

ZSK + FMC Camera, LCD Module AI Tutorial

2020. 11. 27

작성자: 김민석

(주) 리버트론





◆ Contents

1. 개발 환경	1
1.1. Hardware 구성	1
1.2. Software 개발 환경	3
2. AI Inference Application 개발 과정	4
2.1. Overview	4
2.2. CNN Model Training on PC	오류! 책갈피가 정의되어 있지 않습니다.
2.3. Quantization & Compile	오류! 책갈피가 정의되어 있지 않습니다.
2.4. Linux Application	오류! 책갈피가 정의되어 있지 않습니다.
2.5. Hybrid Compilation	오류! 책갈피가 정의되어 있지 않습니다.
2.6. Inference on Board	오류! 책갈피가 정의되어 있지 않습니다.
3. 예시	오류! 책갈피가 정의되어 있지 않습니다.
3.1. MNIST Classification using Resnet18	오류! 책갈피가 정의되어 있지 않습니다.
3.2. Face detection	오류! 책갈피가 정의되어 있지 않습니다.
4. 별첨	오류! 책갈피가 정의되어 있지 않습니다.
4.1. PC S/W 설치	오류! 책갈피가 정의되어 있지 않습니다.

1. 개발 환경

1.1. Hardware 구성

1) ZSK (Zynq Starter Kit)

Item	사진	비고
ZSK		동작 보드
TFT-LCD		동작 영상 출력
Camera		영상인식
Connector		카메라와 FMC Connector 보드 연결

2) FMC Camera & LCD Module

제품	Part	Spec	Note
ZSK	XC7Z020-FCLG484C		
Display	TFT-LCD	- 800 x 480 Pixel (7 인치) - RGB Parallel Interface	
	Touch-Panel	- 7 인치 CAP Touch (5-Point) - Controller IC 내장 (I2C)	
Camera	CMOS Image Sensor	- 1080P / 30FPS - Parallel Interface	
Connector	FMC	- Support VITA 57.1 (LPC x 1EA)	
	PMOD	- PMOD Connector x 2EA	
Power	Power Input	- FMC Power Input (+12V, +3.3V, ADJ)	
제품 size	LCD Module	- 205 x 125 x Adjustable (W x D x H)	
	Camera Module	- 60 x 60 x 17 (W x D x H)	

➤ ZSK + FMC Camera LCD Module 연결 모습



1.2. Software 개발 환경

1) PC

이름	정 보
OS	Ubuntu 18.04 64bit
S/W	Vitis-AI (V1.2.1)
설치 정보	정보 안내
Vitis-AI	https://github.com/Xilinx/Vitis-AI

※ 본 Tutorial은 Vitis AI에 동봉된 Caffe-Xilinx를 기반으로 실습되었다. 따라서 본 자료를 기반으로 실습을 할 사용자들은 상기의 자료에 나와 있는 사이트의 내용 등을 통해서 개발 환경을 (Vitis AI V1.2.1)를 미리 설치하기를 권장한다.

2) Board

- OS: PetaLinux 2020.1
- OpenCV API
- Vitis AI DNNDK Library ????

2. AI Inference Application 개발 과정

2.1. Overview

1) Block Diagram

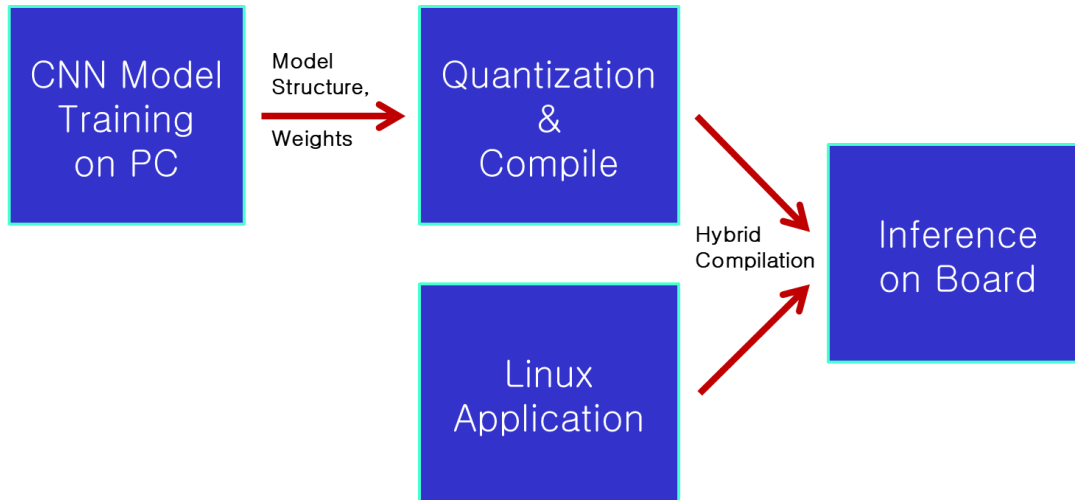


그림 1. Overview Block Diagram

2) Description

- CNN Model Training on PC
 - ✓ Caffe 또는 Tensorflow AI 프레임워크를 사용하여 CNN 모델을 학습시킨다.
 - ✓ 학습의 결과물인 CNN 모델의 구조와 가중치 값을 도출한다.
- Quantization & Compile
 - ✓ PC 에서 학습한 CNN 모델이 FPGA 에서 동작하게 하기 위해서는 실수형 가중치 데이터를 정수형으로 변환해야 하며, 이러한 과정을 Quantization 이라 한다.
 - ✓ Xilinx Vitis AI 는 모델의 정확도 감소를 최소화하면서 Quantization 하는 기능을 제공한다.
 - ✓ Xilinx Vitis AI 는 Quantization 이 완료된 모델을 Linux Application 에서 로드할 수 있도록 컴파일 하는 기능을 제공한다.
- Linux Application
 - ✓ 일반적인 AI application 은 아래와 같이 구성된다.
 - ✓ 입력 데이터 로드 -> AI 모델에 입력 -> Inference 결과 출력
 - ✓ 입력 데이터는 카메라, 각종 센서, 이미지 파일 등이 될 수 있다.

