eIQ i.MX Hands-on Irvine, CA

Version 1.0, August 27, 2019

Table of Contents

Chapter 1. Overview

This document was completely based on eIQ Machine Learning (ML) Software. The goal of this training is to introduce the user to the ML software by running examples that were based on it. The materials used on this hands-on can be found on eIQ Sample Apps repo or the eIQ Community page.

1.1. AGENDA

- **Face Recognition using TensorFlow Lite** (the one we will be focusing)

  This section uses Haar Feature-based Cascade Classifiers for real time face detection. After detecting the faces the demo uses TensorFlow for recognizing the faces.

  - Compiling the Face Recognition Application
  - Running the Face Recognition Application (inst)

- **OpenCV Single Shot Detection (SSD)** (optional)

  This section uses the Deep Neural Network (DNN) module for object detection. The first example uses a file input and the second part uses camera as input.

  - Deploying the OpenCV SSD Applications
  - Running the OpenCV SSD Applications (inst)

- **Hand Written Digits using MNIST Dataset** (optional)

  This section focuses on a comparison of inference time between different models (TensorFlow and Caffe) for handwritten digits recognition.

  - Compiling the Handwritten Digits Application
  - Running the MNIST Applications (inst)

- **Object Recognition using Arm NN** (optional)

  This section shows how to run the prebuilt demo from eIQ to recognize objects using a *MIPI* camera.

  - Preparing the Object Recognition Application
  - Running the Object Recognition Application (inst)

The procedures described in this document target a *GNU/Linux* (`Ubuntu 18.04`).

Chapter 2. Hands-on Kit

Your `hands-on kit` contains:

- i.MX 8MM EVK Board
- USB Cable (micro-B to standard-A)
- USB Type-C to A Adapter
- USB Type-C 45W Power Delivery Supply
- IMX-MIPI-HDMI Daughter Card
- MINISASTOCSI Camera Daughter Card
- 2×Mini-SAS Cable
- AI/ML BSP flashed into the SD Card
- USB Mouse
- Ethernet Cable
- HDMI Monitor Cable
- Monitor

IF any hardware is missing, **please inform the presenters**.

Steps requiring software downloads were previous prepared and placed in the `~/Desktop/files` folder in the ***host GNU/Linux PC***.

The sections with the `inst` abbreviation were previous installed to save time.

The sections with `configured` mention were previous installed to save time.

Training usually refers to the process of preparing a machine learning model to be useful by feeding it data from which it can learn.

Inference refers to the process of taking a model that's already been trained and using that trained model to make useful predictions.

Chapter 3. Preparing the i.MX 8MM EVK Board

The following section describes the steps to boot the *i.MX 8MM EVK*.

