY IMAGES
import os
import random
import cv2
def add_noise(img):
    row, col = img.shape
    number_of_pixels = random.randint(300, 10000)
    for i in range(number_of_pixels):
        y_coord = random.randint(0, row - 1)
        x_coord = random.randint(0, col - 1)
        img[y_coord, x_coord] = 255
    number_of_pixels = random.randint(300, 10000)
    for i in range(number_of_pixels):
        y_coord = random.randint(0, row - 1)
        x_coord = random.randint(0, col - 1)
        img[y_coord, x_coord] = 0
    return img
def loop_images(directory):
    images = []
    for filename in os.listdir(directory):
        if filename.endswith(".jpg") and "_Converted_gray" not in filename and "_vertical_motion_blur" not in filename:
            images.append(os.path.join(directory, filename))
    return images
images = loop_images('PATH_TO_THE_FOLDER_CONTAINING_THE_IMAGES')
for image_path in images:
    img = cv2.imread(image_path)
    if img is None:
        print(f"Error loading image {image_path}")
        continue
    if len(img.shape) == 3:
        gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    else:
        gray_img = img
    noisy_img = add_noise(gray_img)
    base_name = os.path.splitext(image_path)[0]
    cv2.imwrite(base_name + '_salt_and_pepper.jpg', noisy_img)
```



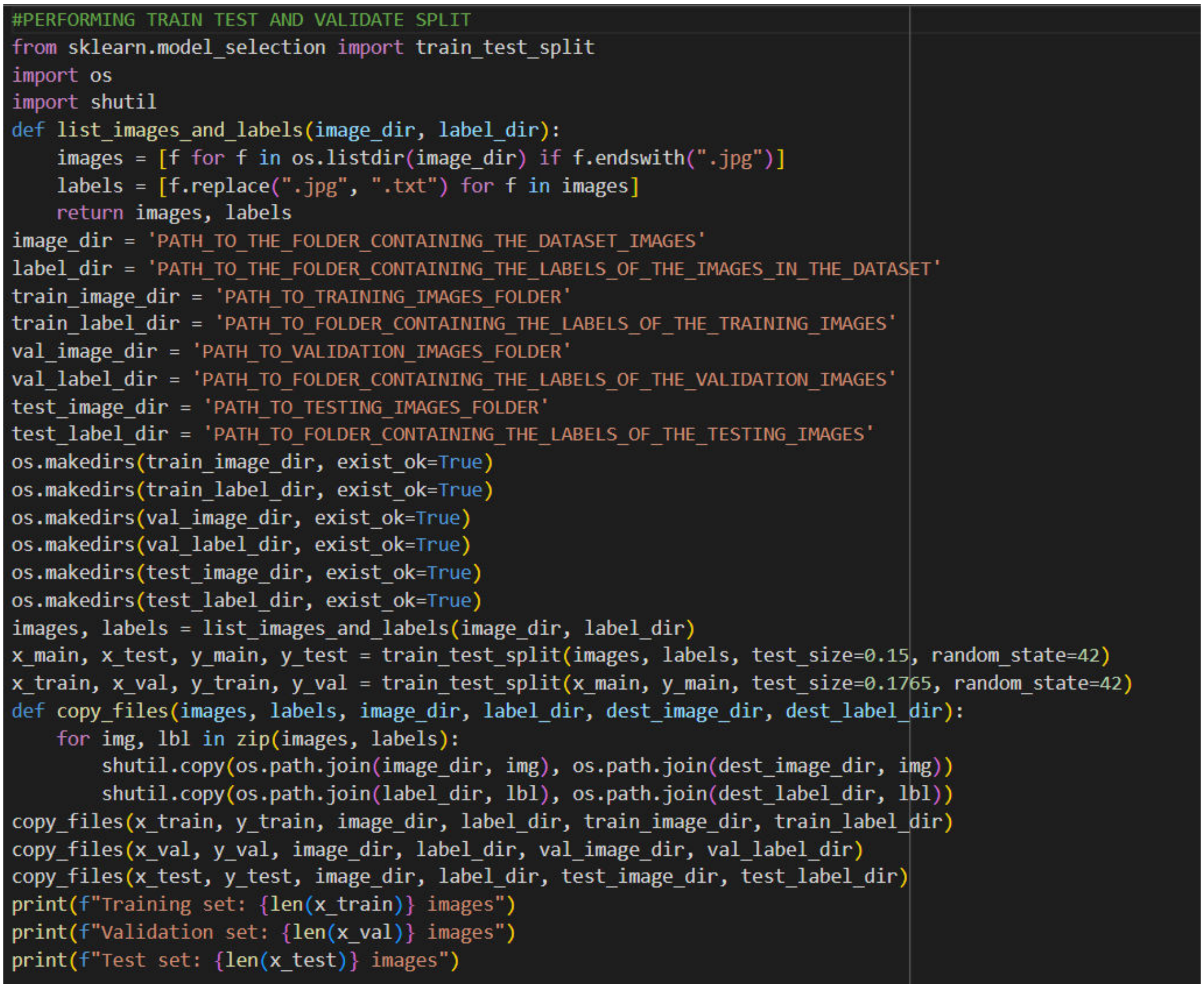
```
#brightened and darkened images

import os
import cv2
def loop_images(directory):
    images = []
    for filename in os.listdir(directory):
        if filename.endswith(".jpg") and "_Converted_gray" not in filename and "_vertical_motion_blur" not in filename and "_salt_and_pepper" not in filename:
            images.append(os.path.join(directory, filename))
    return images
def adjust_brightness_contrast(img, alpha, beta):
    return cv2.convertScaleAbs(img, alpha=alpha, beta=beta)
directory = 'PATH_TO_THE_FOLDER_CONTAINING_THE_IMAGES'
images = loop_images(directory)
