501b-inference-with-a-robotic-arm

March 29, 2024

1 Simulation of production line with defects - Dataset creation and Inference

This notebook is originally created by [@paularamos](https://github.com/paularamo) for CVPR-2022 Tutorial How to get quick and performant model for your edge application. From data to application

1.0.1 Definitions

Anomalib: Anomalib is a deep learning library that aims to collect state-of-the-art anomaly detection algorithms for benchmarking on both public and private datasets. Anomalib provides several ready-to-use implementations of anomaly detection algorithms described in the recent literature, as well as a set of tools that facilitate the development and implementation of custom models. The library has a strong focus on image-based anomaly detection, where the goal of the algorithm is to identify anomalous images, or anomalous pixel regions within images in a dataset.

Dobot The Magician is an education robot arm portable and capable to run various automation tasks. With an interface in C++ and python we can control the robot using this notebook.

> NOTE: If you don't have the robot you can replace it by your custom problem.

1.0.2 Use case

Using the Dobot Magician we could simulate a production line system. Imagine we have a cubes factory and they need to know when a defect piece appear in the process. We know very well what the aspect ratio of the normal cubes is. Defects are coming no often and we need to put those defect cubes out of the production line.

Class	Yellow cube	Red cube	Green cube	Inferencing using Anomalib
Normal				
Abnormal				

Using Anomalib we are expecting to see this result.

```
[ ]: """501b_inference_with_a_robotic_arm.ipynb."""

     # Anomalib imports
     from __future__ import annotations
```

```python
from typing import TYPE_CHECKING
import sys
import time  # time library
from datetime import datetime
from pathlib import Path
from threading import Thread

if TYPE_CHECKING:
    import numpy as np

# importing required libraries
import cv2  # OpenCV library
from anomalib.deploy import OpenVINOInferencer
```

1.0.3 Helper funtions

Here you will find funtions to create filenames, capture images, run the inference and read the confidence of the detection.

[ ]:
```python
# Prepare the path to the datasets and the weights
dataset_path = Path.cwd() / "cubes"
weights_path = Path.cwd() / "weights"
```

[ ]:
```python
def create_filename(path: Path) -> str:
    """Create the filename for new data(images).

    Args:
        path (Path): Initial path to save new images and results.

    Returns:
        str: Captured image filename
    """
    path.mkdir(exist_ok=True, parents=True)
    now = datetime.now().strftime("%Y%m%d%H%M%S")
    return str(path / f"input_{now}.jpg")
```

1.0.4 Prepare the mode (acquisition or inference mode) and define the work directory

[ ]:
```python
# Acquisition mode
acquisition = False  # True False
source = 0  # number of the camera you want to use
folder = "abnormal"  # normal or abnormal

# If acquisition is False this notebook will work in inference mode
if acquisition is False:
    # If you are running inference check where the OpenVINO model is stored
    openvino_model_path = weights_path / "openvino" / "model.bin"
    metadata_path = weights_path / "openvino" / "metadata.json"

    print("OpenVINO model exist: ", openvino_model_path.exists())
    print("OpenVINO path: ", openvino_model_path)
    print("Metadata model exist: ", metadata_path.exists())
    print("Metadata path: ", metadata_path)

    inferencer = OpenVINOInferencer(
        path=openvino_model_path,  # Path to the OpenVINO IR model.
        metadata=metadata_path,  # Path to the metadata file.
        device="CPU",  # We would like to run it on an Intel CPU.
    )

    if dataset_path.exists() is False:
        print("Make sure you have the dataset in a proper folder or it i␣
 ↪already created")
else:
    dataset_path.mkdir(parents=True, exist_ok=True)
```

1.0.5 Helper class for implementing multi-threading

Using multi-threading we will open the video to auto-capture an image when the robot locates the cube in front of the camera.

