EIQ Machine Learning Software Application Development based on OpenCV Neural Network Framework on QorIQ Layerscape Platforms

NXP created eIQ machine learning software for QorIQ Layerscape applications processors, a set of ML tools which allows developing and deploying ML applications on the QorIQ Layerscape family of devices.
OpenCV is an open-source computer vision library. It offers a unitary solution for both the neural network inference (DNN module) and the standard machine learning algorithms (ML module). It includes many computer vision functions, making it easier to build complex machine learning applications in a short amount of time and without being dependent on other libraries.
This document describe applications YOLO object detection, Image segmentation, Image colorization, Image classification, Human pose estimation and Text detection developed based on OpenCV DNN framework.

1. NXP eIQ software introduction

The NXP eIQ machine learning software development environment provides a shell script to install machine learning applications targeted at NXP QorIQ Layerscape processors. The NXP eIQ software is concerned only with neural networks inference and standard machine-learning algorithms, leaving neural network training to other specialized software tools and dedicated hardware. The NXP eIQ is continuously expanding to include data-acquisition and curation tools and model conversion for a wide range of NN frameworks and inference engines, such as TensorFlow Lite, Arm NN, and Arm Compute Library.
The current version of NXP eIQ software delivers machine learning enablement by providing ML support in LSDK for the two QorIQ Layerscape processors LS1046A and LX2160A. The NXP eIQ software contains these main features:
• OpenCV 4.0.1
• Arm Compute Library 20.08
• Arm NN 20.08
• TensorFlow Lite 2.2.0
• Onnxruntime 1.1.2
• PyTorch 1.6.0
eIQ software architecture is described as the following.

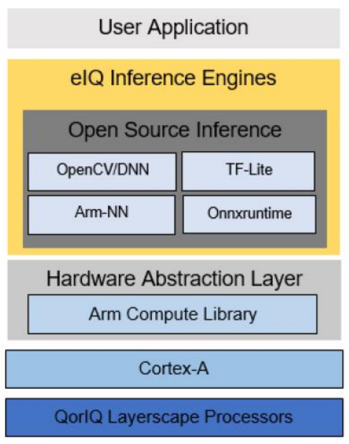


Figure 1. NXP eIQ machine learing software

2. OpenCV Software Introduction

OpenCV is an open-source computer vision library. One of its modules (called ML) provides traditional machine learning algorithms. Another important module in the OpenCV is the DNN, which provides support for neural network algorithms.

OpenCV offers a unitary solution for both the neural network inference (DNN module) and the standard machine learning algorithms (ML module). It includes many computer vision functions, making it easier to build complex machine learning applications in a short amount of time and without being dependent on other libraries.

OpenCV has wide adoption in the computer vision field and is supported by a strong and active community. The key algorithms are specifically optimized for various devices and instructions sets. For QorIQ Layerscape processor, OpenCV uses the Arm NEON acceleration. The Arm NEON technology is an advanced SIMD (Single Instruction Multiple Data) architecture extension for the Arm Cortex-A series. The Arm NEON technology is intended to improve multimedia user experience by accelerating the audio and video encoding/decoding, user interface, 2D/3D graphics, or gaming. The Arm NEON can also accelerate the signal-processing algorithms and functions to speed up applications such as the audio and video processing, voice and facial recognition, computer vision, and deep learning.

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 the supported models and layers, see the official OpenCV DNN wiki page.

On the other hand, the OpenCV ML module contains classes and functions for solving machine learning problems such as 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 more information, see the official reference manual and machine learning overview. For more details about OpenCV 4.0.1, see the official OpenCV change log web page.

