TI Deep learning Library on TDAx

User Guide

December 2018

Products		**Applications**	
Audio	www.ti.com/audio	Automotive & Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications & Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers & Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energyapps
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics & Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video & Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Applications Processors	www.ti.com/omap	**TI E2E Community**	e2e.ti.com
Wireless Connectivity	www.ti.com/wirelessconnectivity		

Table of Contents

IMPORTANT NOTICE 1-2

1 READ THIS FIRST 1-5

1.1 About This Manual 1-5

1.2 Intended Audience 1-5

1.3 How to Use This Manual 1-5

1.4 Related Documentation From Texas Instruments 1-5

1.5 Abbreviations 1-5

1.6 Text Conventions 1-6

1.7 Product Support 1-6

1.8 Trademarks 1-6

2 INTRODUCTION 2-1

2.1 Overview of XDAIS 2-1

2.1.1 XDAIS Overview 2-1

2.2 Overview of TI Deep Learning Library 2-2

3 INSTALLATION OVERVIEW 3-4

3.1 System Requirements 3-4

3.1.1 Hardware 3-4

3.1.2 Software 3-4

3.2 Installing the Component 3-4

3.2.1 Installing the compressed archive 3-4

3.3 Building Sample Test Application 3-5

3.3.1 Installing XDAIS tools (XDAIS) 3-6

3.3.2 Installing Code Generation Tools 3-6

3.3.3 DMA utility Library 3-6

3.3.4 Building the Test Application Executable through GMAKE 3-6

3.4 Configuration File 3-7

3.4.1 Test Application Configuration File 3-7

3.5 Host emulation build for source package 3-8

3.5.1 Installing Visual Studio 3-8
3.5.2 Building source in host emulation 3-8

3.6 Running a Model trained by user 3-9
3.6.1 Building Model import tool 3-9
3.6.2 Importing the Model and Parameters 3-9
3.6.3 Building TIDL reference executable 3-11
3.6.4 Importing Caffe-Jacinto-Models 3-12
3.6.5 Importing Tensorflow Models 3-13

3.7 Input and Output Data Formats 3-13

3.8 Matching TIDL inference result 3-14

3.9 TIDL Limitation 3-15

3.10 Uninstalling the Component 3-17

4 SAMPLE USAGE 4-18

4.1 Overview of the Test Application 4-18

4.2 Algorithm Instance Creation and Initialization 4-19

4.3 Process Call 4-19

4.4 Algorithm Instance Deletion 4-20

4.5 Frame Buffer Management 4-20
4.5.1 Input and Output Frame Buffer 4-20

5 API REFERENCE 5-21

6 FAQ 6-21

1 Read This First

1.1 About This Manual

This document describes how to install and work with Texas Instruments' (TI) TI Deep learning Library Module implemented on TI's TMS320C66x DSP and EVE. It also provides a detailed Application Programming Interface (API) reference and information on the sample application that accompanies this component.

TI Deep learning Library Module implementations are based on IVISION interface. IVISION interface is an extension of the eXpressDSP Algorithm Interface Standard (XDAIS).

1.2 Intended Audience

This document is intended for system engineers who want to integrate TI's vision and imaging algorithms with other software to build a high level vision system based on C66x DSP or EVE.

This document assumes that you are fluent in the C language, and aware of vision and image processing applications. Good knowledge of eXpressDSP Algorithm Interface Standard (XDAIS) standard will be helpful.

1.3 How to Use This Manual

This document includes the following chapters:

Chapter 2 - Introduction, provides a brief introduction to the XDAIS standards. It also provides an overview of TI Deep learning Library Module and lists its supported features.

Chapter 3 - Installation Overview, describes how to install, build, and run the algorithm.

Chapter 4 - Sample Usage, describes the sample usage of the algorithm.

Chapter 5 - API Reference, describes the data structures and interface functions used in the algorithm.

Chapter 6 - Frequently Asked Questions, provides answers to frequently asked questions related to using TI Deep learning Library Module.

1.4 Related Documentation From Texas Instruments

This document frequently refers TI's DSP algorithm standards called XDAIS. To obtain a copy of document related to any of these standards, visit the Texas Instruments website at www.ti.com.

1.5 Abbreviations

The following abbreviations are used in this document.

Table 1 List of Abbreviations

Abbreviation	**Description**
API	Application Programming Interface

Abbreviation	Description
DMA	Direct Memory Access
DSP	Digital Signal Processing
EVM	Evaluation Module
EVE	Embedded Vision Engine
XDAIS	eXpressDSP Algorithm Interface Standard

1.6 Text Conventions

The following conventions are used in this document:

Text inside back-quotes (‘‘) represents pseudo-code.

Program source code, function and macro names, parameters, and command line commands are shown in a `mono-spaced` font.

1.7 Product Support

When contacting TI for support on this product, quote the product name (TI Deep learning Library Module on TMS320C66x DSP and EVE) and version number. The version number of the TI Deep learning Library Module is included in the Title of the Release Notes that accompanies the product release.

1.8 Trademarks

Code Composer Studio, eXpressDSP, TI Deep learning Library Module are trademarks of Texas Instruments.

2 Introduction

This chapter provides a brief introduction to XDAIS. It also provides an overview of TI's implementation of TI Deep learning Library Module on the C66x DSP 7 EVE and its supported features.

2.1 Overview of XDAIS

TI's vision analytics applications are based on IVISION interface. IVISION is an extension of the eXpressDSP Algorithm Interface Standard (XDAIS). Please refer documents related to XDAIS for further details.

2.1.1 XDAIS Overview

An eXpressDSP-compliant algorithm is a module that implements the abstract interface IALG. The IALG API takes the memory management function away from the algorithm and places it in the hosting framework. Thus, an interaction occurs between the algorithm and the framework. This interaction allows the client application to allocate memory for the algorithm and also share memory between algorithms. It also allows the memory to be moved around while an algorithm is operating in the system. In order to facilitate these functionalities, the IALG interface defines the following APIs:

`algAlloc()`

`algInit()`

`algActivate()`

`algDeactivate()`

`algFree()`

The `algAlloc()` API allows the algorithm to communicate its memory requirements to the client application. The `algInit()` API allows the algorithm to initialize the memory allocated by the client application. The `algFree()` API allows the algorithm to communicate the memory to be freed when an instance is no longer required.

Once an algorithm instance object is created, it can be used to process data in real-time. The `algActivate()` API provides a notification to the algorithm instance that one or more algorithm processing methods is about to be run zero or more times in succession. After the processing methods have been run, the client application calls the `algDeactivate()` API prior to reusing any of the instance's scratch memory.

The IALG interface also defines three more optional APIs `algControl()`, `algNumAlloc()`, and `algMoved()`. For more details on these APIs, see *TMS320 DSP Algorithm Standard API Reference* (literature number SPRU360).