- ✓ Inference 결과를 출력하는 방법 역시 LCD 출력, 콘솔 로그 출력 등 다양한 방식으로 가능하다.

2.2. Classification with ResNet18

1) Abstract

ResNet 은 2015 년 ILSVRC 대회(Imagenet 이미지 인식 대회)에서 우승한 CNN 모델이다. "Deep Residual Learning for Image Recognition - Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun" 논문에서 발표되었으며, Layer 의 깊이에 따라 ResNet18, ResNet50, ResNet152 등으로 나뉘어진다. 그 중 18 Layer 로 구성된 ResNet18 을 이용하여 손글씨를 추론하는 예시를 소개한다.

2) Training on PC

■ 학습 준비하기

ResNet18 은 Xilinx Model Zoo 에서 배포되며, 학습은 Vitis AI 에 포함된 Caffe-Xilinx 를 통해 진행된다. 리버트론은 mnist_resnet18.tar.gz 파일을 실습을 위해 배포하고 있다. 해당 압축파일을 Vitis AI 디렉토리에 압축 해제하여 실습을 준비한다. 리버트론에서 제공하는 Solver 및 Train/Test Model 은 아래와 같다.

- Train Model : mnist_resnet18/float/trainval.prototxt
- Test Model : mnist_resnet18/float/test.prototxt
- Solver : mnist_resnet18/float/solver.prototxt

ResNet18 이 Image Classification 을 수행하도록 학습하기 위한 Training Dataset 은 MNIST 를 사용한다. 기존의 MNIST 데이터셋은 28x28 픽셀 이미지로 구성되어 있지만 ResNet18 의 경우, 224x224 크기의 이미지를 입력 받기 때문에 데이터셋을 그대로 사용할 수 없다. 따라서 ResNet18 이 학습할 수 있도록 Resize 및 Noise Filtering 등을 적용하여 데이터셋을 가공하였다. MNIST 데이터셋은 제공된 mnist_resnet18.tar.gz 파일의 dataset 폴더에 있다.

- Images Dataset: mnist_resnet18/dataset

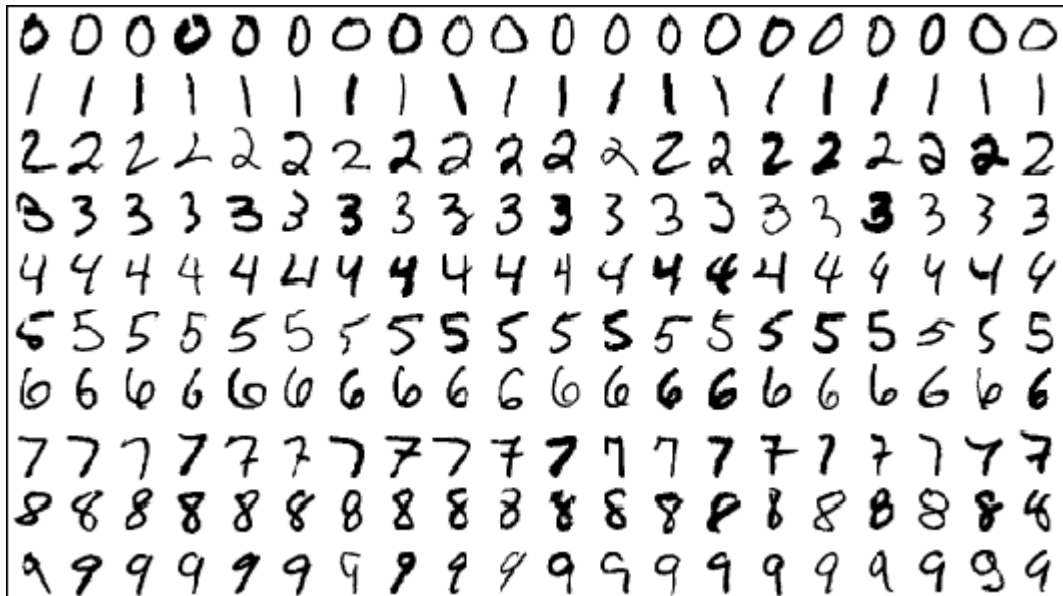


그림 2. MNIST 데이터셋

기본적으로 Neural Network 모델을 CPU 만을 사용하여 학습하는 것은 너무나도 오랜 시간이 소요된다. 따라서 Vitis AI 는 될 수 있으면 GPU Docker 를 사용하도록 한다. 만약 GPU 가 갖춰지지 않은 환경이라면 CPU Docker 를 사용하여 모델을 학습하거나 과감하게 Training Section 을 건너뛰고 다음 장으로 이동하자. Training Section 을 건너뛰는 경우, 제공된 Pre-trained Model 과 함께 Quantize & Compile Section 으로 이동한다.