3. Building EIQ OpenCV Components in LSDK

Download LSDK flexbuild tarball from www.nxp.com/lsdk

```
    $ tar xvzf flexbuild_<version>.tgz
    $ cd flexbuild_<version>
    $ source setup.env
```

Build EIQ OpenCV components into rootfs filesystem.

```
[root@fbubuntu flexbuild]$ flex-builder -i mkrfs
[root@fbubuntu flexbuild]$ flex-builder -c opencv -a arm64
```

Modify OpenCV DNN application source code in flexbuild_lsdk2004/packages/apps/eiq/opencv/samples/dnn, and rebuild opencv again.

```
[root@fbubuntu flexbuild]$ flex-builder -c opencv -a arm64
[root@fbubuntu flexbuild]$ flex-builder -i merge-component -B eiq
[root@fbubuntu flexbuild]$ flex-builder -i packrfs
```

Deploy LSDK distro with eIQ OpenCV images to SD card:

```
[root@fbubuntu flexbuild]$ flex-installer -i pf -d /dev/sdx
[root@fbubuntu flexbuild]$ flex-installer -r build/images/rootfs_<version>_LS_arm64_main.tgz
-d /dev/sdx
```

eIQ components binaries and libraries will be installed in target rootfs folder:

```
    /usr/local/bin
    /usr/local/lib
```

4. OpenCV DNN demo Applications

OpenCV DNN demos are installed in this folder: /usr/local/bin/
However, the input data, model configurations, and model weights are not located in this folder, because of their size. These files must be downloaded to the device before running the demos:
Download the opencv_extra.zip package at this link:
github.com/opencv/opencv_extra/tree/4.0.1, unpack the file to the directory <home_dir>.

```
$ unzip 4.0.1.zip
```

The input images, model configurations for some OpenCV examples are available at this folder:
<home_dir>/opencv_extra-4.0.1/testdata/dnn
The models files can be obtained with two options:

- Option 1: Run opencv_extra download script to download all opencv_extra dependencies,

```
$ python download_models.py
```

The script downloads the NN models, configuration files, and input images for some OpenCV examples. This operation may
take a long time.

• Option 2: Direct download required models using wget command

For example, download caffe model of SqueezeNet,

```
$ wget https://raw.githubusercontent.com/DeepScale/SqueezeNet/
b5c3f1a23713c8b3fd7b801d229f6b04c64374a5/SqueezeNet_v1.1/squeezenet_v1.1.caffemo
del
```

4.1 YOLO object detection Application

This demo performs the object detection using the You Only Look Once (YOLO) detector (arxiv.org/abs/1612.08242). It detects objects in an image.

Demo dependencies (taken from the "opencv_extra" package):

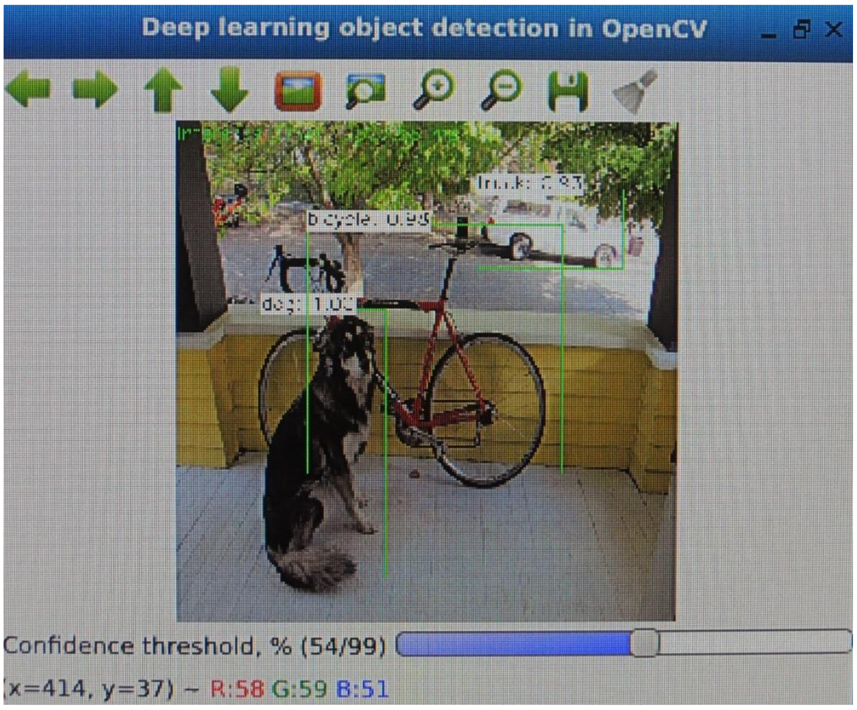
• dog416.png
• yolov3.weights
• yolov3.cfg

Other demo dependencies:

• models.yml
• object_detection_classes_yolov3.txt from /usr/local/OpenCV/.

