실험10 Hyper-parameter tuning for board implementation

작성자: 황현하

• 실험 목적

실험2에서는 best accuracy를 갖는 model을 생성하는 것을 목표로 MNIST 데이터셋을 이용한 hyper-parameter optimization 실습을 진행하였다. 하지만 board implementation을 위해서는 accuracy 외에도 resource utilization을 고려해야 한다. 본 실험에서는 quantized bit과 pruning sparsity 등 board implementation을 위해 고려해야할 hyper-parameter를 추가적으로 고려하여 tuning하는 실험을 진행한다.

• 실험 전에 준비해야할 내용

Ray는 distributed application을 위해 개발된 간단하고 범용적인 API이다. Distributed application을 build하고 run하기 위한 간단한 primitive를 제공하고 코드를 거의 변화시키지 않고 하나의 machine code를 병렬화 할 수 있다. 뿐만 아니라 복잡한 application의 구현을 위한 다양한 application, library, tool의 ecosystem을 제공한다.

Ray Core 상에는 machine learning에 적용하기 위한 Tune(Scalable Hyperparameter Tuning), RLlib(Industry-Grade Reinforcement Learning), Ray Train(Distributed Deep Learning), Datasets(Distributed Data Loading and Compute) 등의 library가 존재한다.

Tune은 어떠한 scale에도 적용할 수 있는 hyper-parameter tuning을 위한 python library이다. 중요한 특징으로는 아래와 같은 것들이 있다.

- 10줄 이하의 code로 multi-node distributed hyper-parameter sweep을 실행한다.
- PyTorch, XGBoost, MXNet, Keras 등을 포함한 어떠한 machine learning framework에도 적용이 가능하다.
- 자동으로 checkpoint를 관리하고 TensorBoard상에 logging한다.
- Population based training(PBT), BayesOptSearch, HyperBand/ASHA와 같은 최신 알고리즘에 적용이 가능하다.

Ray에 대한 자세한 내용은 아래의 링크에서 확인 가능하다.

https://docs.ray.io/

• 실습 진행을 위한 환경 세팅

이 실습은 Host 서버에 접속하기 위해 terminal 프로그램을 설치하여 미리 만들어 놓은 VItis-AI Docker 환경에서 실습을 진행한다. Host 서버는 서울대학교 전기정보공학보 실습용 서버이며 세팅 된 환경에서 실습을 진행한다.

a) Terminal 설치 및 사용

Host 서버에 접속하기 위해 MobaXterm terminal 프로그램을 설치하는 방법.

