

Lab2: Create the DPU program 2025

The AMD-Xilinx DPU (Deep Learning Processor Unit) is a programmable engine optimized for deep neural networks. It is a group of parameterizable 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 lab 1, you interacted with the DPU using precompiled models. In this lab, we will compile a pretrained deep learning model generating the xmodel file necessary to use the DPU. Additionally, we will compile our own model starting from a TensorFlow definition.

Part 0. Prepare Notebooks

Exercise 0a: Save a copy of dpu_resnet50.ipynb

You have already worked in lab 1 with resnet50 notebook. An Overview of ResNet50 and its Variants can be found here:

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

Make a copy of the Jupyter notebook in a new one called Ex2.1_resnet50.ipynb in order to work with. You will compile in this lab your own xmodel, introduce some modifications and answer questions at the end of the notebook.

Exercise 0b: 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. We already use this notebook in lab1.

make a copy of the notebook in a new one called Ex2.2_dpu_mnist_classifier in order to answer questions at the end of the notebook.

Build Machine Learning Models to program the DPU (the Lab itself).

Objective: We will regenerate the `dpu_resnet50.xmodel` and `dpu_mnist_classifier.xmodel` files that program the DPU (Deep Learning Unit).

We will use VITIS-AI to compile the pretrained model of resnet50 and train mnist classifier.

A. Lab Setup. Open the provided Virtual Machine

We provide an Ubuntu 18.04 virtual machine with minor modifications from the plain vanilla installation downloaded from oxboxes.org. Ubuntu user: `osboxes`; passw: `fpga`

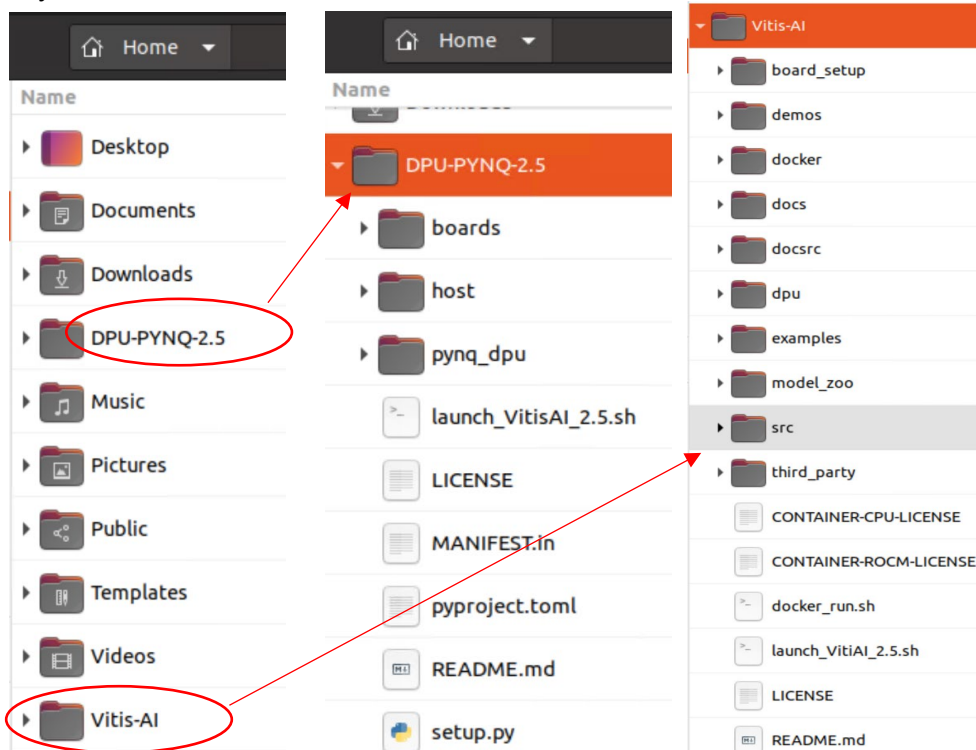
We have already installed (you DO NOT need to install it again):

- Dockers. <https://docs.docker.com/engine/install/ubuntu/>
- Vitis AI. following the instructions at: <https://github.com/Xilinx/Vitis-AI>
We downloaded version 2.5 (docker pull xilinx/vitis-ai:2.5)
- The DPU-Pynq. Using <https://github.com/Xilinx/DPU-PYNQ/blob/master/host/README.md> we cloned the repository

```
git clone -b v2.5 --recurse-submodules https://github.com/Xilinx/DPU-PYNQ.git
```

B. Review the key folders in the /home/ directory.

Namely, Vitis-AI and DPU-PYNQ. For the next of this lab, we will use the DPU-PYNQ folder.



