PYNQ DPU Overlay on KRIA for AI Inference

Install PYNQ-DPU Overlay

- KV260

1. Download [Ubuntu 20.04]and put it in SDcard through [balenaetcher]
2. Boot Ubuntu 20.04 from KV260
3. Update Ubuntu package
   - sudo apt update
   - sudo apt upgrade
4. Install Xilinx system management snap package ---> sudo snap install xlnx-config --classic –channel=1.x

```
ubuntu@kria:~$ sudo snap install xlnx-config --classic
error: cannot perform the following tasks:
- Run install hook of "xlnx-config" snap if present (run hook "install":
-----
* Release: focal
*************** Running xlnx-config install hook ******************
* Release: focal
You are attempting to install the 22.04 Jammy version of xlnx-config on an incom
patible release
Please choose the proper xlnx-config 1.0 track:
   20.04 Focal:   sudo snap install xlnx-config --classic --channel=1.x

Exiting the snap installer
-----)
ubuntu@kria:~$
```

Install PYNQ-DPU Overlay

- KV260

5. Download the full package from Xilinx [DPU-PYNQ]() github

```
ubuntu@kria:~$ git clone https://github.com/Xilinx/Kria-PYNQ.git
Cloning into 'Kria-PYNQ'...
remote: Enumerating objects: 93, done.
remote: Counting objects: 100% (47/47), done.
remote: Compressing objects: 100% (37/37), done.
remote: Total 93 (delta 27), reused 14 (delta 10), pack-reused 46
Unpacking objects: 100% (93/93), 1.26 MiB | 2.38 MiB/s, done.
```

6. Install PYNQ package

```
ubuntu@kria:~/Kria-PYNQ$ sudo bash install.sh -b KV260

This version of Kria-PYNQ is not compatible with Ubuntu 20.04 please checkout ta
g v1.0 with the command

                git checkout tags/v1.0
```

Install PYNQ-DPU Overlay

- KV260

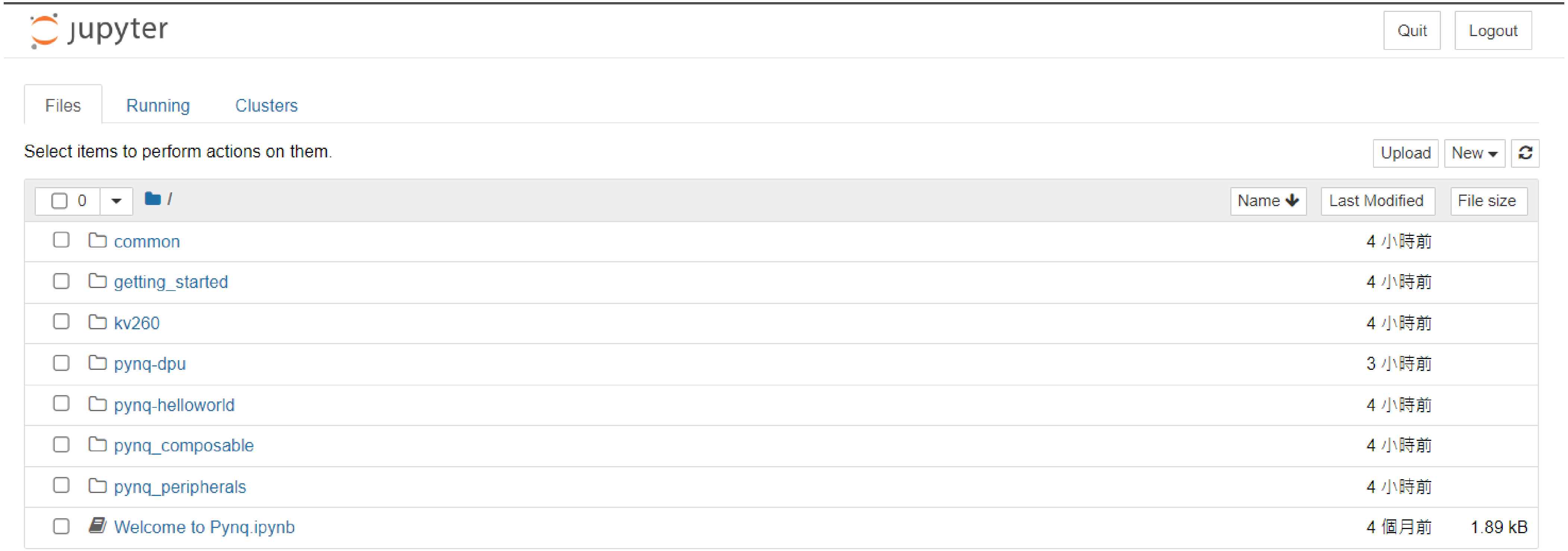
7. PYNQ-DPU has been installed successfully

```
You should consider upgrading via the '/usr/local/share/pynq-venv/bin/python3 -m pip install --upgrade pip' com
mand.
/usr/local/share/pynq-venv/lib/python3.8/site-packages/pynq/pl_server/xrt_device.py:59: UserWarning: xbutil fai
led to run - unable to determine XRT version
  warnings.warn("xbutil failed to run - unable to determine XRT version")
The following notebooks modules will be delivered:
- pynq_peripherals (source: pynq_peripherals)
- pynq-helloworld (source: pynq_helloworld)
- pynq-dpu (source: pynq_dpu)
- pynq_composable (source: pynq_composable)
- kv260 (source: kv260)
Do you want to proceed? [Y/n] Delivering notebooks '/home/root/jupyter_notebooks/pynq_peripherals'...
Delivering notebooks '/home/root/jupyter_notebooks/pynq-helloworld'...
Downloading file 'resizer.hwh'. This may take a while...
Downloading file 'resizer.bit'. This may take a while...
Delivering notebooks '/home/root/jupyter_notebooks/pynq-dpu'...
Delivering notebooks '/home/root/jupyter_notebooks/pynq_composable'...
Delivering notebooks '/home/root/jupyter_notebooks/kv260'...
PYNQ Installation completed.

To continue with the PYNQ experience, connect to JupyterLab via a web browser using this url: 192.168.1.52:9090
/lab or kria:9090/lab - The password is xilinx

ubuntu@kria:~/Kria-PYNQ$
```

