Segmentation with ENet on PYNQ

2023/04

Workflow

1. Environment Setup
2. Install Vitis-AI docker
3. Download a model from model-zoo
4. Try a model on GPU
5. Try a model on DPU
6. Python code mapping GPU -> DPU

1. Environment Setup

- PC
  - Ubuntu 18.04 / 20.04 / 22.04
  - NVIDIA GPU
  - Vitis-AI v2.5
- Board:
  - Any PYNQ-capable boards: ZCU104, PYNQ-ZU, KV260, …
  - PYNQ image v3.0 / v3.0.1

2. Install Vitis-AI docker (1)

1. Install NVIDIA driver

```
$ sudo apt install nvidia-driver-520 nvidia-utils-520
```

2. Install Docker: https://docs.docker.com/engine/install/ubuntu/

- Perform Post-Installation: https://docs.docker.com/engine/install/linux-postinstall/
- Reboot system

3. Install NVIDIA Docker Runtime

```
$ curl -s -L https://nvidia.github.io/nvidia-container-runtime/gpgkey  | \
  sudo apt-key add -
$ distribution=$(. /etc/os-release;echo $ID$VERSION_ID)
$ curl -s -L https://nvidia.github.io/nvidia-container-runtime/$distribution/nvidia-container-runtime.list | \
  sudo tee /etc/apt/sources.list.d/nvidia-container-runtime.list
$ sudo apt-get update
$ sudo apt-get install nvidia-container-toolkit nvidia-container-runtime
```

2. Install Vitis-AI docker (2)

4. On PC, open Terminal and run:

```
$ cd ~
$ git clone -b 2.5 https://github.com/Xilinx/Vitis-AI
$ cd Vitis-AI/docker
$ ./docker_build_gpu.sh
```

5. Run Vitis-AI docker

```
$ cd ~/Vitis-AI
$ ./docker_run.sh xilinx/vitis-ai-gpu:latest
```

3. Download a model from model-zoo

After opening Vitis-AI GPU docker, run:

```
Vitis-AI /workspace > cd model_zoo/
Vitis-AI /workspace/model_zoo > python downloader.py
…
tf:tensorflow1.x  tf2:tensorflow2.x  cf:caffe  dk:darknet  pt:pytorch  all: list all model
input:pt
chose model
…
15 : pt_face-mask-detection_512_512_0.59G_2.5
16 : pt_ENet_cityscapes_512_1024_8.6G_2.5
17 : pt_BCC_shanghaitech_800_1000_268.9G_2.5
…
input num:16
chose model type
0: all
1 : GPU
2 : zcu102 & zcu104 & kv260
…
input num:1
pt_ENet_cityscapes_512_1024_8.6G_2.5.zip
                                  100.0%|100%
done
```

4. Try a model on GPU

After downloading model, extract and examine it

```
Vitis-AI /workspace/model_zoo > unzip pt_ENet_cityscapes_512_1024_8.6G_2.5.zip
Vitis-AI /workspace/model_zoo > cd pt_ENet_cityscapes_512_1024_8.6G_2.5
```

Review README

```
$ cat readme.md
```

Activate pytorch environment

```
$ conda activate vitis-ai-pytorch
```

Install dependencies

```
$ pip install --user -r requirements.txt
```

Dataset

- Download **leftImg8bit_trainvaltest.zip [11GB]** from https://www.cityscapes-dataset.com/downloads.
- Put it in data/cityscapes (review in README)

Run Demo

```
$ bash run_demo.sh # The result will be put in data/demo_results.
```

5. Try a model on DPU

https://github.com/haipnh/DPU-PYNQ/blob/enet_cityscapes/pynq_dpu/notebooks/dpu_enet_cityscapes.ipynb

