TI MCU, DSP 및 Xilinx FPGA 프로그래밍 전문가 과정

Innova Lee(이상훈)
gcccompil3r@gmail.com

TI Deep Learning(TIDL)

3.14.1. Introduction:

3.14.1.1. Deep Learning Inference in Embedded Device

TIDL 은 애플리케이션이 EVE 및 C66x DSP 연산 엔진에서 TI 의 독점적이고
고도로 최적화된 CNN/DNN 구현을 활용할 수 있게함으로써 Deep Learning 을 가능하게 한다.
TIDL 은 처음에 AM57x SoC 에 대한 2D(일반적으로 비전) 데이터 사용 사례를 대상으로 한다.
1D 또는 3D 입력 데이터 Blob(과학적 정보 등등)에 대한 TIDL 의 사용을 막는 근본적인 제한은 없다.

TIDL 은 ARM 코어에서 EVE 및 / 또는 C66x DSP 와 같은 HW 가속기에 이르기까지
Deep Learning(추론 전용) 작업 부하를 계산할 수 있는 Open Source Linux 소프트웨어 패키지 및 도구 세트다.
TIDL 의 목적은 Machine Learning/Neural Network 애플리케이션을 위한
이기종 장치의 복잡성을 숨기고 개발자가 특정 요구 사항에 집중할 수 있도록 돕는 것이다.
이러한 방식으로 ARM 코어는 Deep Learning Task 의
무거운 연산 부하를 없애고 애플리케이션의 다른 역할에 사용할 수 있다.
이것은 또한 전통적인 컴퓨터 비전(OpenCV 를 통해)을 사용하여 Deep Learning Algorithm 을 보완한다.

현재 TIDL SW 는 주로 Device 파일 시스템에 저장된 Offline Pre-Trained Model 을 사용하여
Convolution Neural Network 추론을 활성화한다(Target Device 에 대한 Training 이 필요 없음)
Caffe 또는 Tensorflow-slim Framework 를 사용하여 Train 된 Model 을
TI Devices 에서 효율적으로 실행하기 위해 import 및 convert(제공된 import tool 사용) 할 수 있다.

추가적인 성능 향상은 Caffe 의 Caffe-Jacinto fork 를 사용하여 train 하여 얻을 수 있다.
여기에는 Convolution weight tensor 를 희박하게 만들기 위한 함수가 포함되어 있어
Convolution Layer 의 3 배 ~ 4 배 성능 향상을 위한 기회를 제공한다.

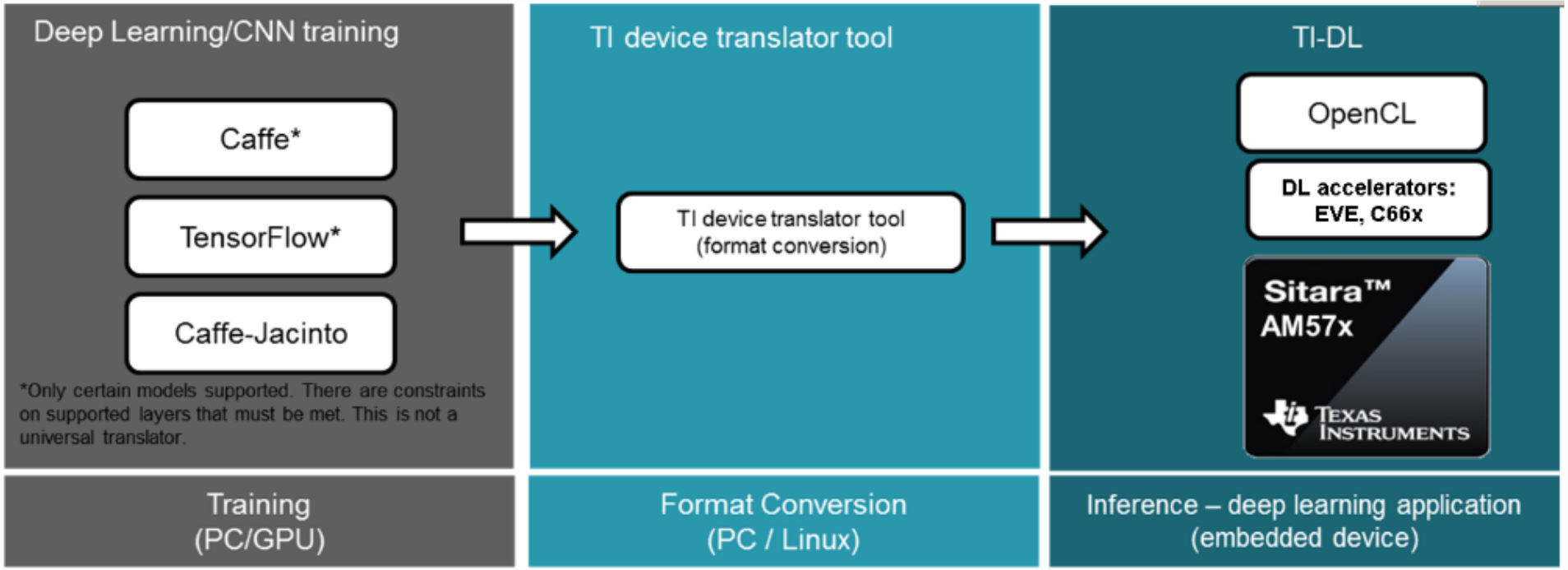


Fig. 3.3 Deep Learnining development flow on AM57xx devices

3.14.1.2. Application space

TIDL SW 의 최신 버전은 Computer Vision Deep Learning 응용 프로그램을 Target 으로 한다.
예를 들어 비전 컴퓨터, 바코드 판독기, 머신 비전 카메라, 산업 자동화 시스템,
광학 검사 시스템, 산업용 로봇, 통화 카운터, 점유 감지기, 스마트 가전 및 무인 차량이 있다.
이러한 경우, Color Image 텐서(BGR 평면에 대해 폭 x 높이 x 3)가 통상적으로 사용되지만
다른 데이터 유형(예를 들어, gray scale 및 depth plane: 폭 x 높이 x 2 텐서)을 사용할 수 있다.

모델 및 작업에 따라 TIDL 입력 데이터는 비슷하지만 작업에 따라 출력 데이터가 달라진다:

Task	Output Data
Image Classification	종류를 판별하기 위한 가능성을 가진 1D 벡터 최상위 순위는 분류된 종류에서 승자를 나타낸다(즉 데이터 클래스의 객체가 입력됨)
Image Pixel Segmentation	2D 행렬: 너비 x 높이. 각 셀은 Model 이 식별 할 수 있는 0 ~ max_class_index 사이의 정수 값으로 설정된다.
Image Object Detection	튜플 목록 각 튜플은 종류 색인, 탐지 확률, 왼쪽 위 모서리(xmin, ymin), 경계 상자의 폭과 높이를 포함한다.

다른 적용 분야(음성, 오디오, 예측 유지)를 다루는 추가 예제가 Road Map 에 나와 있다.
Convolution Network Topologies 외에도 RNN / LSTM 계층 및 Topology 지원,
순차적 데이터 처리의 대상 지정은 향후 release 에서 계획하고 있다.

3.14.1.3. Supported Devices

TIDL SW 는 DSP 또는 EVE 또는 가속기를 모두 갖춘 AM57xx Sitara Device 용이다.

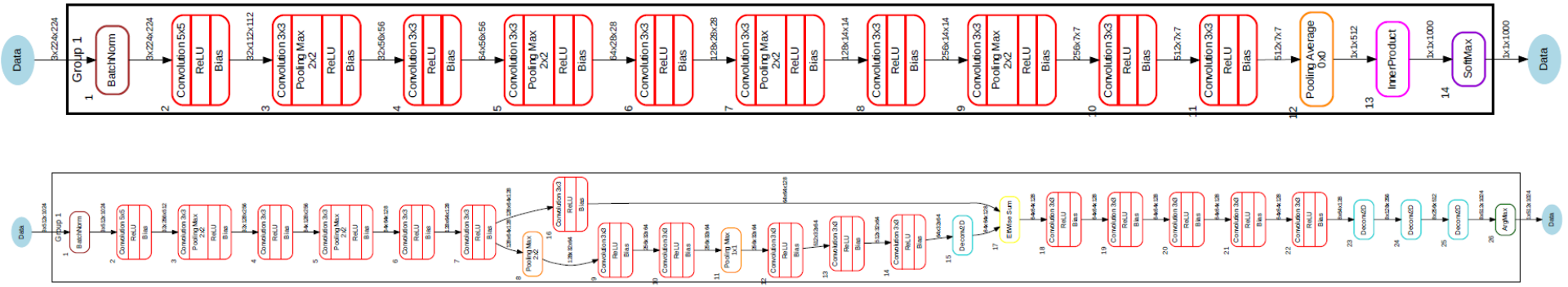
- **AM5749 (2xEVEs + 2xC66x DSPs)**
- **AM5746 (2xC66x DSPs)**
- **AM572x (2xC66x DSPs)**
- **AM571x (1xC66x DSP)**
- **AM5706 (1xC66x DSP)**
- **AM5708 (2xC66x DSP)**

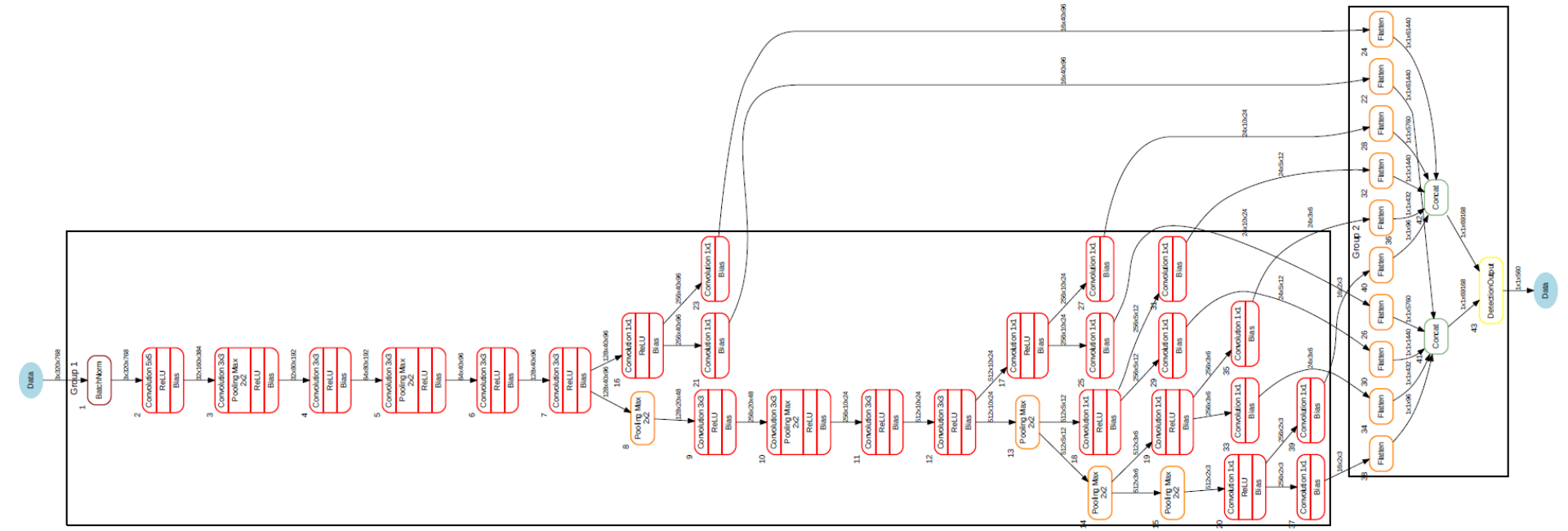
3.14.1.4. Verified networks topologies

Caffe 또는 TF 에서 실행할 수 있는 임의의 네트워크 토폴로지에 대해
TIDL 은 검증되지 않았다는 점을 이해해야 한다.
대신 우리는 전력 제한 임베디드 디바이스에서 실행되는 것이
합리적인 레이어 및 네트워크 토폴로지의 검증에 중점을 두었다.
일부 네트워크는 수 와트의 전력 예산에 비해 너무 복잡하다.
따라서 매우 계산 집약적이지 않고 매우 높은 메모리 요구 사항이 없는 네트워크에 중점을 둔다.
TIDL SW 에서 검증된 토폴로지는 아래와 같다:

- **Jacinto11(ResNet10 과 유사), 분류 네트워크**
- **JSeg21, 픽셀 수준 Segmentation Network**
- **JDetNet(SSD-300 / 512 와 유사), Object Detection Network**
- **SqueezeNet**
- **InceptionV1**
- **MobileNetV1**

처음 세 가지 그래프(TIDL Viewer Tool 을 사용하여 생성)는 아래와 같다:





다른 네트워크 토폴로지는 가능하지만 검증해야 한다.
어떤 경우에는 특정 매개 변수 관련 제약이 있지만 분류, 분할 및 탐지 작업을 위해 필요한 계층의 대부분(추론을 위해서만!)이 구현된다.

3.14.1.5. Neural network layers supported by TIDL

다음 레이어 유형 / 추론 기능이 지원된다:

1. Convolution Layer
2. Pooling Layer (Average and Max Pooling)
3. ReLU Layer
4. Element Wise Layer (Add, Max, Product)
5. Inner Product Layer (Fully Connected Layer)
6. Soft Max Layer
7. Bias Layer
8. Deconvolution Layer
9. Concatenate layer
10. ArgMax Layer
11. Scale Layer
12. PReLU Layer
13. Batch Normalization layer
14. ReLU6 Layer
15. Crop layer
16. Slice layer
17. Flatten layer
18. Split Layer
19. Detection Output Layer

3.14.1.6. Constraints on layer parameters

TIDL Lib 의 현재 release 의 Layer 에는 특정 매개 변수와 관련된 제약 조건이 있다:

- Convolution Layer
  – 최대 7 x 7 kernel 크기 확인(더 높은 값에 대해서도 작동하지만 유효성을 검증하지 않음)
  – 매개 변수 값에 대한 확대 / 축소 : 1, 2, 4
  – stride 값 1 과 2 가 지원됨
  – 조밀한 Convolution Flow 는 stride = 1 및 dilation = 1 인 1 x 1 및 3 x 3 kernel 에서만 지원됨
  – 최대 지원 입력 및 출력 채널 수는 1024 에 해당함

- Deconvolution Layer
  – Group 수는 채널 수와 같아야 함
  – 지원되는 Stride 값은 2 다.

- Arg Max
  – EVE 코어는 최대 15 개의 입력 채널을 지원하고 DSP 코어는 최대 6 개의 채널을 지원함
  – out_max_val = false 및 top_k = 1 (기본값) 및 axis = 1 (채널에서만 지원됨)

- InnerProductLayer
  – 지원되는 최대 입력 및 출력 노드는 4096 임
  – 입력 데이터를 병합해야함(즉 입력 데이터에 대해 C = 1 및 H = 1)
  – 평평한 층은 C > 1 및 H > 1 에서 이 Layer 이전에 사용될 수 있음
  – 전체 평균 풀링이 출력을 flatten 시킬 수 있다면

- Spatial Pooling Layer
  – 평균 및 최대 풀링은 Stride 1, 2, 4 및 kernel 크기 2 x 2, 3 x 3, 4 x 4 등에서 지원된다.
    고정되지 않은 Pooling 은 지원되지 않는다.
  – Global 풀링은 평균 및 최대 모두를 지원한다.
    출력 데이터 N = 1 및 H = 1 이며 출력 W 는 입력 'N' 으로 갱신된다.

- **BiasLayer**
  – 채널당 하나의 Scalar Bias 만 지원됨

- **CancatLayet**
  – 연결은 채널(axis = 1; 기본값)에서만 지원됨

- **CropLayer**
  – 공간 자르기만 지원된다(axis = 2; 기본값)

- **FlattenLayer**
  – 'N' 을 변경하지 않고 C = 1, H = 1 로 함

- **ScaleLayer**
  – 채널당 하나의 Scalar Scale 과 Bias 만 지원됨

- **SliceLayer**
  – Slice 는 채널 전체에서만 지원됨(axis = 1; 기본값)

- **SoftmaxLayer**
  – 입력 데이터를 병합해야함(즉 입력 데이터에 대해 C = 1 및 H = 1)

- **SSD**
  – Caffe-Jacinto 기반 SSD 네트워크만 검증됨
  – Reshape, Permute Layer 는 SSD 네트워크의 컨텍스트에서만 지원됨
  – "share_location" 이 true 여야함
  – 4 및 5 head 로 테스트되었음
  – SaveOutputParameter 는 TIDL 추론에서 무시됨
  – Code_type 은 CENTER_SIZE 로만 테스트됨

- **Tensorflow**
  – Slim 기반 모델만 검증되며 Model 제작을 위한 예로 InceptionNetV1 및 mobilenet_1.0 을 아래에서 참조하라.
  – TF-Slim: https://github.com/tensorflow/models/tree/master/research/slim

3.14.1.7. Important TIDL API framework features

TIDL API 의 현재 구현은 아래와 같은 중요 기능을 제공한다:

- 서로 다른 EVE/DSP 의 여러 네트워크를 동시에 실행할 수 있다.
  가속기를 사용할 수 있는 여러 network 를 동시에 실행 할 수 있다.
  AM5749 의 경우 최대 개수는 4 개(2xEVE 에 2 개, 2xDSP 에 2 개) - 기본 CMEM Block 크기를 늘릴 필요가 있다.
  (가속기 당 최대 64 MB 이상이 좋고 - 기본 설정은 192 MB 에 해당한다)
  태스크 실행의 단일 유닛은 완전한 네트워크다.
  즉 일반적으로 하나의 EVE 또는 DSP 에서만 처음부터 끝까지 실행된다.
  하나의 EVE 에서 자체 출력을 사용하고 두 번째 EVE 에서 자체 입력을 사용하는
  동일하거나 다른 Network 를 병렬로 하나의 Network 를 시작할 수 있다.

- 동일한 TIDL API 기반 응용 프로그램 코드를 EVE 또는 DSP 가속기에서 실행할 수 있음
  이것은 device type parameter 를 수정하여 간단히 달성 할 수 있고
  http://downloads.ti.com/mctools/esd/docs/tidl-api/api.html#_CPPv2N4tidl10DeviceTypeE
  더 자세한 것은 Introduction to Programming Model 을 보라.
  http://software-dl.ti.com/processor-sdk-linux/esd/docs/05_00_00_15/linux/Foundational_Components_TIDL.html#introduction-to-programming-model

- 특정 Network 는 EVE 와 DSP 간에 분리 될 수 있다.
  특정 Network(JDetNet 은 하나의 예)의 경우 Layer Group 이라는 개념을 사용하는 것이 좋다.
  하나의 Layer Group 은 EVE 와 DSP 에서 실행된다.
  DSP 에서 더 빠르게 실행되는 Layer 는 거의 없다.
  SoftMax, Flatten 및 Concat Layer 가 빠르다.
  따라서 이 경우 단일 Network 에 DSP + EVE 를 사용하고 있다.

3.14.2. Examples and Demos:
3.14.2.1. TIDL API examples

이 TIDL release 는 소스에서 5 가지 예가 제공되며
Linux x86 에서 최상위 Makefile(tidl-examples 를 대상으로 사용)
또는 target 파일 시스템에서 아래와 같이 cross compile 할 수 있다.
/usr/share/ti/tidl/examples 에서 make 를 사용한다.

Example	Link
Imagenet Classification	Image classification
Segmentation	Pixel segmentation
SSD_multibox	Single shot Multi-box Detection
test	Unit test
Classification with class filtering	tidl-matrix-gui-demo

3.14.2.2. Matrix GUI demo

부팅 할 때 많은 데모를 시작 할 수 있는 다중 버튼으로 Matrix-GUI 가 시작된다.
현재 출시된 PLSDK 5.0 에는 3 개의 버튼이 있는 "TI Deep Learning" 하위 메뉴가 있다.

- 미리 녹화딘 실제 비디오 클립이 있는 Jacinto11 Model 을 사용한 ImageNet Classification 에 해당한다.
- 미리 녹화 된 비디오 클립과 함께 Jacinto11 Model,
  여러 정적 비디오 이미지(ImageMagick 변환 도구 사용)를 사용하는 ImageNet Classification
- 라이브 카메라 입력이 있는 Jacinto11 모델을 사용한 ImageNet Classification

미리 녹화된 클립에서 비디오 입력이 나오며 Jacinto11 Model 을 사용하는 Imagenet Classification 이다.
https://github.com/tidsp/caffe-jacinto-models/tree/caffe-0.17/trained/image_classification/imagenet_jacintonet11v2/sparse
GStreamer 파이프라인 (IVAHD 관련)을 통해 실시간으로 디코딩되어 OpenCV 처리 파이프라인으로 전송된다.
라이브 카메라 입력(기본 640 x 480 해상도) 또는 디코딩 된 비디오 클립(320 x 320 해상도)은
TIDL API 에 보내기 전에 실행 시간(OpenCV API 사용)에서 224 x 224 로 중앙 축소 된다.
이 처리 결과는 표준 Imagenet Classification 출력(1000 요소가 있는 1D 벡터)이다.
또한 비디오 클립이나 라이브 카메라 입력에 포함될 것으로
예상되는 객체의 하위 집합을 정의하는 규정이 있으므로 분류 결과가 발생한다.
이렇게하면 command line 인수로 제공된 class 목록을 사용하여 추가적인 Decision Filtering 을 수행 할 수 있다.
파란색 경계 사각형(주 이미지 창에서)은 유효한 탐지가 보고 된 경우에만 표시된다.
마지막으로 성공한 분류 문자열은 다음 검색까지 보존된다.
(따라서 객체가 발견되지 않으면 파란색 사각형은 사라지지만 마지막 븐류 문자열은 그대로 유지된다)

