NXP Semiconductors
Application Note

Document Number: AN12224
Rev. 0, 07/2018

ML SDK with OpenCV

Contents

1. Introduction

1.1. Document Structure

This document is comprised of three primary sections. Section *2* introduces “ML SDK with OpenCV” by describing what it is and what it aims to achieve. Section *3*, The ML SDK Getting Started Guide contains details about hardware and software prerequisites, followed by step-by-step guide for enabling OpenCV 3.4.1, running DNN demos and running non-NN demos. Section *4* describes how to prepare your own models for OpenCV inference, and is focused on two deep learning frameworks – Caffe and TensorFlow.

NOTE

OpenCV 3.4.1. also supports Torch and Darknet models; these are not covered in this document.

Section *5* presents the interoperability of ML SDK and Google Cloud services. Section *6* answers some of the most frequent questions, and in the Appendices, there is a list of supported TensorFlow layers in OpenCV 3.4.1.

1.2. What is Machine Learning

Machine Learning (ML) is a computer science domain having its roots in the 1960's. ML provides algorithms capable of finding patterns and rules in data. Today, machine learning is an inherent part of our lives, from services like Google search, YouTube, voice assistants (e.g. Siri, Alexa) to powerful predictions and optimizations used in Amazon for fast delivery

© 2018 NXP B.V.

of ordered goods, discovering new music using Spotify-like services, or even autonomous driving. And in the future, we can expect even more.

In 2010, a huge boom started called Deep Learning - it is a fast-growing subdomain of ML, based on Neural Networks (NN). Inspired by the human brain, Deep Learning has achieved state of the art results in various tasks e.g. Computer Vision (CV) and Natural Language Processing (NLP). Neural Nets are capable of learning complex patterns from millions of examples. We can expect huge adaptation in the embedded world – an area where NXP is a leader. As a starting point for enabling its customers, NXP has created ML SDK with OpenCV, which is described in following pages. ML SDK with OpenCV is a collection of capabilities supporting i.MX 8 devices, including model deployment using OpenCV, Google Cloud Interoperability, and several Vision components.

Disclaimer regarding software and models referenced in this document

The software and models referenced in this document are publicly-available third-party software and models, which are not provided or licensed by NXP, and NXP grants no rights to such software and models. Your use of such publicly-available third-party software and models is subject to the terms of each applicable license. You must agree to the terms of each applicable license, or you cannot use the referenced software or models. To the maximum extent permitted by law, NXP expressly disclaims any warranty for the software and models referenced in this document. The references to such software and models are provided "AS IS", without warranty of any kind, either express or implied, including without limitation the implied warranties of merchantability, fitness for a particular purpose, or non-infringement. You assume the entire risk arising out of the use or performance of the software and models referenced in this document, or any systems you design using such software and models (if any). NXP SHALL HAVE NO LIABILITY WHATSOEVER ARISING OUT OF OR IN CONNECTION WITH YOUR USE OF OR INABILITY TO USE THE SOFTWARE AND MODELS REFERENCED IN THIS DOCUMENT OR PRODUCT(S) THAT MAY USE SUCH SOFTWARE AND MODELS.

2. ML SDK with OpenCV

2.1. Overview of ML SDK with OpenCV

ML SDK with OpenCV aims to deliver essential machine learning enablement by providing OpenCV machine learning support on i.MX 8. OpenCV is an open source computer vision library and one of its modules, called ML, provides traditional machine learning algorithms. Another important module in OpenCV is Deep Neural Network (DNN); it provides support for neural network algorithms.

As its name suggests, ML SDK with OpenCV is based on OpenCV 3.4.1 and contains:

- Yocto recipes for OpenCV 3.4.1
- Documentation describing how to enable OpenCV 3.4.1
- Getting started guide for neural network demos
- Getting started guide for non-neural network demos

At the time of writing this documentation, Linux L4.9.51 Beta2 is the latest Linux Yocto BSP Release for i.MX 8QuadMax. It already includes OpenCV 3.4.0, but you must use OpenCV 3.4.1 for the improved and updated support for neural network algorithms. Future BSP Releases will be shipped with OpenCV 3.4.1 built-in by default.

Why OpenCV? OpenCV offers a unitary solution for both neural network inference (DNN module) and classic machine learning algorithms (ML module). Moreover, it includes many computer vision functions, making it easier to build complex machine learning applications in a short amount of time and without having dependencies on other libraries.

OpenCV has wide adoption in the Computer Vision field and is supported by a strong and very active community. Key algorithms are specifically optimized for various devices and instructions sets. For i.MX, OpenCV uses Arm® NEON acceleration.

At its core, the OpenCV DNN module implements an inference engine and does not provide any functionalities for neural network training. For more details about supported models and supported layers, check the official OpenCV DNN wiki page.

The OpenCV ML module contains classes and functions for solving machine learning problems e.g. classification, regression or clustering. It involves algorithms such as support vector machine (SVM), decision trees, random trees, expectation maximization, k-nearest neighbors, classic Bayes classifier, logistic regression, and boosted trees. For further information, visit the official reference manual and machine learning overview.

NOTE

The ML module in OpenCV 3.x has changed and code is not compatible with OpenCV 2.x.

For more details about OpenCV 3.4.1, check the official OpenCV ChangeLog.

2.2. Known Limitations

- Tested boards: i.MX 8QuadMax MEQ
- Not tested with board sensors (cameras, etc.)
- Not tested on the GPU

3. Getting Started Guide

3.1. Prerequisites

Prerequisites for ML SDK with OpenCV are the same as for the Yocto project setup. For more information see the official documentation for Linux L4.9.51 for i.MX 8QuadMax Beta2

3.1.1. Hardware Requirements

- 1 x Linux Host Machine with 120 GB HDD space available and Internet connection

- 1 x i.MX 8QuadMax MEK board with Internet connection
- 1 x LVDSI to HDMI adapter (optional for video output)
- 1 x SD card (tested on 16 GB uSD card + uSD/SD adaptor)

3.1.2. Software Requirements

- **Host OS**: Ubuntu (>=14.04, tested on 16.04)
- **Host packages:**

Essential Yocto Project host packages are:

```
$: sudo apt-get install gawk wget git-core diffstat unzip texinfo \
   gcc-multilib build-essential chrpath socat libsdl1.2-dev
```

i.MX layers host packages for an Ubuntu host setup are:

```
$: sudo apt-get install libsdl1.2-dev xterm sed cvs subversion \
   coreutils texi2html docbook-utils python-pysqlite2 help2man gcc \
   g++ make desktop-file-utils libgl1-mesa-dev libglu1-mesa-dev \
   mercurial autoconf automake groff curl lzop asciidoc u-boot-tools
```

3.2. Building Yocto with OpenCV 3.4.1 Support

There are two ways of building the image: with or without Qt 5 support. For some of the demos referred in this documentation, Qt 5 support is optional, but others need Qt 5 to be enabled when building the image. Whether deciding to have Qt 5 support or not, steps for building the image are the same until step 3.2.3.