for image_path in images:
    img = cv2.imread(image_path)
    if img is None:
        print(f"Error loading image {image_path}")
        continue
    brightened_img = adjust_brightness_contrast(img, alpha=3, beta=50)
    darkened_img = adjust_brightness_contrast(img, alpha=0.4, beta=-5)
    base_name = os.path.splitext(image_path)[0]
    brightened_path = base_name + '_brightened.jpg'
    darkened_path = base_name + '_darkened.jpg'
    cv2.imwrite(brightened_path, brightened_img)
    cv2.imwrite(darkened_path, darkened_img)
```



```
#ADDING LABELS FOR AUGMENTED IMAGES
import os
import shutil
def duplicate_labels_for_augmentations(label_dir):
    original_labels = [f for f in os.listdir(label_dir) if f.endswith(".txt")]
    for original_label in original_labels:
        base_name = os.path.splitext(original_label)[0]
        augmented_image_types = ["_Converted_gray.txt", "_brightened.txt", "_darkened.txt", "_salt_and_pepper.txt", "_vertical_motion_blur.txt"]
        for aug_type in augmented_image_types:
            augmented_label_path = os.path.join(label_dir, base_name + aug_type)
            shutil.copyfile(os.path.join(label_dir, original_label), augmented_label_path)
label_dir = 'PATH_TO_THE_FOLDER_CONTAINING_THE_LABELS_FOR_THE_IAMGES'
duplicate_labels_for_augmentations(label_dir)
```

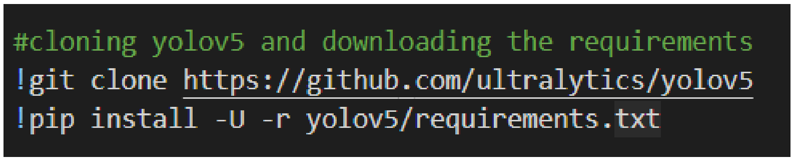
Performing the Train, Test and Validation Split:

Replace Create three new folders Train,Validate and Test each containing two subfolders : 'Images' and 'Labels' and replace the paths for these

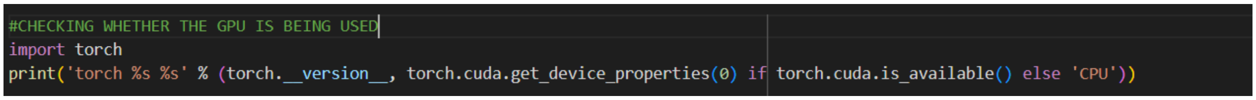


```python
#PERFORMING TRAIN TEST AND VALIDATE SPLIT
from sklearn.model_selection import train_test_split
import os
import shutil
def list_images_and_labels(image_dir, label_dir):
    images = [f for f in os.listdir(image_dir) if f.endswith(".jpg")]
    labels = [f.replace(".jpg", ".txt") for f in images]