2.2 Overview of TI Deep Learning Library

- Convolution neural network (CNN) based Machine learning algorithms are used in many ADAS applications and self-driving cars. These algorithms are defined in the form of network structure with thousands of parameters. These algorithms needs to be accelerated on TI devices (with DSP and EVE cores) without much effort from the algorithm developer/customer
- **Interoperability** : There are many tools/frameworks are available in PC for algorithm development (Training and fine tuning). We need to have a solution to support most of these popular frameworks in TI devices with optimal resource utilization.
- **High Compute** : The MAC requirements of these CNN networks are in the range of 500 Giga MACs to 10 Tera MACs for 1 MP image real-time processing. TI TDA2x+ devices can deliver up to 70 GMACs for CNN algorithms.
- **High Memory Bandwidth** : CNN networks normally consists of multiple layers (20 to 100s), each of these layers have multiple channels (around 4 Mbytes output from each layer). As we understand the embedded devises have limited memory bandwidth and it plays a important role in device compute utilization.
  - **Scalability** :
    - API : The algorithms (layers) used in these CNN networks are evolving, there are many layers are getting defined in continuously. The interface and data flow shall be scalable to add any new layers.
    - Performance: TI devices have multiple compute cores (EVE, DSP, ARM) in each SoC. The solutions shall be scalable to utilize these core simultaneously without much effort from users

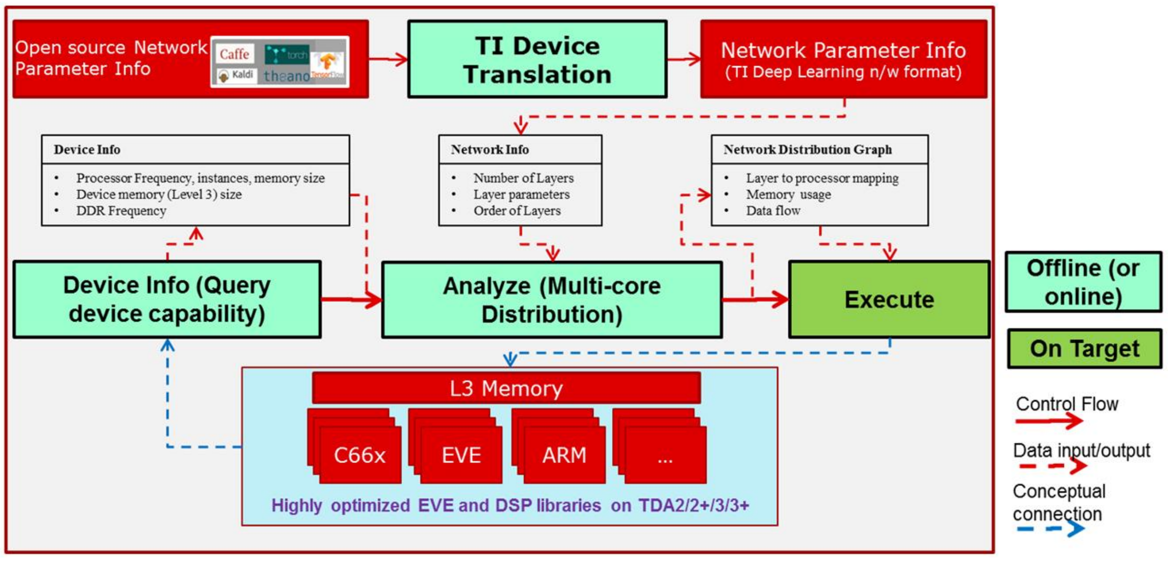

Open source Network Parameter Info
Caffe
torch
Kaldi
theano
TI Device Translation
Network Parameter Info
(TI Deep Learning n/w format)
Device Info
Processor Frequency, instances, memory size
Device memory (Level 3) size
DDR Frequency
Network Info
Number of Layers
Layer parameters
Order of Layers
Network Distribution Graph
Layer to processor mapping
Memory usage
Data flow
Device Info (Query device capability)
Analyze (Multi-core Distribution)
Execute
Offline (or online)
On Target
L3 Memory
C66x
EVE
ARM
...
Highly optimized EVE and DSP libraries on TDA2/2+/3/3+
Control Flow
Data input/output
Conceptual connection

Figure 1 Fundamental blocks of TI Deep Learning Library

The TI Deep Learning Library addresses the above mentioned problems with following solutions

- Software Architecture to reuse the same data flow between EVE and DSP, only compute kernels are different

- Focus on exploiting sparse coefficients and effectively higher MACs
- Extendable API and software architecture to allow integration of newer layers
- Appropriate usage of memory hierarchy and data flow to optimally utilize the memory bandwidth at DDR and OCMC

3 Installation Overview

This chapter provides a brief description on the system requirements and instructions for installing TI Deep learning Library Module. It also provides information on building and running the sample test application.

3.1 System Requirements

This section describes the hardware and software requirements for the normal functioning of the algorithm component.

3.1.1 Hardware

This algorithm has been built and tested TI's C66x DSP and EVE on TDA2x and TDA3x platform. The algorithm shall work on any future TDA platforms hosting C66x DSP or EVE.

3.1.2 Software

The following are the software requirements for the stand alone functioning of the TI Deep learning Library Module:

Development Environment: This project is developed using TI's DSP Code Generation Tool 7.4.2 and EVE Code Generation Tool 1.0.7. Other required tools used in development are mentioned in section 3.3

The project are built using g-make (GNU Make version 3.81). GNU tools comes along with CCS installation.

DMA utility Library Version 00.08.00.02 for programming EDMA

3.2 Installing the Component

The algorithm component is released as install executable. Following sub sections provided details on installation along with directory structure.

3.2.1 Installing the compressed archive

The algorithm component is released as a compressed archive. To install the algorithm, extract the contents of the zip file onto your local hard disk.

Figure 2 Component Directory Structure

Table 2 Component Directories

Sub-Directory	Description
\common	Common files for building different modules
\makerules	Make rules files
\modules	Top level folder containing different app modules
\modules\ti_dl	TI Deep learning Library module for C66x DSP and EVE
\modules \ti_dl\docs	User guide and Datasheet for TI Deep learning Library module
\modules\ti_dl\inc	Contains itidl_ti.h interface file
\modules\ti_dl\lib	Contains TI Deep learning Library
\modules\ti_dl\test	Contains standalone test application source files
\modules\ti_dl\test\out	Contains test application .out executable
\modules\ti_dl\test\src	Contains test application source files
\modules\ti_dl\test\testvecs	Contains config, input, output, reference test vectors
\modules\ti_dl \test\testvecs\config	Contain config file to set various parameters exposed by TI Deep learning Library module
\modules\ti_dl \test\testvecs\input	Contains sample input feature vector .bin file
\modules\ti_dl \test\testvecs\output	Contains output file
\modules\ti_dl \test\testvecs\reference	Contains reference file
\modules\ti_dl\utils	Contains the source code and binary of the PC tool used importing the trained models and their parameters to the TIDL

3.3 Building Sample Test Application

This TI Deep learning Library has been accompanied by a sample test application. To run the sample test application XDAIS tools are required.