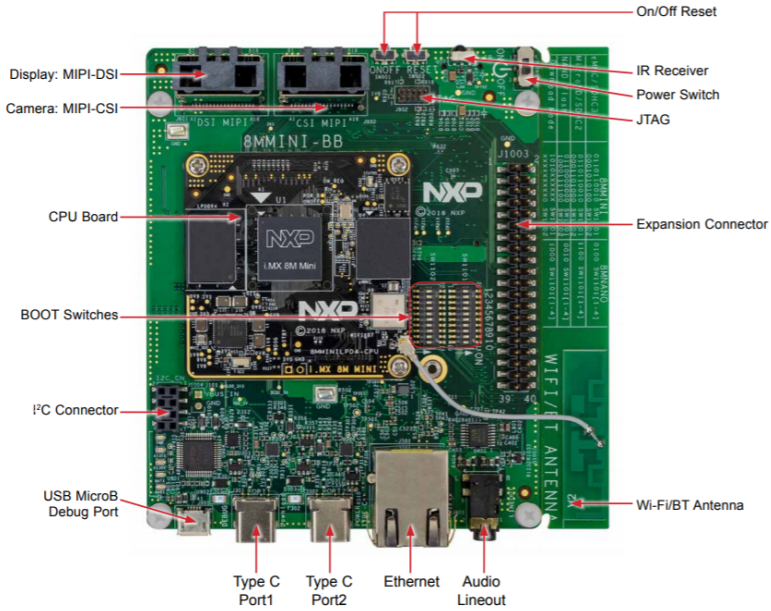

Figure 1. i.MX 8MM EVK Board Top View

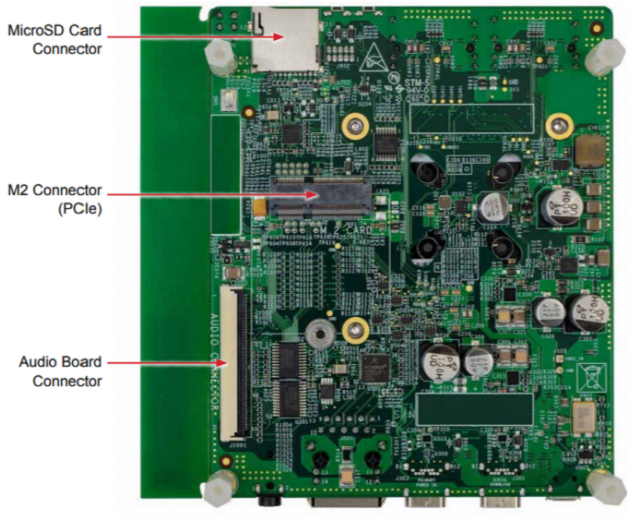

Figure 2. i.MX 8MM EVK Board Back View

1. Connect the *IMX-MIPI-HDMI Daughter Card* to the *Mini-SAS Cable* and into connector labeled *DSI MIPI* (J801) and then connect the *HDMI Monitor Cable* into it.

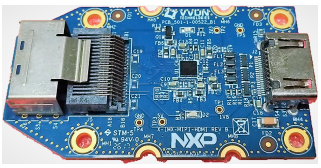

Figure 3. IMX-MIPI-HDMI Daughter Card

2. Connect the *MINISASTOCSI Camera Daughter Card* to the *Mini-SAS Cable* and into connector labeled *CSI MIPI* (J802).

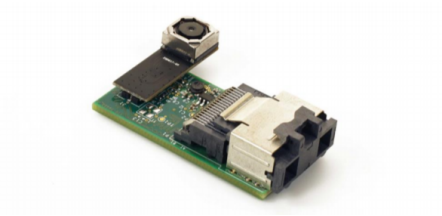

Figure 4. MINISASTOCSI Camera Daughter Card

3. Connect the *micro-B* end of the supplied *USB Cable* into *Debug UART* port J901.
4. Connect the other end of the cable to the *host GNU/Linux PC*.
5. Insert the *MicroSD Card* into the *MicroSD Card slot* J701 in the back board side. To boot the board from the *MicroSD Card*, change the *Boot Switches* SW1101 and SW1102 (Figure 5) according to the table below:

Table 1. Boot Device Setting

BOOT Device	SW1101	SW1102
MicroSD/uSDHC2	0110110010	0001101000

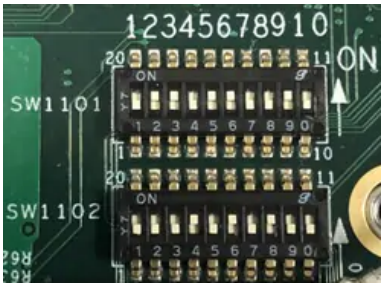

Figure 5. Boot Device Settings

6. Connect the *Power Supply Cable* to the *Power Connector* J302 and power **on** the board by flipping the switch SW101.

For more details: i.MX 8MM EVK Getting Started.

Nick Bostrom - Machine intelligence is the last invention that humanity will ever need to make.

Chapter 4. Getting Started with the Prerequisites

4.1. Minicom

`Minicom` was previous installed and configured to the *host GNU/Linux PC*. To learn how to communicate with the board through serial, **follow** the Communicate with the i.MX Board.

4.1.1. Configure the Minicom (configured)

1. Open a terminal on the *host GNU/Linux PC*, then:

   On Host GNU/Linux PC Terminal execute the following commands:

   ```
   $ minicom -s ①
   ```

   ① Open the `Minicom` settings.

2. Go to `serial port setup` and change the settings to:
   - Serial Device as: `/dev/ttyUSB1`
   - Bps/Par/Bits as: `115200 8N1`
   - Hardware Flow Control : `No`
   - Software Flow Control : `No`
3. Then choose `Save setup as dfl`, then `exit`.

4.1.2. Communicate with the i.MX Board

1. Connect the serial cable to the *i.MX board* and the *host GNU/Linux PC*, then:

   On Host GNU/Linux PC Terminal execute the following commands:

   ```
   $ minicom ①
   ```

   ① Open `Minicom` with the default settings.

