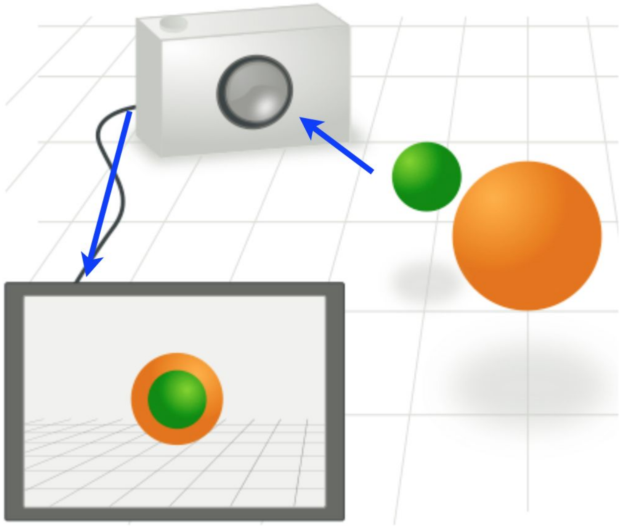
Embedded computing for scientific and industrial imaging applications

Lecture 9 - Intro to OpenCV

Ref : Designing Computer Vision Apps @ CMU

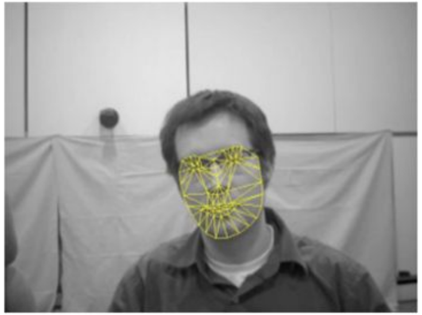
Computer Vision

Computer vision is an interdisciplinary field that deals with how computers can be made to gain high-level understanding from digital images or videos.

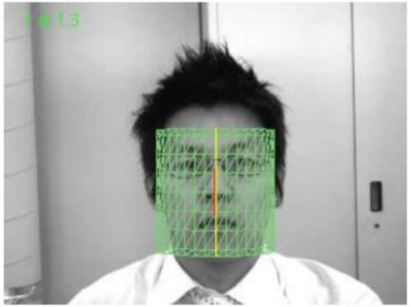


Extracting structure from pixels.

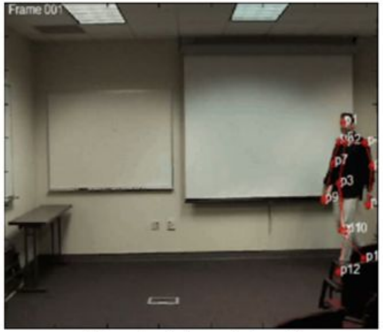
Applications of Computer Vision



“Face Recognition”



“Pose Estimation”



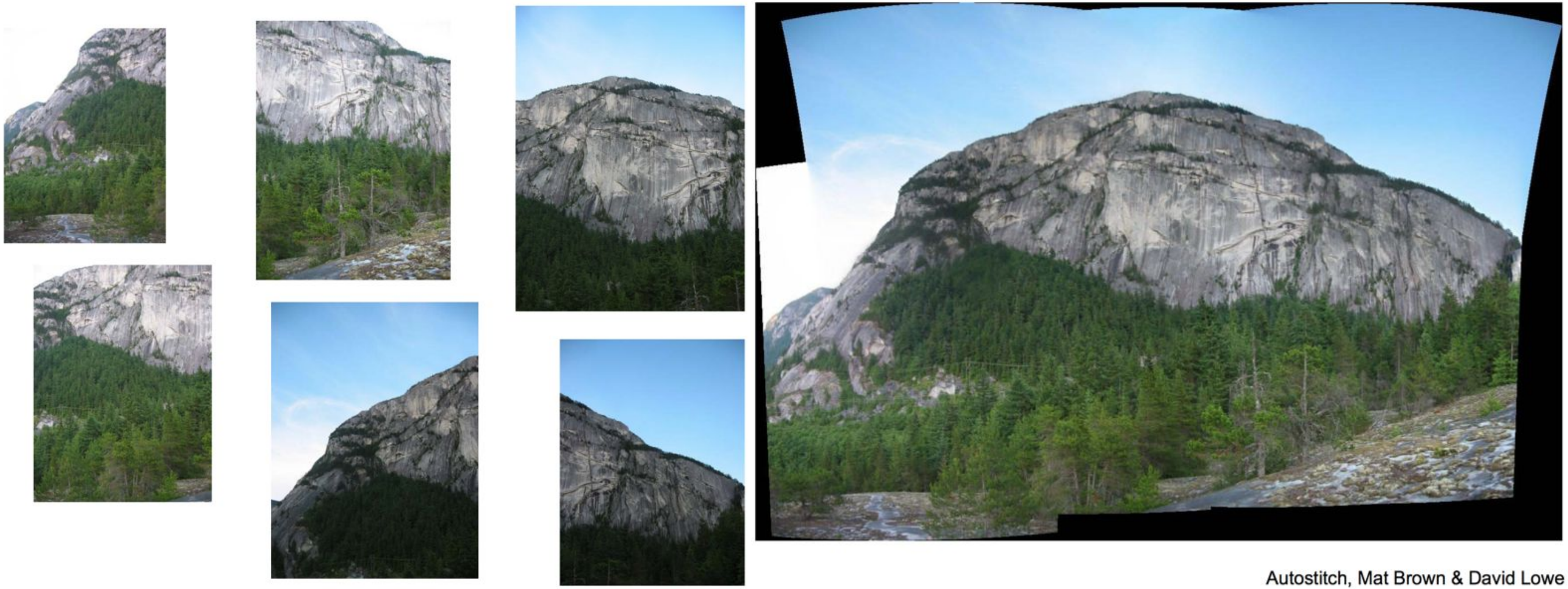
“Body Tracking”

“Speech Reading”

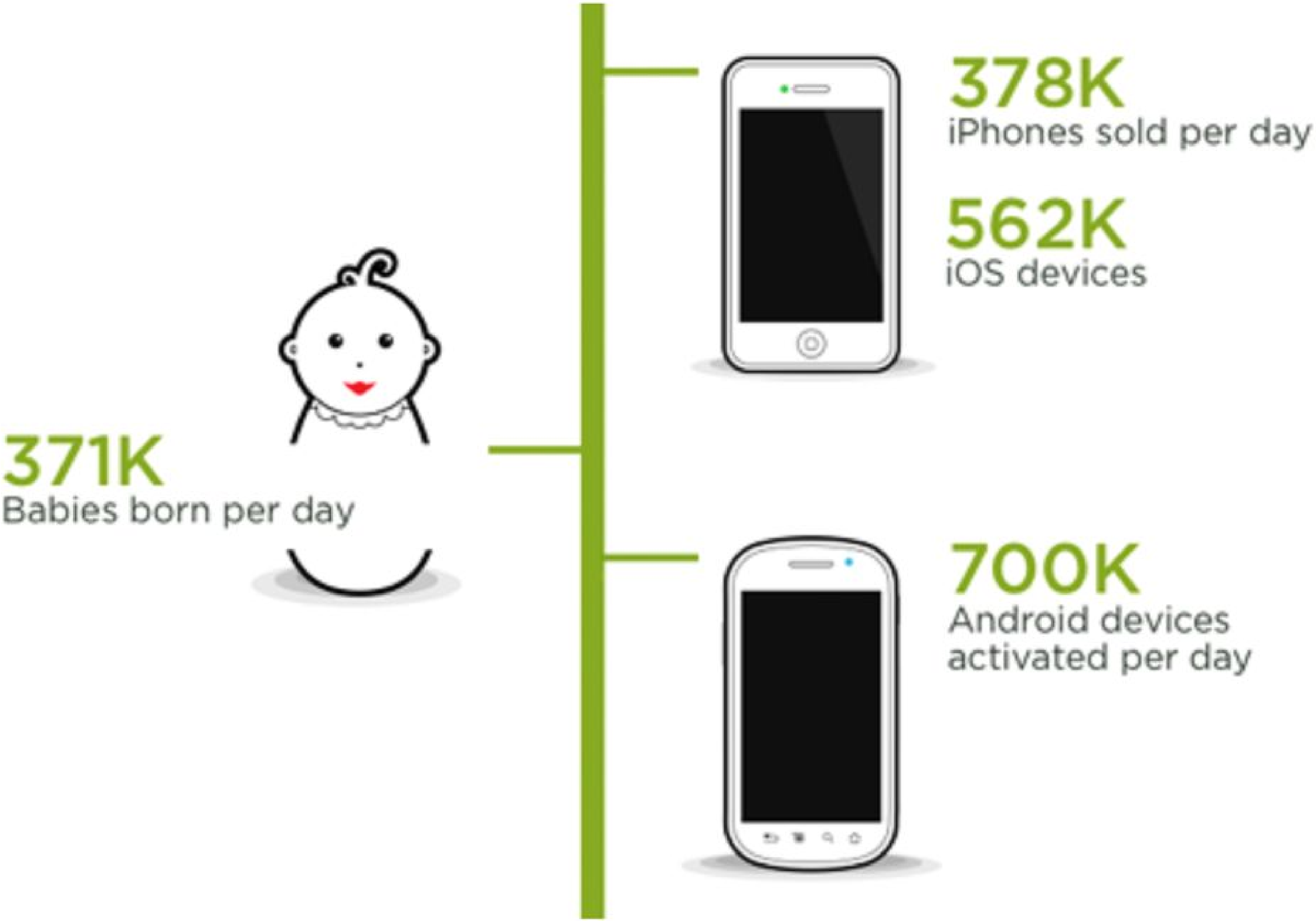
“Palm Recognition”

