Running YOLO V2 with Xilinx ML Suite

AWS F1 instance with AWS FPGA Developer AMI

Sept. 10, 2018.

It is assumed that you have an AWS account, EC2 F1 instance with AWS FPGA developer AMI.

Prepared by Ando Ki (adki@future-ds.com)

FUTURE Design Systems

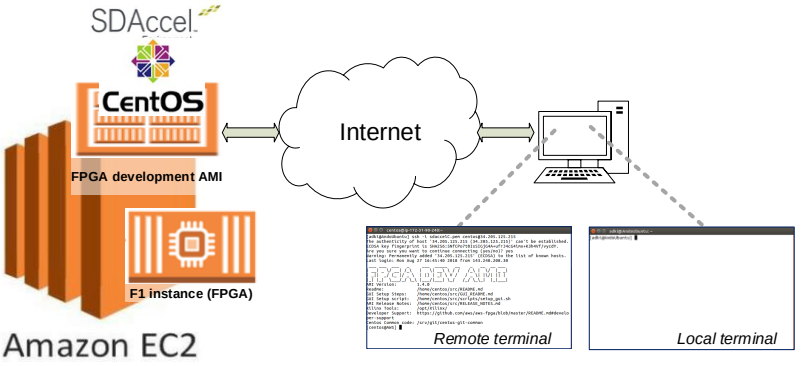
Overall steps

- 1. AWS log in
- 2. Log in F1 instance
- 3. One time settings
- 4. AWS specific steps
  - ►set up SDAccel for AWS environment
  - ►get root permission in order to use FPGA hardware
- 5. Run YOLO
- Scripts
  - ►'run.sh' and 'yolo.py'
- To test your own picture
  - ►'run_yolo_one.sh' and 'yolo_one.py'

Prepared by Ando Ki (adki@future-ds.com)

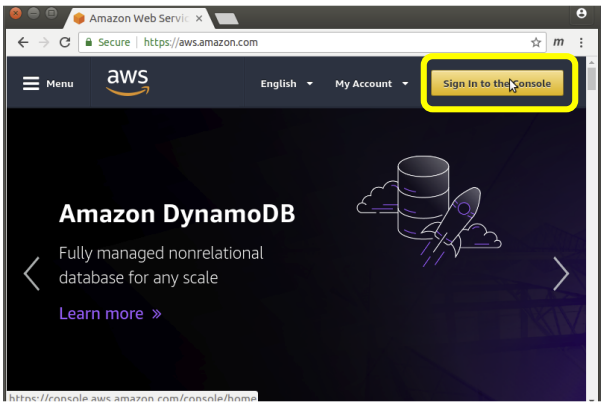
FUTURE Design Systems

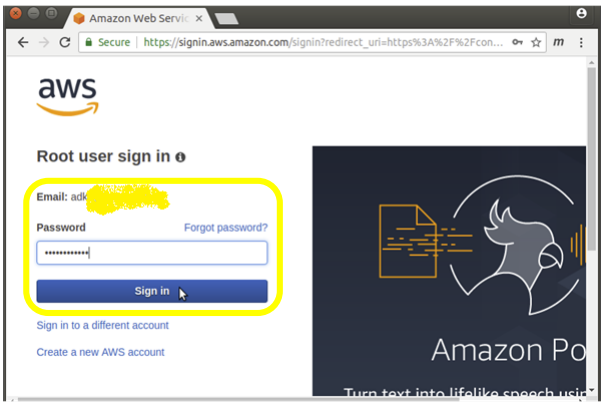
AWS environment at a glance



1. AWS log in

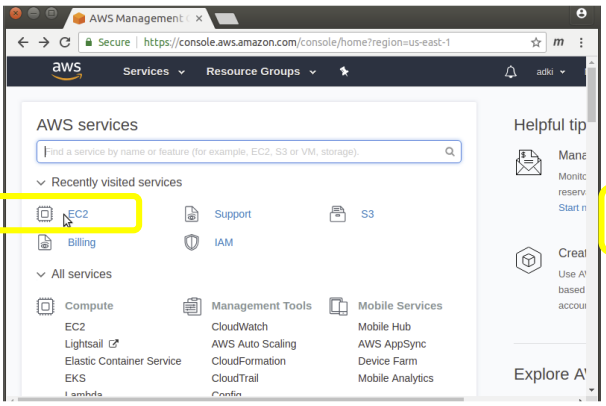
- https://aws.amazon.com
- Use user account