Running the C++ example with the image input from the default location:

```
# example_dnn_object_detection --width=1024 --height=1024 --scale=0.00392
--input=dog416.png --rgb --zoo=/usr/local/OpenCV/models.yml
--classes=/usr/local/OpenCV/data/dnn/object_detection_classes_yolov3.txt yolo
```


Deep learning object detection in OpenCV
truck: 0.93
bicycle: 0.99
dog: 1.00
Confidence threshold, % (54/99)
(x=414, y=37) ~ R:58 G:59 B:51

4.2 Image segmentation Application

The image segmentation means dividing the image into groups of pixels based on some criteria. You can do this grouping based on color, texture, or some other criteria that you choose.

Demo dependencies (taken from the "opencv_extra" package):

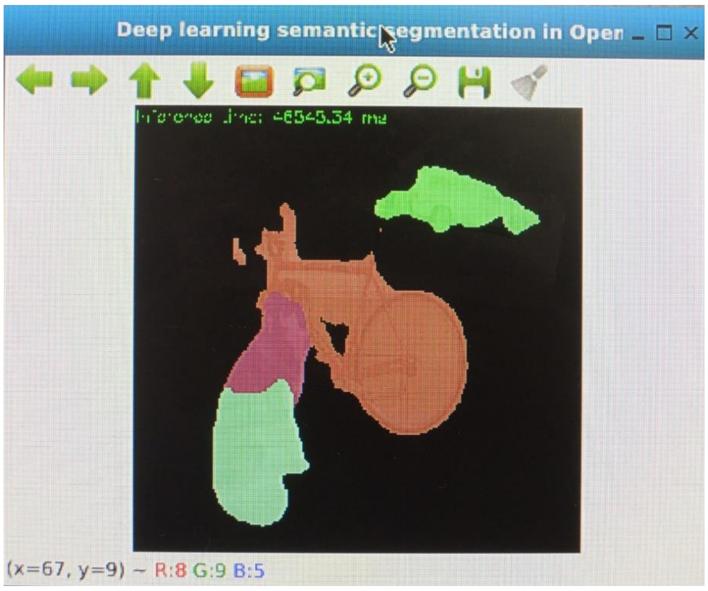
• dog416.png
• fcn8s-heavy-pascal.caffemodel
• fcn8s-heavy-pascal.prototxt

Other demo dependencies:

• models.yml

Running the C++ example with the image input from the default location:

```
# example_dnn_segmentation --width=500 --height=500 --rgb --mean=1 --input=dog416.png -
-zoo=/usr/local/OpenCV/models.yml fcn8s
```


Deep learning semantic segmentation in Open
(x=67, y=9) ~ R:8 G:9 B:5

```
Inference time: 46545.34 ms
```

4.3 Image colorization Application

This example demonstrates the recoloring of grayscale images using DNN. The demo supports input images only.

Demo dependencies (taken from the "opencv_extra" package):

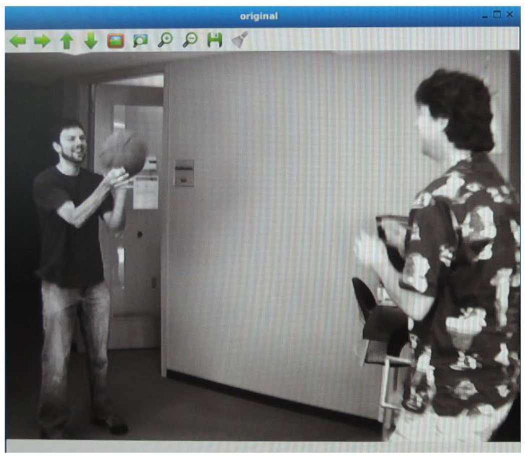
• colorization_release_v2.caffemodel
• colorization_deploy_v2.prototxt