Matrix-GUI 에서 호출 된 실행 파일은 /usr/share/ti/tidl/examples/tidl_classification 에 있다.

```
root@am57xx-evm:/usr/share/ti/tidl/examples/classification# ./tidl_classification  -h
Usage: tidl
  Will run all available networks if tidl is invoked without any arguments.
  Use -c to run a single network.
Optional arguments:
-c                    Path to the configuration file
-n <number of cores>  Number of cores to use (1 - 4)
-t <d|e>              Type of core. d -> DSP, e -> EVE
-l                    List of label strings (of all classes in model)
-s                    List of strings with selected classes
-i                    Video input (for camera:0,1 or video clip)
-v                    Verbose output during execution
-h                    Help
```

다음은 라이브 카메라 입력을 사용하는 분류의 예다(/usr/share/ti/tidl/examples/classification 폴더에서 호출됨)
주 이미지 창에서 마우스 오른쪽 버튼을 클릭하면 언제든지 중지 할 수 있다:

```
cd /usr/share/ti/tidl/examples/classification
./tidl_classification -n 2 -t d -l ./imagenet.txt -s ./classlist.txt -i 1 -c ./stream_config_j11_v2.txt
```

/usr/share/ti/tidl/examples/classification 폴더에서 호출 된 또 다른 예제,
사전 녹화된 비디오 입력(test2.mp4)을 사용한 classification –
주 이미지 창에서 마우스 오른쪽 버튼을 클릭하여 언제든지 중지할 수 있다:
비디오 클립은 최대 프레임 수(stream_config_j11_v2.txt 에 지정된 값)를 초과하지 않는 한 반복된다.

```
cd /usr/share/ti/tidl/examples/classification
./tidl_classification -n 2 -t d -l ./imagenet.txt -s ./classlist.txt -i ./clips/test2.mp4 -c ./stream_config_j11_v2.txt
```

AM5749 에서 위의 두 예제는 "-t" 매개 변수를 "d" 에서 "e" 로 변경하여 EVE 에서도 실행할 수 있다:

```
cd /usr/share/ti/tidl/examples/classification
./tidl_classification -n 2 -t e -l ./imagenet.txt -s ./classlist.txt -i 1 -c ./stream_config_j11_v2.txt
./tidl_classification -n 2 -t e -l ./imagenet.txt -s ./classlist.txt -i ./clips/test2.mp4 -c ./stream_config_j11_v2.txt
```

imagenet.txt 는 구성 파일(stream_config_j11_v2.txt)에
지정된 Model 에서 감지 할 수 있는 모든 클래스(레이블)의 목록이다.
필터링 된(허용 된) 탐지 목록은 ./classlist.txt 에 지정된다(imagenet.txt 의 문자열 하위 집합 사용):
예) 현재 다음 하위 집합이 사용된다:

- coffee_mug
- coffeepot
- tennis_ball
- baseball
- sunglass
- sunglasses
- water_bottle
- pill_bottle
- beer_glass
- fountain_pen
- laptop
- Notebook

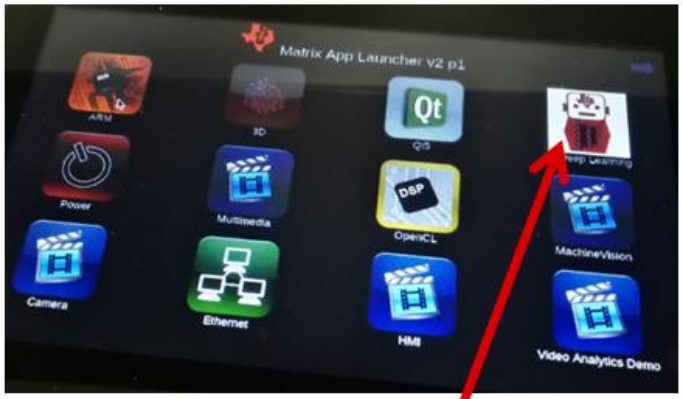
다른 입력을 사용하는 class 의 다른 group 을 사용자 정의 test 에 사용할 수 있다.
이 경우 정사각형 Image(정사각형이 아닌 경우 먼저 중앙 자르기)를 다운로드하고
Linux x86(ImageMagick 및 ffmpeg 이 설치된 이미지) 폴더에 저장하라.
분류 예제에서 사용할 수 있는 합성 비디오 클립을 만드려면 아래 명령을 실행해야 한다:

```
# Linux x86 commands to create short video clip out of several static images that are morphed into each other
convert ./*.jpg -delay 500 -morph 300 -scale 320x320 %05d.jpg
ffmpeg -i %05d.jpg -vcodec libx264 -profile:v main -pix_fmt yuv420p -r 15 test.mp4
```

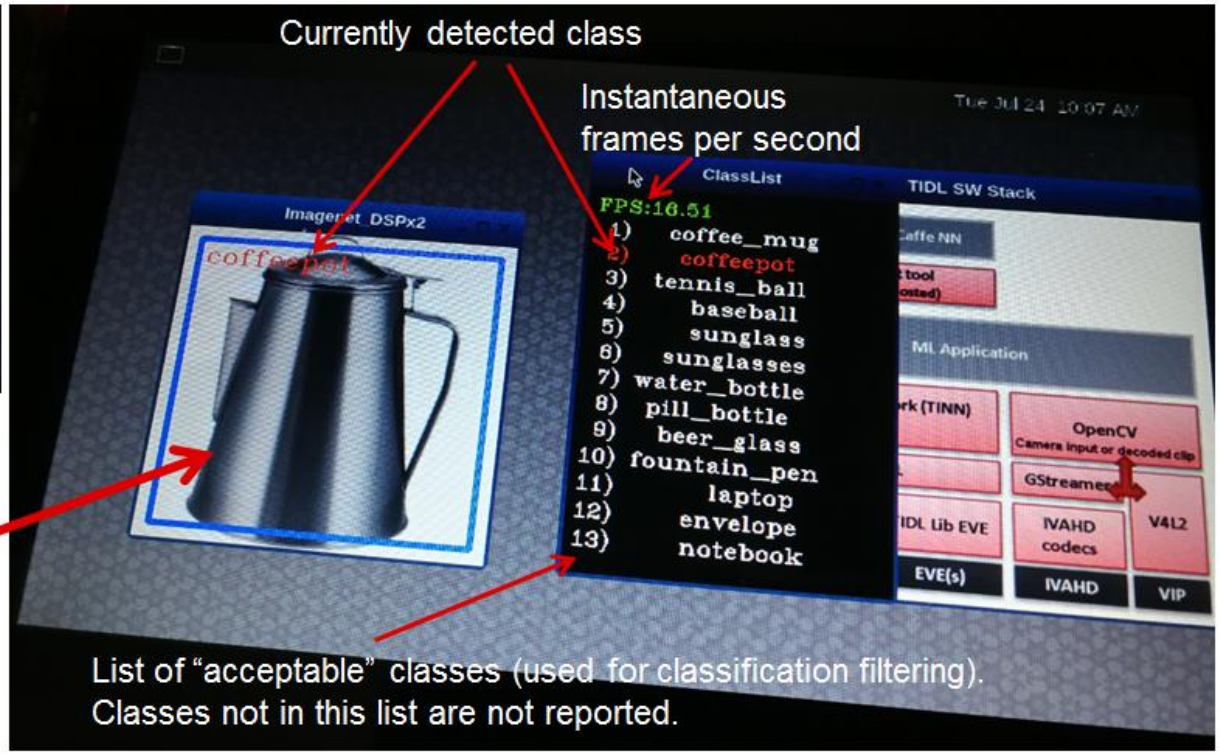
비디오 클립을 캡쳐하거나 외부에서 준비하는 경우(예: 스마트 폰 사용)
Object 를 중앙에 배치해야 한다(실행시 크기 조정 및 중앙 자르기가 필요함)
그런 다음 /usr/share/ti/tidl/examples/classification/clips/ 에 복사하거나 같은 폴더에 있는 test1.mp4 를 덮어 쓴다.

3.14.2.2.1. Description of Matrix-GUI example

예제는 "iamgenet" 및 "test" 예제를 기반으로하며 의사 결정 필터링 및 시각화와 관련된 추가 사항은 거의 없다.
두 개의 소스 파일만 있다:



Deep Learning Submenu



Main image window (either from live camera or decoded video clip) – to exit, do right-click on this window

List of "acceptable" classes (used for classification filtering).
Classes not in this list are not reported.

- **main.cpp**
  - **명령 줄 인수(ParseArgs)를 구문 분석하고 프로그램 사용 방법을 보여줌**
  - **실행 개체(프레임 입력 및 출력 버퍼)뿐 아니라 구성(네트워크 모델 사용) 및 실행 프로그램(DSP 또는 EVE)을 초기화한다.**
  - **사용 가능한 클래스 목록이 있는 TIDL 정적 이미지, 디코딩 된 클립 또는 사용 가능한 카메라 목록 및 사용 가능한 클래스 목록이 있는 카메라 입력으로 창을 만든다.**
  - **main 처리 루프가 RunConfiguration 에 있음.**
  - **추가 기능: tf_postprocess(탐지 정렬 및 하위 집합에서 최상위 후보가 활성화되어 있는지 확인) 및 ShowRegion(마지막 3 개 프레임에 대한 결정이 안정된 경우)**

- **findclasses.cpp**
  - **모든 모델 레이블(예: 1000 클래스 imagenet 분류 모델의 경우 1000 문자열)을 읽는 populate_labels() 함수**
  - **함수 populate_selected_items() 는 결정 필터링에 사용되는 레이블 이름을 읽고 검증한다(이전 유효한 값 목록 사용)**

3.14.3. Developer’s guide:

3.14.3.1. Software Stack

SW 의 복잡성은 더 나은 이해를 위해서 제공된다.
사용자는 TIDL API 만을 기반으로 프로그래밍을 수행해야 한다.