    return images, labels
image_dir = 'PATH_TO_THE_FOLDER_CONTAINING_THE_DATASET_IMAGES'
label_dir = 'PATH_TO_THE_FOLDER_CONTAINING_THE_LABELS_OF_THE_IMAGES_IN_THE_DATASET'
train_image_dir = 'PATH_TO_TRAINING_IMAGES_FOLDER'
train_label_dir = 'PATH_TO_FOLDER_CONTAINING_THE_LABELS_OF_THE_TRAINING_IMAGES'
val_image_dir = 'PATH_TO_VALIDATION_IMAGES_FOLDER'
val_label_dir = 'PATH_TO_FOLDER_CONTAINING_THE_LABELS_OF_THE_VALIDATION_IMAGES'
test_image_dir = 'PATH_TO_TESTING_IMAGES_FOLDER'
test_label_dir = 'PATH_TO_FOLDER_CONTAINING_THE_LABELS_OF_THE_TESTING_IMAGES'
os.makedirs(train_image_dir, exist_ok=True)
os.makedirs(train_label_dir, exist_ok=True)
os.makedirs(val_image_dir, exist_ok=True)
os.makedirs(val_label_dir, exist_ok=True)
os.makedirs(test_image_dir, exist_ok=True)
os.makedirs(test_label_dir, exist_ok=True)
images, labels = list_images_and_labels(image_dir, label_dir)
x_main, x_test, y_main, y_test = train_test_split(images, labels, test_size=0.15, random_state=42)
x_train, x_val, y_train, y_val = train_test_split(x_main, y_main, test_size=0.1765, random_state=42)
def copy_files(images, labels, image_dir, label_dir, dest_image_dir, dest_label_dir):
    for img, lbl in zip(images, labels):
        shutil.copy(os.path.join(image_dir, img), os.path.join(dest_image_dir, img))
        shutil.copy(os.path.join(label_dir, lbl), os.path.join(dest_label_dir, lbl))
copy_files(x_train, y_train, image_dir, label_dir, train_image_dir, train_label_dir)
copy_files(x_val, y_val, image_dir, label_dir, val_image_dir, val_label_dir)
copy_files(x_test, y_test, image_dir, label_dir, test_image_dir, test_label_dir)
print(f"Training set: {len(x_train)} images")
print(f"Validation set: {len(x_val)} images")
print(f"Test set: {len(x_test)} images")
```

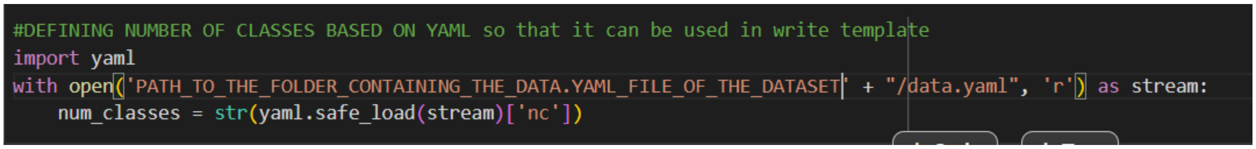
Cloning YOLOv5 and Training it to create a custom model:



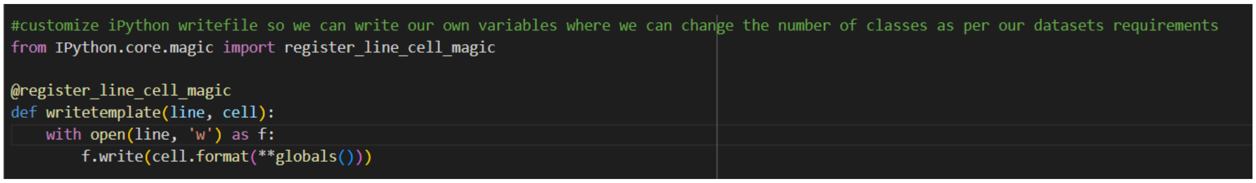
```
