DPU on PYNQ (3)

DNNDK

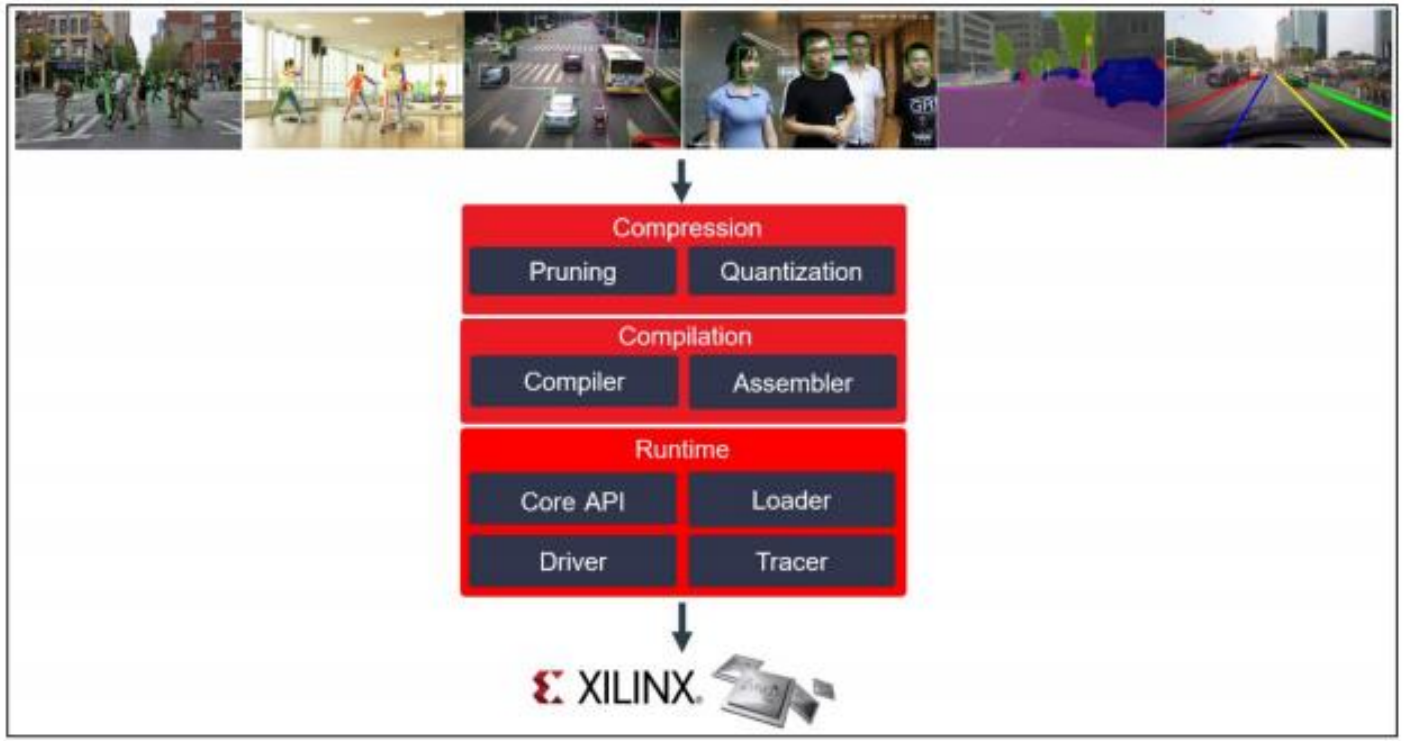
Outline

- DNNDK
  - Overview
  - Toolchain Introduction
- DNNDK lab - Host
- DNNDK lab - Board

DNNDK

Deep Neural Network Development Kit (DNNDK) is a full-stack deep learning SDK for the Deeplearning Processor Unit (DPU). It provides a unified solution for deep neural network inference applications by providing pruning, quantization, compilation, optimization, and run time support.

- A complete set of optimized tool chains, including compression, compilation and runtime.
- Lightweight C/C++ and Python programming APIs.
- Easy-to-use with gradual learning curve.

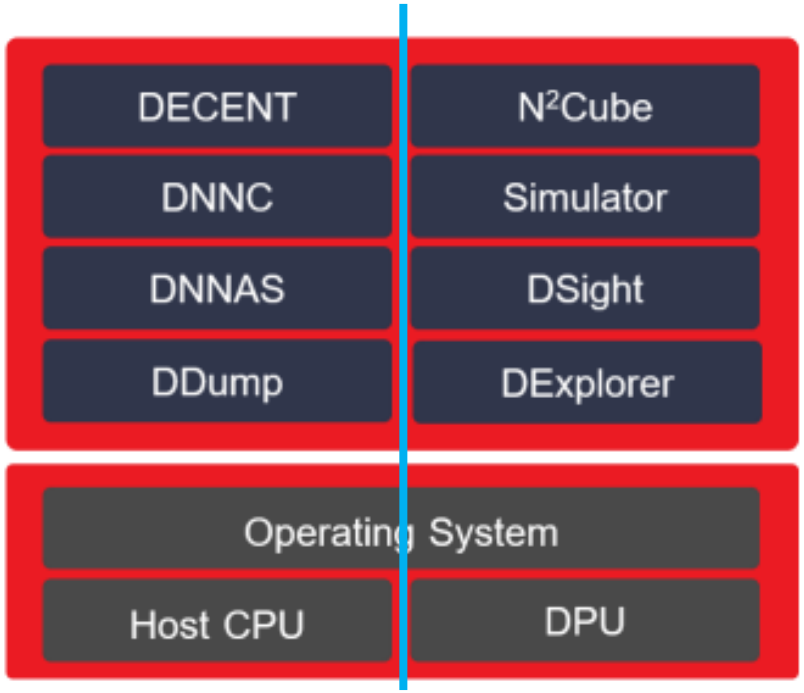


DNNDK framework

DNNDK Framework

As shown in the figure below, DNNDK framework is composed of following components

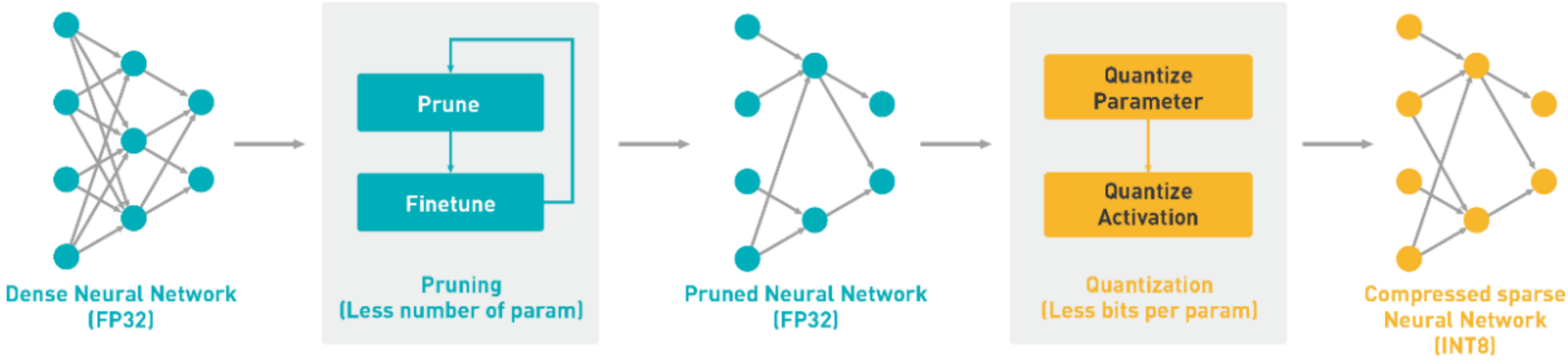
- Deep Compression Tool (DECENT)
- Deep Neural Network Compiler (DNNC)
- Deep Neural Network Assembler (DNNAS)
- Neural Network Runtime (N2Cube)
- DPU Simulator
- Profiler
- DExplorer
- DDump



DNNDK Toolchain

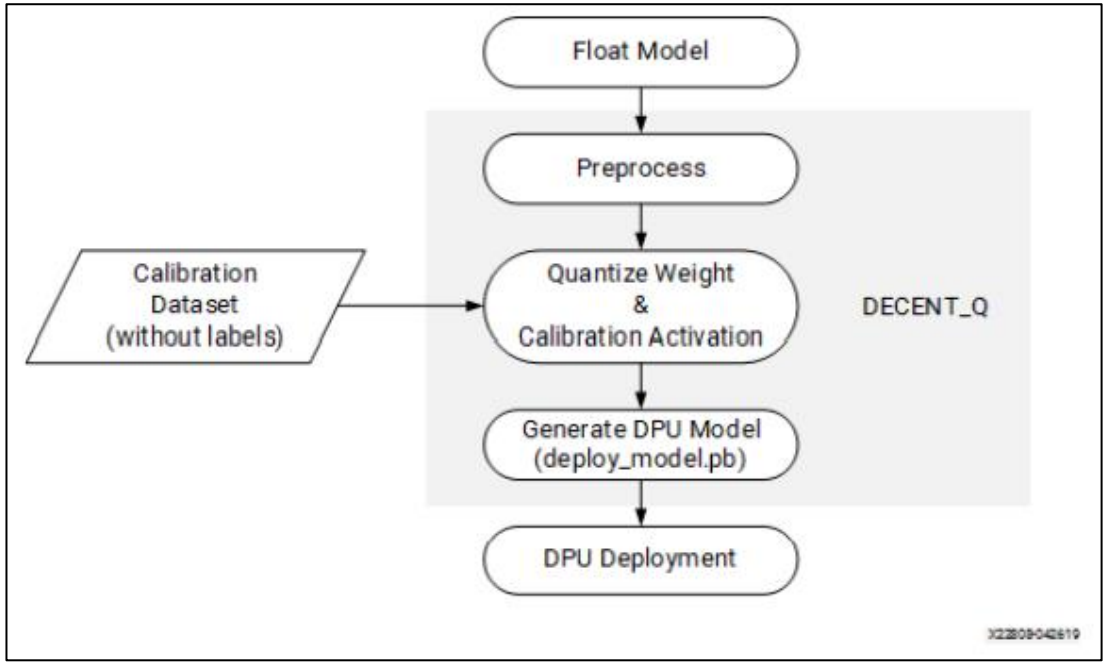
DECENT

DECENT (Deep Compression Tool), employs coarse-grained pruning, trained quantization and weight sharing to address these issues while achieving high performance and high energy efficiency with very small accuracy degradation.



DECENT Pruning and Quantization Flow

DECENT Workflow



DECENT Workflow

UG1327

DECENT Syntax

```
$decent_q quantize \
   --input_frozen_graph ${frozen_graph_pb_file} \
   --input_nodes  ${input_nodes} \
   --input_shapes  ${input_shapes} \
   --output_nodes   ${output_nodes} \
   --input_fn  ${input_fn} \
   [options]
```

```
decent_q quantize \
  --input_frozen_graph frozen_resnet_v1_50.pb \
  --input_nodes input \
  --input_shapes ?,224,224,3 \
  --output_nodes resnet_v1_50/predictions/Reshape_1 \
  --input_fn resnet_v1_50_input_fn.calib_input \
  --method 1 \
  --gpu 0 \
  --calib_iter 10 \
  --output_dir ./quantize_results \
```

decent_q syntax (tensorflow)

```
calib_image_dir = "../../calibration_data/images/"
calib_image_list = "../../calibration_data/calib.txt"
calib_batch_size = 50
def calib_input(iter):
  images = []
  line = open(calib_image_list).readlines()
  for index in range(0, calib_batch_size):
    curline = line[iter * calib_batch_size + index]
    calib_image_name = curline.strip()
    image = cv2.imread(calib_image_dir + calib_image_name)
    image = central_crop(image, 224, 224)
    image = mean_image_subtraction(image, MEANS)
    images.append(image)
  return {"input": images}
```

decent_q input_fn

DLet

- DLet is DNNDK host tool designed to parse and extract various DPU configuration parameters from DPU Hardware Handoff file HWH generated by Vivado.
- The usage info of DLet is shown below.

```
Usage: dlet <option>
 Options are:
  -v  --version    Display version of DLet
  -f  --file       Specity hardware hand-off(HWH) file
  -h  --help       Display the usage of DLet
```

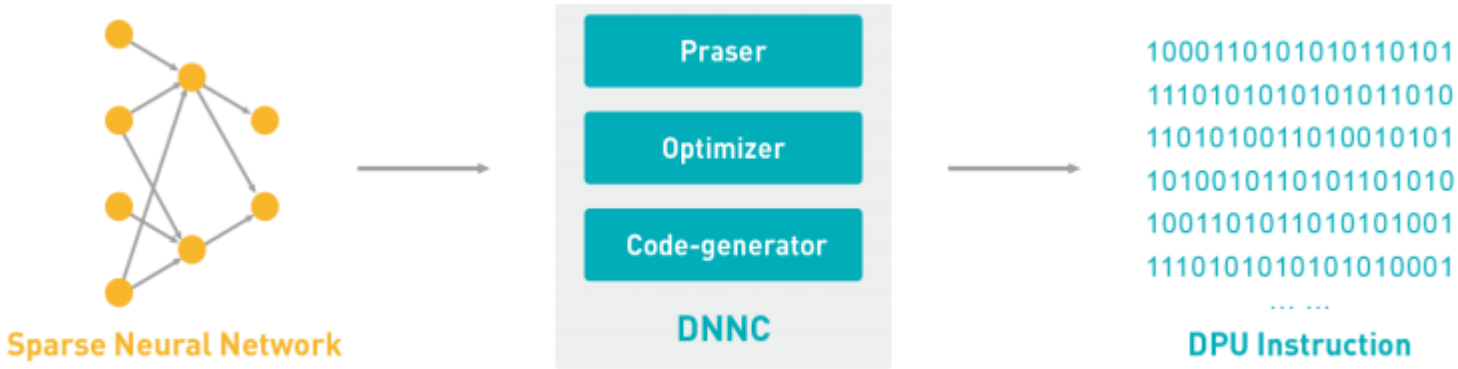
Dlet usage

Ex: **dlet -f ./pynqz2_dpu.hwh**

Output: *.dcf file

DNNC

The architecture of the Deep Neural Network Compiler (DNNC) compiler is shown in the following figure. The front-end parser is responsible for parsing the Caffe/TensorFlow model and generates an intermediate representation (IR) of the input model. The optimizer handles optimizations based on the IR, and the code generator maps the optimized IR to DPU instructions.



