D3 Engineering

Define | Design | Deploy

TI / AWS Deep Learning Demo

Defect Detection using the AM57x Sitara Processor

Revision Control			
Revision	**Date**	**Author**	**Notes**
0.1	6/24/2020	J. Kiggins, M. Simpson - D3 Engineering	Initial draft for internal review.
0.2	6/29/2020	J. Kiggins, M. Simpson - D3 Engineering	Revisions based on feedback.
0.3	07/10/2020	J. Kiggins	Revision based on code changes.
1.0	09/01/2020	J. Kiggins	Revision based on code changes

TABLE OF CONTENTS

1 Introduction 1
2 Demo Components 2
2.1 Hardware Components 2
2.2 Software Components 2
3 The Neural Network Model 4
4 Building the Demo 5
4.1 Clone the Demo Code from the Git Repository to the Host 5
4.2 Download and Install the TI Processor SDK on the Host 6
4.3 Create a Bootable SD Card for the Target 7
4.4 Install the SD Card on the Target and Boot the System 7
4.5 Download the Motor Controller Library to the Host 7
4.6 Cross Compile the Motor Controller Library on the Host 8
4.7 Cross Compile and Build the Demo on the Host 8
4.8 Install the DLR Runtime on the Target 8
4.9 Copy the Demo to the Target 8
5 Running the Demo 9
5.1 Configuring the Demo 9
6 Modifying the Demo 10
6.1 Training Locally 11
6.2 Training in the Cloud 11
7 Summary 13

1 INTRODUCTION

This document describes a defect detection demo developed by D3 Engineering for Texas Instruments and Amazon. The demo shows the ability of the Texas Instruments Sitara platform, specifically the AM5729, to run AI applications at the edge. The AM5729 architecture has multiple processors that can be leveraged for a variety of compute intensive tasks, in this case running a MobileNetV2 neural network trained to detect defects in parts.

The parts chosen are camera modules and the defects the model is trained to detect are missing parts. In order to simulate parts on an assembly line passing by an inspection camera, a turntable was constructed containing the set of parts to be inspected. The turntable rotates a part into the field-of-view of an inspection camera, the neural network determines a pass/fail status, and then the sequence continues.

2 DEMO COMPONENTS

2.1 HARDWARE COMPONENTS

The physical setup consists of a stepper motor turntable upon which the parts are placed, a USB motor controller, a USB webcam, and an HDMI display. These elements along with a host system are connected to the target system, a TI AM5729 Industrial Development Kit (IDK) where the demo runs. A bootable SD card contains the Linux operating system and filesystem for the target. This is shown below.

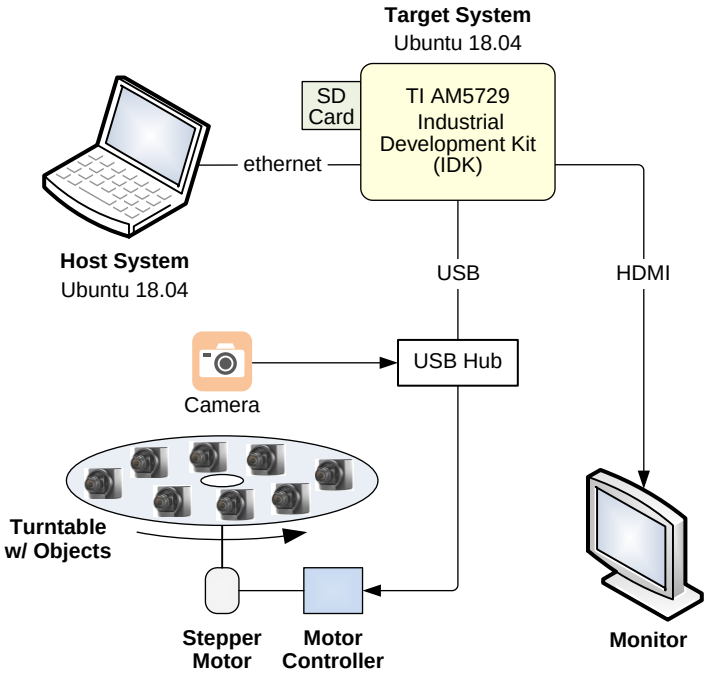

Figure 1 The demo setup.