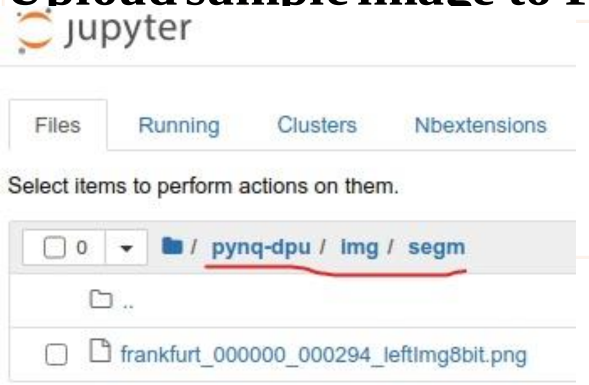
Download pre-trained & quantized model

- review 3. Download a model from model-zoo
- choose 2 : zcu102 & zcu104 & kv260

Upload xmodel to PYNQ

- Extract downloaded file and upload ENet_cityscapes_pt.xmodel to PYNQ board.
- (Optional) Rename it to dpu_enet_cityscapes.xmodel.

Upload sample image to PYNQ



jupyter

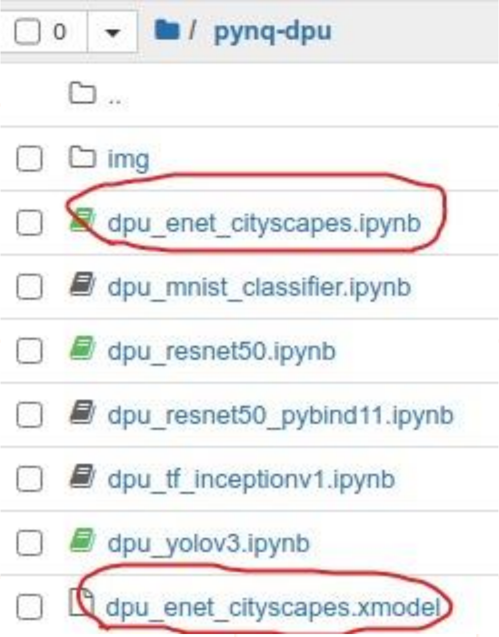
Files | Running | Clusters | Nbextensions

Select items to perform actions on them.

□ 0 ▾ / pynq-dpu / img / segm

..

□ frankfurt_000000_000294_leftImg8bit.png



□ 0 ▾ / pynq-dpu

..

□ img

□ dpu_enet_cityscapes.ipynb

□ dpu_mnist_classifier.ipynb

□ dpu_resnet50.ipynb

□ dpu_resnet50_pybind11.ipynb

□ dpu_tf_inceptionv1.ipynb

□ dpu_yolov3.ipynb

□ dpu_enet_cityscapes.xmodel

Upload example notebook to PYNQ

6. Python code mapping GPU -> DPU (1)

Main of the work was retrieved from

pt_ENet_cityscapes_512_1024_8.6G_2.5/code/test/test.py

Processing Flow:

- **GPU:**

Input: 2048x1024 -> Resize to 1024x512 -> Model -> 1024x512x19 tensor -> max() -> 1024x512 -> convert to uint8 -> interpolate to 2048x1024 output -> putpalette() -> Output: 2048x1024

- **DPU:**

Input: 2048x1024 -> Resize to 1024x512 -> Model -> 1024x512x19 tensor -> max() -> 1024x512 -> convert to uint8 -> ~~interpolate to 2048x1024 output~~ -> putpalette() -> Output: 1024x512

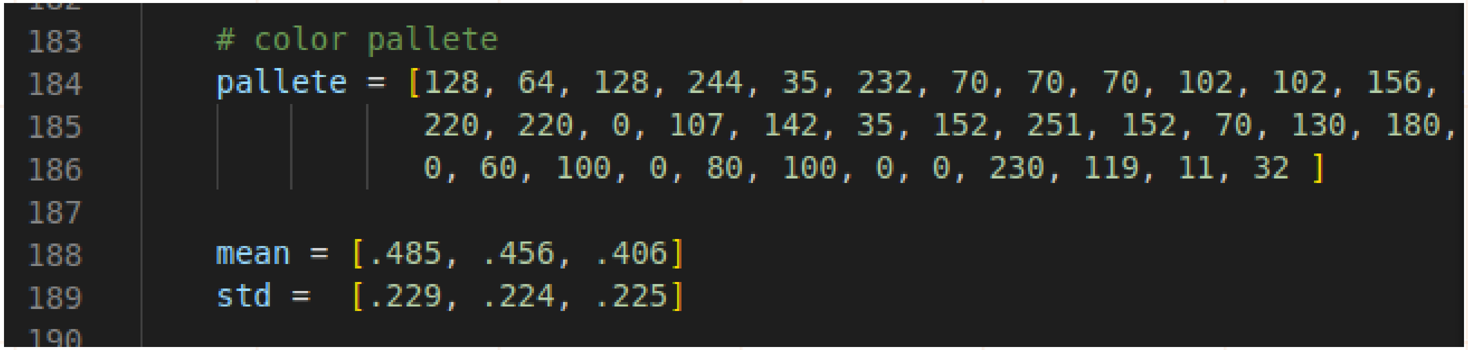
6. Python code mapping GPU -> DPU (2)

Main of the work was retrieved from

pt_ENet_cityscapes_512_1024_8.6G_2.5/code/test/test.py

Pre-process

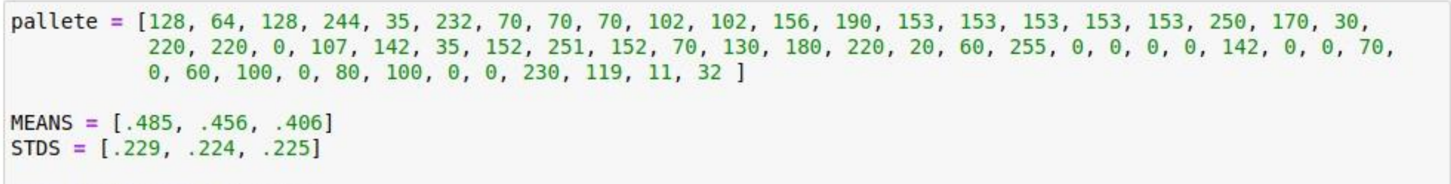
- **GPU:**



```
    # color pallete
    pallete = [128, 64, 128, 244, 35, 232, 70, 70, 70, 102, 102, 156,
                220, 220, 0, 107, 142, 35, 152, 251, 152, 70, 130, 180,
                0, 60, 100, 0, 80, 100, 0, 0, 230, 119, 11, 32 ]

    mean = [.485, .456, .406]
    std =  [.229, .224, .225]
```

- **DPU:**



```
pallete = [128, 64, 128, 244, 35, 232, 70, 70, 70, 102, 102, 156, 190, 153, 153, 153, 153, 153, 250, 170, 30,
           220, 220, 0, 107, 142, 35, 152, 251, 152, 70, 130, 180, 220, 20, 60, 255, 0, 0, 0, 0, 142, 0, 0, 70,
           0, 60, 100, 0, 80, 100, 0, 0, 230, 119, 11, 32 ]

MEANS = [.485, .456, .406]
STDS = [.229, .224, .225]
```

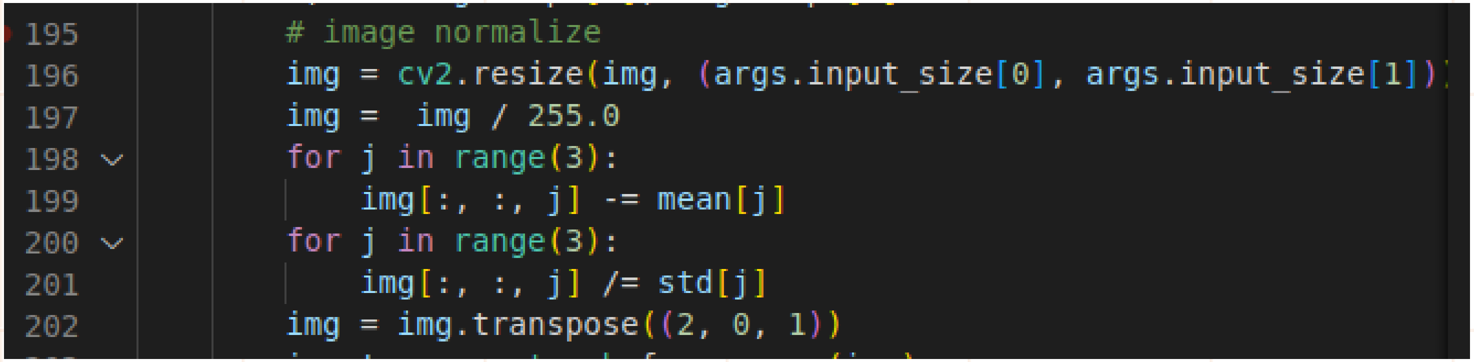
6. Python code mapping GPU -> DPU (3)

Main of the work was retrieved from

pt_ENet_cityscapes_512_1024_8.6G_2.5/code/test/test.py

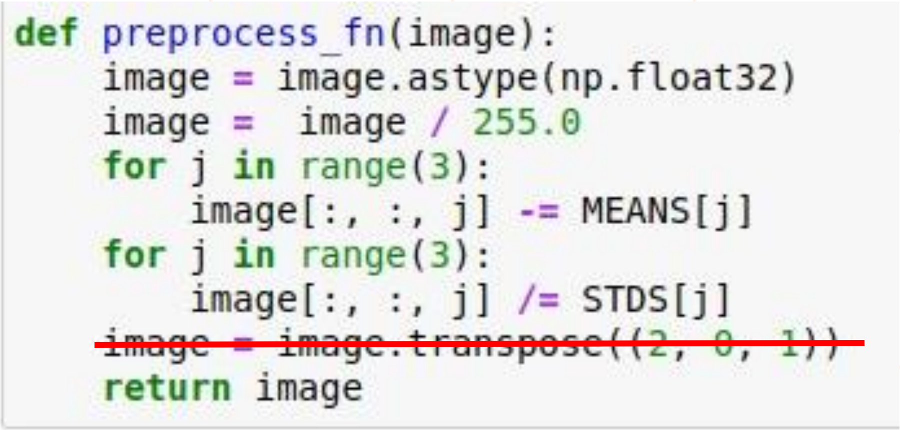
Pre-process

- **GPU:**



```
    # image normalize
    img = cv2.resize(img, (args.input_size[0], args.input_size[1])
    img =  img / 255.0
    for j in range(3):
        img[:, :, j] -= mean[j]
    for j in range(3):
        img[:, :, j] /= std[j]
    img = img.transpose((2, 0, 1))
```

- **DPU:**



```
def preprocess_fn(image):
    image = image.astype(np.float32)
    image =  image / 255.0
    for j in range(3):
        image[:, :, j] -= MEANS[j]
    for j in range(3):
        image[:, :, j] /= STDS[j]
    ~~image = image.transpose((2, 0, 1))~~
    return image
```

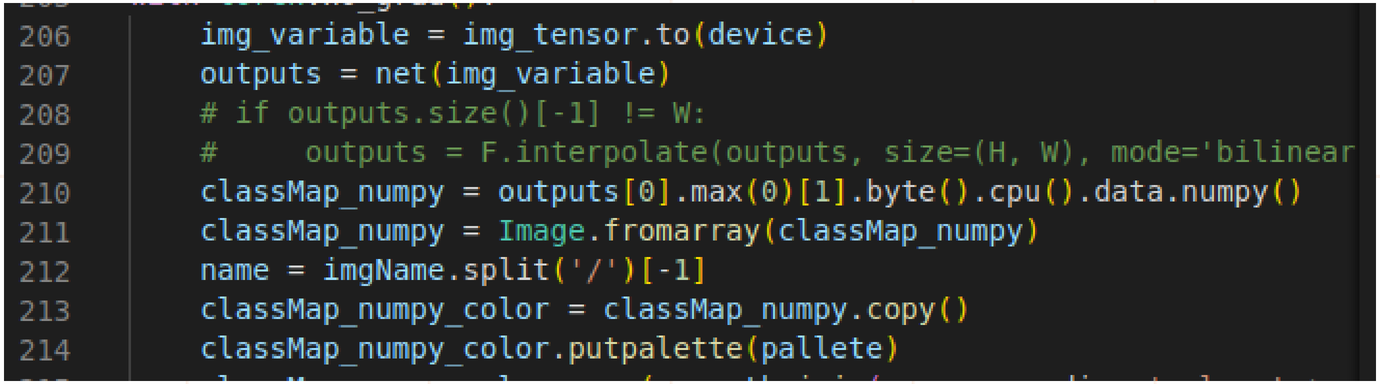
6. Python code mapping GPU -> DPU (4)