“Car Tracking”

Applications of Computer Vision



Autostitch, Mat Brown & David Lowe



378K
iPhones sold per day

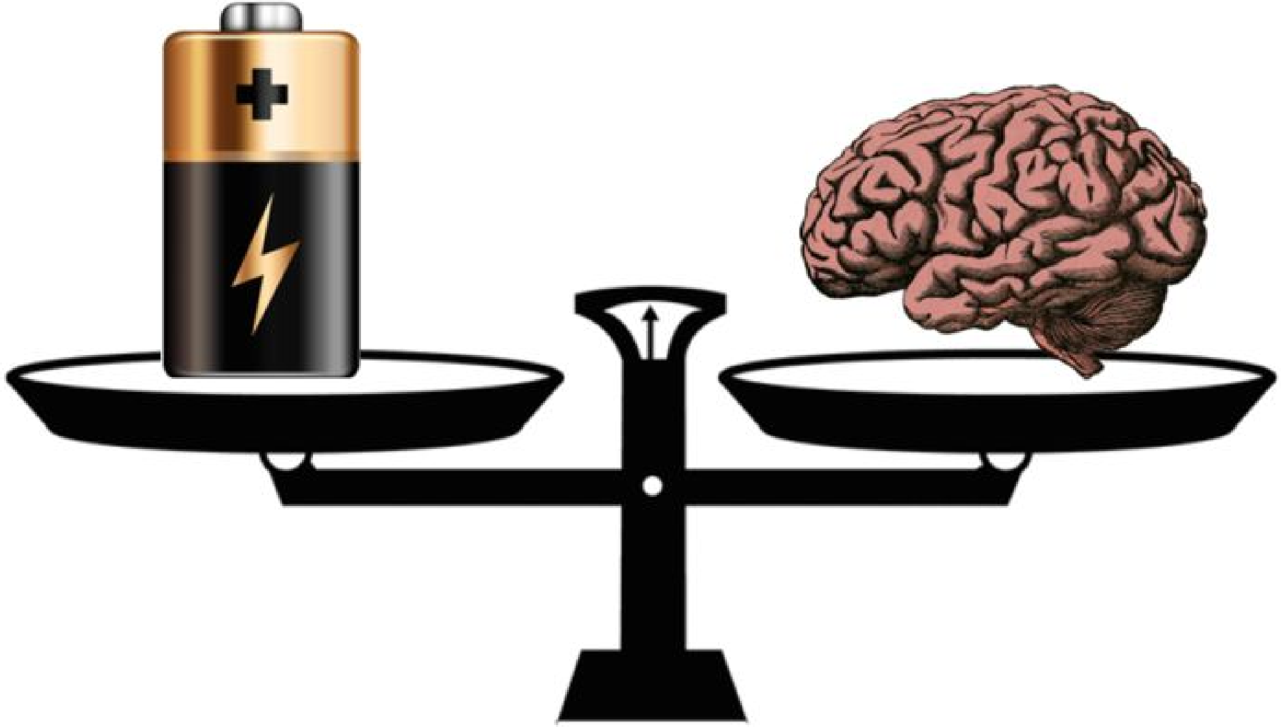
562K
iOS devices

371K
Babies born per day

700K
Android devices
activated per day

Why is Mobile CV Different?

Balancing Power versus Perception

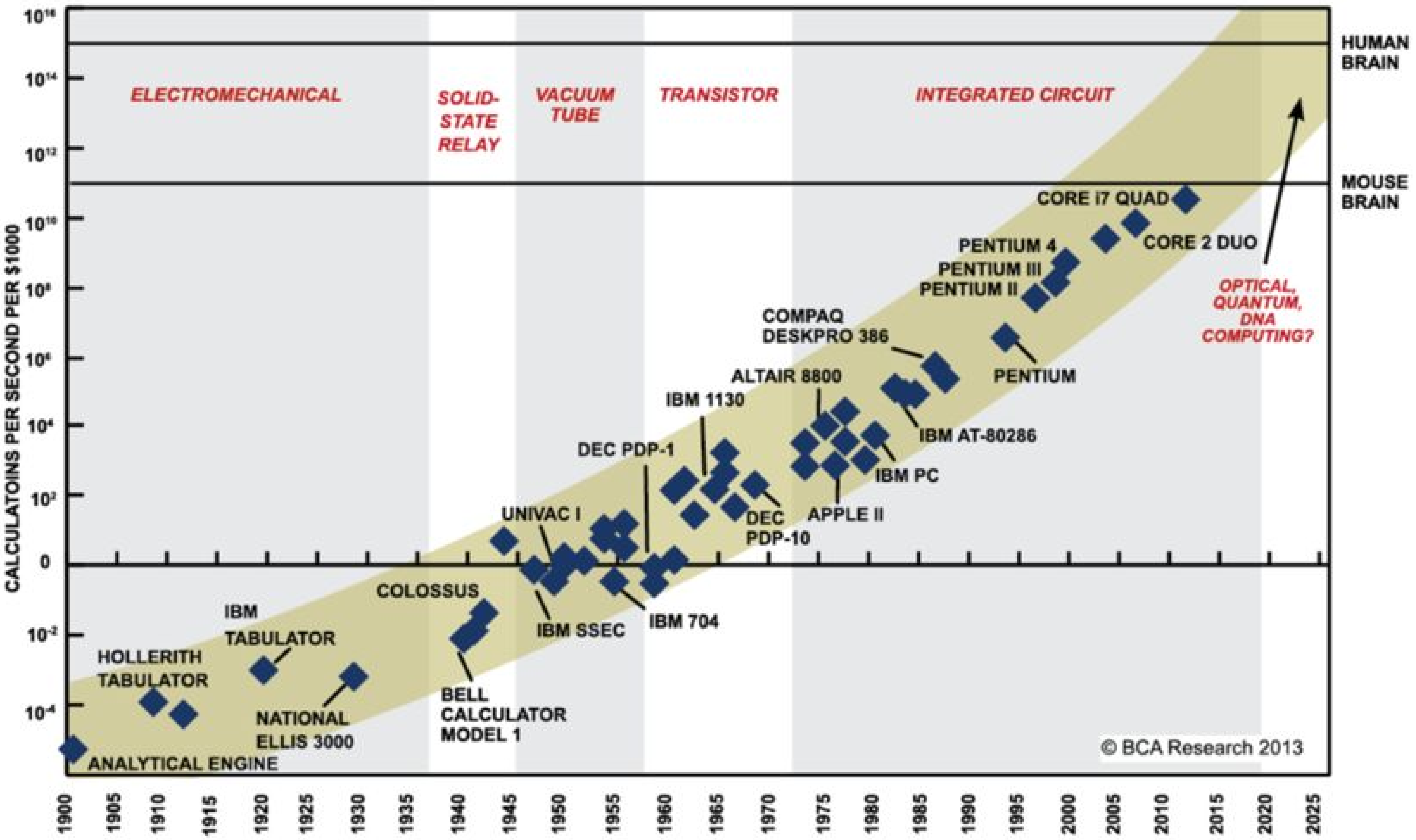


Low Power Image Recognition Challenge

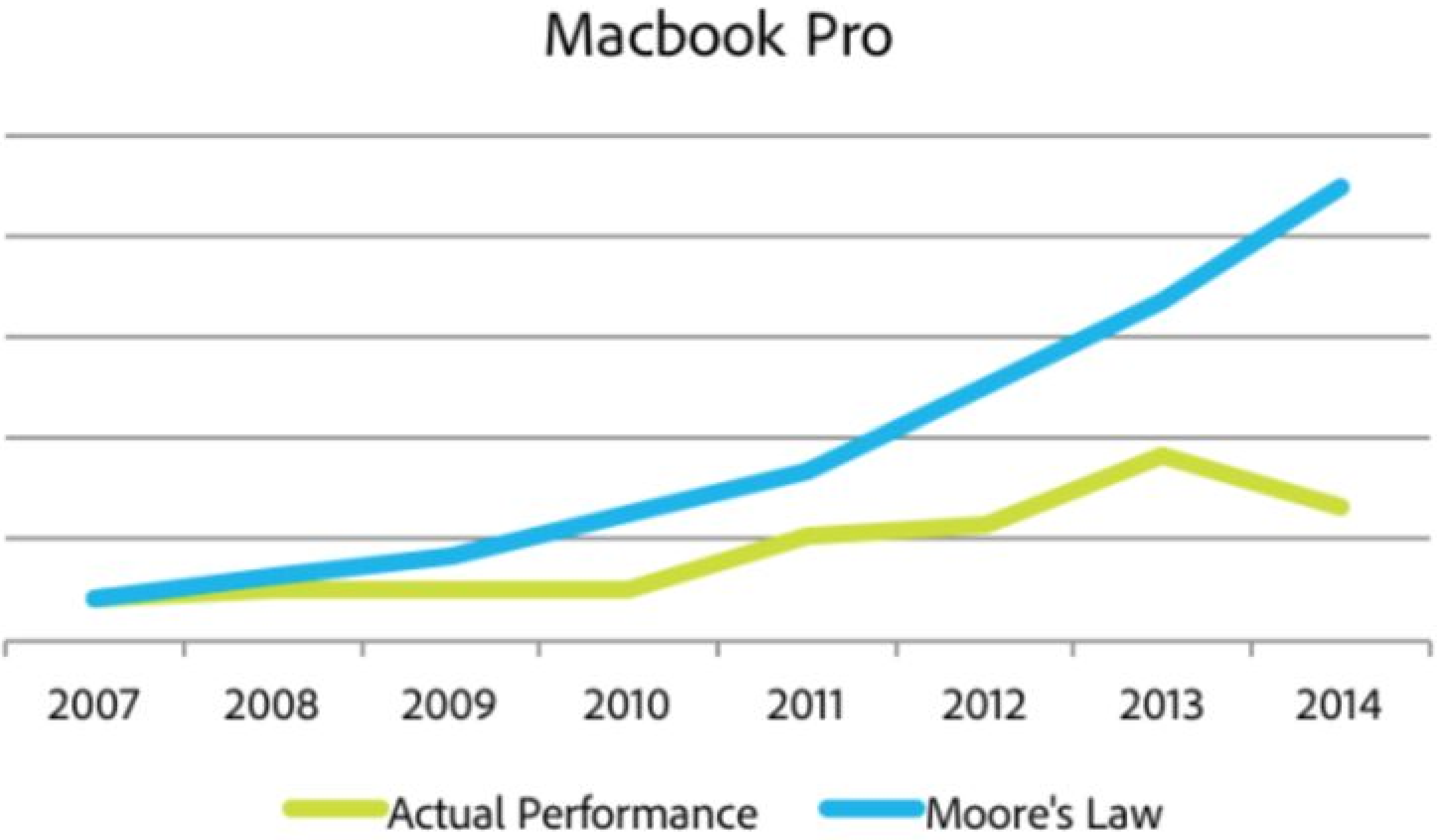
“Many mobile systems (smartphones, electronic glass, autonomous robots) can capture images. These systems use batteries and energy conservation is essential. This challenge aims to discover the best technology in both image recognition and energy conservation. Winners will be evaluated based on both high recognition accuracy and low power usage.”

http://lpirc.net

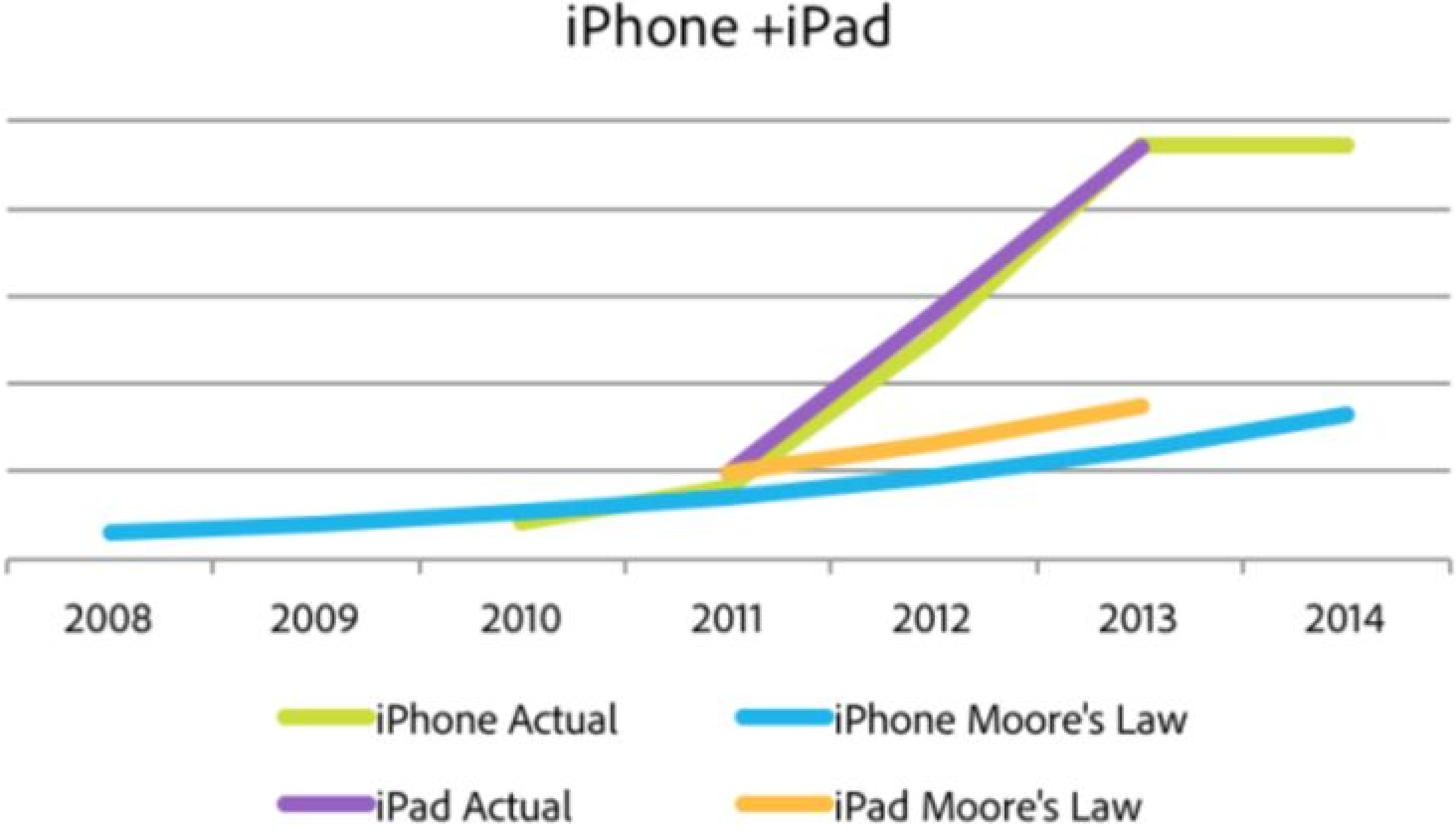
Moore’s Law



SOURCE: RAY KURZWEIL, "THE SINGULARITY IS NEAR: WHEN HUMANS TRANSCEND BIOLOGY", P.67, *THE VIKING PRESS*, 2006. DATAPOINTS BETWEEN 2000 AND 2012 REPRESENT BCA ESTIMATES.

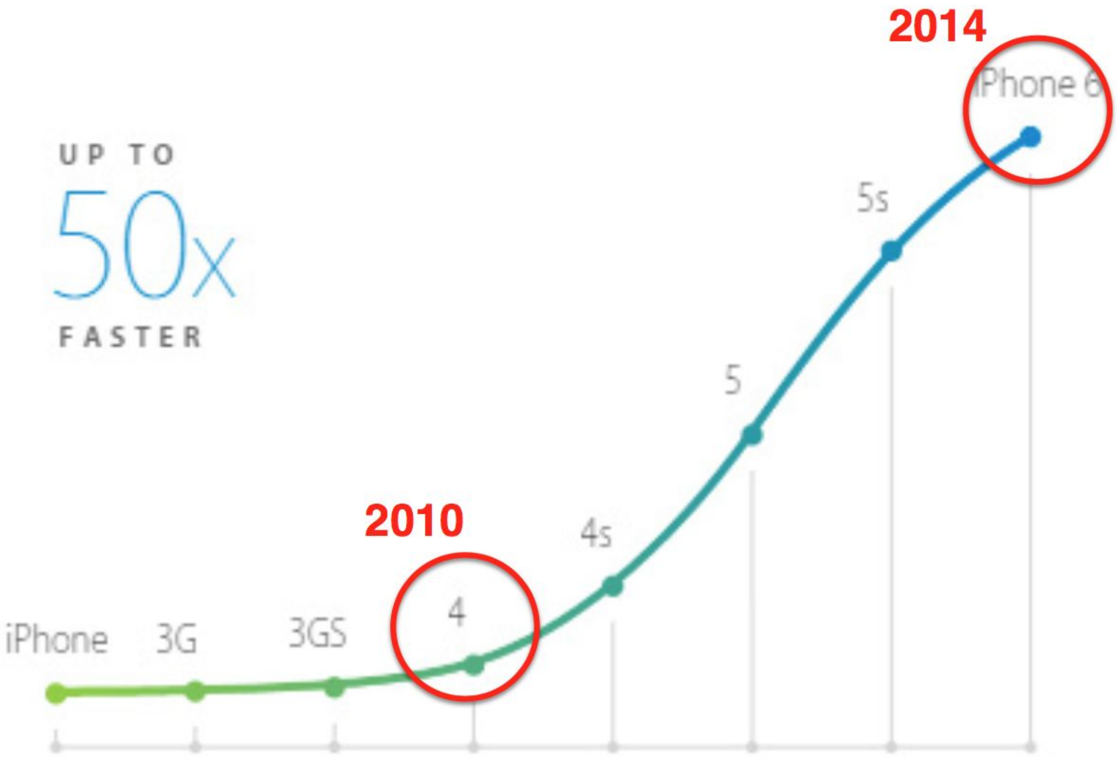


Source: “Moore's Law Is Dead! (But Not In Mobile)” ReadWrite, April 2015.



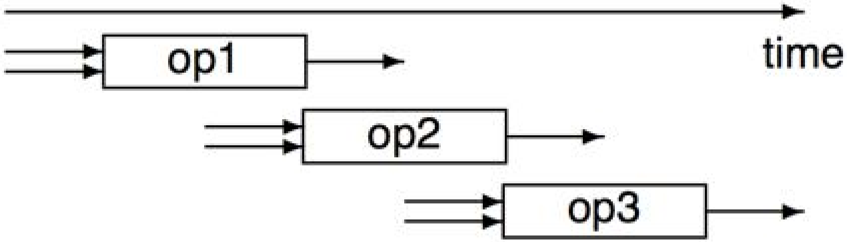
Source: “Moore's Law Is Dead! (But Not In Mobile)” ReadWrite, April 2015.

CPU performance



Ideal Von Neumann Processor

- Each cycle, CPU takes data from registers, does an operation, and puts the result back
- Load/store operations (memory ↔ registers) also take one cycle
- CPU can do different operations each cycle output of one operation can be input to next



- CPU's haven't been this simple for a long time!

http://people.maths.ox.ac.uk/gilesm/cuda/lecs/lec0.pdf

CPU clock is stuck!!!!

- CPU clock stuck at about 3GHz since 2006 due to high power consumption (up to 130W per chip)
- chip circuitry still doubling every 18-24 months

  ⇒ more on-chip memory and MMU (memory management units)
  ⇒ specialised hardware (e.g. multimedia, encryption)
  ⇒ multi-core (multiple CPU’s on one chip)

- peak performance of chip still doubling every 18-24 months

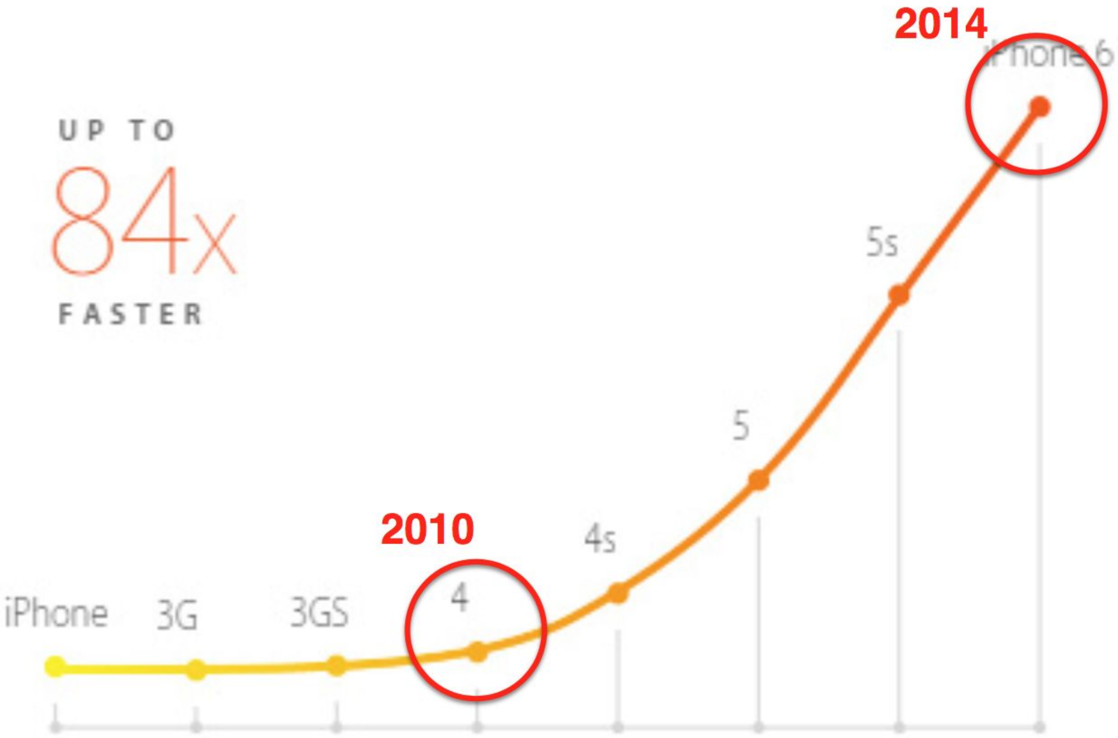
System on a Chip (SoC)

- SoCs attempt to find balance between energy and programmability.
- Designed with emphasis on low power consumption.
- SOC shares the same system bus with CPU, GPU and DSP.
- Therefore has much lower memory bandwidth.
- Useful for computer vision algorithm design as one can switch between CPU and GPU with little memory overhead.
- Not possible on conventional architecture.
- More on this later.....

ARM

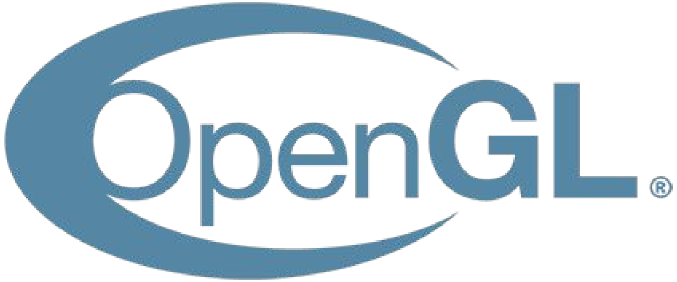
- ARM - originally Acorn RISC Machines.
- Acorn Computers is a British computer manufacturer originally famous the the BBC Micro computer (in the 80s). Common misconception that ARM make chips, they do NOT they instead license the chip designs to third parties.
- Big manufacturers of ARM chips include Apple, Samsung, Qualcomm, etc.
- RISC-based computer design approach means ARM processors require significantly fewer transistors than typical CISC (i.e. x86) based processors.
- Approach reduces costs, heat and power use.

GPU Performance



NVIDIA.
CUDA.

OpenCL


OpenGL®

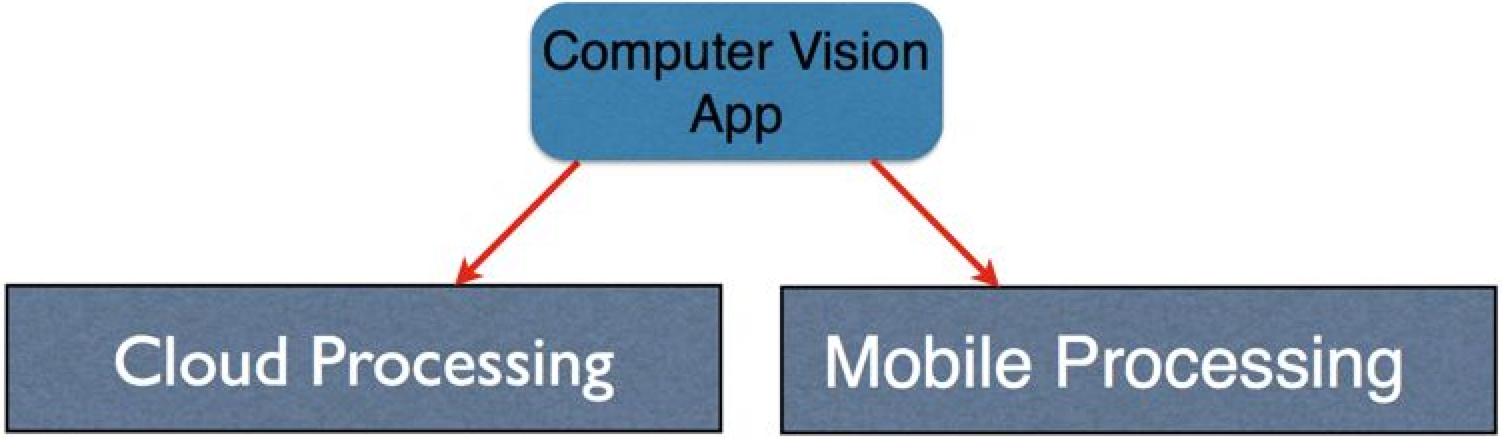
Metal

OpenGL|ES™

Vulkan™

OpenCL vs. CUDA

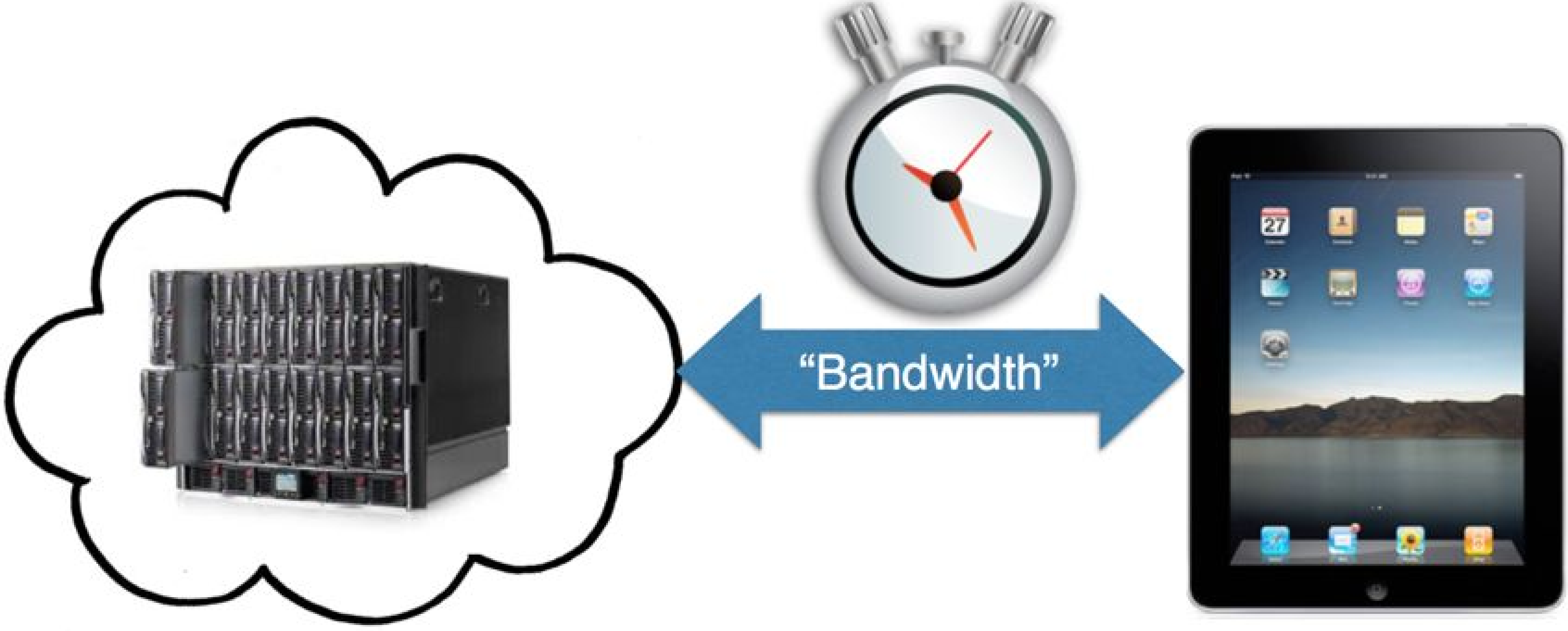
- Open Computing Language (OpenCL)
- OpenCL is the currently the dominant open general- purpose GPU computing language, and is an open standard.
- OpenCL is actively supported on Intel, AMD, Nvidia and ARM platforms.
- OpenCL is based on the C99 language.
- Compute Unified Device Architecture (CUDA)
- Dominant proprietary (NVIDIA) framework.
- Designed to work with well known languages such as C, C++ and Fortran.
- OpenCV 3.0 now has support for both.



Computer Vision App

Cloud Processing

Mobile Processing



“Bandwidth”

Algorithm

Software

Architecture

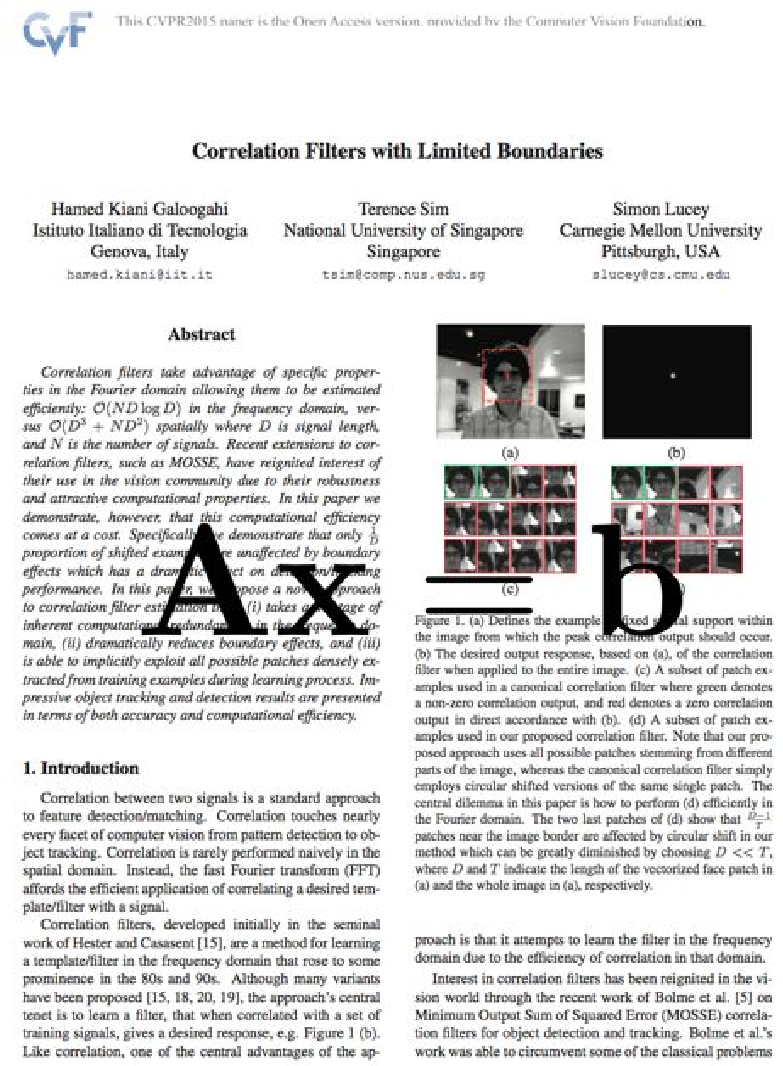
SOC Hardware

Algorithm

Software

Architecture

SOC Hardware



$$\mathbf{A}\mathbf{x} = \mathbf{b}$$

Correlation Filters with Limited Boundaries

Hamed Kiani Galoogahi
Istituto Italiano di Tecnologia
Genova, Italy
hamed.kiani@iit.it

Terence Sim
National University of Singapore
Singapore
tsim@comp.nus.edu.sg

Simon Lucey
Carnegie Mellon University
Pittsburgh, USA
slucey@cs.cmu.edu

Abstract

Correlation filters take advantage of specific properties in the Fourier domain allowing them to be estimated efficiently: $\mathcal{O}(ND \log D)$ in the frequency domain, versus $\mathcal{O}(D^3 + ND^2)$ spatially where D is signal length, and N is the number of signals. Recent extensions to correlation filters, such as MOSSE, have reignited interest of their use in the vision community due to their robustness and attractive computational properties. In this paper we demonstrate, however, that this computational efficiency comes at a cost. Specifically we demonstrate that only $\frac{1}{D}$ proportion of shifted examples are unaffected by boundary effects which has a dramatic effect on detection/tracking performance. In this paper, we propose a novel approach to correlation filter estimation that: (i) takes advantage of inherent computational redundancies in the frequency domain, (ii) dramatically reduces boundary effects, and (iii) is able to implicitly exploit all possible patches densely extracted from training examples during learning process. Impressive object tracking and detection results are presented in terms of both accuracy and computational efficiency.

1. Introduction

Correlation between two signals is a standard approach to feature detection/matching. Correlation touches nearly every facet of computer vision from pattern detection to object tracking. Correlation is rarely performed naively in the spatial domain. Instead, the fast Fourier transform (FFT) affords the efficient application of correlating a desired template/filter with a signal.

Correlation filters, developed initially in the seminal work of Hester and Casasent [15], are a method for learning a template/filter in the frequency domain that rose to some prominence in the 80s and 90s. Although many variants have been proposed [15, 18, 20, 19], the approach's central tenet is to learn a filter, that when correlated with a set of training signals, gives a desired response, e.g. Figure 1 (b). Like correlation, one of the central advantages of the approach is that it attempts to learn the filter in the frequency domain due to the efficiency of correlation in that domain.

Figure 1. (a) Defines the example of fixed spatial support within the image from which the peak correlation output should occur. (b) The desired output response, based on (a), of the correlation filter when applied to the entire image. (c) A subset of patch examples used in a canonical correlation filter where green denotes a non-zero correlation output, and red denotes a zero correlation output in direct accordance with (b). (d) A subset of patch examples used in our proposed correlation filter. Note that our proposed approach uses all possible patches stemming from different parts of the image, whereas the canonical correlation filter simply employs circular shifted versions of the same single patch. The central dilemma in this paper is how to perform (d) efficiently in the Fourier domain. The two last patches of (d) show that $\frac{D-1}{T}$ patches near the image border are affected by circular shift in our method which can be greatly diminished by choosing $D << T$, where D and T indicate the length of the vectorized face patch in (a) and the whole image in (a), respectively.

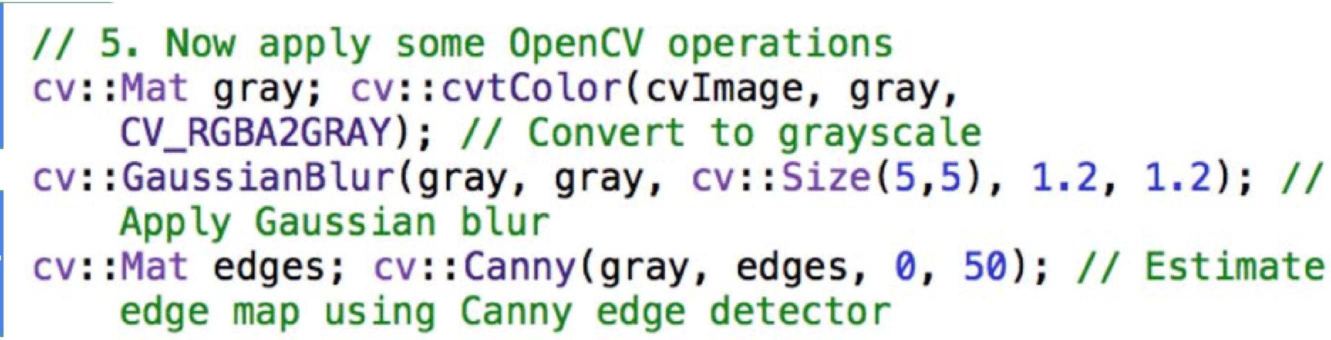
Interest in correlation filters has been reignited in the vision world through the recent work of Bolme et al. [5] on Minimum Output Sum of Squared Error (MOSSE) correlation filters for object detection and tracking. Bolme et al.'s work was able to circumvent some of the classical problems

Algorithm

Software

Architecture

SOC Hardwar



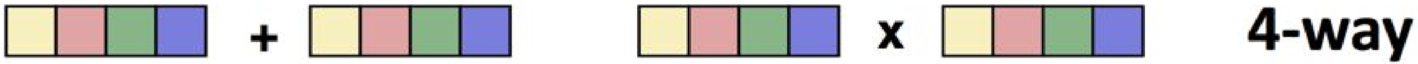
```
// 5. Now apply some OpenCV operations
cv::Mat gray; cv::cvtColor(cvImage, gray,
    CV_RGBA2GRAY); // Convert to grayscale
cv::GaussianBlur(gray, gray, cv::Size(5,5), 1.2, 1.2); //
    Apply Gaussian blur
cv::Mat edges; cv::Canny(gray, edges, 0, 50); // Estimate
    edge map using Canny edge detector
```

Algorithm

Software

Architecture

SOC Hardware



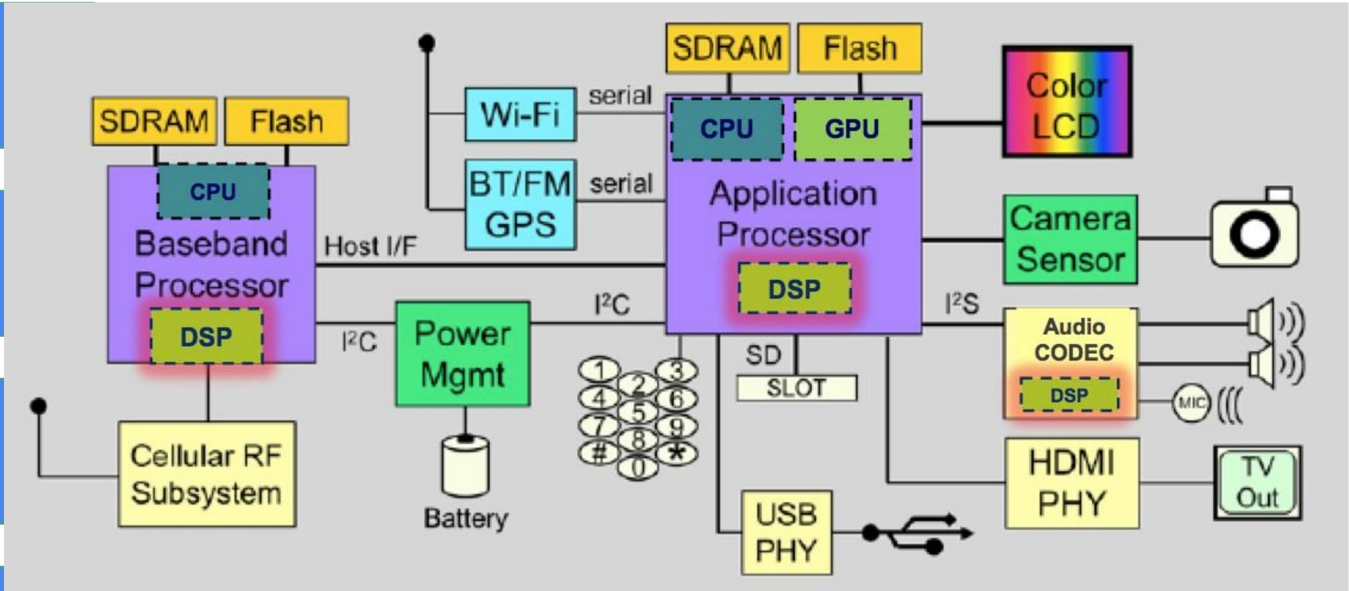
SIMD (Single Instruction, Multiple Data)

Algorithm

Software

Architecture

SOC Hardware



Some Insights for Mobile CV

- Very difficult to write the fastest code.
  - When you are prototyping an idea you should not worry about this, but
  - You have to be aware of where bottlenecks can occur.
- Highest performance in general is non-portable.
  - If you want to get the most out of your system it is good to go deep.
  - However, options like OpenCV are good when you need to build something quickly that works.
- To build good computer vision apps you need to know them algorithmically.
  - Simply knowing how to write fast code is not enough.
  - You need to also understand computer vision algorithmically.
  - OpenCV can be dangerous here.

OpenCV

- An open source BSD licensed computer vision library.
  - Patent-encumbered code isolated into “non-free” module.
  - SIFT, SURF, some of the Face Detectors, etc.
- Available on all major platforms
  - Android, iOS, Linux, Mac OS X, Windows
- Written primarily in C++
  - Bindings available for Python, Java, even MATLAB (in 3.0).
- Well documented at http://docs.opencv.org
- Source available at https://github.com/Itseez/opencv

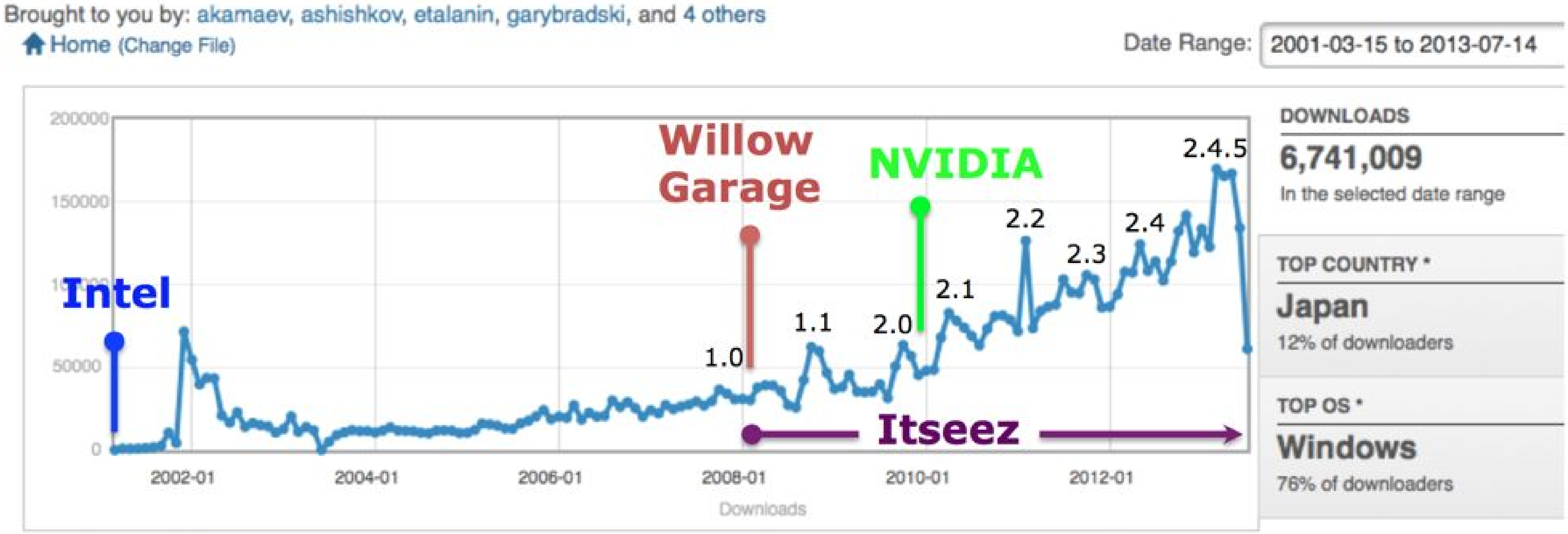
History of OpenCV

- OpenCV started by Intel Research in 1998.
- Goals originally were:
  - Advance vision research by providing not only open but also optimized code for basic vision infrastructure. No more reinventing the wheel.
  - Disseminate vision knowledge by providing a common infrastructure that developers could build on, so that code would be more readily readable and transferable.
  - Advance vision-based commercial applications by making portable, performance-optimized code available for free—with a license that did not require to be open or free themselves.
- Originally released at CVPR 2000.