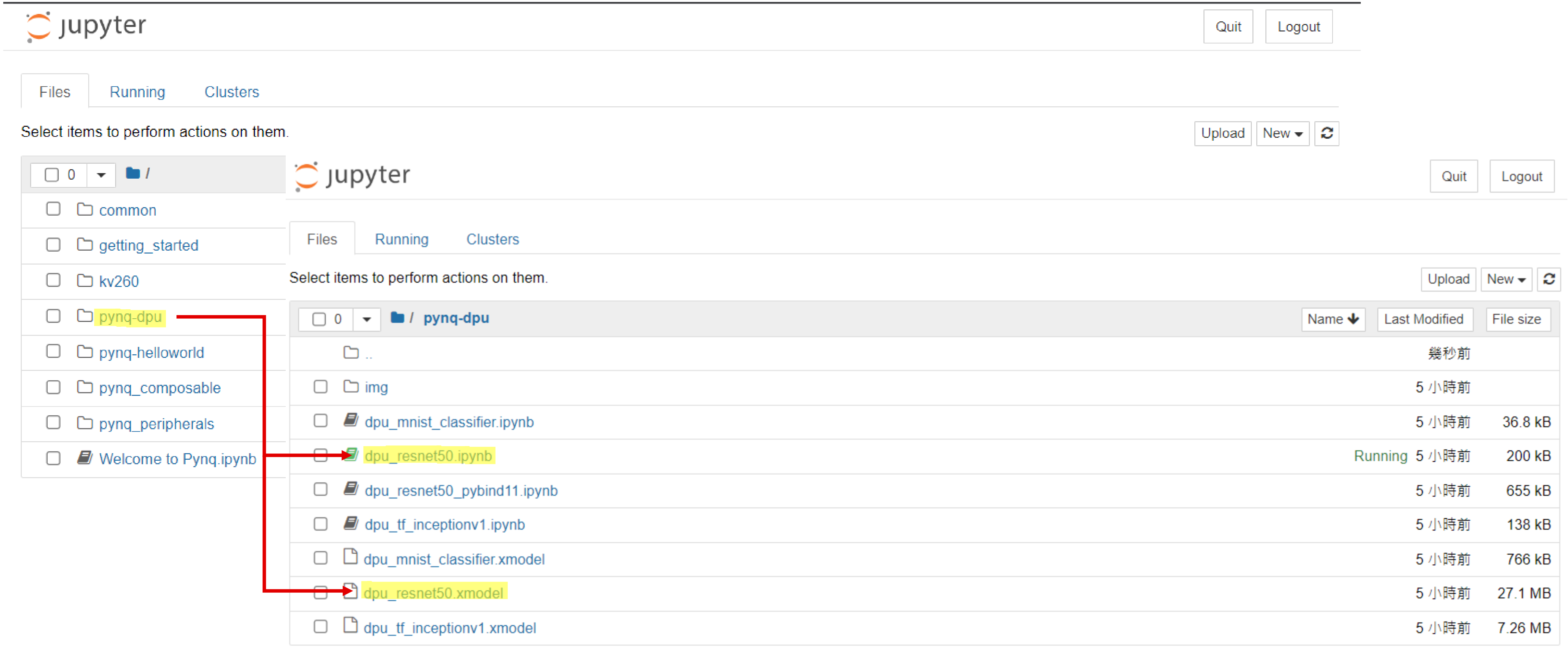
Install PYNQ-DPU Overlay

- KV260

8. Open Jupyter notebook on web browser



PYNQ-DPU Quick Start



PYNQ-DPU Quick Start

- dpu_resnet50.ipynb

DPU example: Resnet50

Aim/s

- This notebooks 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

- Mar 3, 2021
  - Initial revision
- Dec 17, 2021
  - Calling load_model after importing cv2 to avoid memory allocation issues on KV260

1. Prepare the overlay

We will download the overlay onto the board.

In [1]:
```
from pynq_dpu import DpuOverlay
overlay = DpuOverlay("dpu.bit")
```

2. Utility functions

In this section, we will prepare a few functions for later use.

In [2]:
```
import os
import time
import numpy as np
import cv2
import matplotlib.pyplot as plt
%matplotlib inline
```

PYNQ-DPU Quick Start

- dpu_resnet50.ipynb

The `load_model()` method will automatically prepare the `graph` which is used by VART.

Note For the KV260 board you may see TLS memory allocation errors if cv2 gets loaded before loading the vitis libraries in the Jupyter Lab environment. Make sure to load cv2 first in these cases.

In [3]:
```
overlay.load_model("dpu_resnet50.xmodel")
```

Let's first define a few useful preprocessing functions. These functions will make sure the DPU can take input images with arbitrary sizes.

In [4]:
```
_R_MEAN = 123.68
_G_MEAN = 116.78
_B_MEAN = 103.94

MEANS = [_B_MEAN,_G_MEAN,_R_MEAN]

def resize_shortest_edge(image, size):
    H, W = image.shape[:2]
    if H >= W:
        nW = size
        nH = int(float(H)/W * size)
    else:
        nH = size
        nW = int(float(W)/H * size)
    return cv2.resize(image,(nW,nH))

def mean_image_subtraction(image, means):
    B, G, R = cv2.split(image)
    B = B - means[0]
    G = G - means[1]
    R = R - means[2]
    image = cv2.merge([R, G, B])
    return image

def BGR2RGB(image):
    B, G, R = cv2.split(image)
    image = cv2.merge([R, G, B])
    return image