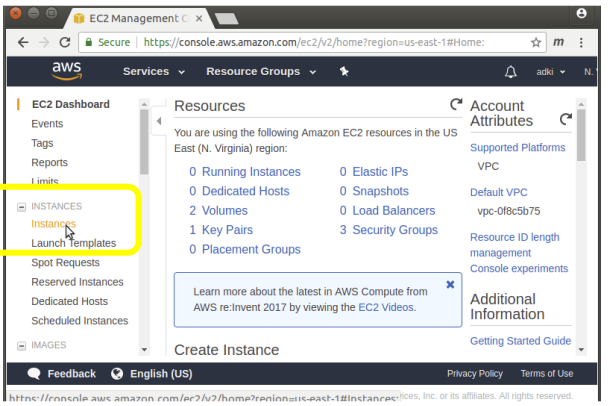


1. AWS log in

■ Select EC2 service



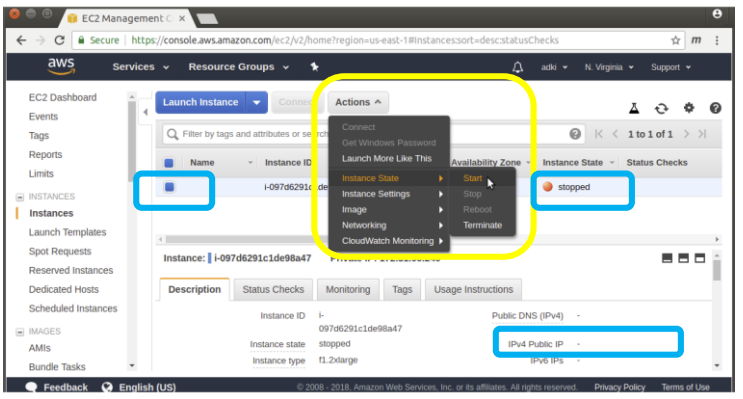
■ Select Instances

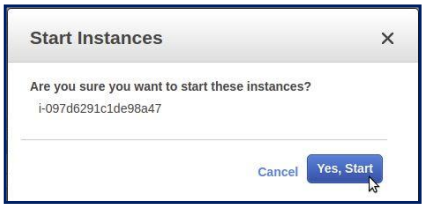


1. AWS log in

■ Start instance

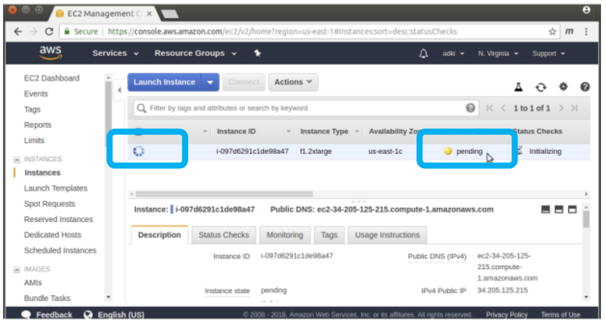
►Actions → Instance State → Start

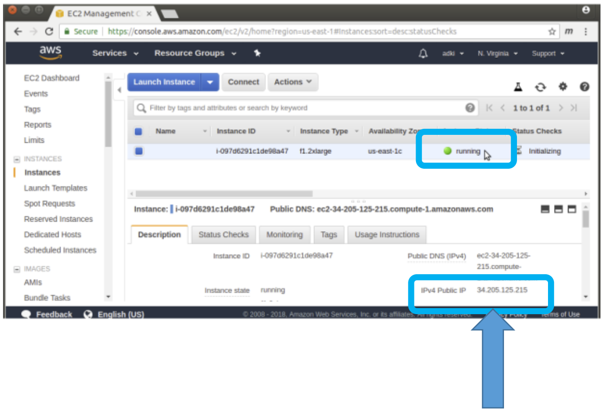




1. AWS log in

- Wait during ‘pending’ state
- Wait for ‘running’ state
  - ►Get IPv4 Public IP





2. F1 instance log in from your local terminal

- Open a terminal and connect to your AWS F1 instance using authentication key (i.e., PEM, privacy enhanced mail file)

- ►Use: ssh
- ►Pem: “your own file”
- ►Login account: centos
- ►IP: Public IPv4 – varying every time you start the instance

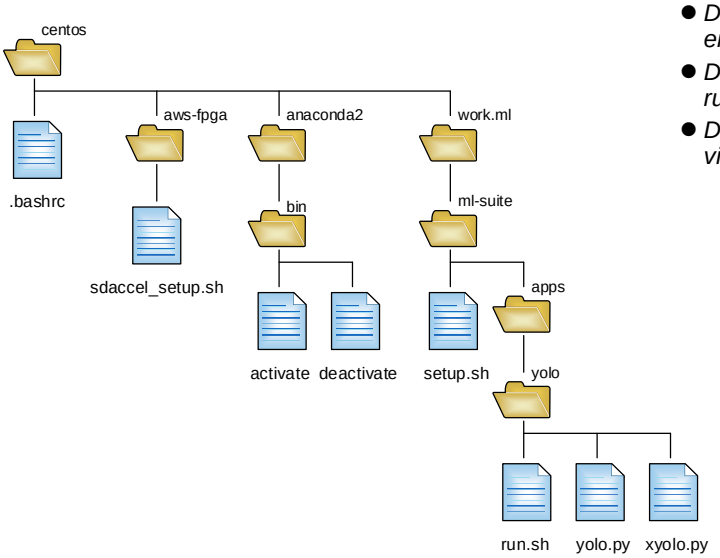
Make your F1 instance stopped, when you leave AWS, after finishing your work on AWS.