#cloning yolov5 and downloading the requirements
!git clone https://github.com/ultralytics/yolov5
!pip install -U -r yolov5/requirements.txt
```



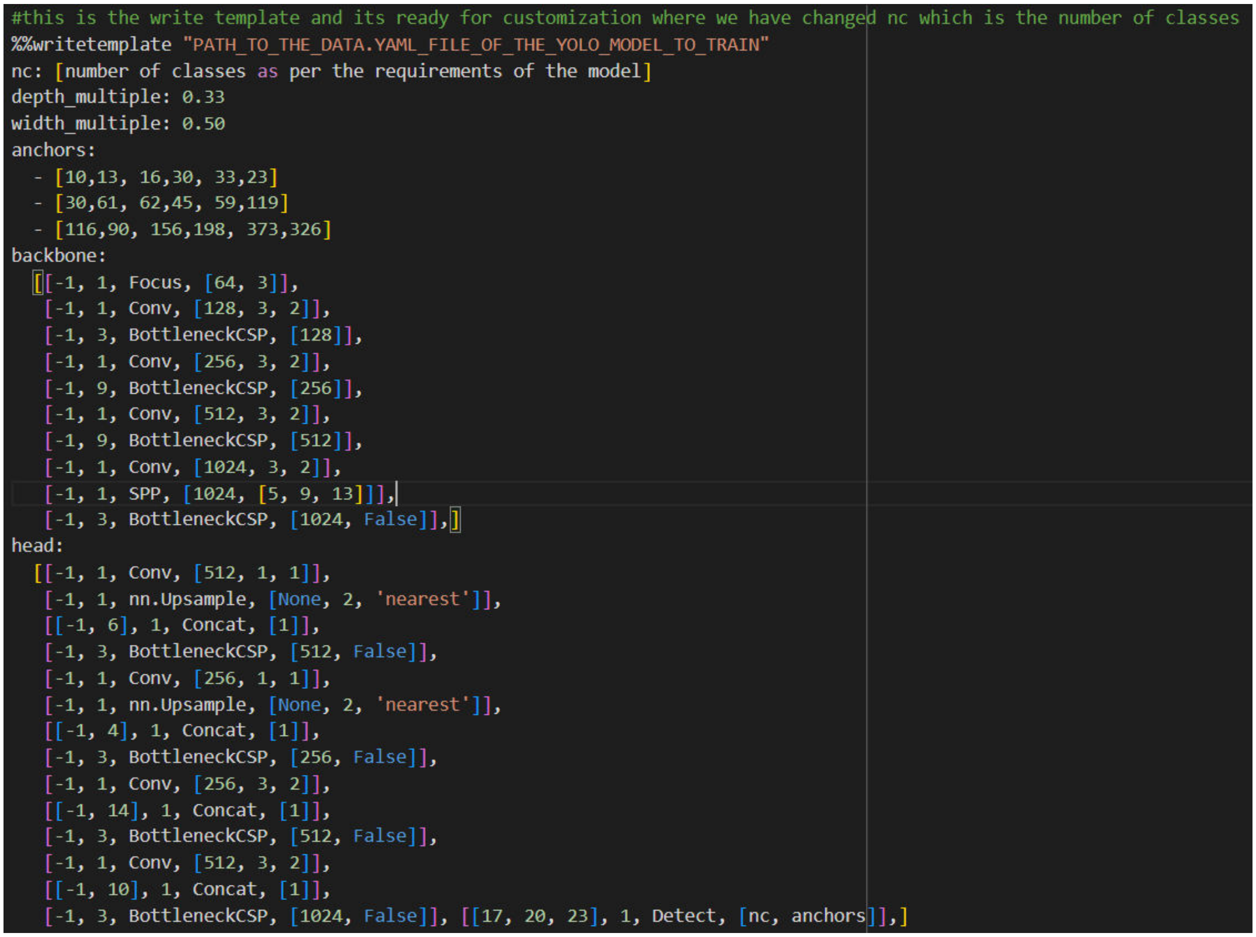
```python
#CHECKING WHETHER THE GPU IS BEING USED
import torch
print('torch %s %s' % (torch.__version__, torch.cuda.get_device_properties(0) if torch.cuda.is_available() else 'CPU'))
```



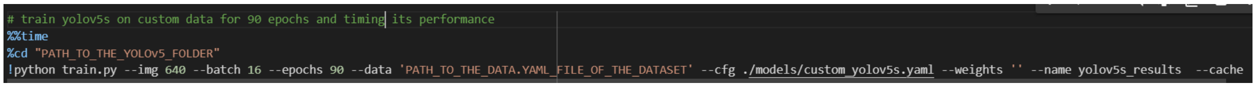
```python
#DEFINING NUMBER OF CLASSES BASED ON YAML so that it can be used in write template
import yaml
with open('PATH_TO_THE_FOLDER_CONTAINING_THE_DATA.YAML_FILE_OF_THE_DATASET' + "/data.yaml", 'r') as stream:
    num_classes = str(yaml.safe_load(stream)['nc'])
```



```
#customize iPython writefile so we can write our own variables where we can change the number of classes as per our datasets requirements
from IPython.core.magic import register_line_cell_magic

@register_line_cell_magic
def writetemplate(line, cell):
    with open(line, 'w') as f:
        f.write(cell.format(**globals()))
```

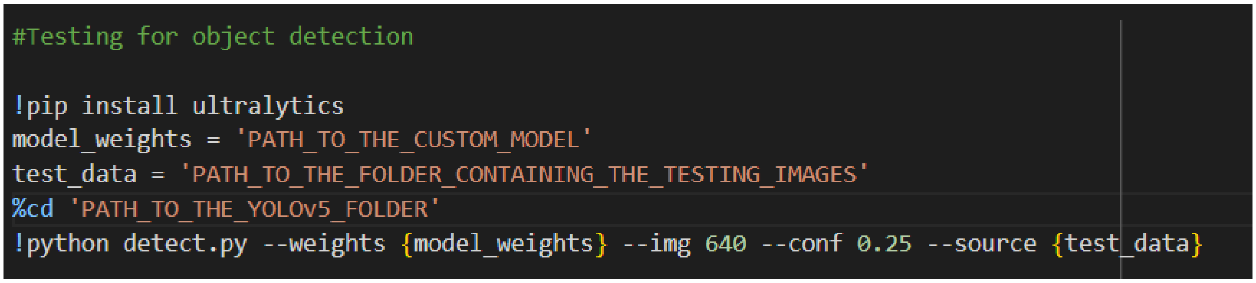


```
#this is the write template and its ready for customization where we have changed nc which is the number of classes
%%writetemplate "PATH_TO_THE_DATA.YAML_FILE_OF_THE_YOLO_MODEL_TO_TRAIN"
nc: [number of classes as per the requirements of the model]
depth_multiple: 0.33
width_multiple: 0.50
anchors:
  - [10,13, 16,30, 33,23]
  - [30,61, 62,45, 59,119]
  - [116,90, 156,198, 373,326]
backbone:
  [[-1, 1, Focus, [64, 3]],
   [-1, 1, Conv, [128, 3, 2]],
   [-1, 3, BottleneckCSP, [128]],
   [-1, 1, Conv, [256, 3, 2]],
   [-1, 9, BottleneckCSP, [256]],
   [-1, 1, Conv, [512, 3, 2]],
   [-1, 9, BottleneckCSP, [512]],
   [-1, 1, Conv, [1024, 3, 2]],
   [-1, 1, SPP, [1024, [5, 9, 13]]],
   [-1, 3, BottleneckCSP, [1024, False]],]
head:
  [[-1, 1, Conv, [512, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 6], 1, Concat, [1]],
   [-1, 3, BottleneckCSP, [512, False]],