def central_crop(image, crop_height, crop_width):
    image_height = image.shape[0]
    image_width = image.shape[1]
    offset_height = (image_height - crop_height) // 2
    offset_width = (image_width - crop_width) // 2
    return image[offset_height:offset_height + crop_height, offset_width:
                 offset_width + crop_width, :]
```

PYNQ-DPU Quick Start

- Custom Model - YoloV4-tiny

The `load_model()` method will automatically prepare the `graph` which is used by VART.

Note For the KV260 board you may see TLS memory allocation errors if cv2 gets loaded before loading the vitis libraries in the Jupyter Lab environment. Make sure to load cv2 first in these cases.

In [3]:
```
overlay.load_model("dpu_resnet50.xmodel")
```

Let's first define a few useful preprocessing functions. These functions will make sure the DPU can take input images with arbitrary sizes.

In [4]:
```
_R_MEAN = 123.68
_G_MEAN = 116.78
_B_MEAN = 103.94

MEANS = [_B_MEAN,_G_MEAN,_R_MEAN]

def resize_shortest_edge(image, size):
    H, W = image.shape[:2]
    if H >= W:
        nW = size
        nH = int(float(H)/W * size)
    else:
        nH = size
        nW = int(float(W)/H * size)
    return cv2.resize(image,(nW,nH))

def mean_image_subtraction(image, means):
    B, G, R = cv2.split(image)
    B = B - means[0]
    G = G - means[1]
    R = R - means[2]
    image = cv2.merge([R, G, B])
    return image

def BGR2RGB(image):
    B, G, R = cv2.split(image)
    image = cv2.merge([R, G, B])
    return image

def central_crop(image, crop_height, crop_width):
    image_height = image.shape[0]
    image_width = image.shape[1]
    offset_height = (image_height - crop_height) // 2
    offset_width = (image_width - crop_width) // 2
    return image[offset_height:offset_height + crop_height, offset_width:
                 offset_width + crop_width, :]
```

In [3]:
```
overlay.load_model("pt_yolov4-tiny-0208.xmodel")
```

Let's first define a few useful preprocessing functions.