2. Turn the target board power switch to **on**, when it reaches the login prompt, enter: `root`.

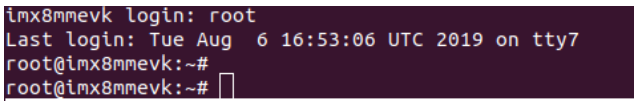

Figure 6. Prompt log in

`Minicom` is just one serial console communication, you can try others at home.

4.2. Toolchain Installation (inst)

The 4.19 toolchain must be installed to *cross-compile* the applications in the *host GNU/Linux PC*.

1. Open a terminal on the *host GNU/Linux PC*, then:

   On Host GNU/Linux PC Terminal execute the following commands:

   ```
   $ cd ~/Desktop/files/toolchain ①
   $ chmod +x fsl-imx-internal-xwayland-glibc-x86_64-test-internal-qt5-aarch64-toolchain-4.19-thud.sh ②
   ```

   ① entering the `toolchain` folder where is located the cross-compiler.

   ② attributing execution permission to the toolchain.

2. Install the toolchain **accepting** all the **default settings** and enter the password: `123456`

   On Host GNU/Linux PC Terminal execute the following commands:

   ```
   $ sh ./fsl-imx-internal-xwayland-glibc-x86_64-test-internal-qt5-aarch64-toolchain-4.19-thud.sh ①
   ```

   ① running the toolchain to be installed.

This toolchain is used to compile the `C++` examples (*face* and *digits*).

4.3. Network Configuration

This step explains how to establish a *host GNU/Linux PC* and *i.MX board* network connection.

1. Connect the Ethernet cable to the *i.MX board* and the *host GNU/Linux PC*.

   The ***i.MX board*** is configured with a static IP: `192.168.0.100`

   The ***host GNU/Linux PC*** is configured with a static IP: `192.168.0.200`

2. Export the *i.MX board* IP to transfer the files in the next steps:

   On Host GNU/Linux PC Terminal execute the following commands:

   ```
   $ IMX_INET_ADDR=192.168.0.100 ①
   $ export IMX_INET_ADDR ②
   ```

   ① creating an environment variable to hold the board IP

   ② exporting this environment variable.

This environment variable will only be set for this shell and its child processes. If the shell/terminal is closed and starts a new one, this setting won't be retained.

4.4. Flashing SD Card (flashed)

The provided SD Card is flashed with an image built following the eIQ Machine Learning Software.

1. **IF** you want to learn how to flash the image to the SD Card, follow the next steps:

You need root privileges (sudo) to use `dd` for writing to `/dev/sd<x>`.

<x> refers to the SD Card device. You can check it using `lsblk` command.

On host GNU/Linux PC Terminal execute the following commands:

```
$ cd ~/Desktop/files/image ①
$ bunzip2 -f eIQ-4.19-Irvine-Hands-on-image.sdcard.bz2 ②
$ dd if=eIQ-4.19-Irvine-Hands-on-image.sdcard of=/dev/sd<x> status=progress bs=1M && sync ③
```

① entering to the `image` folder where is located the built image.

② decompressing the built image.

③ writing the built image to the SD Card.

The SD Card image has all the required files to run the applications.

4.5. Home Readings

Here are a few documents to read at home and learn more about this training:

1. eIQ Machine Learning Software
2. eIQ Sample Apps Repository
3. eIQ Community Page

4.6. Starting with Machine Learning Applications

- ☑ Minicom Communication
- ☑ Toolchain Installation
- ☑ Network Configuration
- ☑ Flashing SD Card

Once done with the above steps, the next sections show information and steps on where to get the *source code* and *models*, how to transfer files to the board, how to prepare *models* for *inference*, how to compile/run applications and more. **Hope you enjoy it!**

Chapter 5. Face Recognition using TensorFlow Lite

This application uses Haar Feature-based Cascade Classifiers for real time face detection. The pre-trained Haar Feature-based Cascade Classifiers for face is already contained in OpenCV.

5.1. Compiling the Face Recognition Application

1. Compile the Face Recognition application on *host GNU/Linux PC* (it can be compiled on i.MX):

   On Host GNU/Linux PC Terminal execute the following commands:

   ```
   $ cd ~/Desktop/files/face ①
   $ source /opt/fsl-imx-internal-xwayland/4.19-thud/environment-setup-aarch64-poky-linux ②
   $ make ③
   ```

   ① entering the `face` folder where is located the application.