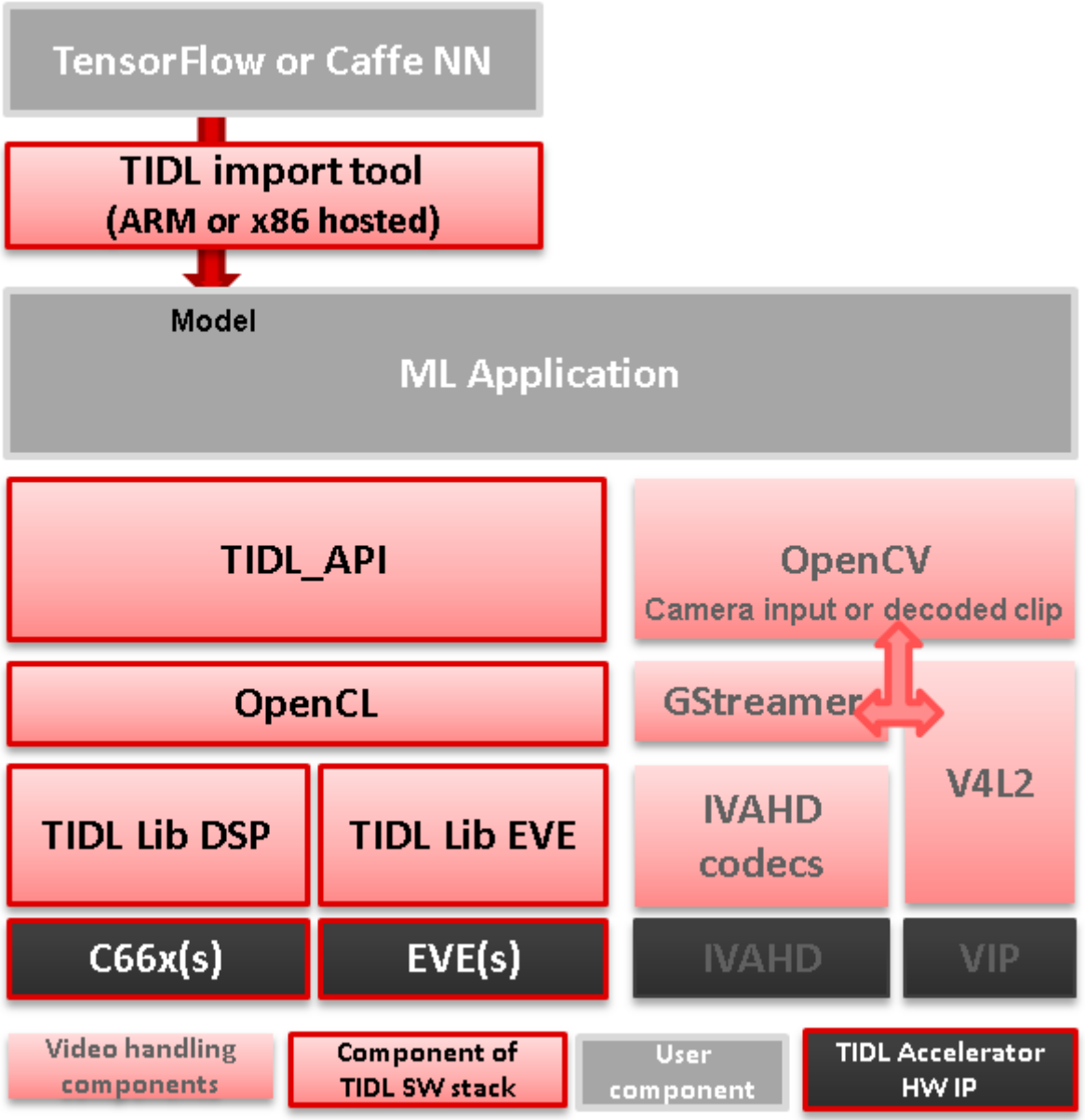
TIDL 이 DSP 를 가속기로 사용하는 경우 세 가지 SW 계층이 있다:

- DSP C66 에서 실행되는 TIDL 라이브러리
- A15 에서 실행되는 OpenCL Run-Time 및 DSP
- TIDL API Host Wrapper, User 공간 라이브러리

TIDL 에서 EVE 를 가속기로 사용하는 경우 4 개의 SW 계층이 있다:

- EVE 에서 실행되는 TIDL 라이브러리
- EVE 와 A15 간의 프록시 역할을 하는 M4 서비스 레이어(OpenCL 의 일부로 간주되는)
- A15 에서 실행되는 OpenCL Run-Time 뿐만 아니라 M4(TIDL OpenCL 모니터 역할을 위해 예약된 IPU1)
- TIDL API Host Wrapper, 사용자 공간 라이브러리

TIDL API 패키지 API 는 DSP 를 사용하든 EVE 를 사용하든 (또는 둘 다) 상관없이 동일하다.
사용자는 매개 변수를 통해 가속기만 지정하면 된다.



3.14.3.2. Additional public TI resources

TI 팀이 관리하는 두 가지 Caffe 관련 리포지토리를 따르면 train 단계에 필요한 도구를 제공한다.
TIDL 추론 SW 에 대한 train 을 위한 주요 정보 소스로 사용하라.
Caffe 소스 트리를 수정하여 TIDL 추론을 통해 더 높은 컴퓨팅 성능을 구현할 수 있다.

Repo/URL	Description
Caffe-jacinto	BVLC/Caffe 에서 파생 된 NVIDIA/caffe 의 복제본이다. 이 복제본의 수정으로 인해 희박한 양자화 된 CNN Model 을 사용할 수 있어 임베디드 플랫폼에서 사용할 수 있는 복잡성이 낮은 Model 이 되었다. 이 버전의 Caffe 를 복제, 컴파일 및 설치하는 방법은 README.md 를 참조하라.
Caffe-jacinto-models	tidsp/caffe-jacinto 를 사용하 sparse Model 을 교육하기 위한 예제 스크립트를 제공한다. 이 스크립트는 희소한 CNN Model 을 train 하여 임베디드 플랫폼에서 사용할 수 있는 복잡성이 낮은 Model 을 만든다. 이 저장소는 또한 사전 훈련된 모델을 포함한다. https://github.com/tidsp/caffe-jacinto-models/tree/caffe-0.17/trained 데이터 집합을 준비하고 train 을 실행하는데 사용할 수 있는 추가 스크립트는 Script 폴더에서도 사용할 수 있다. https://github.com/tidsp/caffe-jacinto-models/tree/caffe-0.17/scripts

3.14.3.3. Introduction to Programming Model

공개된 TIDL API 인터페이스는
http://downloads.ti.com/mctools/esd/docs/tidl-api/intro.html 에 자세히 설명되어 있다.

현재 release 에는 독립적인 EVE 또는 C66x Core 에서 병렬로 실행할 수 있는 하나
또는 여러 개의 신경망 모델을 사용할 수 있는 3 가지 클래스만 있다.

단일 처리 단위는 완성 될 때까지 단일 가속기(EVE 또는 DSP)에 의해
전형적으로 처리되는 텐서(예를 들어 하나의 이미지 프레임이지만 다른 입력도 가능)다.
그러나 어떤 경우에는 성능 측면에서 Network 계층을 두 개의 계층 그룹으로 분리하는 것이 정당화된다.
하나의 Layer Group 을 EVE 에서 실행하고 두 번째 Layer Group 을 DSP 에서 실행 할 수 있다.
이것은 순차적으로 이루어진다.

최상위 계층 TIDL API 및 OpenCL 은 주로 (NID 가 아닌 TIDL SW 와 관련하여) 서비스 SW 계층이며,
프로그래밍 모델, IPC 메커니즘 및 메모리 관리를 간소화하는데 도움이 된다.
원하는 기능은 RTOS 환경에서 EVE 또는 DSP 로 실행되는 TIDL Lib 에 의해 제공된다.
이 SW 계층은 폐쇄형 펌웨어로 제공되며 최종 사용자가 그대로 사용한다.

3.14.3.4. Target file-system:

3.14.3.4.1. Firmware

OpenCL 펌웨어에는 사용자 정의 가속기 모델에 따라
미리 코딩 된 DSP TIDL Lib(하드 코드 된 kernel 포함) 및 EVE TIDL Lib 이 포함된다.
OpenCL 펌웨어는 Linux 부팅 직후 DSP 및 M4/EVE 에 다운로드 된다.

```
- dra7-ipu1-fw.xem4 -> /lib/firmware/dra7-ipu1-fw.xem4.opencl-monitor
- dra7-dsp1-fw.xe66 -> /lib/firmware/dra7-dsp1-fw.xe66.opencl-monitor
- dra7-dsp2-fw.xe66 -> /lib/firmware/dra7-dsp2-fw.xe66.opencl-monitor
```

3.14.3.4.2. User space components

사용자 공간 TIDL 구성 요소는 /usr/share/ti/tidl 폴더와 해당 하위 폴더에 포함된다.
하위 폴더 이름 및 설명은 아래와 같다:

Sub folder	Description of content
examples	테스트(파일에서 파일로 예제), imagenet 분류, 이미지 분할 및 SSD 멀티 박스 예제가 있다. imagenet 하나를 기반으로 하는 matrix GUI 예제는 tidl_classification 폴더에 있다.
utils	Model 및 시뮬레이션 import 를 위한 ARM 32 비트 바이너리 및 예제 import tool 동작을 실행하기 위한 구성 파일이 있다.
viewer	가져온 Model Parser 및 Dot Graph Creator 에 해당한다. 입력은 TIDL Model 이며 출력은 x86 용 Dot Utility 를 사용하여 PNG 또는 PDF 형식으로 변환할 수 있는 .dot 파일이다.
tidl_api	TIDL API 구현 소스

3.14.3.5. Input data format

현재 release 는 주로 2D 입력과 함께 사용된다.
가장 빈번한 2D 입력 텐서는 color 이미지다.
Model Training 중에 사용된 것과 같은 형식으로 준비되어야 한다.
일반적으로 이것은 BGR 평면 Interlaced Format(OpenCV 에서 공통)을 따른다.
즉 첫 번째 2D 배열은 파란색 평면이고 그 다음은 녹색 평면이고 마지막은 빨간색 평면이다.

그러나 예로 입력 전용의 두 평면을 갖는 것이 완벽하게 가능하다.
예: Lidar 거리 측정이 가능한 하나의 평면과 조명이 있는 두 번째 평면이다.
이것은 training 도중 동일한 형식이 사용되었다고 가정한다.

3.14.3.6. Output data format

- Image classification
  출력에는 1D 벡터가 있으며, class 당 1 바이트(log(softmax))에 해당한다.
  Model 의 class 가 100 개인 경우 출력 버퍼의 길이는 100 바이트이며
  Model 의 class 가 1000 개인 경우 출력 버퍼의 길이는 1000 바이트가 된다.

- Image segmentation
  출력 버퍼는 일반적으로 WxH(입력 이미지의 너비와 높이)인 2D 버퍼에 해당한다.
  각 바이트는 입력 이미지에서 픽셀의 클래스 색인이다.
  일반적으로 클래스 수는 1 ~ 2 개에 해당한다(단 255 개 미만이어야 한다)

- Object detection
  출력 버퍼는 클래스 인덱스, 경계 상자(매개변수 4 개) 및 선택적으로 확률 metric 을 포함한 튜플의 목록이다.

3.14.3.7. Import Process

Import 절차는 두 단계로 이루어진다.

- 첫 번째 단계에서는 Model Parameter 및 Network Topology 를 Parsing 하고
  TIDL Lib 이 이해할 수 있는 사용자 지정 형식으로 변환한다.