DNNC Components

DNNC sample

```
#!/usr/bin/env bash

net="resnet50"
CPU_ARCH="arm64"
DNNC_MODE="debug"
dnndk_board="ZCU102"
dnndk_dcf="../../../dcf/ZCU102.dcf"

model_dir="decent_output"
output_dir="dnnc_output"

if [ ! -d "$model_dir" ]; then
    echo "Can not found directory of $model_dir"
    exit 1
fi

[ -d "$output_dir" ] || mkdir "$output_dir"

echo "Compiling Network ${net}"
dnnc    --prototxt=${model_dir}/deploy.prototxt      \
        --caffemodel=${model_dir}/deploy.caffemodel \
        --output_dir=${output_dir}                   \
        --net_name=${net}                            \
        --dcf=${dnndk_dcf}                           \
        --mode=${DNNC_MODE}                          \
        --cpu_arch=${CPU_ARCH}
```

DNNC Compilation Script for Caffe ResNet-50

```
Compiling network: resnet50
[DNNC][Warning] layer [prob] (type: Softmax) is not supported in DPU, deploy it in CPU instead.

DNNC Kernel topology "resnet50_kernel_graph.jpg" for network "resnet50"
DNNC kernel list info for network "resnet50"
                               Kernel ID : Name
                                       0 : resnet50_0
                                       1 : resnet50_1

                             Kernel Name : resnet50_0
--------------------------------------------------------------------------------
                             Kernel Type : DPUKernel
                               Code Size : 1.28MB
                              Param Size : 24.35MB
                            Workload MACs : 3262.50MOPS
                          IO Memory Space : 2.25MB
                               Mean Value : 104, 107, 123,
                               Node Count : 55
                              Tensor Count : 56
                     Input Node(s)(H*W*C)
                                 conv1(0) : 224*224*3
                    Output Node(s)(H*W*C)
                                 fc1000(0) : 1*1*1000


                             Kernel Name : resnet50_1
--------------------------------------------------------------------------------
                             Kernel Type : CPUKernel
                     Input Node(s)(H*W*C)
                                    prob : 1*1*1000
                    Output Node(s)(H*W*C)
                                    prob : 1*1*1000
```

DNNC Compilation Log for Caffe ResNet-50

DDump

- DDump is a utility tool introduced to dump the info encapsulated inside DPU ELF file or hybrid executable or DPU shared library. It can facilitate the users to analyze and debug various issues.
- DDump is available for both x86 Linux host and DNNDK evaluation boards. Its usage info is shown in the figure below.

```
DDump - Xilinx DNNDK utility to parse and dump DPU ELF file or
        DPU hybrid executable file
Usage: ddump <option>
 At least one of the following switches must be given:
  -f  --file       Specify DPU hybird executable or DPU ELF object
  -k  --klist      Display each kernel general info from DPU ELF file
                   or DPU hybrid executable file
  -d  --dpu        Display DPU architecture info for each kernel
  -c  --compiler   Display the DNCC compiler version for each kernel
  -a  --all        Display all above info
  -v  --version    Display DDump version info
  -h  --help       Display this help info
```

DDump Usage Options

DDump

ddump -f dpu_resnet50_0.elf -k

```
DPU Kernel List from file dnnc_output/dpu_resnet50_0.elf
                     ID:  Name
                      0:  resnet50_0

DPU Kernel name: resnet50_0
--------------------------------------------------------------
 -> DPU Kernel general info
                   Mode:  NORMAL
              Code Size:  1.28MB
             Param Size:  24.35MB
           Workload MACs:  7358.50MOPS
         IO Memory Space:  2.25MB
              Mean Value:  104, 107, 123
              Node Count:  55
             Tensor Count:  56
         Tensor In(H*W*C)
              Tensor ID-0:  224*224*3
        Tensor Out(H*W*C)
             Tensor ID-55:  1*1*1000
```

DDump DPU Kernel Info for ResNet50

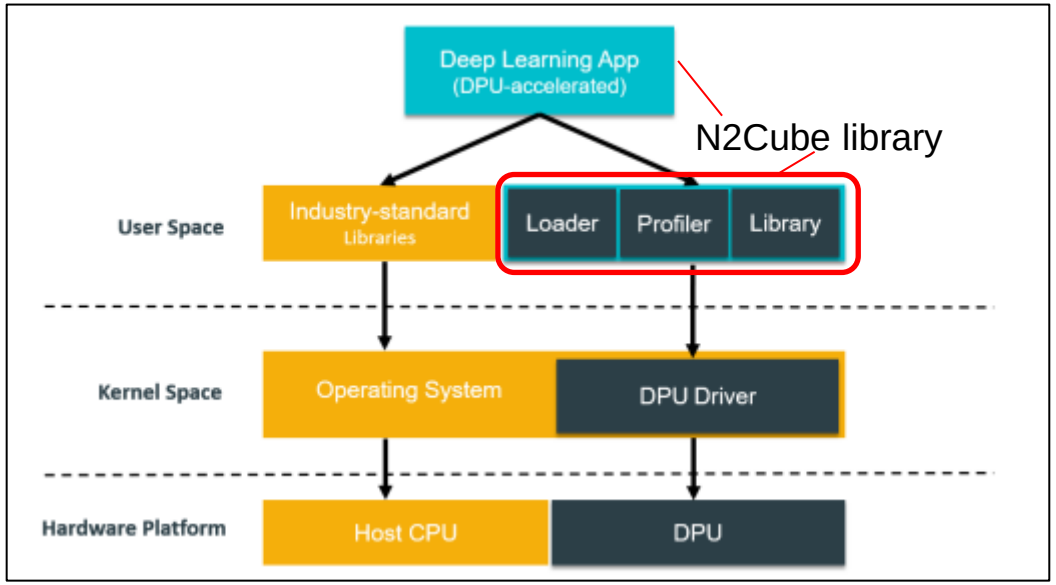
ddump -f dpu_resnet50_0.elf -d

```
DPU Kernel List from file dpu_resnet50_0.elf
                     ID:  Name
                      0:  resnet50_0

DPU Kernel name: resnet50_0
--------------------------------------------------------------
 -> DPU architecture info
              DPU ABI Ver:  v2.0
DPU Configuration Parameters
            DPU Target Ver:  1.4.0
             DPU Arch Type:  B512
                 RAM Usage:  high
              DepthwiseConv:  Enabled
      DepthwiseConv+Relu6:  Enabled
           Conv+Leakyrelu:  Enabled
               Conv+Relu6:  Enabled
     Channel Augmentation:  Enabled
              Average Pool:  Disabled
```

DDump DPU Arch Info for ResNet50

N2Cube

- The Cube of Neutral Networks (N2Cube) is the DPU runtime engine.
- It acts as the loader for the DNNDK applications and handles resource allocation and DPU scheduling.
- Its core components include DPU driver, DPU loader, tracer, and programming APIs for application development.



DPU Runtime

DExplorer

- DExplorer is a utility running on the target board.
- It provides DPU running mode configuration, DNNDK version checking, DPU status checking, and DPU core signature checking.
- The following figure shows the help information about the usage of DExplorer.

```
Usage: dexplorer <option>
 Options are:
  -v  --version    Display version info for each DNNDK component
  -s  --status     Display the status of DPU cores
  -w  --whoami     Display the info of DPU cores
  -m  --mode       Specify DNNDK N2Cube running mode: normal, profile, or debug
  -t  --timeout    Specify DPU timeout limitation in seconds under integer range of [1, 100]
  -h  --help       Display this information
```

DExplorer Usage Options

DExplorer

```
root@dp-n1:~# dexplorer -s
[DPU cache]
Enabled

[DPU mode]
normal

[DPU timeout limitation (in seconds)]
5

[DPU Debug Info]
Debug level     : 9
Core 0 schedule : 0
Core 0 interrupt: 0

[DPU Resource]
DPU Core        : 0
State           : Idle
PID             : 0
TaskID          : 0
Start           : 0
End             : 0

[DPU Registers]
VER             : 0x05c1c6bd
RST             : 0x000000ff
ISR             : 0x00000000
IMR             : 0x00000000
IRSR            : 0x00000000
ICR             : 0x00000000

DPU Core        : 0
HP_CTL          : 0x07070f0f
ADDR_IO         : 0x00000000
ADDR_WEIGHT     : 0x00000000
ADDR_CODE       : 0x00000000
ADDR_PROF       : 0x00000000
```

DExplorer Status

```
root@xilinx-zcu102-2019_1:~$dexplorer -w
[DPU IP Spec]
IP  Timestamp           : 2019-07-24 11:15:00
DPU Core Count          : 3

[DPU Core Configuration List]
DPU Core                : #0
DPU Enabled             : Yes
DPU Arch                : B4096
DPU Target Version      : v1.4.0
DPU Freqency            : 325 MHz
Ram Usage               : Low
DepthwiseConv           : Enabled
DepthwiseConv+Relu6     : Enabled
Conv+Leakyrelu          : Enabled
Conv+Relu6              : Enabled
Channel Augmentation    : Enabled
Average Pool            : Enabled

DPU Core                : #1
```

Sample DPU Signature with Configuration Parameters

DSight

DSight is the DNNDK performance profiling tool. It is a visual performance analysis tool for neural network model profiling.

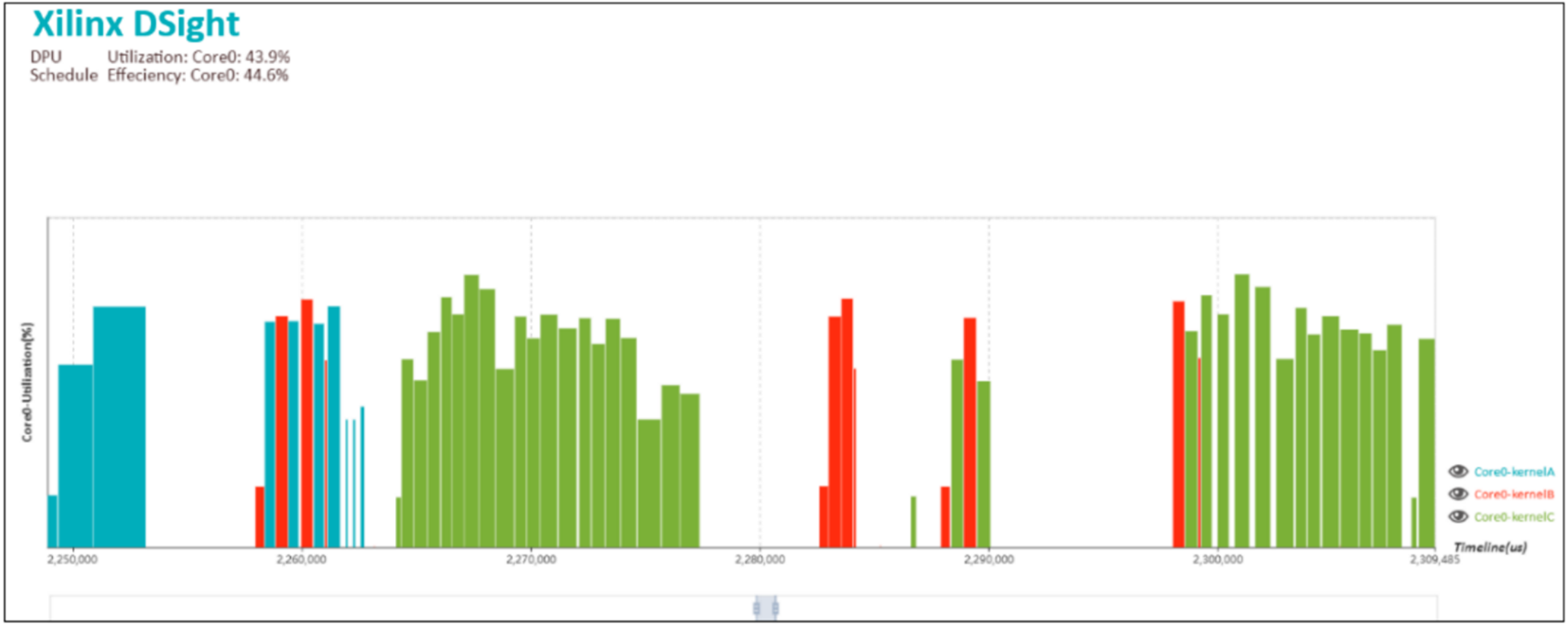
Dsight profiling Step:

1. Set N2Cube to profile mode using the command *dexplorer -m profile*
2. Run the deep learning application. When finished, a profile file with the name *dpu_trace_[PID].prof* is generated. (PID is the process ID of the deep learning application).
3. Generate the html file with the DSight tool using the command: *dsight -p dpu_trace_[PID].prof*. An html file with the name *dpu_trace_[PID].html* is generated.
4. Open the generated html file with web browser.

```
root@xlnx:~# dsight -h
Usage: dsight <option>
 Options are:
  -p  --profile    Specify DPU trace file for profiling
  -v  --version    Display DSight version info
  -h  --help       Display this information
```

DSight Help Info

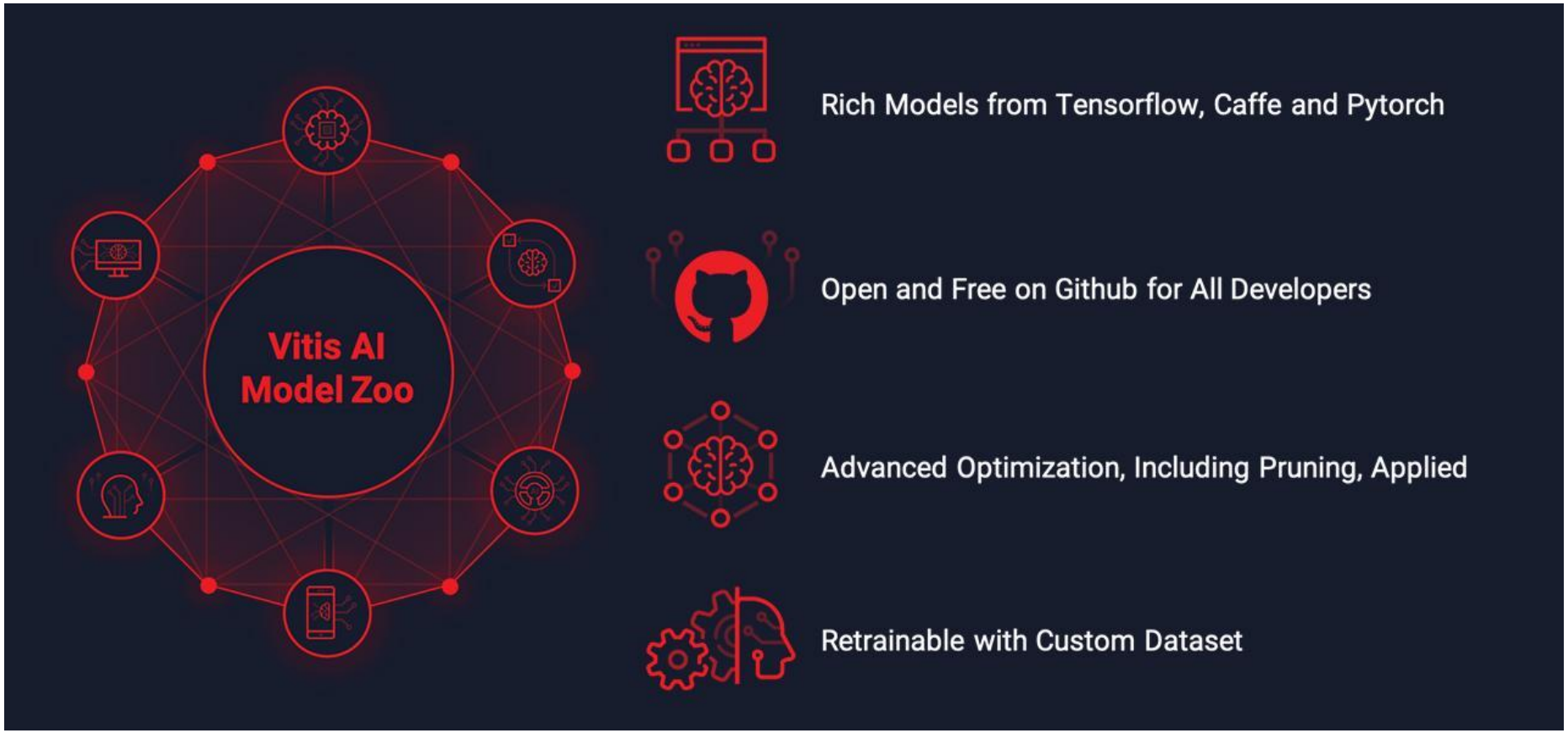
DSight Profiling Charts



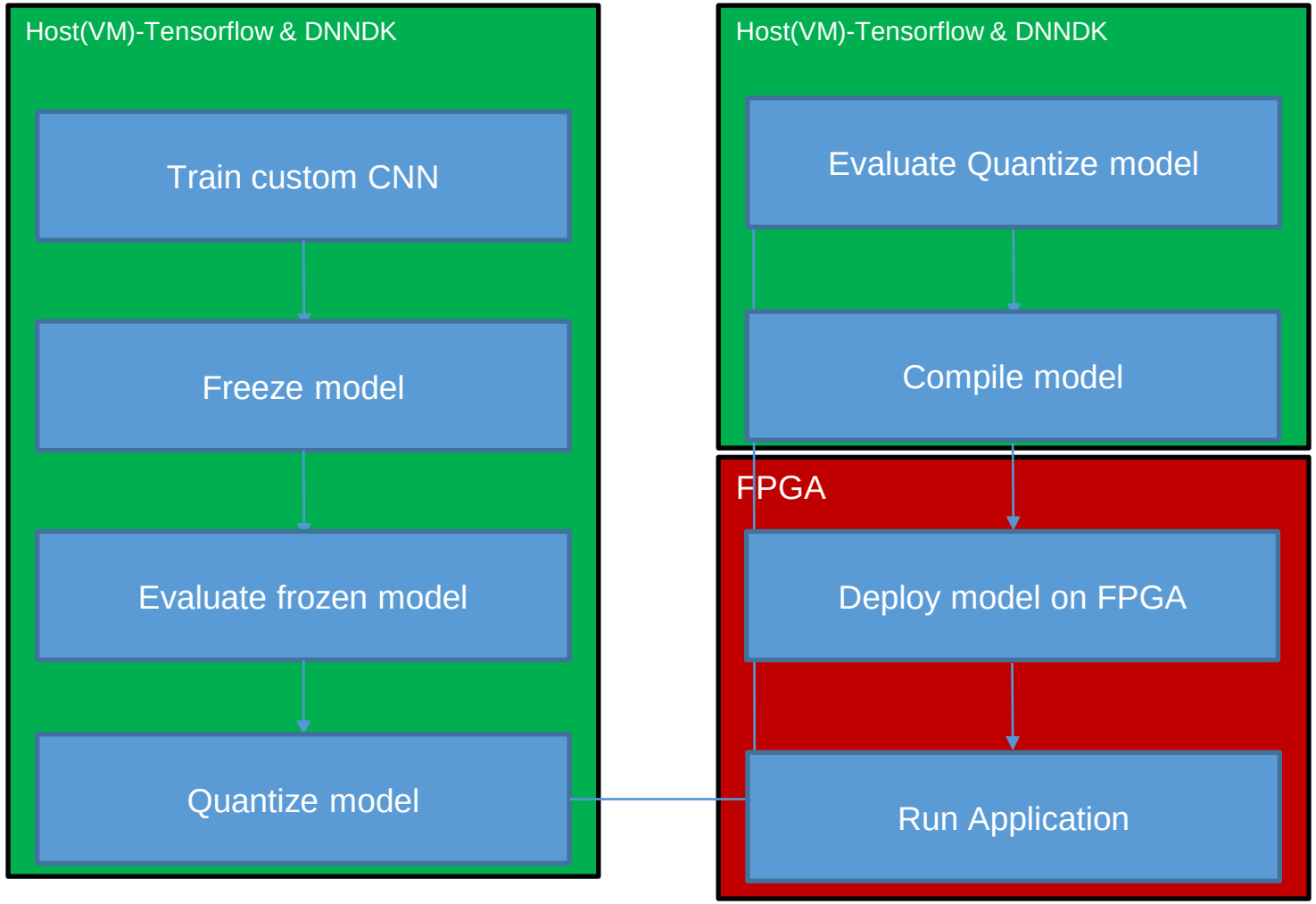
UG1327

Xilinx Vitis-ai AI-Model-Zoo

Vitis-ai AI-Model-Zoo https://github.com/Xilinx/Vitis-AI/tree/master/models/AI-Model-Zoo
Example code for each model: https://github.com/Xilinx/Vitis-AI/tree/master/models/AI-Model-Zoo/caffe-xilinx/examples



Today’s Lab

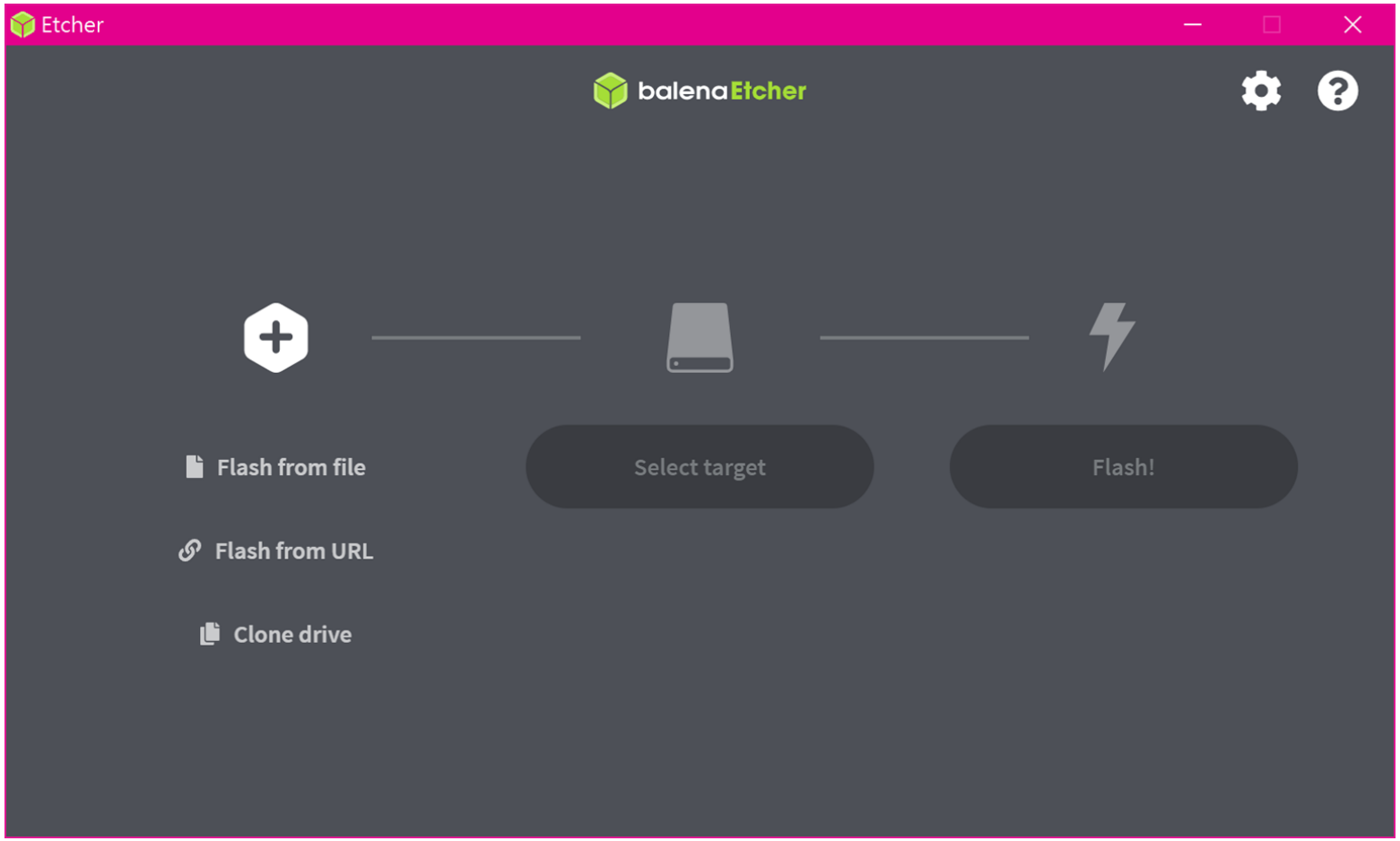

Host(VM)-Tensorflow & DNNDK
Train custom CNN
Freeze model
Evaluate frozen model
Quantize model
Host(VM)-Tensorflow & DNNDK
Evaluate Quantize model
Compile model
FPGA
Deploy model on FPGA
Run Application

