

Lab1: Start with Pynq DPU 2025

This document assumes that you have installed beforehand the “DPU on Pynq”. The documentation and steps are at GitHub. <https://github.com/Xilinx/DPU-PYNQ>. At the repository section (link available in moodle), you have the image containing Pynq 3.0.1 and Pynq DPU 2.5 (HPA4ML_DPU_FINN_image.7z)

The AMD-Xilinx DPU (Deep Learning Processor Unit) is a programmable engine optimized for deep neural networks. It is a group of parametrizable IP cores pre-implemented on the hardware with no place and route required. The DPU is released with the Vitis AI specialized instruction set, allowing efficient implementation of many deep learning networks.

In this lab, we will start interacting with the DPU using pretrained and compiled deep learning models. In the next lab (lab 2), we will compile our model (generate .xmodel files).

Setup. Explore the DPU notebook folder

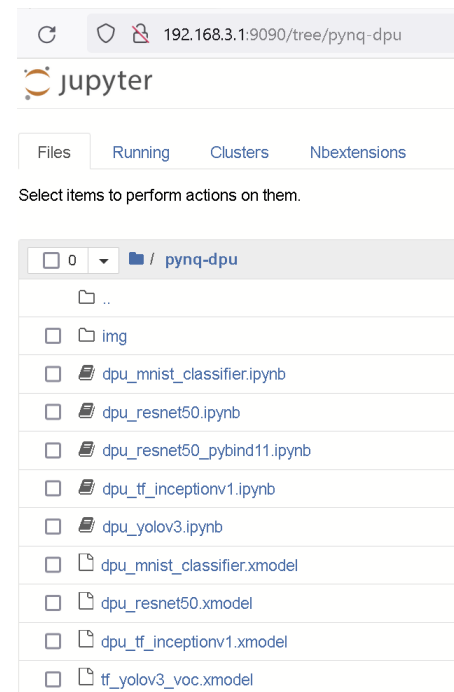
The board setup is explained in “Lab0: Start with Ultra96v2 and Pynq”. Ensure that you understand all the steps.

You can list the jupyter_notebook/pynq_dpu folder or simply browse into the Jupyter notebook and find the same information.

```
(pynq-venv) xilinx@pynq:/$ sudo su
[sudo] password for xilinx:
root@pynq:/# cd /home/xilinx/jupyter_notebooks/pynq-dpu/
root@pynq:/home/xilinx/jupyter_notebooks/pynq-dpu# ls
dpu_mnist_classifier.ipynb  dpu_resnet50.xmodel  img
dpu_mnist_classifier.xmodel  dpu_tf_inceptionv1.ipynb  tf_yolov3_voc.xmodel
dpu_resnet50.ipynb  dpu_tf_inceptionv1.xmodel
dpu_resnet50_pybind11.ipynb  dpu_yolov3.ipynb
root@pynq:/home/xilinx/jupyter_notebooks/pynq-dpu# ls -l
total 100640
-rw-r--r-- 1 root root 37019 Jan 6 20:26 dpu_mnist_classifier.ipynb
-rw-r--r-- 1 root root 778385 Jan 6 17:44 dpu_mnist_classifier.xmodel
-rw-r--r-- 1 root root 199985 Jan 6 17:44 dpu_resnet50.ipynb
-rw-r--r-- 1 root root 655582 Jan 6 17:44 dpu_resnet50_pybind11.ipynb
-rw-r--r-- 1 root root 27539805 Jan 6 17:44 dpu_resnet50.xmodel
-rw-r--r-- 1 root root 199954 Jan 6 18:42 dpu_tf_inceptionv1.ipynb
-rw-r--r-- 1 root root 7410095 Jan 6 17:44 dpu_tf_inceptionv1.xmodel
-rw-r--r-- 1 root root 169209 Jan 6 17:44 dpu_yolov3.ipynb
drwxr-xr-x 2 root root 4096 Jan 6 18:19 img
-rw-r--r-- 1 root root 66037434 Jan 6 17:44 tf_yolov3_voc.xmodel
root@pynq:/home/xilinx/jupyter_notebooks/pynq-dpu#
```

Observe, there are:

- Five notebooks (.ipynb)
- Four programs for the DPU (.xmodel)
- Img folder containing four images.



Exercise 1: Evaluate and interact with dpu_tf_inceptionv1.ipynb

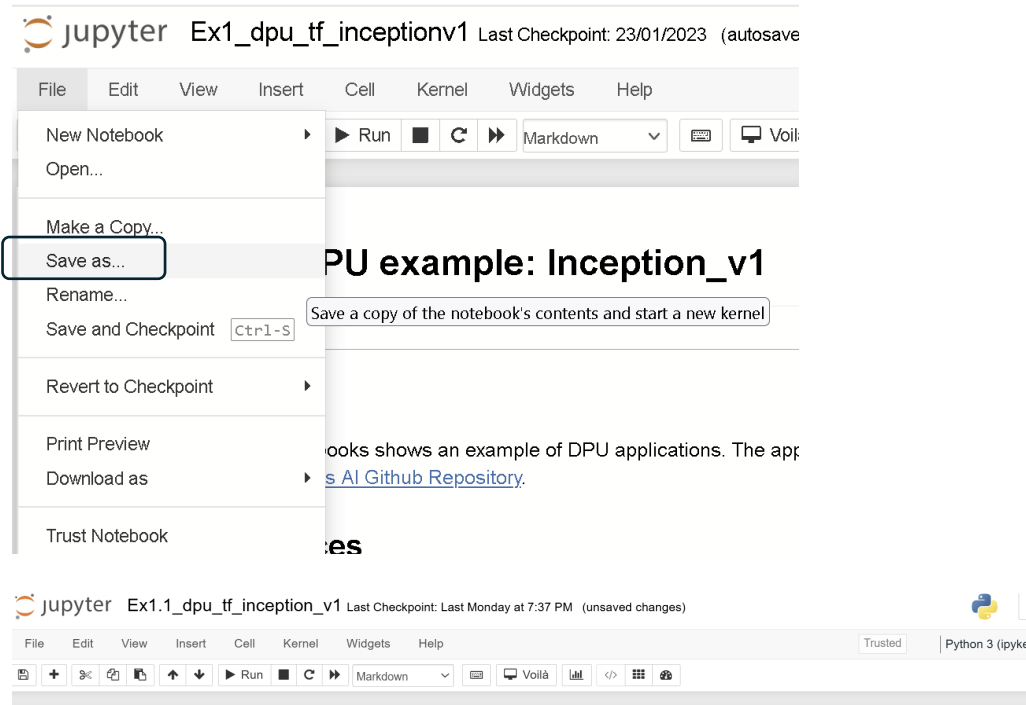
This notebook shows how to use **Python API** to run DPU tasks. You can read more over Inception (GooLeNet) CNN classifier here:

<https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202>

<https://medium.com/coinmonks/paper-review-of-googlenet-inception-v1-winner-of-ilsvlc-2014-image-classification-c2b3565a64e7>

1.1 Copy the Notebook

Make a copy of the Jupyter notebook in a new one called Ex1.1_dpu_tf_inception_v1.ipynb in order to work with. You will introduce some modifications and answer questions at the end of the notebook.



DPU example: Inception_v1 (Ex1.1)

Aim/s

- This notebook shows an example of DPU applications. The application, as well as the DPU IP, is pulled from the official [Vitis AI Github Repository](#).

References

- [Vitis AI Github Repository](#).

Last revised

1.2 Run and interact with the notebook

- A. Observe how the DPU is initialized.
- B. In a simple execution, look at the steps necessary to run an inference.
- C. Change the image that you are recognizing.
- D. Run the “multiple RUN application”. Note that same image is recognized multiple times

1.3 Questions about this exercise

Questions to be answered at the end of the notebook of exercise 1.1:

- A. How do you download the DPU to the FPGA?
- B. What is an “overlay” in the Pynq context?
- C. What is VART in this context?
- D. How do you program the DPU for a specific machine learning model?
- E. In multiple executions, how many images/second are recognized?
- F. How are the input and output tensor shapes?

Exercise 2: Modify the dpu_tf_inception1.ipynb

We will add some modifications to the original example.

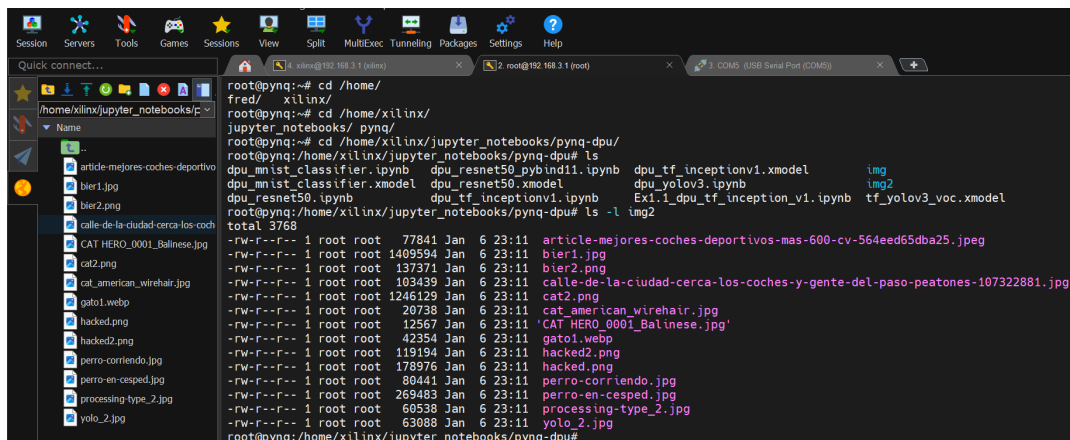
2.1 Copy the Notebook

Make a copy of the notebook in a new one called Ex1.2_dpu_inception_v1.ipynb in order to work with. You will introduce some modifications and answer questions at the end of the notebook.

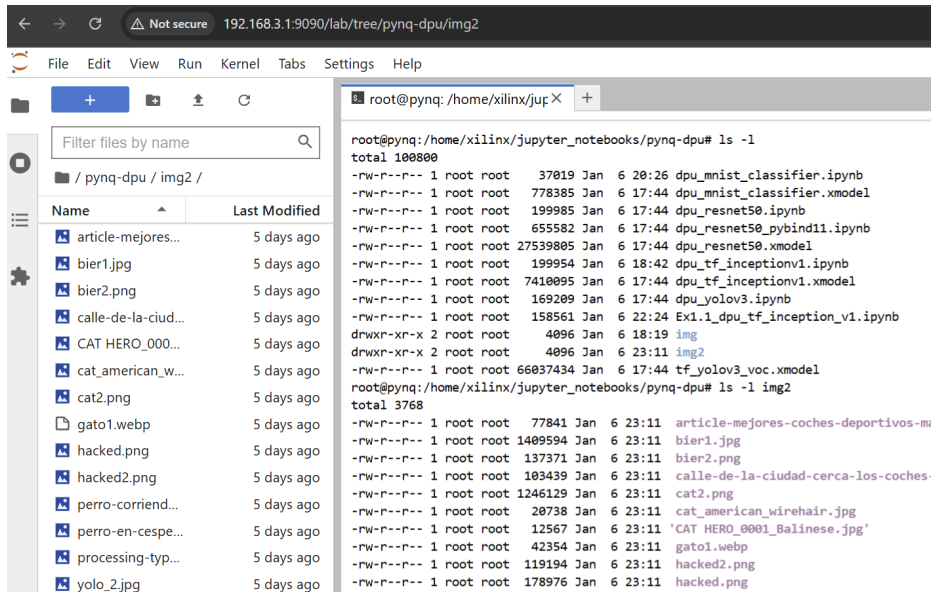
2.2 generate a new “img2” folder with additional jpeg images.

You can copy previous images but add at least 4 extra images.
Validate the results.

Hint 1: (use ssh session in mobaxterm or jupyterlab to create the folder and copy files):ls



```
root@pynq:~# cd /home/fred/ xilinx/
root@pynq:~# cd /home/xilinx/
jupyter_notebooks/ pynq/
root@pynq:~# cd /home/xilinx/jupyter_notebooks/pynq-dpu/
root@pynq:/home/xilinx/jupyter_notebooks/pynq-dpu# ls
dpu_mnist_classifier.ipynb  dpu_resnet50_pybind11.ipynb  dpu_tf_inceptionv1.xmodel  img
dpu_mnist_classifier.xmodel  dpu_resnet50.xmodel          dpu_yolov3.ipynb          img2
dpu_resnet50.ipynb          dpu_tf_inceptionv1.ipynb    Ex1.1_dpu_tf_inception_v1.ipynb  tf_yolov3_voc.xmodel
root@pynq:/home/xilinx/jupyter_notebooks/pynq-dpu# ls -l img2
total 3768
-rw-r--r-- 1 root root 77841 Jan 6 23:11 article-mejores-coches-deportivos-mas-600-cv-564eed65dba25.jpeg
-rw-r--r-- 1 root root 1409594 Jan 6 23:11 bier1.jpg
-rw-r--r-- 1 root root 137371 Jan 6 23:11 bier2.png
-rw-r--r-- 1 root root 103439 Jan 6 23:11 calle-de-la-ciudad-cerca-los-coches-y-gente-del-paso-peatones-107322881.jpg
-rw-r--r-- 1 root root 1246129 Jan 6 23:11 cat2.png
-rw-r--r-- 1 root root 20738 Jan 6 23:11 cat_american_wirehair.jpg
-rw-r--r-- 1 root root 12567 Jan 6 23:11 'CAT_HERO_0001_Balinese.jpg'
-rw-r--r-- 1 root root 42354 Jan 6 23:11 gato1.webp
-rw-r--r-- 1 root root 119194 Jan 6 23:11 hacked2.png
-rw-r--r-- 1 root root 178976 Jan 6 23:11 hacked.png
-rw-r--r-- 1 root root 80441 Jan 6 23:11 perro-corriendo.jpg
-rw-r--r-- 1 root root 269483 Jan 6 23:11 perro-en-cesped.jpg
-rw-r--r-- 1 root root 60538 Jan 6 23:11 processing-type_2.jpg
-rw-r--r-- 1 root root 63088 Jan 6 23:11 yolo_2.jpg
root@pynq:/home/xilinx/jupyter_notebooks/pynq-dpu#
```



Hint 2: (use previous defined function “run”) and change folder:

```
image_folder = 'img2'
original_images = [i for i in os.listdir(image_folder) if i.endswith((".png", ".jpg", ".jpeg", ".gif", ".bmp", "webp"))]
total_images = len(original_images)
```

Check that you recognize the images:

```
print("Recognizing all images in '", image_folder, "' Folder\n")
for i in range(total_images):
    run(i, True)
```

Recognizing all images in ' img2 ' Folder

Classification: tabby, tabby cat

Classification: English foxhound

Classification: golden retriever

Classification: malamute, malemute, Alaskan malamute

Classification: tiger cat

Classification: beer glass

Classification: Siamese cat, Siamese

Classification: beer glass

2.3 Calculate the time on pre-processing, inference, and post-processing.

For one image (or all the images) discriminate the time spent on preprocessing, post-processing and inference itself. The idea is recognizing where the time is spent in the computation.

Hint: divide the execution into functions and add timers in between

```
def pre_process(image_index):
    path = os.path.join(image_folder, original_images[image_index])
    img = cv2.imread(path)
    preprocessed = preprocess_fn(img)
    image[0,...] = preprocessed.reshape(shapeIn[1:])

def exec_dpu():
    job_id = dpu.execute_async(input_data, output_data)
    dpu.wait(job_id)

def post_proc():
    temp = [j.reshape(1, outputSize) for j in output_data]
    softmax = calculate_softmax(temp[0][0])
    print("Classification: {}".format(predict_label(softmax)))
```

```
ind = 1
t_start = time.time()
pre_process(ind)
t_proc = time.time()
exec_dpu()
t_dpu = time.time()
post_proc()
t_end = time.time()
```

Print time and analyze the results. For example:

Classification: tabby, tabby cat	Classification: English foxhound
Pre proc: 0.021371 49.88 %	Pre proc: 0.137526 86.72 %
DPU proc: 0.019592 45.73 %	DPU proc: 0.019098 12.04 %
Pos proc: 0.019592 4.39 %	Pos proc: 0.019098 1.23 %
Total Ex: 0.042846 i.e. 23.3392 fps	Total Ex: 0.158580 i.e. 6.3060 fps

2.5 (optional) capture images from webcam

You can capture an image from a webcam and recognize the content of the picture

Hint: in /common/ folder there are two notebooks for inspiration (display_port_introduction.ipynb and usb_webcam.ipyn)

```
from PIL import Image as PIL_Image
from pynq.lib.video import *
```

```
import os
import cv2
import matplotlib.pyplot as plt

capture = cv2.VideoCapture(0)

capture.set(3, 1280)
capture.set(4, 720)
```

```
if capture.isOpened():
    ret, frame = capture.read()
    if ret:
        _, ax = plt.subplots(1)
        _ = ax.imshow(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))
    else:
        print("frame not captured")
else:
    print("cannot open camera")
```

```
preprocessed = preprocess_fn(frame)
image[0,...] = preprocessed.reshape(shapeIn[1:])
exec_dpu()
post_proc()
```

```
from PIL import Image as PIL_Image

orig_img_path = 'webcam.jpg'
!fswebcam --no-banner --save {orig_img_path} -d /dev/video0 2> /dev/null

img = cv2.imread(orig_img_path)
preprocessed = preprocess_fn(img)
image[0,...] = preprocessed.reshape(shapeIn[1:])
exec_dpu()
post_proc()

img = PIL_Image.open(orig_img_path)
img
```

Exercise 3: Interact with `dpu_mnist_classifier`

This notebook shows how to deploy a Convolutional Neural Network (CNN) model for hand-written digit recognition. The network was trained on the well-known MNIST dataset.

3.1 Copy the Notebook

Make a copy of the notebook in a new one called `Ex1.3_dpu_mnist_classifier.ipynb` in order to work with. You will introduce some modifications and answer questions at the end of the notebook.

3.2 Connect to internet

You need to be connected to the internet to download mnist data.
You can use the Wi-Fi notebook from `/common/wifi.ipynb`

3.3 Run the notebook and answer the following Questions

- A. How are the input and output tensor shapes?
- B. Run the classification of 10.000 digits and inform the accuracy and Throughput.

3.4 Compare accuracy and execution time with FINN

FINN is a machine learning framework by the Integrated Communications and AI Lab of AMD Research & Advanced Development. It is not a generic DNN acceleration solution but relies on co-design and design space exploration for quantization and parallelization tuning so as to optimize a solutions with respect to resource and performance requirements. We will see more details during the course.