3.2.1. Install the `repo` utility

This must be done just once.

```
$: mkdir ~/bin
$: curl https://storage.googleapis.com/git-repo-downloads/repo > ~/bin/repo
$: chmod a+x ~/bin/repo
$: PATH=${PATH}:~/bin
```

3.2.2. Download the Yocto Project Environment into your directory

```
$: mkdir fsl-arm-yocto-bsp
$: cd fsl-arm-yocto-bsp
$: repo init -u https://source.codeaurora.org/external/imx/imx-manifest -b imx-linux-morty -m
imx-4.9.51-8qm_beta2_ml.xml
```

```
$: repo sync
```

3.2.3. Setup the Yocto build

```
EULA=1 MACHINE=imx8qmmek DISTRO=fsl-imx-xwayland source ./fsl-setup-release.sh -b bld-xwayland
```

3.2.4. Modify conf/local.conf

Depending on the image type you want to build, you should append a few lines in the file `conf/local.conf`:

- Without Qt 5 support:

```
IMAGE_INSTALL_append += " opencv wget"
PACKAGECONFIG_append_pn-opencv_mx8 = " dnn opencl"
```

- With Qt 5 support:

```
IMAGE_INSTALL_append += " opencv wget"
PACKAGECONFIG_append_pn-opencv_mx8 = " dnn opencl qt5"
```

OpenCV Python bindings will be based on Python 3.5.

If you need Python 2, you need to disable Python 3 because they can't be both installed at the same time. In this case, the lines that need to be added to `conf/local.conf` are:

- Without Qt 5 support:

```
IMAGE_INSTALL_append += " opencv wget"
PACKAGECONFIG_append_pn-opencv_mx8 = " dnn opencl python2"
PACKAGECONFIG_remove_pn-opencv_mx8 = "python3"
```

- With Qt 5 support:

```
IMAGE_INSTALL_append += " opencv wget"
PACKAGECONFIG_append_pn-opencv_mx8 = " dnn opencl python2 qt5"
PACKAGECONFIG_remove_pn-opencv_mx8 = "python3"
```

3.2.5. Build the image

- Without Qt 5 support:

```
$: bitbake fsl-image-gui
```

- With Qt 5 support:

```
$: bitbake fsl-image-qt5
```

3.2.6. Flash the SD card image

The result of the build process is a compressed image which can be found in the following location: `tmp/deploy/images/imx8qmmek/fsl-image-<type>-imx8qmmek-<timestamp>.rootfs.sdcard.bz2`, where `<type>` is one of `gui` or `qt5`, depending on whether you've built an image with or without Qt 5 support and `<timestamp>` is the image timestamp (i.e. 20180509080732).

First step before flashing the image on the SD card is to decompress it:

```
bunzip -k -f tmp/deploy/images/imx8qmmek/fsl-image-<type>-imx8qmmek-
<timestamp>.rootfs.sdcard.bz2
```

Now flash the SD card (replace `sdX` with the actual SD card device):

```
dd if= tmp/deploy/images/imx8qmmek/fsl-image-<type>-imx8qmmek-<timestamp>.rootfs.sdcard
of=/dev/sdX bs=1M && sync
```

NOTE

ML applications require a lot of disk space for storing the input model data. By default, the SD card image is created with a small amount of extra space in the rootfs which may not be enough for ML applications. Therefore, after flashing the SD card image, it is strongly advised that you create a new ext4 partition on the SD card. The new partition will be automatically mounted at boot time in: `/run/media/mmcblk1p3/`

Example 1. Preparing the SD card

Run parted utility on the SD card device (replace sdX with the actual device):

```
sudo parted /dev/sdX
GNU Parted 3.2
Using /dev/sdX
Welcome to GNU Parted! Type 'help' to view a list of commands.
```

Print the current partition table of the device:

```
(parted) print
Model: Generic STORAGE DEVICE (scsi)
Disk /dev/sdX: 15,5GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos
Disk Flags:

Number  Start   End     Size    Type     File system  Flags
 1      8389kB  41,9MB  33,6MB  primary  fat16        lba
 2      41,9MB  2676MB  2634MB  primary  ext4
```

Create a new primary partition in the unused space of the device (use the end offset of the second partition as the start offset of the new partition and use 100% as the end offset to fill up the entire free space):

```
(parted) mkpart primary ext4 2676MB 100%
```

Print the partition table to check that the partition has been created:

```
(parted) print
Model: Generic STORAGE DEVICE (scsi)
Disk /dev/sdX: 15,5GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos
Disk Flags:

Number  Start   End     Size    Type     File system  Flags
 1      8389kB  41,9MB  33,6MB  primary  fat16        lba
 2      41,9MB  2676MB  2634MB  primary  ext4
 3      2676MB  15,5GB  12,8GB  primary  ext4         lba
```

You can now exit parted:

```
(parted) quit
Information: You may need to update /etc/fstab.
```

Create an ext4 filesystem on the new partition:

```
sudo mkfs.ext4 /dev/sdX3
```

3.2.7. Build the Yocto SDK

Yocto SDK provides a set of tools (compilers, libraries, header files) for cross-compiling code for the previously built images.

- Without Qt 5 support:

```
$: bitbake fsl-image-gui -c populate_sdk
```

- With Qt 5 support:

```
$: bitbake fsl-image-qt5 -c populate_sdk
```

After the build process finishes, it produces an installer script that can be used to install the SDK on the developing system. The script is created in: `tmp/deploy/sdk/fsl-imx-xwayland-glibc-x86_64-fsl-image-<type>-aarch64-toolchain-4.9.51-mx8-beta.sh`

3.3. How to run DNN demos

By default, DNN demos are built when building the Yocto image and are installed in the rootfs in `/usr/share/OpenCV/samples/bin/`. However, input data, model configurations, and model weights are not copied in the rootfs of the image and should be downloaded to the device before running the demos.

3.3.1. caffe_googlenet – GoogLeNet in Caffe

This demo performs image classification using a pretrained GoogLeNet network.