MobaXterm 다운로드 주소 https://mobaxterm.mobatek.net/

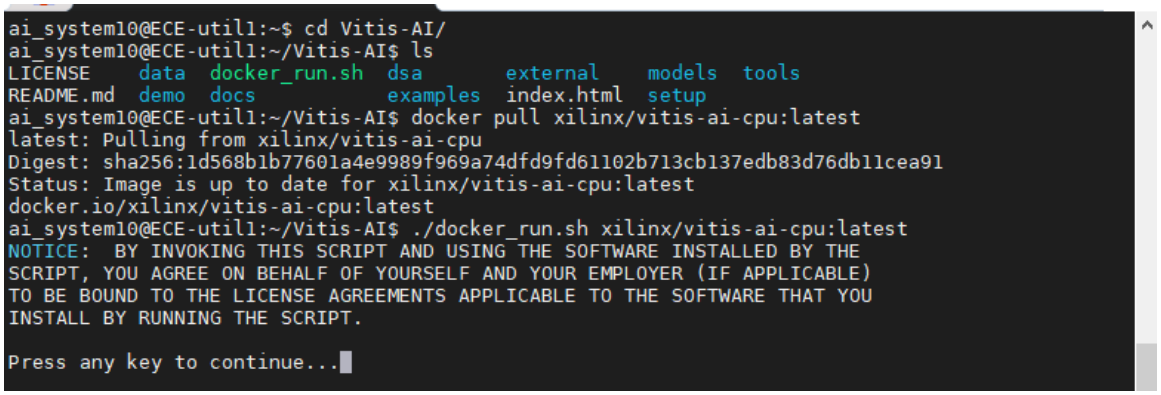
b) Vitis-AI docker 실행

VItis-AI Docker 환경을 실행하는 명령어

$cd Vitis-AI

$docker pull xilinx/vitis-ai-cpu:latest

$./docker_run.sh xilinx/vitis-ai-cpu:latest



```
Do you agree to the terms and wish to proceed [y/n]? y
Setting up ai_system10 's environment in the Docker container...
Running as vitis-ai-user with ID 0 and group 0


==========================================


__      ___ _   _                 /\   |_____|
\ \    / (_) | (_)               /  \    | |
 \ \  / / _| |_ _ ___ _____     / /\ \   | |
  \ \/ / | | __| / __|_____|   / /__\ \  | |
   \  /  | | |_| \__ \       / ______ \_| |_
    \/   |_|\__|_|___/      /_/      \_\_____|

==========================================

Docker Image Version:  1.4.916
Build Date: 2021-07-20
VAI_ROOT: /opt/vitis_ai

For TensorFlow 1.15 Workflows do:
     conda activate vitis-ai-tensorflow
For Caffe Workflows do:
     conda activate vitis-ai-caffe
For PyTorch Workflows do:
     conda activate vitis-ai-pytorch
For TensorFlow 2.3 Workflows do:
     conda activate vitis-ai-tensorflow2
Vitis-AI /workspace >
```

현재까지의 실험 환경 세팅 과정은 기존의 실험 방식과 동일하다. 하지만 hls4ml 패키지와 ray[tune]을 이용하기 위해서는 conda 환경을 새로 설정해 주어야 한다. 실험에 필요한 환경은 제공되는 코드의 environment_ray.yml에 적혀 있다.

```yaml
name: hls4ml_ray
channels:
  - conda-forge
dependencies:
  - python=3.7
  - jupyterhub
  - pydot
  - graphviz
  - pip
  - pip:
    - jupyter
    - tensorflow==2.3.1
    - git+https://github.com/google/qkeras.git#egg=qkeras
    - scikit-learn
    - git+https://github.com/thesps/conifer.git
    - matplotlib
    - pandas
    - pyyaml
    - seaborn
    - ray[tune]
```

위와 같은 환경을 이용하기 위해 다음 명령어를 이용하면 된다.

$conda env create -f environment_ray.yml

$conda activate hls4ml_ray

- **Hyper-parameter optimization**

Terminal 상에서 vim을 이용하여 python code를 작성한다.

$vim tune_mnist_ray.py

a) 프로그램 실행을 위해 필요한 패키지를 import 한다.

```
import argparse
import os

from filelock import FileLock
from tensorflow.keras.datasets import mnist

import ray
from ray import tune
from ray.tune.schedulers import AsyncHyperBandScheduler
from ray.tune.integration.keras import TuneReportCallback

from qkeras.qlayers import QDense, QActivation
from qkeras.quantizers import quantized_bits, quantized_relu
from qkeras.qconvolutional import QConv2D
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dropout,Flatten,InputLayer, MaxPooling2D, Activation
import numpy as np
```

b) Qkeras를 이용하여 MNIST dataset을 traning하는 코드를 참고하여 train_mnist 함수를 작성한다. config은 tuning할 hyper-parameter에 대한 정보를 담고 있다.

```
def train_mnist(config):
    import tensorflow as tf
    from callbacks import all_callbacks

    epochs = 10
    (x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

    x_train = x_train.astype("float32") / 255
    x_test = x_test.astype("float32") / 255
    x_train = np.expand_dims(x_train,-1)
    x_test = np.expand_dims(x_test,-1)
    y_train = tf.keras.utils.to_categorical(y_train, 10)
    y_test = tf.keras.utils.to_categorical(y_test, 10)

    input_shape=(28,28,1)
    model = Sequential()
    model.add(InputLayer(input_shape=input_shape))
    model.add(QConv2D(16, kernel_size=(3, 3),kernel_quantizer=quantized_bits(config["quant_bit"],0,alpha=1),  bias_quantizer=quantized_bits(config["quant_bit"],0,alpha=1)))
    model.add(QActivation(activation=quantized_relu(config["quant_bit"])))
    model.add(MaxPooling2D(pool_size=(2,2)))
    model.add(QConv2D(16, kernel_size=(3, 3),kernel_quantizer=quantized_bits(config["quant_bit"],0,alpha=1),  bias_quantizer=quantized_bits(config["quant_bit"],0,alpha=1)))
    model.add(QActivation(activation=quantized_relu(config["quant_bit"])))
    model.add(MaxPooling2D(pool_size=(2,2)))
    model.add(Flatten())
    model.add(Dropout(0.5))
    model.add(QDense(10,kernel_quantizer=quantized_bits(config["quant_bit"],0,alpha=1),  bias_quantizer=quantized_bits(config["quant_bit"],0,alpha=1)))
    model.add(Activation(activation='softmax'))
    model.build()

    from tensorflow_model_optimization.python.core.sparsity.keras import prune, pruning_callbacks, pruning_schedule
    from tensorflow_model_optimization.sparsity.keras import strip_pruning
    pruning_params = {"pruning_schedule" : pruning_schedule.ConstantSparsity(config["sparsity"], begin_step=0, frequency=100)}
    model = prune.prune_low_magnitude(model, **pruning_params)
    adam= tf.keras.optimizers.Adam(lr=config["lr"])
    model.compile(
        loss=["categorical_crossentropy"],
        optimizer=adam,
        metrics=["accuracy"])
    callbacks = all_callbacks(stop_patience = 1000,
                              lr_factor = 0.5,
                              lr_patience = 10,
                              lr_epsilon = 0.00001,
                              lr_cooldown = 2,
                              lr_minimum = 0.0000001,
                              outputDir = 'model_mnist_tuning')
    callbacks.callbacks.append(pruning_callbacks.UpdatePruningStep())
    callbacks.callbacks.append(TuneReportCallback({"mean_accuracy":"accuracy"}))
    model.fit( x_train, y_train,batch_size=128, epochs=epochs, validation_split=0.2, shuffle=True, callbacks=callbacks.callbacks)
    model = strip_pruning(model)
```