Flash Boot Image

Mount SD card and open Etcher

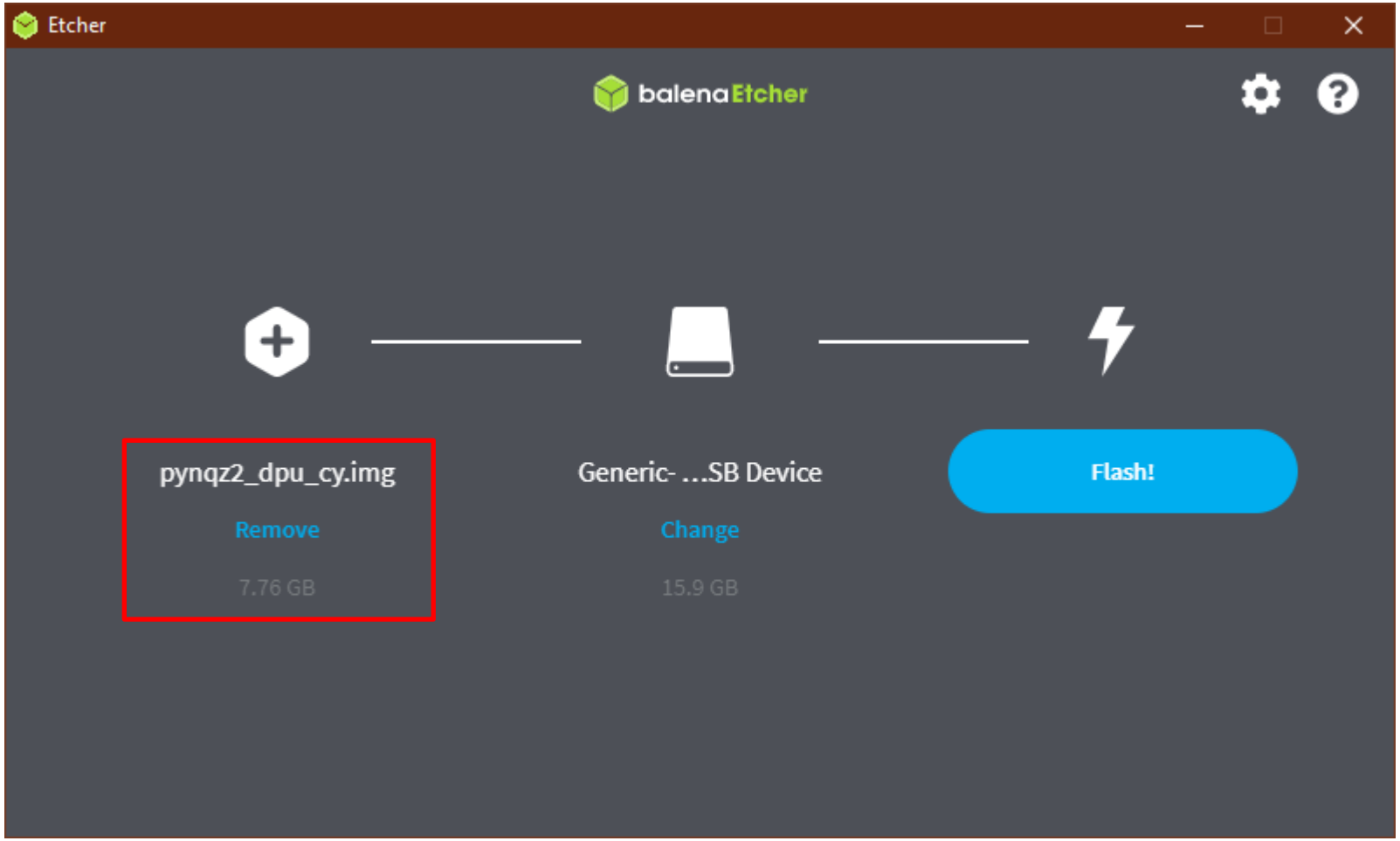
Path:Desktop(桌面)/Advanced_FPGA_Design/EtcherPortable/

請記得把小張的SD卡插進轉接卡後
再把整張轉卡插進電腦裡



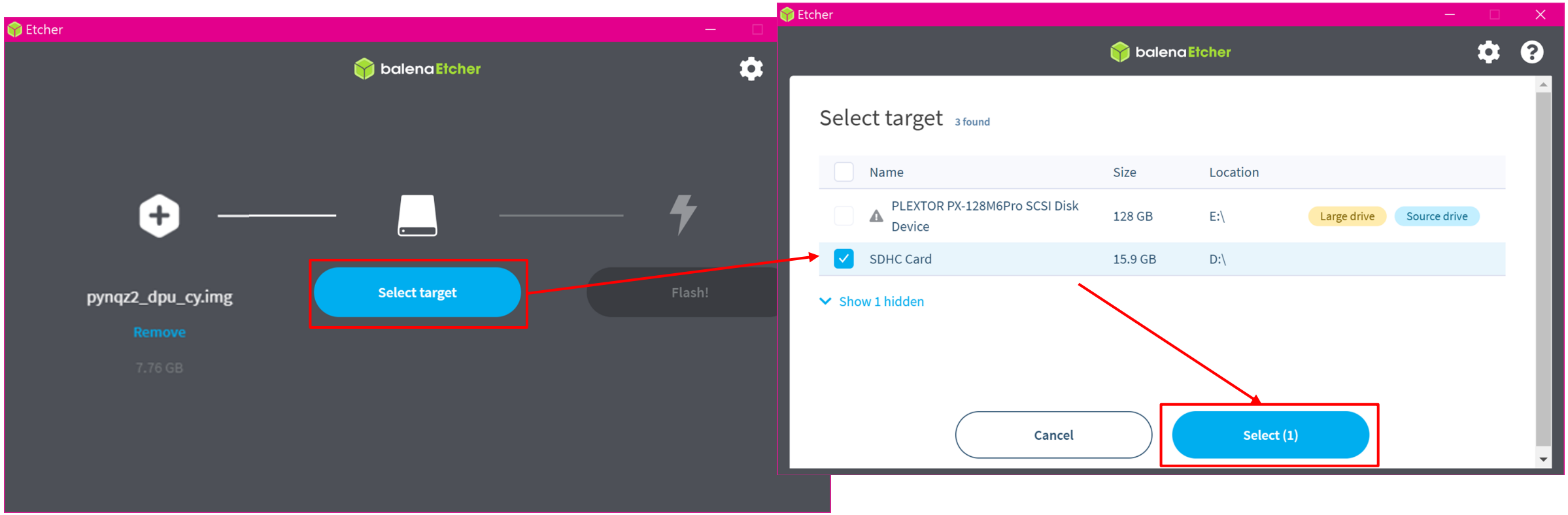
Select img file

select pynqz2_dpu_cy_v2.img , Download link



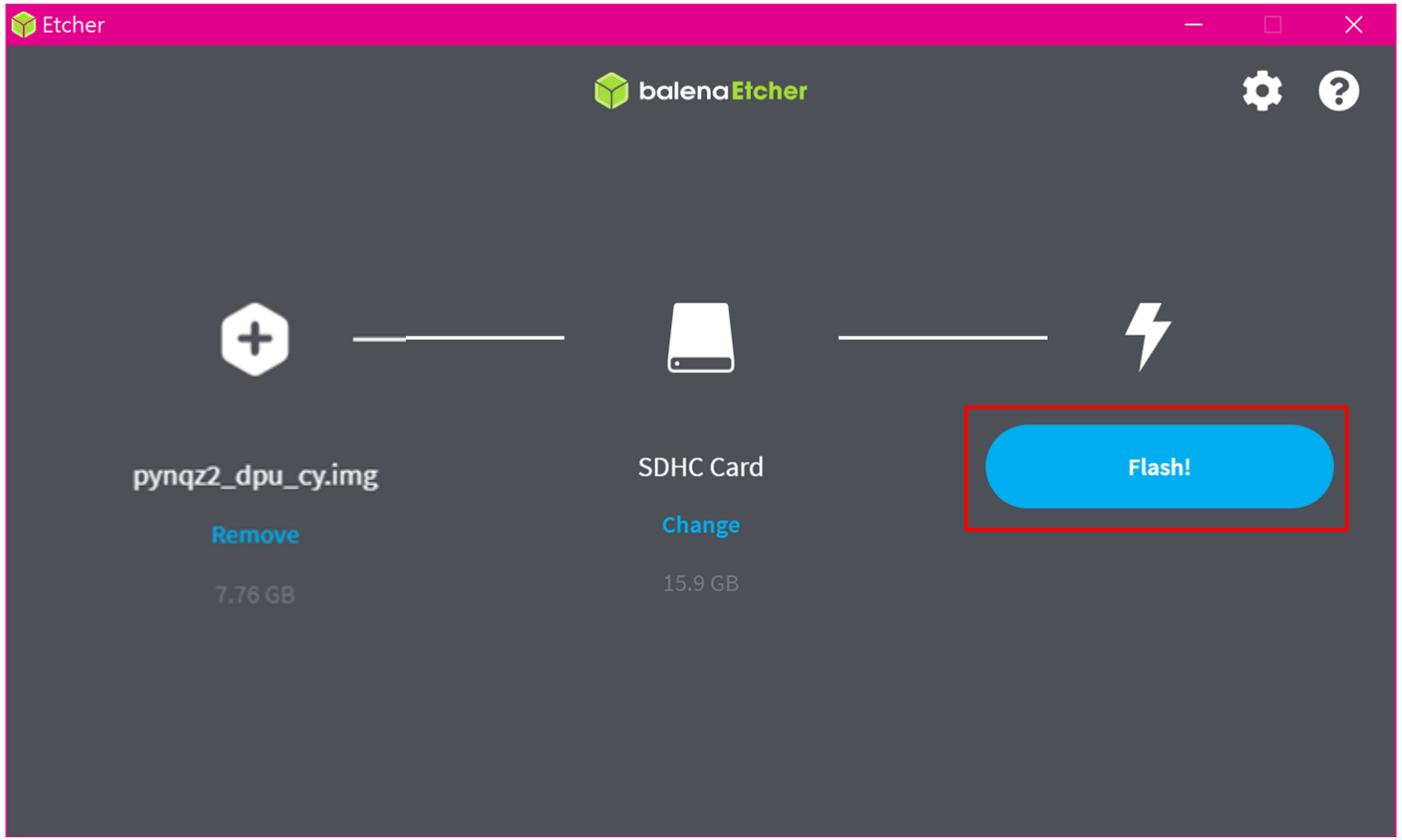
Select Target

select SDHC



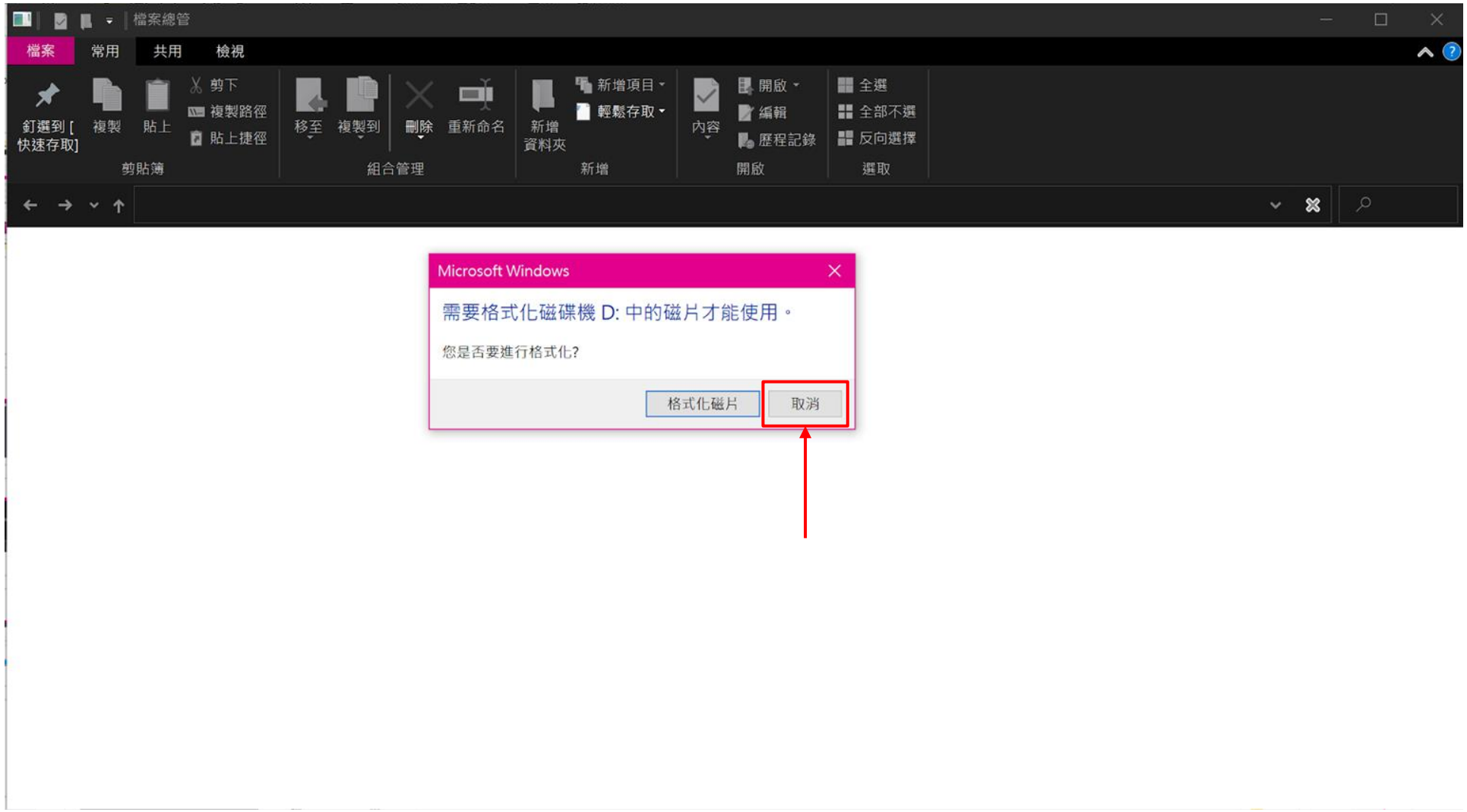
Click Flash

Please make sure your etcher looks like this

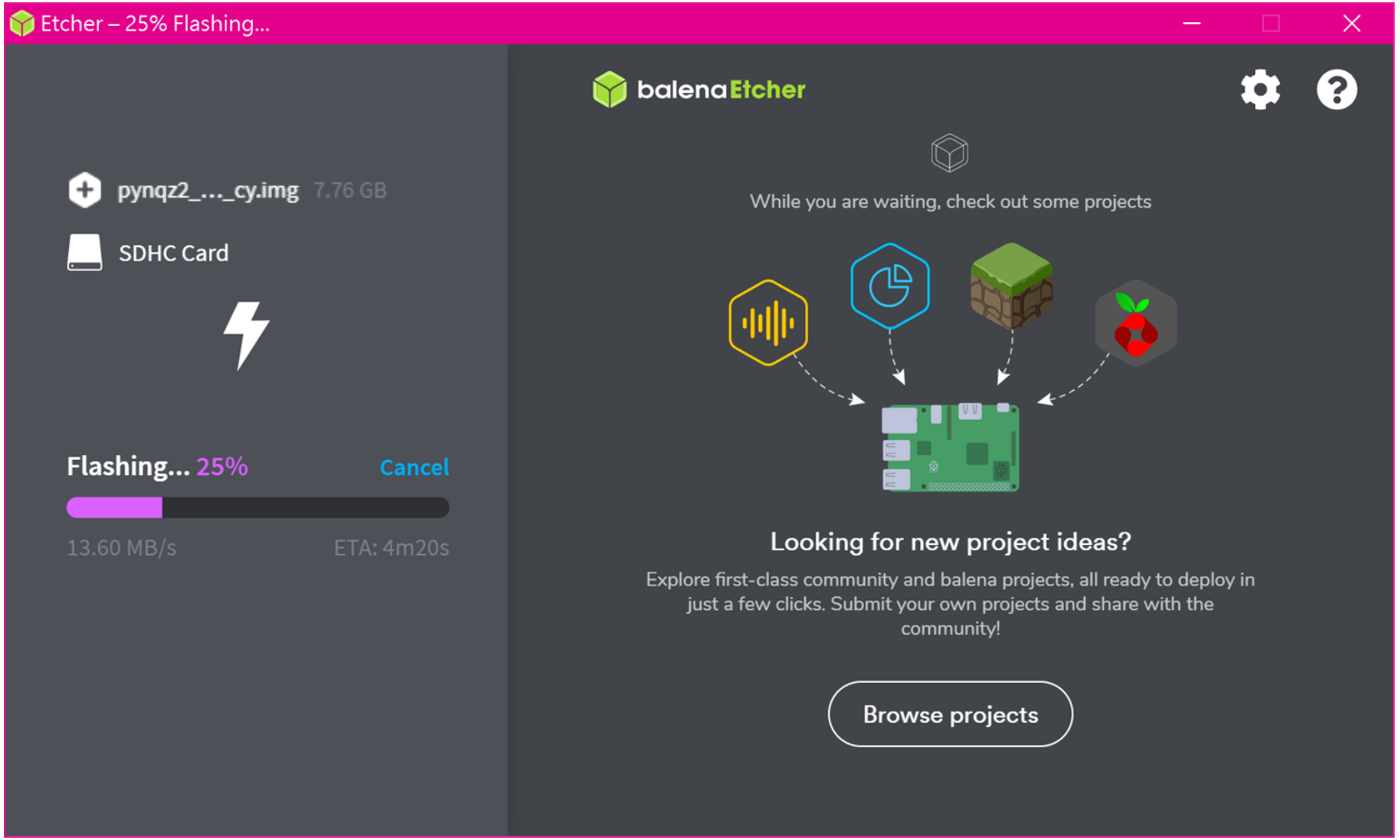


Select cancel and ignore error message if this pop out.

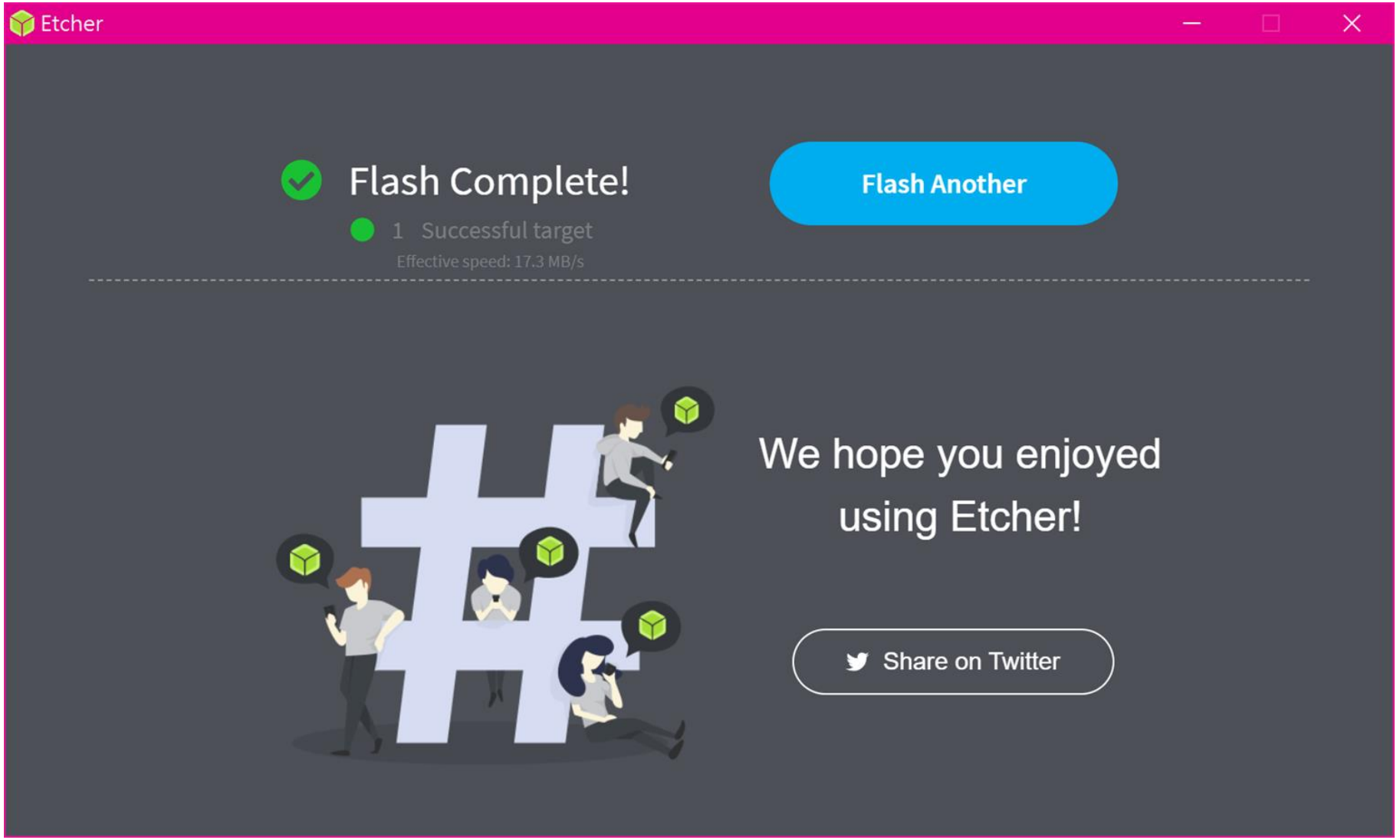
Do not format the disk manually!



Make sure it’s flashing and wait for it

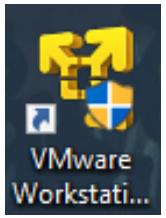


Done!

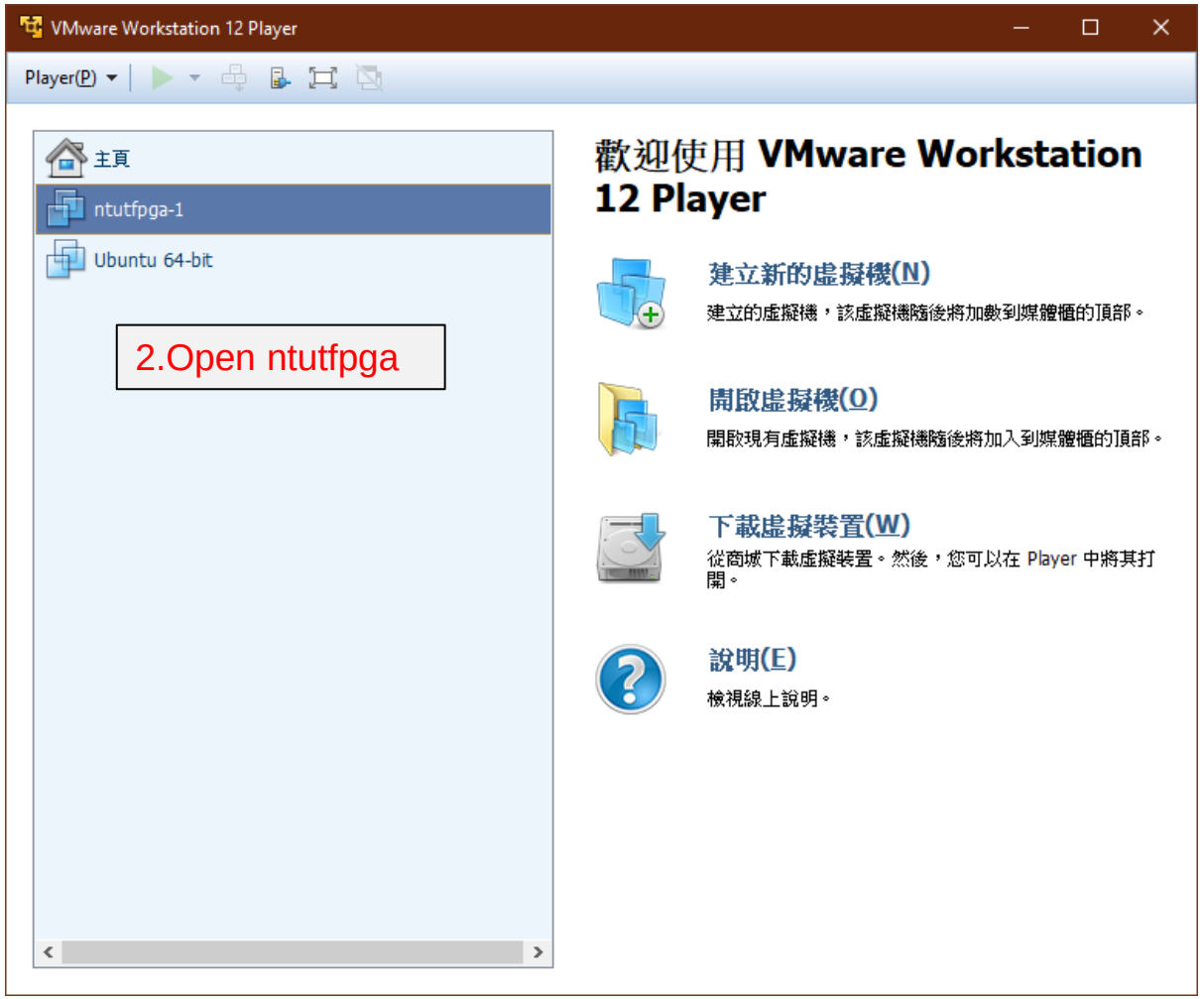

Etcher
Flash Complete!
1 Successful target
Effective speed: 17.3 MB/s
Flash Another
We hope you enjoyed using Etcher!
Share on Twitter