Exercise 1: Generate the resnet50 executable (xmodel file)

1.1 compile the model using the provided script

AMD-Xilinx provides pretrained, quantized and pruned neural networks direct to compile for the DPU accelerator. To this end, Xilinx organized this set of models in a git repository and calling it as “Vitis AI model zoo” (https://github.com/Xilinx/Vitis-AI/tree/2.5/model_zoo). In the following table you have a few of those models:

No.	Application	Name	Float Accuracy	Quantized Accuracy	Input Size	OPS
1	General	tf_inceptionresnetv2_imagenet_299_299_26.35G_2.5	0.8037	0.7946	299*299	26.35G
2	General	tf_inceptionv1_imagenet_224_224_3G_2.5	0.6976	0.6794	224*224	3G
3	General	tf_inceptionv2_imagenet_224_224_3.88G_2.5	0.7399	0.7331	224*224	3.88G
4	General	tf_inceptionv3_imagenet_299_299_11.45G_2.5 ★	0.7798	0.7735	299*299	11.45G
5	General	tf_inceptionv3_imagenet_299_299_0.2_9.1G_2.5 ★	0.7786	0.7668	299*299	9.1G
...						
32	General	tf_mobilenetEdge0.75_imagenet_224_224_624M_2.5	0.7201	0.6489	224*224	624M
33	General	tf2_resnet50_imagenet_224_224_7.76G_2.5	0.7513	0.7423	224*224	7.76G
34	General	tf2_mobilenetv1_imagenet_224_224_1.15G_2.5	0.7005	0.5603	224*224	1.15G
35	General	tf2_inceptionv3_imagenet_299_299_11.5G_2.5	0.7753	0.7694	299*299	11.5G

Comment: Vitis-AI tool is continuing evolution. At the time of writing this document the latest version is 3.5 (https://github.com/Xilinx/Vitis-AI/tree/master/model_zoo). This lab will use v2.5

We will start using the tensorflow2 (tf2) model.

The Vitis AI tools will be run on a docker inside the Virtual machine.

In a console in the virtual machine. You can run the following commands now:

```
cd DPU-PYNQ/host
./docker_run.sh xilinx/vitis-ai:2.5
(or equivalently run ./launch_vitisAI2.5.sh)
```

In order to compile using TensorFlow2, activate:

```
conda activate vitis-ai-tensorflow2
Vitis-AI /workspace > conda activate vitis-ai-tensorflow2
(vitis-ai-tensorflow2) Vitis-AI /workspace > █
```

Inside the docker you will run the script that automates all the processes. You can review options using:

```
./compile.py --help
```

Run the script (`compile.py`) and analyze the results

```
./compile.py --name tf2_resnet50_imagenet_224_224_7.76G_2.5
```

The script will download the “`tf2_resnet50_imagenet_224_224_7.76G_2.5.zip`” file containing the pretrained and optimized model. After that, the compilation starts producing the xmodel file. The console reports the result and in the **DPU-PYNQ/host/** folder the resulting executable is written.



`tf2_resnet50.xmodel`



`tf2_resnet50_imagenet_224_224_7.76G_2.5.yaml`



`tf2_resnet50_imagenet_224_224_7.76G_2.5.zip`

1.2 Review the compile script (`compile.py`).

Open the `compile.py` file and analyze the step that was automated and run in the previous point. Ensure that you understand the key steps.