c) tuning에 대한 정보를 담고 있는 tune_mnist함수를 작성한다. 이때 name에는 본인의

실습 번호를 작성한다.

```
def tune_mnist(num_training_iterations):
    sched = AsyncHyperBandScheduler(
        time_attr="training_iteration", max_t=400, grace_period=20)

    analysis = tune.run(
        train_mnist,
        name="student#",
        scheduler=sched,
        metric="mean_accuracy",
        mode="max",
        stop={
            "mean_accuracy": 0.99,
            "training_iteration": num_training_iterations
        },
        num_samples=10,
        resources_per_trial={
            "cpu": 2,
            "gpu": 0
        },
        config={
            "threads": 2,
            "lr": tune.loguniform(0.0001, 0.1),
            "sparsity" : tune.uniform(0.2,0.8),
            "quant_bit":tune.choice([4,8,16])
        })
    print("Best hyperparameters found were: ", analysis.best_config)
```

본인의 실습 번호

d) main 함수를 작성한다.

```
if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument(
        "--smoke-test", action="store_true", help="Finish quickly for testing")
    parser.add_argument(
        "--server-address",
        type=str,
        default=None,
        required=False,
        help="The address of server to connect to if using "
        "Ray Client.")
    args, _ = parser.parse_known_args()
    if args.smoke_test:
        ray.init(num_cpus=4)
    elif args.server_address:
        ray.init(f"ray://{args.server_address}")

    tune_mnist(num_training_iterations=5 if args.smoke_test else 300)
```

e) :wq를 입력하여 vim에 작성한 코드를 저장하고 terminal창으로 돌아온다. 아래의 명령어를 통해 작성한 코드를 실행한다.

$python tune_mnist_ray.py

f) 실행 결과를 확인한다.

Trial name	status	loc	lr	quant_bit	sparsity	acc	iter	total time (s)
train_mnist_d6eea_00000	TERMINATED	147.46.121.38:1380	0.0821527	16	0.352726	0.747521	10	151.533
train_mnist_d6eea_00001	TERMINATED	147.46.121.38:1382	0.00159374	4	0.586115	0.933083	10	255.979
train_mnist_d6eea_00002	TERMINATED	147.46.121.38:1381	0.0180424	8	0.548283	0.954146	10	254.699
train_mnist_d6eea_00003	TERMINATED	147.46.121.38:1383	0.0463168	8	0.485828	0.896187	10	160.755
train_mnist_d6eea_00004	TERMINATED	147.46.121.38:1384	0.0299614	8	0.432519	0.927437	10	253.482
train_mnist_d6eea_00005	TERMINATED	147.46.121.38:1389	0.000828098	8	0.532399	0.93475	10	255.08
train_mnist_d6eea_00006	TERMINATED	147.46.121.38:1388	0.000150681	16	0.62216	0.859271	10	161.929
train_mnist_d6eea_00007	TERMINATED	147.46.121.38:1386	0.0570404	8	0.516026	0.854563	10	160.094
train_mnist_d6eea_00008	TERMINATED	147.46.121.38:1385	0.000121393	8	0.354302	0.883833	10	162.396
train_mnist_d6eea_00009	TERMINATED	147.46.121.38:1387	0.000300546	8	0.225767	0.936562	10	163.26