DNNDK Lab - host

Open VMware



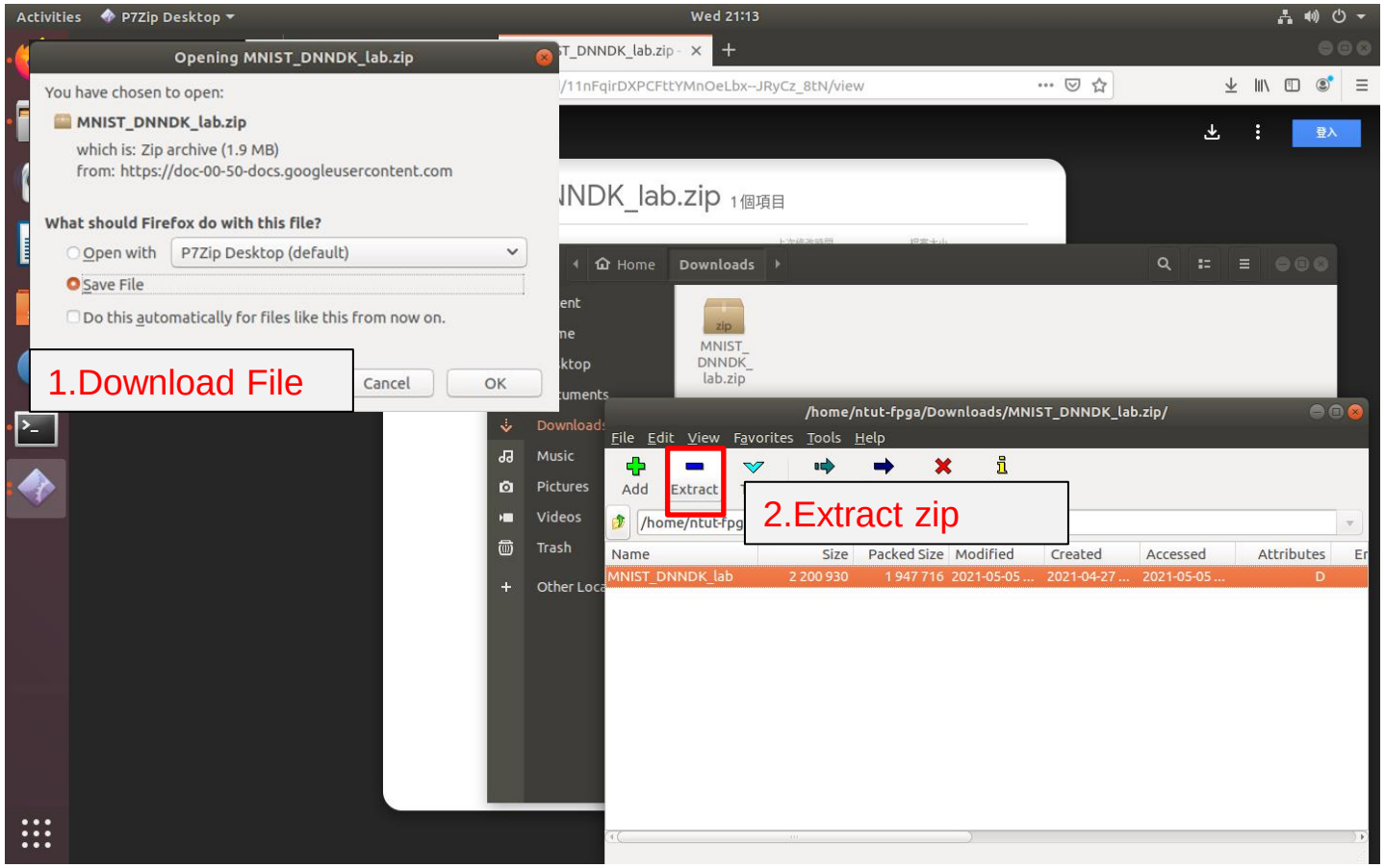
1.Open VMware

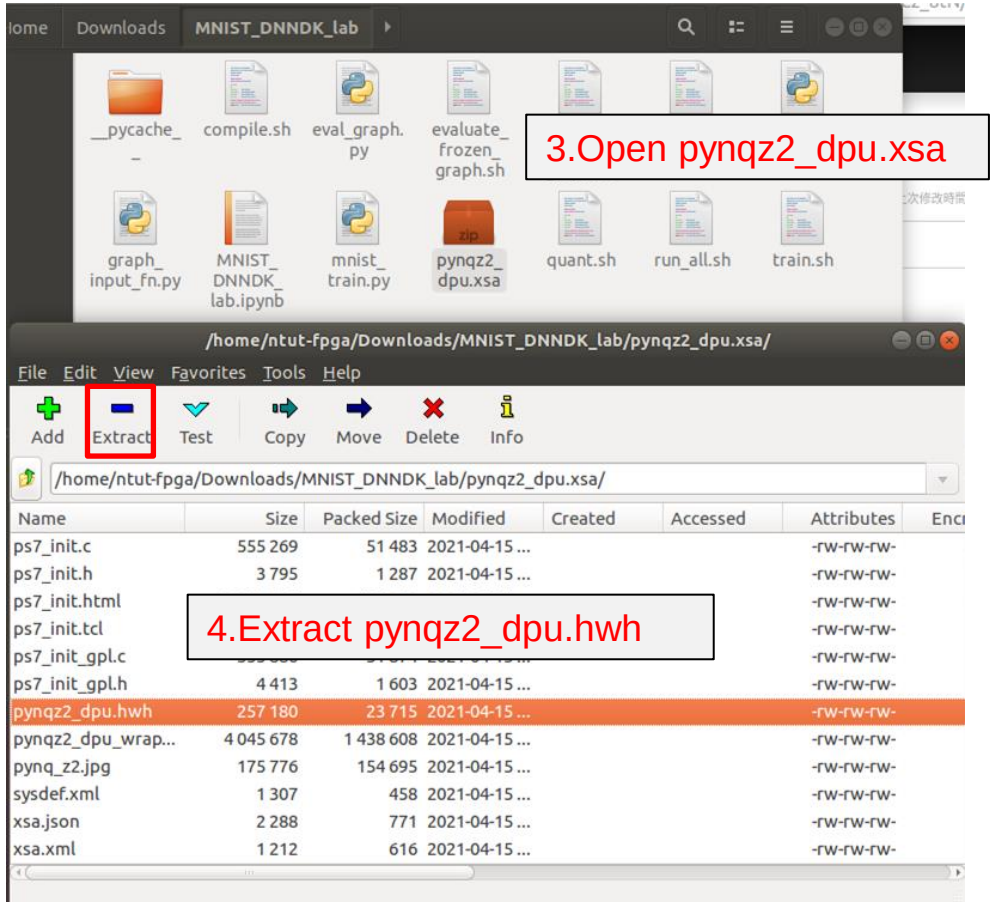


Password:ntutfpga

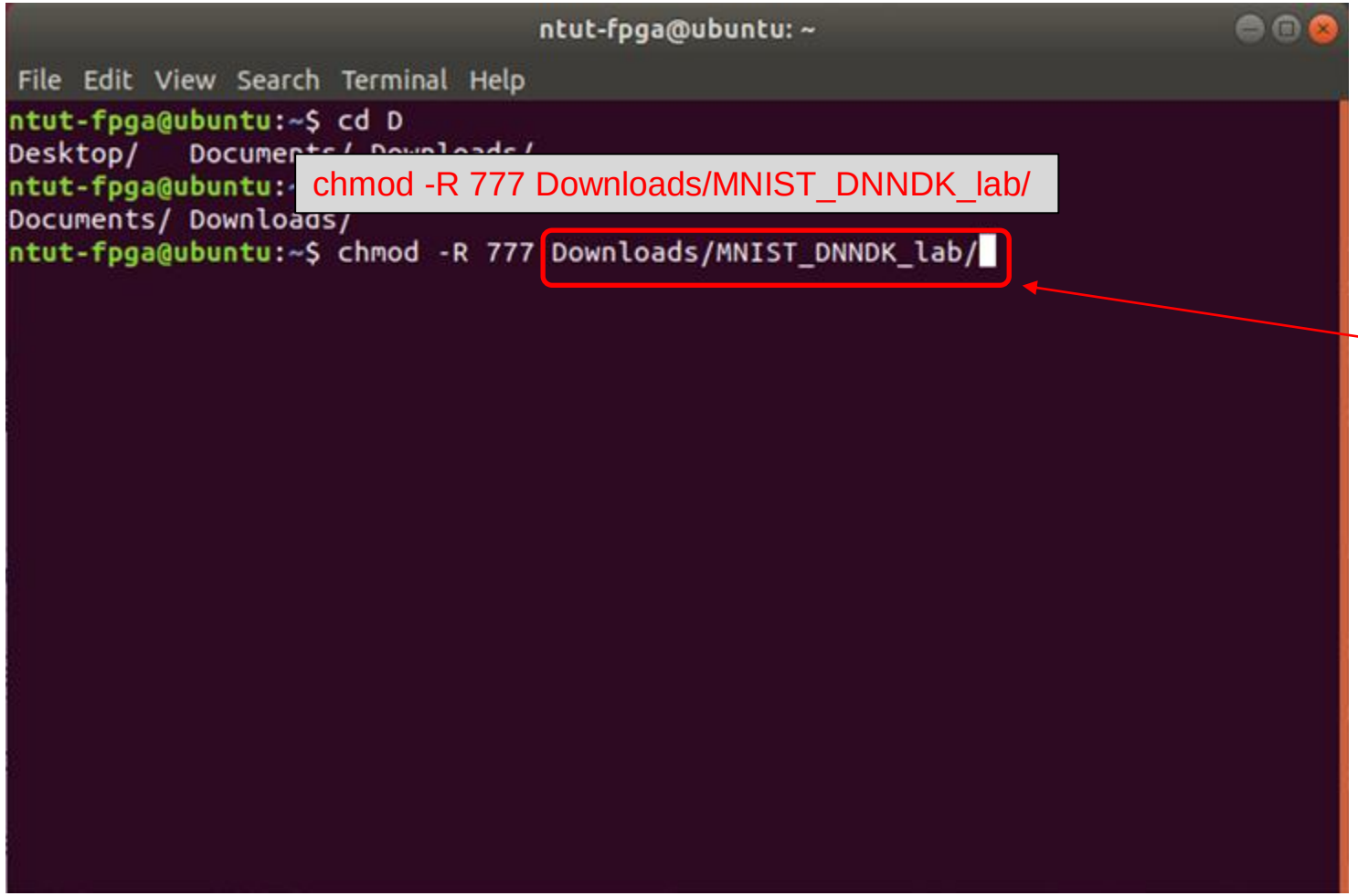
Download lab file on VM

Lab file Download Link:https://drive.google.com/file/d/11nFqirDXPCFttYMnOeLbx--JRyCz_8tN/view?usp=sharing





Change lab file permission



Directory of the lab file that you just downloaded and extracted

Run Jupyter notebook

python3 -c "from notebook.auth import passwd; print(passwd('', algorithm='sha1'))"

-> sha1:a46679d19a61:58b4dd703239e6de7f445dff05a8407640d19dcb

jupyter-lab --ServerApp.ip="*" --ServerApp.password="sha1:a46679d19a61:58b4dd703239e6de7f445dff05a8407640d19dcb"

```
ntut-fpga@ubuntu:~$ python3 -c "from notebook.auth import passwd; print(passwd('', algorithm='sha1'))"
sha1:92e29861c388:3bb89b37d2022aff96c3c4798f11e2171d26056a
ntut-fpga@ubuntu:~$ jupyter-lab --ServerApp.ip="*" --ServerApp.password="sha1:92e29861c388:3bb89b37d2022aff96c3c4798f11e2171d26056a"
```

(New terminal)

