#1. first, import all necessary modules

from pathlib import Path

import blobconverter

import cv2

import depthai

import numpy as np

#2. Create pipelin

pipeline = depthai.Pipeline()

# 2.1. First, we want the Color camera as the output

cam\_rgb = pipeline.createColorCamera()

#2.2.1

cam\_rgb.setPreviewSize(300, 300) # 300x300 will be the preview frame size, available as 'preview' output of the node

cam\_rgb.setInterleaved(False)

detection\_nn = pipeline.createMobileNetDetectionNetwork()

# Blob is the Neural Network file, compiled for MyriadX. It contains both the definition and weights of the model

# We're using a blobconverter tool to retreive the MobileNetSSD blob automatically from OpenVINO Model Zoo

detection\_nn.setBlobPath(blobconverter.from\_zoo(name='mobilenet-ssd', shaves=6))

# Next, we filter out the detections that are below a confidence threshold. Confidence can be anywhere between <0..1>

detection\_nn.setConfidenceThreshold(0.5)

# XLinkOut is a "way out" from the device. Any data you want to transfer to host need to be send via XLink

xout\_rgb = pipeline.createXLinkOut()

xout\_rgb.setStreamName("rgb")

xout\_nn = pipeline.createXLinkOut()

xout\_nn.setStreamName("nn")

cam\_rgb.preview.link(xout\_rgb.input)

cam\_rgb.preview.link(detection\_nn.input)

detection\_nn.out.link(xout\_nn.input)

# Pipeline is now finished, and we need to find an available device to run our pipeline

# we are using context manager here that will dispose the device after we stop using it

with depthai.Device(pipeline) as device:

# From this point, the Device will be in "running" mode and will start sending data via XLink

# To consume the device results, we get two output queues from the device, with stream names we assigned earlier

q\_rgb = device.getOutputQueue("rgb")

q\_nn = device.getOutputQueue("nn")

# Here, some of the default values are defined. Frame will be an image from "rgb" stream, detections will contain nn results

frame = None

detections = []

# Since the detections returned by nn have values from <0..1> range, they need to be multiplied by frame width/height to

# receive the actual position of the bounding box on the image

def frameNorm(frame, bbox):

normVals = np.full(len(bbox), frame.shape[0])

normVals[::2] = frame.shape[1]

return (np.clip(np.array(bbox), 0, 1) \* normVals).astype(int)

while True:

# we try to fetch the data from nn/rgb queues. tryGet will return either the data packet or None if there isn't any

in\_rgb = q\_rgb.tryGet()

in\_nn = q\_nn.tryGet()

if in\_rgb is not None:

# If the packet from RGB camera is present, we're retrieving the frame in OpenCV format using getCvFrame

frame = in\_rgb.getCvFrame()

if in\_nn is not None:

# when data from nn is received, we take the detections array that contains mobilenet-ssd results

detections = in\_nn.detections

if frame is not None:

for detection in detections:

# for each bounding box, we first normalize it to match the frame size

bbox = frameNorm(frame, (detection.xmin, detection.ymin, detection.xmax, detection.ymax))

# and then draw a rectangle on the frame to show the actual result

cv2.rectangle(frame, (bbox[0], bbox[1]), (bbox[2], bbox[3]), (255, 0, 0), 2)

# After all the drawing is finished, we show the frame on the screen

cv2.imshow("preview", frame)

# at any time, you can press "q" and exit the main loop, therefore exiting the program itself

if cv2.waitKey(1) == ord('q'):

break