- 두 번째 단계는 각 Layer 에 대한 활성화 범위를 찾아 Dynamic Quantization 프로세스의 교정을 수행한다.
  이는 양자화 프로세스에 중요한 초기값을 추정하는 시뮬레이션(기본 C 구현 사용)을 호출하여 수행된다.
  이 값은 나중에 입력 프레임간에 강한 시간 상관 관계가 있다고 가정할 때 Frame 단위로 업데이트 된다.

import 프로세스 동안, 일부 Layer 는 하나의 TIDL Layer(예: Convolution 및 ReLU Layer)로 통합된다.
이는 특정 작업을 무료로 허용하는 EVE Architecture 보다 효과적으로 활용하기 위해 수행된다.
TIDL Network Viewer 를 사용하여 변환된(그러나 동급인) Network 구조를 확인할 수 있다.

우리는 PC 에서 caffe framework 또는 tensorflow framework 를 사용하여
train 된 Model 및 Parameter 를 가져오기 위한 도구(Linux x86 또는 ARM Linux 포트)를 제공한다.
이 도구는 import 구성 파일을 통해 다양한 parameter 를 받아들이고
여러 EVE 및 DSP 코어에서 TIDL Library 를 사용하여 코드가 실행될 Model 및 Parameter 파일을 생성한다.
import 구성 파일은 {TIDL_install_path}/test/testvecs/config/import 에 있다.

샘플 사용법: tidl_model_import.out ./test/testvecs/config/import/tidl_import_jseg21.txt

import 구성 parameter 목록은 아래와 같다:

Parameter	Configuration
randParams	0 또는 1 일 수 있고 기본값은 0 이다. 이 값을 0 으로 설정하면 Model 에서 양자화 parameter 가 생성되고 그렇지 않으면 임의의 양자화 parameter 가 생성된다.
modelType	0 또는 1 일 수 있고 기본 값은 0 이다. caffeImport 프레임워크에서 읽으려면 이 값을 0 으로 설정하고 tensorflow 프레임워크에서 읽으려면 1 로 설정한다.
quantizationStyle	training 프레임워크에 의한 고정된 양자화의 경우 '0', 동적 양자화의 경우 '1' 이 될 수 있다. 기본값은 1 이며 현재 동적인 양자화만 지원된다.
quantRoundAdd	0 ~ 100 사이의 값을 취할 수 있으며 기본값은 50 이다. integer 로 반올림하는 동안 quantRoundAdd / 100 이 추가된다.
numParamBits	4 ~ 12 까지의 값을 가질 수 있고 기본값은 8 이다. 이것은 parameter 를 양자화하는데 사용되는 비트 수다.
preProcType	0 ~ 3 까지의 값을 취할 수 있고 기본값은 0 이다.
Conv2dKernelType	각 Layer 에 대해 0 또는 1 이 될 수 있고 모든 Layer 에 대한 기본값은 0 이다. Sparse Convolution 을 사용하려면 0 으로 설정하고 그렇지 않으면 1 로 설정하여 dense Convolution 을 사용한다.
inElementType	0 또는 1 일 수 있고 기본값은 1 이다. 8 비트 부호없는 입력은 0, 8 비트 부호 있는 입력은 1 로 설정한다.
inQuantFactor	0 보다 큰 값을 취할 수 있고 기본값은 -1 이다.

Parameter	Configuration
rawSampleInData	0 또는 1 일 수 있고 기본값은 0 이다. 입력 데이터가 Encoding 되어 있으면 0 으로 설정하고 입력이 RAW 데이터면 1 로 설정한다.
numSampleInData	0 보다 크고 기본값은 1 이다.
foldBnInConv2D	0 또는 1 일 수 있고 기본값은 1 이다.
inWidth	너비는 입력 이미지의 크기 보다 클 수 있고 0 보다 크다.
inHeight	입력 이미지의 높이이며 0 보다 크다.
inNumChannels	입력 채널 수 이고 1 ~ 1024 까지 가능하다.
sampleInData	입력 데이터 파일명임
tidlStatsTool	TIDL reference 가 실행 파일인지 여부
inputNetFile	입력 net 파일명(Training 프레임워크로부터)
inputParamsFile	입력 매개 변수 파일 이름(Training 프레임워크로부터)
outputNetFile	stats 로 업데이트 될 Output Model net 파일명
outputParamsFile	출력 매개 변수는 파일명
layersGroupId	주어진 CORE 에서 처리해야하는 Layer Group 을 나타낸다. 사용예는 SSD import config 를 참조하라.

3.14.3.7.1. Example of import config files

Model 변환 명령(configuration 파일에 지정됨)

```
./tidl_model_import.out  ./import/tidl_import_j11_v2.txt
```

3.14.3.7.2. Sample configuration file (tidl_import_j11_v2.txt)

Model 변환 명령(configuration 파일에 지정됨)

```
# Default - 0
randParams          = 0

# 0: Caffe, 1: TensorFlow, Default - 0
modelType           = 0

# 0: Fixed quantization By tarininng Framework, 1: Dynamic quantization by TIDL, Default - 1
quantizationStyle   = 1

# quantRoundAdd/100 will be added while rounding to integer, Default - 50
quantRoundAdd       = 50

# 0 : 8bit Unsigned, 1 : 8bit Signed Default - 1
inElementType       = 0

rawSampleInData     = 1
```

```
# Fold Batch Normalization Layer into TIDL Lib Conv Layer
foldBnInConv2D     = 1

# Weights are quantized into this many bits:
numParamBits       = 12

# Specify sparse of dense
Conv2dKernelType   = 1

# Network topology definition file
inputNetFile       = "import/dogs_deploy.prototxt"

# Parameter file
inputParamsFile    = "import/DOGS_iter_34000.caffemodel"

# Translated network stored into two files:
outputNetFile      = "tidl_net_imagenet_jacintonet11v2.bin"
outputParamsFile   = "tidl_param_imagenet_jacintonet11v2.bin"

# Calibration image file
sampleInData = "import/test.raw"

# Reference implementation executable, used in calibration (processes calibration image file)
tidlStatsTool = "import/eve_test_dl_algo_ref.out"
```

3.14.3.7.3. Import tool traces

변환하는 동안 import tool 은 탐지된 Layer 및 해당 parameter 를 report 하는 trace 를 생성한다.
(마지막 열은 입력 텐서 치수 및 출력 텐서 치수를 나타냄)

```
Processing config file ./tempDir/qunat_stats_config.txt !
0, TIDL_DataLayer                , 0,  -1 , 1 , x , x , x , x , x , x , x , 0 ,    0 ,    0
1, TIDL_BatchNormLayer           , 1,   1 , 1 , 0 , x , x , x , x , x , x , x , 1 ,    1 ,    3
2, TIDL_ConvolutionLayer         , 1,   1 , 1 , 1 , x , x , x , x , x , x , x , 2 ,    1 ,    3
3, TIDL_ConvolutionLayer         , 1,   1 , 1 , 2 , x , x , x , x , x , x , x , 3 ,    1 ,   32
4, TIDL_ConvolutionLayer         , 1,   1 , 1 , 3 , x , x , x , x , x , x , x , 4 ,    1 ,   32
5, TIDL_ConvolutionLayer         , 1,   1 , 1 , 4 , x , x , x , x , x , x , x , 5 ,    1 ,   64
6, TIDL_ConvolutionLayer         , 1,   1 , 1 , 5 , x , x , x , x , x , x , x , 6 ,    1 ,   64
7, TIDL_ConvolutionLayer         , 1,   1 , 1 , 6 , x , x , x , x , x , x , x , 7 ,    1 ,  128
8, TIDL_ConvolutionLayer         , 1,   1 , 1 , 7 , x , x , x , x , x , x , x , 8 ,    1 ,  128
9, TIDL_ConvolutionLayer         , 1,   1 , 1 , 8 , x , x , x , x , x , x , x , 9 ,    1 ,  256
10, TIDL_ConvolutionLayer        , 1,   1 , 1 , 9 , x , x , x , x , x , x , x , 10 ,   1 ,  256
11, TIDL_ConvolutionLayer        , 1,   1 , 1 , 10 , x , x , x , x , x , x , x , 11 ,  1 ,  512
12, TIDL_PoolingLayer            , 1,   1 , 1 , 11 , x , x , x , x , x , x , x , 12 ,  1 ,  512
13, TIDL_InnerProductLayer       , 1,   1 , 1 , 12 , x , x , x , x , x , x , x , 13 ,  1 ,    1
14, TIDL_SoftMaxLayer            , 1,   1 , 1 , 13 , x , x , x , x , x , x , x , 14 ,  1 ,    1
15, TIDL_DataLayer               , 0,   1 , -1 , 14 , x , x , x , x , x , x , x , 0 ,  1 ,    1
Layer ID    ,inBlkWidth  ,inBlkHeight ,inBlkPitch  ,outBlkWidth ,outBlkHeight,outBlkPitch ,numInChs    ,num
  2           72           64           72           32           28           32           3           32
  3           40           30           40           32           28           32           8            8
  4           40           30           40           32           28           32           32          64
  5           40           30           40           32           28           32           16          16
  6           40           30           40           32           28           32           64         128
  7           40           30           40           32           28           32           32          32
  8           24           16           24           16           14           16          128         256
  9           24           16           24           16           14           16           64          64
 10           24            9           24           16            7           16          256         512
 11           24            9           24           16            7           16          128         128
  Processing Frame Number : 0
```

최종 출력(구성 파일에 제공된 calibration 원시 이미지를 기반으로 함)은 예약 된 이름의 파일에 저장된다.
stats_tool_out.bin 이 파일의 크기를 출력 클래스의 개수(분류의 경우)와 동일해야한다.
예: imagenet 1000 클래스의 경우 1000 바이트가 커야 한다.
최종 BLOB 외에도 모든 중간 결과(개별 Layer 의 활성화)는
./tempDir 폴더(import 가 호출되는 폴더 내부)에 저장된다.
다음은 중간 활성화가 있는 파일의 샘플 목록이다:

- trace_dump_0_224x224.y <- 이 첫 번째 Layer 는
  Desktop Caffe 에서 사용되는 데이터 blob 과 동일해야함(유효성 검사 도중)
- trace_dump_1_224x224.y
- trace_dump_2_112x112.y
- trace_dump_3_56x56.y
- trace_dump_4_56x56.y
- trace_dump_5_28x28.y
- trace_dump_6_28x28.y
- trace_dump_7_14x14.y
- trace_dump_8_14x14.y
- trace_dump_9_7x7.y
- trace_dump_10_7x7.y
- trace_dump_11_7x7.y
- trace_dump_12_512x1.y
- trace_dump_13_9x1.y
- trace_dump_14_9x1.y

3.14.3.7.4. Splitting layers between layers groups

DSP 와 EVE 가속기를 모두 사용하려면 Layergroup 의 개념을 사용하여
Network 를 두 개의 하위 그래프로 분리 할 수 있다.
하나의 Layergroup 이상이 EVE 와 DSP 에서 실행될 수 있다.
EVE 에서 실행되는 첫 번째 group 의 출력이 DSP 의 입력으로 사용된다.

