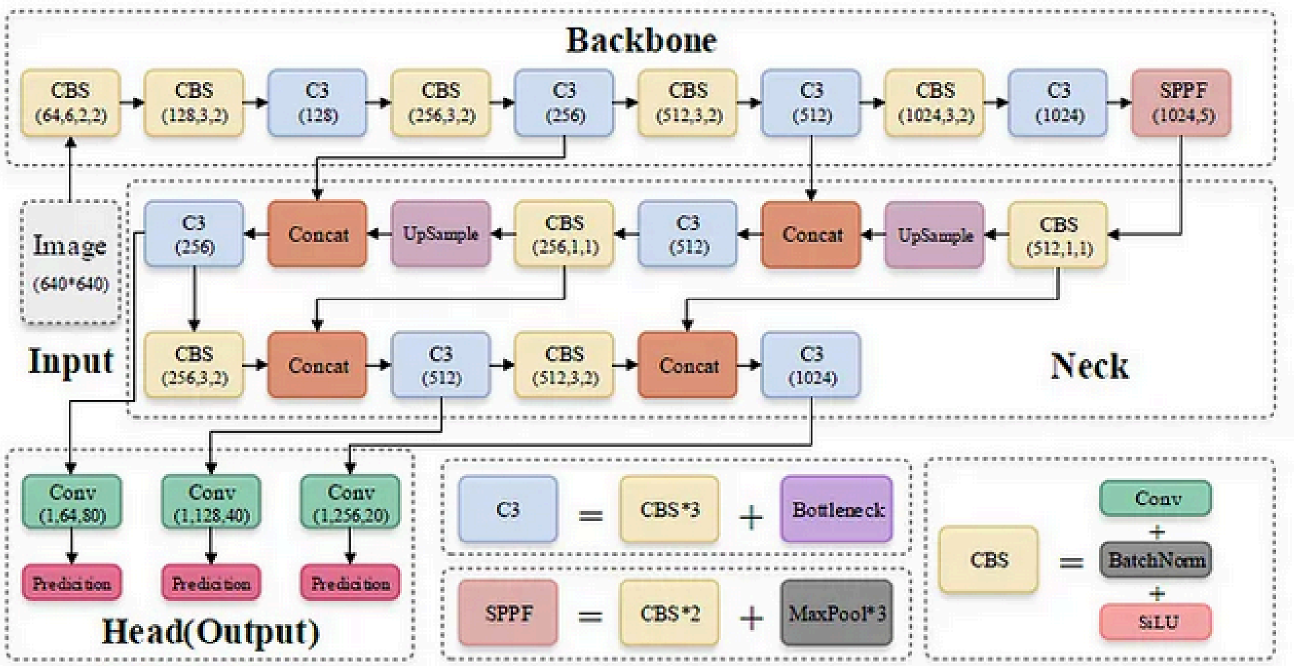
YOLOv5 in FPGA

YOLOv5 in FPGA

- This repo is about how can we deploy the **yolov5** object detection model on **Kria KV260** FPGA board.
- To deploy yolov5 in FPGA board, we need to modify the layers of the model because all of the layers of yolov5 are not supportable in **DPU** of FPGA board. So, we need to find out those unsupported layers/operators either by mannually reading from [here] or by doing inspection.

About YOLOv5:

- Yolov5 build upon the foundation laid by YOLO that has granered significant attention due to its exceptional performance and efficiency.
- This model addresses the trade-off between accuracy and inference speed across various yolov5 variants **(n,s,m,l,x)**.



Architecture:

- The architecture of yolov5 consists of three stages i.e. **Backbone, Neck and Head**.