- Pre-trained train Model: mnist_resnet18/pre_trained/train.prototxt
- Pre-trained test Model: mnist_resnet18/pre_trained/test.prototxt
- Pre-trained Weights: mnist_resnet18/pre_trained/

■ Vitis AI 시작하기

Vitis AI 는 Docker 환경에서 실행된다. Vitis AI Docker 는 CPU Docker, GPU Docker 로 구분된다. 각각의 Docker 실행은 아래와 같다.

- CPU Docker Run


```
lbt@sw:~/vitis_ai$ ./docker_run.sh xilinx/vitis-ai-cpu
```

- GPU Docker Run

```
lbt@sw:~/vitis_ai$ ./docker_run.sh xilinx/vitis-ai-gpu
```

Docker 가 실행되면 다음과 같은 화면이 출력된다.

```
lbt@sw$ ./docker_run.sh xilinx/vitis-ai-gpu
:
Press any key to continue...

Do you agree to the terms and wish to proceed [y/n]? y
Running as lbt
passwd: password expiry information changed.
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

=====
VITIS-AI
=====

Docker Image Version: latest
Build Date: Sun Nov 29 19:39:21 MST 2020
VAI_ROOT=/opt/vitis_ai
For TensorFlow Workflows do:
  conda activate vitis-ai-tensorflow
For Caffe Workflows do:
  conda activate vitis-ai-caffe
For Neptune Workflows do:
  conda activate vitis-ai-neptune
For pytorch Workflows do:
  conda activate vitis-ai-pytorch
For optimizer_darknet Workflows do:
  conda activate vitis-ai-optimizer_darknet
For optimizer_caffe Workflows do:
  conda activate vitis-ai-optimizer_caffe
For optimizer_tensorflow Workflows do:
  conda activate vitis-ai-optimizer_tensorflow
lbt@sw:/workspace$
```

Vitis AI Docker 는 Caffe, TensorFlow, PyTorch 등의 Framework 이 지원되며(Vitis AI v1.2.1 기준), 각각의 Framework 환경은 Conda 를 통해 활성화될 수 있다. 본 실습에서는 Caffe Framework 을 사용한다. 다음 커맨드를 입력하여 Caffe Conda 환경을 활성화한다.

```
lbt@sw:/workspace$ conda activate vitis-ai-caffe
(vitis-ai-caffe)lbt@sw:/workspace$
```

Conda 가상환경이 활성화되면 사용자 계정 앞에 현재 Conda 환경이 표시된다. 위 화면에서는 Vitis AI Caffe Conda 가 활성화 된 것을 확인할 수 있다.

■ Caffe-Xilinx 로 학습하기

신경망을 학습하기 위해서 먼저 mnist_resnet18 디렉토리로 이동한다.

```
(vitis-ai-caffe)lbt@sw:/workspace$ cd mnist_resnet18
(vitis-ai-caffe)lbt@sw:/workspace/mnist_resnet18$
```

신경망 학습을 위한 파라미터는 Solver 에 정의되어있다. 제공된 Solver 의 내용은 다음과 같다.

```
net: "float/trainval.prototxt"
test_iter: 100
test_interval: 10
test_initialization: true
display: 10
average_loss: 1000
base_lr: 0.01
lr_policy: "step"
stepsize: 400
gamma: 0.1
max_iter: 500
momentum: 0.9
weight_decay: 0.0001
snapshot: 100
snapshot_prefix: "float/snapshots/mnist_resnet18"
solver_mode: GPU
```

대부분의 경우에 제공된 Solver 로 학습이 원활하게 진행되겠지만, 사용자의 요구사항에 맞춰 Solver 를 수정할 수 있다. 예를 들어, 학습 Iteration 수를 늘리고 싶다면 "max_iter" 속성의 값을 더 큰 값으로 설정한다. 또한, 실습에 사용될 GPU 의 메모리 공간이 부족하여 Memory Allocation Error 가 발생하는 경우 trainval.prototxt 파일의 Batch Size 를 조절하여 GPU Memory Allocation Error 를 예방한다.

학습을 위한 준비가 완료되었다면 Caffe Framework 을 사용하여 ResNet18 신경망을 학습한다.

```
(vitis-ai-caffe)lbt@sw:/workspace/mnist_resnet18$ caffe train -solver float/
solver.prototxt -gpu 0 2>&1 | tee train.log
```

제공되는 Solver 를 그대로 사용하였다면 학습은 500 Iteration 이 진행된다. 학습의 진행과정은 로그를 통해 나타나며 다음과 같이 확인할 수 있다.

```

:
I1201 19:51:44.850847 330 solver.cpp:523] Test net output #1: top_5 = 1
I1201 19:51:44.904991 330 solver.cpp:270] Iteration 490 (7.06514 iter/s,
1.4154s/10 iter), loss = 0.27486, remaining 0 hours and 0 minutes
I1201 19:51:44.905017 330 solver.cpp:291] Train net output #0: loss =
0.115353 (* 1 = 0.115353 loss)
I1201 19:51:44.905756 330 sgd_solver.cpp:106] Iteration 490, lr = 0.001
I1201 19:51:45.423990 330 solver.cpp:935] Snapshotting to binary proto
file float/snapshots/mnist_resnet18_iter_500.caffemodel
I1201 19:51:45.696494 330 sgd_solver.cpp:273] Snapshotting solver state to
binary proto file float/snapshots/mnist_resnet18_iter_500.solverstate
I1201 19:51:45.746800 330 solver.cpp:384] Iteration 500, loss = 0.270056
I1201 19:51:45.756327 330 solver.cpp:424] Iteration 500, Testing net (#0)
I1201 19:51:45.756445 330 net.cpp:754] Ignoring source layer loss
I1201 19:51:45.756460 381 net.cpp:754] Ignoring source layer loss
I1201 19:51:45.756471 380 net.cpp:754] Ignoring source layer loss
I1201 19:51:46.582989 330 solver.cpp:523] Test net output #0: top_1 =
0.985
I1201 19:51:46.583019 330 solver.cpp:523] Test net output #1: top_5 = 1
I1201 19:51:46.583029 330 solver.cpp:392] Optimization Done (5.46359
iter/s).
I1201 19:51:46.798827 330 caffe.cpp:250] Optimization Done.
(vitis-ai-caffe) lbt@sw:/workspace/mnist_resnet18$