   [-1, 1, Conv, [256, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 4], 1, Concat, [1]],
   [-1, 3, BottleneckCSP, [256, False]],
   [-1, 1, Conv, [256, 3, 2]],
   [[-1, 14], 1, Concat, [1]],
   [-1, 3, BottleneckCSP, [512, False]],
   [-1, 1, Conv, [512, 3, 2]],
   [[-1, 10], 1, Concat, [1]],
   [-1, 3, BottleneckCSP, [1024, False]], [[17, 20, 23], 1, Detect, [nc, anchors]],]
```



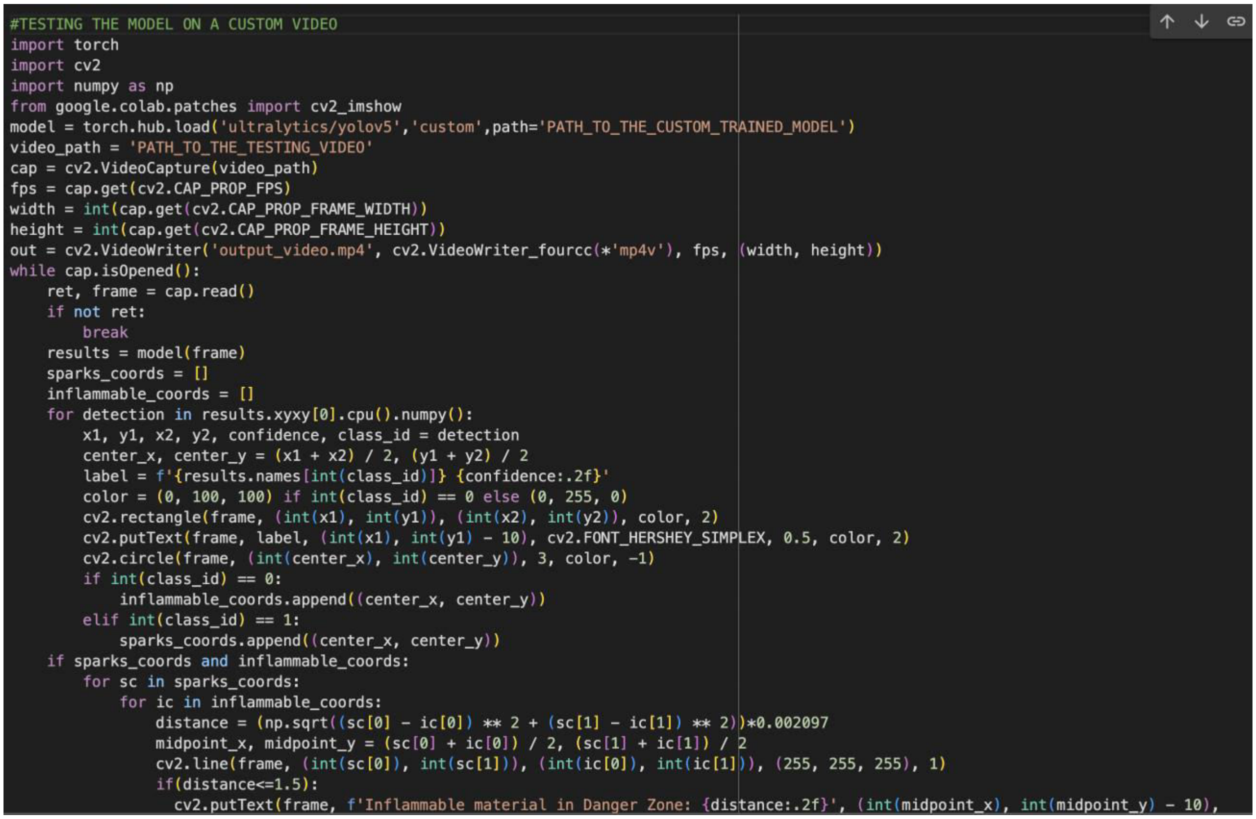
```
# train yolov5s on custom data for 90 epochs and timing its performance
%%time
%cd "PATH_TO_THE_YOLOv5_FOLDER"
!python train.py --img 640 --batch 16 --epochs 90 --data 'PATH_TO_THE_DATA.YAML_FILE_OF_THE_DATASET' --cfg ./models/custom_yolov5s.yaml --weights '' --name yolov5s_results  --cache
```

Testing the custom trained YOLOv5 Model for object detection which works for "PPE Detection" and "Safety net and Harness Detection":

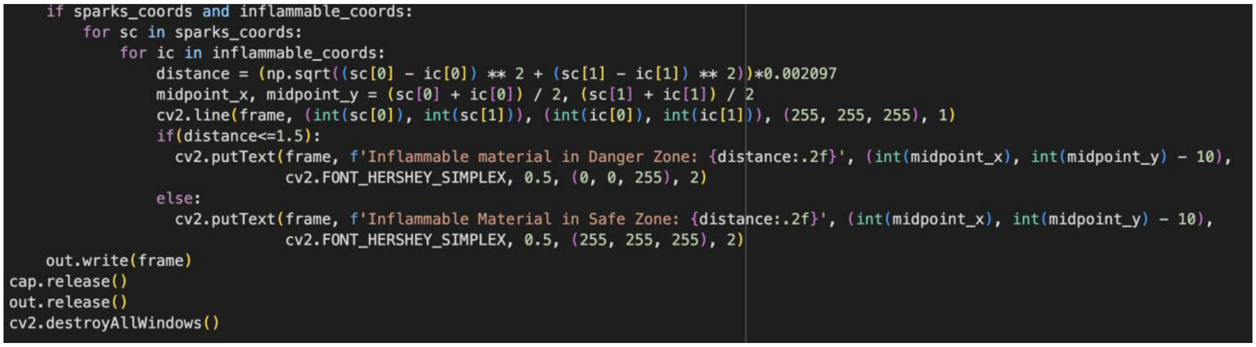


```
#Testing for object detection

!pip install ultralytics
model_weights = 'PATH_TO_THE_CUSTOM_MODEL'
test_data = 'PATH_TO_THE_FOLDER_CONTAINING_THE_TESTING_IMAGES'
%cd 'PATH_TO_THE_YOLOv5_FOLDER'
!python detect.py --weights {model_weights} --img 640 --conf 0.25 --source {test_data}
```

Adding Testing for the "Fire Hazard Detection" model:

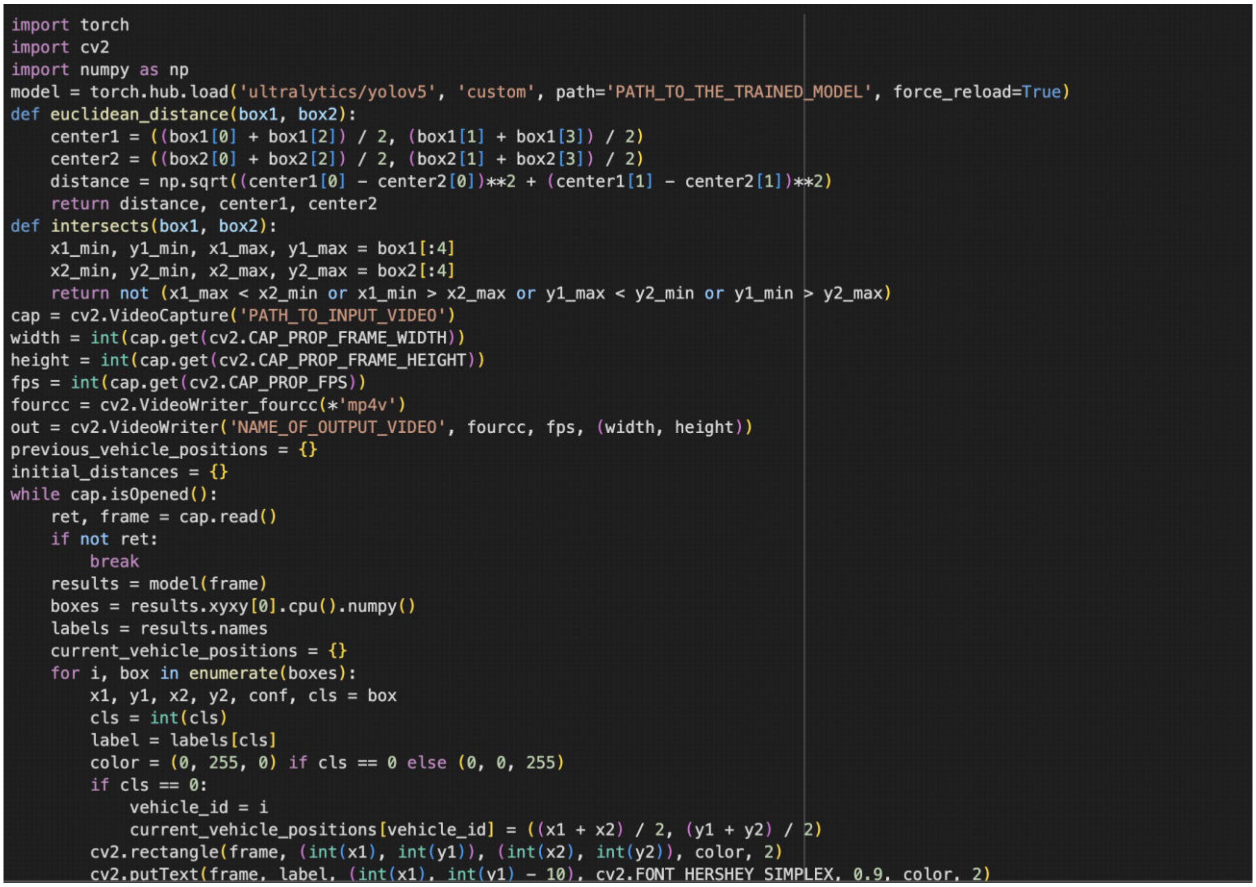


```
#TESTING THE MODEL ON A CUSTOM VIDEO
import torch
import cv2
import numpy as np
from google.colab.patches import cv2_imshow
model = torch.hub.load('ultralytics/yolov5','custom',path='PATH_TO_THE_CUSTOM_TRAINED_MODEL')
video_path = 'PATH_TO_THE_TESTING_VIDEO'
cap = cv2.VideoCapture(video_path)
fps = cap.get(cv2.CAP_PROP_FPS)
width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
out = cv2.VideoWriter('output_video.mp4', cv2.VideoWriter_fourcc(*'mp4v'), fps, (width, height))
while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break
    results = model(frame)
    sparks_coords = []