This version of the TI Deep learning Library has been validated with XDAIS tools containing IVISION interface version. Other required components (for test application building) version details are provided below.

- DSP Code Generation Tool version 7.4.2
- EVE/ARP32 Code Generation Tool version 1.0.7

- XDAIS version 7.22.00.03
- DMA utility Library Version 00.08.00.02

3.3.1 Installing XDAIS tools (XDAIS)

XDAIS version 7.24 can be downloaded from the following website:

http://downloads.ti.com/dsps/dsps_public_sw/sdo_sb/targetcontent/xdais/

Extract the XDAIS zip file to the same location where Code Composer Studio has been installed. For example:

C:\CCStudio5.0

Set a system environment variable named "XDAIS_PATH" pointing to <install directory>\<xdais_directory>

3.3.2 Installing Code Generation Tools

Install EVE and DSP Code generation Tools from the link

https://www-a.ti.com/downloads/sds_support/TICodegenerationTools/download.htm

After installing the CG tools, set the environment variable to "DSP_TOOLS" and "ARP32_TOOLS" to corresponding the installed directory like <install directory>\<cgtools_directory>

3.3.3 DMA utility Library

Install DMA utility library for EVE and DSP from the link

https://cdds.ext.ti.com/ematrix/common/emxNavigator.jsp?objectId=28670.42872.62652.37497

After installing the DMA Utility Library, Set a system environment variable named "DMAUTILS_PATH" pointing to <install directory>\ dmautils

3.3.4 Building the Test Application Executable through GMAKE

The sample test application that accompanies TIDL module will run in TI's Code Composer Studio development environment. To build and run the sample test application through gmake, follow these steps:

1) Verify that you have installed code generation tools as mentioned.
2) Verify that you have installed XDAIS as mentioned
3) Verify that appropriate environment variables have been set as discussed in this above sections.
    a. Environment variables can be either directly set by updating the system environment variables or by editing the config.mk file in makerules folder
4) Build the sample test application project

a. For DSP

   modules\ti_dl\test> gmake CORE=dsp all

b. For EVE

   modules\ti_dl\test> gmake CORE=eve all

5) The above step creates executable files, “dsp_test_dl_algo.out” and “eve_test_dl_algo.out” in the modules\ti_dl\test\out sub-directory.

6) Open CCS with TDA2x platform selected configuration file. Select Target > Load Program on C66x DSP/EVE, browse to the modules\ti_dl\test\out sub-directory, select the executable created in step 5, and load it into Code Composer Studio in preparation for execution.

7) Select Target > Run on C66x DSP/EVE window to execute the sample test application.

8) Sample test application takes the input files stored in the \test\testvecs\input sub-directory, runs the module.

9) The reference files stored in the \test\testvecs\reference sub-directory can be used to verify that the TIDL is functioning as expected..

10) User should compare with the reference provided in \test\testvecs\reference directory. Both the content should be same to conclude successful execution.

3.4 Configuration File

This algorithm is shipped along with:

tidl_conv_seg_net.cfg – specifies the configuration parameters used by the test application to configure the TIDL for semantic segmentation algorithm.

3.4.1 Test Application Configuration File

A sample tidl_conv_seg_net.cfg file is as shown.

```
#------------------------------------------------------------------#
# Common Parameters                                                #
#------------------------------------------------------------------#
inData  =  "../testvecs/input/000100_1024x512_bgr.y" # Input BGR file

traceDumpBaseName =  "../testvecs/output/trace_dump_" # Base filename for trace dumps

outData =  "../testvecs/output/seg_out.bin" # Output filepath

netBinFile = "../testvecs/config/tidl_net_seg.bin" #Network model bin file generated
```

```
by Import tool

paramsBinFile = "../testvecs/config/tidl_pamas_seg.bin" #Network Parameters bin
generated by Import tool
```

3.5 *Host emulation build for source package*

The source release of TIDL module can be built in host emulation mode. This option speeds up development and validation time by running the platform code on x86/x64 PC.

3.5.1 Installing Visual Studio

Building host emulation for TI Deep Learning Library module requires Microsoft Visual Studio 11.0 (2012) or later

3.5.2 Building source in host emulation

After installing the required components, navigate to TI Deep Learning Library install path and run `vcvarsall.bat` to setup the required environment variables

```
{TIDL_install_path} > {…\Microsoft Visual Studio
11.0\VC\vcvarsall.bat}
```

Once the environment variables are setup build the TI Deep Learning Library source in host emulation mode

```
{TIDL_install_path} > gmake CORE=eve TARGET_BUILD=debug
TARGET_PLATFORM=PC all
```

This will build the host emulation executable under the path

```
{TIDL_install_path}\test\out\eve_test_dl_algo.out.exe
```

```
TIDL_install_path} > gmake CORE=dsp TARGET_BUILD=debug
TARGET_PLATFORM=PC all
```

This will build the host emulation executable under the path

```
{TIDL_install_path}\test\out\dsp_test_dl_algo.out.exe
```

To build the example in host emulation mode for c6x DSP you will need to install the c6xsim which is available at

http://processors.wiki.ti.com/index.php/Run_Intrinsics_Code_Anywhere

Install this package in the common folder

If you observe below error while building

```
linkage specification contradicts earlier specification for '_ti_rotl
```

Update below line in “c6xsim\C6xSimulator.h”

```
uint32 _rotl(uint32 a, int32 b);
```

to

```
uint32 _rotl(uint32 a, uint32 b);
```

Build with below setting (in “tidl_alg_int.h” file) to build the code to run the inference faster on emulation mode on PC.

```
#define ENABLE_TRACE_PROFILE      (0)

#define ENABLE_REF_COMPARISION    (0)

#define ENABLE_CN_CODE            (1)

#define ENABLE_PRINTFS            (0)
```

Note: The target code has to be built at least once before building HOST emulation build. The target build generates few header files which would be needed for HOST emulation build

3.6 Running a Model trained by user

3.6.1 Building Model import tool

The TIDL release package contains pre-built import tool for below training frame works.

- Caffe
- TensorFlow

This tool is built using the protobuf library version 3.2.0rc2.