[ ]:
```
class CameraStream:
    """Read video stream from camera via multi-threading."""

    def __init__(self, stream_id: int = 0) -> None:
        self.stream_id = stream_id

        # opening video capture stream
        self.video_capture = cv2.VideoCapture(self.stream_id)
        if self.video_capture.isOpened() is False:
            print("[Exiting]: Error accessing cam stream.")
            sys.exit(0)
        fps_input_stream = int(self.video_capture.get(5))  # hardware fps
        print(f"FPS of input stream: {fps_input_stream}")

        # reading a single frame from vcap stream for initializing
        self.grabbed, self.frame = self.video_capture.read()
        if self.grabbed is False:
            print("[Exiting] No more frames to read")
            sys.exit(0)
        # self.stopped is initialized to False
        self.stopped = True
        # thread instantiation
        self.thread = Thread(target=self.update, args=())
        self.thread.daemon = True  # daemon threads run in background

    def start(self) -> None:
        """Start thread."""
        self.stopped = False
        self.thread.start()

    def update(self) -> None:
        """Update the next available frame."""
        while True:
            if self.stopped is True:
                break
            self.grabbed, self.frame = self.video_capture.read()
            if self.grabbed is False:
                print("[Exiting] No more frames to read")
                self.stopped = True
                break
        self.video_capture.release()

    def read(self) -> np.ndarray:
        """Read the next frame."""
        return self.frame

    def stop(self) -> None:
        """Stop reading frames."""
        self.stopped = True
```

1.0.6 Function to visualize the prediction

1.0.7 Using a webcam or a USB camera for running the inference

Connect and identify your USB camera, we will use a a video player to embed the video in this notebook.

We will now work with the robot, and the driver must remain in the same 501 folder. Please move the files from `./501_dobot/dobot_api` to `./501_dobot` by copying and pasting. Ascertain that you run the notebook 501a if the `dobot_api` folder hasn't been created.

> NOTE: If you don't have the robot you can replace it by your custom problem. See the comments below.

[ ]:
```
# Dobot/general imports
# pylint: disable=wrong-import-order
import DobotDllType as dType

CON_STR = {
    dType.DobotConnect.DobotConnect_NoError: "DobotConnect_NoError",
    dType.DobotConnect.DobotConnect_NotFound: "DobotConnect_NotFound",
    dType.DobotConnect.DobotConnect_Occupied: "DobotConnect_Occupied",
}

# Load Dll and get the CDLL object
api = dType.load()

# Connect Dobot
state = dType.ConnectDobot(api, "", 115200)[0]
print("Connect status:", CON_STR[state])

use_popup = True  # True

if state == dType.DobotConnect.DobotConnect_NoError:
    print(
        "[HOME] Restore to home position at first launch, please wait 30␣
↪seconds after turnning on the Dobot Magician.",
    )
    print(
        "[BLOCKS] Place them besides the non-motor side of the conveyor belt,"
        " the same side where the pick and place arm is.",
    )
    print("[PLACING BLOCKS] Place the blocks by 3x3.")
    print("[CALIBRATION POINT] Looking from the back of Dobot, the top left␣
↪block is the calibration point.")
    print("[CALIBRATION] Set the first variable to 0 to test the calibration␣
↪point, then set 1 to start running.")
    print(
        "[DIRECTION] Standing behind Dobot Magician facing its front direction,␣
↪X is front and back direction, "
        "Y is left and right direction. ",
    )
    print("[CONNECTION] Motor of the conveyor belt connects to port Stepper1.")

    Calibration__0__Run__1 = 1
    Calibration_X = 221.2288
    Calibration_Y = -117.0036
    Calibration_Z = -42.3512
    Place_X = 23.7489  # 42.2995 #
    Place_Y = -264.2602  # -264.6927 #
    Place_Z = 18.0862  # 63.65 #
    Anomaly_X = -112  # -84.287 #
    Anomaly_Y = -170  # -170.454 #
    Anomaly_Z = 90  # 61.5359 #
    dType.SetEndEffectorParamsEx(api, 59.7, 0, 0, 1)
    j = 0
    k = 0
    dType.SetPTPJointParamsEx(api, 400, 400, 400, 400, 400, 400, 400, 400, 1)
    dType.SetPTPCommonParamsEx(api, 100, 100, 1)
    dType.SetPTPJumpParamsEx(api, 40, 100, 1)
    dType.SetPTPCmdEx(api, 0, Calibration_X, Calibration_Y, Calibration_Z, 0, 1)
    dType.SetEndEffectorSuctionCupEx(api, 0, 1)
    STEP_PER_CRICLE = 360.0 / 1.8 * 10.0 * 16.0
    MM_PER_CRICLE = 3.1415926535898 * 36.0
    vel = float(0) * STEP_PER_CRICLE / MM_PER_CRICLE
    dType.SetEMotorEx(api, 1, 0, int(vel), 1)

    if Calibration__0__Run__1:
        for _ in range(9):
            # initializing and starting multi-threaded webcam input stream
            cam_stream = CameraStream(stream_id=0)  # 0 id for main camera
            cam_stream.start()

            dType.SetPTPCmdEx(api, 0, (Calibration_X - j), (Calibration_Y - k),␣
↪(Calibration_Z - 10), 0, 1)
            dType.SetEndEffectorSuctionCupEx(api, 1, 1)
            dType.SetPTPCmdEx(api, 0, (Place_X - 0), (Place_Y - 0), (Place_Z +␣
↪90), 0, 1)

            # adding a delay for simulating video processing time
            delay = 0.3  # delay value in seconds
            time.sleep(delay)
            # Capture a frame from the video player - start thread
            frame = cam_stream.read()

            if acquisition:
                # create filename to next frame
                filename = create_filename(path=(dataset_path / folder))
                cv2.imwrite(filename, frame)
                dType.SetPTPCmdEx(api, 0, Place_X, Place_Y, Place_Z, 0, 1)

            else:
                # Get the inference results.
                frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
                # INFERENCE WITH OPENVINO
                predictions = inferencer.predict(image=frame)
                print(predictions.pred_score)
                if predictions.pred_score > 0.48:  # modify the threshold␣
↪depending of your needs
                    dType.SetPTPCmdEx(api, 0, Anomaly_X, Anomaly_Y, Anomaly_Z,␣
↪0, 1)  # define point for abnormalities
                else:
                    dType.SetPTPCmdEx(api, 0, Place_X, Place_Y, Place_Z, 0, 1)

            dType.SetEndEffectorSuctionCupEx(api, 0, 1)
            j = j + 25
            if j == 75:
                k = k + 25
                j = 0
            dType.SetPTPCmdEx(api, 7, 0, 0, 20, 0, 1)
            time_start = dType.gettime()[0]
            STEP_PER_CRICLE = 360.0 / 1.8 * 10.0 * 16.0
            MM_PER_CRICLE = 3.1415926535898 * 36.0
            vel = float(50) * STEP_PER_CRICLE / MM_PER_CRICLE
            dType.SetEMotorEx(api, 1, 1, int(vel), 1)
            filename = None
            score = 0
            while True:
                if (dType.gettime()[0]) - time_start >= 0.5:  # Time over␣
↪conveyor belt
                    STEP_PER_CRICLE = 360.0 / 1.8 * 10.0 * 16.0
                    MM_PER_CRICLE = 3.1415926535898 * 36.0
                    vel = float(0) * STEP_PER_CRICLE / MM_PER_CRICLE
                    dType.SetEMotorEx(api, 1, 0, int(vel), 1)
                    break
        dType.SetEndEffectorSuctionCupEx(api, 0, 1)
        dType.SetPTPCmdEx(api, 0, Calibration_X, Calibration_Y, Calibration_Z,␣
↪0, 1)
        cam_stream.stop()  # stop the webcam stream
```