이것은 아래와 같은 방법으로 수행 할 수 있다(Jacinto11 Network 에 대한 예제 제공):

```
# Default - 0
randParams         = 0

# 0: Caffe, 1: TensorFlow, Default - 0
modelType          = 0

# 0: Fixed quantization By tarininng Framework, 1: Dynamic quantization by TIDL, Default - 1
quantizationStyle  = 1

# quantRoundAdd/100 will be added while rounding to integer, Default - 50
quantRoundAdd      = 25

numParamBits       = 8
# 0 : 8bit Unsigned, 1 : 8bit Signed Default - 1
inElementType      = 0

inputNetFile       = "../caffe_jacinto_models/trained/image_classification/imagenet_jacintonet11v2/sparse/deploy.prototxt"
inputParamsFile    = "../caffe_jacinto_models/trained/image_classification/imagenet_jacintonet11v2/sparse/imagenet_jacintonet11v2_iter_160000.caffemodel"
outputNetFile      = "./tidl_models/tidl_net_imagenet_jacintonet11v2.bin"
outputParamsFile   = "./tidl_models/tidl_param_imagenet_jacintonet11v2.bin"

sampleInData = "./input/preproc_0_224x224.y"
tidlStatsTool = "./bin/eve_test_dl_algo_ref.out"
layersGroupId    = 0       1       1       1       1       1       1       1       1       1       1       1       2       2       2       0
conv2dKernelType = 0       0       0       0       0       0       0       0       0       0       0       0       1       1       1       1
```

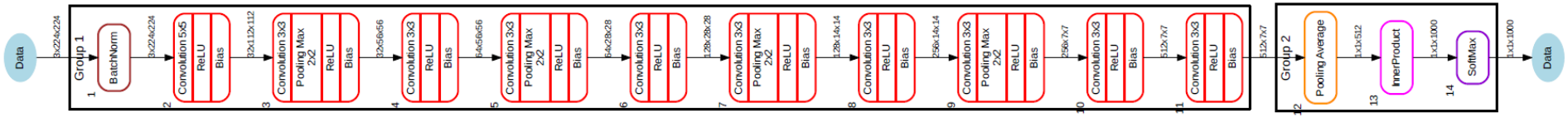
입력 및 출력 Layer 는 Layergroup 0 에 속한다.
Layergroup 1 은 EVE 에 전달되고 Layergroup 2 는 DSP 에 전달된다.

두 번째 행(conv2dKernelType)은 계산이 희박(0)인지 밀도가 높은지(1)를 나타낸다.

변환 후 Network 를 시각화 할 수 있다:

```
tidl_viewer -p -d ./j11split.dot ./tidl_net_imagenet_jacintonet11v2.bin
dot -Tpdf ./j11split.dot -o ./j11split.pdf
```

아래는 그래프다(group 1 은 EVE 에서 실행되고 group 2 는 DSP 에서 실행됨)



Layergroup 1 의 출력은 Layergroup 2 의 입력 버퍼와 공유(공통)되므로 여분의 버퍼 복사 Overhead 가 없다.
이 버퍼 할당으로 인해 EVE 및 DSP 의 순차적 작동이 필요하다.

3.14.3.7.5. Calculating theoretical GMACs needed

이는 계산 중 가장 집중적인 Layer 에 대해 계산할 수 있다:
Convolution Layers 및 Fully Connected Layer
각 Convolution Layer 에는 특정 개수의 입력 및 출력 기능 맵(2D 텐서)이 있다.
입력 특징 맵은 convolution kernel(일반적으로 3x3 이나 5x5, 7x7 ..) 과 함께 사용된다.
따라서 총 MAC 수는 아래와 같이 계산할 수 있다.
Height_input_map x Width_input_map x N_input_maps x N_output_maps x size_of_kernel

```
E.g. for 112x112 feature map, with 64 inputs, 64 outputs and 3x3 kernels, we need:
112x112x64x64x3x3 MAC operations = 4624229916 MAC operations
```

마찬가지로 N_inputs 및 N_outputs 를 사용하여 Fully Connected Layer 의 경우 총 MAC 연산 수가 아래에 있다.

```
E.g. N_inputs = 4096 and N_outputs = 1000,
Fully Connected Layer MAC operations = N_inputs * N_outputs = 4096 * 1000 = 4096000 MAC operations
```

명백히 Convolution Layer 작업 부하가 상당히 높다.

3.14.3.7.6. Mapping to EVE capabilities

각 EVE 코어는 주기 당 16 개의 MAC 작업을 수행 할 수 있다.
누산 된 결과는 40 비트 누산기에 저장되며 Local 메모리에 저장되기 전에 Barrel Shift 될 수 있다.
또한 EVE 는 무료로 ReLU 작업을 수행 할 수 있으므로
Convolution Layer 또는 Fully Connected Layer 가 ReLU Layer 와 통합된다.

이러한 작업을 지원하려면 Local 메모리에 대한 넓은 경로가 필요하다.
외부 DDR 메모리에서 동시 전송은 전용 EDMA 엔진을 사용하여 수행된다.
따라서 EVE 가 Convolution 을 수행 할 때 고속 Local 메모리에 이미 존재하는 활성화 및 가중치에 항상 접근한다.

하나 또는 두 개의 Layer 가 주로 벡터 엔진 및 EDMA 프로그래밍에 사용되는 EVE Local RISC CPU 에 구현된다.
이 드문 경우 EVE CPU 는 완전히 프로그래밍 가능하지만 느린 계산 엔진으로 사용된다.
SoftMax Layer 는 범용 CPU 를 사용하여 구현되며 DSP 또는 A15 구현보다 훨씬 느리다.
SoftMax Layer 는 terminal Layer 이므로 A15(사용자 공간)
또는 DSP(JDetNet 예제에서 구현된 layer2)를 사용하여 SoftMax 를 수행하는 것이 좋다.

3.14.3.8. Viewer tool

Viewer 도구는 import 된 Network Model 을 시각화한다.
자세한 내용은 http://downloads.ti.com/mctools/esd/docs/tidl-api/viewer.html 에서 확인하라.
아래는 command line 예제다:

```
root@am57xx-evm:/usr/share/ti/tidl/examples/test/testvecs/config/tidl_models# tidl_viewer
Usage: tidl_viewer -d <dot file name> <network binary file>
Version: 01.00.00.02.7b65cbb
Options:
-p              Print network layer info
-h              Display this help message
```

```
root@am57xx-evm:/usr/share/ti/tidl/examples/test/testvecs/config/tidl_models# tidl_viewer -p -d ./jacinto11.dot ./tidl_net_imagenet_jacintonet11v2.bin
  #  Name                gId   #i   #o   i0   i1   i2   i3   i4   i5   i6   i7   o   #roi    #ch      h      w   #roi    #ch      h      w
  0, Data              ,  0,  -1 ,  1 ,  x ,  x ,  x ,  x ,  x ,  x ,  x ,  x ,  0 ,    0 ,    0 ,    0 ,    0 ,    1 ,    3 ,  224 ,  224 ,
  1, BatchNorm         ,  1,   1 ,  1 ,  0 ,  x ,  x ,  x ,  x ,  x ,  x ,  x ,  1 ,    1 ,    3 ,  224 ,  224 ,    1 ,    3 ,  224 ,  224 ,
  2, Convolution       ,  1,   1 ,  1 ,  1 ,  x ,  x ,  x ,  x ,  x ,  x ,  x ,  2 ,    1 ,    3 ,  224 ,  224 ,    1 ,   32 ,  112 ,  112 ,
  3, Convolution       ,  1,   1 ,  1 ,  2 ,  x ,  x ,  x ,  x ,  x ,  x ,  x ,  3 ,    1 ,   32 ,  112 ,  112 ,    1 ,   32 ,   56 ,   56 ,
  4, Convolution       ,  1,   1 ,  1 ,  3 ,  x ,  x ,  x ,  x ,  x ,  x ,  x ,  4 ,    1 ,   32 ,   56 ,   56 ,    1 ,   64 ,   56 ,   56 ,
  5, Convolution       ,  1,   1 ,  1 ,  4 ,  x ,  x ,  x ,  x ,  x ,  x ,  x ,  5 ,    1 ,   64 ,   56 ,   56 ,    1 ,   64 ,   28 ,   28 ,
  6, Convolution       ,  1,   1 ,  1 ,  5 ,  x ,  x ,  x ,  x ,  x ,  x ,  x ,  6 ,    1 ,   64 ,   28 ,   28 ,    1 ,  128 ,   28 ,   28 ,
  7, Convolution       ,  1,   1 ,  1 ,  6 ,  x ,  x ,  x ,  x ,  x ,  x ,  x ,  7 ,    1 ,  128 ,   28 ,   28 ,    1 ,  128 ,   14 ,   14 ,
  8, Convolution       ,  1,   1 ,  1 ,  7 ,  x ,  x ,  x ,  x ,  x ,  x ,  x ,  8 ,    1 ,  128 ,   14 ,   14 ,    1 ,  256 ,   14 ,   14 ,
  9, Convolution       ,  1,   1 ,  1 ,  8 ,  x ,  x ,  x ,  x ,  x ,  x ,  x ,  9 ,    1 ,  256 ,   14 ,   14 ,    1 ,  256 ,    7 ,    7 ,
 10, Convolution       ,  1,   1 ,  1 ,  9 ,  x ,  x ,  x ,  x ,  x ,  x ,  x , 10 ,    1 ,  256 ,    7 ,    7 ,    1 ,  512 ,    7 ,    7 ,
 11, Convolution       ,  1,   1 ,  1 , 10 ,  x ,  x ,  x ,  x ,  x ,  x ,  x , 11 ,    1 ,  512 ,    7 ,    7 ,    1 ,  512 ,    7 ,    7 ,
 12, Pooling           ,  1,   1 ,  1 , 11 ,  x ,  x ,  x ,  x ,  x ,  x ,  x , 12 ,    1 ,  512 ,    7 ,    7 ,    1 ,    1 ,    1 ,  512 ,
 13, InnerProduct      ,  1,   1 ,  1 , 12 ,  x ,  x ,  x ,  x ,  x ,  x ,  x , 13 ,    1 ,    1 ,    1 ,  512 ,    1 ,    1 ,    1 , 1000 ,
 14, SoftMax           ,  1,   1 ,  1 , 13 ,  x ,  x ,  x ,  x ,  x ,  x ,  x , 14 ,    1 ,    1 ,    1 , 1000 ,    1 ,    1 ,    1 , 1000 ,
 15, Data              ,  0,   1 , -1 , 14 ,  x ,  x ,  x ,  x ,  x ,  x ,  x ,  0 ,    1 ,    1 ,    1 , 1000 ,    0 ,    0 ,    0 ,    0 ,
```

출력 파일은 jacinto11.dot 이며 이 파일은 예를 사용하여 Linux x86 에서 PNG 또는 PDF 파일로 변환 할 수 있다.

```
dot -Tpdf ./jacinto11.dot -o ./jacinto11.pdf
```

- **JDetNet 의 Layer group 을 가리키고 Graph Partioning 을 두 group 으로 나누어 EVE 최적 그룹 및 DSP 최적 그룹을 만드는 방법을 설명한다.**

3.14.3.9. Simulation Tool

우리는 PLSDK ARM 파일 시스템과 Linux x86 시뮬레이션 도구 모두에서 시뮬레이션 도구를 제공한다.
이것은 비트 정확한 시뮬레이션이므로 시뮬레이션 툴의 출력은
A5749 또는 AM57xx Target 의 출력과 동일할 것으로 예상된다.
이 도구는 편의 도구로만 사용하라(예: Target EVM 이 없는 설정의 Testing Model)

시뮬레이션 도구는 변환된 Model 정확도(FP32 vs 8 비트 구현)를 검증하는데도 사용할 수 있다.
더 많은 수의 코어를 활용하는 x86 에서 병렬로 실행될 수 있다(시뮬레이션 도구는 단일 스레드 구현임)
비트 정확한 시뮬레이션으로 인해 시뮬레이션 도구의 성능을 사용하여
Target Execution Time 을 예측할 수는 없지만 Model 정확성을 검증하는데 사용할 수 있다.