2.2 SOFTWARE COMPONENTS

There are several software components used to create the defect detection demo. On the host system, there is the Processor SDK for AM57x Sitara Processors - Linux[1] that provides a set of AM5729 development tools and enables the creation of a bootable Linux system image for the AM5729-based target system. The Processor SDK also provides TVM/Neo, a deep learning compiler that takes a model trained on the host and optimizes it for execution on the AM5729 Sitara processor.

On the target system, there is the TI SDK embedded Linux system created using the Processor SDK that provides the operating system for the AM5729 evaluation board. Also required on the target is TIDL (Texas Instruments Deep Learning), a set of open-source Linux software packages and tools for executing deep learning models on TI embedded systems. TIDL provides the ability to offload deep learning compute intensive workloads from the general-purpose ARM cores to the EVE (Embedded Vision Engine) cores and C66x DSP cores on the AM5729 Sitara processor. This can significantly speed up these types of workloads. TIDL also includes the Neo-AI-DLR (Deep Learning Runtime) an open source common runtime for deep learning models and decision tree models compiled by TVM, AWS SageMaker Neo, or Treelite. TIDL requires pre-trained compiled models stored on the target device filesystem as training on the target device is not currently supported. TIDL and TVM are included in the Processor SDK software distribution.

[1] Referred to as the *Processor SDK* elsewhere in the documentation.

Also residing on the target are two trained MobileNetV2 neural network models, one compiled for the ARM general purpose CPUs and the other for the EVE and C66x DSP hardware accelerators on the AM5729 Sitara processor. These models are the software components that perform the actual defect detection.

Lastly is the defect detection demo application. The application code is written in C++ and performs model loading and switching, controls the turntable, acquires images, invokes the model, displays output, and provides a command line interface to control the demo.

The major software components described above are shown below.

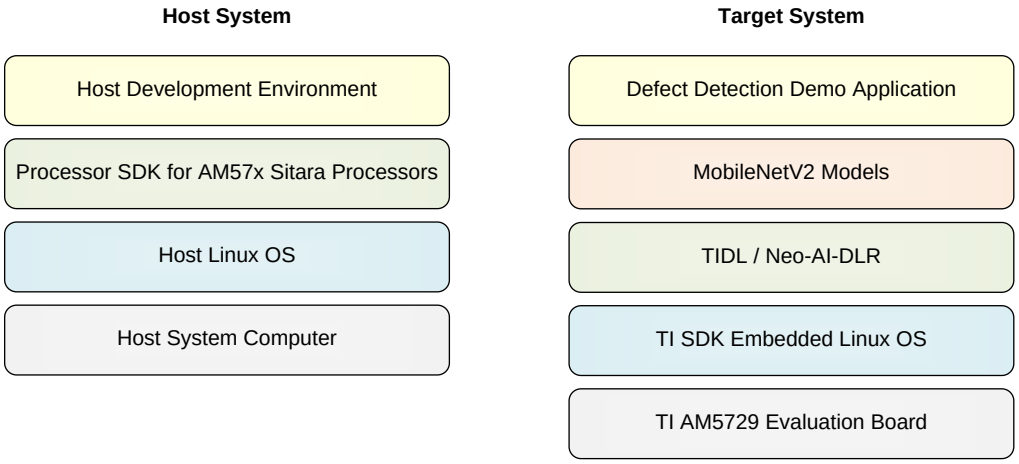

Figure 2 The major software components used for the demo.

3 THE NEURAL NETWORK MODEL

The demo utilizes MobileNetV2, a general-purpose computer vision neural network designed for mobile devices. MobileNet is based on the foundational work by Google researchers as described in the publication MobileNets: Efficient Convolutional Neural Networks for Mobile Vision. MobileNetV2 builds upon the ideas from MobileNetV1 as described in a Google AI Blog entry. The source code for MobileNetV2 is available on GitHub.

The demo models were trained using a set of annotated images of the objects used in the demo, in this case camera modules presented to the system on a turntable. Acquiring training images, annotating them, training the model, compiling it for the target, and evaluating its effectiveness can be a complex and time-consuming process. The reader is encouraged to seek out information on the internet and other sources to familiarize themselves with the practical implications of creating and training convolutional neural network models, in particular the MobileNetV2 model used in the demo.

The demo includes pretrained models compiled for the AM5729 processor.

4 BUILDING THE DEMO

This section describes how to build the demo. It assumes that the requisite hardware components as described in section 2.1 have been assembled and the host and target systems are ready for software installation.

The high-level steps to build the demo are shown below.

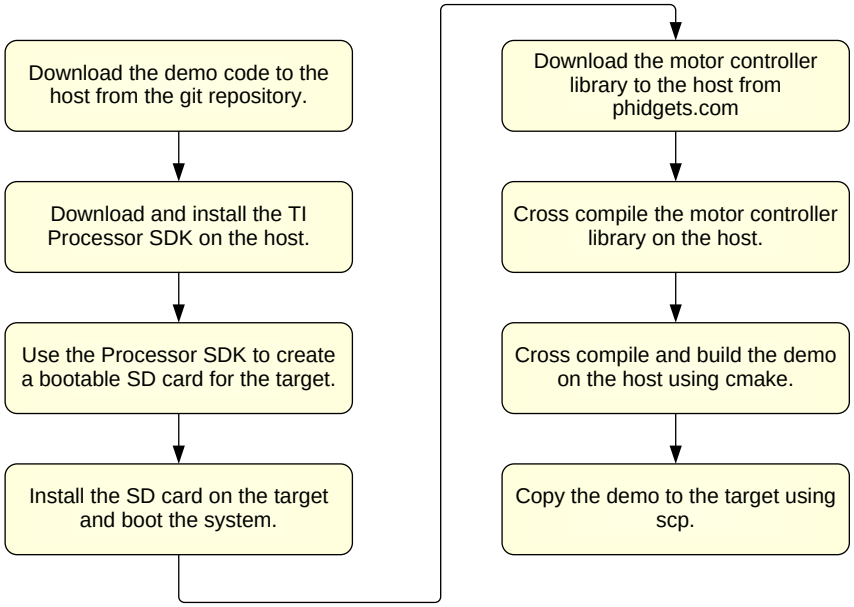

Figure 3 Flow chart for building the demo.

The following sections describe each step.

4.1 CLONE THE DEMO CODE FROM THE GIT REPOSITORY TO THE HOST

To acquire the demo code, it must be cloned from the git repository to the host system. Create a directory on the host, and from that directory issue the following command.

- `git clone --recursive https://github.com/typemismatch/ti-d3-kit.git`

The directory structure of the demo code in the git repository is shown below.

```
.
├── docs
│   ├── TI-AWS-Deep-Learning-Demo.docx
│   └── TI-AWS-Deep-Learning-Demo.pdf
├── sagemaker
│   ├── config.py
│   ├── data
│   ├── data.py
│   ├── defect-demo.ipynb
│   ├── mobilenet.py
│   ├── mobilenet_v2.py
│   ├── reqs.py
│   ├── requirements.txt
│   ├── train.py
│   └── utils
├── target
│   ├── build
│   ├── CMakeLists.txt
│   ├── dlr-1.2.0.linux-armv7l.tar.gz
│   ├── env.sh
│   ├── include
│   ├── lib
│   ├── models
│   ├── scripts
│   ├── src
│   ├── stops.txt
│   ├── ti-processor-sdk -> /home/user/docs/software/ti-processor-sdk-linux-am57xx-evm-06.03.00.106
│   ├── video
│   └── video.mkv
└── tvm -> release-1.2.0
```

Figure 4 The demo source directory structure.

4.2 Download and Install the TI Processor SDK on the Host

To develop code for AM5729 Sitara processor and create a bootable SD card, the Processor SDK for AM57x Sitara Processors - Linux must be installed on the host system. For a complete overview of the SDK, please refer to the Processor SDK Linux Software Developer's Guide (SDG). The demo file **target/Makefile** expects the Processor SDK to be extracted to **target/ti-processor-sdk**. This can be its actual install location or a symlink. To add all the cross-compile items (libs, compilers, etc.) source **env.sh** (. env.sh) available in the **target** directory of the downloaded demo code.

The Processor SDK contains the code and tools used to develop for support TI devices and has the following top-level directories and files.

```
├── bin
├── board-support
├── docs
├── example-applications
├── filesystem
├── linux-devkit
├── Makefile
├── Rules.make
└── setup.sh
```

Figure 5 Processor SDK top-level directory structure.

- **bin** - Contains the helper scripts for configuring the host system and target device. Most of these scripts are used by the setup.sh script.

- **board-support** - Contains the SDK components that need to be modified when porting to a custom platform. This includes the kernel and boot loaders as well as any out of tree drivers.
- **docs** - Contains various SDK documentation such as the software manifest and additional user's guide. This is also the location where you can find the training directory with the device training materials.
- **example-applications** - Contains the sources for the TI provided example applications seen during the out-of-box demonstration.
- **filesystem** - Contains the reference file systems. These include the smaller base file system as well as the full-featured SDK file system.
- **linux-devkit** - Contains the cross-compile toolchain and libraries to speed development for the target device.
- **Makefile** - Provides build targets for many of the SDK components from the top-level of the SDK.
- **Rules.make** - Sets default values used by the top-level Makefile as well as sub-component Makefiles.
- **setup.sh** - Configures the host system as well as the target system for development.

4.3 Create a Bootable SD Card for the Target

Create a bootable SD card for the AM5729 target system using the instructions provided by Texas Instruments. Note that a 16GB SD card is recommended. The reader is encouraged to familiarize themselves with the TI SD card documentation as there are many installation permutations possible. Briefly, the steps to create a bootable SD card on the host system are as follows.

1) Create the 1st partition: 100MiB, FAT32, label="boot", boot flag
2) Create the 2nd partition: 8+GiB, ext3 or ext4, label="rootfs"
3) Deploy the filesystem to rootfs partition:
   - `sudo tar pxf ${TIDL_PLSDK}/filesystem/tisdk-rootfs-image-am57xx-evm.tar.xz -C /path/to/mounted/rootfs/`
4) Deploy pre-builts to boot partition:
   - `cp ${TIDL_PLSDK}/board-support/prebuilt-images/MLO-am57xx-evm /path/to/mounted/boot/MLO`
   - `cp board-support/prebuilt-images/uEnv.txt /path/to/mounted/boot/uEnv.txt`
   - `cp board-support/prebuilt-images/u-boot-am57xx-evm.img /path/to/mounted/boot/u-boot.img`

4.4 Install the SD Card on the Target and Boot the System

Insert the bootable SD card into the AM5729 IDK board, power up the unit, and verify that it boots correctly. You should see the Matrix[2] application from the Processor SDK displayed on the HDMI monitor. Matrix can also be accessed remotely through a web browser on the host system. You can also connect to the target using a terminal emulator to view the serial console and interact with the embedded Linux system directly, for example to run ifconfig to get the IP address of the target board in order to connect to it over the network.

4.5 Download the Motor Controller Library to the Host

A Phidgets stepper motor controller is used to control the turntable. Download the libphidget library source (libphidget22-1.6.20200306.tar.gz) and extract it to a convenient location on the host.

[2] Matrix is an HTML 5 based application launcher created to highlight available applications and demos provided in the Processor SDK.

4.6 Cross Compile the Motor Controller Library on the Host

Cross-compile the libphidget motor controller library and add it to the Processor SDK standard library path using the steps shown below.

- `cd libphidget22-1.6.20200306`
- `./configure --host=arm-linux-gnueabihf \`
- `--prefix=<path-to>/ti-processor-sdk-linux-am57xx-evm-06.02.00.81/linux-devkit/sysroots/armv7at2hf-neon-linux-gnueabi/usr/`
- `make -j4`
- `sudo "PATH=$PATH" make install # sudo is required due to Processor SDK folder permissions`

4.7 Cross Compile and Build the Demo on the Host

The demo uses cmake to control the compilation and build process. The normal set of commands can be used to build the full demo. Note: If you've already sourced **env.sh**, no need to do it again. The build steps are shown below.

- `. env.sh  # source env.sh if you haven't already`
- `mkdir build  # or cd if it already exists`
- `cmake ..`
- `make –j4`

4.8 Install the DLR Runtime on the Target

To take advantage of the TIDL accelerated model, a corresponding version of the neo-ai-dlr runtime is needed. To install this, copy `target/dlr-1.2.0.linux-armv7l.tar.gz` to the target, and extract at the root directory.

```
scp target/dlr-1.2.0.linux-arm7l.tar.gz am57:
cd /
tar –xzvf ~/dlr-1.2.0.linux-arm7l.tar.gz
```

4.9 Copy the Demo to the Target

All demo libraries are linked statically, so there is no need to install any shared libraries on the target. Simply copy over the target folder using the following command.

- `scp ./build/target <target-ip-or-hostname>:`

This folder contains the following.

- `tvm-live – the demo binary`
- `models/`
    i. `model_arm/ – compiled model for execution on arm processors (not included for current release).`
    ii. `model_tidl/ – compiled model for execution on eve cores.`
- `video/ - folder containing images, representing a video for input to the demo`

It may be faster to setup sshfs and use rsync for copying files to the target, if doing development.

```
mkdir /tmp/am57
sshfs root@<target-ip>: /tmp/am57
rsync –av ./build/target/ /tmp/am57/target
```

5 RUNNING THE DEMO

There are command line options to start the demo in one of a few different modes. If a user wishes to switch modes, they must stop the demo then launch it again. All supported options are listed below.

```
Defect Detection Demo

Usage: tvm-live [OPTION...]

  -s, --source arg  Image Data Source (Required)
  -f, --file arg    Video File (Required if source=video)
  -h, --help        Print Help
  -v, --noinfer     Don't run inference
  -n, --notable     Don't spin the table
  -i, --imgcap PATH Dump images to PATH, ex: PATH/img0000.png
  -m, --model arg   Compiled Model Path (Required)
```

To switch between running inference on **ARM** or **TIDL,** specify `-m model_arm` or `-m model_tidl` respectively.

The demo supports inference on images captured from a camera and turn-table setup, or a series of images that were prerecorded. To use camera/turn-table mode specify `-s camera,` to use prerecorded input specify `-s video` and `-f video.mkv`.

Before running the demo with `model_tidl` or `model_arm` libdlr.so needs to added to the library search path.

```
export LD_LIBRARY_PATH=/usr/dlr
```

If running the demo with `-m model_tidl an additional environment variable must be set`

```
export TIDL_SUBGRAPH_DIR=models/model_tidl
```

5.1 CONFIGURING THE DEMO

There are several configuration options for the demo. Currently they all reside in tvm-live.cpp, so the demo must be rebuilt after changing any config options. Below are some of the config options:

- Image capture width and height
- Display width and height + crop origin
- Model label map
- Model input and output details
- Stats display font size
- Turntable movement details, including settle time and ticks per rotation

6 MODIFYING THE DEMO

Aside from the configuration changes in the previous section, the only other change that can be made to the demo is to alter the MobileNetV2 model to recognize defects in a different set of objects. The basic process involves acquiring a new set of training images and using them to retrain and recompile the model. To speed up the process, transfer learning[3] can be used whereby the existing demo model is used as a starting point.

The details of training convolutional neural networks are beyond the scope of this document. But at a high-level there are two basic options for retraining the MobileNetV2 model used in the demo: train and compile locally on the host system or train and compile in the cloud using AWS SageMaker Neo. For deploying the model to the target from SageMaker, an option is to use AWS Greengrass. These options are depicted below.

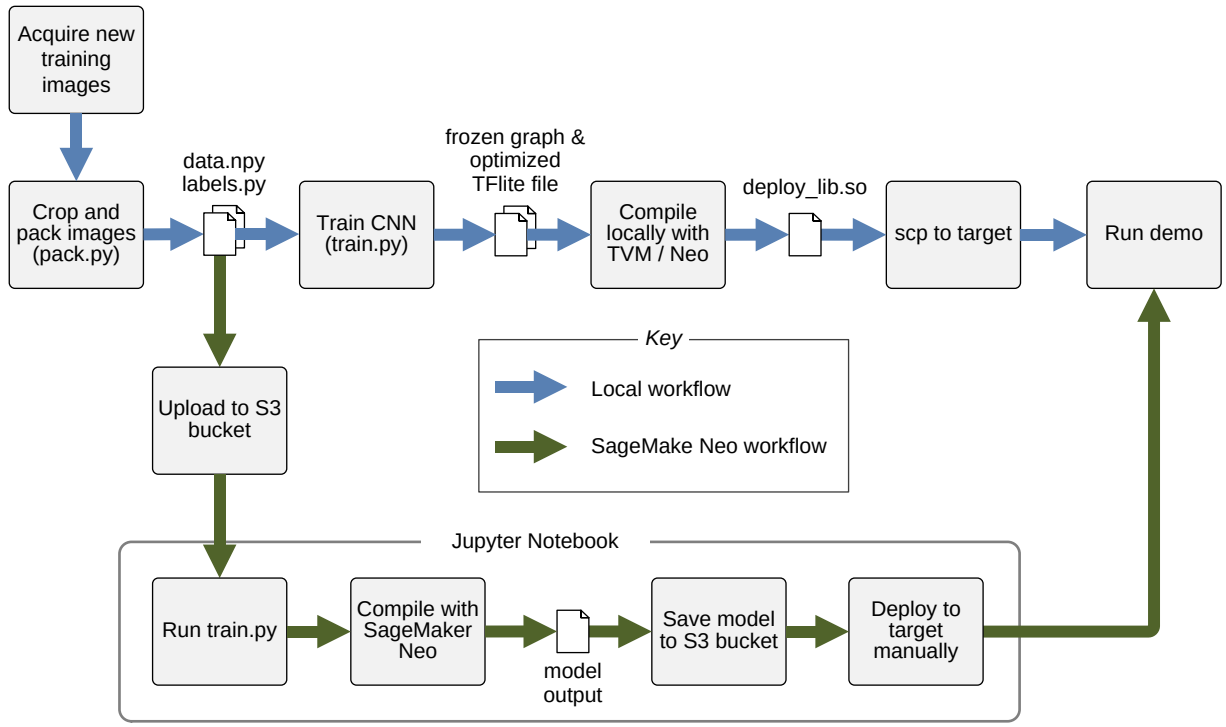

Figure 6 Flowchart for retraining the demo model.

The model definition and training code are split up among a few files:

sagemaker/mobilenet.py - Defines the linear bottleneck layer (shared among MobileNet variants).
sagemaker/mobilenet_v2.py - Provides code for instantiating and deploying the defect detection model.
sagemaker/config.py - Configuration code for model definition and training.
sagemaker/train.py - Script for training a model, supports both SageMaker Neo and host system training.

There are also a few utility scripts which can be helpful:

sagemaker/eval.py - Evaluates model performance for a given model checkpoint file.
sagemaker/freeze.py - Generates a frozen graph .pbtxt file from a checkpoint.
sagemaker/utils/txmodel.py – Transforms the model to be compatible with TVM compile.
sagemaker/utils/compile-tvm.py - Compiles model using TVM, sitara_am57xx is targeted by default.

[3] https://en.wikipedia.org/wiki/Transfer_learning

6.1 TRAINING LOCALLY

Create a virtual environment on the host with the provided **requirements.txt** file available in the top-most directory of the demo code. This will install TensorFlow 1.15.0, along with other dependencies. If you require GPU support please install tensorflow-gpu==1.15.0, along with CUDA. See instructions for GPU support here.

The default configuration expects two files for training:

data/data.npy - should contain N images in NHWC[4] format.
data/labels.npy - should contain N labels. Each element either 0 (pass) or 1 (fail).

All supported changes to hyperparameters can be made in **sagemaker/config.py**. These include changing the learning rate, optimizer, adding regularization, etc.

Before training open config.py and verify the options are set as needed. Specifically, that the number of epochs is right, that the data path is pointing to the proper directory, and that model-checkpoint is enabled if desired. To begin training locally run...

```
./train.py --local
```

This will generate a unique folder under artifacts, ex: `mobilenetv2-sgd__rplat_20-06-17_06:56:23.`

This folder will at least contain the following files after training

- **training.png** - a graph of training and validation loss
- **data/** - a folder containing train, test, and validation files, as they were randomly chosen from the original data.npy and labels.npy
- **config.json** - the config file used during that training run
- **history.pkl** - the history object returned by keras' model.fit() function, exported by pickel.

Depending on your config.py settings, this directory may also contain various `epoch-val_acc.ckpt` files, saved by the ModelCheckpoint callback.

6.2 TRAINING IN THE CLOUD

Amazon SageMaker Neo is a cloud-based machine-learning service that enables developers to train, compile and deploy neural network models on embedded systems and edge-devices. Training data is placed in an S3 bucket and a Jupyter notebook included with the demo (sagemaker/defect-demo.ipynb) sets up the training environment and calls the training script. After training, the exported model is picked up by the Neo compile framework and built for AM5729 Sitara target device. At this point, Neo saves the training artifacts in an S3 bucket created for the purpose. It will be named according to the AWS region targeted, for example **sagemaker-us-east-1-296527333824.** The compiled model artifacts are stored in a tarball, named depending on the target hardware specified in the Juypter notebook. For this demo the file is named **model-sitara_am57x.tar.gz.** At this point, the model could be downloaded to the target manually, or deployed using GreenGrass.

To train and compile using Sagemaker, first create a new notebook instance, then upload sagemaker/defect-demo.ipybn to the top-level of the notebook. Next, create a subfolder named src and upload the following files.

- **sagemaker/data.py -> src/data.py**
- **sagemaker/mobilenet.py -> src/mobilenet.py**
- **sagemaker/mobilenet_v2.py -> src/mobilenet_v2.py**
- **sagemaker/config.py -> src/config.py**
- **sagemaker/train.py -> src/train.py**
- **sagemaker/reqs.py -> src/reqs.py**
- **sagemaker/utils/txmodel.py -> src/txmodel.py**

[4] NHWC = NumSamples x Height x Width x Channels

To kick off training, open defect-demo.ipynb, and select cells->run all. The specific steps taken by the Juypter notebook, and its interaction with the SageMaker API are documented as part of the notebook.

7 SUMMARY

The TI AM5729 Industrial Development Kit provides a powerful platform for running AI at the edge. As the deep learning defect detection demo illustrates, using TIDL to make use of the Sitara accelerator cores can significantly speed up the execution of a convolutional neural network. Utilizing AWS SageMaker Neo for model training and compiling can also significantly speed up the training and evaluation phases of model development.