```

신경망의 가중치는 mnist_resnet18/float/snapshots 에 100 Iteration 마다 저장된다. 저장된 가중치 파일은 신경망을 Quantize 하고 Compile 하는데 사용된다.

■ Trained Model Test

신경망 학습이 완료되면 리버트론에서 제공하는 테스트 프로그램을 사용하여 신경망의 정확도를 확인해볼 수 있다. 테스트는 지정된 경로상의 이미지를 추론하여 Top-5 결과를 출력해준다. 테스트 프로그램은 Python 으로 작성되었으며 사용법은 아래와 같다.

```
(vitis-ai-caffe)lbt@sw:/workspace/mnist_resnet18$ python test/classification
.py --src test/images/ --model float/test.prototxt --weights float/snapshots
/mnist_resnet18_iter_500.caffemodel --label test/words.txt
```

각 매개변수의 의미는 다음과 같다.

- src: 테스트 이미지들이 있는 디렉토리
- model: 테스트 신경망 모델 파일
- weights: 테스트 신경망 가중치 파일
- label: 데이터셋 라벨 파일

테스트 결과는 아래와 같이 텍스트로 제공된다.

```

:
Top[3] 0.0001    5
Top[4] 0.0001    2
-----
Top-5 of images/2.jpg
      Score  Label
Top[0] 0.9997    2
Top[1] 0.0002    0
Top[2] 0.0001    7
Top[3] 0.0000    3
```

```

Top[4] 0.0000      4
-----
Top-5 of images/3.jpg

      Score  Label
Top[0] 0.9985    3
Top[1] 0.0007    2
Top[2] 0.0007    5
Top[3] 0.0000    1
Top[4] 0.0000    8
-----
(vitis-ai-caffe) lbt@sw:/workspace/mnist_resnet18$

```

만약 weights 매개변수에 학습이 덜 진행된 100 Iteration 의 가중치 파일을 입력하면 신경망이 아래와 같이 부정확한 결과를 내보이는 것을 확인할 수 있다.

```

:
Top[3] 0.0036      6
Top[4] 0.0027      2
-----
Top-5 of images/2.jpg

      Score  Label
Top[0] 0.9960     2
Top[1] 0.0026     0
Top[2] 0.0011     6
Top[3] 0.0002     4
Top[4] 0.0001     3
-----
Top-5 of images/3.jpg

      Score  Label
Top[0] 0.6186     2
Top[1] 0.3669     3
Top[2] 0.0055     7
Top[3] 0.0046     5
Top[4] 0.0030     0
-----
(vitis-ai-caffe) lbt@sw:/workspace/mnist_resnet18$

```

3) Quantization & Compile

■ Quantization

일단 신경망의 학습이 완료되면 Quantization 및 Compile 은 다른 Network 와 동일한 일반적인 과정을 거친다. 다만 Quantization 을 시작하기 전에 Quantization 을 위한 약간의 준비과정이 필요하다.

- ① Trained Model Test 에 사용되었던 test.prototxt 파일 (경로: mnist_resnet18/float/test.prototxt)의 "Input" 레이어를(레이어의 첫 번째 내용) "ImageData" 레이어로 변경하고 Image Data Source 를 명시해준다.

Example)

[Before]	[After]
<pre> layer { name: "data" type: "Input" top: "data" input_param { shape { dim: 1 dim: 3 dim: 224 dim: 224 } } } layer { name: "conv1" type: "Convolution" bottom: "data" : </pre>	<pre> layer { name: "data" type: "ImageData" top: "data" top: "label" include { phase: TRAIN } transform_param { mirror: false crop_size:224 } image_data_param { source: "dataset/calibration.txt" batch_size: 16 } } layer { name: "conv1" type: "Convolution" bottom: "data" : </pre>

- ② Image Path 와 Label Information 을 포함하고 있는 calibration.txt 파일을 준비한다. calibration.txt 는 mnist_resnet18/dataset/calibration.txt 에 위치한다.

```
images/000001.jpg 1
images/000002.jpg 1
images/000003.jpg 1
:
```

calibration.txt 에 명시된 Label Information 은 Fake Value 이다. 즉, Calibration 에 사용되지는 않지만 명시되어야 하는 0 혹은 1 과 같은 임의의 값을 의미한다. calibration.txt 파일에 명시된 이미지 파일은 Calibration 에 사용되는데, Calibration 이미지는 학습에 사용되었던 이미지를 사용하는 것이 좋다. Calibration 이미지 수에는 제약이 없으나 해당 실습에서는 100 장을 사용한다.