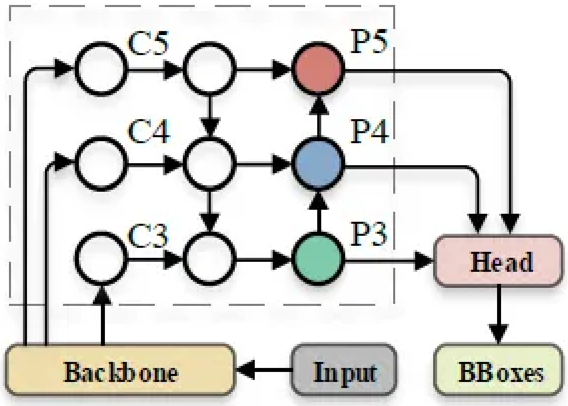
1. **Backbone:**

- It is basically a stack of convolutional layers that extracts the features from input image at different scales.
- Here, backbone is **modified CSPDarknet53**.

- It stacks multiple **CBS (Convolution + BatcNormalization + SiLU)** modules and **C3** modules and at last it has **SPPF** module which is used to increase the **receptive** field of the model.
- Here, **CBS** is used to assist the **C3** module in feature extraction process.

2. **Neck:**

- This stage combines the features of different scales came from backbone and produces the fine-grained fetures so that the objects of different scales can be detected.
- Yolov5 uses **PAN (Path Aggregation Network)** in neck which first **upsamples** the high level feature maps and comibines with low level feature maps and **downsamples** the low level feature maps which is then combines with high level featur maps to produce fine-grained features as shown in figure below:



3. **Head:**

- This stage generates the final outputs i.e. bounding box offsets and

logits for class probabilities.

- In head of yolov5, the ouput are decoded in different format than in previous **yolo**.
- The equations used to decode the ouput of yolov5 are:

$$g_x = 2\sigma(s_x) - 0.5 + r_x$$
$$g_y = 2\sigma(s_y) - 0.5 + r_y$$
$$g_h = p_h(2\sigma(s_h))^2$$
$$g_w = p_w(2\sigma(s_w))^2$$

- Here, **sx, sy** are predicted center point of bounding boxes which are relative to **top left** corner of cell of grid, **sh, sw** are predicted offset bounding box height and width, **rx, ry** are top left corner point of cell of grid and **ph, pw** are height and width of **anchors**.

Model Check:

- As we know that the author of yolov5 is **Ultralytics** who developed yolov5 architecture on the basis of previous YOLO and also it was the first time the yolo version was implemented using **PyTorch**.
- We can use different variants of yolov5 for object detection from official github repo of ultralytics as i have mentioned in referece section [4].
- But here, i have used yolov5 object detection model from a github repo as i have mentioned reference section [1]. This repo is also referenced from ultralytics and there is no difference in model so i used this repo.
- So first of all, i have checked the architecture and model by using its pretrained weight to make sure that the model is working or not, which was good.
- There was no python script for the inference, but there is jupyter notebook. So i created an inference and evaluation scirpt for this.

Deployment:

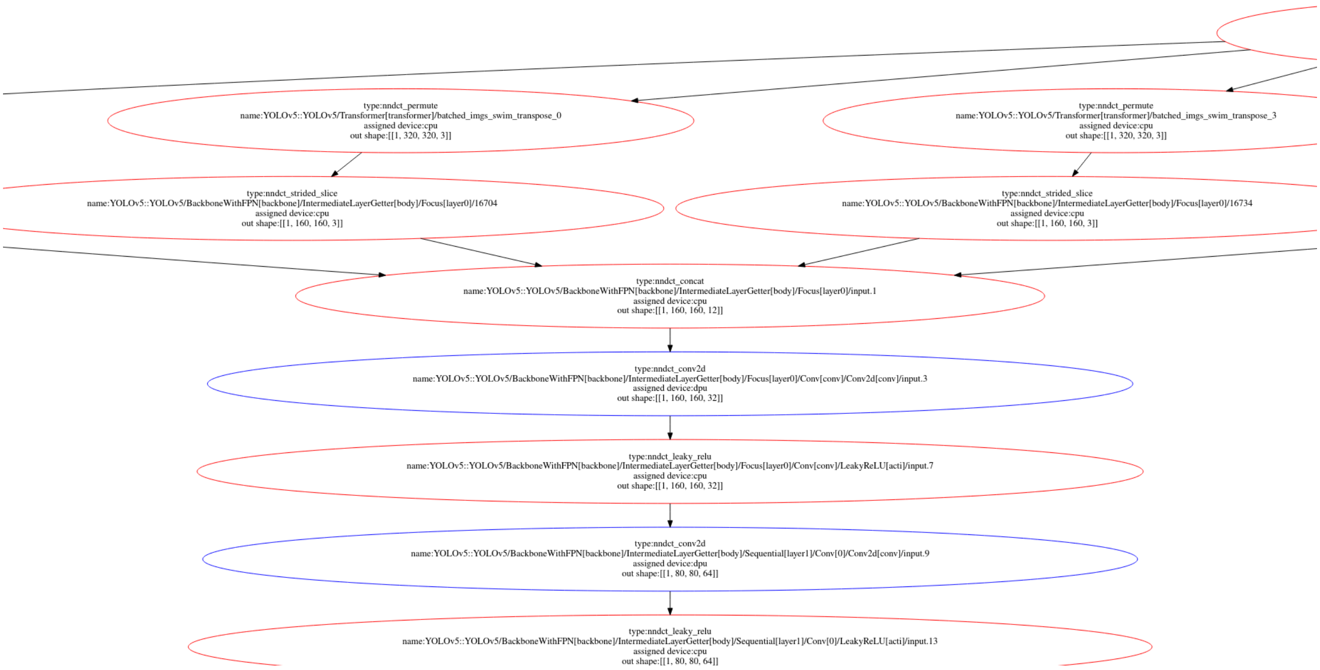
- To deploy any **CNN** model, we need to first check that our model is supportable in **DPU** or not otherwise we will get multiple **DPU subgraph** and we have to run model in **DPU** and **CPU** separately.

Inspection:

- Use **inspector** from **pytorch_nndct.apis** in **Vitis-AI**.
- If some operators can't be assigned to DPU, message like this can be seen:

```
[VAIQ_NOTE]: The operators assigned to the CPU are as follows(see more details in 'Deployment/inspection_results/inspect_DPUCZDX8G_ISA1_B4096.txt'):
node name                                                                                                                         op Type           hardware constraints
-----------------------------------------------------------------------------------------------------------------------------     ----------------  ----------------------------------
--------
YOLOv5::YOLOv5/BackboneWithFPN[backbone]/IntermediateLayerGetter[body]/Sequential[layer1]/ConcatBlock[concat]/Sequential[tail]/LeakyReLU[1]/input.39   nndct_leaky_relu  nndct_leaky_relu can't be assigned
 to DPU.
YOLOv5::YOLOv5/BackboneWithFPN[backbone]/IntermediateLayerGetter[body]/Sequential[layer2]/ConcatBlock[concat]/Sequential[tail]/LeakyReLU[1]/input.99   nndct_leaky_relu  nndct_leaky_relu can't be assigned
 to DPU.
YOLOv5::YOLOv5/BackboneWithFPN[backbone]/IntermediateLayerGetter[body]/Sequential[layer3]/ConcatBlock[concat]/Sequential[tail]/LeakyReLU[1]/input.159  nndct_leaky_relu  nndct_leaky_relu can't be assigned
 to DPU.
YOLOv5::YOLOv5/BackboneWithFPN[backbone]/PathAggregationNetwork[fpn]/ConcatBlock[inner_blocks]/ModuleList[2]/Sequential[tail]/LeakyReLU[1]/input.207   nndct_leaky_relu  nndct_leaky_relu can't be assigned
 to DPU.
YOLOv5::YOLOv5/BackboneWithFPN[backbone]/PathAggregationNetwork[fpn]/ConcatBlock[inner_blocks]/ModuleList[1]/Sequential[tail]/LeakyReLU[1]/input.245   nndct_leaky_relu  nndct_leaky_relu can't be assigned
 to DPU.
YOLOv5::YOLOv5/BackboneWithFPN[backbone]/PathAggregationNetwork[fpn]/ConcatBlock[inner_blocks]/ModuleList[0]/Sequential[tail]/LeakyReLU[1]/input.283   nndct_leaky_relu  nndct_leaky_relu can't be assigned
 to DPU.
```

- And also inspector gives a **svg** image where we can see those operators which are assigned to **CPU** are in **RED** oval and those which are assigned to **DPU** are in **BLUE** oval.