처리할 프레임 수, 입력 이미지 파일(하나 이상의 원시 이미지 포함),
입력 이미지 파일의 숫자 형식(부호 또는 부호 없음), 추적 폴더 및 모델 파일의 지정을 포함하는 구성 파일의 예:

```
rawImage     = 1
numFrames    = 1
inData    = "./tmp.raw"
inElementType       = 0
traceDumpBaseName    = "./out/trace_dump_"
outData    = "stats_tool_out.bin"
netBinFile       = "./tidl_net_imagenet_jacintonet11v2.bin"
paramsBinFile    = "./tidl_param_imagenet_jacintonet11v2.bin"
```

여러 이미지를 처리해야하는 경우 아래(또는 유사한) Script 를 사용할 수 있다.

여러 이미지를 처리해야하는 경우 아래(또는 유사한) Script 를 사용할 수 있다.

```
SRC_DIR=$1

echo "############################################################" >  TestResults.log
echo "Testing in $SRC_DIR" >> TestResults.log
echo "############################################################" >> TestResults.log
for filename in $SRC_DIR/*.png; do
     convert $filename -separate +channel -swap 0,2 -combine -colorspace sRGB ./sample_bgr.png
     convert ./sample_bgr.png  -interlace plane BGR:sample_img_256x256.raw
     ./eve_test_dl_algo.out sim.txt
     echo "$filename Results " >> TestResults.log
     hd stats_tool_out.bin | tee -a TestResults.log
done
```

시뮬레이션 도구 ./eve_test_dl_algo.out 은 single command line 인수로 호출된다:

```
./eve_test_dl_algo.out sim.txt
```

시뮬레이션 구성 파일에는 실행할 Network Model 목록이 포함된다(이 경우에는 하나 뿐임)
tidl_config_j11.txt 목록은 아래와 같이 종료됨: "0"

```
1 ./tidl_config_j11_v2.txt
0
```

시뮬레이션 도구에서 사용되는 샘플 구성 파일(tidl_config_j11_v2.txt)이다:

```
rawImage     = 1
numFrames    = 1
preProcType  = 0
inData    = "./sample_img_256x256.raw"
traceDumpBaseName    = "./out/trace_dump_"
outData    = "stats_tool_out.bin"
updateNetWithStats    = 0
netBinFile       = "./tidl_net_model.bin"
paramsBinFile    = "./tidl_param_model.bin"
```

SRC_DIR 의 모든 이미지에 대한 결과는 TestResults.log 로 보내지며 Caffe-Jacinto Desktop 실행에 대해 Test 할 수 있다.

3.14.3.10. Summary of model porting steps

- Desktop 프레임워크(Caffe 또는 TF)를 사용하여 Model 을 만든 후에는
  Model 의 정확성을 학인해야 한다(Caffe / Caffe-Jacinto 또는 TensorFlow).

- 위의 import 절차를 사용하여 최종 모델을 가져온다(Caffe-Jacinto 의 경우 "희소" 단계의 마지막에서)

- 시뮬레이션 도구를 사용하여 정확성을 확인하라(첫 번째 단계에서 사용 된 것보다 작은 Test 데이터 집합 사용)
  - 정확도 저하(첫 단계)는 커야 한다(몇 퍼센트)

- TIDL API 기반 프로그램 및 import 된 Model 을 사용하여 Target 에서 Network 를 Test 한다.

3.14.4. Training

Layer 가 지원되고 Parameter Constraints 가 충족되는 한 기존 Caffe 및 TF-Slim Model 을 가져 올 수 있다.
그러나 일반적으로 이러한 Model 에는 밀도가 높은 행렬이 포함된다.
TIDL Lib 의 장점을 활용하고 3x4x 성능 향상(Convolution Layer 용)을 얻으려면
http://github.com/tidsp/caffe-jacinto 에서
제공하는 caffe-jacinto caffe fork 를 사용하여 train 과정을 반복해야 한다.
Convolution Neural Network 의 연산 부하에서 가장 큰 기여는
Convolution Layer(종종 80-90% 범위)에서 발생하므로
Convolution Layer 처리를 최적화하는데 특별한 주의가 필요하다.

데이터 집합 준비는 일반적으로 LMDB 파일을 만드는 표준 Caffe 방식을 따라야 한다.
이후 해당 training 은 3 단계로 완료된다:

- 밀도 높은 Model 을 만드는 초기 Training(일반적으로 L2 정규화)
  이 단계는 실제로 Desktop 에 적용되는 일반적인 training 절차다.
  이 단계가 끝나면 Model 의 정확성을 검증해야 한다.
  가중치 텐서는 밀도가 높기 때문에 성능 목표를 달성할 수 없지만 다음 단계를 수행하면 성능을 향상시킬 수 있다.
  정확도가 충분하지 않으면 추가 단계를 진행하는 것이 좋다
  (정확도를 향상시키지 않는다. 실제로 1-2% 의 누적 발생이 예상됨)
  대신 정확도 목표가 충족 될 때까지 training parameter 를 수정하거나
  데이터 집합을 향상시키고 training 을 반복한다.

- L1 regularization
  이 단계는 (L2 와 반대) 다른 것들을 희생시켜
  특정 가중치를 선호하게 만드는데 필요하며 특정 가중치 보다 큰 것을 작게 만들어준다.
  나머지 가중친은 feature(특징) 추출기처럼 작동한다(다음 단계에서 필요함)

- Sparse(“sparsification”)
  무게 임계값을 점진적으로 조정하여 (작은 것부터 높은 것까지)
  희박 Target 을 각 단계에서(예:70% 혹은 80%) 테스트한다.
  이 절차는 작은 가중치를 제거하고 큰 기여자만 남긴다.
  이는 Convolution Layers 에만 적용된다.

- 정확도 저하에 따라 희박화에 대한 수용 가능한 기준을 정의함
  FP32 표현에서 가중치(및 8 비트 활성화)의 8-12 비트 표현으로 변환되기 때문에
  허용 가능한 정확도 저하는 Model 에 따라 1-2% 범위 내에 있어야 한다.(예: Model 에 따라 다름)
  Caffe-Jacinto Desktop Model 의 분류 정확도가 70%(초기 단계 이후 Model 사용)인 경우
  희박화 및 양자화 된 Model 의 정확도가 68% 미만으로 떨어지지 않아야 한다.

3.14.4.1. Example of training procedure

- 특정 작은 개체의 데이터 집합 수집을 위한 설정
  공개적으로 사용 가능한 많은 Image Data 집합을 제외하고는
  특정 사용 사례를 위해 새로운 데이터 집합을 수집해야 하는 경우가 종종 있다.
  예: 산업 환경에서 일반적으로 더 예측 가능하며 좋은 조명으로 통제 된 환경을 보장하는 것이 가능하다.
  Pick & Place 애플리케이션의 경우, Camera 시야에 나타날 수 있는
  객체 집합은 무한대가 아니라 수십 또는 수십 개의 클래스로 제한된다.
  조명이 좋은 탁자 및 사진 부스를 상요하면 빠른 데이터 집합 수집이 가능하다.

- AM57xx 를 사용한 데이터 집합 수집
  데이터 집합 이미지는 외부 카메라 장치 또는 카메라 도터 카드(AM57xx)를 사용하여 기록 할 수 있다.
  제안된 레코딩 포맷은 H264 로 품질이 좋고 GStreamer 파이프라인을 사용하여 효율적으로 디코딩 할 수 있다.
  15 - 20 초 동안만 지속 될 수 있고(턴 테이블 회전 주기) 느린 fps(10 – 15 fps) 에서는 200 – 300 프레임을 제공한다.
  절차는 거리와 고도(3 – 4 번)를 변경하여 반복 할 수 있으므로
  총 이미지 수는 클래스 당 최대 2000 ~ 3000 프레임이 될 수 있다.
  이는 단일 클래스 데이터 수집 시간을 5 - 10 분으로 제한 할 수 있다.

- Post-processing
  Offline Post-Processing 를 위해 Video Clip 을 Linux x86 으로 복사해야 한다.
  FFMPEG 패키지를 사용하면 Video 클립을 개별 이미지로 쉽게 분할 수 있다.
  균일한 배경에 대해 녹음하기 때문에 자동 라벨링 절차를 적용 할 수도 있다.
  이미지 확대 스크립트를 사용하여 추가 데이터 집합을 향상시킬 수 있어 이미지 수를 10 ~ 20 배 쉽게 늘릴 수 있다.

- LMDB 파일을 교육용으로 준비하라.
  https://github.com/tidsp/caffe-jacinto-models/tree/caffe-0.17/scripts 에서 사용 가능한 Script 를 참조하라.

- 처음부터 training 하거나 학습 전송을 수행함(미세 조정)
  종종 ImageNet 과 같은 일반 데이터 집합으로 작성된 초기 가중치를 사용하여 training 을 시작하는 것이 좋다.
  하위 Layer 는 Feature 추출기와 같은 역할을 하며 방금 수집한 데이터 집합(이전 집합에서 설명한대로)를 사용하여
  상위 1 개 또는 몇 개의 Layer 만 미세 조정할 필요가 있다.
  Jacinto11 의 경우, 좋은 출발점은 "초기" 단계 후에 만들어진 Model 이다.
  초기 단계를 반복해야하지만 새로운 데이터 집합을 사용하고
  이전 Model 로 미리 로드하려는 Layer 에 동일한 Layer 이름을 사용해야 한다.
  또한 base_lr(train.prototxt 에서)을 줄이고 맨 위 또는 몇 개의 Layer 에서 lr 을 늘림으로써 training 을 조정할 수 있다.