   ② execute the toolchain to set up the environment variables.

   ③ running make to compile the application.

2. Copy the binary and the pre-trained Haar Feature-based Cascade to the *i.MX board*:

   On Host GNU/Linux PC Terminal execute the following commands:

   ```
   $ scp FaceRecognition haarcascade_frontalface_alt.xml root@${IMX_INET_ADDR}:/opt/face ①
   ```

   ① copy the files to the board thru network.

5.2. Running the Face Recognition Application (inst)

The MobileFaceNets application is re-trained with a smaller batch size and input size to get a higher performance on a *host GNU/Linux PC*. The trained model is loaded as a source file in this demo.

1. **Run the application**:

   On i.MX Board Minicom Terminal execute the following commands:

   ```
   root@imx8mmevk:/opt/face# ./FaceRecognition -c 0 -h 0.85 ①
   ```

   ① running the demo where `-c` is used to specify the camera index, `0` means the MIPI/USB camera is mounted on `/dev/video0`, and `-h` is a threshold for the prediction score.

2. When the face is detected, enter the name using keyboard. Then, click on `Add new person`.

Once new faces are added, it will create a folder named `data` in current directory. If you want to remove the new face from the data set, just delete it in `data`.

Chapter 6. OpenCV Single Shot Detection (SSD)

This application was based on SSD: Single Shot MultiBox Detector and Caffe SSD Implementation and uses OpenCV *Deep Neural Network* (DNN) model which is basically an inference engine.

6.1. Deploying the OpenCV SSD Applications

1. Deploy the OpenCV SSD files and model to the *i.MX Board*:

   On Host GNU/Linux PC Terminal execute the following commands:

   ```
   $ cd ~/Desktop/files/opencv ①
   $ scp -r file camera model/ media/ root@${IMX_INET_ADDR}:/opt/opencv ②
   ```

   ① entering the `opencv` folder where is located the application.

   ② copying the `file`, `camera` and `model` folder to the board thru network.

2. The folder structure must be equal to:

   On i.MX Board Minicom Terminal execute the following commands:

   ```
   ├──── file ①
   ├──── camera ②
   ├──── media ③
   │     └──── test.jpg ④
   ├──── model ⑤
   │     ├──── MobileNetSSD_deploy.caffemodel ⑥
   │     └──── MobileNetSSD_deploy.prototxt ⑦
   ```

   ① python script to run inference on file.

   ② python script to run inference on camera.

   ③ folder to hold the images to run inference on it.

   ④ image test that is used to run inference.

   ⑤ folder to hold the model.

   ⑥ caffe model

   ⑦ protocol buffer

   

   The applications of this section were written using `Python3`.

6.2. Running the OpenCV SSD Applications (inst)

6.2.1. File Based

This application runs inference in the images located in `media` folder.

1. **IF** you want to copy new images to the `media/` folder (**do it at home**):

   On Host GNU/Linux PC Terminal execute the following commands:

   ```
   $ scp <picture> root@${IMX_INET_ADDR}:/opt/opencv/media ①
   ```

   ① copying new images to the board thru network.

2. **Run the application**:

   On i.MX Board Minicom Terminal execute the following commands:

   ```
   root@imx8mmevk:/opt/opencv# ./file ①
   ```

   ① running the application (the board contains one image to be tested).

   

   This above line **does not show an output message**.

   The processed images are available in the `media-labeled/` folder. See before and after labeling:

   

   Figure 7. Comparison

3. To display the labeled image use `gstreamer` pipeline:

   

   Replace `<image>` to the correct image name to be opened.

   On i.MX Board Minicom Terminal execute the following commands:

   ```
   root@imx8mmevk:/opt/opencv/media-labeled# gst-launch-1.0 filesrc location=<image> \
   ! jpegdec ! imagefreeze ! autovideosink ①
   ```

   ① opens an image using gstreamer pipelines.

6.2.2. MIPI Camera Based

This application uses MIPI camera to run real time inference on each captured frame:

1. **Run the application**:

   On i.MX Board Minicom Terminal execute the following commands:

```
root@imx8mmevk:/opt/opencv# ./camera ①
```

   ① running the application.