You can find out more information about this demo, by accessing the “Loading Caffe framework models” OpenCV tutorial: https://docs.opencv.org/3.4.1/d5/de7/tutorial_dnn_googlenet.html.

In the tutorial, some of the links seem to be broken, so you can download the network configuration file and the class names from:

- https://raw.githubusercontent.com/opencv/opencv/3.4.1/samples/data/dnn/bvlc_googlenet.prototxt
- https://raw.githubusercontent.com/opencv/opencv/3.4.1/samples/data/dnn/synset_words.txt

Example 2. image classification using a pretrained GoogLeNet network

```
# Downloading the model files
wget https://raw.githubusercontent.com/opencv/opencv/3.4.1/samples/data/dnn/synset_words.txt
wget
https://raw.githubusercontent.com/opencv/opencv/3.4.1/samples/data/dnn/bvlc_googlenet.prototxt
wget http://dl.caffe.berkeleyvision.org/bvlc_googlenet.caffemodel


# Download the test image
wget
https://raw.githubusercontent.com/opencv/opencv_extra/3.4.1/testdata/dnn/space_shuttle.jpg

# Running the demo:
/usr/share/OpenCV/samples/bin/example_dnn_caffe_googlenet --model=bvlc_googlenet.caffemodel --proto=bvlc_googlenet.prototxt --label=synset_words.txt  --image=space_shuttle.jpg
```

3.3.2. tf_inception – Inception(GoogLeNet) in TensorFlow

This application demonstrates how to load a TensorFlow model (Inception).

Example 3. how to load a TensorFlow model (Inception)

```
# Preparing the model files:
wget https://storage.googleapis.com/download.tensorflow.org/models/inception5h.zip
unzip inception5h.zip


# Download the test image
wget
https://raw.githubusercontent.com/opencv/opencv_extra/3.4.1/testdata/dnn/space_shuttle.jpg


# Running the demo:
/usr/share/OpenCV/samples/bin/example_dnn_tf_inception --model=tensorflow_inception_graph.pb --c_names=imagenet_comp_graph_label_strings.txt --image=space_shuttle.jpg
```

3.3.3. ssd_mobilenet_object_detection – Object detection with MobileNet based SSD

The demo runs MobileNet Single-Shot Detector(https://arxiv.org/abs/1704.04861) to detect objects on camera/video/image. The demo renders the original image decorated with bounding boxes around the

detected objects and confidence levels for each detected object. On i.MX 8QuadMax, only the image file variant was tested. The demo requires a Yocto image with Qt 5 enabled.

Example 4. MobileNet Single-Shot Detector to detect objects

```
# Preparing the model files:
wget https://raw.githubusercontent.com/chuanqi305/MobileNet-
SSD/master/MobileNetSSD_deploy.prototxt
wget https://raw.githubusercontent.com/chuanqi305/MobileNet-
SSD/master/MobileNetSSD_deploy.caffemodel

# Download the test image
wget https://raw.githubusercontent.com/opencv/opencv_extra/3.4.1/testdata/dnn/dog416.png

# Running the demo:
/usr/share/OpenCV/samples/bin/example_dnn_ssd_mobilenet_object_detection --
model=MobileNetSSD_deploy.caffemodel --proto=MobileNetSSD_deploy.prototxt --video=dog416.png
```

3.3.4. resnet_ssd_face – Face detection with ResNet based SSD

The demo uses Single Shot Detector (SSD) (https://arxiv.org/abs/1512.02325) with ResNet-10 architecture to detect faces on camera/video/image. On i.MX 8QuadMax, only the image file variant was tested. The demo requires a Yocto image with Qt 5 enabled.

Example 5. Face detection with ResNet based SSD

```
# Preparing the model files:
wget
https://raw.githubusercontent.com/opencv/opencv_3rdparty/dnn_samples_face_detector_20170830/r
es10_300x300_ssd_iter_140000.caffemodel
wget
https://raw.githubusercontent.com/opencv/opencv/3.4.1/samples/dnn/face_detector/deploy.protot
xt

# Download the test image
wget
https://raw.githubusercontent.com/opencv/opencv/3.4.1/samples/winrt/FaceDetection/FaceDetecti
on/Assets/group1.jpg

# Running the demo:
/usr/share/OpenCV/samples/bin/example_dnn_resnet_ssd_face --
model=res10_300x300_ssd_iter_140000.caffemodel --proto=deploy.prototxt --video=group1.jpg
```

3.3.5. ssd_object_detection – Object detection with VGG based SSD

The demo runs Single-Shot Detector(https://arxiv.org/abs/1512.02325) to detect objects on camera/video/image. The demo renders the original image decorated with bounding boxes around the detected objects and confidence levels for each detected object. On i.MX 8QuadMax, only the image file variant was tested. The demo requires a Yocto image with Qt 5 enabled.

Example 6. Object detection with VGG based SSD

Preparing the model files: download the models from https://github.com/weiliu89/caffe/tree/ssd#models. We validated the demo with `07+12+COCO SSD300*`.

```
Extract from the archive VGG_VOC0712_SSD_300x300_ft_iter_120000.caffemodel and
deploy.prototxt
tar --strip 4 -xvf models_VGGNet_VOC0712_SSD_300x300_ft.tar.gz --wildcards "*.caffemodel"
"*/deploy.prototxt"

# Download the test image
wget https://raw.githubusercontent.com/opencv/opencv_extra/3.4.1/testdata/dnn/dog416.png

# Running the demo:
/usr/share/OpenCV/samples/bin/example_dnn_ssd_object_detection --
model=VGG_VOC0712_SSD_300x300_ft_iter_120000.caffemodel --proto=deploy.prototxt --
video=dog416.png
```

3.3.6. yolo_object_detection – Object detection with YOLO

This demo performs object detection using You only look once (YOLO)-Detector (https://arxiv.org/abs/1612.08242). It detects objects on camera/video/image. On i.MX 8QuadMax, only the image file variant was tested.

You can find out more information about this demo by accessing the "Loading Caffe framework models" OpenCV tutorial: https://docs.opencv.org/3.4.1/da/d9d/tutorial_dnn_yolo.html.

This demo requires a Yocto image with Qt 5 enabled.