Calibration 이미지가 준비되었다면 모델을 Quantization 해준다. Quantization 은 Vitis AI 를 사용하여 진행한다. Quantization 방법은 아래와 같다.

```
(vitis-ai-caffe)lbt@sw:/workspace/mnist_resnet18$ vai_q_caffe quantize -mode
1 float/test.prototxt -weights float/snapshots/mnist_resnet18_iter_500.caffe
model -method 1 -output_dir quantize/ -calib_iter 10
```

- model: Quantization 될 신경망 모델 파일
- weights: Quantization 될 신경망 가중치 파일
- output_dir: Quantization 된 모델 파일 및 가중치 파일이 저장될 경로
- method: Quantization 방법 (0: non-overflow, 1: min-diffs)
- calib_iter: Calibration 반복 횟수

Calibration 이 진행되면 다음과 같은 로그가 출력됨을 확인할 수 있다.

```
:
```

```
I1202 01:31:43.225636 2332 net.cpp:284] Network initialization done.
I1202 01:31:43.240381 2332 vai_q.cpp:182] Start Calibration
I1202 01:31:43.667488 2332 vai_q.cpp:206] Calibration iter: 1/10 ,loss: 0
I1202 01:31:43.810667 2332 vai_q.cpp:206] Calibration iter: 2/10 ,loss: 0
I1202 01:31:43.965862 2332 vai_q.cpp:206] Calibration iter: 3/10 ,loss: 0
```

```

I1202 01:31:44.118415 2332 vai_q.cpp:206] Calibration iter: 4/10 ,loss: 0
I1202 01:31:44.235973 2332 vai_q.cpp:206] Calibration iter: 5/10 ,loss: 0
:
I1202 01:31:45.501041 2332 vai_q.cpp:360] Start Deploy
W1202 01:31:45.807193 2332 convert_proto.cpp:1355] [DEPLOY WARNING] Layer
data's output blob is all zero, this may cause error for DNNC compiler.
Please check the float model.
I1202 01:31:45.818264 2332 vai_q.cpp:368] Deploy Done!
-----
Output Quantized Train&Test Model:  "quantize/quantize_train_test.prototxt"
Output Quantized Train&Test Weights:
"quantize/quantize_train_test.caffemodel"
Output Deploy Weights: "quantize/deploy.caffemodel"
Output Deploy Model:  "quantize/deploy.prototxt"
(vitis-ai-caffe) lbt@sw:/workspace/mnist_resnet18$

```

Quantization 이 완료되면 "quantize" 디렉토리에 "deploy.prototxt",
 "deploy.caffemodel", "quantize_train_test.prototxt", "quantize_train_test.caffemodel"
 파일이 생성된다.

- deploy.prototxt: Compile 될 신경망 모델 파일
- deploy.caffemodel: Compile 될 신경망 가중치 파일
- quantize_train_test.prototxt: 정확도 측정 테스트 및 Quantize
Finetuning 에 사용되는 신경망 모델 파일
- quantize_train_test.caffemodel: 정확도 측정 및 테스트 및 Quantize
Finetuning 에 사용되는 신경망 가중치 파일

■ Compile

Vitis AI Compiler 는 Quantization 으로 생성된 모델 및 가중치 파일이 입력으로
 사용된다. Compile 을 진행하기 전, Quantization 의 결과인 deploy.prototxt 파일이
 하나의 Input Layer 를 갖도록 아래와 같이 수정한다.

Example)

[Before]	[After]
<pre> layer { name: "data" type: "Input" </pre>	<pre> layer { name: "data" type: "Input" </pre>

<pre> top: "data" transform_param { } input_param { shape { dim: 1 dim: 3 dim: 224 dim: 224 } } } layer { name: "data" type: "Input" top: "data" phase: TRAIN input_param { shape { dim: 1 dim: 3 dim: 224 dim: 224 } } } layer { name: "conv1" type: "Convolution" bottom: "data" : </pre>	<pre> top: "data" transform_param { } input_param { shape { dim: 1 dim: 3 dim: 224 dim: 224 } } } layer { name: "conv1" type: "Convolution" bottom: "data" : </pre>
---	---

deploy.prototxt 파일의 수정이 완료되면 다음과 같은 방법으로 Vitis AI Compiler 를 사용한다.

```
(vitis-ai-caffe)lbt@sw:/workspace/mnist_resnet18$ vai_c_caffe --prototxt quantize/deploy.prototxt --caffemodel quantize/deploy.caffemodel --output_dir compile/ --net_name mnist_resnet18 --arch common/arch.json
```

- prototxt: Compile 될 INT8 신경망 모델 파일
- caffemodel: Compile 될 INT8 신경망 가중치 파일
- output_dir: Compile 된 .elf 파일이 저장될 경로
- arch: DPU Target, DPU Configuration 파일, CPU Architecture 등의 설정 값이 정의된 아키텍처 파일
- net_name: Compile 후 사용될 신경망에 대한 DPU 커널의 이름

Compile 을 통해 생성된 .elf 파일은 Linux Application 에서 Hybrid Compile 을 통해 어플리케이션과 융화된다. elf 파일의 이름은 사용자가 옵션으로 지정한 net_name 이 적용된다. net_name 은 Linux Application 에서 DPU 커널을 불러올 때도 사용된다.

