Machine Learning for Embedded Software Engineers

Class 4: Machine Vision with OpenMV

April 25, 2019

Jacob Beningo

DesignNews

CEC CONTINUING EDUCATION CENTER

Presented by: Digi-Key ELECTRONICS

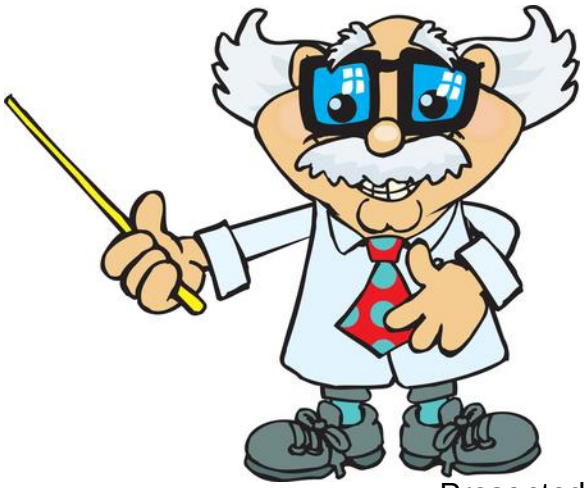
Course Overview

Topics:

- Introduction to Machine Learning
- Machine Learning Architectures for Embedded Systems
- Machine Learning Applications: Vision and Speech
- **Machine Vision with OpenMV**
- Near Real-time Machine Learning using Coral

Session Overview

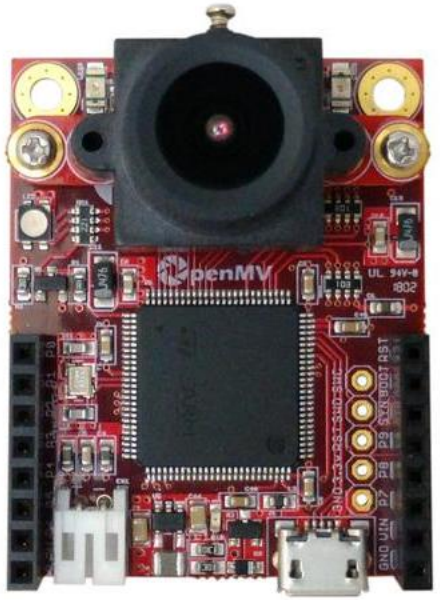
- Introduction to OpenMV
- The OpenMV hardware
- An example
  - Smile detection
  - Training with Caffe
  - Converting the model
  - Deploying the model



OpenMV

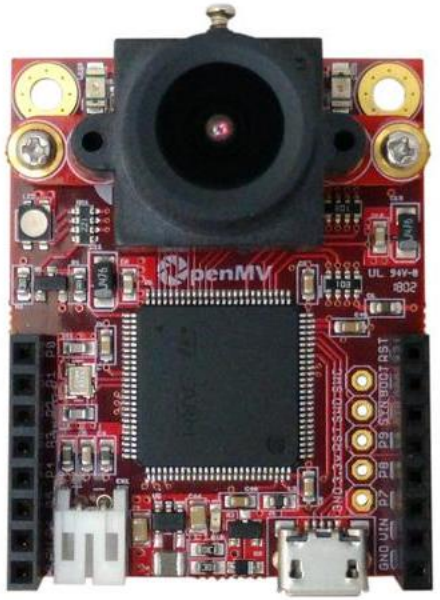
OpenMV is a project that brings machine vision into a simple Arduino like environment for “simple” applications.

- Can write scripts in Python
- Expandable I/O for interfacing
- Built-in camera module
- Expandable camera options
- Custom IDE for developing and deploying scripts



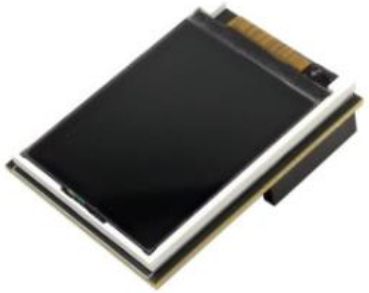
The OpenMV Hardware

- Arm Cortex-M7 Processor
  - STM32H743VI
  - 400 MHz
  - 1MB RAM
  - 2 MB Flash
- OV7725 image sensor
  - 640x480 16-bit RGB565 @ 60 FPS
  - 640x480 8-bit Grayscale images
- 2.8 mm lens



OpenMV Expansion Hardware

LCD



Wi-Fi

Servo

Pan and Tilt

Motor

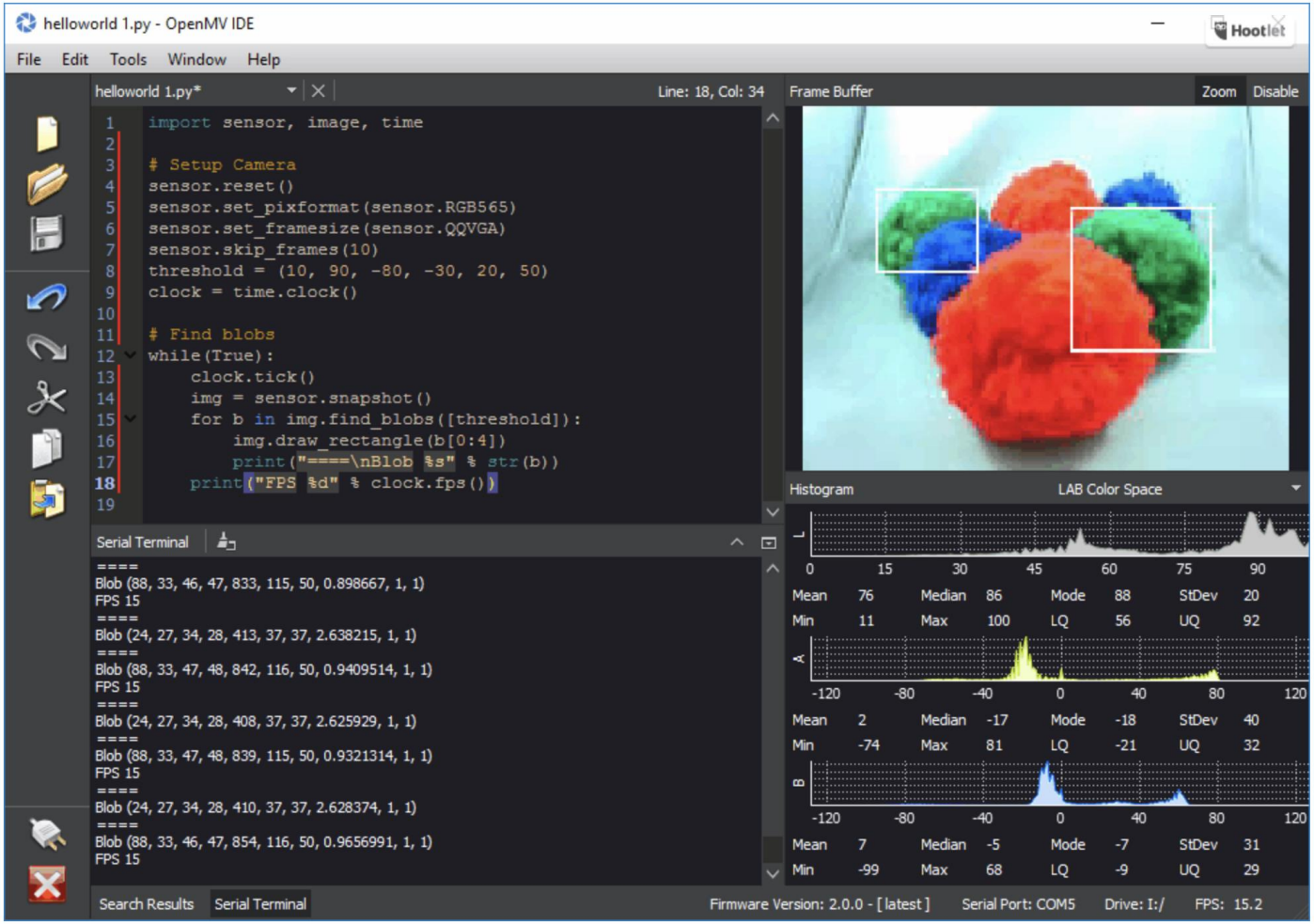
Flit Camera Adapter

Getting Started with OpenMV

- Use a screw driver to remove the two lens mount screw from the lens mount on your OpenMV Cam.
- Apply some isopropyl alcohol to a small part of the cloth.
- Rub the wet part of the cloth on the camera IC gently. Any dirt spots on the camera IC will be microscopic to the human eye so just try to generally rub down the top of the camera IC. Note that the top of the camera IC is glass.
- After cleaning the camera IC make sure the alcohol has evaporated completely and that no cloth strands were left behind. Note that we're using isopropyl alcohol versus water since isopropyl alcohol evaporates quickly and doesn't leave anything behind.
- Use the screw driver to re-attach the lens mount. Make sure that the set screw on the lens mount points off the top/back of the OpenMV Cam.

Source: docs.openmv.io

The OpenMV IDE

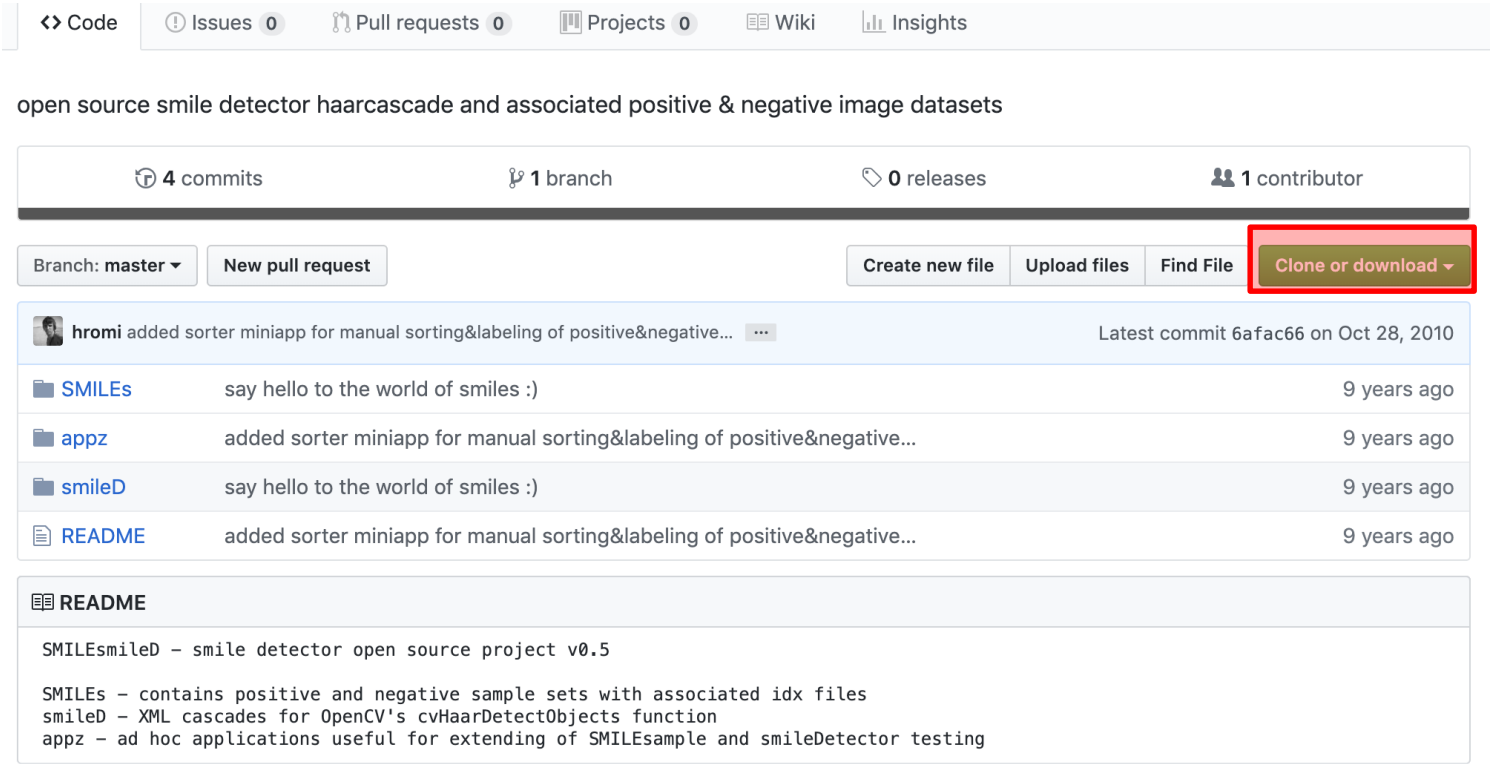


Smile Recognition

1) Download the smile dataset
2) Train a model using Caffe
3) Quantize the model
4) Convert the model to a binary
5) Deploy to the OpenMV module
6) Run the model

Download the Smile Dataset

https://github.com/hromi/SMILEsmileD



The Smile Dataset

- Contains
  - 3,000 positive images
  - 9,000 negative images
- The skewed dataset could result in bias!

Augment the images using the following script:

https://github.com/openmv/openmv/blob/master/tools/augment_images.py

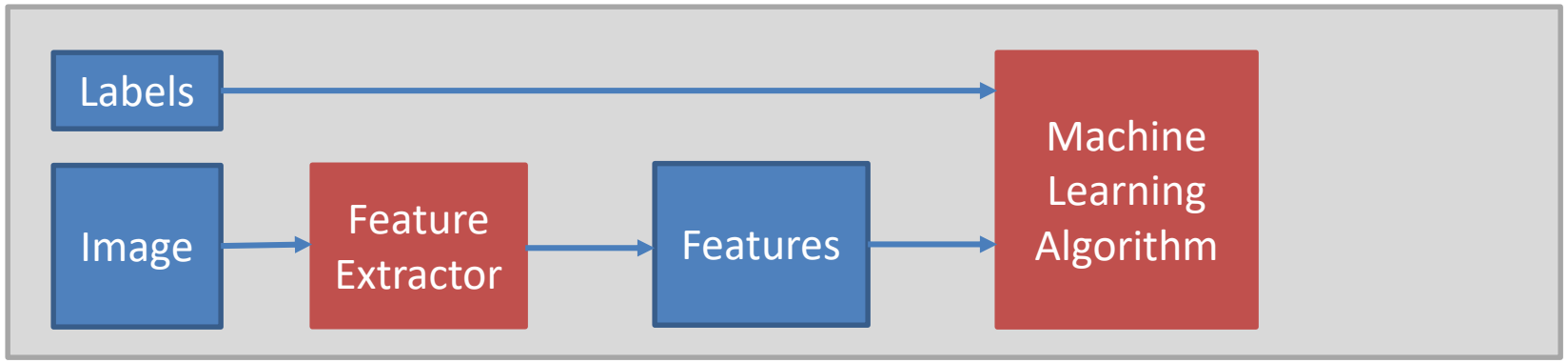
The Smile Dataset

- Download the script or copy it to a file
- Use the following to run the script:

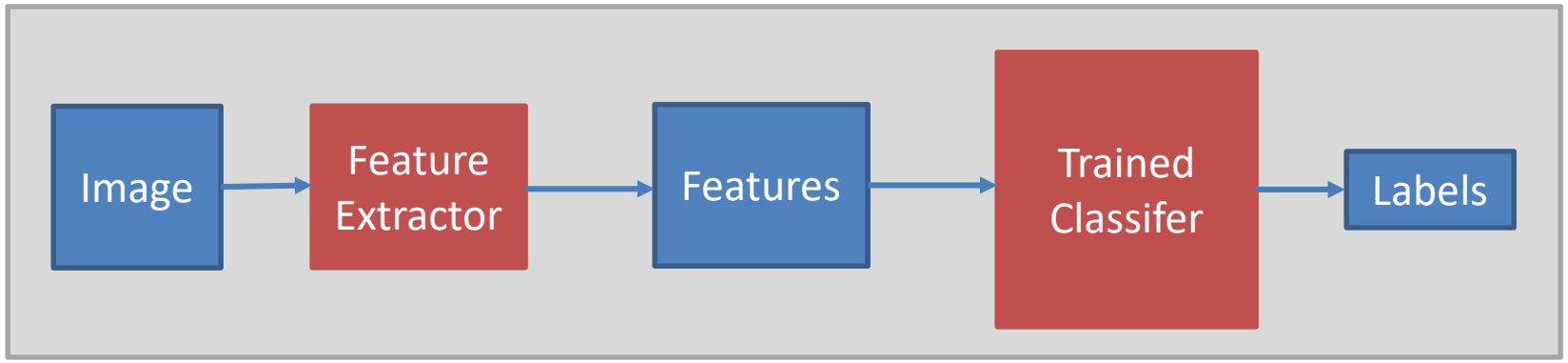
Python augment_images.py –input /Smiles_POSITIVE_DIR –output /Smiles_POSITIVE_2 –count 4

Training using Caffe

Training Phase



Prediction Phase



Training using Caffe

Caffe is a deep learning framework created with expression, speed and modularity in mind.

- Developed by Berkeley AI Research
- Download Caffe from https://caffe.berkeleyvision.org/

4 Steps to Training a CNN

1) Prepare the data (convert to Caffe friendly format)
2) Define the model (create config file, .prototxt)
3) Define the solver (solver parameters, .prototxt)
4) Model training (execute the model)

Quantizing the Model

After model training, need to quantize the model

- Convert weights and activations from floating point to fixed point
- Arm provides a script for Caffe at https://github.com/ARM-software/ML-examples/blob/master/cmsisnn-cifar10/nn_quantizer.py
- Run the script:

```
python2 nn_quantizer.py --model
models/smile/smile_train_test.prototxt --weights
models/smile/smile_iter_*.caffemodel --save
models/smile/smile.pkl
```

Converting the Model to Binary

Need to output code for each layer in the NN with dimensions and weights.

Download the NN Converter Script:

- https://github.com/openmv/openmv/blob/master/ml/cmsisnn/nn_convert.py

python2 nn_convert.py --model models/smile/smile.pkl --mean /path/to/mean.binaryproto --output smile.network

Deploying on the OpenMV

Start by loading the NN into memory:

```
# Load Smile Detection network
net = nn.load('/smile.network')

# Load Face Detection Haar Cascade
face_cascade = image.HaarCascade("frontalface", stages=25)
print(face_cascade)
```

Deploying on the OpenMV

Next, capture an image and find the faces:

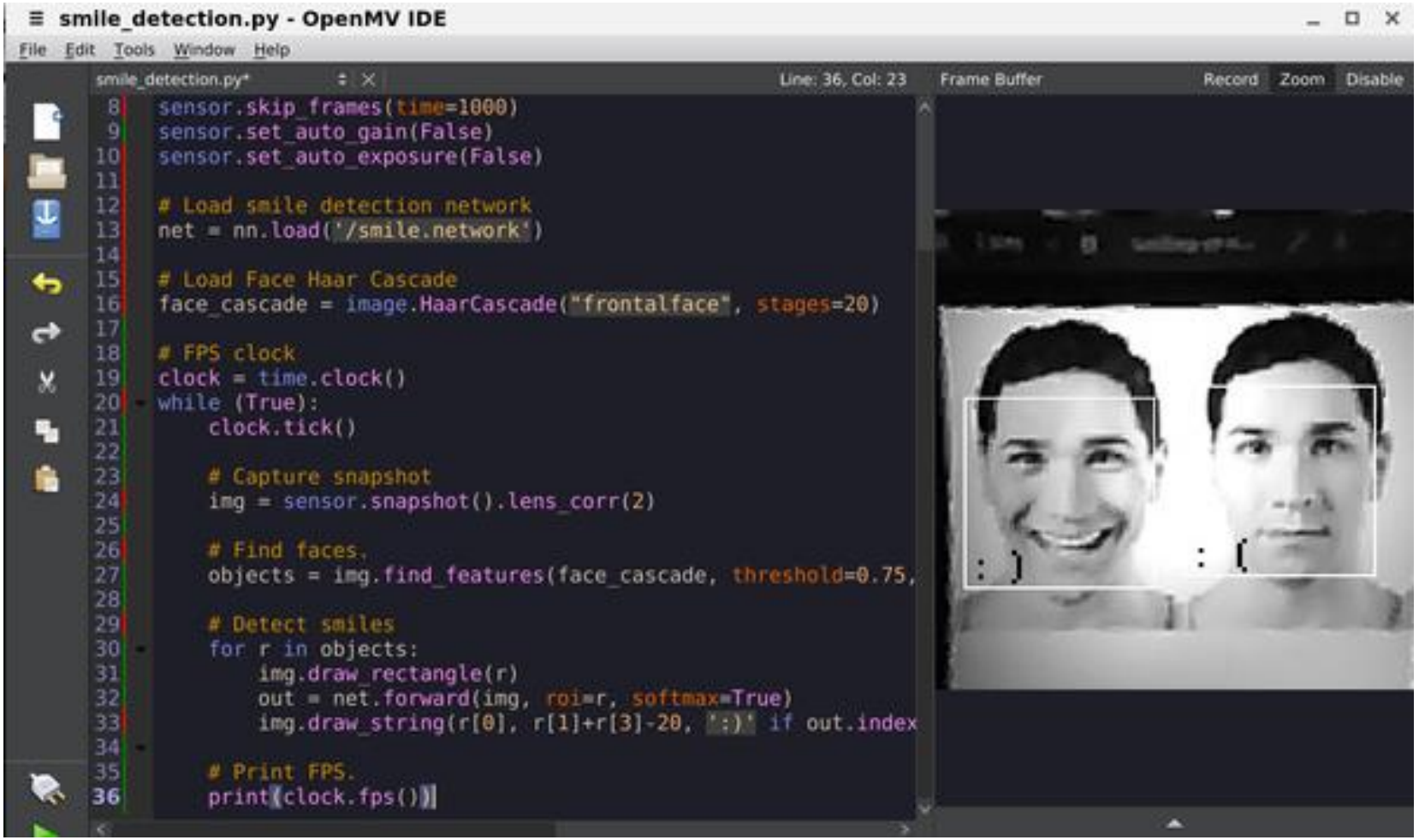
```
# Capture snapshot
img = sensor.snapshot()

# Find faces.
objects = img.find_features(face_cascade, threshold=0.75,
scale_factor=1.25)
```

Deploying on the OpenMV

Pass each image to the neural network:

```
for r in objects:
  # Resize and center detection area
  r = [r[0]+10, r[1]+25, int(r[2]*0.70), int(r[2]*0.70)]
  out = net.forward(img, roi=r, softmax=True)
  img.draw_string(r[0], r[1], ':)' if (out[0] > 0.8) else ':(', color=0,
scale=2)
```

The Results


smile_detection.py - OpenMV IDE
File Edit Tools Window Help
smile_detection.py*
Line: 36, Col: 23
Frame Buffer
Record
Zoom
Disable
sensor.skip_frames(time=1000)
sensor.set_auto_gain(False)
sensor.set_auto_exposure(False)
# Load smile detection network
net = nn.load('/smile.network')
# Load Face Haar Cascade
face_cascade = image.HaarCascade("frontalface", stages=20)
# FPS clock
clock = time.clock()
while (True):
clock.tick()
# Capture snapshot
img = sensor.snapshot().lens_corr(2)
# Find faces.
objects = img.find_features(face_cascade, threshold=0.75,
# Detect smiles
for r in objects:
img.draw_rectangle(r)
out = net.forward(img, roi=r, softmax=True)
img.draw_string(r[0], r[1]+r[3]-20, ':)' if out.index
# Print FPS.
print(clock.fps())
:)
: (

References and Resources

- https://openmv.io/pages/download
- https://openmv.io/blogs/news/deep-learning-on-a-cortex-m7-camera-3ma-deep-learning
- http://adilmoujahid.com/posts/2016/06/introduction-deep-learning-python-caffe/

Additional Resources

- Download Course Material for
  - C/C++ Doxygen Templates
  - Example source code
  - Blog
  - YouTube Videos
- Embedded Bytes Newsletter
  - http://bit.ly/1BAHYXm

From www.beningo.com under

- Blog > CEC – Machine Learning for Embedded Software Engineers