Machine Learning for Makers

AutoML Workshop 2017
Rob DeLine, Microsoft Research
Intelligent Devices team
50 Billion Disruptive Devices
Many of these devices will be intelligent

What’s an “intelligent device”?

• Often small
• Often mobile (may be disconnected or semi-connected to the cloud)
• Sensor-based (e.g., camera, microphone, motion, temperature, touch)
• Intelligence:

Interpret sensor output \rightarrow \text{Inferences and predictions about user and environment} \rightarrow \text{Intelligent decisions}
<table>
<thead>
<tr>
<th>Present State</th>
<th>Desired State</th>
</tr>
</thead>
<tbody>
<tr>
<td>mostly, dumb devices connected to an intelligent cloud</td>
<td>embedded intelligence supplemented by intelligent cloud</td>
</tr>
<tr>
<td>low-accuracy embedded intelligence</td>
<td>state-of-the-art accuracy</td>
</tr>
<tr>
<td>building an intelligent device requires large team of experts for months-years</td>
<td>embedded intelligence provided to anyone with basic technical competence for hours-days</td>
</tr>
<tr>
<td>ad-hoc scripts; reinventing the wheel</td>
<td>streamlined workflow supported by tools</td>
</tr>
<tr>
<td></td>
<td>embedded computer vision</td>
</tr>
</tbody>
</table>
So many devices!

It’s easy to generate a very long list of applications for intelligent devices.

To produce all these, we need broad participation.

• Small-scale entrepreneurs
• Makers
• Tech enthusiasts

We are far from this today.
Strength Training Activity Tracker
Multiclass classification over inertial sensors
(4 PhDs, 4 contractors, 1 PM, ~1.5 person-years)
Workflow for Activity Tracker


- Define Problem
- Collect Raw Data
- Align & Clean
- Segment & Label
- Filter & Features
- Machine Learning
- Deploy & Evaluate

8 trial and error

Signal processing expertise

Hired contractors

$15K commercial software

Ad hoc code and data munging in Excel

Multiclass classification

SVM

30 participants in Lab

~1 hour each
A Common Workflow Across Devices

Interviewed other colleagues.
Examined documented processes in the literature.
Created our own models (gesture detector for the micro:bit, sleep cycle classifier from Band data).
Whiteboarded a variety of application scenarios.
Challenges in Streamlining the Workflow

Define Problem

Collect Raw Data

Align & Clean

Segment & Label

Filter & Features

Machine Learning

Deploy & Evaluate

Support for iteration and experimentations

Feasibility testing

Problem decomposition

Semi-automated sensor alignment

Semi-automated labeling

Visual featuring language

Interactive featuring and training

Debugging support

Lightweight feedback mechanisms

Cost/benefit estimates of obtaining more data
Dogs versus Cats

One peculiarity of the Darknet image dataset is that the labels are often quite specific. For example, rather than labeling an image of a dog as "dog", then model uses dozens of labels for various dog breeds, for example, golden retriever and black-and-white coonhound. Similarly, cat images are labeled with cat breeds. If we want to predict whether an image is a dog or a cat (or neither), then these breed labels are too specific.

In this tutorial, we use a prebuilt Darknet image recognition model to get a prediction for an image. Then we process the prediction to broaden it from a breed-specific label to the more generic label of "dog" or "cat". Since we are only interested in the distinction between dogs and cats, we also allow some "near misses". For example, a picture of a domestic cat might be predicted to be a lynx (a type of wild cat). A biologist might object to mixing up wild cats and domestic cats, but for our purposes, a lynx counts as a cat as much as a Persian or tabby.

Loading the prebuilt Darknet model

We start by downloading a pretrained Darknet model, including its labels and weights.

```
In [1]:
from ell.util.downloader import Downloader
dl = Downloader("darknet_demo")
cfg_path = dl.download("https://raw.githubusercontent.com/pjreddie/darknet/master/cfg/darknet.cfg", 'darknet.cfg')
labels_path = dl.download("https://github.com/Microsoft/ELL/blob/master/Tutorials/vision/GettingStarted/darknetImageNetLabels.txt")
weights_path = dl.download_binary("https://pjreddie.com/media/files/darknet.weights", 'darknet.weights')
```

Next, we convert this pretrained model to an ELL model.

```
In [2]:
import ell.vision.darknet_to_ell as darknet_to_ell
import ell.vision.modelHelper as helper
import sys

helper = helper.ModelHelper(sys.argv, "darknetReference", [cfg_path, weights_path], labels_path)
predictor = darknet_to_ell.predictor_from_darknet_model(helper.model_files[0], helper.model_files[1])
```
on gesture Running:
- show animation rainbow for 500 ms
- set all pixels to black

on gesture Jump:
- change jumpCount by 1
- if jumpCount > 5
  - play sound magic wand
Embedded Learning Library

The Embedded Learning Library (ELL) allows you to build and deploy machine-learned pipelines onto embedded platforms, like Raspberry Pis, Arduinos, micro:bits, and other microcontrollers. The deployed machine learning model runs on the device, disconnected from the cloud. Our APIs can be used either from C++ or Python.

This project has been developed by a team of researchers at Microsoft Research. It’s a work in progress, and we expect it to change rapidly, including breaking API changes. Despite this code churn, we welcome you to try it and give us feedback!

A good place to start is the tutorial, which allows you to do image recognition on a Raspberry Pi with a web cam, disconnected from the cloud. The software you deploy to the Pi will recognize a variety of common objects on camera and print a label for the recognized object on the Pi’s screen.