Unsupported operators found in Yolov5:

- Sigmoid activation function.
- Negative slope of LeakyReLU.
- Tensor slicing.

Made unsupported operators supportable in FPGA

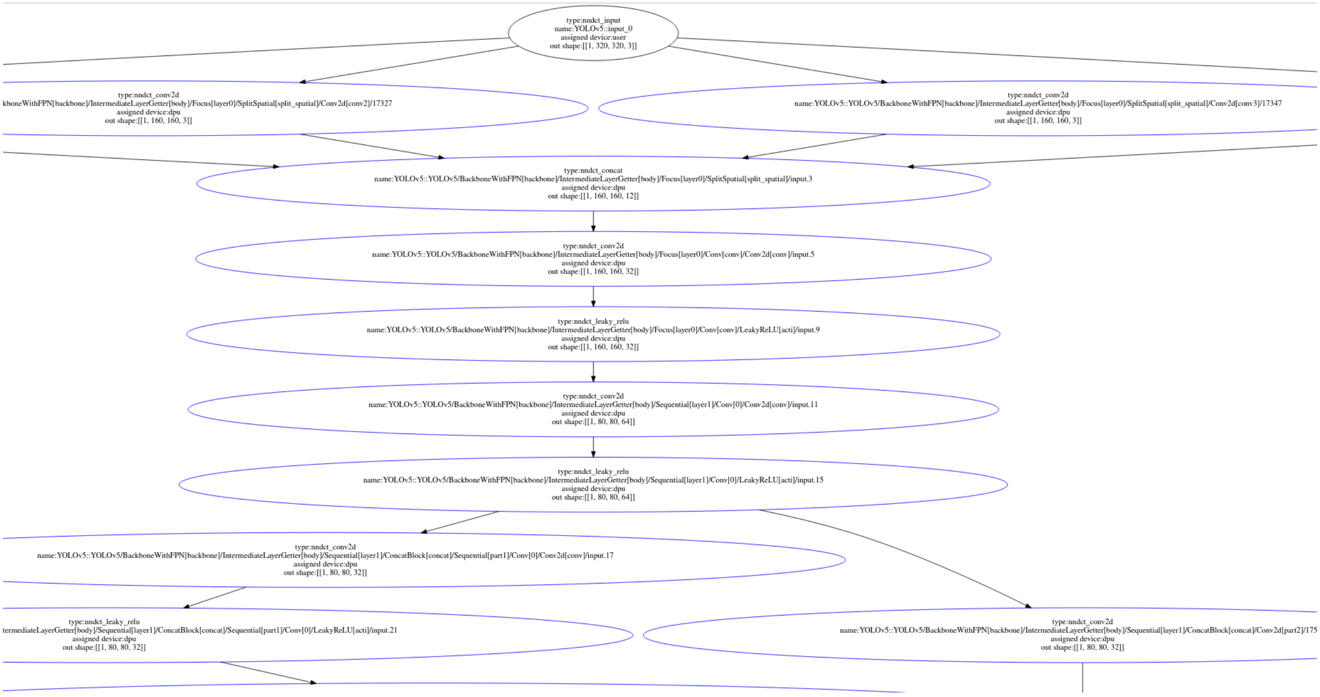
- Since **LeakyReLU** with negative slope **0.1** throws this error: **YOLOv5YOLOv5_BackboneWithFPN_backboneIntermediateLayerGetter_bodyFocus_layer0Conv_convConv2d_convinput_3, type = conv2d-fix has been assigned to CPU: (DPU does not support activation type: LEAKYRELU. Its alpha is 0.100000, but DPU only support 0.1015625.)**, replaced negative slope of leaky relu activation function by **(26/256 = 0.1015625)**.
- Replaced **torch.sigmoid()** function by **torch.Hardsigmoid()** because sigmoid is not supported in DPU.
- Put **Transformer** class outside of **YOLOv5** class(model) because it performs augmentation and preprocessing steps like flip, mosaic, resize, etc which contains unsupported operators like tensor slicing, permute,etc.
- Put the post processing steps happening during the evaluation/inference at the last layer of head that contains unsupported operators like permute, view, etc outside of the model.
- Replaced tensor slicing steps in backbone **(in Focus class)** by a convolutinal layer:

```python
class SplitSpatial(nn.Module):
    def __init__(self,in_ch):
        super(SplitSpatial, self).__init__()
        self.conv1 = nn.Conv2d(in_ch, in_ch, kernel_size=2, stride=2, bias=False).requires_grad_(False)
        self.conv2 = nn.Conv2d(in_ch, in_ch, kernel_size=2, stride=2, bias=False).requires_grad_(False)
        self.conv3 = nn.Conv2d(in_ch, in_ch, kernel_size=2, stride=2, bias=False).requires_grad_(False)
        self.conv4 = nn.Conv2d(in_ch, in_ch, kernel_size=2, stride=2, bias=False).requires_grad_(False)

        with torch.no_grad():
            wts1 = torch.zeros(in_ch, in_ch, 2,2)
            wts2 = torch.zeros(in_ch, in_ch, 2,2)
            wts3 = torch.zeros(in_ch, in_ch, 2,2)
            wts4 = torch.zeros(in_ch, in_ch, 2,2)
            for i in range(in_ch):
                wts1[i, i, 0, 0] = 1
                wts2[i, i, 1, 0] = 1
                wts3[i, i, 0, 1] = 1
                wts4[i, i, 1, 1] = 1

            self.conv1.weight.copy_(wts1)
            self.conv2.weight.copy_(wts2)
            self.conv3.weight.copy_(wts3)
            self.conv4.weight.copy_(wts4)

    def forward(self, x):
        x1 = self.conv1(x)
        x2 = self.conv2(x)
        x3 = self.conv3(x)
        x4 = self.conv4(x)
        return torch.cat((x1, x2, x3, x4), dim=1)
```

- Since, LeakyReLU used with batchnormalization only and from Vitis-AI User Guide 1414 **Activations would be fused to adjacent operations such as convolution and add**. So added a convolutional layer with leakyrelu function which has non-trainable parameters and weights are all 1.
- After replacing all these operator, the **inspector** will gives a message like this:

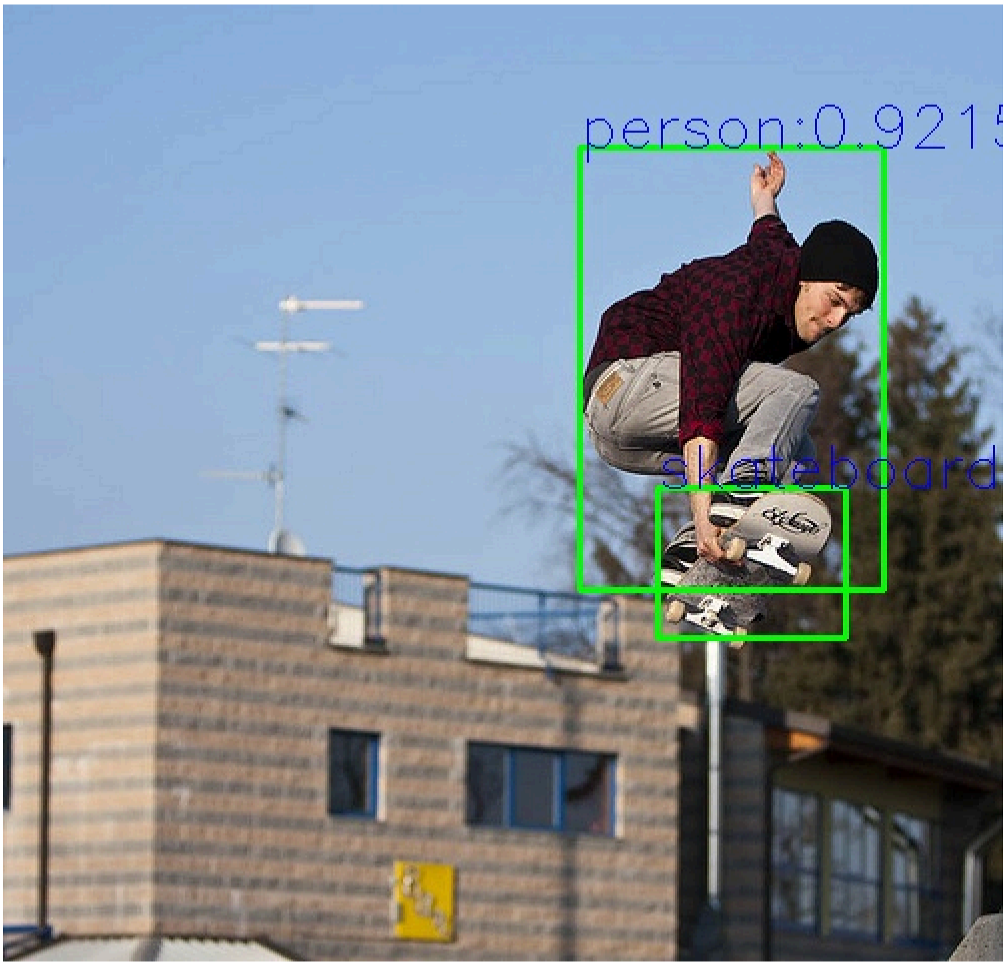
```
[VAIQ_NOTE]: All the operators are assigned to the DPU(see more details in 'Deployment/inspection_results/inspect_DPUCZDX8G_ISA1_B4096.txt')
[VAIQ_NOTE]: Dot image is generated.(Deployment/inspection_results/inspect_DPUCZDX8G_ISA1_B4096.svg)
[VAIQ_NOTE]: =>Finish inspecting.
```

- Then the inspector will give an svg image where we can see all the operators are assigned to DPU and are in **blue** oval.

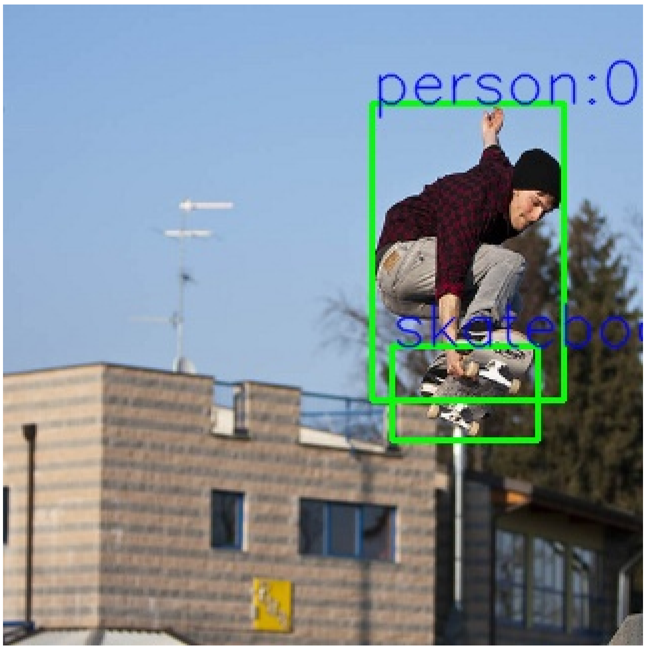


Result:

- From default yolov5 architecture using given pretrained weight, plotted on original image **where input images was resized using padding to keep aspect ratio same of original and resized image**:



- From **DPU supportable** yolov5 architecture using given pretrained weight, plotted on resized image **where input image and plotted image was resized using opencv without considering its aspect ratio**:



Quantization and Compilation:

- Quantized and compiled yolov5 with **Vitis-AI** quantizer and compiler.
- During quantization, if there is no error in quantization code and in model, then we can see the logs like this:

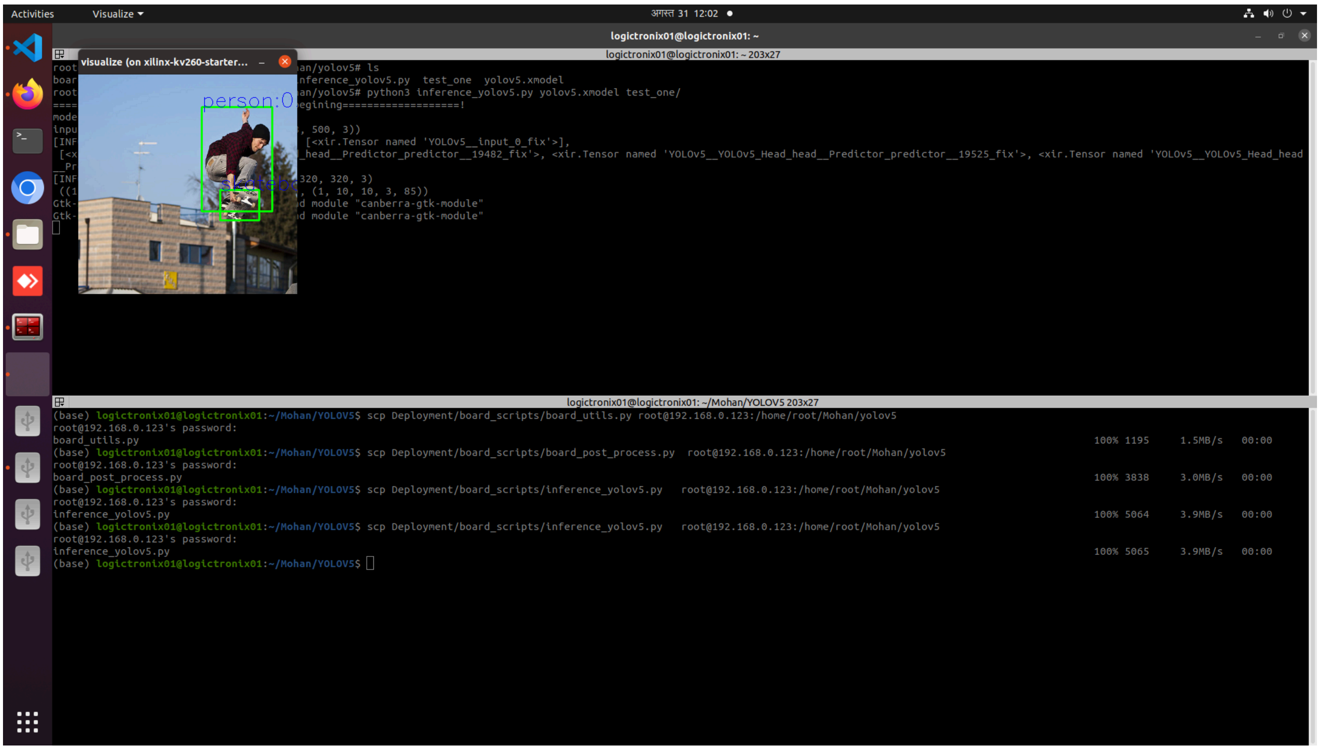
```
(vitis-ai-pytorch) vitis-ai-user@logictronix01:/mohan_env/YOLOV5$ python3 Deployment/yolo_quantization.py --mode calib --calib_data /calib_data/

[VAIQ_NOTE]: Loading NNDCT kernels...
loading annotations into memory...
Done (t=0.44s)
creating index...
index created!
[INFO] The len of wts keys in filtered wts and own model is : (369, 381)

[VAIQ_NOTE]: OS and CPU information:
               system --- Linux
                 node --- logictronix01
              release --- 5.15.0-134-generic
              version --- #145~20.04.1-Ubuntu SMP Mon Feb 17 13:27:16 UTC 2025
              machine --- x86_64
            processor --- x86_64

[VAIQ_NOTE]: Tools version information:
                  GCC --- GCC 9.4.0
               python --- 3.7.12
              pytorch --- 1.12.1
        vai_q_pytorch --- 3.0.0+a44284e+torch1.12.1

[VAIQ_NOTE]: GPU information:
          device name --- NVIDIA GeForce RTX 2070 SUPER
     device available --- True
         device count --- 1
       current device --- 0

[VAIQ_NOTE]: Quant config file is empty, use default quant configuration

[VAIQ_NOTE]: Quantization calibration process start up...

[VAIQ_NOTE]: =>Quant Module is in 'cuda'.

[VAIQ_NOTE]: =>Parsing YOLOv5...

[VAIQ_NOTE]: Start to trace and freeze model...

[VAIQ_NOTE]: The input model YOLOv5 is torch.nn.Module.

[VAIQ_NOTE]: Finish tracing.

[VAIQ_NOTE]: Processing ops...
| 241/241 [00:00<00:00, 3347.79it/s, OpInfo: name = return_0, type = Return]

[VAIQ_NOTE]: =>Doing weights equalization...

[VAIQ_NOTE]: =>Quantizable module is generated.(Deployment/quantized_result/YOLOv5.py)

[VAIQ_NOTE]: =>Get module with quantization.

[VAIQ_NOTE]: OS and CPU information:
               system --- Linux
```