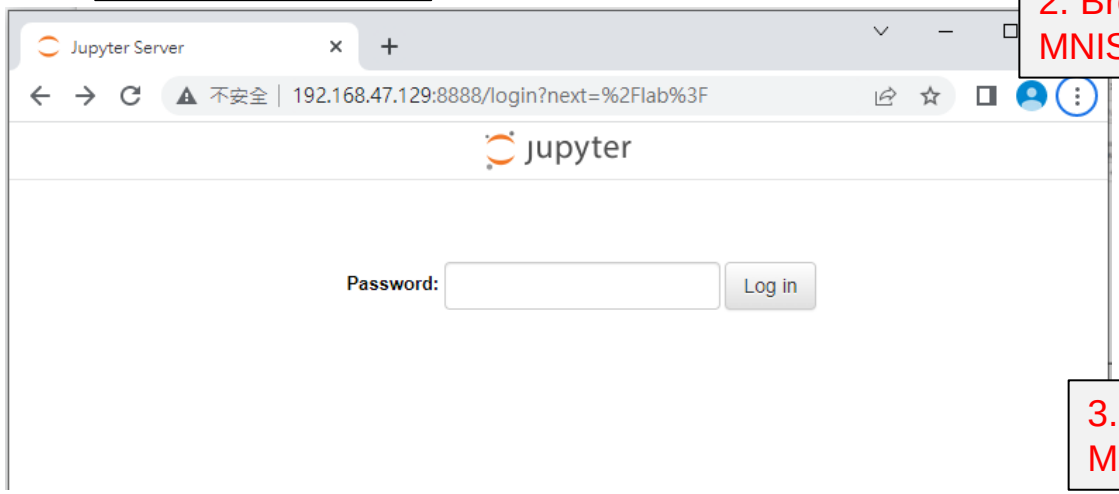
ifconfig

```
ntut-fpga@ubuntu:~$ ifconfig
docker0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        inet 172.17.0.1  netmask 255.255.0.0  broadcast 172.17.255.255
        ether 02:42:32:1e:2e:0b  txqueuelen 0  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.47.129  netmask 255.255.255.0  broadcast 192.168.47.255
        inet6 fe80::dce2:90f8:63c0:37ac  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:14:ba:ca  txqueuelen 1000  (Ethernet)
        RX packets 640006  bytes 942999267 (942.9 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 102652  bytes 7605296 (7.6 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

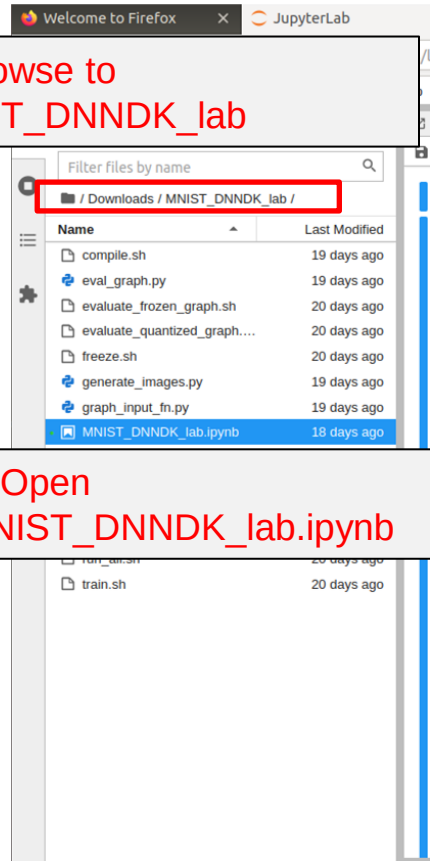
Browse 192.168.47.129:8888 in windows browser and log in.

run Jupyter-lab

1.login jupyter-lab

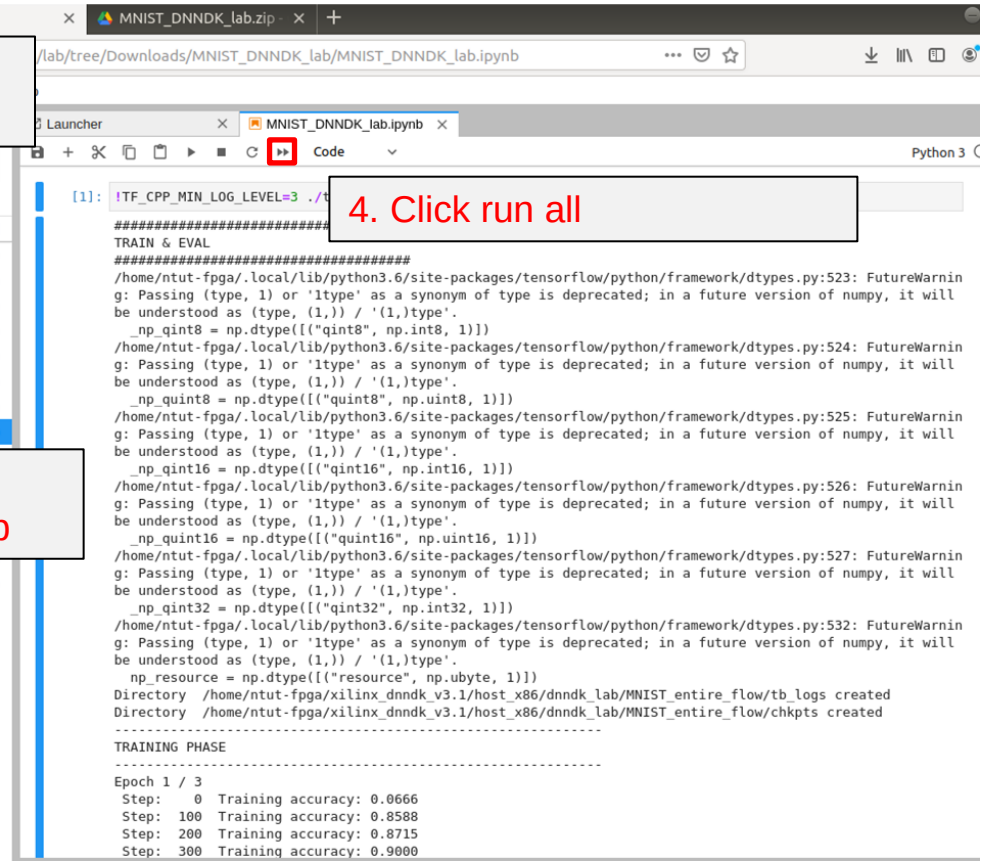


2. Browse to MNIST_DNNDK_lab



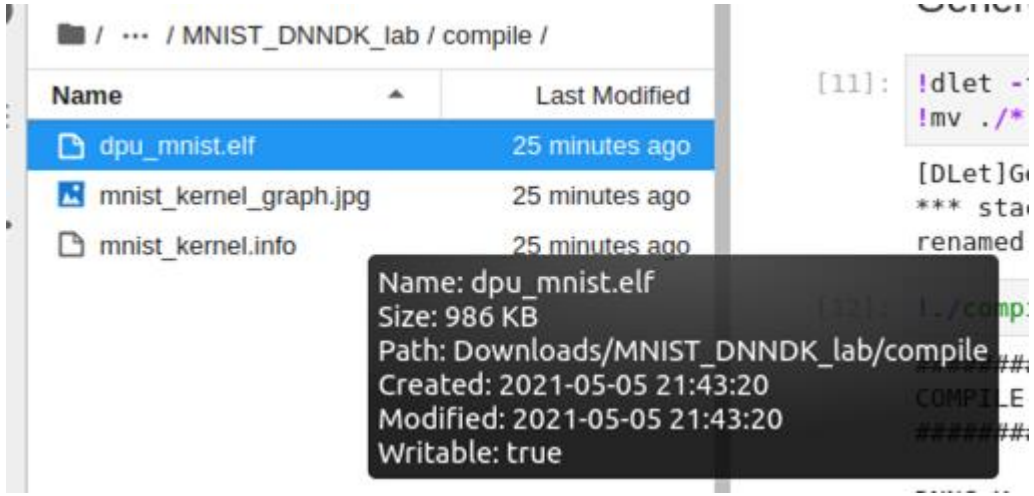
3. Open MNIST_DNNDK_lab.ipynb

4. Click run all

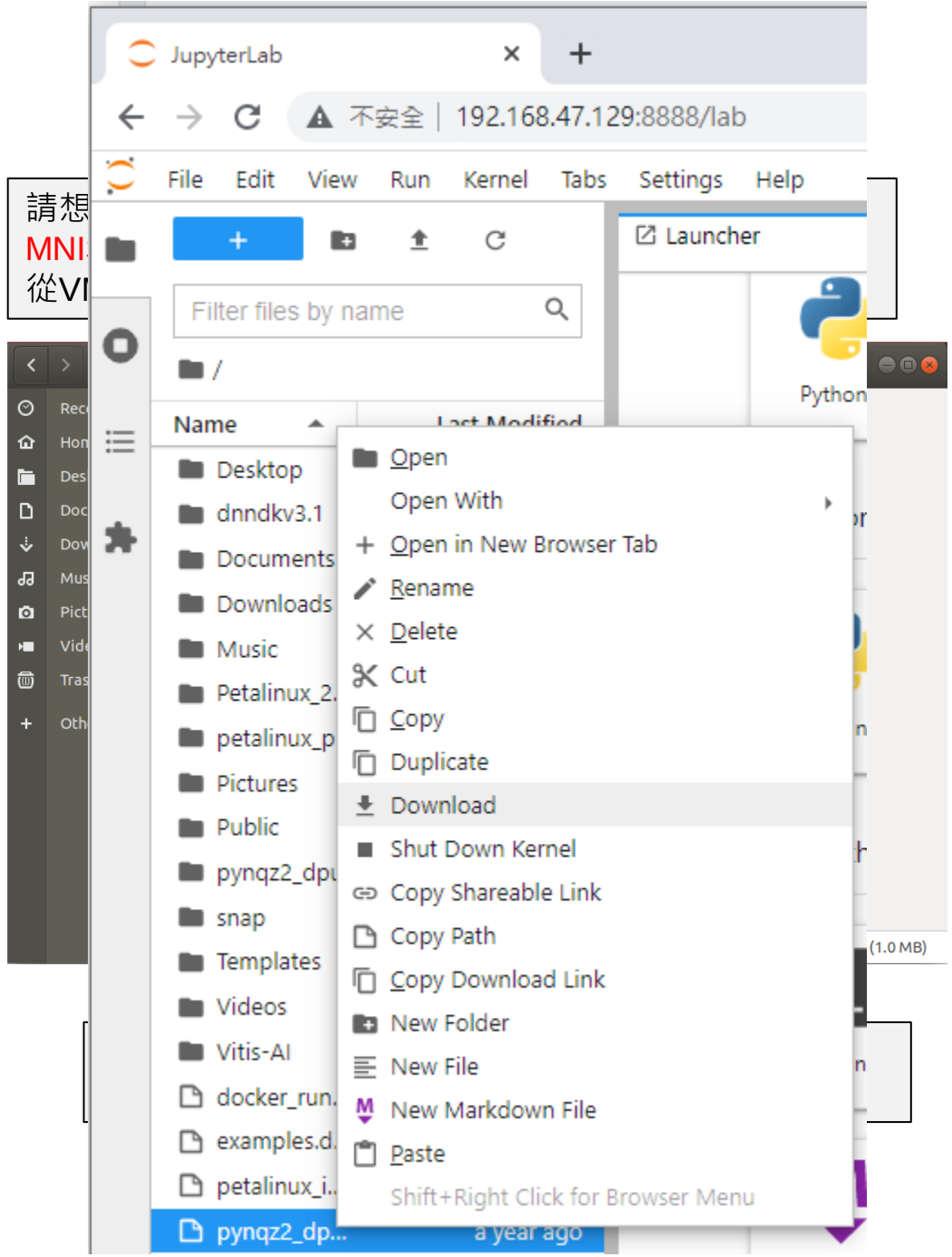


Copy mnist.elf from VM

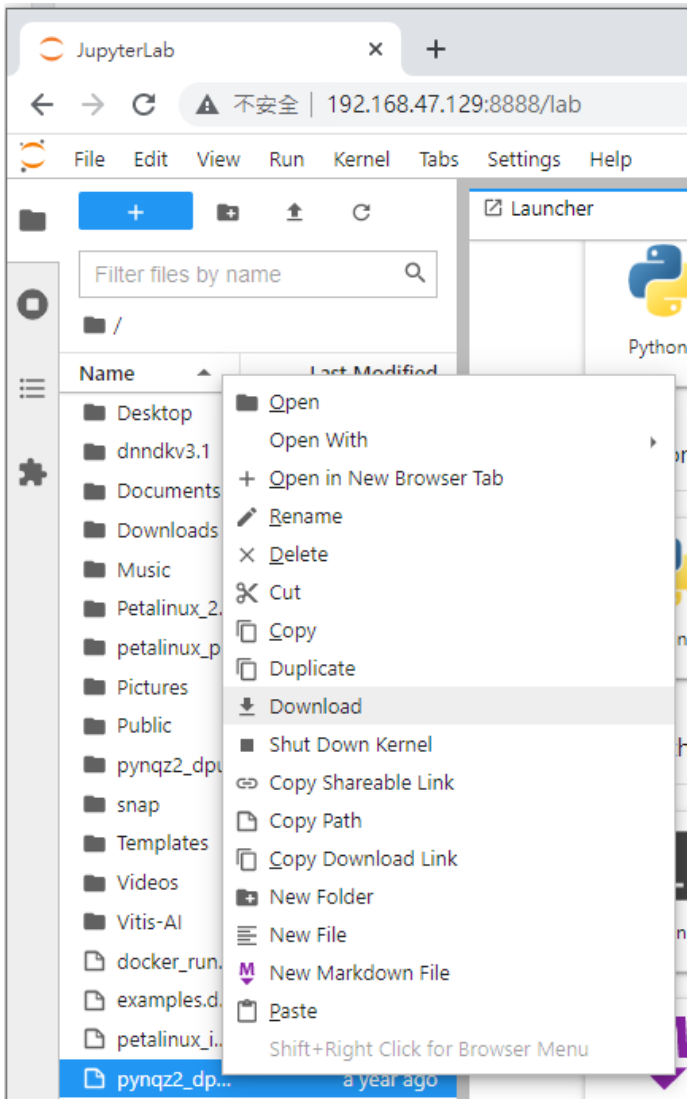
Trying to copy
MNIST_DNNDK_lab/compile/dpu_mnist.elf
from VM to Windows !



Do not use ctrl + c or usb dirve. Please use cloud drive or other ways through internet.



Copy mnist.elf from VM



Example

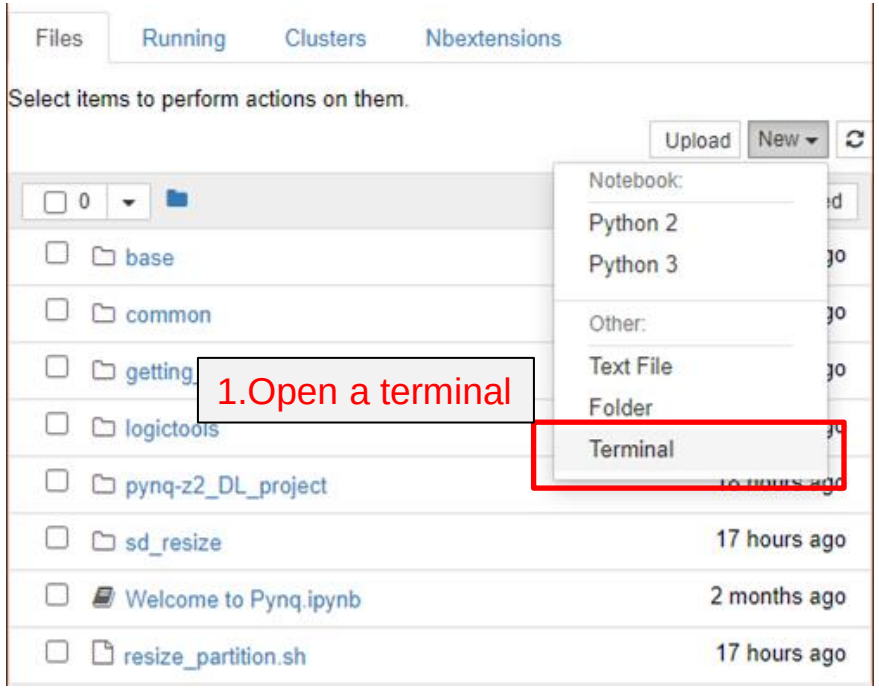
若是使用windows訪問VM中的jupyter-lab，可直接用download功能將檔案透過瀏覽器下載到windows環境中

If the jupyter-lab server in the VM is accessed by the browser under windows, files can be directly downloaded from browser and saved in windows.

DNNDK Lab - Board

Resize partition

Please enter pynqz2's Jupyter notebook on Windows first.



2.cd jupyter_notebooks/

