Building Machine Vision Applications using OpenMV

Class 4: Utilizing Machine Learning to Detect Objects

June 11, 2020

Jacob Beningo

Presented by:

DesignNews

CEC CONTINUING EDUCATION CENTER

Digi-Key ELECTRONICS

Course Overview

Topics:

- Introduction to Machine Vision and OpenMV
- Writing our First OpenMV Application
- Working with the OpenMV I/O
- **Utilizing Machine Learning to Detect Objects**
- Designing a Machine Vision Application

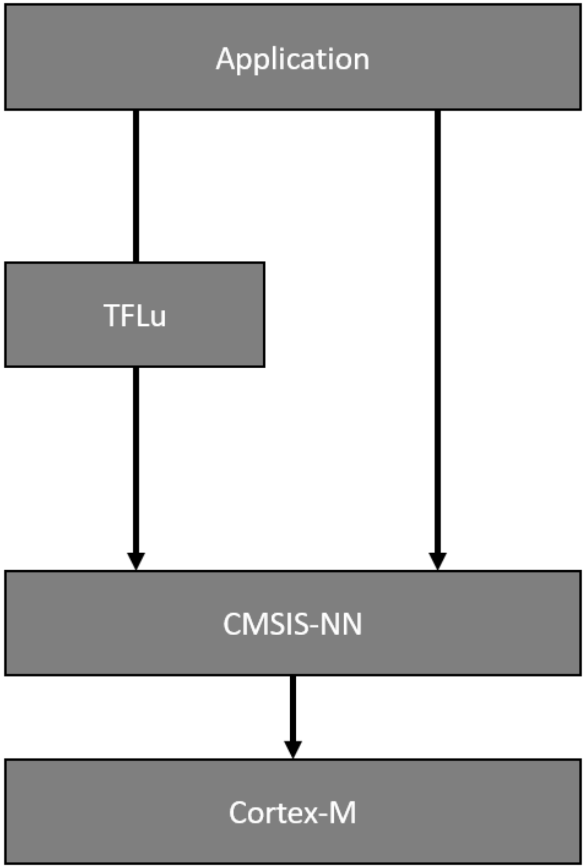
Session Overview

- Introduction
- Machine Learning
- Image Classification Example

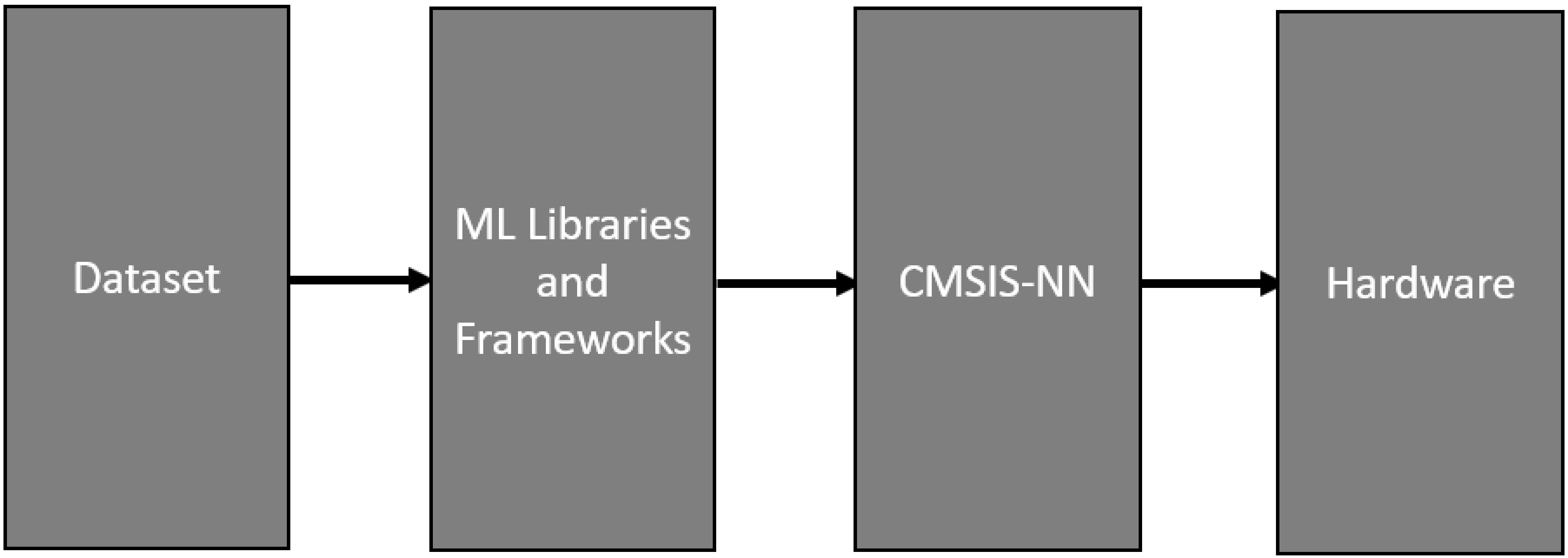
Presented by:

Introduction

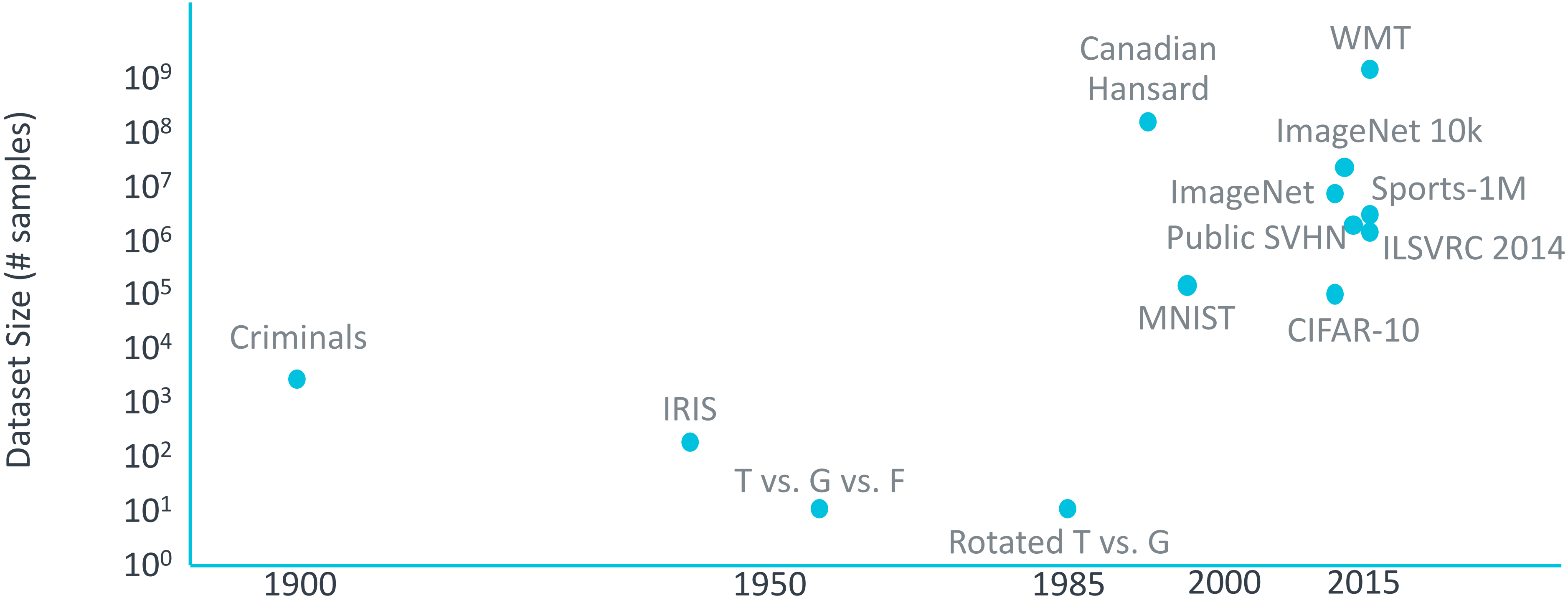
- Running ML framework on Cortex-M systems is impractical
- Need to run bare-metal code to efficiently use the limited resources
- TFLu: Tensor Flow Lite for Microcontrollers
- **CMSIS-NN:** optimized low-level NN functions for Cortex-M CPUs
- CMSIS-NN APIs may also be directly used in the application code



Introduction



Datasets



Presented by:

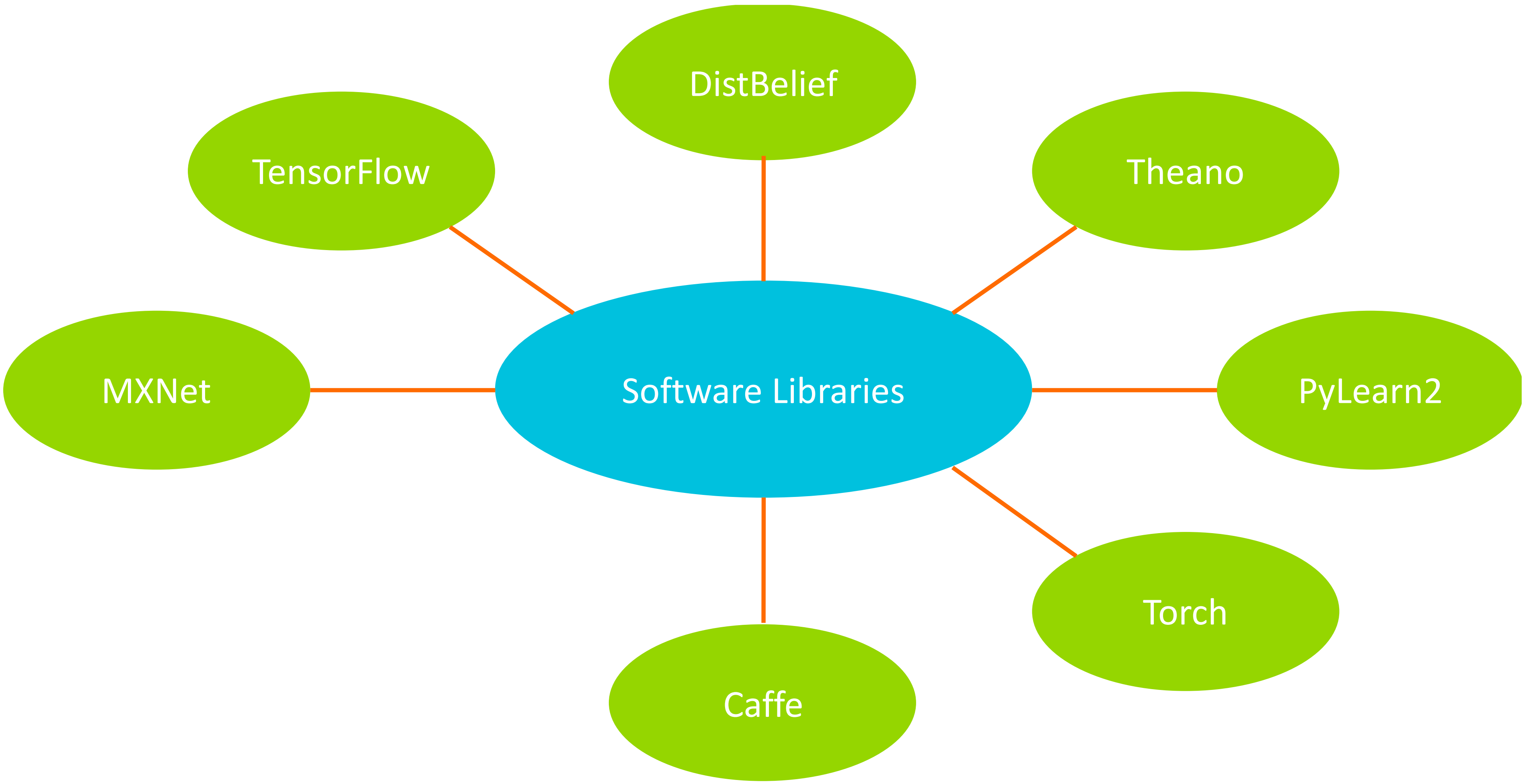
Datasets

CIFAR-10 is a collection of 60,000 images, each at 32-pixel by 32-pixel from 10 image classes that include:

- Airplanes
- Cars
- Birds
- Cats
- Deer
- Dogs
- Frogs
- Horses
- Ships
- Trucks

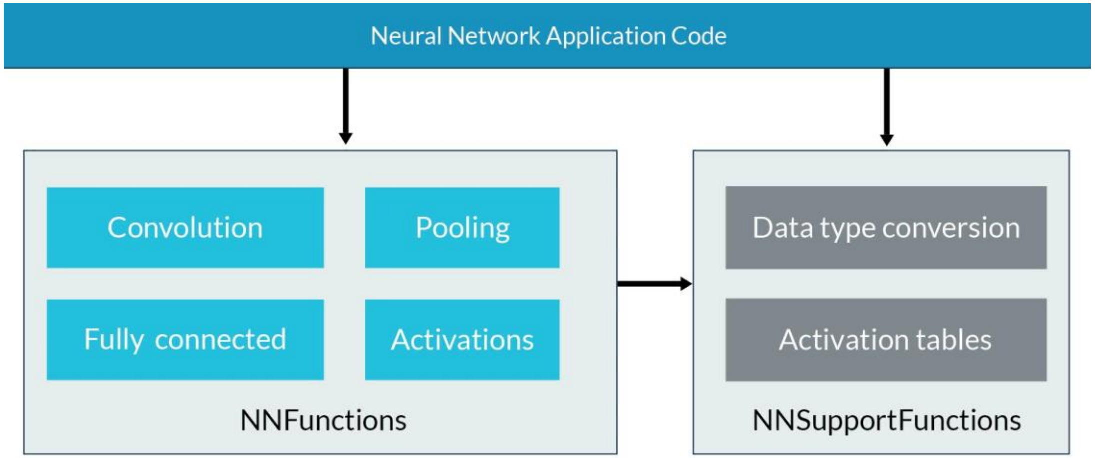
Source: researchgate.net

ML Libraries and Frameworks

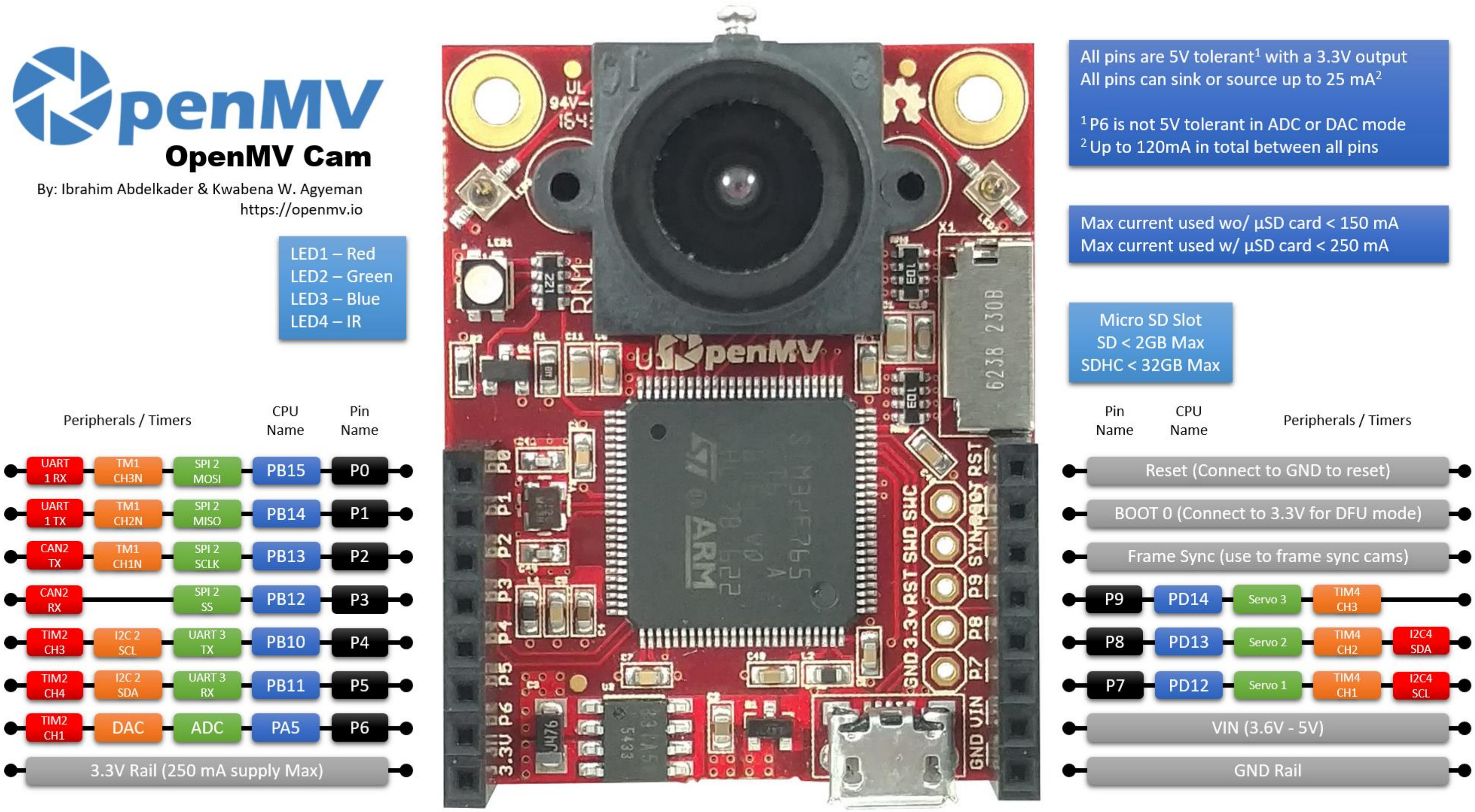


CMSIS-NN

- CMSIS-NN: collection of optimized neural network functions for Cortex-M CPUs
- Key considerations:
  – Improve performance using SIMD instructions
  – Minimize memory footprint
  – NN-specific optimizations: data-layout and offline weight reordering



Hardware



OpenMV
OpenMV Cam
By: Ibrahim Abdelkader & Kwabena W. Agyeman
https://openmv.io
LED1 – Red
LED2 – Green
LED3 – Blue
LED4 – IR
All pins are 5V tolerant[1] with a 3.3V output
All pins can sink or source up to 25 mA[2]
[1] P6 is not 5V tolerant in ADC or DAC mode
[2] Up to 120mA in total between all pins
Max current used wo/ µSD card < 150 mA
Max current used w/ µSD card < 250 mA
Micro SD Slot
SD < 2GB Max
SDHC < 32GB Max
Peripherals / Timers
CPU Name
Pin Name
UART 1 RX | TM1 CH3N | SPI 2 MOSI | PB15 | P0
UART 1 TX | TM1 CH2N | SPI 2 MISO | PB14 | P1
CAN2 TX | TM1 CH1N | SPI 2 SCLK | PB13 | P2
CAN2 RX | SPI 2 SS | PB12 | P3
TIM2 CH3 | I2C 2 SCL | UART 3 TX | PB10 | P4
TIM2 CH4 | I2C 2 SDA | UART 3 RX | PB11 | P5
TIM2 CH1 | DAC | ADC | PA5 | P6
3.3V Rail (250 mA supply Max)
Pin Name
CPU Name
Peripherals / Timers
Reset (Connect to GND to reset)
BOOT 0 (Connect to 3.3V for DFU mode)
Frame Sync (use to frame sync cams)
P9 | PD14 | Servo 3 | TIM4 CH3
P8 | PD13 | Servo 2 | TIM4 CH2 | I2C4 SDA
P7 | PD12 | Servo 1 | TIM4 CH1 | I2C4 SCL
VIN (3.6V - 5V)
GND Rail

Image Classification Example

1. From the top menu, click Tools -> machine learning -> CNN Network Library
2. In the pop-up window, navigate to CMSIS-NN -> cifar10
3. Click the cifar10.network file and select open
4. Another window will pop-up. This window is asking where to save the selected file. Navigate to the OpenMV mass storage device drive that appeared when you connected the camera. Click save.

Image Classification Example

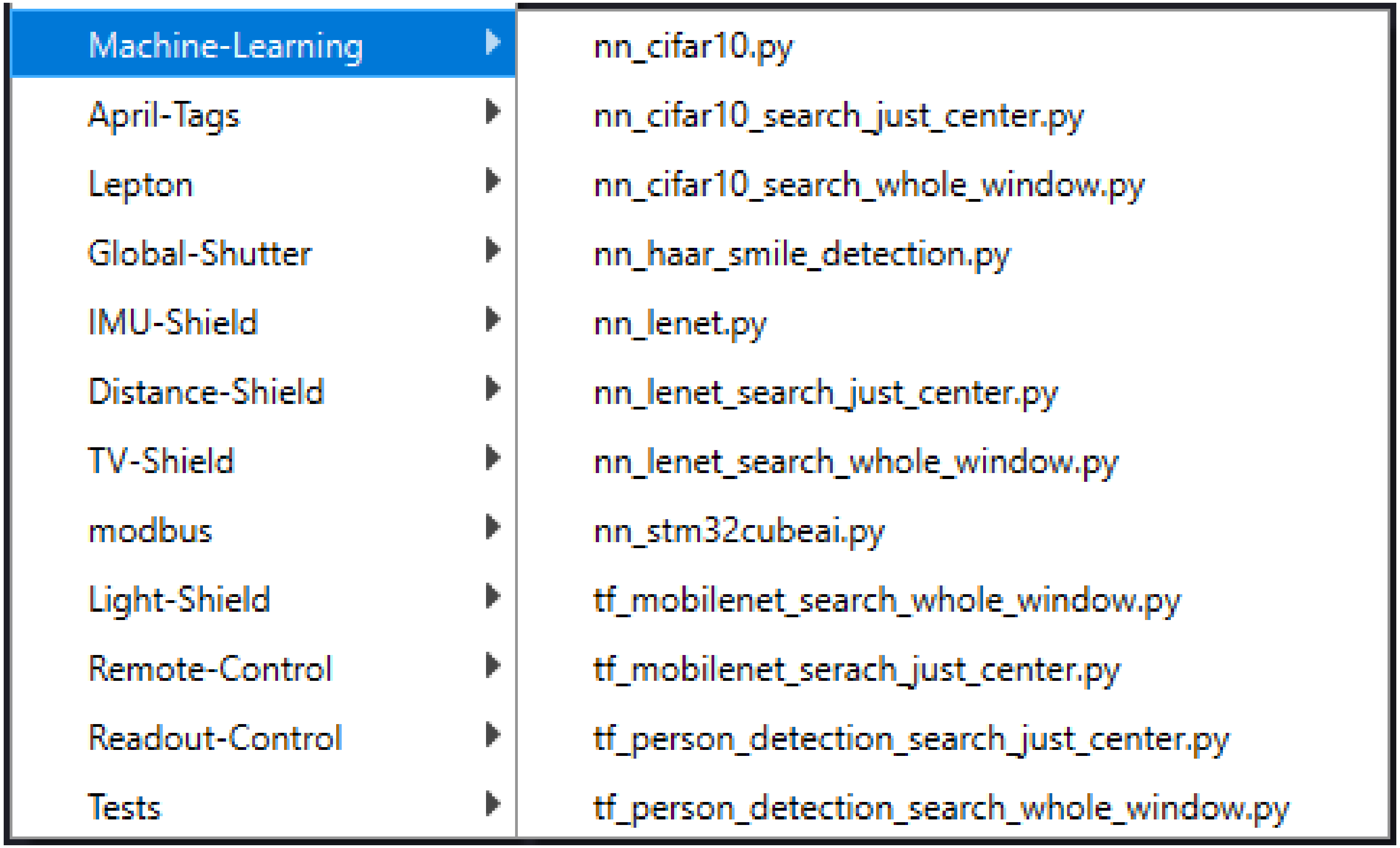


Image Classification Example

```
# CIFAR-10 Search Whole Window Example
#
# CIFAR is a convolutional neural network designed to classify its field of
# view into several different object types and works on RGB video data.
#
# In this example, we slide the LeNet detector window over the image and get
# a list of activations where there might be an object. Note that using a CNN
# with a sliding window is extremely compute expensive, so for an exhaustive
# search do not expect the CNN to be real-time.

import sensor, image, time, os, nn
```

Image Classification Example

```
sensor.reset()                              # Reset and initialize the sensor.
sensor.set_pixformat(sensor.RGB565)         # Set pixel format to RGB565
sensor.set_framesize(sensor.QVGA)           # Set frame size to QVGA (320x240)
sensor.set_windowing((128, 128))            # Set 128x128 window.
sensor.skip_frames(time=750)                # Don't let autogain run very long.
sensor.set_auto_gain(False)                 # Turn off autogain.
sensor.set_auto_exposure(False)             # Turn off whitebalance.
```

Image Classification Example

```
# Load the cifar10 network (You can get the network from OpenMV IDE).
net = nn.load('/cifar10.network')

# Faster, smaller and less accurate.
# net = nn.load('/cifar10_fast.network')
labels = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']

clock = time.clock()
while(True):
    clock.tick()

    img = sensor.snapshot()
```

Image Classification Example

net.search() will search an roi in the image for the network
(or the whole image if the roi is not specified). At each location to
look in the image if one of the classifier outputs is larger than
threshold the location and label will be stored in an object list and
returned. At each scale the detection window is moved around in the ROI
using x_overlap (0-1) and y_overlap (0-1) as a guide.

If you set the overlap to 0.5 then each detection window will overlap
the previous one by 50%. Note the computational workload goes WAY up
the more overlap. Finally, for mult-scale matching after sliding the
network around in the x/y dimensions the detection window will shrink
by scale_mul (0-1)down to min_scale (0-1). For example, if scale_mul is
0.5 the detection window will shrink by 50%.
Note that at a lower scale there's even more area to search if
x_overlap and y_overlap are small... contrast_threshold skips running
the CNN in areas that are flat.

Image Classification Example

```
for obj in net.search(img, threshold=0.6, min_scale=0.5, scale_mul=0.5, \
    x_overlap=0.5, y_overlap=0.5, contrast_threshold=0.5):
        print("Detected %s - Confidence %f%%"% (labels[obj.index()],\
        obj.value()))

    img.draw_rectangle(obj.rect(), color=(255, 0, 0))
        print(clock.fps())
```

Serial Terminal

```
Detected airplane - Confidence 0.711765%
1.893939
Detected airplane - Confidence 0.715686%
1.893939
Detected airplane - Confidence 0.704902%
1.893939
Detected airplane - Confidence 0.701961%
1.893939
Detected airplane - Confidence 0.700000%
1.893939
Detected airplane - Confidence 0.695098%
1.893939
```

Additional Resources

- Beningo.com
  - Blog, White Papers, Courses
  - Embedded Bytes Newsletter
    - http://bit.ly/1BAHYXm
- OpenMV.io

From www.beningo.com under

- Blog > CEC – Building Machine Vision Applications using OpenMV