Basically, the script executes two commands:

```
105 download_model(args.name)
```

```
106 compile_model(args.name, arch=args.arch, output_dir=args.output_dir)
```

a. Before you run the script, it is necessary to activate **conda** environment depending on the model (otherwise the compilation will not work)

“`conda activate vitis-ai-tensorflow`”

or

“`conda activate vitis-ai-tensorflow2`”

or

“`conda activate vitis-ai-pytorch`”

b. Download the model.

```
17 def download_model(model_name):
18     """Downloads and extracts the contents of a quantized Vitis AI modelzoo model. The model.yaml file for the model
19     must contain the "float & quantized" version.
20
21     model_name -- the model name string as seen on in the modelzoo list (e.g. tf_inceptionv1_imagenet_224_224_3G_2.5)
22     """
23
24     # In order to retrieve the model download link, we parse the model.yaml file for that model
25     yaml_url = f"https://github.com/Xilinx/Vitis-AI/raw/v2.5/model_zoo/model-list/{model_name}/model.yaml"
26     download_url = None
27
```

It's made in several steps.

b1. Download a “.yaml” description of the model.

b2. Parse the yaml and obtain the name of the compressed file.

b3. Download the pretrained model (a zip file)

b4. Uncompress the file

c. Compile the model

```
17 def download_model(model_name):
18     """Downloads and extracts the contents of a quantized Vitis AI modelzoo model. The model.yaml file for the model
19     must contain the "float & quantized" version.
20
21     model_name -- the model name string as seen on in the modelzoo list (e.g. tf_inceptionv1_imagenet_224_224_3G_2.5)
22     """
23
24     # In order to retrieve the model download link, we parse the model.yaml file for that model
25     yaml_url = f"https://github.com/Xilinx/Vitis-AI/raw/v2.5/model_zoo/model-list/{model_name}/model.yaml"
26     download_url = None
27
28     output = sp.run([f'wget', f'{yaml_url}', '-O', f'{model_name}.yaml'],
29                     stdout=sp.PIPE, universal_newlines=True)
30
31     assert output.returncode == 0, "Failed to download the model.yaml file."
32     assert os.path.isfile(f'{model_name}.yaml'), "Downloaded yaml has an unexpected name."
33
34     # Open yaml and look for the 'float & quantized' type, the board that will
```


Firstly, parse the first two characters of the model to know the compiler

Then call the appropriate compiler tool:

vai_c_tensorflow or vai_c_tensorflow or vai_c_xir (Xilinx intermediate representation)


Observe that compilers use quantized model

The resulting xmodel is generated in the working folder

▼  tf2_resnet50_imagenet_224_224_7.76G_2.5

▶  code

▼  data

 calib_list.txt

 demo_list.txt

▼  float

 resnet_50.h5

▼  quantized

 quantized.h5

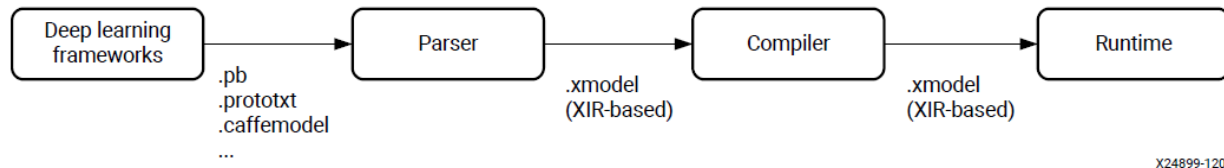
 readme.md

 requirements.txt

1.3 Review the downloaded model from “Vitis AI model Zoo”

In order to understand what the input for the compiler is, observe the Unzipped result of **tf2_resnet50_imagenet_224_224_7.76G_2.5.zip** and observe that you have the trained model (float folder) and the quantized model.

Figure 26: Compilation Flow



X24899-120920

1.4 Test the compiled code into the board

a. You need to compile the model for the Ultra96v2 board. You need to call the compile script informing you of the DPU configuration of the board. In our case, "arch_ultra96.json"

```
./compile.py -a arch_ultra96.json -n  
tf2_resnet50_imagenet_224_224_7.76G_2.5
```

b. Copy **tf2_resnet50.xmodel** from the host into the **~/jupyter_notebook/pynq-dpu** folder in the target embedded system.

 tf2_resnet50.xmodel

 tf2_resnet50_imagenet_224_224_7.76G_2.5.yaml

 tf2_resnet50_imagenet_224_224_7.76G_2.5.zip

```
ilinx@pynq:~/jupyter_notebooks/pynq-dpu$ ls *resn*.xmodel -l  
rw-r--r-- 1 root root 27539805 Jan 22 11:35 dpu_resnet50.xmodel  
rw-r--r-- 1 root root 12849923 Feb 14 19:34 pt_resnet50_07.xmodel  
rw-r--r-- 1 root root 27724553 Feb 14 19:34 pt_resnet50.xmodel  
rw-r--r-- 1 root root 27539805 Feb 14 19:37 tf2_resnet50.xmodel  
ilinx@pynq:~/jupyter_notebooks/pynq-dpu$ ls *resn*.xmodel -l
```

c. In the Ex2.1_dpu_resnet50, change the referenced model with the new generated one ("overlay.load_model("resnet50.xmodel")")

d. Check that everything works as before the model changes.