    inflammable_coords = []
    for detection in results.xyxy[0].cpu().numpy():
        x1, y1, x2, y2, confidence, class_id = detection
        center_x, center_y = (x1 + x2) / 2, (y1 + y2) / 2
        label = f'{results.names[int(class_id)]} {confidence:.2f}'
        color = (0, 100, 100) if int(class_id) == 0 else (0, 255, 0)
        cv2.rectangle(frame, (int(x1), int(y1)), (int(x2), int(y2)), color, 2)
        cv2.putText(frame, label, (int(x1), int(y1) - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)
        cv2.circle(frame, (int(center_x), int(center_y)), 3, color, -1)
        if int(class_id) == 0:
            inflammable_coords.append((center_x, center_y))
        elif int(class_id) == 1:
            sparks_coords.append((center_x, center_y))
    if sparks_coords and inflammable_coords:
        for sc in sparks_coords:
            for ic in inflammable_coords:
                distance = (np.sqrt((sc[0] - ic[0]) ** 2 + (sc[1] - ic[1]) ** 2))*0.002097
                midpoint_x, midpoint_y = (sc[0] + ic[0]) / 2, (sc[1] + ic[1]) / 2
                cv2.line(frame, (int(sc[0]), int(sc[1])), (int(ic[0]), int(ic[1])), (255, 255, 255), 1)
                if(distance<=1.5):
                  cv2.putText(frame, f'Inflammable material in Danger Zone: {distance:.2f}', (int(midpoint_x), int(midpoint_y) - 10),
```

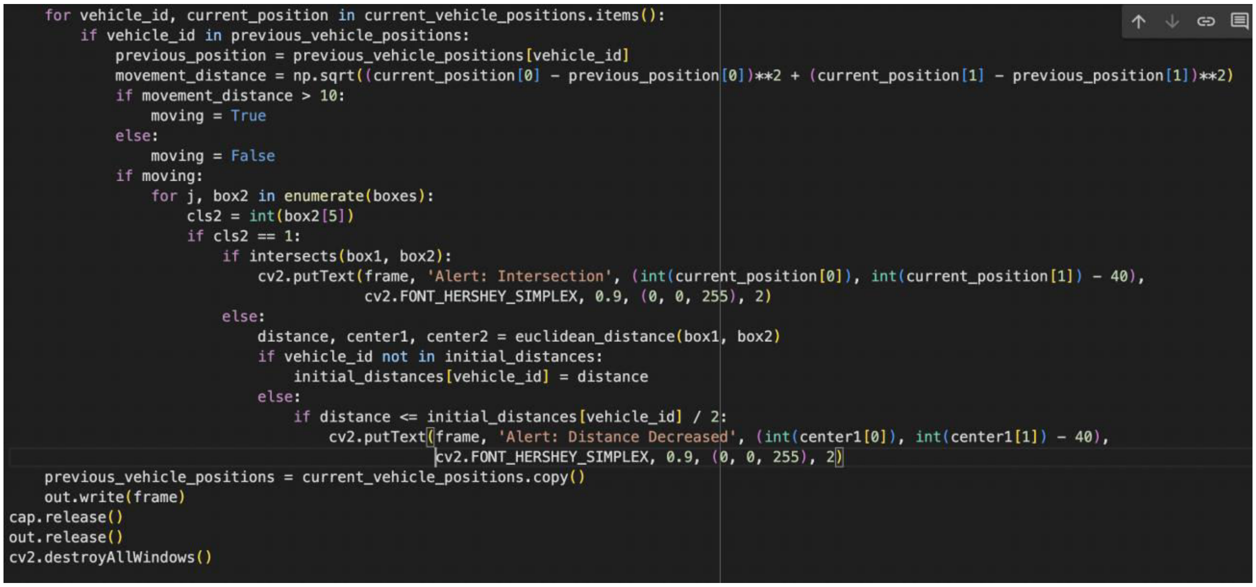


```
    if sparks_coords and inflammable_coords:
        for sc in sparks_coords:
            for ic in inflammable_coords:
                distance = (np.sqrt((sc[0] - ic[0]) ** 2 + (sc[1] - ic[1]) ** 2))*0.002097
                midpoint_x, midpoint_y = (sc[0] + ic[0]) / 2, (sc[1] + ic[1]) / 2
                cv2.line(frame, (int(sc[0]), int(sc[1])), (int(ic[0]), int(ic[1])), (255, 255, 255), 1)
                if(distance<=1.5):
                  cv2.putText(frame, f'Inflammable material in Danger Zone: {distance:.2f}', (int(midpoint_x), int(midpoint_y) - 10),