The above mentioned module can be downloaded from below link.

https://developers.google.com/protocol-buffers/docs/downloads

This tool also can be re-built using visual studio complier with below command. Please refer the *makefile* in the `{TIDL_install_path}\utils\tidlModelImport` folder and update the ptotobuf module install paths.

```
        Gmake
```

Note : All the pre-built windows executable are built with Microsoft visual studio V11.0 (2012). Please install Visual C++ Redistributable for Visual Studio 2012 to run these on your windows machine.
https://www.microsoft.com/en-in/download/details.aspx?id=30679

3.6.2 Importing the Model and Parameters

We provide a PC tool to import the Model and Parameters trained using either caffe frame work or tensor flow frame work in PC. This PC tool will accept various parameters through import configuration file and generate the Model and Parameter file that the code will be executed using TIDL library across multiple EVE and DSP cores. The source code to the tool is also provided in the package for user to build it on different platform (Uinix/Linux) or extend it for their use case.

The import configuration file is available in `{TIDL_install_path}\test\testvecs\config\import`

Binary Usage:

```
EXE "Import configuration File Name"
```

```
Example:
```

```
>tidl_model_import.out.exe ..\..\test\testvecs\config\import\tidl_import_jseg21.txt
```

The list of import configuration parameters is as below:

Parameter	Comment
`randParams`	It can be either 0 or 1. Default value is 0. If it is set to 0, the tool will generate the quantization parameters from model, otherwise it will generate random quantization parameters.
`modelType`	It can be either 0 or 1. Default value is 0. Set this to 0 to read from caffeImport frame work, and set it to 1 to read from tensor flow frame work.
`quantizationStyle`	It can be '0' for fixed quantization by the training framework or '1' for dynamic quantization by. Default value is 1. Currently, only dynamic quantization is supported.
`quantRoundAdd`	It can take any value from 0 to 100. Default value is 50. quantRoundAdd/100 will be added while rounding to integer
`numParamBits`	It can take values from 4 to 12. Default value is 8. This is the number of bits used to quantize the parameters.
`preProcType`	It can take values from 0 to 5. Default value is 0. Refer to "tidl_image_preproc.c" file in the test folder for more details
`Conv2dKernelType`	It can be either 0 or 1 for each layer. Default value is 0 for all the layers. Set it to 0 to use sparse convolution, otherwise, set it to 1 to use dense convolution. Refer to FAQ 21 for more details
`inElementType`	It can be either 0 or 1. Default value is 1. Set it to 0 for 8-bit unsigned input or to 1 for 8-bit signed input.
`inQuantFactor`	It can take values >0. Default value is -1.
`rawSampleInData`	It can be either 0 or 1. Default value is 0. Set it to 0, if the input data is encoded (and set preProcType properly), or set it to 1, if the input is RAW data (and preProcType is ignored).
`numSampleInData`	It can be > 0. Default value is 1.
`foldBnInConv2D`	It can be either 0 or 1. Default value is 1.
`inWidth`	Width of the input image, it can be >0.
`inHeight`	Height of the input image, it can be >0.
`inNumChannels`	input number of channels. It can be from 1 to 1024.

sampleInData	Input data File name.
tidlStatsTool	TIDL reference executable.
inputNetFile	Input net file name (From Training frame work)
inputParamsFile	Input Params file name (From Training frame work)
outputNetFile	Output Model net file name, to be updated with stats.
outputParamsFile	Output Params file name.
layersGroupId	Group of layers that needs to be processed on a given CORE. Refer FAQ 21 for more details

Example (tidl_import_jseg21.txt): Ignored parameters will be set to the default values.

```
randParams         = 0

modelType          = 0

quantizationStyle  = 1

quantRoundAdd      = 25

numParamBits       = 8

inElementType      = 0

inputNetFile       =
..\..\test\testvecs\config\caffe_models\tiscapes_jseg21\jacintonet11.prototxt

inputParamsFile    =
..\..\test\testvecs\config\caffe_models\tiscapes_jseg21\jacintonet11.caffemodel

outputNetFile      = "..\..\test\testvecs\config\tidl_models\tidl_net_jsegnet21v2.bin"

outputParamsFile   = "..\..\test\testvecs\config\tidl_models\tidl_param_jsegnet21v2.bin"

rawSampleInData = 1

sampleInData = "..\..\test\testvecs\input\000100_1024x512_bgr.y"

tidlStatsTool = "..\quantStatsTool\eve_test_dl_algo.out.exe"
```

3.6.3 Building TIDL reference executable

The tidlStatsTool can be built using the TIDL source code. Use below command to build this tool from {TIDL_install_path} folder

```
gmake TARGET_PLATFORM=PC TARGET_BUILD=release CORE=eve RUN_REF_FOR_STATS=1 all
```

Use below commands to build with openCv for reading compressed images. Update makefile in {TIDL_install_path}\test for your openCV install directory

```
gmake TARGET_PLATFORM=PC TARGET_BUILD=release CORE=eve RUN_REF_FOR_STATS=1
BUILD_WITH_OPENCV=1 all
```

After completion of build copy the executable to
`{TIDL_install_path}\utils\quantStatsTool`

3.6.4 Importing Caffe-Jacinto-Models

The caffe-models trained using Caffe-Jacinto framework is available in the below GitHub repository.

https://github.com/tidsp/caffe-jacinto-models.

Please follow the below steps to import and run inference using these models on TIDL.

1. Download/Clone the caffe-jacinto-models from the GitHub
2. Copy the downloaded repo under the {TIDL_install_path}\test\config\caffe_jacinto_models folder. Below two folders shall be available now
   - {TIDL_install_path}\test\testvecs\config\caffe_jacinto_models\trained\image_classification
   - {TIDL_install_path}\test\testvecs\config\caffe_jacinto_models\trained\image_segmentation
3. Now run the “importTestCases.bat” from the below path.
   a. {TIDL_install_path}\utils\tidlModelImport
4. On successful execution of above script, imported net and param files shall be available. This scripts generated trace files also in the current directory.
   a. {TIDL_install_path}\test\testvecs\config\tidl_models\tidl_net_*.bin
   b. {TIDL_install_path}\test\testvecs\config\tidl_models\tidl_param_*.bin
5. These imported models can be executed using the configurations files available in the below path
   a. {TIDL_install_path}\test\testvecs\config\infer
6. To run TIDL in Host emulation mode, Build the source code in host emulation mode as explained in the earlier build section. Run the “runFuncTests.bat” script available in the below path. This also generates traces and output in corresponding test case folders. These trace files and *_out.bin file can be compared with the traces generated by import tool for correctness. Update the DSP/EVE executable name in the script before execution
   a. {TIDL_install_path}\utils\hostEmulationTest
7. To run TIDL on target EVM, Build the source code for target mode as explained in the earlier build section. Update the “{TIDL_install_path}\test\testvecs\config config_list.txt” with the test case that needs to be executed. Load and run the test application out file using the

CCS. The generated *_out.bin file can be compared with the traces generated by import tool for correctness.