Overall steps

- 1. AWS log in
- 2. Log in F1 instance
- 3. One time settings
- 4. AWS specific steps
  - ►set up SDAccel for AWS environment
  - ►get root permission in order to use FPGA hardware
- 5. Run YOLO
- Scripts
  - ►'run.sh' and 'yolo.py'
- To test your own picture
  - ►'run_yolo_one.sh' and 'yolo_one.py'
  - ►'

- 3.1. Install Anaconda2
- 3.2. Create ml-suite Anacodas virtual environment
- 3.3. Get Xilinx ML-Suite
  - ►make sure 'git-lfs' is installed

Directory structure



- *Do not forget to set up AWS FPGA environment after you log on*
- *Do not forget to get root permission before run the application*
- *Do not forget to set up Anaconda2 ml-suit virtual environment*

3.1 Install Anaconda2

- Let make ‘Anaconda2’ directory at the home directory
  - ►$ cd
- Download Anaconda2
  - ►$ wget https://repo.anaconda.com/archive/Anaconda2-5.1.0-Linux-x86_64.sh
- Run the installer (Installer requires bzip, please install it if you don't have it)
  - ►$ bash ./Anaconda2-5.1.0-Linux-x86_64.sh
- Ensure that your .bashrc is preparing Anaconda, by including these lines
  - ►~/.bashrc: export PATH=${HOME}/anaconda2/bin:$PATH
  - ►~/.bashrc: . /${HOME}/anaconda2/etc/profile.d/conda.sh

```
# added by Anaconda2 installer (at .bashrc file)
export PATH="/home/centos/anaconda2/bin:$PATH"
. /home/centos/anaconda2/etc/profile.d/conda.sh
```

Do not use “${HOME}” for “/home/centos”

- After updating the bashrc source it to load the new anaconda path
  - ►$ source ~/.bashrc
- As a precaution unset PYTHONPATH to avoid conflicts with packages on your rootfs
  - ►$ unset PYTHONPATH

```
// To remove Anaconda2
$ conda install anaconda-clean
$ anaconda-clean --yes
// Then remove directories
•~/anaconda2
•~/.anaconda_backup
```


3.2 Create ml-suite anacoda2 virtual environment

- Create Virtual Environment
  - ►$ conda create --name ***ml-suite*** python=2.7 x264=20131218 caffe pydot pydot-ng graphviz keras scikit-learn -c conda-forge
    - you can choose any name for virtual environment, here we pick ‘ml-suite’
- Fix symbolic links between pre-compiled Caffe (libcaffe.so), and C
  - ►$ bash ml-suite/fix_caffe_opencv_symlink.sh
- Activate Environment
  - ►$ conda activate ml-suite
- Verify your environment by importing caffe in python
  - ►(ml-suite) $ python -c "import caffe"
- Install TensorFlow version 1.8 (optional for this example)
  - ►(ml-suite) $ pip install tensorflow==1.8
    - *<There should be no message at all>*
- Install Jupyter to the ml-suite env (optional)
  - ►(ml-suite) $ pip install jupyter ← may not required for command-line case
- Exit from Anaconda: (ml-suit) $ conda deactivate

```
$ conda activate ml-suite
$ conda deactivate
```

```
// To check packages in the conda
$ conda list
```

```
// To check conda environment
$ conda env list
```

```
// To remove conda environment
$ conda-env remove -n ml-suite
or
$ conda remove --name ml-suite --all
```

3.3 Get Xilinx ML-Suite

- Install “git lfs” first
  - ►$ curl -s https://packagecloud.io/install/repositories/github/git-lfs/script.rpm.sh | sudo bash
  - ►$ sudo yum install git-lfs
  - ►$ git lfs install
- Get ML-Suite (let make ‘work.ml’ directory and carry out our example in there)
  - ►$ mkdir work.ml && cd work.ml
  - ►$ git clone https://github.com/Xilinx/ml-suite.git
  - ►$ git pull
  - ►$ git lfs pull

- Git is a distributed version control system, meaning the entire history of the repository is transferred to the client during the cloning process. For projects containing large files, particularly large files that are modified regularly, this initial clone can take a huge amount of time, as every version of every file has to be downloaded by the client.
- Git LFS (Large File Storage) is a Git extension developed by Atlassian, GitHub, and a few other open source contributors, that reduces the impact of large files in your repository by downloading the relevant versions of them lazily. Specifically, large files are downloaded during the checkout process rather than during cloning or fetching.

3.3 Create ml-suite anacoda2 virtual environment: numpy

- Xilinx ML-Suite YOLO2 requires numpy 1.14.1
  - ► If numpy is not 1.14.1, uninstall numpy and then install numpy 1.14.1.

```
$ conda activate ml-suit
(ml-suite) $ python -c “import numpy; print
(numpy.version.version)”
1.15.0
(ml-suite) $ pip uninstall numpy==1.15.0
... ...
(ml-suite) $ pip install numpy==1.14.5
... ...
(ml-suite) $ python -c “import numpy; print
(numpy.version.version)”
1.14.5
(ml-suite)
```

Overall steps

- 1. AWS log in
- 2. Log in F1 instance
- 3. One time settings
- 4. AWS specific steps
  - ►set up SDAccel for AWS environment
  - ►get root permission in order to use FPGA hardware
- 5. Run YOLO
- Scripts
  - ►'run.sh' and 'yolo.py'
- To test your own picture
  - ►'run_yolo_one.sh' and 'yolo_one.py'

- Prepare runtime library
  - ►Get AWS FPGA package
  - ►Run 'sdaccel_setup.sh'
- Use Anaconda VM as root
- Run application with FPGA

4. AWS specific steps for command line (1/2)

- Get AWS FPGA (This step should be done every time you login again.)
  - ►If 'aws-fpga' is ready then skip this step
    - $ cd
    - $ git clone https://github.com/aws/aws-fpga.git
  - ►$ cd
  - ►$ cd aws-fpga
  - ►$ source sdaccel_setup.sh
  - ►checking environment (optional)
    - $ echo $XILINX_SDX
      - ◆"/opt/Xilinx/SDx/2017.4.op"
    - $ source $XILINX_SDX/settings64.sh
      - ◆"$ clinfo -l" does not report any OpenCL devices.