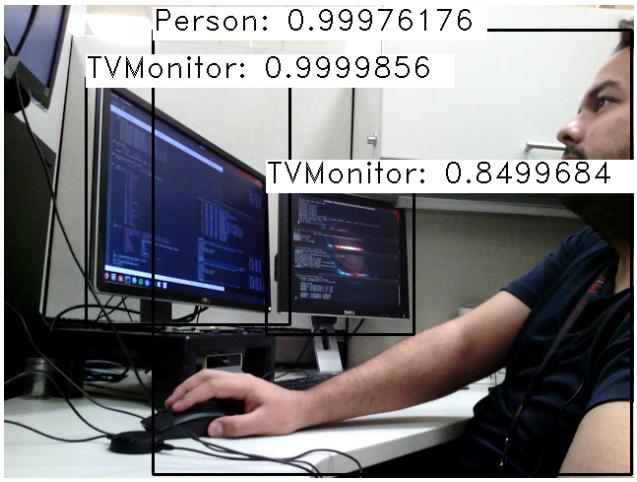

Figure 8. Real Time Inference

The list of objects that can be recognized with this model: *Aeroplane*, *Bicycle*, *Bird*, *Boat*, *Bottle*, *Bus*, *Car*, *Cat*, *Chair*, *Cow*, *Dining Table*, *Dog*, *Horse*, *Motorbike*, *Person*, *Potted Plant*, *Sheep*, *Sofa*, *Train* and *TV Monitor*.

You can easily change this model and run the inference using this same code.

This application is **not** using GPU acceleration.

Chapter 7. Handwritten Digits using MNIST Dataset

This application provides a comparison of a Caffe and TensorFlow models for *Handwritten Digit Recognition*. The data set used is called `MNIST` from Yann Lecun.

Follow a sample of this dataset:

Figure 9. MNIST data set

These images are processed in 28x28 resolution pixels.

The `MNIST` is a large database of handwritten digits commonly used for training various image processing systems.

7.1. Compiling the Handwritten Digits Application

1. Compile the MNIST application on *host GNU/Linux PC*:

   On Host GNU/Linux PC Terminal execute the following commands:

   ```
   $ cd ~/Desktop/files/mnist ①
   $ source /opt/fsl-imx-internal-xwayland/4.19-thud/environment-setup-aarch64-poky-linux ②
   $ ${CXX} -Wall -Wextra -O3 -std=c++14 caffe_inference.cpp -o caffe_inference -larmnn -larmnnCaffeParser ③
   $ ${CXX} -Wall -Wextra -O3 -std=c++14 tensorflow_inference.cpp -o tensorflow_inference -larmnn -larmnnTfParser ④
   ```

   ① entering the `mnist` folder where is located the application.

   ② exporting the toolchain.

   ③ compiling the caffe example source code.

   ④ compiling the tensorflow example source code.

2. Deploy the built files to the board:

   On Host GNU/Linux PC Terminal execute the following commands:

```
$ scp -r caffe_inference tensorflow_inference data/ model/ root@${IMX_INET_ADDR}:/opt/mnist ①
```

① copy the files to the board thru network.

3. The folder structure must be equal to:

On i.MX Board Minicom Terminal execute the following commands:

```
├──── caffe_inference ①
├──── tensorflow_inference ②
├──── data ③
│    ├──── t10k-images-idx3-ubyte ④
│    └──── t10k-labels-idx1-ubyte ⑤
└──── model ⑥
    ├──── lenet_iter_9000.caffemodel ⑦
    ├──── optimized_mnist_tf.pb ⑧
    ├──── simple_mnist_tf.pb ⑨
    └──── simple_mnist_tf.prototxt ⑩
```

① caffe binary.

② tensorflow binary.

③ folder to hold the dataset.

④ images dataset.

⑤ label dataset.

⑥ folder to hold the models

⑦ caffe model.

⑧ optimized tensorflow model.

⑨ tensorflow model.

⑩ protocol buffer.

The input images are in the binary form and can be found at the t10k-images-idx3-ubyte.gz package from Yann Lecun.

7.2. Running the MNIST Applications (inst)

These tests run inference on MNIST dataset (`Actual`) and outputs the inference results (`Predict`).