Example 7. Object detection with YOLO

```
# Preparing the model files:
wget https://pjreddie.com/media/files/yolov2.weights
wget https://raw.githubusercontent.com/pjreddie/darknet/master/cfg/yolov2.cfg
wget https://raw.githubusercontent.com/pjreddie/darknet/master/data/coco.names

# Download the test image
wget https://raw.githubusercontent.com/pjreddie/darknet/master/data/person.jpg

# Running the demo
/usr/share/OpenCV/samples/bin/example_dnn_yolo_object_detection --cfg=yolov2.cfg --
model=yolov2.weights --class_names=coco.names --source=person.jpg
```

3.3.7. faster_rcnn – Object detection with Faster R-CNN

This demo runs Faster-RCNN and R-FCN object detection models with OpenCV. The demo renders the original image decorated with bounding boxes around the detected objects and confidence levels for each detected object. The demo requires a Yocto image with Qt 5 enabled.

Example 8. Object detection with Faster R-CNN

```
# Preparing the model files:
wget -O faster_rcnn_models.tgz
https://dl.dropboxusercontent.com/s/o6ii098bu51d139/faster_rcnn_models.tgz?dl=0
tar --strip 1 -xvf faster_rcnn_models.tgz
wget
https://raw.githubusercontent.com/opencv/opencv_extra/3.4.1/testdata/dnn/faster_rcnn_zf.proto
txt
```

```
wget
https://raw.githubusercontent.com/opencv/opencv_extra/3.4.1/testdata/dnn/faster_rcnn_vgg16.prototxt

# Download the test image
wget https://raw.githubusercontent.com/opencv/opencv_extra/3.4.1/testdata/dnn/dog416.png

# Running the demo (2 options):
/usr/share/OpenCV/samples/bin/example_dnn_faster_rcnn --model=ZF_faster_rcnn_final.caffemodel
--proto=faster_rcnn_zf.prototxt -i=dog416.png

/usr/share/OpenCV/samples/bin/example_dnn_faster_rcnn --
model=VGG16_faster_rcnn_final.caffemodel --proto=faster_rcnn_vgg16.prototxt -i=dog416.png
```

3.3.8. colorization – Image colorization

The goal of this demo is to colorize a black & white image. The resulting image is displayed on the screen. This demo requires a Yocto image with Qt 5 enabled.

Example 9. Colorization-image colorization

```
# Preparing the model files:
wget
http://eecs.berkeley.edu/~rich.zhang/projects/2016_colorization/files/demo_v2/colorization_release_v2.caffemodel
wget
https://raw.githubusercontent.com/richzhang/colorization/master/colorization/models/colorization_deploy_v2.prototxt

# Download the test image
wget https://raw.githubusercontent.com/opencv/opencv/3.4.1/samples/data/basketball1.png

# Running the demo:
/usr/share/OpenCV/samples/bin/example_dnn_colorization --
model=colorization_release_v2.caffemodel --proto=colorization_deploy_v2.prototxt --
image=basketball1.png
```

3.3.9. fcn_sgmem – Image segmentation with FCN

This demo performs image segmentation. It displays an image where each segment is represented with a different color, based on the segment type. The demo requires a Yocto image with Qt 5 enabled.

Except for the input image, all input parameters are expected to be in the current folder.

Example 10. fcn_sgmem-Image segmentation with FCN

```
# Preparing the model files:
wget https://raw.githubusercontent.com/opencv/opencv_extra/3.4.1/testdata/dnn/fcn8s-heavy-pascal.prototxt
wget http://dl.caffe.berkeleyvision.org/fcn8s-heavy-pascal.caffemodel
wget https://raw.githubusercontent.com/opencv/opencv/3.4.1/samples/data/dnn/pascal-classes.txt

# Download the test image
wget https://raw.githubusercontent.com/opencv/opencv/3.4.1/samples/data/dnn/rgb.jpg
```

```
# Running the demo:
/usr/share/OpenCV/samples/bin/example_dnn_fcn_semsegm rgb.jpg
```

3.4. How to run Non-NN demos

Non-Neural Net demos are built when building the Yocto image and are installed on path: /usr/share/OpenCV/samples/bin/ .

3.4.1. Introduction to SVM

This example demonstrates how to create and train an SVM model using training data. Once the model is trained, labels for test data are predicted. The full description of the example can be found in *tutorial_introduction_to_svm*. For displaying the result, a Yocto image with Qt 5 enabled is required.

```
#Run demo, show result on screen (requires Qt 5 support)
/usr/share/OpenCV/samples/bin/example_tutorial_introduction_to_svm
```

Result:

- The code opens an image and shows the training examples of both classes. The points of one class are represented with white circles, and other class uses black points.
- The SVM is trained and used to classify all the pixels of the image. This results in a division of the image into a blue region and a green region. The boundary between both regions is the optimal separating hyperplane.
- Finally, the support vectors are shown using gray rings around the training examples.

3.4.2. SVM for Non-Linearly separable data

This example deals with non-linearly separable data and shows how to set parameters of SVM with linear kernel for this data. For more details, go to this link: *SVM_non_linearly_separable_data*.

```
#Run demo, show result on screen (requires Qt 5 support)
/usr/share/OpenCV/samples/bin/example_tutorial_non_linear_svms
```

Result:

- The code opens an image and shows the training data of both classes. The points of one class are represented with light green, the other class uses light blue points.
- The SVM is trained and used to classify all the pixels of the image. This results in a division of the image into blue green regions. The boundary between both regions is the separating hyperplane. Since the training data is non-linearly separable, some of the examples of both classes are misclassified; some green points lay on the blue region and some blue points lay on the green one.
- Finally, the support vectors are shown using gray rings around the training examples.

3.4.3. Introducing PCA

Principal Component Analysis (PCA) is a statistical method that extracts the most important features of a dataset. In this tutorial you will learn how to use PCA to calculate the orientation of an object. For more details, check the OpenCV tutorial: *Introduction_to_PCA*.

```
#Run demo, show result on screen (requires Qt 5 support)
cd /usr/share/OpenCV/samples/bin
/usr/share/OpenCV/samples/bin/example_tutorial_introduction_to_pca
```

Result:

- The code opens an image (loaded from ../data/pca_test1.jpg), finds the orientation of the detected objects of interest, and then visualizes the result by drawing the contours of the detected objects of interest, the center point, and the x-axis, y-axis regarding the extracted orientation.

3.4.4. Logistic regression

In this sample, logistic regression is used for prediction of two characters (0 or 1) from an image. First, every image matrix is reshaped from its original size of 28x28 to 1x784. A logistic regression model is created and trained on 20 images. After training, the model can predict labels of test images. Source code is located on the link *logistic_regression* and can be run by typing the following command:

```
#Run sample:
cd /usr/share/OpenCV/samples/bin
/usr/share/OpenCV/samples/bin/example_cpp_logistic_regression
```

Result:

- Training and test data are shown and comparison between original and predicted labels is displayed. (The trained model reaches 95% accuracy.)