Run the `/finn_examples/0_mnist_with_fc_networks.ipynb` notebook and report

- A. Run the classification of 1000 digits and inform the accuracy and Throughput.
- B. Compare the numbers

DPU	FINN
Classifying 10000 digit pictures ... Overall accuracy: 0.9881 Execution time: 5.1715s Throughput: 1933.6589FPS	Final accuracy: 92.93% 436227.237782 images per second including data movement

Exercise 4: Interact dpu_resnet50 and dpu_resnet50_pybind

An Overview of ResNet and its Variants can be found here:

<https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035>

These two notebooks shows the “pure” python approach (dpu_resnet50.ipynb) and the (dpu_resnet50_pybind11.ipynb) shows how C++ API looks like.

In the seconds notebook (dpu_resnet50_pybind11.ipynb) it shows how to leverage **Pybind11** to call **C++** VART API. I.E. How to communicate with the DPU using C++ instead of python API (We will not use C++ API in next labs).

4.1 Copy the Notebooks

Make a copy of the Jupyter notebooks. Rename the first one as Ex1.4a_dpu_resnet50.ipynb and the second one as Ex1.4b_dpu_resnet50_pybind.ipynb in order to work with. You will introduce some modifications and answer questions at the end of the notebook.

4.2 Run and interact with the Python Notebook.

A. Run the pure python notebook. There is no big difference with other labs examples. This resnet50 model was trained to recognize the same classes as inception_v1.

As a reference, this table from Xilinx expresses the complexity in terms of GOPS y max FPS for different models

Network Model	Workload (GOPs per image)	Input Image Resolution	Accuracy (DPUCZDX8G)	Frames per second (FPS)
Inception-v1	3.16	224*224	Top-1: 0.6984	472.5
ResNet50	7.7	224*224	Top-1: 0.7334	194.1
MobileNet_v2	0.59	224*224	Top-1: 0.6349	747.3
YOLO-V3-VOC	65.42	416*416	mAP: 0.8127	34.7
YOLO-V3_ADAS_	5.46	512*256	mAP: 0.5305	272.6

Fig1. Source <https://docs.xilinx.com/r/en-US/pg338-dpu/Xilinx-Model-Zoo-Performance>

B. As in Exercise2 (you can reuse the code) “Calculate the time in pre-processing, inference and - post processing”. Compare with inception_v1 results

Resnet50	Inception_v1 (ex1.2)
Classification: cougar, puma, catamount, mountain lic	Classification: tabby, tabby cat
Pre proc: 0.020329 34.73 %	Pre proc: 0.021371 49.88 %
DPU proc: 0.036300 62.02 %	DPU proc: 0.019592 45.73 %
Pos proc: 0.036300 3.24 %	Pos proc: 0.019592 4.39 %
Total Ex: 0.058526 i.e. 17.0863 fps	Total Ex: 0.042846 i.e. 23.3392 fps

4.4 Run and interact with the notebook C++ code

- Observe how the DPU is initialized. There is no difference with other labs examples.
- Observe the cell that starts with “%%pybind11 resnet50;{cflags};{ldflags}”. Try to understand what the functions do.
- Compile the C++ code into a shared object (RUN the cell). **The compilation can take up to 30 seconds; please be patient.** During compilation, several files are generated in the root folder and deleted when it finishes.
- Run the last cell and understand the generated results.

4.5 Measure the time

Copy last cell and add timer to the called lines of python code

```
#Init timer
with sys_pipes():
    resnet50.run(img_folder)
#stop timer
#Report time.
```

```
from wurlitzer import sys_pipes
import resnet50
import time

# get the start time
st = time.time()

with sys_pipes():
    resnet50.run(xmodel_file,img_folder)

# get the end time
et = time.time()

# get the execution time
elapsed_time = et - st
print('\n Total Python Execution time:', elapsed_time, 'seconds (for ', len(image_files), ' images
```

Total Python Execution time: 0.9769301414489746 seconds (for 4 images)