3.14.4.2. Where does the benefit of sparsification come from?

- 처음에는 Deep Learning Network 가 Single Precision Floating Point Arithmetic(FP32)을 사용하여 구현되었다.
  지난 몇 년 동안 양자화 영향 및 산술 연산의 정확도 감소에 대한 더 많은 연구가 있었다.
  대부분의 경우 올바른 동작을 위해서는 8 비트 또는 그 이하(2 - 4 비트까지)가 충분하다고 여겨진다.
  이것은 모두 작동 정확도에 기여하는 많은 매개 변수(가중치)로 설명된다.
  DSP 및 EVE 추론 구현의 경우,
  가중치(import tool configuration file 의 매개 변수로 제어됨)를 8 - 12 비트 정확도로 양자화 할 수 있다.
  Activation Layer 출력(뉴런 출력)은 8 비트 정확도(1 바이트)로 메모리에 저장된다.
  누산은 40 비트 정확도로 수행되지만 최종 출력은 1 바이트가 메모리에 저장되기 전에 오른쪽으로 쉬프트 된다.
  오른쪽 쉬프트 수는 각 Layer 및 Frame 당 한 번씩 고유하게 동적으로 결정된다.
  자세한 내용은
  http://openaccess.thecvf.com/content_cvpr_2017_workshops/w4/papers/Mathew_Sparse_Quantized_Full_CVPR_2017_paper.pdf 에서 확인할 수 있다.

- 추가 최적화(위의 article 에서 설명)는 Convolution Layer 가중치의 희소성에 기반한다.
  개인 가중치는 훈련 중 0 으로 강제 된다.
  이것은 "L1 reglarization" 단계(다른 사람을 희생하여 더 적은 가중치를 적용하는 것)와
  작은 가중치를 0 으로 Clamp 할 때 "Sparse" 단계에서 달성된다.
  원하는 학습 목표(예: 모든 가중치의 70% 또는 80%)를 지정할 수 있다.
  추론 중에 계산이 다시 구성되어 단일 가중치 매개 변수를 통한 곱셈이 모든 입력 값에서 수행된다.
  가중치가 0 이면 모든 입력 데이터(해당 입력 채널에 대한)에 대한 곱셈은 건너 뛴다.
  모든 계산은 미리로드 된 Block 을 Local L2 메모리에 사용하여 수행된다("Shadow" EDMA 전송 사용)

3.14.5. Performance data:

3.14.5.1. Computation performance of verified networks

- **J11, JSEg21, JDetNet, Mobilenet, SqueezeNet**

Network topology	ROI size	MMAC (million MAC)	Sparsity (%)	EVE using sparse model
MobileNet	224x224	567.70	1.42	•
SqueezeNet	227x227	390.8	1.46	•
InceptionNetV1	224x224	1497.37	2.48	•
JacintoNet11	224x224	405.81	73.15	125.9ms
JSegNet21	1024x512	8506.5	76.47	378.18ms
JDetNet	768x320	2191.44	61.84	•

EVE using dense model	DSP using sparse model	DSP using dense model	EVE + DSP (optimal model)
682.63ms	•	717.11ms	•
289.76ms	•	1008.92ms	•
785.43ms	•	2235.99ms	•
235.70ms	115.91ms	370.64ms	73.55ms
1236.84ms	1101.12ms	3825.95ms	•
•	•	•	197.55ms

- 위 표에 제공된 희박성은 모든 Convolution Layers 에서 평균 희박성을 가지고 있다.
- 최적 모델 - EVE 와 DSP 사이에 최적의 Layer 배치(특정 NN Layer 에는 SoftMax 와 같이 DSP 에서 더 빠르게 실행되며 EVE 에서는 ARP32 가 SW 에서 부동 연산을 에뮬레이트하므로 다소 느릴 수 있음)
- 다음 release 에서는 최적의 Layer 배치(EVE 가 느린 SoftMax Layer 구현을 가짐)를 사용하여 성능이 향상된다. 예를 들어 AM5749 기준으로 Jacinto11 ~ 28-30 fps 를 달성할 수 있다.

3.14.5.2. Accuracy of selected networks

아래 표는 편의를 위해 https://github.com/tidsp/caffe-jacinto-models 문서에서 복사된다.

- Image classification:
  Top-1 분류 정확도는 진실도가 가장 높은 확률을 나타낸다.
  Top-5 분류 정확도는 진실도가 5 순위 후보 중 하나일 확률을 나타낸다.

Configuration-Dataset Imagenet (1000 classes)	Top-1 accuracy
JacintoNet11 non-sparse	60.9%
JacintoNet11 layerwise threshold sparse (80%)	57.3%
JacintoNet11 channelwise threshold sparse (80%)	59.7%

- Image segmentation:
  Union 에 대한 평균 교집합은 True Positives 와
  False Positives, False Negatives 및 True Positives 의 합의 비율이다.

Configuration-Dataset Cityscapes (5-classes)	Pixel accuracy	Mean IOU
Initial L2 regularized training	96.20%	83.23%
L1 regularized training	96.32%	83.94%
Sparse fine tuned (~80% zero coefficients)	96.11%	82.85%
Sparse (80%), Quantized (8-bit dynamic fixed point)	95.91%	82.15%

- Object Detection:
  유효성 검사 정확도는 분류 정확도 또는 Mean Average Precision(map)에 있을 수 있다.
  "Initial"(dense) 및 "Sparse" Model 간에 정확도가 변경되었음을 유의하라(성능 향상은 2x-4x 일 수 있음)

Configuration-Dataset VOC0712	mAP
Initial L2 regularized training	68.66%
L1 regularized fine tuning	68.07%
Sparse fine tuned (~61% zero coefficients)	65.77%

3.14.6. Troubleshooting

- OpenCL Stack 이 실행되는지 Linux 부팅시 확인하고 OpenCL Firmware 는 DSP 와 EVE 에 다운로드 된다.
  EVE 를 제어하는 IPU1 용 OpenCL Monitor 가 새로 추가되었으므로 여기에 예상되는 trace 가 있다.
  Target 에 아래 명령을 입력하라.
  cat /sys/kernel/debug/remoteproc/remoteproc0/trace0
  사용 가능한 EVE 가속기 수를 나타내는 아래 출력이 예상된다(AM5729 trace 에서 4 EVE 를 나타낸다)

```
[0][      0.000] 17 Resource entries at 0x3000
[0][      0.000] [t=0x000aa3b3] xdc.runtime.Main: 4 EVEs Available
[0][      0.000] [t=0x000e54bf] xdc.runtime.Main: Creating msg queue...
[0][      0.000] [t=0x000fb885] xdc.runtime.Main: OCL:EVEProxy:MsgQ ready
[0][      0.000] [t=0x0010a1a1] xdc.runtime.Main: Heap for EVE ready
[0][      0.000] [t=0x00116903] xdc.runtime.Main: Booting EVEs...
[0][      0.000] [t=0x00abf9a9] xdc.runtime.Main: Starting BIOS...
[0][      0.000] registering rpmsg-proto:rpmsg-proto service on 61 with HOST
[0][      0.000] [t=0x00b23903] xdc.runtime.Main: Attaching to EVEs...
[0][      0.007] [t=0x00bdf757] xdc.runtime.Main: EVE1 attached
[0][      0.010] [t=0x00c7eff5] xdc.runtime.Main: EVE2 attached
[0][      0.013] [t=0x00d1b41d] xdc.runtime.Main: EVE3 attached
[0][      0.016] [t=0x00db9675] xdc.runtime.Main: EVE4 attached
[0][      0.016] [t=0x00dc967f] xdc.runtime.Main: Opening MsgQ on EVEs...
[0][      1.017] [t=0x013b958a] xdc.runtime.Main: OCL:EVE1:MsgQ opened
[0][      2.019] [t=0x019ae01a] xdc.runtime.Main: OCL:EVE2:MsgQ opened
[0][      3.022] [t=0x01fa62bf] xdc.runtime.Main: OCL:EVE3:MsgQ opened
[0][      4.026] [t=0x025a4a1f] xdc.runtime.Main: OCL:EVE4:MsgQ opened
[0][      4.026] [t=0x025b4143] xdc.runtime.Main: Pre-allocating msgs to EVEs...
[0][      4.027] [t=0x0260edc5] xdc.runtime.Main: Done OpenCL runtime initialization. Waiting for messages...
```

- **CMEM 이 활성 상태이며 실행 중인지 확인하라:**
  – cat /proc/cmem
  – lsmod | grep "cmem"
  – 2 개 이상의 EVE 가 있는 장치의 경우 기본 CMEM 크기가 충분하지 않다(EVE 당 약 56 ~ 64 MB 사용 가능)

- **Model 준비 절차 검증**

- **Import 절차가 외부 Model import 에 실패하면 충분한 정보를 제공하지 못할 수 있다.**
  예: Format 이 인식되지 않으면 아래 report 가 표시 될 수 있다(이 경우 Keras Model import 시도가 이루어짐)

```
$ ./tidl_model_import.out ./modelInput/tidl_import_mymodel.txt
TF Model File : ./modelInput/mymodel
Num of Layer Detected :   0
Total Giga Macs : 0.0000

Processing config file ./tempDir/qunat_stats_config.txt !
  0, TIDL_DataLayer               ,  0,   0 ,  0 ,  x ,  x ,  x ,  x ,  x ,  x ,  x ,  x ,  0 ,   0 ,   0 ,   0 ,   0 ,   0 ,   0 ,   0 ,   0 ,

Processing Frame Number : 0

End of config list found !
```

- **데이터 집합 준비 문제**
  – training 집합 준비하는 동안 좋은 조명이 매우 바람직함
  – 증가

- **Desktop Caffe 실행과 Target 실행 간의 동등성**
  이를 위해 Simulation Tool 을 EVE 또는 DSP 실행으로 비트 정밀도를 높여 사용할 수 있다.
  Simulation Tool 에 의해 생성 된 Trace 는 Desktop Caffe 추론 후에
  저장되는 데이터 Blob 과 시각적으로 비교 될 수 있다.
  나머지가 모두 맞으면 중간 결과를 비교해 볼 가치가 있다.
  Caffe Desktop 계산(단 정밀도 FP32 사용)과
  Target 계산(8 비트 활성화 및 8 - 12 비트 가중치 사용) 사이의 수치 동등성은 예상되지 않는다.
  여전히 기능적 인지도(중간 계층의 지도)는 다소 비슷하다.
  무엇인가가 크게 다른 경우 가중치 비트 수를 변경하거나 더 많은 대표 이미지로 가져오기 처리를 반복하라.
  이런 종류의 문제는 거의 발생하지 않아야 한다.

- **Run-Time 의 일반적인 오류(플랫폼 reboot 시점)**

```
... inc/executor.h:199: T* tidl::malloc_ddr(size_t) [with T = char; size_t = unsigned int]: Assertion `val != nullptr' failed.
This means that previous run failed to de-allocate CMEM memory. Reboot is one option, restarting ti-mctd deamon is another option.
```