                              cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)
                else:
                  cv2.putText(frame, f'Inflammable Material in Safe Zone: {distance:.2f}', (int(midpoint_x), int(midpoint_y) - 10),
                              cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 2)
    out.write(frame)
cap.release()
out.release()
cv2.destroyAllWindows()
```

Adding Testing for the "Danger Zone Detection" model:



```python
import torch
import cv2
import numpy as np
model = torch.hub.load('ultralytics/yolov5', 'custom', path='PATH_TO_THE_TRAINED_MODEL', force_reload=True)
def euclidean_distance(box1, box2):
    center1 = ((box1[0] + box1[2]) / 2, (box1[1] + box1[3]) / 2)
    center2 = ((box2[0] + box2[2]) / 2, (box2[1] + box2[3]) / 2)
    distance = np.sqrt((center1[0] - center2[0])**2 + (center1[1] - center2[1])**2)
    return distance, center1, center2
def intersects(box1, box2):
    x1_min, y1_min, x1_max, y1_max = box1[:4]
    x2_min, y2_min, x2_max, y2_max = box2[:4]
    return not (x1_max < x2_min or x1_min > x2_max or y1_max < y2_min or y1_min > y2_max)
cap = cv2.VideoCapture('PATH_TO_INPUT_VIDEO')
width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
fps = int(cap.get(cv2.CAP_PROP_FPS))
fourcc = cv2.VideoWriter_fourcc(*'mp4v')
out = cv2.VideoWriter('NAME_OF_OUTPUT_VIDEO', fourcc, fps, (width, height))
previous_vehicle_positions = {}
initial_distances = {}
while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break
    results = model(frame)
    boxes = results.xyxy[0].cpu().numpy()
    labels = results.names
    current_vehicle_positions = {}
    for i, box in enumerate(boxes):
        x1, y1, x2, y2, conf, cls = box
        cls = int(cls)
        label = labels[cls]
        color = (0, 255, 0) if cls == 0 else (0, 0, 255)
        if cls == 0:
            vehicle_id = i
            current_vehicle_positions[vehicle_id] = ((x1 + x2) / 2, (y1 + y2) / 2)
        cv2.rectangle(frame, (int(x1), int(y1)), (int(x2), int(y2)), color, 2)
        cv2.putText(frame, label, (int(x1), int(y1) - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.9, color, 2)
```



```python
    for vehicle_id, current_position in current_vehicle_positions.items():
        if vehicle_id in previous_vehicle_positions:
            previous_position = previous_vehicle_positions[vehicle_id]
            movement_distance = np.sqrt((current_position[0] - previous_position[0])**2 + (current_position[1] - previous_position[1])**2)
            if movement_distance > 10:
                moving = True
            else:
                moving = False
            if moving:
                for j, box2 in enumerate(boxes):
                    cls2 = int(box2[5])
                    if cls2 == 1:
                        if intersects(box1, box2):
                            cv2.putText(frame, 'Alert: Intersection', (int(current_position[0]), int(current_position[1]) - 40),
                                        cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 0, 255), 2)
                        else:
                            distance, center1, center2 = euclidean_distance(box1, box2)
                            if vehicle_id not in initial_distances:
                                initial_distances[vehicle_id] = distance
                            else:
                                if distance <= initial_distances[vehicle_id] / 2:
                                    cv2.putText(frame, 'Alert: Distance Decreased', (int(center1[0]), int(center1[1]) - 40),
                                                cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 0, 255), 2)
    previous_vehicle_positions = current_vehicle_positions.copy()
    out.write(frame)
cap.release()
out.release()
cv2.destroyAllWindows()
```