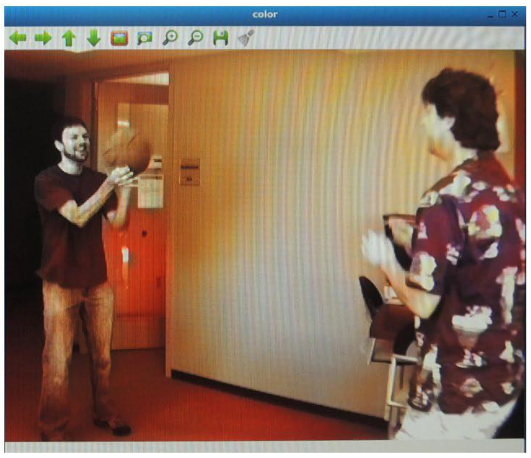
Other demo dependencies from /usr/local/OpenCV/data/.

• basketball1.png

Running the C++ example with the image input from the default location:

```
# example_dnn_colorization --model=colorization_release_v2.caffemodel --
proto=colorization_deploy_v2.prototxt --image=/usr/local/OpenCV/data/basketball1.png
```





4.4 Image classification Application

This demo performs image classification using a pre-trained SqueezeNet network.

Demo dependencies (taken from <home_dir>/opencv_extra-4.0.1/testdata/dnn):

- dog416.png
- squeezenet_v1.1.prototxt
- squeezenet_v1.1.caffemodel

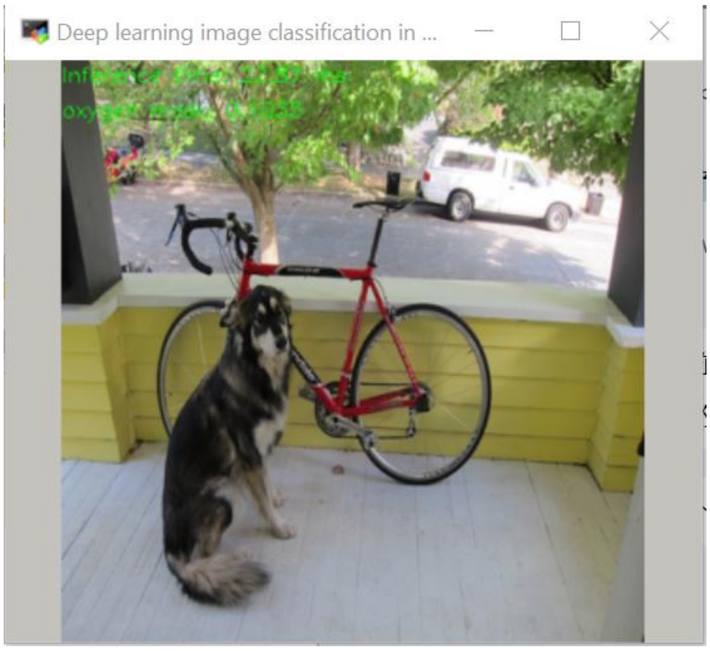
```
$ wget
https://raw.githubusercontent.com/DeepScale/SqueezeNet/b5c3f1a23713c8b3fd7b801d229f6
b04c64374a5/SqueezeNet_v1.1/squeezenet_v1.1.caffemodel
```

Other demo dependencies:

- classification_classes_ILSVRC2012.txt from /usr/local/OpenCV/data/dnn/
- models.yml from /usr/local/OpenCV/

Running the C++ example with the image input from the default location:

```
$ example_dnn_classification --input=./opencv_extra-4.0.1/testdata/dnn/
dog416.png --zoo=/usr/local/OpenCV/models.yml
--classes=/usr/local/OpenCV/data/dnn/classification_classes_ILSVRC2012.txt squeezenet
```



4.5 Human pose estimation Application

This application demonstrates the human or hand pose detection with a pretrained OpenPose DNN. The demo supports only input images, not the live camera input.