1. **Run the applications**:

   Enter as an argument the number of predictions (`0` to `9999`) to be inferenced.

   a. The `caffe_inference` uses a `Caffe` model for inference:

   On i.MX Board Minicom Terminal execute the following commands:

```
root@imx8mmevk:/opt/mnist# ./caffe_inference 10 ①
[0]     Caffe >> Actual: 7  Predict: 7  Time: 0.0336484s
[1]     Caffe >> Actual: 2  Predict: 2  Time: 0.028399s
[2]     Caffe >> Actual: 1  Predict: 1  Time: 0.0283713s
[3]     Caffe >> Actual: 0  Predict: 0  Time: 0.0284133s
[4]     Caffe >> Actual: 4  Predict: 4  Time: 0.0280637s
[5]     Caffe >> Actual: 1  Predict: 1  Time: 0.0281574s
[6]     Caffe >> Actual: 4  Predict: 4  Time: 0.0285136s
[7]     Caffe >> Actual: 9  Predict: 9  Time: 0.0283779s
[8]     Caffe >> Actual: 5  Predict: 5  Time: 0.0283902s
[9]     Caffe >> Actual: 9  Predict: 9  Time: 0.0283282s
Total Time: 0.296081s   Successfull: 10  Failed: 0
```

   ① running the application passing 10 test examples.

   b. The `tensorflow_inference` uses a `TensorFlow` model for inference:

   On i.MX Board Minicom Terminal execute the following commands:

```
root@imx8mmevk:/opt/mnist# ./tensorflow_inference 10 ①
[0]     Tensor >> Actual: 7  Predict: 7  Time: 0.00670075s
[1]     Tensor >> Actual: 2  Predict: 2  Time: 0.00377025s
[2]     Tensor >> Actual: 1  Predict: 1  Time: 0.0036785s
[3]     Tensor >> Actual: 0  Predict: 0  Time: 0.0036815s
[4]     Tensor >> Actual: 4  Predict: 4  Time: 0.00372875s
[5]     Tensor >> Actual: 1  Predict: 1  Time: 0.003669s
[6]     Tensor >> Actual: 4  Predict: 4  Time: 0.00367825s
[7]     Tensor >> Actual: 9  Predict: 9  Time: 0.0036955s
[8]     Tensor >> Actual: 5  Predict: 6  Time: 0.00367488s      FAILED
[9]     Tensor >> Actual: 9  Predict: 9  Time: 0.0036025s
Total Time: 0.0414569s  Successfull: 10  Failed: 1
```

   ① running the application passing 10 test examples.

   Notice that `Caffe` model is slower than `TensorFlow`, however more accurate than the latter.

Change the argument to compare further results between models.

Chapter 8. Object Recognition using Arm NN

This application recognizes objects within *Flash Cards* **black borders** using *Arm NN Framework*.

8.1. Preparing the Object Recognition Application

1. Copy the Arm NN application on *host GNU/Linux PC*:

   On Host GNU/Linux PC Terminal execute the following commands:

   ```
   $ cd ~/Desktop/files/armnn ①
   $ scp -r object_recognition_camera data/ model/ parser.sh labels.csv root@${IMX_INET_ADDR}:/opt/armnn/ ②
   ```

   ① entering the `armnn` folder where is located the application.

   ② copying files to the board thru network.

2. The folder structure must be equal to:

   On i.MX Board Minicom Terminal execute the following commands:

   ```
   ├──── object_recognition_camera ①
   ├──── labels.csv ②
   ├──── parser.sh ③
   ├──── data ④
   │     ├──── Cat.jpg ⑤
   │     ├──── Dog.jpg ⑥
   │     └──── shark.jpg ⑦
   ├──── models ⑧
   │     └──── inception_v3_2016_08_28_frozen.pb ⑨
   ```

   ① python script to run inference on flash cards.

   ② file containing labels that can be recognized.

   ③ script to parser the results.

   ④ folder to hold the example images.

   ⑤ cat example image.

   ⑥ dog example image.

   ⑦ shark example image.

   ⑧ folder to hold the model.

   ⑨ tensorflow model.

This application is only an extension of `TfInceptionV3-Armnn`.

8.2. Running the Object Recognition Application (inst)

This section shows how to use the `TfInceptionV3-Armnn` test from eIQ for general object detection.

The list of all object detection supported by this model can be found here.

1. **Run the application**:

   On i.MX Board Minicom Terminal execute the following commands:

   ```
   root@imx8mmevk:/opt/armnn# ./object_recognition_camera ①
   ```

   ① running the application.

   

   This runs the `TfInceptionV3-Armnn` test and parses the inference results to return any recognized object.

2. Show the *Flash Cards* to the camera and wait for the message:
   - `Image captured, wait`

   

   The *Flash Cards* **MUST NOT** be **twisted** or **curved**.