Vitis AI Compile 이 완료되면 다음과 같은 Model Summary 가 출력된다.

```
*****
* VITIS_AI Compilation - Xilinx Inc.
*****
[VAI_C][Warning] layer [prob] (type: Softmax) is not supported in DPU, deploy it in CPU instead.

Kernel topology "mnist_resnet18_kernel_graph.jpg" for network "mnist_resnet18"
kernel list info for network "mnist_resnet18"
      Kernel ID : Name
              0 : mnist_resnet18_0
              1 : mnist_resnet18_1

      Kernel Name : mnist_resnet18_0
-----
--
      Kernel Type : DPUKernel
      Code Size : 0.16MB
      Param Size : 10.66MB
      Workload MACs : 3652.82MOPS
      IO Memory Space : 0.91MB
      Mean Value : 0, 0, 0,
      Total Tensor Count : 24
      Boundary Input Tensor(s) (H*W*C)
      data:0(0) : 224*224*3

      Boundary Output Tensor(s) (H*W*C)
      fc1000:0(0) : 1*1*10

      Total Node Count : 23
      Input Node(s) (H*W*C)
```

```

conv1(0) : 224*224*3
Output Node(s) (H*W*C)
fc1000(0) : 1*1*10

Kernel Name : mnist_resnet18_1
-----
Kernel Type : CPUKernel
Boundary Input Tensor(s) (H*W*C)
prob:0(0) : 1*1*10

Boundary Output Tensor(s) (H*W*C)
prob:0(0) : 1*1*10

Input Node(s) (H*W*C)
prob : 1*1*10

Output Node(s) (H*W*C)
prob : 1*1*10

(vitis-ai-caffe) lbt@sw:/workspace/mnist_resnet18$

```

위 로그에서 mnist_resnet18 신경망이 mnist_resnet18_0 과 mnist_resnet18_1 로 나뉘어진 것을 볼 수 있다. 이는 기존 ResNet18 신경망에 들어있는 Softmax Layer 가 현재 ZSK DPU 에 포함되어있지 않아 Softmax Layer 만 따로 CPU Kernel 로 분리된 것이다.

4) Linux Application

ResNet18 을 사용한 손 글씨 추론 어플리케이션은 카메라의 Captured Image 를 전달받아 이를 DPU 에 Input Image 로 사용한다. DPU 의 결과는 Softmax 를 통해 해당 이미지가 각 Class 일 확률의 백분율로 변환되고 Top-1 의 결과를 화면에 출력한다. Application Project 의 경로는 mnist_resnet18/project 이다.

■ Block Diagram

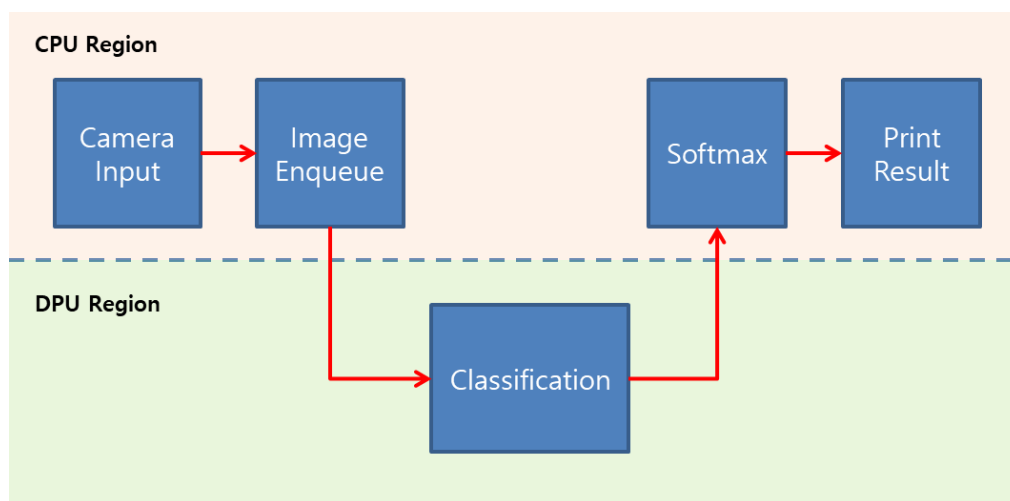


그림 3. Classification Block Diagram

■ Camera Image Enqueue

Camera 가 이미지를 캡처하면 Callback Function 에 의해 800x480 이미지를 Image Queue 에 Enqueue 한다. putQueue 함수는 inference.cpp 파일에 정의되어있다.

```
static void putQueue(VIDEO_DATA *pVD)
{
    FrameInfo fgInfo;
    fgInfo.buf = (unsigned char*)pVD->data;
    fgQueue.push(fgInfo);
    pthread_cond_signal(&fgQueueCondEmpty);
}

extern "C" void videoCaptureCallback(VIDEO_DATA *pdata)
{
```

```
putQueue(pdata);
}
```

■ Image Queue Pop

Camera 가 이미지를 Image Queue 에 Push 하면 이를 네트워크 모델의 입력 데이터로 사용하기 위해 Image Queue Pop 을 실행한다.

```
void* capture(void* arg)
{
    FrameInfo captureFrameInfo;

    PINFERENCE pInference = (PINFERENCE) arg;

    saveCount = 0;
    while(pInference->captureLoop){
        pthread_mutex_lock(&fgQueueMutex);
        while (fgQueue.empty()) {
            pthread_cond_wait(&fgQueueCondEmpty, &fgQueueMutex);
        }

        captureFrameInfo = fgQueue.front();
        fgQueue.pop();

        findRoI(captureFrameInfo.buf, FB_X_RES, FB_Y_RES);

        pthread_cond_signal(&fgQueueCondFull);
        pthread_mutex_unlock(&fgQueueMutex);
    }
    return NULL;
}
```