3.6.5 Importing Tensorflow Models

TIDL supports slim based tensorflow models. We have used Checkpoint from below to validation. We have validated Inception V1 and MobileNet_v1_1.0_224

https://github.com/tensorflow/models/tree/master/research/slim

TIDL only accepts optimized frozen graphs.

Please refer below file to convert Checkpoint to the frozen graph.

https://gist.github.com/StanislawAntol/656e3afe2d43864bb410d71e1c5789c1

Then use "tensorflow\python\tools\optimize_for_inference.py" on the output of above step to optimize for inference.

Example : python "tensorflow\python\tools\optimize_for_inference.py" --input=mobilenet_v1_1.0_224.pb --output=mobilenet_v1_1.0_224_final.pb --input_names=input --output_names="softmax/Softmax"

We have developed/defined TIDL library layers based on the layer types Caffe framework. Most of our test cases (Layer level and network level) and demos are based caffe framework. With respect to Tensorflow, we have validated two pre-trained models from tensorflow github (Slim based Mobilenet V1 and Googlenet/inceptionetV1), this covers most of the CNN layers (Convolution, Max Pooling , Average pooling, Batch norm, Fully connected layer, softmax, Relu, Relu6, concate etc).

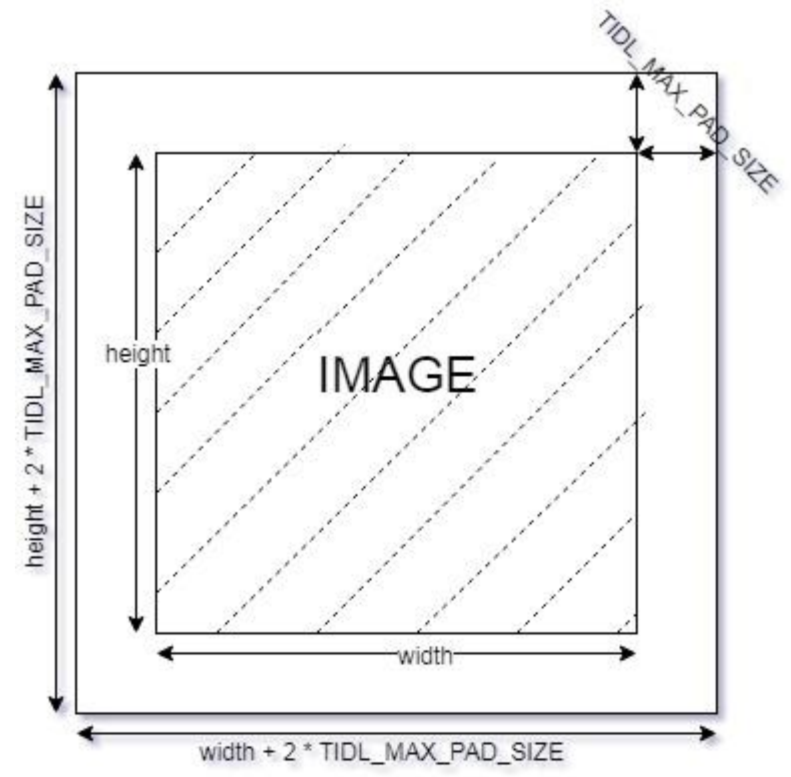
Refer to below trained network on CIFAR-10 dataset (Tf.keras based) to build your network based on this.

https://e2e.ti.com/support/arm/automotive_processors/f/1021/t/689876

We have also have “Tensorflow to TIDL translation” source code as part of our release (OBJ release also has this source files). This can be updated to map any new layer in the tensorflow to TIDL

3.7 Input and Output Data Formats

TIDL Library expects input format per channel like on image below. The TIDL_MAX_PAD_SIZE is defined in algorithm interface file.



3.8 Matching TIDL inference result

The TIDL import step runs the inference on PC and the result generates expected output (With caffe or tensorflow inference). If you observe difference at this stage please follow below steps to debug.

1. Caffe inference input and TIDL inference input shall match. Import step dumps input of the first layer at “trace_dump_0_*”, make sure that this is same for caffe as well.
2. If the input is matching, then dump layer level features from caffe and match with TIDL import traces.
3. TDIL trace is in fixed point and can be converted to floating point (using OutQ printed in the import log). Due to quantization the results will not exactly match, but will be similar.
4. Check the parameters of the layer where the mismatch is observed.
5. Share the input and Parameter with TI for further debug.

We use the statistics collected from the previous process for quantizing the activation dynamically in the current processes. So, results we observe during the process on target will NOT be same (but similar) for same input images compared to import steps. We have validated this logic with semantic segmentation application on input video sequence

TIDL maintains range statistics for previously processed frames. It quantizes the current inference activations using range statistics from history for processes (weighted average range).

Below is the parameters controls quantization.

quantMargin is margin added to the average in percentage.

quantHistoryParam1 weights used for previously processed inference during application boot time

quantHistoryParam2 weights used for previously processed inference during application execution (After initial few frames)

To get the same result in TIDL target same as import step for an image. Please set below parameters during algorithm creation.

```
createParams.quantHistoryParam1 = 0;
createParams.quantHistoryParam2 = 0;
createParams.quantMargin = 0;
```

Set with below parameters for running on video sequence.

```
createParams.quantHistoryParam1 = 20;
createParams.quantHistoryParam2 = 10;
createParams.quantMargin = 20;
```

3.9 TIDL Limitation

- Convolution Layer
  - We have tested the kernel size up to 7x7 (Shall work for higher values also, but not validated)
  - Dilation is tested with 1,2,4.
  - We support only stride 1 and 2. Any value higher than 2 is not supported.
  - Dense convolution flow is supported for only 1xN and 3x3 kernels with stride = 1 and dilation =1
  - Maximum number of input and output channel supported in 1024
- Deconvoltion Layer
  - Number of groups shall be equal to the number of input channels and number of input channels shall be equal to the number of output channels.
  - Only supported stride value is 2
- Arg Max

- – Up to 15 input channels are supported for EVE core and up to 6 channels are supported for DSP core.
- – `out_max_val` = `false` and `top_k = 1` (Defaults) and `axis  = 1` (Supported only across channel)