g) 실행 결과 파일을 확인한다. $cd ~/ray_results/student#/를 입력하면 본인의 실험 결과를 확인할 수 있다.

```
'train_mnist_d6eea_00000_0_lr=0.082153,quant_bit=16,sparsity=0.35273_2021-12-05_00-42-04'/
'train_mnist_d6eea_00001_1_lr=0.0015937,quant_bit=4,sparsity=0.58611_2021-12-05_00-42-04'/
'train_mnist_d6eea_00002_2_lr=0.018042,quant_bit=8,sparsity=0.54828_2021-12-05_00-42-04'/
'train_mnist_d6eea_00003_3_lr=0.046317,quant_bit=8,sparsity=0.48583_2021-12-05_00-42-04'/
'train_mnist_d6eea_00004_4_lr=0.029961,quant_bit=8,sparsity=0.43252_2021-12-05_00-42-04'/
'train_mnist_d6eea_00005_5_lr=0.0008281,quant_bit=8,sparsity=0.5324_2021-12-05_00-42-04'/
'train_mnist_d6eea_00006_6_lr=0.00015068,quant_bit=16,sparsity=0.62216_2021-12-05_00-42-04'/
'train_mnist_d6eea_00007_7_lr=0.05704,quant_bit=8,sparsity=0.51603_2021-12-05_00-42-04'/
'train_mnist_d6eea_00008_8_lr=0.00012139,quant_bit=8,sparsity=0.3543_2021-12-05_00-42-04'/
'train_mnist_d6eea_00009_9_lr=0.00030055,quant_bit=8,sparsity=0.22577_2021-12-05_00-42-04'/
```

h) 본인이 원하는 실험 결과 폴더로 이동하여 생성된 모델을 확인한다.

```
(hls4ml_ray) Vitis-AI ~/ray_results/student#/train_mnist_d6eea_00002_2_lr=0.018042,quant_bit=8,sparsity=0.54828_2021-12-05_00-42-04/model_mnist_tuning > l
full_info.log             KERAS_check_best_model_weights.h5  KERAS_check_model_last.h5          logs/
KERAS_check_best_model.h5  KERAS_check_model_epoch10.h5       KERAS_check_model_last_weights.h5  losses.log
```

- **Jupyter notebook 실행**

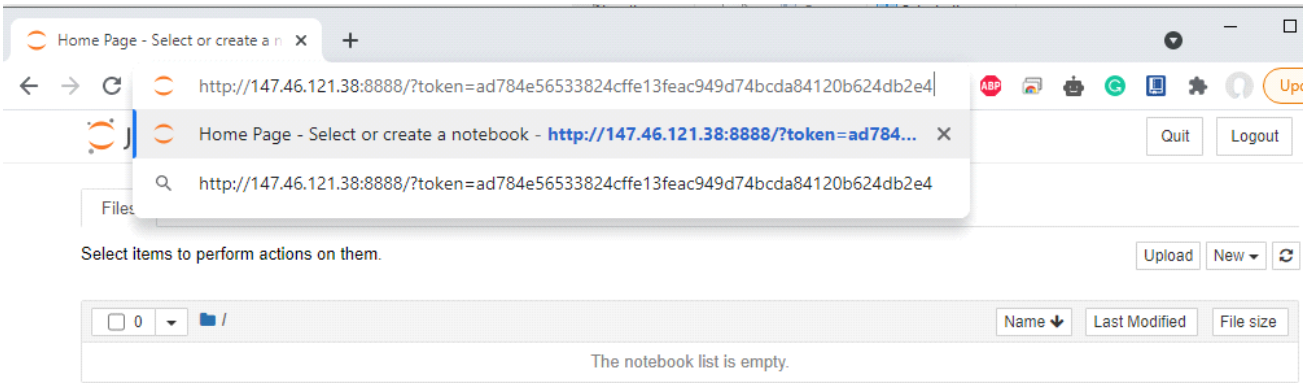
기존 실험과 마찬가지로 본 실험에서도 jupyter notebook을 이용하여 실험을 진행한다.

$ jupyter notebook

```
(vitis-ai-tensorflow) Vitis-AI /workspace/home/student0 > jupyter notebook --ip=147.46.121.38
[I 23:07:41.532 NotebookApp] Writing notebook server cookie secret to /home/vitis-ai-user/.loc
al/share/jupyter/runtime/notebook_cookie_secret
[I 23:07:41.826 NotebookApp] Serving notebooks from local directory: /workspace/home/student0
[I 23:07:41.827 NotebookApp] Jupyter Notebook 6.3.0 is running at:
[I 23:07:41.827 NotebookApp] http://147.46.121.38:8888/?token=ad784e56533824cffe13feac949d74bc
da84120b624db2e4
[I 23:07:41.827 NotebookApp]  or http://127.0.0.1:8888/?token=ad784e56533824cffe13feac949d74bc
da84120b624db2e4
[I 23:07:41.827 NotebookApp] Use Control-C to stop this server and shut down all kernels (twic
e to skip confirmation).
[W 23:07:41.832 NotebookApp] No web browser found: could not locate runnable browser.
[C 23:07:41.832 NotebookApp]

    To access the notebook, open this file in a browser:
        file:///home/vitis-ai-user/.local/share/jupyter/runtime/nbserver-140-open.html
    Or copy and paste one of these URLs:
        http://147.46.121.38:8888/?token=ad784e56533824cffe13feac949d74bcda84120b624db2e4
     or http://127.0.0.1:8888/?token=ad784e56533824cffe13feac949d74bcda84120b624db2e4
```

주소를 복사하여 인터넷 브라우저에서 jupyter notebook 접속



- **Hls4ml 모델로 변환**

a) MNIST dataset을 tf.keras에서 load한다.

```
import tensorflow as tf
import numpy as np
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
# Scale images to the [0, 1] range
x_train = x_train.astype("float32") / 255
x_test = x_test.astype("float32") / 255
# Make sure images have shape (28, 28, 1)
x_train = np.expand_dims(x_train, -1)
x_test = np.expand_dims(x_test, -1)
y_train = tf.keras.utils.to_categorical(y_train, 10)
y_test = tf.keras.utils.to_categorical(y_test, 10)
```

b) dataset shape을 확인한다.

```
print(x_train.shape, x_test.shape,y_train.shape, y_test.shape)
```

```
(60000, 28, 28, 1) (10000, 28, 28, 1) (60000, 10) (10000, 10)
```

c)Ray tune을 통해 생성된 모델 중 원하는 모델을 load한다.

```
from tensorflow_model_optimization.python.core.sparsity.keras import prune
from tensorflow_model_optimization.sparsity.keras import strip_pruning
from qkeras.utils import load_qmodel
with prune.prune_scope():
    model = load_qmodel('/home/vitis-ai-user/ray_results/student#/train_mnist_d6eea_00002_2_lr=0.018042
    model = strip_pruning(model)
```

d)hls4ml 모델로 변환 전 accuracy, number of parameter 등을 확인한다.

```
from sklearn.metrics import accuracy_score
y_qkeras=model.predict(x_test)
print("Accuracy pruned, quantized: {}".format(accuracy_score(np.argmax(y_test, axis=1), np.argmax(y_qkeras, axis=1))))
model.summary()
```

```
Accuracy pruned, quantized: 0.9802
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
q_conv2d (QConv2D)           (None, 26, 26, 16)        160
_________________________________________________________________
q_activation (QActivation)   (None, 26, 26, 16)        0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 16)        0
_________________________________________________________________
q_conv2d_1 (QConv2D)         (None, 11, 11, 16)        2320
_________________________________________________________________
q_activation_1 (QActivation) (None, 11, 11, 16)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 5, 16)          0
_________________________________________________________________
flatten (Flatten)            (None, 400)               0
_________________________________________________________________
dropout (Dropout)            (None, 400)               0
_________________________________________________________________
q_dense (QDense)             (None, 10)                4010
_________________________________________________________________
activation (Activation)      (None, 10)                0
=================================================================
Total params: 6,490
Trainable params: 6,490
Non-trainable params: 0
_________________________________________________________________
```

e)hls4ml 모델로 변환 후 Performance를 확인한다.

```
import hls4ml
from hls4ml.converters.keras_to_hls import keras_to_hls
import plotting
import yaml

hls4ml.model.optimizer.OutputRoundingSaturationMode.layers = ['Activation']
hls4ml.model.optimizer.OutputRoundingSaturationMode.rounding_mode = 'AP_RND'
hls4ml.model.optimizer.OutputRoundingSaturationMode.saturation_mode = 'AP_SAT'



config = hls4ml.utils.config_from_keras_model(model, granularity='name')
config['Backend']='VivadoAccelerator'
config['OutputDir'] = 'mnist-hls-test-ray-tune'
config['ProjectName'] = 'myproject_mnist_ray_tune'
config['XilinxPart']= 'xczu7ev-ffvc1156-2-e'
config['Board'] = 'zcu104'
config['ClockPeriod'] = 5
config['IOType'] = 'io_stream'
config['HLSConfig']={}
config['HLSConfig']['Model']={}
config['HLSConfig']['Model']=config['Model']
config['HLSConfig']['LayerName']=config['LayerName']

del config['Model']
del config['LayerName']
config['AcceleratorConfig']={}
config['AcceleratorConfig']['Interface'] = 'axi_stream'
config['AcceleratorConfig']['Driver'] = 'python'
config['AcceleratorConfig']['Precision']={}
config['AcceleratorConfig']['Precision']['Input']= 'float'
config['AcceleratorConfig']['Precision']['Output']= 'float'
config['KerasModel'] = model



print("-----------------------------------")
print("Configuration")
plotting.print_dict(config)
print("-----------------------------------")
hls_model = keras_to_hls(config)
hls_model.compile()
y_hls = hls_model.predict(x_test)
```

```
from sklearn.metrics import accuracy_score
print("Accuracy pruned, quantized: {}".format(accuracy_score(np.argmax(y_test, axis=1), np.argmax(y_qkeras, axis=1))))
print("Accuracy hls4ml: {}".format(accuracy_score(np.argmax(y_test, axis=1), np.argmax(y_hls, axis=1))))
```

```
Accuracy pruned, quantized: 0.9802
Accuracy hls4ml: 0.98
```

```
import matplotlib.pyplot as plt
mnist_classes=['0','1','2','3','4','5','6','7','8','9']

fig, ax = plt.subplots(figsize=(9, 9))
_ = plotting.makeRoc(y_test, y_qkeras, mnist_classes, linestyle='--')
plt.gca().set_prop_cycle(None) # reset the colors
_ = plotting.makeRoc(y_test, y_hls,mnist_classes, linestyle=':')

from matplotlib.lines import Line2D
lines = [Line2D([0], [0], ls='-'),
         Line2D([0], [0], ls='--')
         ]
from matplotlib.legend import Legend
leg = Legend(ax, lines, labels=[ 'pruned, quantized', 'hls4ml'],
             loc='lower left', frameon=False)
ax.add_artist(leg)
```

- **Synthesis**

a) Now let's synthesize this quantized, pruned model. The synthesis will take a while

While the C-Synthesis is running, we can monitor the progress looking at the log file by opening a terminal from the notebook home, and executing: tail -f mnist-hls-test4/vivado_hls.log

```
import os
os.environ['PATH'] = '/workspace/home/Xilinx/Vivado/2019.2/bin:' + os.environ['PATH']
hls_model.build(csim=False,synth=True,export=True)
```

b) Check the reports

Print out the reports generated by Vivado HLS. Pay attention to the Utilization Estimates' section in particular this time.

```
hls4ml.report.read_vivado_report('mnist-hls-test-ray-tune')
```

- **Bitstream generation**

a) Generate bitstream. And check the .bit file and .hwh file.

```
import os
os.environ['PATH'] = '/workspace/home/Xilinx/Vivado/2019.2/bin:' + os.environ['PATH']
curr_dir = os.getcwd()
os.chdir('mnist-hls-test-ray-tune')
try:
    os.system('vivado -mode batch -source design.tcl')
except:
    print("Something went wrong, check the Vivado logs")
os.chdir(curr_dir)
```

- # Board Test

a) Check the network condition and the USB connection for ZCU 104 board.

b) move generated bitstream, hwh, x_test.npy, y_test.npy files to the board using terminal.

c) By connecting to the IP adress of the board, open the jupyter notebook file on the board

Board Test

This notebook is to run on the PYNQ! You'll need the bitfile and test dataset file. We will load the bitfile we generated onto the PL of the PYNQ SoC.

