OpenVINO introduction - workshop

Step 1 - Install *Ubuntu 20.04* on the *UP Squared AI Edge X* mini PC

1. Download *Ubuntu 20.04* - https://releases.ubuntu.com/20.04/
2. Install *Ubuntu 20.04* on mini PC following the guide -

   https://ubuntu.com/tutorials/install-ubuntu-desktop, in the option to select the type of installation, select the minimum installation

Step 2 - Global installation of required packages

1. Run the following commands:

```
a. cd ~
b. sudo apt install git pip wget libboost-filesystem1.71.0 ffmpeg -y
c. python3 -m pip install --upgrade pip
d. pip install opencv-python scipy
```

Step 3 - YOLOX-Tiny model test

1. Run the following commands:

```
a. cd ~
b. git clone https://github.com/Megvii-BaseDetection/YOLOX.git
c. cd YOLOX
d. pip install -r requirements.txt
e. pip install -v -e .
f. wget
https://github.com/MegviiBaseDetection/YOLOX/releases/download/0.1.1r
c0/yolox_tiny.pth
g. python3 tools/demo.py video -n yolox-tiny -c yolox_tiny.pth --path
/dev/video0 --conf 0.25 --nms 0.45 --tsize 640 --device cpu --fp16
```

Step 4 – Installing OpenVINO (consumer version)

1. Run the following commands:

```
a. cd ~
b. wget
https://registrationcenterdownload.intel.com/akdlm/irc_nas/18617/l_op
envino_toolkit_p_2022.1.0
.643_offline.sh
c. sudo chmod +x l_openvino_toolkit_p_2022.1.0.643_offline.sh
d. sudo ./l_openvino_toolkit_p_2022.1.0.643_offline.sh
```

2. Install the OpenVINO software using the installation assistant, leave the default options, ignore any messages about missing packages

3. Run the following commands:

```
a. sudo -E
/opt/intel/openvino_2022/install_dependencies/install_openvino_depen
dencies.sh
b. echo 'source /opt/intel/openvino_2022/setupvars.sh' >> ~/.bashrc
c. exec bash
d.
/opt/intel/openvino_2022/runtime/3rdparty/hddl/install_IVAD_VPU_depe
ndencies.sh
```

4. Restart the mini PC

Step 5 – Installing OpenVINO (development version)

1. Run the following commands:

```
a. cd ~
b. python3 -m venv openvino_env
c. source openvino_env/bin/activate
d. python -m pip install --upgrade pip
e. pip install openvino-dev[ONNX,pytorch]==2022.1.0
```

Step 6 – Converting the model to a format supported by OpenVINO

1. Run the following commands (convert from PyTorch -> ONNX):

```
a. cd ~
b. deactivate
c. cp YOLOX/yolox_tiny.pth model.pth
d. export PROTOCOL_BUFFERS_PYTHON_IMPLEMENTATION=python
e. python3 YOLOX/tools/export_onnx.py --output-name model.onnx -f
YOLOX/exps/default/yolox_tiny.py -c model.pth -o 10
```

2. Run the following commands (convert from PyTorch -> ONNX):

```
a. cd ~
b. source openvino_env/bin/activate
```

```
c. mo --input_model model.onnx --input_shape [1,3,640,640] --
data_type
FP16 --output_dir model
```

Step 7 - Test the converted model

1. Run the following commands:

```
a. cd ~
b. deactivate
c. git clone --recurse-submodules
https://github.com/openvinotoolkit/open_model_zoo.git
d. cd open_model_zoo/demos/object_detection_demo/python
```

2. Test of the converted model on the CPU:

```
a. python3 object_detection_demo.py -m ~/model/model.xml --labels
../../../data/dataset_classes/coco_80cl.txt -at yolox -i /dev/video0
--input_size 640 640 -d CPU
```

2. Test of the converted model on the VPU:

```
a. python3 object_detection_demo.py -m ~/model/model.xml --labels
../../../data/dataset_classes/coco_80cl.txt -at yolox -i /dev/video0
--input_size 640 640 -d HDDL
```

Step 8 - Tasks to complete

1. Compare the performance (average frame rate) of model versions: YOLOX without OpenVINO, YOLOX with OpenVINO on CPU, YOLOX with OpenVINO on VPU.
   Use FP32 precision when converting models to OpenVINO format.
2. Compare the performance (average frame rate) of model versions: YOLOX without OpenVINO, YOLOX with OpenVINO on CPU, YOLOX with OpenVINO on VPU.
   Use FP16 precision when converting models to OpenVINO format.
3. Apart from the difference in performance, have you noticed any differences in the performance of models based on the precision of FP16 compared to those based on the precision of FP32? If so, what could they be caused by?
4. Again convert the model to OpenVINO format with FP16 precision, but this time add parameter --disable_fusing. Have you noticed performance differences?
   If so, what could it be caused by?