In [4]:
```
anchor_list = [10,14,  23,27,  37,58,  81,82,  135,169,  344,319]
anchor_float = [float(x) for x in anchor_list]
anchors = np.array(anchor_float).reshape(-1, 2)
print(anchors)
```

```
[[ 10.  14.]
 [ 23.  27.]
 [ 37.  58.]
 [ 81.  82.]
 [135. 169.]
 [344. 319.]]
```

In [5]:
```
'''Get model classification information'''
def get_class(classes_path):
    with open(classes_path) as f:
        class_names = f.readlines()
    class_names = [c.strip() for c in class_names]
    return class_names

classes_path = "img/voc_classes.txt"
class_names = get_class(classes_path)
```


AMD XILINX

PYNQ-DPU Quick Start

- dpu_resnet50.ipynb

3. Use VART

Now we should be able to use VART to do image classification.

In [7]:
```
dpu = overlay.runner

inputTensors = dpu.get_input_tensors()
outputTensors = dpu.get_output_tensors()

shapeIn = tuple(inputTensors[0].dims)
shapeOut = tuple(outputTensors[0].dims)
outputSize = int(outputTensors[0].get_data_size() / shapeIn[0])

softmax = np.empty(outputSize)
```

We can define a few buffers to store input and output data. They will be reused during multiple runs.

In [8]:
```
output_data = [np.empty(shapeOut, dtype=np.float32, order="C")]
input_data = [np.empty(shapeIn, dtype=np.float32, order="C")]
image = input_data[0]
```

Remember that we have a list of `original_images`. We can now define a new function `run()` which takes the image index as the input, and calculate the softmax as the classification result. With the argument `display` set to `True`, the original image as well as the predicted label can be rendered.

It is obvious that the range of `image_index` should be [0, `total_images`-1].

In [9]:
```
def run(image_index, display=False):
    preprocessed = preprocess_fn(cv2.imread(
        os.path.join(image_folder, original_images[image_index])))
    image[0,...] = preprocessed.reshape(shapeIn[1:])
    job_id = dpu.execute_async(input_data, output_data)
    dpu.wait(job_id)
    temp = [j.reshape(1, outputSize) for j in output_data]
    softmax = calculate_softmax(temp[0][0])
    if display:
        display_image = cv2.imread(os.path.join(
            image_folder, original_images[image_index]))
        _, ax = plt.subplots(1)
        _ = ax.imshow(cv2.cvtColor(display_image, cv2.COLOR_BGR2RGB))
        print("Classification: {}".format(predict_label(softmax)))
```

Let's run it for 1 image and print out the predicted label.

In [10]:
```
run(1, display=True)
```

```
Classification: Arctic fox, white fox, Alopex lagopus
```

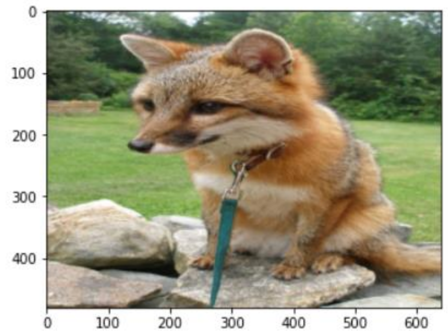
PYNQ-DPU Quick Start

- dpu_resnet50.ipynb

```
In [10]: run(1, display=True)
```

```
Classification: Arctic fox, white fox, Alopex lagopus
```

Result



We can also run it for multiple images as shown below. In this example we have only used 1 thread; in principle, users should be able to boost the performance by employing more threads.

```
In [11]: time1 = time.time()
         [run(i) for i in range(total_images)]
         time2 = time.time()
         fps = total_images/(time2-time1)
         print("Performance: {} FPS".format(fps))
```

```
Performance: 25.196311536959723 FPS
```

We will need to remove references to `vart.Runner` and let Python garbage-collect the unused graph objects. This will make sure we can run other notebooks without any issue.

```
In [12]: del overlay
         del dpu
```


AMD
XILINX

PYNQ-DPU Quick Start

- Custom Model Result - YoloV4-tiny

In [22]:
```
run(1, display=True)
```

```
          [ -0.25,    0.25,    0.25, ...,   -7.5 ,   -9.75,   -6.5 ],
          [  0.5 ,    0.  ,   -1.  , ...,   -6.75,   -7.25,   -4.25]]]],
      dtype=float32)]
Number of detected objects: 7
```

We can also run it for multiple images as shown below. In this example we have only used 1 thread; in principle, users should be able to boost the performance by employing more threads.

Thank you very much for your attention!