■ Classification & Softmax

Camera Image 를 Image Queue 에서 Pop 했다면 이를 ResNet18 에 넣어 결과를 확인할 수 있다. ResNet18 에 이미지를 넣어 결과를 확인하는 함수는 runResnet18()이다. 해당 함수는 inference.cpp 에 정의되어있다.

```
void runResnet18(Mat &img) {
```

```

assert(taskResnet18);

/* Mean value for Lenet specified in Caffe prototxt */
vector<string> kinds, images;

/* Load all kinds words.*/
LoadWords(BASE_WORDS_PATH, kinds);
if (kinds.size() == 0) {
    cerr << "\nError: No words exist in file words.txt." << endl;
    return;
}

/* Get channel count of the output Tensor for Lenet Task */
int channel = dpuGetOutputTensorChannel(taskResnet18, OUTPUT_NODE);

float *softmax = new float[channel];
float *FCResult = new float[channel];

dpuSetInputImage2(taskResnet18, INPUT_NODE, img, 0);

/* Launch RetNet18 Task */
dpuRunTask(taskResnet18);

/* Get FC result and convert from INT8 to FP32 format */
dpuGetOutputTensorInHWCFP32(taskResnet18, OUTPUT_NODE, FCResult, channel);

/* Calculate softmax on CPU and display TOP5 classification results */
CPUCalcSoftmax(FCResult, channel, softmax);
TopK(softmax, channel, 1, kinds);

delete[] softmax;
delete[] FCResult;
}

```

runResnet18() 함수에서는 DPU 에 Input Image 를 설정해주고 이미지가 Neural Network 을 통과하도록 한다. 네트워크가 Input Image 에 대한 Output 을 출력하면 dpuGetOutputTensorInHWCFP32() API 를 통해 출력 데이터의 포맷을 변경해준다. 해당 API 는 DPU Task 의 Output 을 INT8 에서 FP32 로 변환하고 HWC(Height * Width * Channel)의 레이아웃에 따라 CPU 메모리 버퍼에 저장하도록 설정한다. 데이터 포맷이 변경된 후에는 CPUCalcSoftmax() 함수를 사용하여 Softmax 를 수행한다.

■ Top-k & Draw Result

Softmax 를 통해 정규화된 DPU 의 Top-1 추론 결과를 LCD 에 표시해준다. Draw Result 동작은 draw_font() 함수에 정의되어있다.

```
void TopK(const float *d, int size, int k, vector<string> &vkinds) {
    assert(d && size > 0 && k > 0);
    priority_queue<pair<float, int>> q;

    for (auto i = 0; i < size; ++i) {
        q.push(pair<float, int>(d[i], i));
    }

    for (auto i = 0; i < k; ++i) {
        pair<float, int> ki = q.top();
        char* top1 = (char*)(vkinds[ki.second].c_str());
        lcdClear(1120, 480, 100, 100);
        draw_font(pDetectFont, top1, 1, 1120, 480);
        q.pop();
    }
}
```

5) Hybrid Compilation

보드에서 추론 어플리케이션을 실행하기 위해선 신경망에 대한 DPU Instruction 인 .elf 파일과 어플리케이션 소스코드를 Hybrid Compilation 해야 한다. Hybrid Compilation 을 위한 단계는 아래와 같다.

■ Open New Terminal

ZSK 보드의 PS 에는 ARM 프로세서가 탑재되어 있다. 이는 보드에서 추론 어플리케이션이 동작하기 위해서는 x86 기반의 Host PC 에서 ARM 프로세서를 타겟으로 Cross Compile 이 진행되어야 함을 의미한다. 아쉽게도 현재 Vitis AI v1.2.1 Docker 에는 ARM 프로세서를 타겟으로 하는 GCC Compiler 가 들어있지 않다. 물론 Docker 이미지에 ARM GCC Compiler 를 설치해도 되지만 본 Tutorial 에서는 새로운 터미널을 열어 Vitis AI Docker 환경이 아닌 Local 환경에서 Compile 을 진행한다.

새로운 터미널을 열어 mnist_resnet18 이 위치한 경로로 이동한다.

■ PetaLinux SDK 환경설정

PetaLinux SDK 설치 스크립트는 mnist_resnet18/sdk 에 sdk.sh 파일로 제공된다. PetaLinux SDK 를 사용하기 위해 먼저 SDK 를 설치한다.

```
lbt@sw:~/vitis_ai/mnist_resnet18$ sdk/sdk.sh
```

스크립트를 실행하면 PetaLinux SDK Installer 가 실행되며 SDK 설치 위치를 묻는다. 기본값은 /opt/petalinux/2020.1 이지만 본 Tutorial 에서는 mnist_resnet18/sdk 에 설치를 진행하도록 한다.

```
PetaLinux SDK installer version 2020.1
=====
Enter target directory for SDK (default: /opt/petalinux/2020.1): ./sdk/
You are about to install the SDK to "/home/lbt/vitis_ai/mnist_resnet18/sdk".
Proceed [Y/n]? Y
```



```

Extracting
SDK.....done
Setting it up...done
Your environment is misconfigured, you probably need to 'unset LD_LIBRARY_PATH'
but please check why this was set in the first place and that it's safe to unset.
The SDK will not operate correctly in most cases when LD_LIBRARY_PATH is set
.
For more references see:
  http://tldp.org/HOWTO/Program-Library-HOWTO/shared-libraries.html#AEN80
  http://xahlee.info/UnixResource_dir/_/ldpath.html
/opt/vitis_ai_v1.2.1/mnist_resnet18/sdk/post-relocate-setup.sh: Failed to source /opt/vitis_ai_v1.2.1/mnist_resnet18/sdk/environment-setup-cortexa9t2hf-neon-xilinx-linux-gnueabi with status 1
SDK has been successfully set up and is ready to be used.
Each time you wish to use the SDK in a new shell session, you need to source the environment setup script e.g.
$ . /opt/vitis_ai_v1.2.1/mnist_resnet18/sdk/environment-setup-cortexa9t2hf-neon-xilinx-linux-gnueabi
lbt@sw:~/vitis_ai/mnist_resnet18$