It prepares run-time library for OpenCL.
$(XILINX_OPENCL)=/opt/Xilinx/SDx/2017.4.rte.dyn

```
[centos@AWS] clinfo -l
Linux:3.10.0-862.9.1.el7.x86_64:#1 SMP Mon Jul 16 16:29:36 UTC 2018:x86_64
Distribution: CentOS Linux release 7.4.1708 (Core)
GLIBC: 2.17
---
XILINX_OPENCL=""
LD_LIBRARY_PATH="/opt/Xilinx/SDx/2017.4.op/lnx64/tools/opencv/:/opt/Xilinx/SDx/2017.4.op/lib/lnx64.o/Default:/opt/Xilinx/SDx/2017.4.op/lib/lnx64.o:/opt/Xilinx/SDx/2017.4.op/runtime/lib/x86_64"
---
ERROR: No devices found
[centos@AWS]
```

- Now do prepare xclbin using xcpp and xocc (optional)
  - ►This step is only required when you prepare your own design.

4. AWS specific steps for command line (2/2)

- Become root
  - ►$ cd; sudo su
- Set Environment Variables Required by runtime
  - ►[root@AWS]$ source work.ml/ml-suite/overlaybins/setup.sh aws
- Set User Environment Variables Required to run Anaconda
  - ►[root@AWS]$ source ~centos/.bashrc
- Activate the users Anaconda Virtual Environment
  - ►[root@AWS]$ source activate ml-suite
  - ►(ml-suite) [root@AWS]$
    - Note that now you are root and OpenCL device is ready (check '$ clinfo –l')
  - ►(ml-suite) [root@AWS]$ clinfo -l
- Now OpenCL application can be run
  - ►<following will be given in details at the next step>
    - (ml-suite) [root] cd work.ml/ml-suite/apps/yolo
    - (ml-suite) [root] ./run.sh aws e2e

```
[root@AWS] source activate ml-suite
(ml-suite) [root@AWS] clinfo -l
[0]user:0xf010:0x1d51:[xocl:2017.4.5:128]
xclProbe found 1 FPGA slots with xocl driver running
Linux:3.10.0-862.9.1.el7.x86_64:#1 SMP Mon Jul 16 16:29:36 UTC
2018:x86_64
Distribution: CentOS Linux release 7.4.1708 (Core)
GLIBC: 2.17
---
XILINX_OPENCL="/home/centos/work.ml/ml-suite/overlaybins/aws"
LD_LIBRARY_PATH="/home/centos/work.ml/ml-
suite/overlaybins/aws/runtime/lib/x86_64/:/home/centos/work.ml/ml-
suite/xfdnn/rt/xdnn_cpp/build/lib:/home/centos/work.ml/ml-
suite/xfdnn/rt/lib:/home/centos/work.ml/ml-
suite/ext/boost/lib:/home/centos/work.ml/ml-
suite/ext/zmq/libs:/home/centos"
---
Platform #0: Xilinx
 `-- Device #0: xilinx_aws-vu9p-f1-04261818_dynamic_5_0
(ml-suite) [root@AWS]
```


Overall steps

- 1. AWS log in
- 2. Log in F1 instance
- 3. One time settings
- 4. AWS specific steps
  - ►set up SDAccel for AWS environment
  - ►get root permission in order to use FPGA hardware
- 5. Run YOLO
- Scripts
  - ►'run.sh' and 'yolo.py'
- To test your own picture
  - ►'run_yolo_one.sh' and 'yolo_one.py'

5. Run yolo from beginning (1/4)

- Connect to your remote machine
  - ► $ ssh -i <key> centos@<IPv4 Public IP>
- $ source aws-fpga/sdaccel_setup.sh

It is local machine.

It is remote machine, i.e., AWS F1.

```
centos@ip-172-31-90-240:~
[adki@AndoUbuntu] ssh -i sdaccelC.pem centos@18.204.202.46
The authenticity of host '18.204.202.46 (18.204.202.46)' can't be established.
ECDSA key fingerprint is SHA256:5NfCPo7t0IsSIGjG4A+ufrJ4cG4lmx+K3h4Vf/vycdY.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '18.204.202.46' (ECDSA) to the list of known hosts.
Last login: Sat Sep  8 10:03:50 2018 from 143.248.208.30
 ___ ___ ___  _   ___ _____  __   _  __ __ ___
| __| _ \/ __|/_\ |   \ __\ \/ /  /_\ |  \/  |_ _|
| _||  _/ (_ / _ \| |) | _| \  /  / _ \| |\/| || |
|_| |_|  \___/_/ \_\___/|___| \_/  /_/ \_\_|  |_|___|
AMI Version:          1.4.0
Readme:               /home/centos/src/README.md
GUI Setup Steps:      /home/centos/src/GUI_README.md
GUI Setup script:     /home/centos/src/scripts/setup_gui.sh
AMI Release Notes:    /home/centos/src/RELEASE_NOTES.md
Xilinx Tools:         /opt/Xilinx/
Developer Support:    https://github.com/aws/aws-fpga/blob/master/README.md#develo
per-support
Centos Common code: /srv/git/centos-git-common
[centos@AWS] ls
anaconda2/  aws-fpga/  awsver.txt  INDEX.txt  run_log.txt  src/  work/
work.ml/
[centos@AWS] source aws-fpga/sdaccel_setup.sh
```

```
centos@ip-172-31-90-240:~
install -m 755 /home/centos/aws-fpga/SDAccel/userspace/src2/libxrt-aws.so /opt/X
ilinx/SDx/2017.4.rte.dyn/runtime/platforms/xilinx_aws-vu9p-f1-04261818_dynamic_5
_0/driver
install -m 755 /home/centos/aws-fpga/SDAccel/tools/awssak2/xbsak /opt/Xilinx/SDx
/2017.4.rte.dyn/runtime/bin
install -m 755 /opt/Xilinx/SDx/2017.4.op/runtime/bin/xclbincat /opt/Xilinx/SDx/2
017.4.rte.dyn/runtime/bin
install -m 755 /opt/Xilinx/SDx/2017.4.op/runtime/bin/xclbinsplit /opt/Xilinx/SDx
/2017.4.rte.dyn/runtime/bin
install -m 755 /home/centos/aws-fpga/SDAccel/aws_platform/xilinx_aws-vu9p-f1-042
61818_dynamic_5_0/sw/lib/x86_64/libxilinxopencl.so /opt/Xilinx/SDx/2017.4.rte.dy
n/runtime/lib/x86_64
install -m 755 /opt/Xilinx/SDx/2017.4.op/lib/lnx64.o/Default/libstdc++.so* /opt/
Xilinx/SDx/2017.4.rte.dyn/runtime/lib/x86_64
Generating SDAccel F1 runtime environment setup script, /opt/Xilinx/SDx/2017.4.r
te.dyn/setup.sh for bash
Generating SDAccel F1 runtime environment setup script, /opt/Xilinx/SDx/2017.4.r
te.dyn/setup.csh for (t)csh
XILINX_OPENCL=/opt/Xilinx/SDx/2017.4.rte.dyn
INFO: The default AWS Platform has been set to: "AWS_PLATFORM=$AWS_PLATFORM_DYNA
MIC_5_0"
INFO: SDAccel runtime installed
INFO: SDAccel Setup PASSED
[centos@AWS]
```


5. Run yolo from beginning (2/4)

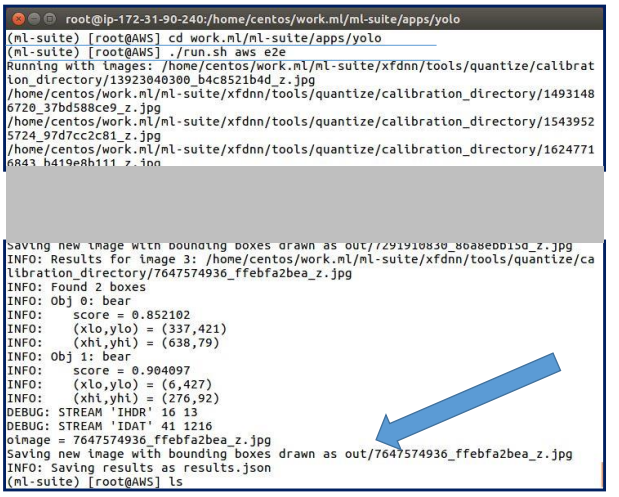
- $ sudo su
- $ source ~centos/.bashrc
- [root@aws]$ source activate ml-suite

```
root@ip-172-31-90-240:/home/centos
[centos@AWS] cd; sudo su
[root@ip-172-31-90-240 centos]# source work.ml/ml-suite/overlaybins/setup.sh aws
make: Entering directory `/home/centos/work.ml/ml-suite/apps/yolo/nms'
cd ./nms_20180209 && make
make[1]: Entering directory `/home/centos/work.ml/ml-suite/apps/yolo/nms/nms_201
80209'
make[1]: Nothing to be done for `all'.
make[1]: Leaving directory `/home/centos/work.ml/ml-suite/apps/yolo/nms/nms_2018
0209'
make: Leaving directory `/home/centos/work.ml/ml-suite/apps/yolo/nms'
[root@ip-172-31-90-240 centos]# source ~centos/.bashrc
[root@AWS] source activate ml-suite
(ml-suite) [root@AWS] clinfo -l
[0]user:0x1042:0x7:[???:??:0]
xclProbe found 1 FPGA slots with xocl driver running
Linux:3.10.0-862.9.1.el7.x86_64:#1 SMP Mon Jul 16 16:29:36 UTC 2018:x86_64
Distribution: CentOS Linux release 7.4.1708 (Core)
GLIBC: 2.17
---
XILINX_OPENCL="/home/centos/work.ml/ml-suite/overlaybins/aws"
LD_LIBRARY_PATH="/home/centos/work.ml/ml-suite/overlaybins/aws/runtime/lib/x86_6
4/:/home/centos/work.ml/ml-suite/xfdnn/rt/xdnn_cpp/build/lib:/home/centos/work.m
l/ml-suite/xfdnn/rt/lib:/home/centos/work.ml/ml-suite/ext/boost/lib:/home/centos
/work.ml/ml-suite/ext/zmq/libs:/home/centos"
---
WARNING: AwsXcl - Cannot open userPF: /dev/dri/renderD0
WARNING: AwsXcl isGood: invalid user handle.
WARNING: xclOpen Handle check failed
[0]user:0xf010:0x1d51:[xocl:2017.4.5:128]
device[0].user_instance : 128
Platform #0: Xilinx
 `-- Device #0: xilinx_aws-vu9p-f1-04261818_dynamic_5_0
(ml-suite) [root@AWS]
```