More details : https://pynq.readthedocs.io/en/latest/overlay_design_methodology/python_overlay_api.html

This notebook is to run on the PYNQ. You'll need the bitfile and test dataset file from previous parts. We will load the bitfile we generated onto the PL of the PYNQ SoC. More details :

https://pynq.readthedocs.io/en/latest/overlay_design_methodology/python_overlay_api.htm

d) Check accuracy of the model on the board.

```python
from pynq import DefaultHierarchy, DefaultIP, allocate
from pynq import Overlay
from datetime import datetime
import pynq.lib.dma
import numpy as np


class NeuralNetworkOverlay(Overlay):
    def __init__(self, bitfile_name, dtbo=None, download=True, ignore_version=False, device=None):

        super().__init__(bitfile_name, dtbo=dtbo, download=download, ignore_version=ignore_version, device=device)

    def _print_dt(self, timea, timeb, N):
        dt = (timeb - timea)
        dts = dt.seconds + dt.microseconds * 10**-6
        rate = N / dts
        print("Classified {} samples in {} seconds ({} inferences / s)".format(N, dts, rate))
        return dts, rate
    def predict(self, X, y_shape, dtype=np.float32, debug=None, profile=False, encode=None, decode=None):
        if profile:
            timea = datetime.now()
        if encode is not None:
            X = encode(X)
        with allocate(shape=X.shape, dtype=dtype) as input_buffer, \
             allocate(shape=y_shape, dtype=dtype) as output_buffer:
            input_buffer[:] = X
            self.hier_0.axi_dma_0.sendchannel.transfer(input_buffer)
            self.hier_0.axi_dma_0.recvchannel.transfer(output_buffer)
            if debug:
                print("Transfer OK")
            self.hier_0.axi_dma_0.sendchannel.wait()
            if debug:
                print("Send OK")
            self.hier_0.axi_dma_0.recvchannel.wait()
            if debug:
                print("Receive OK")
            result = output_buffer.copy()
        if decode is not None:
            result = decode(result)
        if profile:
            timeb = datetime.now()
            dts, rate = self._print_dt(timea, timeb, len(X))
            return result, dts, rate
        return result
```

```python
if __name__ == '__main__':
    x_test = np.load('x_test.npy')
    y_test = np.load('y_test.npy')

    N = 100

    overlay = NeuralNetworkOverlay('design_1.bit')

    out = overlay.predict(x_test[:N],[N,10])
    predicted = out.argmax(axis=1)

    correct = (predicted == y_test[:N]).sum()
    accuracy = correct / N

    print("Model Accuracy: %.2f%%"%(accuracy*100))
```

- **실험 후 보고서에 포함될 내용**

1. 본 실험에서는 quantized bit과 pruning sparsity 등 board implementation을 위해 고려해야할 다양한 hyper-parameter를 optimization하였다. 이번 실험 결과와 지난 실험9의 결과(hyper-parameter optimization을 수행하지 않은 실험 결과)를 비교하고 결과에 대해 분석하시오. (board test는 생략한다)

2. 본 실험에서는 QKeras의 weight와 activation의 quantized bit을 config['quant_bit']으로

통일하여 optimization을 진행했다. Config을 변경하면 Weight과 activation의 quantized bit을 다르게 바꿀 수도 있다. 지난 실험9 보고서의 2번에서 작성한 weight와 activation의 quantized bit을 이용하여 Ray를 통해 hyper-parameter tuning을 진행하고 실험 결과를 논하시오. (board test는 생략한다)

3. 본 실험에서는 여러 hyper-parameter 중 learning rate, pruning sparsity, quantized bit을 tuning하였다. 이 외에도 board implementation을 위해 고려해야할 hyper-parameter가 무엇이 있는 지 적고 해당 hyper-parameter를 ray를 통해 optimization하시오.(여러가지가 있다면 그 중 하나만 optimize하면 된다, board test는 생략한다)

4. 위의 사항을 종합적으로 고려하여 MNIST dataset에 대해 ZCU104에서 implement 가능한 최적의 model을 구현하고 board에 implement하시오. (board에 implement 가능한 best accuracy를 보이는 model을 구현하면 된다.) board test 결과에 대해 논하시오.

제출 기한 : 2021.12.08(수) 오후 11:59

제출 양식 : 이름_학번_보고서10.pdf (ex. 홍길동_2021-12345_보고서10.pdf)

*보고서는 pdf로 변환하여 제출