Exercise 2: Train Your Own DPU Models from Scratch and generate the DPU program from Python

Instead of using the deployable models from the Vitis AI model zoo, we will train the machine learning models. You will use one example in `train_mnist_model.ipynb`.

2.1 Start the training notebook example

You will use the provided Virtual Machine to run the notebook in an environment where the necessary tools are installed. Run the Xilinx docker image containing the Vitis-AI tools.

In a new terminal type:

```
cd DPU-PYNQ/host
./docker_run.sh xilinx/vitis-ai:2.5
(or equivalently run ./launch_vitis_ai.sh)
```

Once you are in the docker environment, if you have not done the following, make sure you do it before running the notebook:

Activate the Tensorflow2

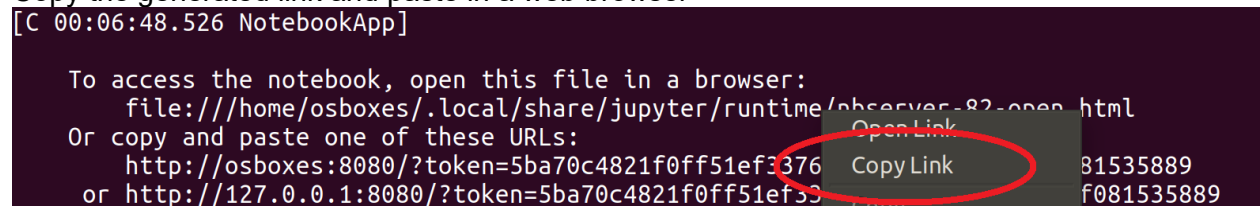
```
conda activate vitis-ai-tensorflow2
```

Then launch jupyter notebook and run the `train_mnist_model.ipynb` step-by-step.

```
jupyter notebook --ip=0.0.0.0 --port=8080
```

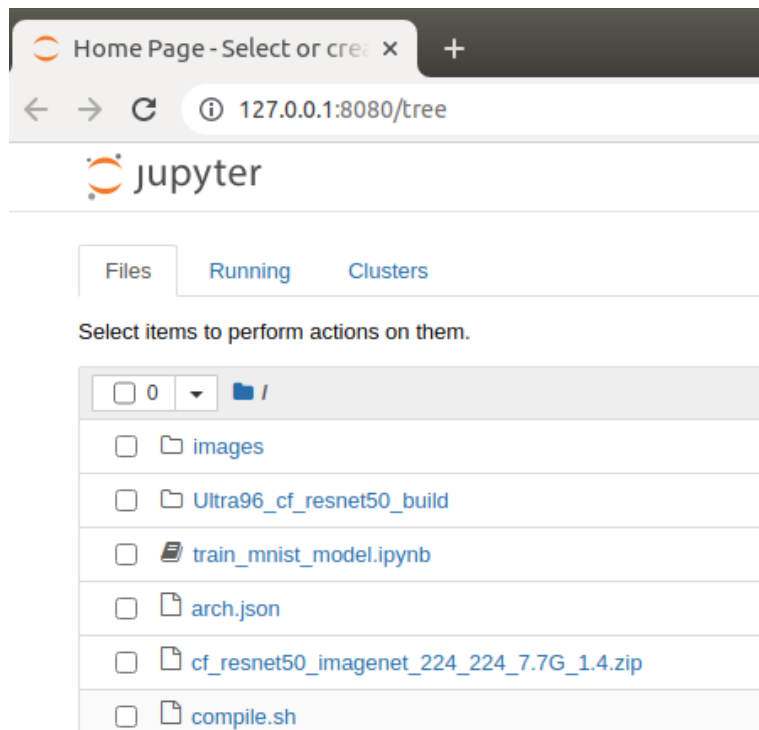
Copy the generated link and paste in a web browser

[C 00:06:48.526 NotebookApp]



```
To access the notebook, open this file in a browser:
file:///home/osboxes/.local/share/jupyter/runtime/observer-82-open.html
Or copy and paste one of these URLs:
http://osboxes:8080/?token=5ba70c4821f0ff51ef337681535889 Copy Link
or http://127.0.0.1:8080/?token=5ba70c4821f0ff51ef33f081535889 Copy Link
```

Open the `train_mnist_model.ipynb` notebook to start training and generating the intermediate files.