5. Run yolo from beginning (3/4)

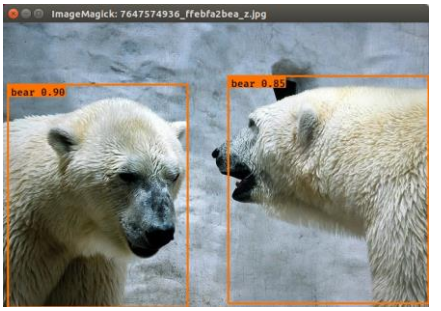
- (ml-suite)[root]$ cd work.ml/ml-suite/apps/yolo
- (ml-suite)[root]$ ./run.sh aws e2e
- (ml-suite) [root] conda deactivate
- [root@AWS] exit
- $

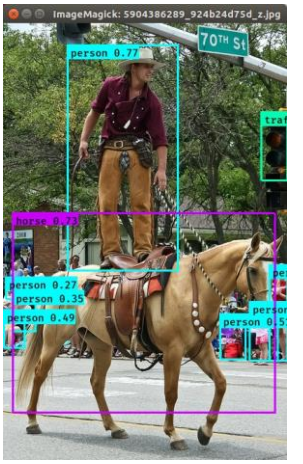
- Now you can copy the results files to the local machine using 'scp'. (note it is not remote)
  - ► $ scp –i key src dst
    - 'key': your pem file
    - 'src': resulting jpg file
      - centos@<ip>:/home/centos/work.ml/apps/yolo/out/*.jpg
    - 'dst': local file name



5. Run yolo from beginning (4/4)

- get result file from remote computer, i.e, AWS F1 instance
- $ scp -i <your key> -r centos@<remote IP>:/home/centos/work.ml/ml-suite/apps/yolo/out out
- $ display out/*.jpg
  - ► <to see next picture, enter "space' key>





Overall steps

- 1. AWS log in
- 2. Log in F1 instance
- 3. One time settings
- 4. AWS specific steps
  - ►set up SDAccel for AWS environment
  - ►get root permission in order to use FPGA hardware
- 5. Run YOLO
- Scripts
  - ►'run.sh' and 'yolo.py'
- To test your own picture
  - ►'run_yolo_one.sh' and 'yolo_one.py'

"run.sh" of yolo (1/2)

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

DEVICE=$1
TEST_TO_RUN=$2
PE_CFG=$3
BITWIDTH=$4
IMGWIDTH=$5

if [ -z $MLSUITE_ROOT ]; then
  MLSUITE_ROOT=../..
fi

XDNN_SIZE=[3,${IMGWIDTH},${IMGWIDTH}]

images=`ls ${MLSUITE_ROOT}/xfdnn/tools/quantize/calibration_directory/*`
echo "Running with images: $images"

XCLBIN=xdnn_v2_32x56_2pe_16b_6mb_bank21.xclbin
if [ "$BITWIDTH" == "8" ]; then
  XCLBIN=xdnn_v2_32x56_2pe_8b_6mb_bank21.xclbin
fi

# Set Enviornment Variables corresponding to HW platform
. ${MLSUITE_ROOT}/overlaybins/setup.sh $DEVICE

# Build Non-Max Suppression C-code
cd nms
make
cd ..
```

“run.sh” of yolo (2/2)

```
echo "=============== pyXDNN ============================="

if [ "$TEST_TO_RUN" == "e2e" ]; then
  ##################
  # End To End, by default this will run 608x608, 16b quantization.
  # Modify configs.py if you want to run a different end to end
  ##################

  python yolo.py

## Note: The xyolo module was written to support being called directly at the CLI
##       yolo.py supports running the offline steps (Compile,Quantize, and bringing in xyolo as a class object)
##       However, xyolo can be called from CLI with command line arguments
##       The code below shows an example of how this can be done, but it won't
##       work unless you have already ran the Compile,Quantize steps, and
##       saved the results locally
#elif [ "$TEST_TO_RUN" == "deploy" ]; then
#  ##############################
#  # Test single deploy, you must have previously compiled, and quantized
#  ##############################
#
#  python xyolo.py --xclbin $XCLBIN_PATH/$XCLBIN --netcfg yolo.cmds --datadir yolo.caffemodel_data --labels coco.names --xlnxlib
$LIBXDNN_PATH --quantizecfg yolo_deploy_${IMGWIDTH}.json --firstfpgalayer conv0 --in_shape $XDNN_SIZE --images $images --style yolo
#
else
  echo "Hello, goodbye!"
fi
```


“yolo.py” of yolo (1/2)

```
import os,sys

from xyolo import xyolo

# Bring in Xilinx Compiler, and Quantizer
# We directly compile the entire graph to minimize data movement between host, and card
#sys.path.insert(0,os.path.abspath("../../"))
from xfdnn.tools.compile.bin.xfdnn_compiler_caffe  import CaffeFrontend as xfdnnCompiler
from xfdnn.tools.quantize.quantize import CaffeFrontend as xfdnnQuantizer

# Select Configuration
from configs import select_config

config = select_config("608_16b");  #config = select_config("224_8b_tend")

mlsuiteRoot = os.getenv("MLSUITE_ROOT", "../..")