3. After a few seconds, the demo returns the detected object.

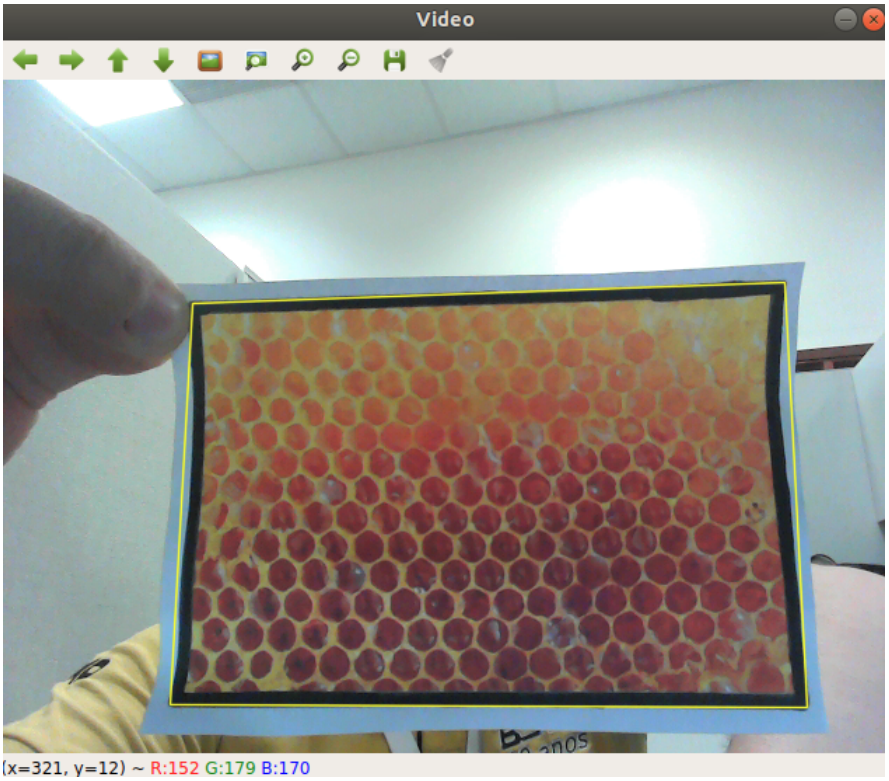

Figure 10. Captured Flash Card

This can return `False` if the image was not correctly captured. In this case, try showing the flash card again.

Chapter 9. Revision History

Version	Author	Changes	Date
1.0	Diego Dorta	Original release 1.0	08/27/2019

Chapter 10. Appendix

THIS SECTION IS NOT PART OF THE HANDS-ON TRAINING!

10.1. Removing Root Privileges

```
$ sudo chown -R $USER ~/Desktop/files
$ sudo adduser $USER dialout
$ sudo chgrp dialout /etc/minicom/minirc.dft
$ sudo chmod g+rw /etc/minicom/minirc.dft
$ reboot
```

10.2. Network Configuration

10.2.1. i.MX Board Static IP

Configuring a static IP:

1. Create a configuration file under `/lib/systemd/network`:

   i.MX Board Minicom Terminal

   ```
   # cat >> /lib/systemd/network/10-eth0.network << EOF
   [Match]
   Name=eth0
   [Network]
   Address=192.168.0.100/24
   Gateway=192.168.0.1
   EOF
   ```

2. Restart the `systemd-networkd service`:

   i.MX Board Minicom Terminal

   ```
   # systemctl restart systemd-networkd.service
   ```

3. Verify using `ip addr`.

10.2.2. Host GNU/Linux PC Static IP

1. Create and edit the generated file adding the following lines:

   Host GNU/Linux PC Terminal

   ```
   $ sudo netplan generate
   $ sudo gedit /etc/netplan/01-netcfg.yaml
   ```

   Host GNU/Linux PC Terminal

   ```
   # Let NetworkManager manage all devices on this system
   network:
     version: 2
     renderer: NetworkManager
     ethernets:
       eno1:
         dhcp4: no
         dhcp6: no
         addresses: [192.168.0.200/24, ]
         gateway4:  192.168.0.1
         nameservers:
           addresses: [8.8.8.8, 8.8.4.4]
   ```

2. Run the following command to apply the changes:

   Host GNU/Linux PC Terminal

   ```
   $ sudo netplan apply
   ```

3. Verify using `ip addr`.