4. Advanced model deployment

Neural networks go through two main phases: training and inference. During the training phase, the network learns and its weights are continuously adjusted. Once the training phase is done, the network is ready for deployment. This is the inference phase when we use the network with the pre-trained weights to process new data.

OpenCV DNN is concerned only with the second phase: the inference. It expects a trained model as an input and is capable of deploying the network to inference and returning a result after processing data we provide to it.

Neural network models can be stored in various file formats. Typically, each machine learning framework (e.g. Caffe, TensorFlow, PyTorch) has its own format. One important aspect to pay attention to is the fact that the training phase requires additional operations and is more complex than inferencing. This means that a model must be prepared before being ready for the inference phase.

For most of the popular neural networks, pre-trained and inference ready models are typically made available. No additional preparation is needed. This is the case for the neural network models used by the OpenCV DNN samples.

For other networks or in case you design and train your own custom network, you also must prepare the model for the inference phase. The next two sections describe the process of preparing a model for inference, using two of the most popular formats: Caffe and TensorFlow.

4.1. Caffe

A neural net is defined by its design/architecture. Caffe saves its architecture into a `*.protoxt` (text file) file while weights are saved into a `*.caffemodel` (binary file). Both of these files are needed for deployment using the OpenCV DNN module.

4.1.1. Modifying the network for deployment

NOTE

As previously mentioned in this section' introduction, if the network is already provided with correct deploy.prototxt file, this step is not necessary.

A trained network model needs the following modifications in order to be deployable:

1. Remove the data layers used for dataset management during training. These data layers consist of labels for the data and are no longer needed.
2. Check the network graph and remove all layers having before-mentioned data layers as input.
3. Setup the network to accept data and output the results.

4.1.2. Example

As an example, let's use a Convolutional Neural Net called ExampleNet, trained on MNIST dataset to do a classification of hand-written numbers into 10 classes. MNIST is a common ML dataset used for classifying handwritten digits; dataset source: http://yann.lecun.com/exdb/mnist/.

This Example Network has two data layers: one related to training and the second one for testing. Each of them refers to the lmdb file, where the images are stored. As the task of this NN is to do classification, the output is a softmax layer. One can see that there is also a loss layer which is used for training purposes. It computes how much the current result differs from the required one. Based on this difference, the weights will be updated. This method is called backpropagation. The accuracy layer is used for the testing phase and NN performance evaluation. We don't need any of them for inference. What we need are layers like Convolutions, MaxPooling, etc. Our example net.prototxt might look similar to:

```
name: "ExampleNet"
layer {
  name: "mnist"
  type: "Data"
  top: "data"
  top: "label"
```

```
  include {
    phase: TRAIN
  }
  ...
}
layer {
  name: "mnist"
  type: "Data"
  top: "data"
  top: "label"
  include {
    phase: TEST
  }
  ...
  data_param {
    source: "examples/mnist/mnist_test_lmdb"
    batch_size: 32
    backend: LMDB
  }
}
layer {
  name: "conv1"
  type: "Convolution"


...

layer {
  name: "accuracy"
  type: "Accuracy"
  bottom: "fc1"
  bottom: "label"
  top: "accuracy"
  include {
    phase: TEST
  }
}
layer {
  name: "loss"
  type: "SoftmaxWithLoss"
  bottom: "fc1"
  bottom: "label"
  top: "loss"
}
```

Following the previously stated steps, we will modify our network `.prototxt` file in order to deploy it using OpenCV DNN module.

4.1.2.1. Remove data layers

Data layers are no longer needed, as we will not be providing any labelled data. The following code will be erased.

```
layer {
  name: "mnist""
```

```
  type: "Data"
  top: "data"
  top: "label"
  include {
    phase: TRAIN
  }
  ...
}
layer {
  name: "mnist""
  type: "Data"
  top: "data"
  top: "label"
  include {
    phase: TEST
  }
  ...
  data_param {
    source: "examples/mnist/mnist_test_lmdb"mnist_test_lmdb"
    batch_size: 32: 32
    backend: LMDB
  }
}
```

4.1.2.2. Remove any layer that is dependent upon data labels

The Accuracy Layer and the SoftmaxWithLoss Layer are expecting labels. As we mentioned before, no labels are provided during inference. These layers must be removed as well:

```
...
layer {
  name: "accuracy"
  type: "Accuracy"
  bottom: "fc1"
  bottom: "label"
  top: "accuracy"
  include {
    phase: TEST
  }
}
layer {
  name: "loss"
  type: "SoftmaxWithLoss"
  bottom: "fc1"
  bottom: "label"
  top: "loss"
}
```

4.1.2.3. Set up the network to accept data and return results

This step is as simple as adding the following first line in *.prototxt. The rest will be solved from the OpenCV side.

```
input: "data"
```

The neural network prior to this step was computing a Softmax with Loss, and we still want to use the Softmax so we have classification results. We will add a new layer to the end of our file to produce a Softmax output (basically replacing the previous one, “SoftmaxWithLoss”). We can do this because there are no weights related to this layer, it is just a computation on top of a fully connected layer called fc1 in this example. The added layer should look like this:

```
layer {
  name: "loss"
  type: "Softmax"
  bottom: "fc1"
  top: "loss"
}
```

After all of the before-mentioned modifications, we should name the file: `deploy.prototxt`. It should look like:

```
input: "data"
name: "ExampleNet""

layer {
  name: "conv1"
  type: "Convolution"


...
layer {
  name: "loss"
  type: "Softmax""
  bottom: "fc1"
  top: "loss"
}
```

4.1.2.4. Using the new network in OpenCV DNN module

Now that our deployment network is complete, we can use it in OpenCV. For this we will need `readNetFromCaffe` and `blobFromImage` functions from OpenCV. For more details see *documentation*.

4.1.2.5. Known issues

```
channel_pruning_VGG-16_3C4x ends up with an error:
OpenCV(3.4.1) Error: Assertion failed ((idx == -1 && size() == 1) || (idx >= 0 && idx <
size())) in get, file /tmp/opencv-20180307-60086-ryy1b3/opencv-
3.4.1/modules/dnn/include/opencv2/dnn/dnn.inl.hpp, line 88
Traceback (most recent call last):
  File "run_caffe.py", line 35, in <module>
    result = net.forward()
cv2.error: OpenCV(3.4.1) /tmp/opencv-20180307-60086-ryy1b3/opencv-
3.4.1/modules/dnn/include/opencv2/dnn/dnn.inl.hpp:88: error: (-215) (idx == -1 && size() ==
1) || (idx >= 0 && idx < size()) in function get
```

http://answers.opencv.org/question/187725/when-loading-caffe-vgg-16-i-get-error-215-idx-1-size-1-idx-0-idx-size-in-function-get/

It is necessary to replace ambiguous pad or `kernel_size` with `pad_h, pad_w`, `kernel_h`, `kernel_w`.

4.2. TensorFlow

The process of deploying a TensorFlow model is related to several operations and modifications. First, we must remove all unnecessary operations related to training (e.g. loss function computations, batch normalization, nodes for dataset management). Second, we want to do optimizations such as operation fusion or removing parts of the graph which are never reached.

Compared to Caffe, TensorFlow (TF) is a more complex framework and there are more issues related to model loading and parsing. In this section, a best-known approach is described. In case model loading still ends up with an error, we suggest reaching out to *OpenCV community support*. Another option is to use a TensorFlow to Caffe model converter.

This section starts by describing the most common TF model formats, then the steps for preparing a model for inference are explained. In the end, there is a list of common issues and solutions.

4.2.1. TensorFlow model formats

There are two main formats used for storing and sharing TensorFlow models: checkpoint files and Protobuf files.

Checkpoint files target the TensorFlow design and training phase, when the user can modify the network architecture or launch new training sessions. TensorFlow introduced checkpoint files to allow the user to save/restore a network configuration and weights. At the moment of writing this document, a TensorFlow checkpoint is a set of three binary files. Note however that checkpoint files are TensorFlow specific, therefore the file format is not standard and can change at any time. Regardless of the format, a checkpoint set of files aims to be a complete description of a model, therefore it contains both the model structure and the model weights.

Protobuf files, on the other hand, target the inference phase, when the training is already done and the network needs to be prepared for deployment. Regarding the format, there are two types of Protobuf files: text files (`*.pbtxt`) and binary files (`*.pb`). A Protobuf file can contain only model architecture information, or also the weights; `*.pbtxt` files are typically used for model architecture, while `*.pb` files are preferred for storing the complete model description due to reduced file size.

For TensorFlow models, OpenCV DNN expects as input a `*.pb` file containing both the model architecture and the model weights. A second requirement besides the file format is for the model to be prepared and ready for inference. More details about how to prepare a model for deployment in the following section.

4.2.2. Model preparation for inference

The diagram below describes the steps needed for preparing a TensorFlow model for inference. The process is TensorFlow specific and uses TensorFlow tools. It starts from the description of the pre-trained model (architecture and model weights), it passes through three main phases and then it generates a `.pb` file with a model ready for deployment. This `.pb` file can now be imported in OpenCV DNN.

There are two main options for providing the pre-trained network and they rely on using TensorFlow checkpoint files:

- Use the TensorFlow checkpoint file containing the weights and the TensorFlow checkpoint file containing the model description
- Use the TensorFlow checkpoint file containing the weights and a .pbtxt file containing model description.

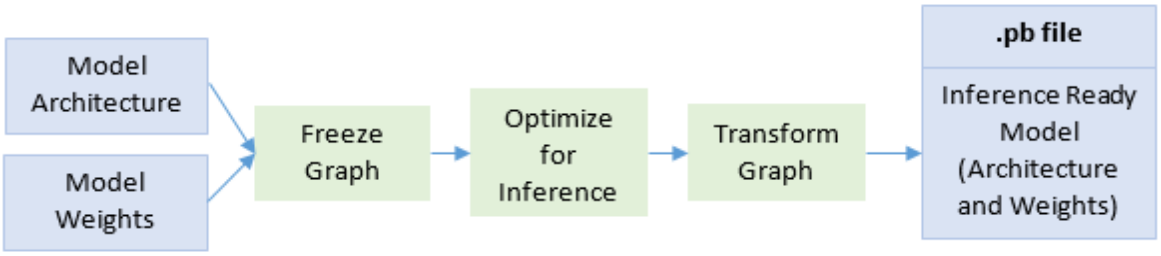

Figure 1. **TensorFlow model preparation for inference**

The table below gives an overview of what each model preparation phase aims to do. The last two columns include links to corresponding TensorFlow scripts and TensorFlow documentation.

Note that some of these tools may not get installed automatically with TensorFlow. In this case, you can build them from source.

Table 1. **TensorFlow model preparation phases**

	Goal	TF Python Script	TF Doc Link
Freeze Graph	• Remove special metadata • Package model and weights into one file • Output: GraphDef object in ProtoBuf format	freeze_graph.py	Link
Optimize for Inference	• Remove other training specific operations • Optimize the graph	optimize_for_inference_lib.py	Link

Transform Graph	• Custom optimizations for complex models (e.g. removing specific nodes, changing node order)	TransformGraph	Link

4.2.3. Known Issues

4.2.3.1. tf.nn.dropout

When the model uses `tf.nn.dropout` nodes, dropout nodes are not removed using model freezing. This leads to a series of errors associated with unknown layers once a model is loaded, even though it is processed with all suggested scripts:

```
OpenCV(3.4.1) Error: Unspecified error (Unknown layer type Shape in op dropout/Shape) in
populateNet, file /tmp/opencv-20180307-60086-ryy1b3/opencv-
3.4.1/modules/dnn/src/tensorflow/tf_importer.cpp, line 1582 Traceback (most recent call
last):
  File "run_tensorflow.py", line 24, in <module>
    net = cv2.dnn.readNetFromTensorflow('optimized_model.pb')
cv2.error: OpenCV(3.4.1) /tmp/opencv-20180307-60086-ryy1b3/opencv-
3.4.1/modules/dnn/src/tensorflow/tf_importer.cpp:1582: error: (-2) Unknown layer type Shape
in op dropout/Shape in function populateNet
```

One solution is to retrain the model using `tf.layers.dropout`, which adds `is_training` flags and is correctly recognized during model freezing. Another possible workaround is to remove dropout layers and reconnect the graph manually or by writing your own script.

NOTE

This might be very tedious work.