Main of the work was retrieved from

pt_ENet_cityscapes_512_1024_8.6G_2.5/code/test/test.py

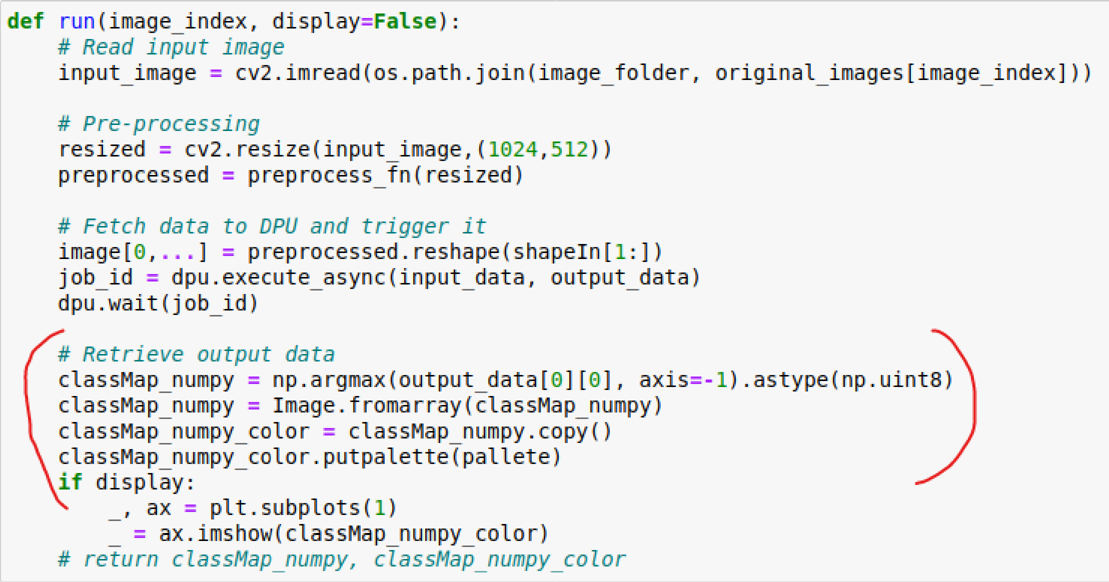
Post-process

- **GPU:**



```
        img_variable = img_tensor.to(device)
        outputs = net(img_variable)
        # if outputs.size()[-1] != W:
        #     outputs = F.interpolate(outputs, size=(H, W), mode='bilinear
        classMap_numpy = outputs[0].max(0)[1].byte().cpu().data.numpy()
        classMap_numpy = Image.fromarray(classMap_numpy)
        name = imgName.split('/')[-1]
        classMap_numpy_color = classMap_numpy.copy()
        classMap_numpy_color.putpalette(pallete)
```

- **DPU:**



```
def run(image_index, display=False):
    # Read input image
    input_image = cv2.imread(os.path.join(image_folder, original_images[image_index]))

    # Pre-processing
    resized = cv2.resize(input_image,(1024,512))
    preprocessed = preprocess_fn(resized)

    # Fetch data to DPU and trigger it
    image[0,...] = preprocessed.reshape(shapeIn[1:])
    job_id = dpu.execute_async(input_data, output_data)
    dpu.wait(job_id)

    # Retrieve output data
    classMap_numpy = np.argmax(output_data[0][0], axis=-1).astype(np.uint8)
    classMap_numpy = Image.fromarray(classMap_numpy)
    classMap_numpy_color = classMap_numpy.copy()
    classMap_numpy_color.putpalette(pallete)
    if display:
        _, ax = plt.subplots(1)
        _ = ax.imshow(classMap_numpy_color)
    # return classMap_numpy, classMap_numpy_color
```