```
root@pynq:/home/xilinx# cd jupyter_notebooks/
```

3. ./resize_partition.sh

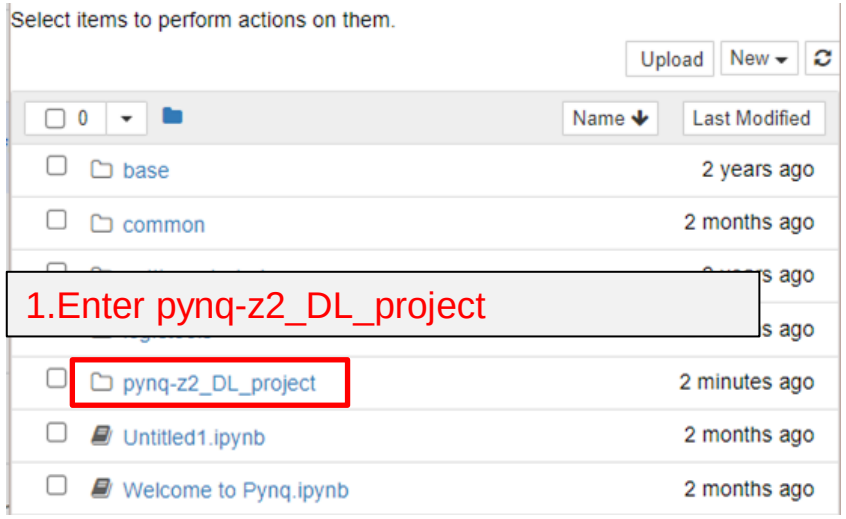
```
root@pynq:/home/xilinx/jupyter_notebooks# ./resize_partition.sh
```

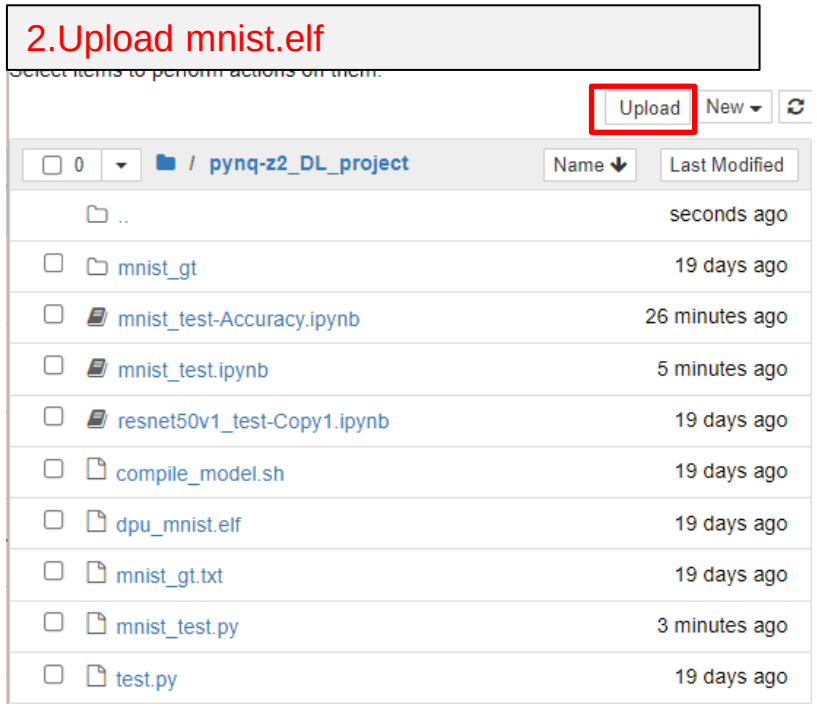
You will see:

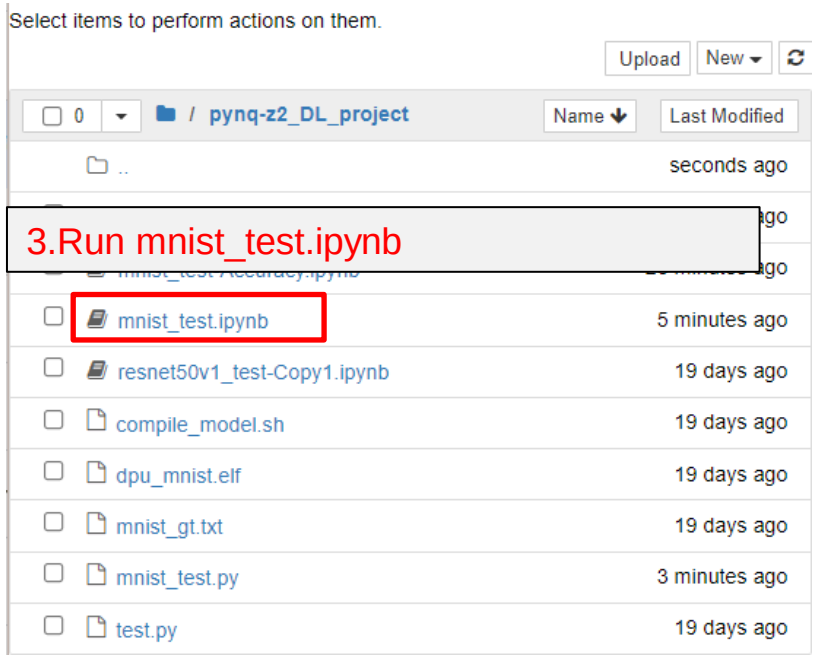
```
root@pynq:/home/xilinx/jupyter_notebooks# ./resize_partition.sh
Checking initial partition info......
Filesystem     1K-blocks    Used Available Use% Mounted on
/dev/root        7226432 6805216     92772  99% /
devtmpfs          123440       0    123440   0% /dev
tmpfs             255024       0    255024   0% /dev/shm
tmpfs             255024    1252    253772   1% /run
tmpfs               5120       0      5120   0% /run/lock
tmpfs             255024       0    255024   0% /sys/fs/cgroup
Install resize tools
```

```
Finished resizing....
checking partition info
Filesystem     1K-blocks    Used Available Use% Mounted on
/dev/root        15084244 6824644   7617064  48% /
devtmpfs          123440       0    123440   0% /dev
tmpfs             255024       0    255024   0% /dev/shm
tmpfs             255024    1252    253772   1% /run
tmpfs               5120       0      5120   0% /run/lock
tmpfs             255024       0    255024   0% /sys/fs/cgroup
done!
root@pynq:/home/xilinx/jupyter_notebooks#
```

Upload mnist.elf and run test notebook





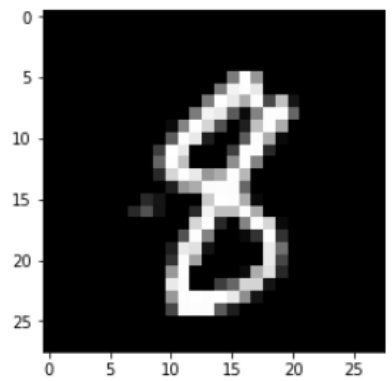


mnist_test.ipynb

```
import sys
sys.path.append('/usr/local/lib/python2.7/dist-packages')
from dnndk import n2cube, dputils
from ctypes import *
import cv2
import numpy as np
import os
import time
from matplotlib import pyplot as plt

def softmax(x):
    """Compute softmax values for each sets of scores in x
    return np.exp(x) / np.sum(np.exp(x), axis=0)

#compile model from elf  to .so
!./compile_model.sh
```

```
'libdpumodelmnist.so' -> '/usr/lib/libdpumodelmnist.so'
```

```
#Load Images
img_path = './mnist_gt'
img = cv2.imread(os.path.join(img_path,'5413.png'))
plt.imshow(img,cmap='gray')

#Pre-processing
img=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
img = img / 255.0
```

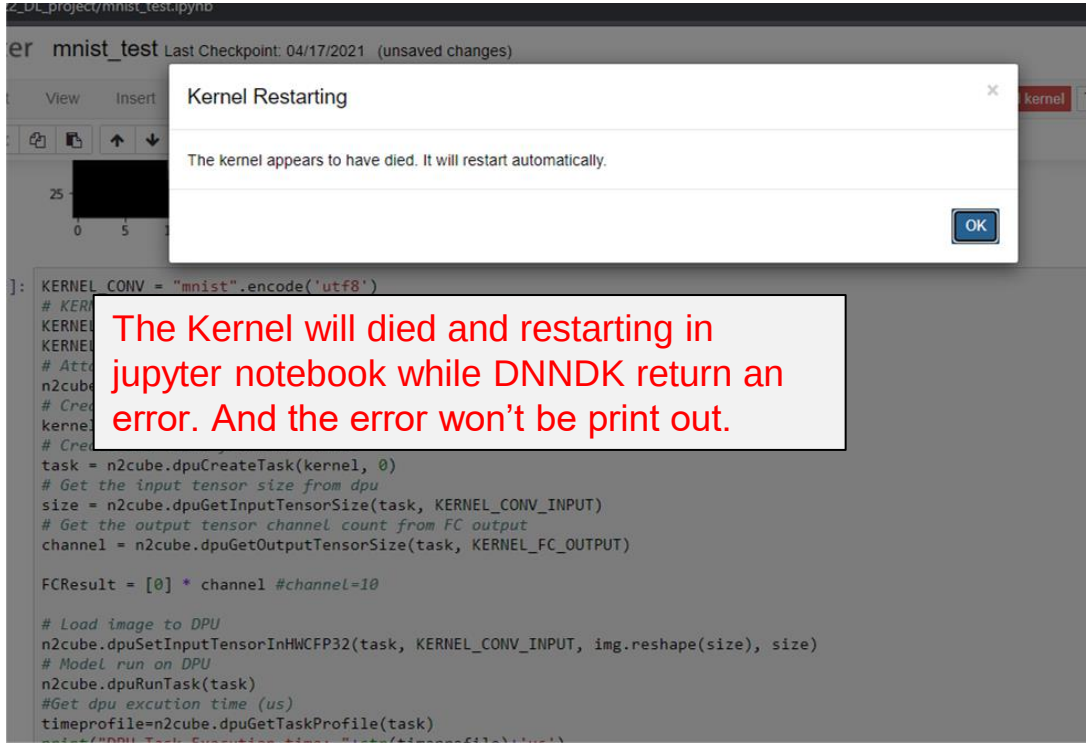



```
KERNEL_CONV = "mnist".encode('utf8')
KERNEL_CONV_INPUT = "conv2d_Conv2D".encode('utf8')
KERNEL_FC_OUTPUT = "dense_1_MatMul".encode('utf8')
# Attach to DPU driver and prepare for runing
n2cube.dpuOpen()
# Create DPU Kernels for ResNet50
kernel = n2cube.dpuLoadKernel(KERNEL_CONV)
# Create DPU Tasks from DPU Kernel
task = n2cube.dpuCreateTask(kernel, 0)
# Get the input tensor size from dpu
size = n2cube.dpuGetInputTensorSize(task, KERNEL_CONV_INPUT)
# Get the output tensor channel count from FC output
channel = n2cube.dpuGetOutputTensorSize(task, KERNEL_FC_OUTPUT)

FCResult = [0] * channel #channel=10

# Load image to DPU
n2cube.dpuSetInputTensorInHWCFP32(task, KERNEL_CONV_INPUT, img.reshape(size), size)
# Model run on DPU
n2cube.dpuRunTask(task)
#Get dpu excution time (us)
timeprofile=n2cube.dpuGetTaskProfile(task)
print("DPU Task Execution time: "+str(timeprofile)+'us')
# Get the output from FC output
n2cube.dpuGetOutputTensorInHWCFP32(task, KERNEL_FC_OUTPUT, FCResult, channel)
#Print Raw dpu output
print("Dpu output: "+str(FCResult))
#caculate softmax
print("Softmax output: ", softmax(FCResult).round(3))
# Get the label
label = FCResult.index(max(FCResult))
print("predict answer: "+str(label))

#close DPU
n2cube.dpuDestroyTask(task)
n2cube.dpuDestroyKernel(kernel)
n2cube.dpuClose()
```

```
DPU Task Execution time: 603us
Dpu output: [-3.875, -2.375, -0.5, 3.375, 0.75, 0.0, -4.0, -3.5, 8.0, 1.375]
Softmax output:  [ 0.     0.     0.     0.01   0.001  0.     0.     0.     0.988  0.001]
predict answer: 8
```

Note: About debugging

DNNDK n2cube is hard to debug in jupyter notebook since the notebook is unable to show error code from n2cube. It's recommended to write the code in .py file and execute in command line.



The Kernel will died and restarting in jupyter notebook while DNNDK return an error. And the error won't be print out.

```
root@pynq:/home/xilinx# cd jupyter_notebooks/pynq-z2_DL_project/
root@pynq:/home/xilinx/jupyter_notebooks/pynq-z2_DL_project# python2 mnist_test.py
[DNNDK] Invalid Node name conv2d_ specified for DPU kernel mnist.

root@pynq:/home/xilinx/jupyter_notebooks/pynq-z2_DL_project#
```

Under command line, the error return from DNNDK will be print out and help you debugging.

Lab goal

- Please trying to modify the code and get the result of model accuracy, FPS
- Please calculate the FPS in entire flow, including pre-processing and post-processing

Ex:

```
accuracy=97.95%
Inference Time=47.8829276561737O6s, FPS:208.84270217990036
```

Hint: DPU Task can be reuse

Ground truth is in mnist_gt.txt

Image file path

```
./mnist_gt/0000.png:7
./mnist_gt/0001.png:2
./mnist_gt/0002.png:1
./mnist_gt/0003.png:0
./mnist_gt/0004.png:4
./mnist_gt/0005.png:1
./mnist_gt/0006.png:4
./mnist_gt/0007.png:9
./mnist_gt/0008.png:5
```

answer

Mnist ground truth

Load ground truth reference

```
#Load Image list & gt
gt_path='./mnist_gt.txt'
with open(gt_path,'r') as f:
    gt_list,img_list=[],[]
    for line in f.readlines():
        img_list.append(line.split(':')[0])
        gt_list.append(int(line.split(':')[1].replace('\n','')))
```

FPS calculate reference

```
import time
total_image_counts = len(img_list) # 10000 test images in mnist data set
start=time.time()
'''Your code here'''
end=time.time()
fps= total_image_counts / (end-start)
```