- InnerProductLayer
  - – Maximum input and output Nodes supported are 4096.
  - – The input data has the flattened (That is C =1 and H =1 for the input data)
  - – A flatten layer cab be used before this layer in C > 1 and H > 1
  - – If a global avg Pooling also can be flattens the output
- Spatial Pooling Layer
  - – Average and Max Pooling are supported with stride 1, 2, 4 and kernel sizes of 2x2,3x3,4x4 etc. STOCHASTIC Pooling not supported
  - – Global Pooling supported for both Average and Max. The output data N=1 and H =1. The output W will be Updated with input 'N'
- BiasLayer
  - – Only one scalar bias per channel is supported.
- CancatLayet
  - – Concatenate is only supported across channel (axis = 1; default).
- CropLayer
  - – Only Spatial crop is supported (axis = 2; default).
- EltWiseLayer
  - – Only supported on two inputs of same or different types
- FlattenLayer
  - – Keeps 'N' unchanged. Makes C=1 and H=1
- ScaleLayer
  - – Only one scalar scale and bias per channel is supported.
- SliceLayer
  - – Slice is only supported across channel (axis = 1; default).
- SoftmaxLayer
  - – The input data has the flattened (That is C =1 and H =1 for the input data)

- SSD
  - Only Caffe-Jacinto based SSD network is validated.
  - Reshape, Permute layers are supported only in the context of SSD network.
  - “share_location” has to be true
  - Tested with 4 and 5 heads.
  - SaveOutputParameter is ignored in TIDL inference.
  - code_type is only tested with CENTER_SIZE.
- **Tensorflow**
  - Only Slim based models are validated. Please refer nceptionNetV1 and mobilenet_1.0 from below as examples for building your models.
  - https://github.com/tensorflow/models/tree/master/research/slim

3.10 Uninstalling the Component

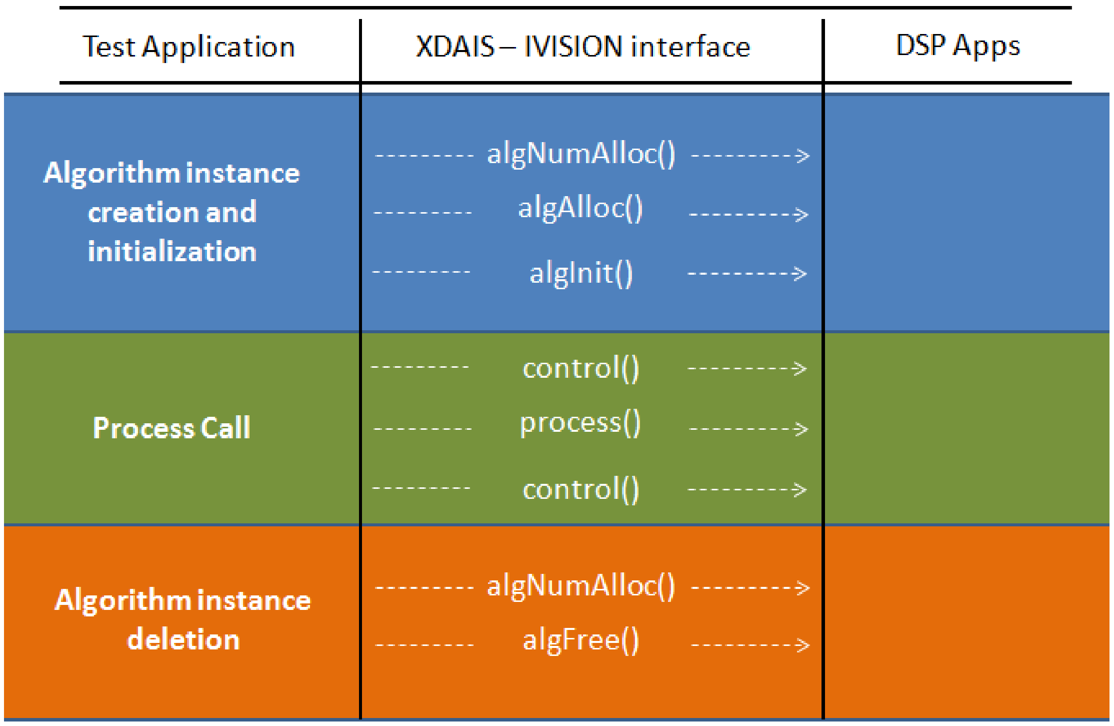
To uninstall the component, delete the algorithm directory from your hard disk.

4 Sample Usage

This chapter provides a detailed description of the sample test application that accompanies this TI Deep Learning Library component.

4.1 Overview of the Test Application

The test application exercises the `IVISION` and extended class of the TI Deep Learning Library Library . The source files for this application are available in the \test\src sub-directories.



Test Application	XDAIS – IVISION interface	DSP Apps
Algorithm instance creation and initialization	--------- algNumAlloc() ---------> --------- algAlloc() ---------> --------- algInit() --------->	
Process Call	--------- control() ---------> --------- process() ---------> --------- control() --------->	
Algorithm instance deletion	--------- algNumAlloc() ---------> --------- algFree() --------->	

Table 3 Test Application Sample Implementation

The test application is divided into four logical blocks:

Parameter setup

Algorithm instance creation and initialization

Process call

Algorithm instance deletion

4.2 Algorithm Instance Creation and Initialization

In this logical block, the test application accepts the various initialization parameters and returns an algorithm instance pointer. The following APIs implemented by the algorithm are called in sequence by `ALG_create()`:

`algNumAlloc()` - To query the algorithm about the number of memory records it requires.

`algAlloc()` - To query the algorithm about the memory requirement to be filled in the memory records.

`algInit()` - To initialize the algorithm with the memory structures provided by the application.

A sample implementation of the create function that calls `algNumAlloc()`, `algAlloc()`, and `algInit()` in sequence is provided in the `ALG_create()` function implemented in the alg_create.c file.

IMPORTANT! In this release, the algorithm assumes a fixed number of EDMA channels and does not rely on any IRES resource allocator to allocate the physical EDMA channels. This EDMA channel allocation method will be moved to IRES based mechanism in subsequent releases.

IMPORTANT! In this release, the algorithm requests two types of internal memory via IALG_DARAM0 and IALG_DARAM1 enums. The performance of the algorithm is validated by allocating DARAM0 to L1D SRAM and DARAM1 to L2 SRAM. Refer datasheet for more information regarding data and program memory sizes.

4.3 Process Call

After algorithm instance creation and initialization, the test application does the following:

Sets the dynamic parameters (if they change during run-time) by calling the `control()` function with the `IALG_SETPARAMS` command.

Sets the input and output buffer descriptors required for the `process()` function call. The input and output buffer descriptors are obtained by calling the `control()` function with the `IALG_GETBUFINFO` command.

Calls the `process()` function to detect objects in the provided feature plane. The inputs to the process function are input and output buffer descriptors, pointer to the `IVISION_InArgs` and `IVISION_OutArgs` structures.