4.6 (optional) add timers inside the c++ code

Add timers and print inside the C++ code. The aim is to measure strictly the time spent in the inference process.

```
int run(char* XModel, string baseImagePath) {
    GraphInfo shapes;
    auto graph = xir::Graph::deserialize(XModel);
    auto subgraph = get_dpu_subgraph(graph.get());
    auto runner = vart::Runner::create_runner(subgraph[0], "run");
    auto inputTensors = runner->get_input_tensors();
    auto outputTensors = runner->get_output_tensors();

    //Added for measure timing
    clock_t start;
    double diff;
    start = clock();

    int inputCnt = inputTensors.size();
    int outputCnt = outputTensors.size();
    TensorShape inshapes[inputCnt];
    TensorShape outshapes[outputCnt];
    shapes.inTensorList = inshapes;
    shapes.outTensorList = outshapes;
    getTensorShape((void*)(runner.get()), &shapes, inputCnt, outputCnt);

    runResnet50((void*)(runner.get()), baseImagePath, &shapes);

    diff = ( std::clock() - start ) / (double)CLOCKS_PER_SEC;
    cout<<"\nTotal C++ computation time: "<< diff <<'\n';
    return 0;
}
```

Note: It is necessary to stop the notebook and recompile.

Total C++ computation time: 0.095417

Total Python Execution time: 1.1661875247955322 seconds (for 4 images)

4.7 Questions of this exercise

Questions to be answered at the end of the notebook of exercise 4b:

- A. "from Wurlitzer import sys_pipes" for what is used?
- B. When you measure the total time in 4.3. What do you obtain? It seems that is slower than pure python code. Does it make sense? Why do you think it is slower than pure python
- C. Why the difference between adding DPU Task executions and total time?

Exercise 5: Interact with YOLO V3

You only look once (YOLO) is a very well-known, real-time object detection system. This Notebook shows how to run a YOLO network based application for object detection.

5.1 Copy the Notebook

Make a copy of the notebook in a new one called Ex1.5_dpu_yolo_v3.ipynb in order to work with. You will introduce some modifications and answer questions at the end of the notebook.

5.2 Run and interact with the notebook

Run the notebook and see results
See how bounding boxes are generated

Change the images and review results for the other images in “img” folder
bellpeppe-994958.JPEG
greyfox-672194.JPEG
irishterrier-696543.JPEG
jinrikisha-911722.JPEG

Add more significant images (at least 2) with several objects in a new folder “img3”

5.3 (optional 1) Calculate the time in pre-processing, inference and - post processing

Modify the run process (rename for example as run2) adding timers to calculate spent time each part of the calculus.

```
run2(0, display=True)
```

```
Pre proc: 0.07204818725585938  
DPU Proc: 0.27930760383605957  
Pos proc: 0.044100284576416016  
Total Ex: 0.39545607566833496    i.e. 2.52872584726373 fps  
Time to display: 0.14720702171325684  
Number of detected objects: 5
```

5.4 (optional 2) Capture the image from webcam

Modify the notebook to be able to capture from the webcam and process the bounding boxes.

Hint: you have an example of a snapshot here:

http://192.168.3.1/notebooks/common/usb_webcam.ipynb

There is a different example code here:

http://192.168.3.1:9090/notebooks/common/display_port_introduction.ipynb

```

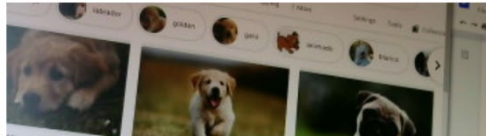
from PIL import Image as PIL_Image

orig_img_path = '/home/xilinx/jupyter_notebooks/common/data/webcam.jpg'
!fswebcam --no-banner --save {orig_img_path} -d /dev/video0 2> /dev/null

img = PIL_Image.open(orig_img_path)

image = cv2.imread(orig_img_path)|
img

```



5.5 Questions of this exercise

Questions to be answered at the end of the notebook of exercise 4:

- How many classes of objects recognize this implementation?
- How many boxes are detected on "jinrikisha-911722.JPEG"?
- Explain briefly the steps to use the DPU after programming using Python.

6. Deliverables

A zip file with modified notebooks and upload to moodle