# Define the compiler, and its parameters
compiler = xfdnnCompiler(
  verbose=False,
  networkfile=config["network_file"], # Prototxt filename: input file
  generatefile=config["netcfg"],      # Script filename: output file
  strategy="all",                     # Strategy for memory allocation
  memory=config["memory"],                          # Available on chip ram within xclbin
  dsp=config["dsp"],                  # Rows in DSP systolic array within xclbin
  ddr=config["ddr"],                  # Memory to allocate in FPGA DDR for activation spill
  weights=config["weights"]                 # Floating Point weights, compiler will convert to framework agnostic directory structure
  )
```

“yolo.py” of yolo (2/2)

```
# Define the quantizer, and its parameters
quantizer = xfdnnQuantizer(
  deploy_model=config["network_file"],          # Prototxt filename: input file
  weights=config["weights"],                      # Floating Point weights
  output_json=config["quantizecfg"],
  calibration_directory=mlsuiteRoot+"/xfdnn/tools/quantize/calibration_directory", # Directory containing calbration images
  calibration_size=8,                           # Number of calibration images to use
  calibration_seed=None,                          # Seed for randomly choosing calibration images
  calibration_indices=None,                       # User can control which images to use for calibration [DEPRECATED]
  bitwidths=config["bitwidths"],                # Fixed Point precision: 8b or 16b
  dims=config["dims"],                          # Image dimensions [Nc,Nw,Nh]
  transpose=[2,0,1],                            # Transpose argument to caffe transformer
  channel_swap=[2,1,0],                         # Channel swap argument to caffe transfomer
  raw_scale=1,                                  # Raw scale argument to caffe transformer
  mean_value=[0,0,0],                             # Image mean per channel to caffe transformer
  input_scale=1                                 # Input scale argument to caffe transformer
  )
compiler.compile(); # Invoke compiler
quantizer.quantize(); # Invoke quantizer
imgDir = mlsuiteRoot+"/xfdnn/tools/quantize/calibration_directory"
images = sorted([os.path.join(imgDir,name) for name in os.listdir(imgDir)])
batch_sz = 4 # This determines how many images will be preprocessed and migrated to FPGA DDR at a time
nbatches = len(images) / batch_sz # Ignore the remainder for now (Don't operate on partial batch)
with xyolo(batch_sz=batch_sz,in_shape=config["dims"],quantizecfg=config["quantizecfg"],
xlnxlib=mlsuiteRoot+"/xfdnn/rt/xdnn_cpp/lib/libxfdnn.so", xclbin=config["xclbin"],netcfg=config["netcfg"],
datadir=config["datadir"],firstfpgalayer=config["firstfpgalayer"],classes=config["classes"],verbose=True) as detector:
  for i in range(nbatches):
    # Invoke detector
    detector.detect(images[i*batch_sz:(i+1)*batch_sz],display=True,coco=False)
    print("Finished batch %d" % (i+1))
  detector.stop()
```


Overall steps

- 1. AWS log in
- 2. Log in F1 instance
- 3. One time settings
- 4. AWS specific steps
  - ►set up SDAccel for AWS environment
  - ►get root permission in order to use FPGA hardware
- 5. Run YOLO
- Scripts
  - ►‘run.sh’ and ‘yolo.py’
- To test your own picture
  - ►‘run_yolo_one.sh’ and ‘yolo_one.py’

To test your own picture

- Prerequisites
  - *Do not forget to set up AWS FPGA environment*
  - *Do not forget to set up Anaconda2 ml-suit virtual environment*
  - *Do not forget to get root permission before run the application*
  - *Do prepare a directory for your pictures (at least two pictures when you set batch-size 2)*
    - *say 'in', in which 'dog.jpg' and 'person.jpg' resides.*
  - *Get new scripts (These are not a part of ML Suite. You can drag & drop the script from the slide, but some statements may be missing.)*
    - *'run_yolo_one.sh' and 'yolo_one.py'*
- Go to the Yolo directory
  - ►(ml-suite)[root]$ cd work.ml/ml-suite/apps/yolo
- Run
  - ►(ml-suite)[root]$ ./run_yolo_one.sh -dir <your dir> -batch_size 1

"run_yolo_one.sh" of yolo

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

DEVICE="aws"
TEST_TO_RUN="e2e"
PE_CFG=
BITWIDTH=16
IMGWIDTH=
IMAGE_DIR=
BATCH_SIZE=1

function func_help() {
  echo "Usage: $0 [options]"
  echo "       -device aws       ; default aws"
  echo "       -test_to_run e2e  ; default e2e"
  echo "       -pe_cfg 608_16b"
  echo "       -bit_width 8|16   ; default 16"
  echo "       -img_width ???    "
  echo "       -dir <image dir>"
  echo "       -batch_size 1     ; batch size"
  echo "       -h"
}

if [ -z $MLSUITE_ROOT ]; then
  MLSUITE_ROOT=../..
fi

while [ "`echo $1|cut -c1`" = "-" ]; do
  case $1 in
    -device) shift
          DEVICE=$1
          ;;
    -test_to_run) shift
          TEST_TO_RUN=$1
          ;;
    -pe_cfg) shift
          PE_CFG=$1
          ;;
    -bit_width) shift
          BITWIDTH=$1
          ;;
    -img_width) shift
          IMGWIDTH=$1
          ;;
    -dir) shift
          IMAGE_DIR=$1
          ;;
    -batch_size) shift
          BATCH_SIZE=$1
          ;;
    -h|-\?)
          func_help
          exit -1
          ;;
    *)
          echo "Unknown option: $1"
          func_help
          exit -1
          ;;
  esac
  shift
done

if [ ! -z "$1" ]; then
  echo un-known options: $1
  exit 1
fi

if [ -z ${IMAGE_DIR} ]; then
  echo "\"-dir image_dir\" should be given"
  exit 1
fi

XDNN_SIZE=[3,${IMGWIDTH},${IMGWIDTH}]

images=`ls ${MLSUITE_ROOT}/xfdnn/tools/quantize/calibration_directory/*`
echo "Running with images: $images"

XCLBIN=xdnn_v2_32x56_2pe_16b_6mb_bank21.xclbin
if [ "$BITWIDTH" == "8" ]; then
  XCLBIN=xdnn_v2_32x56_2pe_8b_6mb_bank21.xclbin
fi

# Set Enviornment Variables corresponding to HW platform
. ${MLSUITE_ROOT}/overlaybins/setup.sh $DEVICE

# Build Non-Max Suppression C-code
cd nms
make
cd ..

echo "=============== pyXDNN ============================="

if [ "$TEST_TO_RUN" == "e2e" ]; then
  GLOG_minloglevel=2 python yolo_one.py ${IMAGE_DIR} ${BATCH_SIZE}
else
  echo "Hello, goodbye!"
fi
```