When the `process()` function is called, the software triggers the start of algorithm.

The `control()` and `process()` functions should be called only within the scope of the `algActivate()` and `algDeactivate()` XDAIS functions, which activate and deactivate the algorithm instance respectively. If the same algorithm is in-use between two process/control function calls, calling these functions can be avoided. Once an algorithm is activated, there can be any ordering of `control()` and `process()` functions. The following APIs are called in sequence:

`algActivate() - To activate the algorithm instance.`

`control()` (optional) - To query the algorithm on status or setting of dynamic parameters and so on, using the eight control commands.

`process()` - To call the Algorithm with appropriate input/output buffer and arguments information.

`control()` (optional) - To query the algorithm on status or setting of dynamic parameters and so on, using the eight available control commands.

`algDeactivate()` - To deactivate the algorithm instance.

The do-while loop encapsulates frame level `process()` call and updates the input buffer pointer every time before the next call. The do-while loop breaks off either when an error condition occurs or when the input buffer exhausts.

If the algorithm uses any resources through RMAN, then user must activate the resource after the algorithm is activated and deactivate the resource before algorithm deactivation.

4.4 Algorithm Instance Deletion

Once `process` is complete, the test application must release the resources granted by the IRES resource Manager interface if any and delete the current algorithm instance. The following APIs are called in sequence:

`algNumAlloc()` - To query the algorithm about the number of memory records it used.

`algFree()` - To query the algorithm to get the memory record information.

A sample implementation of the delete function that calls `algNumAlloc()` and `algFree()` in sequence is provided in the `ALG_delete()` function implemented in the alg_create.c file.

4.5 Frame Buffer Management

4.5.1 Input and Output Frame Buffer

The algorithm has input buffers that stores frames until they are processed. These buffers at the input level are associated with a bufferId mentioned in input buffer descriptor. The output buffers are similarly associated with bufferId mentioned in the output buffer descriptor. The IDs are required to track the buffers that have been processed or locked. The algorithm uses this ID, at the end of the process call, to inform back to application whether it is a free buffer or not. Any buffer given to the algorithm should be considered locked by the algorithm, unless the buffer is returned to the application through `IVISION_OutArgs->inFreeBufID[] and IVISION_OutArgs->outFreeBufID[]`.

For example,

Process Call #	1	2	3	4	5
bufferID (input)	1	2	3	4	5
bufferID (output)	1	2	3	4	5
inFreeBufID	1	2	3	4	5
outFreeBufID	1	2	3	4	5

The input buffer and output buffer is freed immediately once process call returns.

5 API Reference

Please refer the CHM file in the `{TIDL_install_path}\Docs` folder for details on the algorithm API reference.

6 FAQ

1. Where can I find TIDL release?

Object release of TIDL is part of TI's processor software development (SDK) for vision. In the vision SDK, you'll find TIDL with documents at \ti_components\algorithms\.
http://www.ti.com/tool/PROCESSOR-SDK-TDAX

Source release of TIDL is available as standalone release via CDDS. Please work with your local TI representative to get access to the same.

2. What are the platforms on which TIDL is supported?

TIDL is primarily targeted for TDA2x/TDA3x SoCs, but not limited to these TI platform. TIDL provides software to accelerate CNN layers on EVE and C6xx DSP. If the TI SoC has either of these cores, then TIDL can be used on this device.

3. Where can I find example CNN models?

We have GitHub repository where we have few caffe-jacinto trained models. Thew models are validated on using TIDL on TI device. Models can be found in below path
https://github.com/tidsp/caffe-jacinto-models

4. What are the supported model-formats?

We have validated model trained with BVLC-caffe, Caffe-Jacinto, and tensorflow

5. What are tensorflow models validated?

We have validated inceptioNetV1 (googleNet) and mobile net models from below path. These are tensorflow checkpoint, the needs to be saved as frozen graph to run on TIDL.
https://github.com/tensorflow/models/tree/master/research/slim

6. Which tensorflow version is used by the model Import tool

We have used tensorflow 1.0 proto buffer format. The source files for import tool is part of release package. If you are using later version, the import tool can be re-built with latest proto buffer format files from tensorflow repository

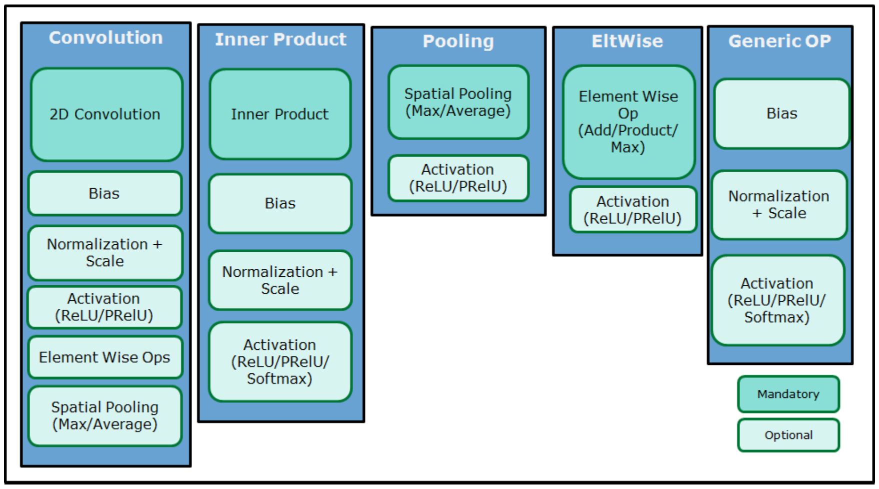
7. What are the supported CNN layers?

Majority of layers needed object classification, Semantic segmentation and Object detection are supported by TIDL. Please refer the data sheet for complete list of supported layers.

8. What is the difference between sparse and dense convolution?

TIDL support two flavors of convolution layer namely Sparse and Dense flow. Functionally both the flows will generate same results. The sparse flow takes advantage for zero kernels co-efficient and improves the execution speed by not computing them. So default sparse convolution can be used, however the sparse flow has relatively high overheads when processing small ROIs (Input/Feature maps smaller than 32x32). For these cases we recommend to use dense convolution flow. Always use a reasonably large resolution (Preferably width is multiple of 32) for performance benchmarking. So while importing models each layer can be set to either sparse or dense using “conv2dKernelType” parameter in import config file.

9. **Some layers are missing after model import. Is this expected?**
TIDL import tool merges layers processing to speed up the execution. Each layer in caffe prototxt does not one to one mapping on imported model. For example, Convolution + Batchnorm + RelU + MaxPool will be merged a on TILD_ConvolutionLayer. Also permute, reshape and softmax layers will be merged in dectectionOutput layer for SSD.