Example:

```
n: name, t: operation type, i: input
...
n: MaxPool, t: MaxPool i: [u'Relu']
n: dropout/Shape, t: Shape i: [u'MaxPool']
n: dropout/random_uniform/min, t: Const i: []
n: dropout/random_uniform/max, t: Const i: []
n: dropout/random_uniform/RandomUniform, t: RandomUniform i: [u'dropout/Shape']
n: dropout/random_uniform/sub, t: Sub i: [u'dropout/random_uniform/max',
u'dropout/random_uniform/min']
n: dropout/random_uniform/mul, t: Mul i: [u'dropout/random_uniform/RandomUniform',
u'dropout/random_uniform/sub']
n: dropout/random_uniform, t: Add i: [u'dropout/random_uniform/mul',
u'dropout/random_uniform/min']
n: dropout/add, t: Add i: [u'Placeholder_1', u'dropout/random_uniform']
n: dropout/Floor, t: Floor i: [u'dropout/add']
n: dropout/div, t: RealDiv i: [u'MaxPool', u'Placeholder_1']
n: dropout/mul, t: Mul i: [u'dropout/div', u'dropout/Floor']
```

```
n: Conv2D_1, t: Conv2D i: [u'dropout/mul', u'Variable_1']
...
It should end up as:
n: name, t: operation type, i: input
...
n: MaxPool, t: MaxPool i: [u'Relu']
n: Conv2D_1, t: Conv2D i: [u'MaxPool', u'Variable_1']

...
```

If this issue occurs frequently, it is possible to write a script which will automatically take care of it.

4.2.3.2. Wrong number of channels

```
OpenCV(3.4.1) Error: Assertion failed (ngroups > 0 && inpCn % ngroups == 0 && outCn
% ngroups == 0) in getMemoryShapes, file /tmp/opencv-20180307-60086-ryy1b3/opencv-
3.4.1/modules/dnn/src/layers/convolution_layer.cpp, line 234

Traceback (most recent call last):
  File "run_tensorflow.py", line 32, in <module>
    result = net.forward()

cv2.error: OpenCV(3.4.1) /tmp/opencv-20180307-60086-ryy1b3/opencv-
3.4.1/modules/dnn/src/layers/convolution_layer.cpp:234: error: (-215) ngroups > 0 &&
inpCn % ngroups == 0 && outCn % ngroups == 0 in function getMemoryShapes
```

This error occurs when an image with wrong number of channels is fed into a Neural Net.

5. Google Cloud Interoperability – Train on Cloud, Inference on Edge

5.1. Requirements

- **Host OS:** Ubuntu 16.04
- **Additional software:** Google Cloud SDK
- **Internet Connectivity**

5.2. Google Account and Resources Initialization

5.2.1. Create a Google Cloud account and log in using GCP (Google Cloud Platform Console)

- Follow the instructions on https://cloud.google.com/ to Create a Google Cloud account
- Install Google Cloud SDK https://cloud.google.com/sdk/docs/quickstart-debian-ubuntu (Ubuntu 16.04)
- Initialize the GCP (Google Cloud Platform Console) by issuing the command '*gcloud init*', this will send you a link to log in with your Google account. After logging in using this link, a verification code will be generated in the web browser which must be used in the gcloud console.

After authenticating, the user has the option to create a project (this step is necessary to run jobs on the cloud). If the account was created earlier and there are already existing projects on this account, an option for picking a cloud project from the available list will be shown. Considering this is the first run of the command, we choose to create a project (consider using a longer id number for the project name to avoid conflicts with other project names already in use).

5.2.2. Create project & set its properties

After logging into your Google account from the console, the next step is to create a project and set some properties for this project. The project may be created also via the web interface or via the `gcloud init` command, in which case you can skip step 5.2.2.1.

5.2.2.1. Create a new project

```
gcloud projects create test-project-23042018
```

Please use a longer id number to avoid the following error:

```
ERROR: (gcloud.projects.create) Project creation failed. The project ID you specified is
already in use by another project. Please try an alternative ID.
```

5.2.2.2. Check the project list associated with the current account

```
gcloud projects list
```

5.2.2.3. Set the project id of the GCP to operate on by default

```
gcloud config set project test-project-23042018
export PROJECT_ID=$(gcloud config list project --format "value(core.project)")
```

5.2.3. Set some properties for the project

5.2.3.1. Enable Google Compute Engine API

```
gcloud services enable compute.googleapis.com
```

5.2.3.2. Enable billing for the project

The following command will show the ACCOUNT_ID:

```
gcloud alpha billing accounts list
```

Run the following command, using the ACCOUNT_ID from the previous command:

```
gcloud alpha billing accounts projects link $PROJECT_ID --billing-account=ACCOUNT_ID
```

NOTE

In some situations, enabling billing from console doesn't work, in which case, you may use the following alternative:

Go to https://cloud.google.com/. Top left you should see:

Clicking the left button will expand a menu which contains the Billing option. Clicking on this option would show the possibility of enabling the billing for this project.

5.2.4. Create a storage bucket for input/output data on the cloud

Create the bucket using the following command:

```
gsutil mb -p $PROJECT_ID gs://$PROJECT_ID-mlengine
```

If the following error message shows up, it means that billing was not enabled for the project and you should review step 5.2.3.2:

```
AccessDeniedException: 403 The project to be billed has not configured billing.
AccessDeniedException: 403 The project to be billed is associated with an absent billing account.
```

You can now check the content of the new bucket (initially it should be empty):

```
export BUCKET_NAME=${PROJECT_ID}-mlengine
gsutil ls gs://$BUCKET_NAME/
```

5.2.5. Set the region (it depends on your location)

```
export REGION=europe-west1
gcloud config set compute/zone $REGION
```

5.2.6. Set the job name and the output path for storing the results

```
export JOB_NAME=train_test
export OUTPUT_PATH=gs://$BUCKET_NAME/$JOB_NAME
```

5.3. Launching the Training Job on the Cloud

Assuming the local test folder is `script_dir` and inside it the training script called `train_script` which is a python script that contains the code for training a neural network written in TensorFlow.

For better organization, you may create 2 separate folders inside the cloud bucket (easiest way to create folders inside a cloud bucket is via Google Cloud Platform Console):

- Input dataset folder: `$OUTPUT_PATH/dataset/`
- Output model folder (for checkpoints & `.pbtxt` file): `$OUTPUT_PATH/model_dir/`

5.3.1. Upload dataset to the cloud

Assuming `MNIST_data` local folder contains MNIST dataset archives, you will upload the dataset on the cloud using the following command:

```
gsutil cp -r MNIST_data/* $OUTPUT_PATH/dataset/
```

5.3.2. Update input & output paths in the training script

The actual path in the cloud environment will be sent to the application through the `job-dir` option of `gcloud ml-engine jobs submit training` command as shown below in step *5.3.3*. Because of this, it must be read inside the script using `argparse`.