“yolo_one.py” of yolo (1/2)

```
import os,sys
from xyolo import xyolo
from xfdnn.tools.compile.bin.xfdnn_compiler_caffe  import CaffeFrontend as xfdnnCompiler
from xfdnn.tools.quantize.quantize import CaffeFrontend as xfdnnQuantizer
from configs import select_config

config = select_config("608_16b")
mlsuiteRoot = os.getenv("MLSUITE_ROOT", "../..")

# Define the compiler, and its parameters
compiler = xfdnnCompiler(
  verbose=False,
  networkfile=config["network_file"], # Prototxt filename: input file
  generatefile=config["netcfg"],      # Script filename: output file
  strategy="all",                     # Strategy for memory allocation
  memory=config["memory"],                      # Available on chip ram within xclbin
  dsp=config["dsp"],                  # Rows in DSP systolic array within xclbin
  ddr=config["ddr"],                  # Memory to allocate in FPGA DDR for activation spill
  weights=config["weights"]           # Floating Point weights, compiler will convert to framework agnostic directory structure
  )

# Define the quantizer, and its parameters
quantizer = xfdnnQuantizer(
  deploy_model=config["network_file"],          # Prototxt filename: input file
  weights=config["weights"],                    # Floating Point weights
  output_json=config["quantizecfg"],
  calibration_directory=mlsuiteRoot+"/xfdnn/tools/quantize/calibration_directory", # Directory containing calbration images
  calibration_size=8,                 # Number of calibration images to use
  calibration_seed=None,              # Seed for randomly choosing calibration images
  calibration_indices=None,           # User can control which images to use for calibration [DEPRECATED]
  bitwidths=config["bitwidths"],      # Fixed Point precision: 8b or 16b
  dims=config["dims"],                # Image dimensions [Nc,Nw,Nh]
  transpose=[2,0,1],                  # Transpose argument to caffe transformer
  channel_swap=[2,1,0],               # Channel swap argument to caffe transfomer
  raw_scale=1,                        # Raw scale argument to caffe transformer
  mean_value=[0,0,0],                 # Image mean per channel to caffe transformer
  input_scale=1                       # Input scale argument to caffe transformer
  )
```


“yolo_one.py” of yolo (2/2)

```
# Invoke compiler
compiler.compile()

# Invoke quantizer
quantizer.quantize()

imgDir = sys.argv[1]
images = sorted([os.path.join(imgDir,name) for name in os.listdir(imgDir)])

batch_sz = int(sys.argv[2]) # This determines how many images will be preprocessed and migrated to FPGA DDR at a time

nbatches = len(images) / batch_sz # Ignore the remainder for now (Don't operate on partial batch)

# Define the xyolo instance
with xyolo(batch_sz=batch_sz,in_shape=config["dims"],quantizecfg=config["quantizecfg"], xlnxlib=mlsuiteRoot+"/xfdnn/rt/xdnn_cpp/lib/libxfdnn.so",
xclbin=config["xclbin"],netcfg=config["netcfg"], datadir=config["datadir"],firstfpgalayer=config["firstfpgalayer"],classes=config["classes"],verbose=True) as
detector:
  for i in range(nbatches):
    # Invoke detector
    detector.detect(images[i*batch_sz:(i+1)*batch_sz],display=True,coco=False)
    print("Finished batch %d" % (i+1))

  detector.stop()
```

Wish list

- How to deal with stream image.
- How to deal with Webcam.