10. **What is Q format of output?**
 The output is 8 bit fixed point representation with the scaling factor. The 8 bit value can be divided by scaling factor to get the floating point value. The scaling factor (in Q8 format) is available in dataQ of (sTIDL_DataParams_t). Import tool also prints this information for sample data as “Out Q : **X**”.
Related E2E query : https://e2e.ti.com/support/arm/automotive_processors/f/1021/t/642684

11. **How to read segmentation and SSD outputs?**
For **segmentation** output is a Binary file having one byte data for heavy pixel in the input image. The one byte data is unsigned char data giving the Pixel category.
0: Background
1 :ROAD
2 : Pedestrian
3 : Road Sign
4 : Vehicle
Overlay code that used in vision SDK semantic segmentation demo to overlay output on the YUV can be used for Visualization.
https://e2e.ti.com/support/arm/automotive_processors/f/1021/t/637598

For **SSD** output is a Binary file having 7 float values for each detected object, so if keep_top_k = 20, then output size is 560 bytes (7x4x20). Refer to below e2e thread file for visualization code, https://e2e.ti.com/support/arm/automotive_processors/f/1021/p/679186/2502331#2502331

12. **How to find sparsity in each layer?**
    The Import tool prints the actual Sparsity considered by the TIDL after quantization and alignment required for SIMD processing. Please find the first number after the each convolution layer after " Sparsity : **X**"

13. **Will TIDL inference match with caffe-jacinto/ caffe?**

Same quantization logic is implemented in caffe-jacinto and TIDL. We did not spend effort to match the outputs. The caffe-jacinto will provide close indication of expected accuracy when it is deployed using TIDL. https://e2e.ti.com/support/arm/automotive_processors/f/1021/t/637598

14. **How to read the import tool log?**
Lot of information in the import log is debug trace. We are planning to mask many of them in the next release. Relevant information are Number of layers, Layer Types, tensor Size, Sparsity in Convolution layer and Out Q of each layer. User can safely ignore rest of the information.

15. **Is caffe SSD model support?**
Yes. TIDL 01.01.00.00 and above releases support SSD based on caffe framework.

16. **Why is softmax layer running slow EVE?**
Sotfmax Layer is implemented using floating point operation on EVE. EVE id fixed point processor and floating point operation are not optimal. As per our understanding, most of the real applications do not require softmax in the inference. If the application requires, In the final system user shall run this layer on C6xx DSP.

17. **What is the input buffer format for TIDL?**
Input is a four dimensional tensor (8-bit) with size of “batch x channel x (height + 2*TIDL_MAX_PAD_SIZE) x (width + 2*TIDL_MAX_PAD_SIZE)”.
Refer “itidl_ti.h” for TIDL_MAX_PAD_SIZE

18. **Do we have example use-case for TIDL in Vision SDK / Processor SDK?**
We have below two use cases in Vision SDK / Processor SDK. Refer correspondiong users guide for more information
chanis_semSeg – Real - time Semantic segmentation demo
chanis_tidl – File IO based use case to validate user model

19. **How could validate the TIDL model accuracy with own test data**
You need to use a script to run the inference multiple times with script. Refer below thread for more info.https://e2e.ti.com/support/arm/automotive_processors/f/1021/t/669604

20. **Do we have a white paper/publication to understand TIDL and sparsity?**
Please refer Below whitepaper for high level TIDL overview
http://www.ti.com/lit/wp/spry314/spry314.pdf
Please refer CVPR paper for Sparse convolution

http://openaccess.thecvf.com/content_cvpr_2017_workshops/w4/papers/Mathew_Sparse_Quantized_Full_CVPR_2017_paper.pdf

21. How to set “layersGroupId” and “conv2dKernelType” parameters in import config file?

The parameter "layersGroupId" specifies the group of layers that needs to be processed on a given CORE, let say layersGroupId = 1 for EVE and layersGroupId =2 for DSP.
So user need to set this parameter for each layer in the import config file based on below condition to get optimal execution of SSD network.
Condition : Only DectectionOutputLayer should run on DSP and rest of the all the layers on EVE in the SSD network.

The parameter " conv2dKernelType" can be either 0 or 1 for each layer. Default value is 0 for all the layers. Set it to 0 to use sparse convolution, otherwise, set it to 1 to use dense convolution.
So, user need to set this parameter for each layer in the import config file based on below condition to get optimal execution of SSD network.
Condition : Use dense convolution for Convolution layers with width x height < 64x64, as dense convolution is optimal for small resolutions and this value is ignored for non-conv layers.

Please refer to SSD import config file “tidl_import_JDetNet.txt” for details,

```
layersGroupId = 0   1      1      1      1      1      1      1      1      1      1
    1      1      1      1      1      1      1      1      1      1      1      1
    1      1      1      1      1      1      1      1      1      1      1      1
    1      1      1      1      1      1      1      1      2      0
```

Here, first and layers are data layers so values can be ignored, and all the remaining values are set to 1 (to execute on EVE) except the DetectionOutput layer(set as 2 to run on DSP).
Please note that in the previous release(01.01.00.00), we used to set some intermediate layers also to 2 (to run on DSP), but in this release all those layers are optimized on EVE so they can be on run EVE now, but user can set/choose layers on run on EVE or DSP based on their network to get optimial performance.

```
conv2dKernelType = 0    0      0      0      0      0      0      0      0      0
    0      0      0      0      0      0      1      1      1      1      1      1
    1      1      1      1      1      1      1      1      1      1      1      1
    1      1      1      1      1      1      1      1      1      1      1
```

Here also, first and layers are data layers so values can be ignored, and in the remaining layers, first few layers are set to 0 (sparse convolution) and others to 1(dense convolution) as they are small layers.

22. How to run models imported above, part of SSD net in EVE and DSP cores ?

This can be done using parameters “layersGroupId” and “runFullNet” in infer config file, please refer to tidl_config_jdetnet.txt as an example.

1. First run on EVE core by setting below values in infer config file,
   ```
    layersGroupId = 1
   runFullNet = 0
   inData   = "..\..\test\testvecs\input\trace_dump_0_768x320.y"
   outData   = ".\stats_tool_out_eve.bin"
   ```
   This will run all those layers which were set to 1 in import config file on EVE and generates intermediate output (stats_tool_out_eve.bin)

2. Now run on DSP core by setting below values in infer config file,

   ```
    layersGroupId = 2
   runFullNet = 0
   inData   = ".\stats_tool_out_eve.bin"
   outData   = ".\stats_tool_out.bin"
   ```

   This will run all those layers which were set to 2 in import config file on DSP and generates final output (stats_tool_out.bin)

   Note that when runFullNet =1, it will ignore “layersGroupId” values and runs all the layers on EVE or DSP on which it is running.