- And after successful quantization and compilation, we will get a compiled model **(.xmodel)** whose **number of DPU subgraphs are 1**.

```
[VAIQ_NOTE]: =>Successfully convert 'YOLOv5' to xmodel.(Deployment/quantized_result/YOLOv5_int.xmodel)
(vitis-ai-pytorch) vitis-ai-user@logictronix01:/mohan_env/YOLOV5$ vai_c_xir --xmodel Deployment/quantized_result/YOLOv5_int.xmodel --arch Deployment/target_name.json --output_dir Deployment/compiled_resu
lts --net_name "yolov5"
**************************************************
* VITIS_AI Compilation - Xilinx Inc.
**************************************************
[UNILOG][INFO] Compile mode: dpu
[UNILOG][INFO] Debug mode: null
[UNILOG][INFO] Target architecture: DPUCZDX8G_ISA1_B4096
[UNILOG][INFO] Graph name: YOLOv5, with op num: 575
[UNILOG][INFO] Begin to compile...
[UNILOG][INFO] Total device subgraph number 5, DPU subgraph number 1
[UNILOG][INFO] Compile done.
[UNILOG][INFO] The meta json is saved to "/mohan_env/YOLOV5/Deployment/compiled_results/meta.json"
[UNILOG][INFO] The compiled xmodel is saved to "/mohan_env/YOLOV5/Deployment/compiled_results/yolov5.xmodel"
[UNILOG][INFO] The compiled xmodel's md5sum is 57c57cc67b36a5629b6792a55289201d, and has been saved to "/mohan_env/YOLOV5/Deployment/compiled_results/md5sum.txt"
(vitis-ai-pytorch) vitis-ai-user@logictronix01:/mohan_env/YOLOV5$
```

Inference on board:

- After writing inference script to run model on DPU, and preprocessing and post processing on CPU which is independent of framework i.e. pytorch, successfully run on board and got output.



References:

[1] Yolov5 used to implement on board: yolov5 ref repo

[2] To understand about yolov5: Medium

[3] Overview of yolov5 architecture: yolov5 overview

[4] Yolov5 ultralytics: original repo

[5] Vitis-AI tutorial for board inference: vitis-ai-tutorial

[6] To write inference script for board: vitis-ai resnet50 demo