OpenCV then and now.....

- Version 1.0 was released in 2006.
- In 2008 obtained corporate support from Willow Garage (Robotics Company).
- OpenCV 2 was released in 2009.
  - Included major changes for C++ (mostly C beforehand).
- In 2012 support for OpenCV was taken over by a non-profit foundation OpenCV.org.
- OpenCV 3 was released in 2014.
  - Seems to be under corporate support from Itseez.
  - More on these changes soon.

OpenCV then and now.....



Brought to you by: akamaev, ashishkov, etalanin, garybradski, and 4 others

Home (Change File)

Date Range: 2001-03-15 to 2013-07-14

DOWNLOADS

6,741,009

In the selected date range

TOP COUNTRY *

Japan

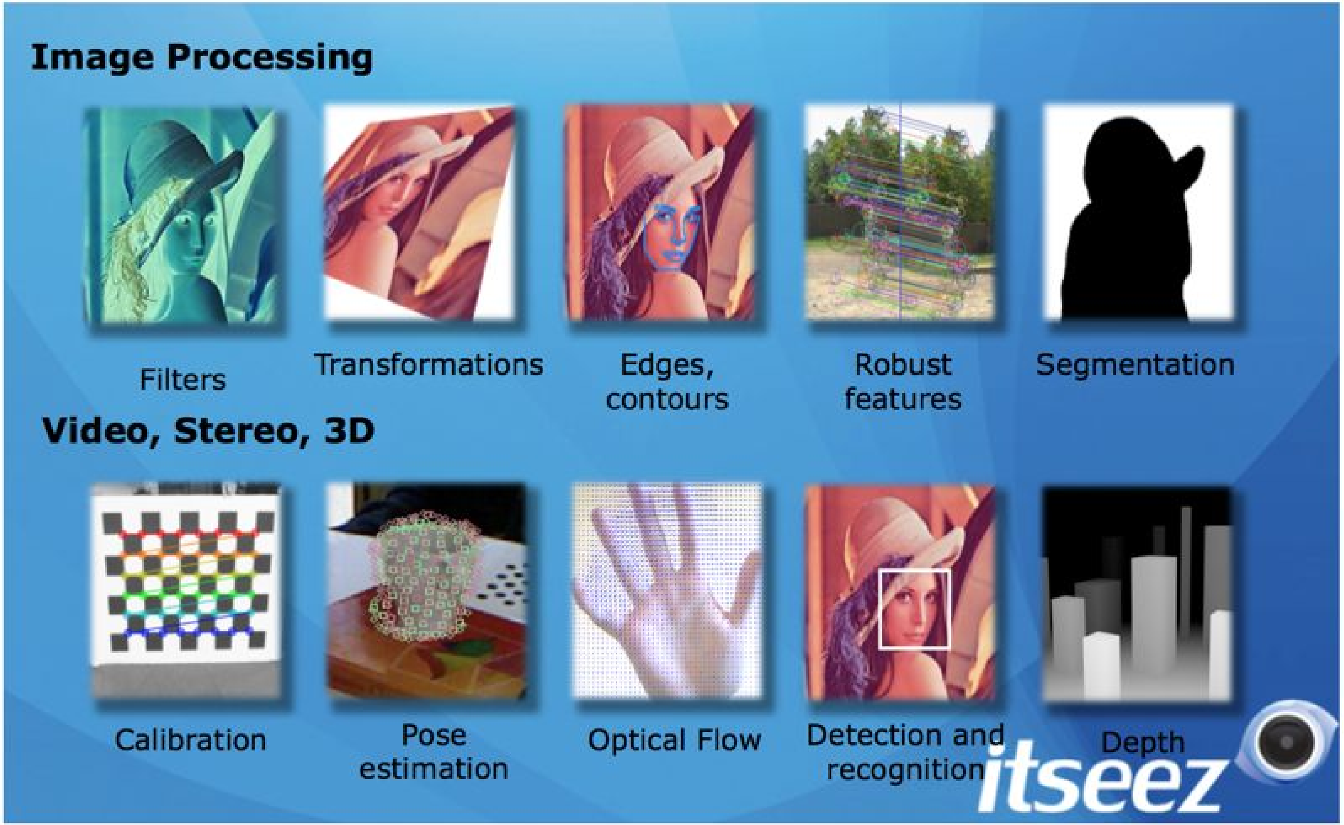
12% of downloaders

TOP OS *

Windows

76% of downloaders

What can OpenCV do?



OpenCV 3.0

Migration is relatively smooth from 2.4

- Mostly cleanings – Refined C++ API
  – Use cv::Algorithm everywhere
- API changes
  – C API will be marked as deprecated
  – Old Python API will be deprecated
  – Monstrous modules will be split into micro modules – Extra modules

OpenCV 3.0

- Sufficiently improved CUDA and OpenCL modules
  - Mobile CUDA support
  - Universal OpenCL binaries (CPU, GPU)
- Hardware Abstraction Layer (HAL)
  - IPP, FastCV-like low-level API to accelerate OpenCV on different HW.
- Check out the transition guide.

Which version will be using?

- OpenCV 3.0 is brand new, and is well worth a look and play.
- Most vision tutorials are still in OpenCV 2.4.X.
- OpenCV 2.4.X is still the de facto library for computer vision and image processing.
- Recommend to use 2.4. (but example code will use 3.1)
- Caution!
    - Danger with OpenCV is that it allows you to do a lot with very little understanding for what is going on.
    - It is also assumed that you know C++ going forward.

Key OpenCV Classes

Point_	Template 2D point class
Point3_	Template 3D point class
Size_	Template size (width, height) class
Vec	Template short vector class
Matx	Template small matrix class
Scalar	4-element vector
Rect	Rectangle
Range	Integer value range
Mat	2D or multi-dimensional dense array (can be used to store matrices, images, histograms, feature descriptors, voxel volumes etc.)
SparseMat	Multi-dimensional sparse array
Ptr	Template smart pointer class

OpenCV Installation and cheatsheet

- OpenCV_3_Windows_10_Installation_Tutorial
  - Installation Cheat Sheet 1 - OpenCV 3 and C++.pdf
- C++ cheatsheet (only for version 2.4)!
- OpenCV: Transition guide http://docs.opencv.org/3.1.0/db/dfa/tutorial_transition_guide.html

Playing with the Mat Object Class

- We are now going to have a play with the Mat Object Class.
- Check link below https://github.com/CSE6000/Fall2016/tree/master/codes/OpenCV/Example_OCV
- If you use in mac OS, use link below https://github.com/slucey-cs-cmu-edu/Example_OCV
- See if you can run make on the command line to create the `Example_OCV` executable.

Displaying an Image in OpenCV

- On your browser please go to the address, https://github.com/CSE6000/Fall2016/tree/master/codes/OpenCV/Show_Lena
- Or again, you can use link below if you use mac OS. https://github.com/slucey-cs-cmu-edu/Show_Lena
- Question: what happens if you set the imread flag to 1?

Detecting a Face in OpenCV

- https://github.com/CSE6000/Fall2016/tree/master/codes/OpenCV/Detect_Lena
- mac OS https://github.com/slucey-cs-cmu-edu/Detect_Lena
- Questions: why do you need to clone the Mat image when displaying?