```

설치가 완료되면 mnist_resnet18/sdk 폴더에 sysroots 디렉토리를 포함한 SDK 파일들이 설치된 것을 확인할 수 있다. 설치된 SDK 를 Sourcing 하여 Hybrid Compilation 환경을 설정한다.

```

lbt@sw:~/vitis_ai/mnist_resnet18$ unset LD_LIBRARY_PATH
lbt@sw:~/vitis_ai/mnist_resnet18$ source sdk/environment-setup-cortexa9t2hf-neon-xilinx-linux-gnueabi
lbt@sw:~/vitis_ai/mnist_resnet18$

```

■ ELF 파일 준비

Hybrid Compilation 을 위해 Vitis AI Compiler 를 통해 생성된 dpu_mnist_resnet18_0.elf 파일을 Application Project 의 lib/inference/model 디렉토리와 main 디렉토리에 복사한다.

```

lbt@sw:~/vitis_ai/mnist_resnet18$ cp compile/dpu_mnist_resnet18_0.elf project/libs/inference/model/
lbt@sw:~/vitis_ai/mnist_resnet18$ cp compile/dpu_mnist_resnet18_0.elf project/main/
lbt@sw:~/vitis_ai/mnist_resnet18$

```

■ Hybrid Compilation

Application Project 디렉토리로 이동하여 Linux Application 과 DPU Instruction ELF 를 함께 Compile 한다.

```
lbt@sw:~/vitis_ai/mnist_resnet18$ cd project/  
lbt@sw:~/vitis_ai/mnist_resnet18/project$ ./debug.sh
```

Hybrid Compilation 이 완료되면 다음과 같은 빌드 로그를 확인할 수 있다.

```
:  
make[2]: Entering directory '/home/lbt/vitis_ai/mnist_resnet18/project/libs/  
inference'  
[Debug Compiling : arm-xilinx-linux-gnueabi-gcc]  
Installing.....[../target/libs/libinference.so.1.0.0]  
make[2]: Leaving directory '/home/lbt/vitis_ai/mnist_resnet18/project/libs/i  
nference'  
make[1]: Leaving directory '/home/lbt/vitis_ai/mnist_resnet18/project/libs'  
  
cd main && make -f Makefile install  
make[1]: Entering directory '/home/lbt/vitis_ai/mnist_resnet18/project/main'  
[Debug Compiling => arm-linux-gnueabihf-gcc]  
Installing.....[../target/fmcTest]  
make[1]: Leaving directory '/home/lbt/vitis_ai/mnist_resnet18/project/main'  
lbt@sw:~/vitis_ai/mnist_resnet18/project$
```

만약 Compile 중 "arm-linux-gnueabihf-gcc: Command not found" 에러가 발생한다면 다음과 같이 ARM Cross Compiler 를 설치해준다.

```
lbt@sw:~/vitis_ai/mnist_resnet18/project$ sudo apt-get install gcc-arm*
```

■ Run Application

Hybrid Compilation 으로 생성된 파일은 target 디렉토리에 저장된다. target 디렉토리의 파일들과 제공된 Linux Boot 파일 및 Font, Words 파일들을 SD 카드에 복사해준다. Linux Boot 파일은 mnist_resnet18/boot 에 있으며 Font 와 Words 파일은 mnist_resnet18/common 에 제공된다.

복사가 완료된 SD 카드의 파일들은 다음과 같다.

이름	수정한 날짜	유형	크기
libs	2020-12-03 오전 11:49	파일 폴더	
BOOT.BIN	2020-11-26 오후 4:05	BIN 파일	4,794KB
boot.scr	2020-11-26 오전 10:31	화면 보호기	2KB
fmcTest	2020-12-03 오전 11:23	파일	11,110KB
image.ub	2020-11-26 오후 3:55	UB 파일	71,110KB
NanumSquareRoundR.ttf	2020-06-03 오전 2:28	트루타입 글꼴 파일	1,039KB
start.sh	2020-11-26 오전 6:54	Shell Script	1KB
words.txt	2020-06-01 오전 6:14	텍스트 문서	1KB

그림 4. Copy to SD Card

SD 카드에 파일을 모두 저장하였다면 보드와 연결된 터미널 화면에서 어플리케이션을 실행한다. 보드와 PC 간의 Serial 통신에는 UART 가 사용된다.

```
root@fmc:~/$ cd /media/card/
root@fmc:/media/card$ ./start.sh
```

카메라의 입력 영상이 TFT-LCD 화면에 나타나고, 특정 ROI 영역의 이미지에 대한 DPU 의 손 글씨 추론 결과가 출력된다.

Revision History

Ver	Date	Revision
1.0	2020-11-30	1 st Initial Version.