You should modify your script by adding the `argparse` utility:

```
import argparse

parser = argparse.ArgumentParser()
parser.add_argument('--job-dir',
                    required=True,
                    type=str,
                    help="""\
                    Google Cloud Platform path for checkpoints,
                    datasets and pbtxt file""")

parse_args, unknown = parser.parse_known_args()
```

Before sending a job to the cloud, the input and output paths of the script should point to cloud storage (cloud bucket) instead of the local storage which is not accessible from the cloud. When using the `argparse` utility as presented before, the cloud storage path is stored in `parse_args.job_dir`.

Inside the script you must use the input/output paths created in paragraph *5.3*:

- Input dataset path: `parse_args.job_dir + "/dataset/ "`
- Output model path (for checkpoints & .pbtxt file): `parse_args.job_dir + "/model_dir/"`

5.3.3. Starting the job

Assuming the training script is in `script_dir/` folder, please issue the following command one folder up `script_dir/` folder (assumes the environment variables were set in previous steps: `OUTPUT_PATH`, `REGION`):

```
gcloud ml-engine jobs submit training $JOB_NAME \
--stream-logs \
--job-dir $OUTPUT_PATH \
--runtime-version 1.4 \
--module-name script_dir.train_script \
--package-path script_dir \
--region $REGION \
--scale-tier STANDARD_1
```

5.3.4. Monitor job completion

To monitor a running job, Google Cloud offers a Jobs section at ML Engine service which could be accessed in a browser when clicking on top left (there is a button near Google Cloud Platform and project). The first time the Cloud Machine Learning Engine API is enabled for the current project, this operation could take a few minutes. When clicking on Jobs section, all the jobs from the current project are displayed. Each job is marked with an icon indicating if the job has finished or is still running.

5.4. Downloading the results

The following command will download all the checkpoint files and the "`.pbtxt`" file of the model from the cloud:

```
gsutil cp -r $OUTPUT_PATH/* <local_path>
```

Having these files locally, you can follow the steps in section *4.2.2* to prepare the model for inference.

5.5. Run inference locally (on the edge)

Once you have downloaded the model and prepared it for inference, you can use it for running inference locally. You only need to take care to use the local path to the transformed model in your program or inference script.

6. FAQ

Q: I receive the following error message when running a demo:

```
"OpenCV(3.4.1) Error: Unspecified error (The function is not implemented. Rebuild the library with Windows, GTK+ 2.x or Carbon support. If you are on Ubuntu or Debian, install libgtk2.0-dev and pkg-config, then re-run cmake or configure script) in cvShowImage"
```

A: You are running a demo that tries to render some image on screen and your image is not built with Qt 5 support.

Q: I receive the following error message when running a demo:

```
“Failed to create display (No such file or directory)”
```

A: You try to run the demo from within a ssh session on the i.MX card. You must set the following environment variable:

```
export XDG_RUNTIME_DIR=/run/user/0
```

Q: Is OpenCV running on the CPU or GPU?

A: On i.MX 8QuadMax, OpenCV 3.4.1 runs on CPU and uses Arm NEON acceleration. OpenCV also has an OpenCL backend for the GPU, but it hasn’t been tested yet.

Q: Why are you not using TensorFlow Lite?

A: TensorFlow Lite is an embedded inference engine supporting only TensorFlow Lite models. OpenCV, on the other hand, is a compute vision library which provides both traditional machine learning algorithms and neural network algorithms. OpenCV’s inference engine supports a wider set of input model formats: TensorFlow, Caffe 1, Torch/PyTorch. For more details about supported models and supported layers, check the official OpenCV DNN wiki page.

Q: How can one do neural network model training using OpenCV?

A: OpenCV DNN model is basically an inference engine. It does not aim to provide any model training capabilities. For training, one should use dedicated solutions, such as machine learning frameworks.

7. TensorFlow layers supported in OpenCV 3.4.1

OpenCV 3.4.1 supports the TensorFlow layers from the list below. The list has been generated by parsing the content of `void TFImporter::populateNet(Net dstNet) from opencv\modules\dnn\src\tensorflow\modelimporter\tf_importer.cpp`.

- Conv2D
- SpaceToBatchND
- DepthwiseConv2dNative
- BiasAdd

- Add
- MatMul
- Reshape
- Flatten
- Transpose
- Const
- LRN
- Concat
- ConcatV2
- MaxPool
- AvgPool
- Placeholder
- Split
- Slice
- Mul
- Pad
- FusedBatchNorm
- Conv2DBackpropInput
- BlockLSTM
- ResizeNearestNeighbor
- L2Normalize
- PriorBox
- DetectionOutput
- Softmax
- Abs
- Tanh
- Sigmoid
- Relu
- Elu
- Identity
- Relu6

8. Revision history

Revision number	Date	Substantive changes
1	07/2018	Initial release

.

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

While NXP has implemented advanced security features, all products may be subject to unidentified vulnerabilities. Customers are responsible for the design and operation of their applications and products to reduce the effect of these vulnerabilities on customer’s applications and products, and NXP accepts no liability for any vulnerability that is discovered. Customers should implement appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, COOLFLUX, EMBRACE, GREENCHIP, HITAG, I2C BUS, ICODE, JCOP, LIFE VIBES, MIFARE, MIFARE CLASSIC, MIFARE DESFire, MIFARE PLUS, MIFARE FLEX, MANTIS, MIFARE ULTRALIGHT, MIFARE4MOBILE, MIGLO, NTAG, ROADLINK, SMARTLX, SMARTMX, STARPLUG, TOPFET, TRENCHMOS, UCODE, Freescale, the Freescale logo, AltiVec, C 5, CodeTEST, CodeWarrior, ColdFire, ColdFire+, C Ware, the Energy Efficient Solutions logo, Kinetis, Layerscape, MagniV, mobileGT, PEG, PowerQUICC, Processor Expert, QorIQ, QorIQ Qonverge, Ready Play, SafeAssure, the SafeAssure logo, StarCore, Symphony, VortiQa, Vybrid, Airfast, BeeKit, BeeStack, CoreNet, Flexis, MXC, Platform in a Package, QUICC Engine, SMARTMOS, Tower, TurboLink, and UMEMS are trademarks of NXP B.V. All other product or service names are the property of their respective owners. AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, µVision, Versatile are trademarks or registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved. Oracle and Java are registered trademarks of Oracle and/or its affiliates. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.

© 2018 NXP B.V.

Document Number: AN12224
Rev. 0
07/2018