2.2 Run the notebook.

a. The first part is a well-known training methodology. The specific part of Vitis-AI start in “3. Quantize” section.

b. The quantization reduces the precision of the internal value to 8 bits to speed up the inference and save memory. It saves the quantized model to an **.h5** file. Namely, to the `tf2_mnist_classifier_quantized.h5` file

c. Freeze TensorFlow Graph

The Vitis AI flow requires a frozen model for the next step (quantization).

d. Evaluate the quantized model.

The quantizer produces a special model called ***quantize_eval_model.pb***, which we can use to load up like a regular Tensorflow binary graph and evaluate its performance.

e. Compile the model.

Using the `vai_c_tensorflow2` (Vitis AI Compiler for Tensorflow2) transform the deploy model into the xmodel executable file.

The cell compiles for the KV260 board.

f. Generate the executable for the Ultra96v2 board.

```
!vai_c_tensorflow2 \
  --model ./tf2_mnist_classifier_quantized.h5 \
  --arch arch_ultra96.json \
  --output_dir . \
  --net_name mnist_classifierU96
```


2.3. Test the compiled code into the board

- a. Copy **mnist_classifierU96.xmodel** from the host into the `~/jupyter_notebook/pynq-dpu` folder in the target embedded system.
- b. Modify the `Ex2.1_dpu_mnist_classifier` and change the referenced model with the new generated one (`overlay.load_model("mnist_classifierU96.xmodel")`)
- c. Check that everything works as before the change of the model.

3. Test other models

The following are optional exercises:

3.1 Generate Resnet50 models from PyTorch definitions

There are several PyTorch resnet50 models. Select one or two and generate xmodels

44	General	pt_resnet50_imagenet_224_224_8.2G_2.5★	0.761/0.929	0.760/0.928	224*224	8.2G
45	General	pt_resnet50_imagenet_224_224_0.3_5.8G_2.5★	0.760/0.929	0.757/0.928	224*224	5.8G
46	General	pt_resnet50_imagenet_224_224_0.4_4.9G_2.5★	0.755/0.926	0.752/0.925	224*224	4.9G
47	General	pt_resnet50_imagenet_224_224_0.5_4.1G_2.5★	0.748/0.921	0.745/0.920	224*224	4.1G
48	General	pt_resnet50_imagenet_224_224_0.6_3.3G_2.5★	0.742/0.917	0.738/0.915	224*224	3.3G
49	General	pt_resnet50_imagenet_224_224_0.7_2.5G_2.5★	0.726/0.908	0.720/0.906	224*224	2.5G

for instance:

```
conda activate vitis-ai-pytorch
./compile.py -a arch_ultra96.json -n
pt_resnet50_imagenet_224_224_0.7_2.5G_2.5
```

Test in hardware.

Questions: Is any difference with tf2 model? Is it faster? Is it More accurate?

Example of results. The DPU process is much faster, pre and post processing takes same ti

Tf2 model	Pytorch quantized 0.7
Classification: cougar, puma, catamount, mc	Classification: wig
Pre proc: 0.021160 35.39 %	Pre proc: 0.021053 44.11 %
DPU proc: 0.036688 61.36 %	DPU proc: 0.024788 51.93 %
Pos proc: 0.036688 3.25 %	Pos proc: 0.024788 3.96 %
Total Ex: 0.059793 i.e. 16.7244 fps	Total Ex: 0.047733 i.e. 20.9497 fps

3.2 Generate Inception model and test in HW.

As shown in resnet50, there are “inception” models to generate the corresponding DPU program (.xmodel)

3.3 Generate yolov3 model and test in HW.

As shown in resnet50, there are yolov3 (and yolov4) models to generate the corresponding DPU program (.xmodel)

4. Deliverables

A zip file with modified notebooks and generated xmodels and upload to moodle