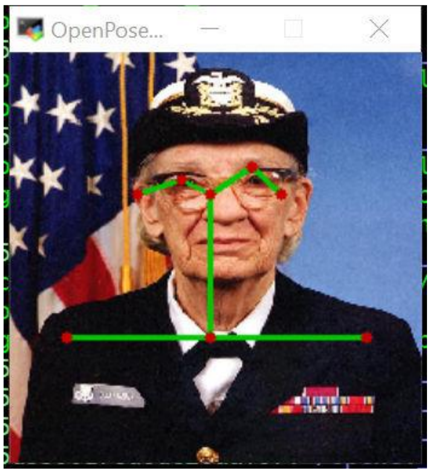
Demo dependencies (taken from <home_dir>/opencv_extra-4.0.1/testdata/dnn):

- grace_hopper_227.png
- openpose_pose_coco.prototxt
- openpose_pose_coco.caffemodel

```
$ wget
http://posefs1.perception.cs.cmu.edu/OpenPose/models/pose/coco/pose_iter_440000.caffemodel -O openpose_pose_coco.caffemodel
```

Running the C++ example with the image input from the default location:

```
$ example_dnn_openpose --model=openpose_pose_coco.caffemodel
--proto=./opencv_extra-4.0.1/testdata/dnn/openpose_pose_coco.prototxt
--image=./opencv_extra-4.0.1/testdata/dnn/grace_hopper_227.png --width=227
--height=227
```



4.6 Text detection Application

This demo is used for text detection in the image using the EAST algorithm.
Demo dependencies (using wget as option):

- frozen_east_text_detection.pb

Download and unpack the model file,

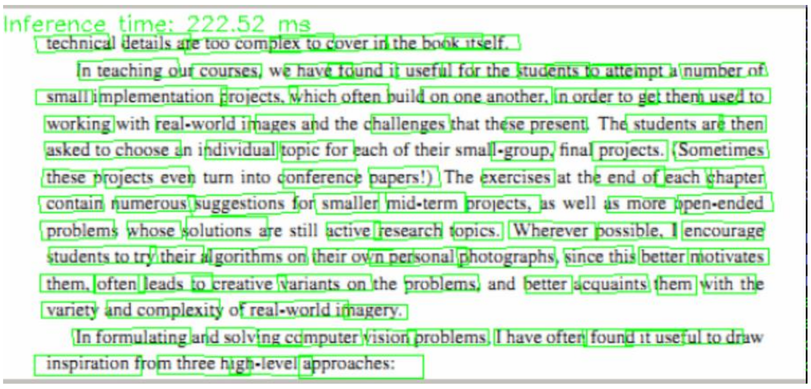
```
$ wget https://www.dropbox.com/s/r2ingd0l3zt8hxs/frozen_east_text_detection.tar.gz?dl=1 -
O frozen_east_text_detection.tar.gz
$ tar xvf frozen_east_text_detection.tar.gz
```

Other demo dependencies(taken from /usr/local/OpenCV/data/):

- imageTextN.png

Running the C++ example with the image input from the default location:

```
$ example_dnn_text_detection --model=frozen_east_text_detection.pb --
input=/usr/local/OpenCV/data/imageTextN.png
```



Inference time: 222.52 ms

technical details are too complex to cover in the book itself.

In teaching our courses, we have found it useful for the students to attempt a number of small implementation projects, which often build on one another, in order to get them used to working with real-world images and the challenges that these present. The students are then asked to choose an individual topic for each of their small-group, final projects. (Sometimes these projects even turn into conference papers!) The exercises at the end of each chapter contain numerous suggestions for smaller mid-term projects, as well as more open-ended problems whose solutions are still active research topics. Wherever possible, I encourage students to try their algorithms on their own personal photographs, since this better motivates them, often leads to creative variants on the problems, and better acquaints them with the variety and complexity of real-world imagery.

In formulating and solving computer vision problems, I have often found it useful to draw inspiration from three high-level approaches: