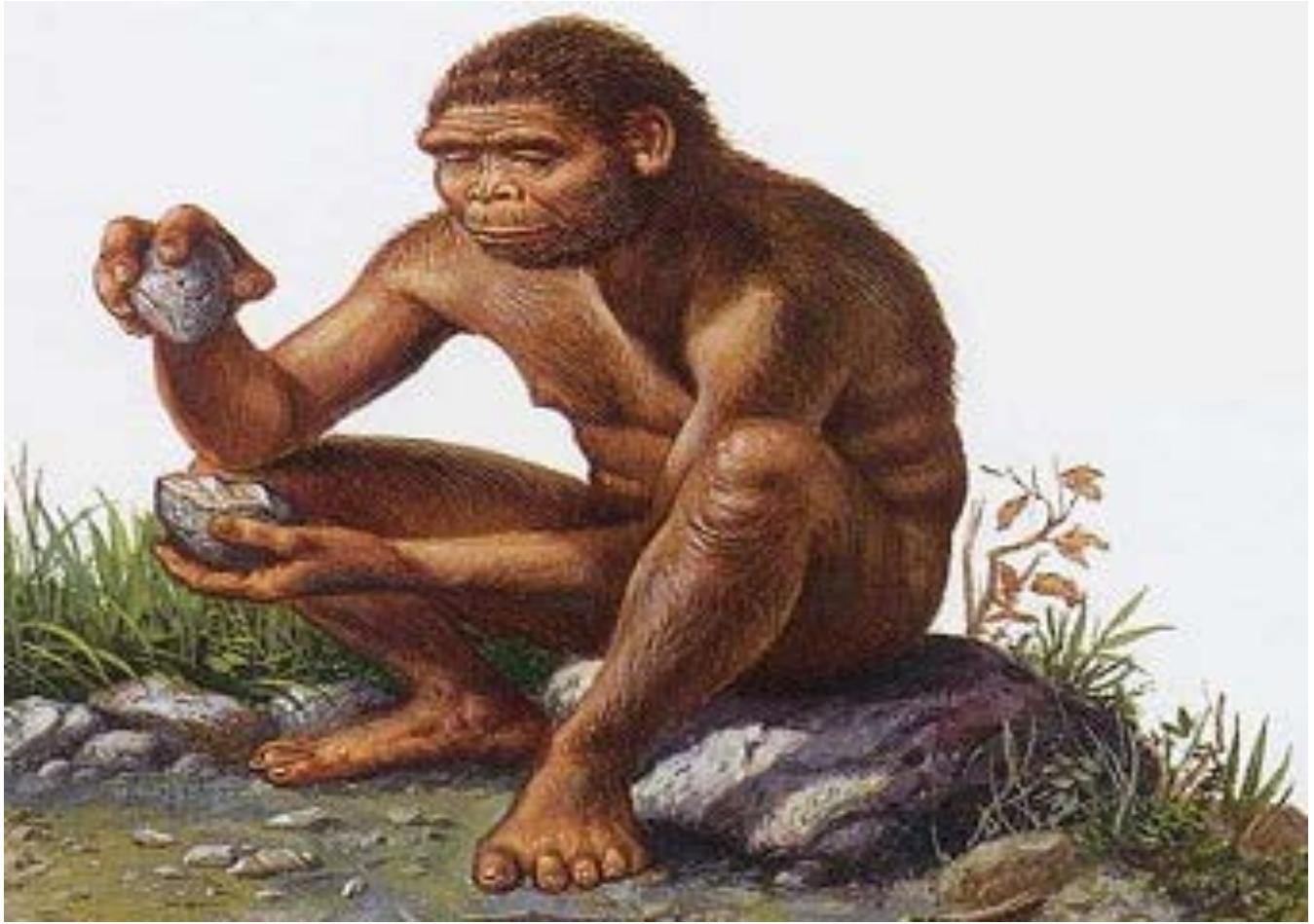
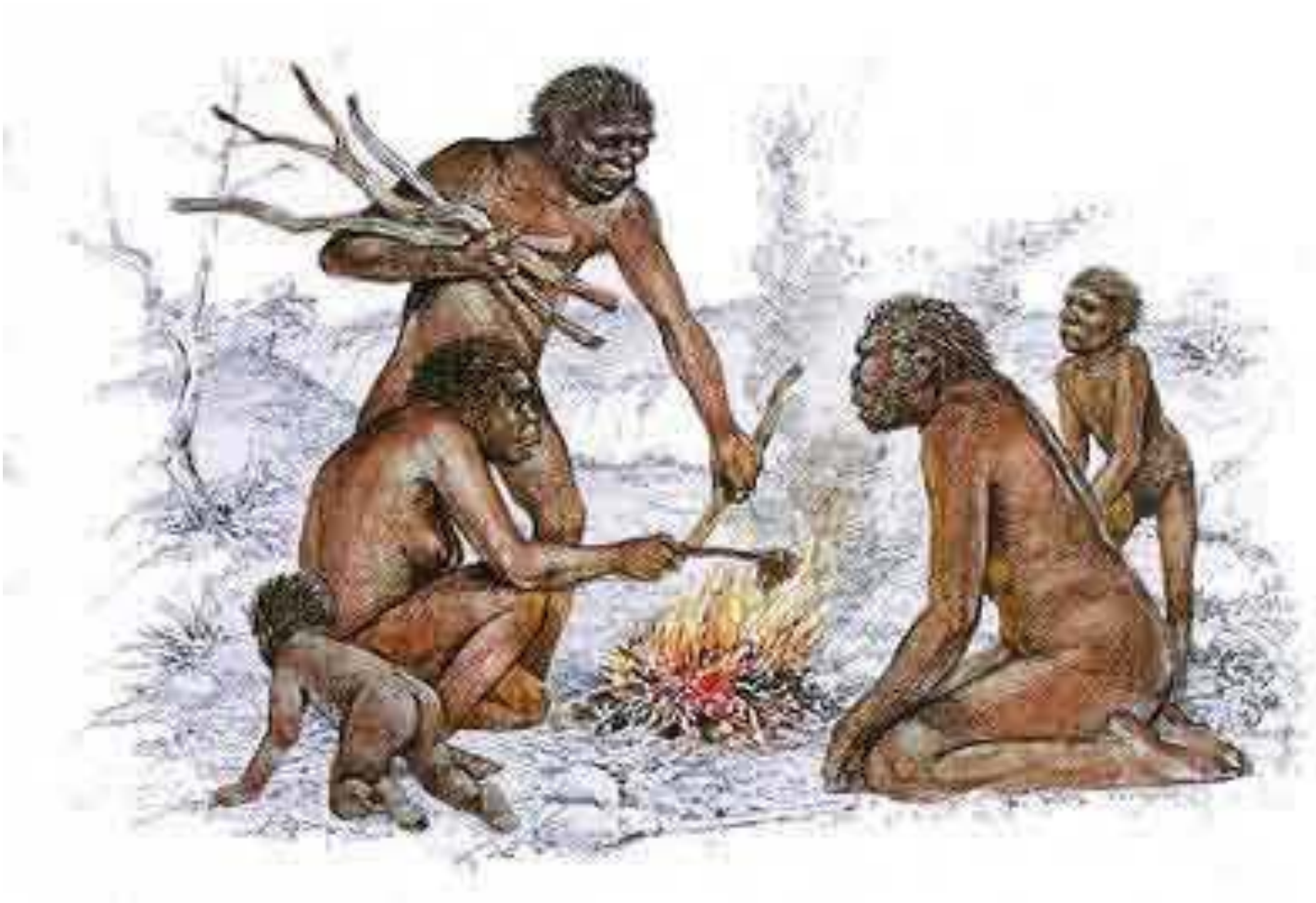
Building the Software 2.0 Stack

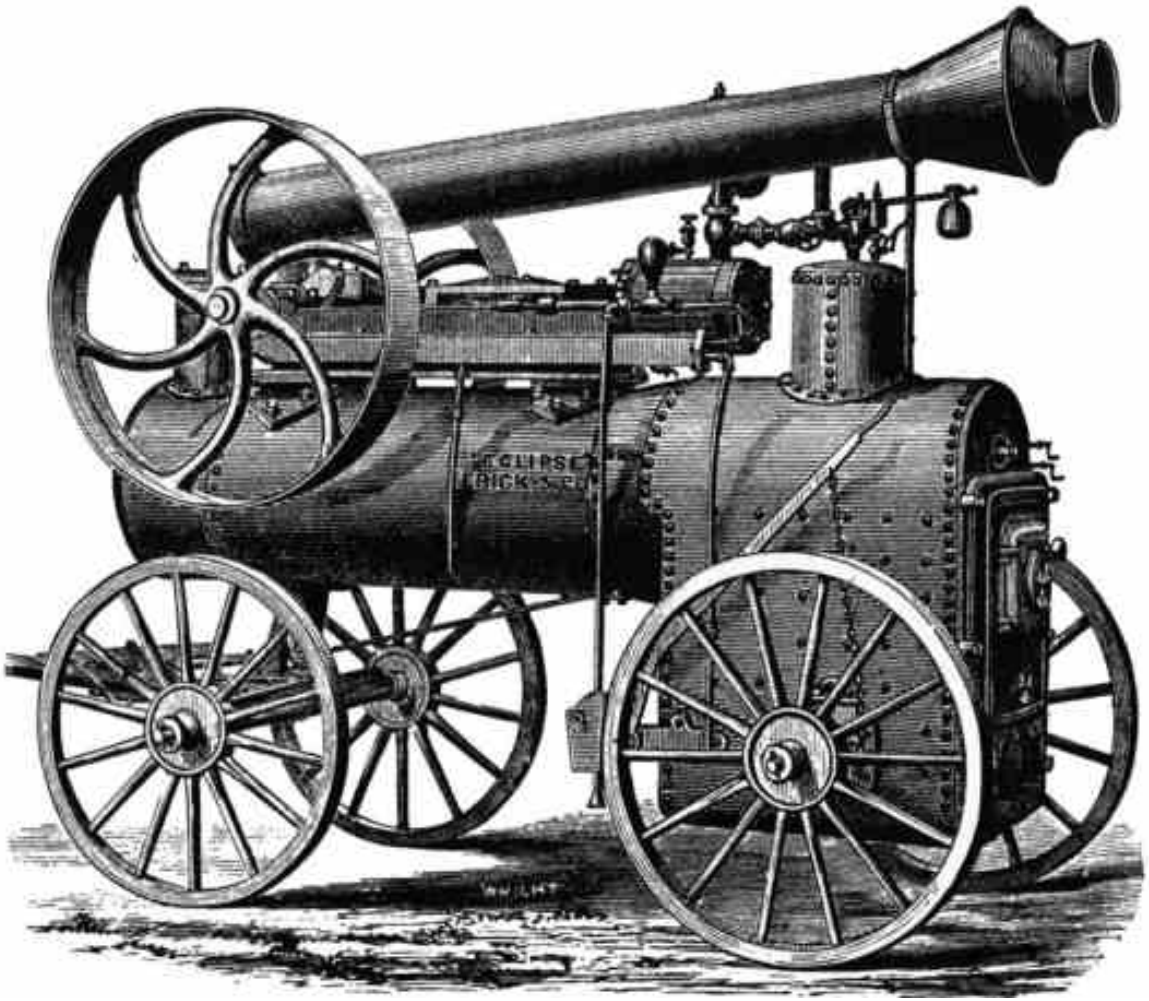
Andrej Karpathy

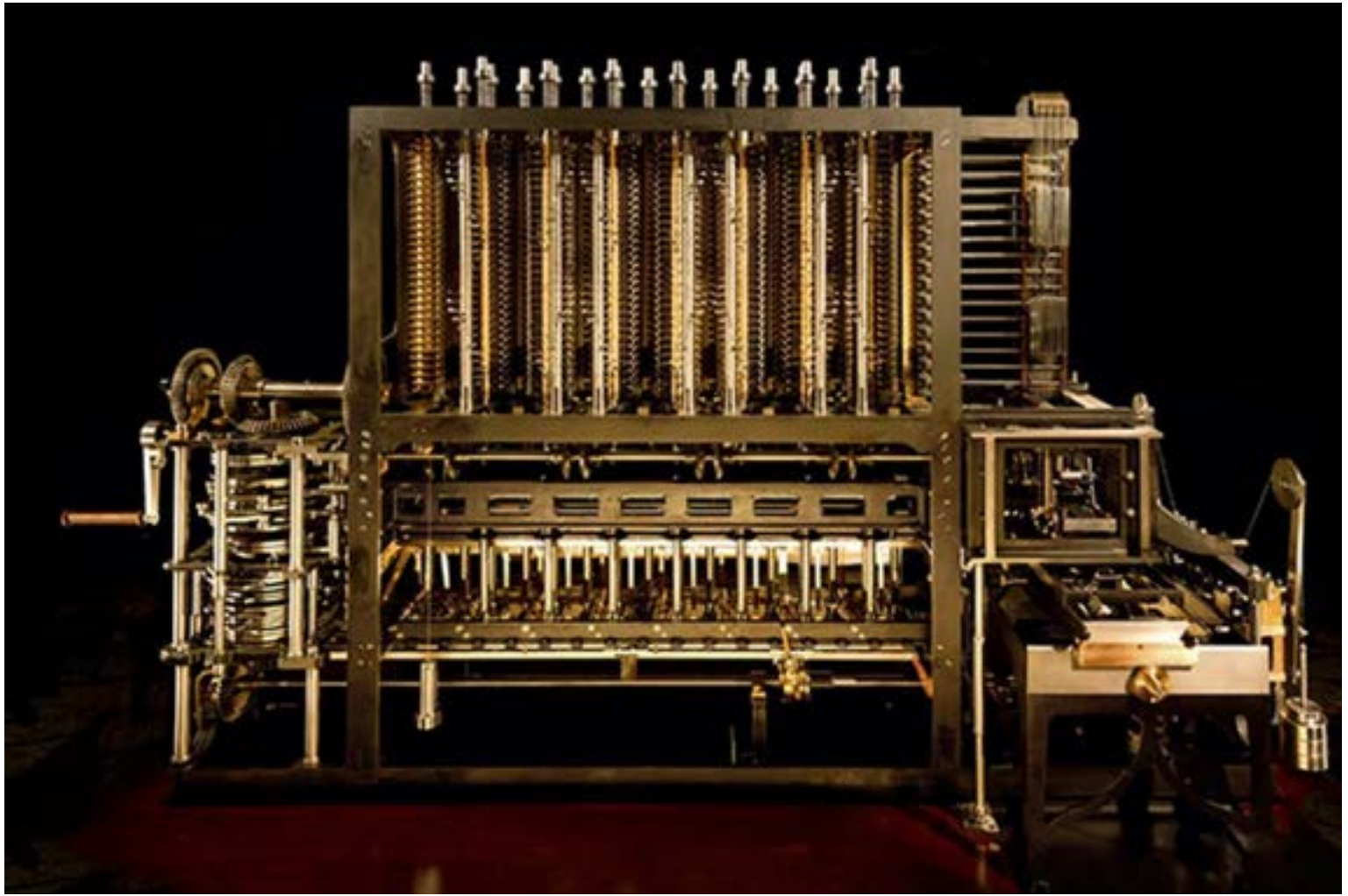
May 10, 2018

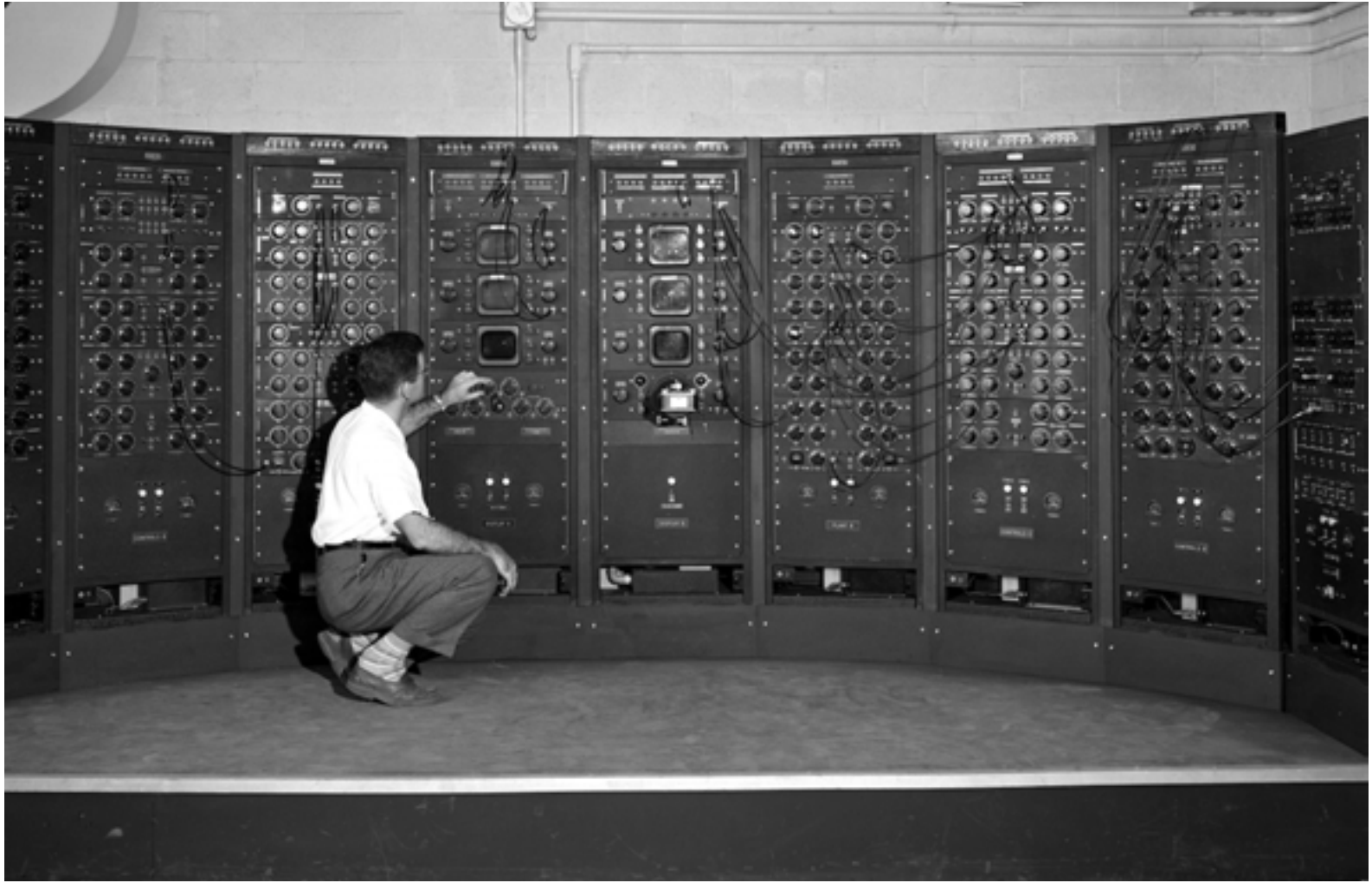
1M years ago









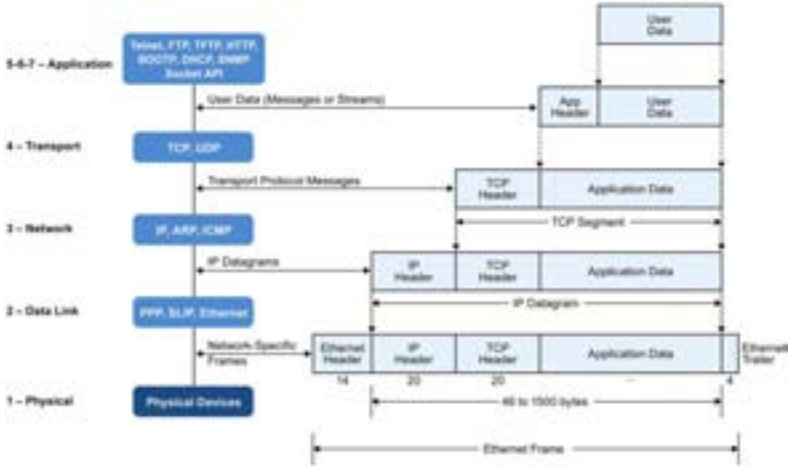


Engineering: approach by decomposition

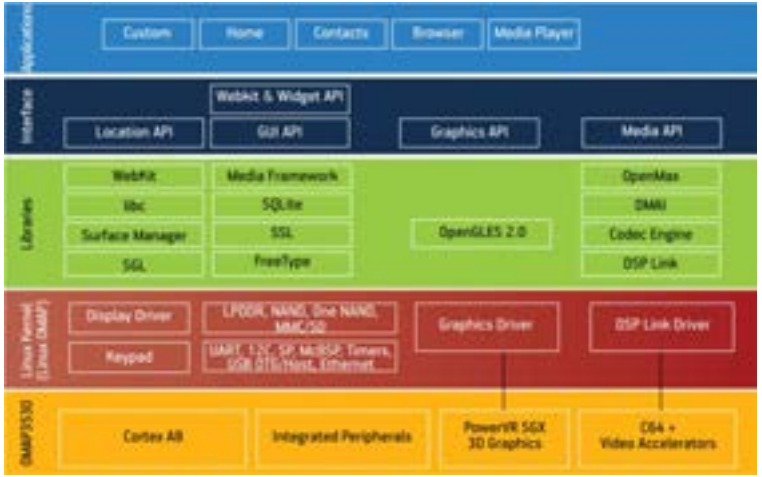
1. Identify a problem
2. Break down a big problem to smaller problems
3. Design algorithms for each individual problem
4. Compose solutions into a system (get a “stack”)

AWS stack

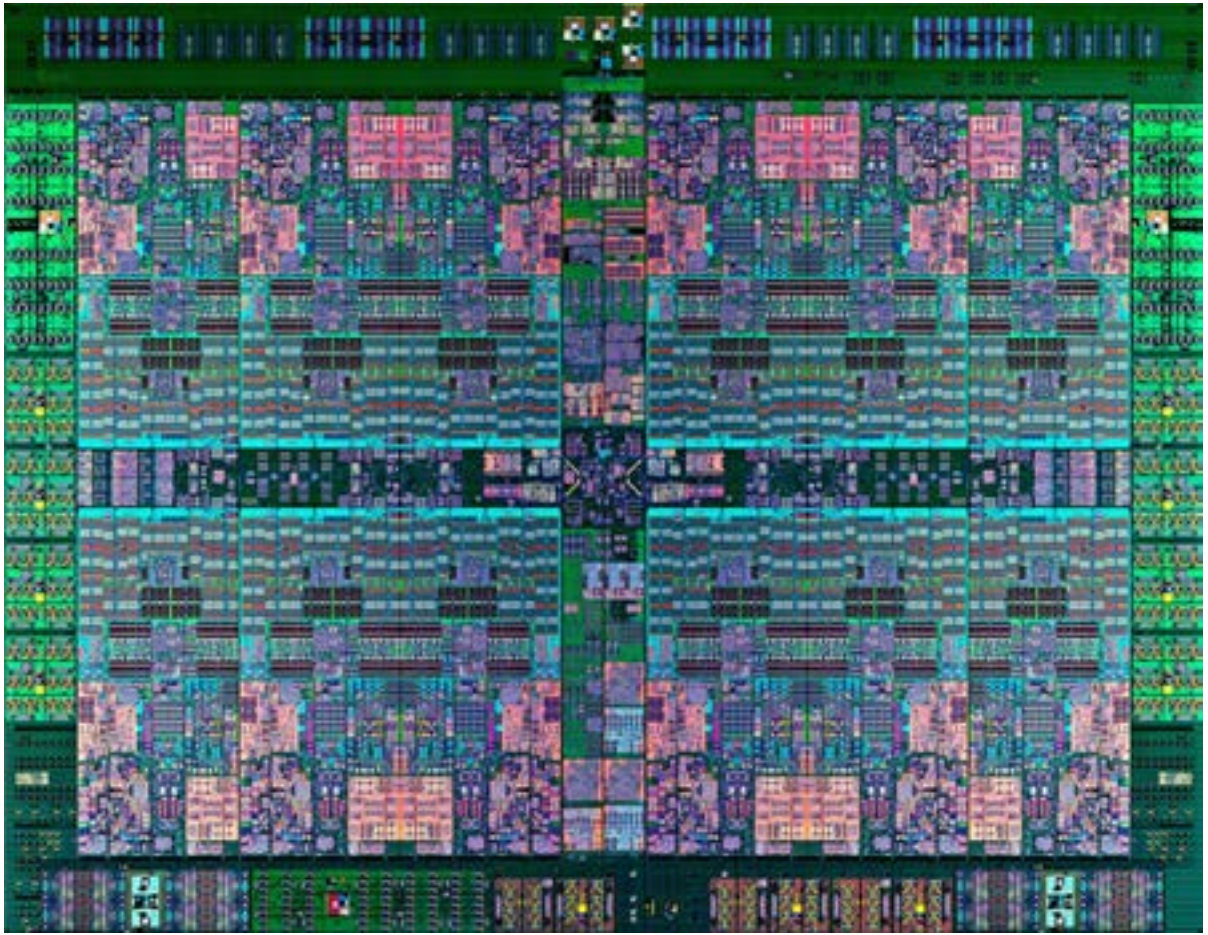
TCP/IP stack



Android software stack



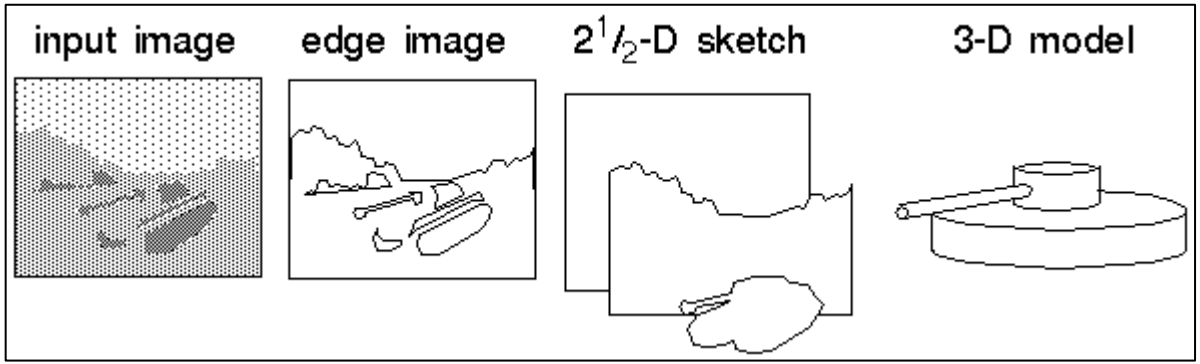
We got surprisingly far...

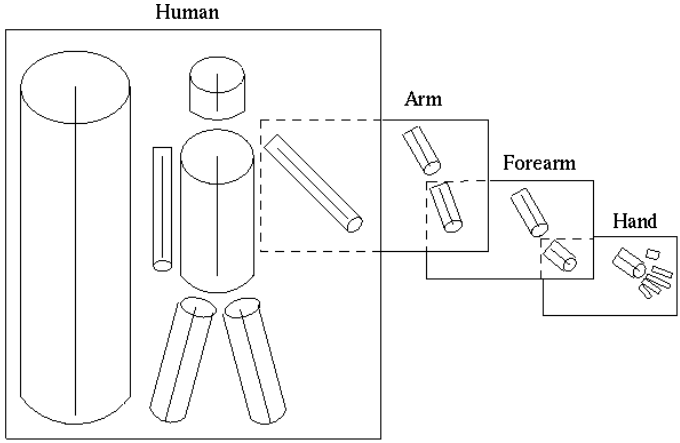


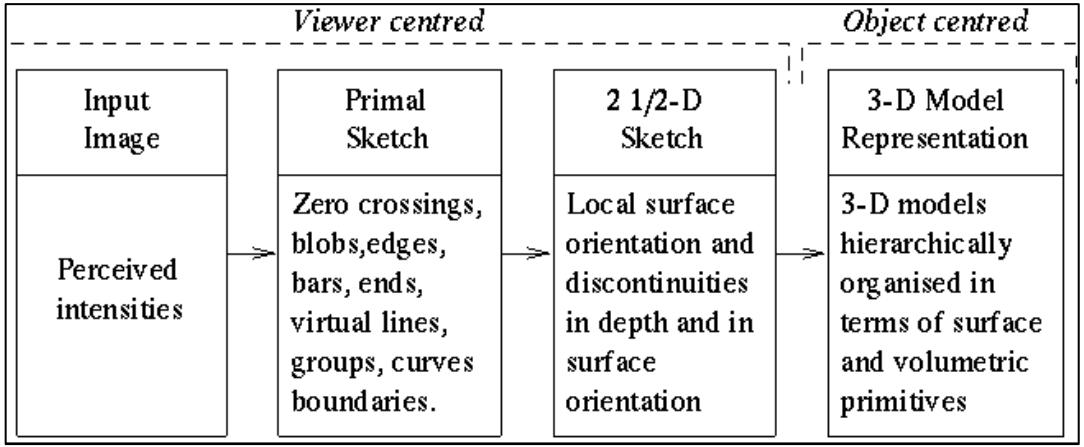
What is the
“recognition stack”?

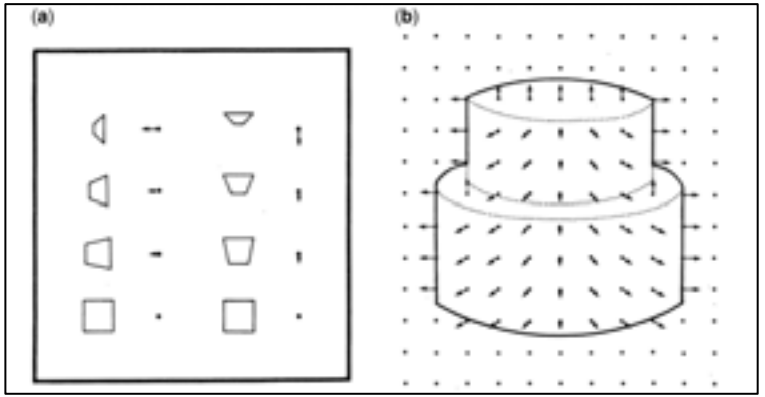
“cat”

Visual Recognition: 1980 ~ 1990



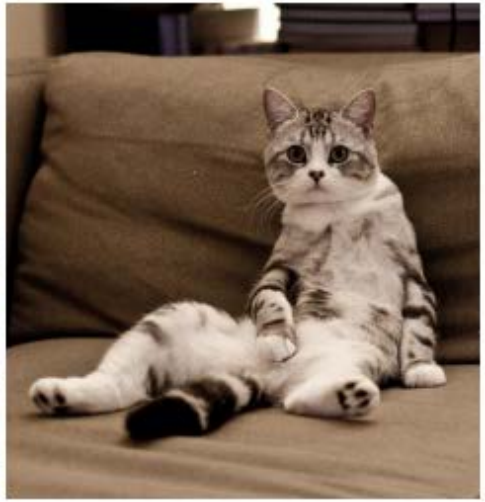






David Marr, Vision





Visual Recognition: ~1990 - 2010

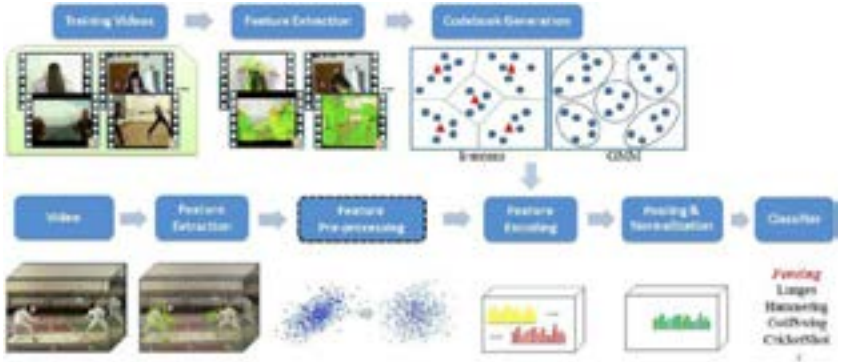
vector describing
various image statistics

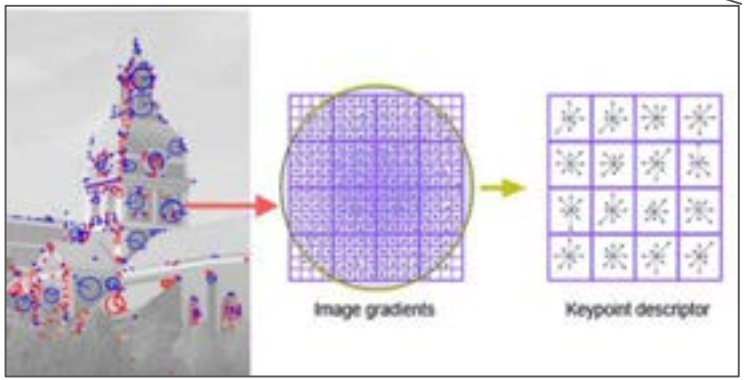
Feature Extraction

f

training

1000 numbers,
indicating class scores





Computer Vision **2011**

4.1. Image Features and Kernels

We selected or designed several state-of-art features that are potentially useful for scene classification. GIST features [21] are proposed specifically for scene recognition tasks. Dense SIFT features are also found to perform very well at the 15-category dataset [17]. We also evaluate sparse SIFTs as used in "Video Google" [27]. HOG features provide excellent performance for object and human recognition tasks [4, 9], so it is interesting to examine their utility for scene recognition. While SIFT is known to be very good at finding repeated image content, the self-similarity descriptor (SSIM) [26] relates images using their internal layout of local self-similarities. Unlike GIST, SIFT, and HOG, which are all local gradient-based approaches, SSIM may provide a distinct, complementary measure of scene layout that is somewhat appearance invariant. As a baseline, we also include Tiny Images [28], color histograms and straight line histograms. To make our color and texton histograms more invariant to scene layout, we also build histograms for specific geometric classes as determined by [13]. The geometric classification of a scene is then itself used as a feature, hopefully being invariant to appearance but responsive to layout.

GIST: The GIST descriptor [21] computes the output energy of a bank of 24 filters. The filters are Gabor-like filters tuned to 8 orientations at 4 different scales. The square output of each filter is then averaged on a 4×4 grid.

HOG2x2: First, histogram of oriented edges (HOG) descriptors [4] are densely extracted on a regular grid at steps of 8 pixels. HOG features are computed using the code available online provided by [9], which gives a 31-dimension descriptor for each node of the grid. Then, 2×2 neighboring HOG descriptors are stacked together to form a descriptor with 124 dimensions. The stacked descriptors spatially overlap. This 2×2 neighbor stacking is important because the higher feature dimensionality provides more descriptive power. The descriptors are quantized into 300 visual words by k-means. With this visual word representation, three-level spatial histograms are computed on grids of 1×1, 2×2 and 4×4. Histogram intersection[17] is used to define the similarity of two histograms at the same pyramid level for two images. The kernel matrices at the three levels are normalized by their respective means, and linearly combined together using equal weights.

Dense SIFT: As with HOG2x2, SIFT descriptors are densely extracted [17] using a flat rather than Gaussian window at two scales (4 and 8 pixel radii) on a regular grid at steps of 5 pixels. The three descriptors are stacked together for each HSV color channels, and quantized into 300 visual words by k-means, and spatial pyramid histograms are used as kernels[17].

Computer Vision **2011**

LBP: Local Binary Patterns (LBP) [20] is a powerful texture feature based on occurrence histogram of local binary patterns. We can regard the scene recognition as a texture classification problem of 2D images, and therefore apply this model to our problem. We also try the rotation invariant extension version [2] of LBP to examine whether rotation invariance is suitable for scene recognition.

Sparse SIFT histograms: As in "Video Google" [27], we build SIFT features at Hessian-affine and MSER [19] interest points. We cluster each set of SIFTs, independently, into dictionaries of 1,000 visual words using k-means. An image is represented by two histograms counting the number of sparse SIFTs that fall into each bin. An image is represented by two 1,000 dimension histograms where each SIFT is soft-assigned, as in [22], to its nearest cluster centers. Kernels are computed with χ^2 distance.

SSIM: Self-similarity descriptors [26] are computed on a regular grid at steps of five pixels. Each descriptor is obtained by computing the correlation map of a patch of 5×5 in a window with radius equal to 40 pixels, then quantizing it in 3 radial bins and 10 angular bins, obtaining 30 dimensional descriptor vectors. The descriptors are then quantized into 300 visual words by k-means and we use χ^2 distance on spatial histograms for the kernels.

Tiny Images: The most trivial way to match scenes is to compare them directly in color image space. Reducing the image dimensions drastically makes this approach more computationally feasible and less sensitive to exact alignment. This method of image matching has been examined thoroughly by Torralba et al.[28] for the purpose of object recognition and scene classification.

Line Features: We detect straight lines from Canny edges using the method described in Video Compass [15]. For each image we build two histograms based on the statistics of detected lines– one with bins corresponding to line angles and one with bins corresponding to line lengths. We use an RBF kernel to compare these unnormalized histograms. This feature was used in [11].

Texton Histograms: We build a 512 entry universal texton dictionary [18] by clustering responses to a bank of filters with 8 orientations, 2 scales, and 2 elongations. For each image we then build a 512-dimensional histogram by assigning each pixel's set of filter responses to the nearest texton dictionary entry. We compute kernels from normalized χ^2 distances.

Color Histograms: We build joint histograms of color in CIE L*a*b* color space for each image. Our histograms have 4, 14, and 14 bins in L, a, and b respectively for a total of 784 dimensions. We compute distances between these histograms using χ^2 distance on the normalized histograms.

Geometric Probability Map: We compute the geometric class probabilities for image regions using the method of Hoiem et al. [13]. We use only the ground, vertical, porous, and sky classes because they are more reliably classified. We reduce the probability maps for each class to 8×8 and use an RBF kernel. This feature was used in [11].

Computer
Vision
2011

Geometry Specific Histograms: Inspired by "Illumination Context" [16], we build color and texton histograms for each geometric class (ground, vertical, porous, and sky). Specifically, for each color and texture sample, we weight its contribution to each histogram by the probability that it belongs to that geometric class. These eight histograms are compared with χ^2 distance after normalization.

\+ code complexity :(

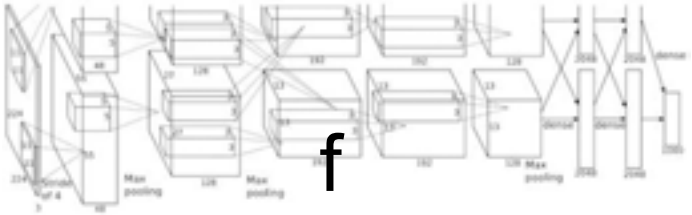
vector describing
various image statistics

Feature Extraction

f

training

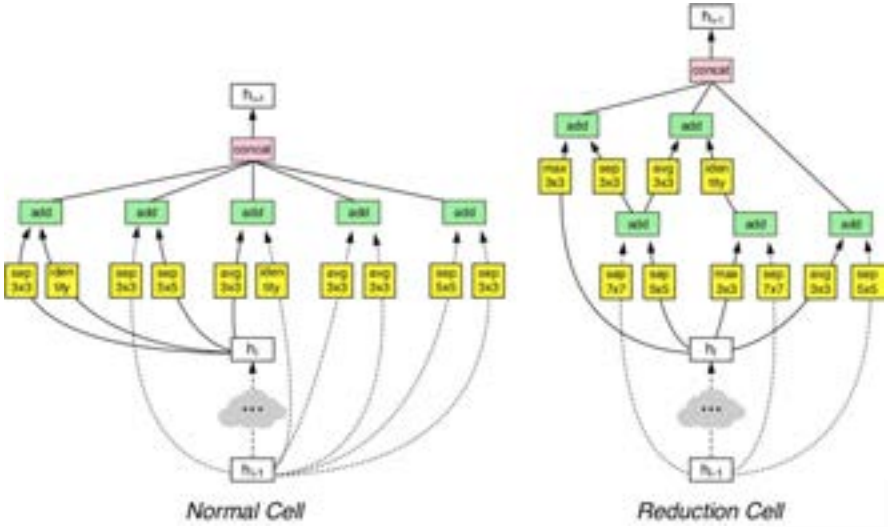
1000 numbers,
indicating class scores

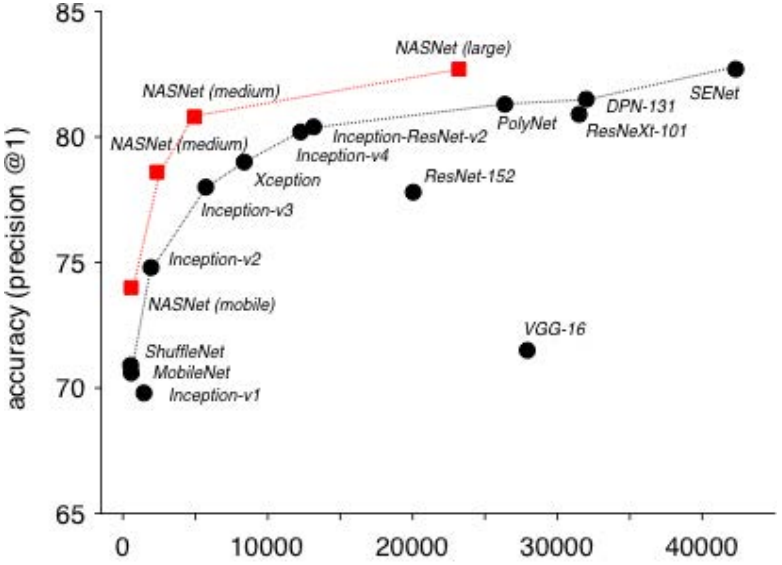


f

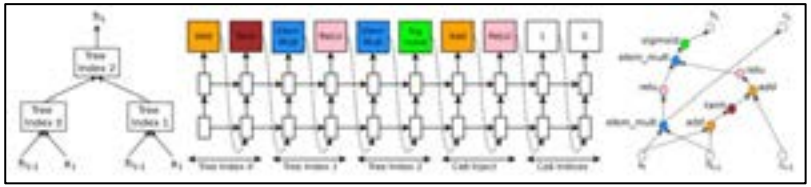
training

1000 numbers,
indicating class scores

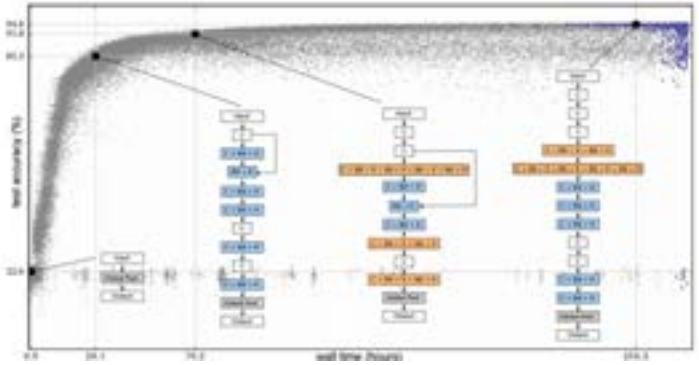




¯_(ツ)_/¯

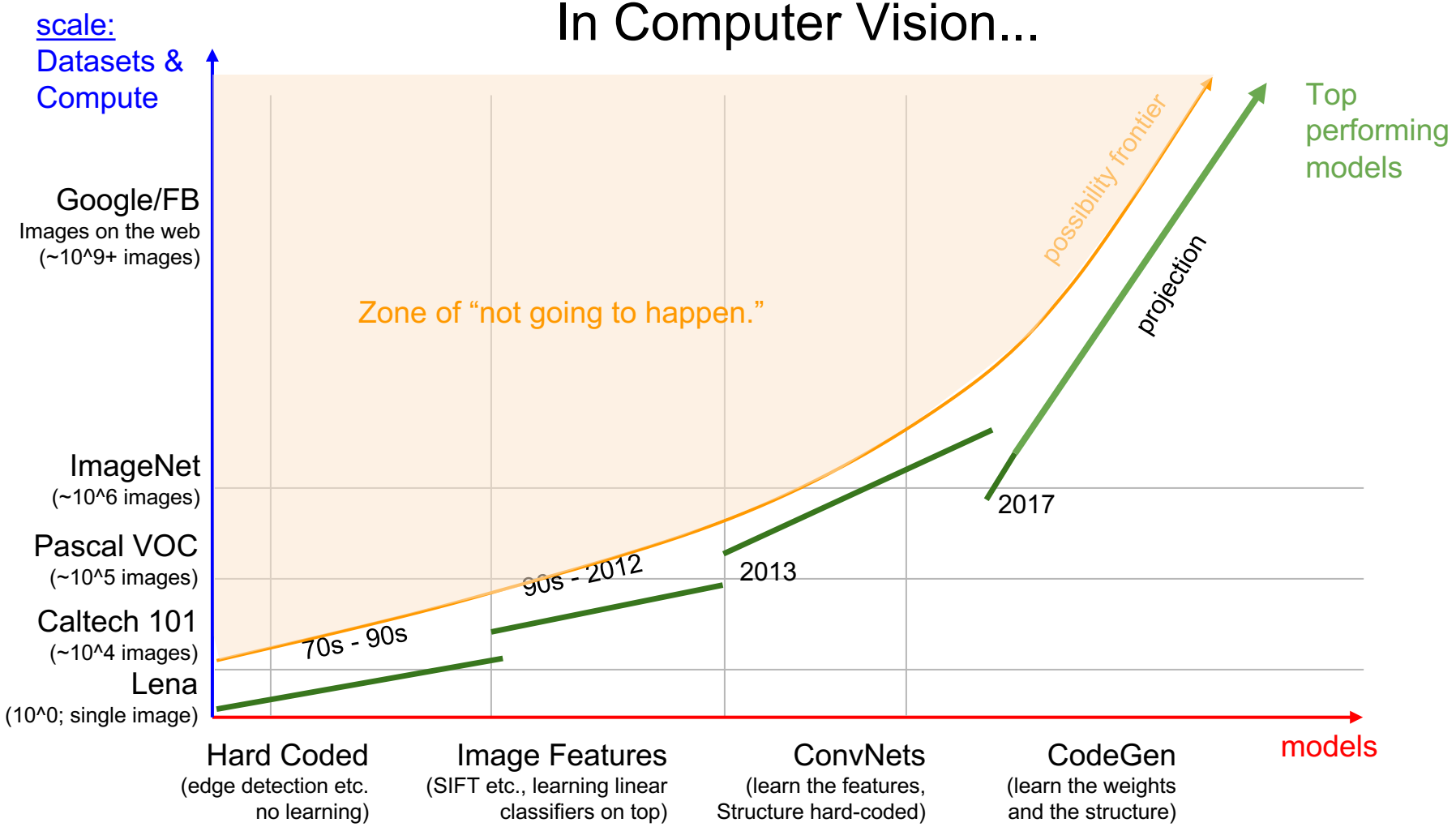


“NEURAL ARCHITECTURE SEARCH WITH REINFORCEMENT LEARNING”, Zoph & Le



Large-Scale Evolution of Image Classifiers
Real et al.

In Computer Vision...



Software 1.0

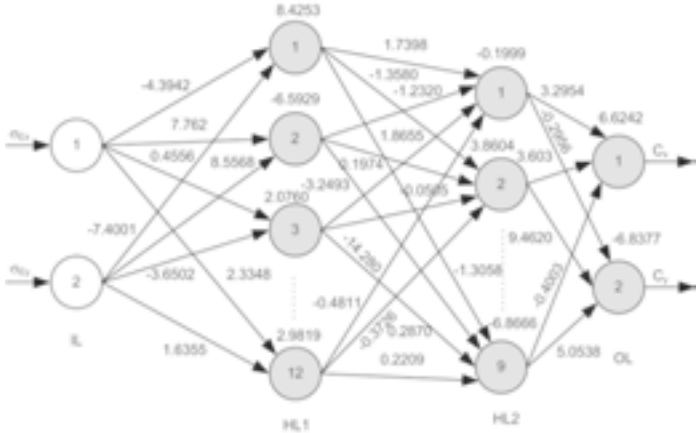
Written in code (C++, …)

Requires domain expertise

1. Decompose the problem
2. Design algorithms
3. Compose into a system

Measure performance

Software 2.0

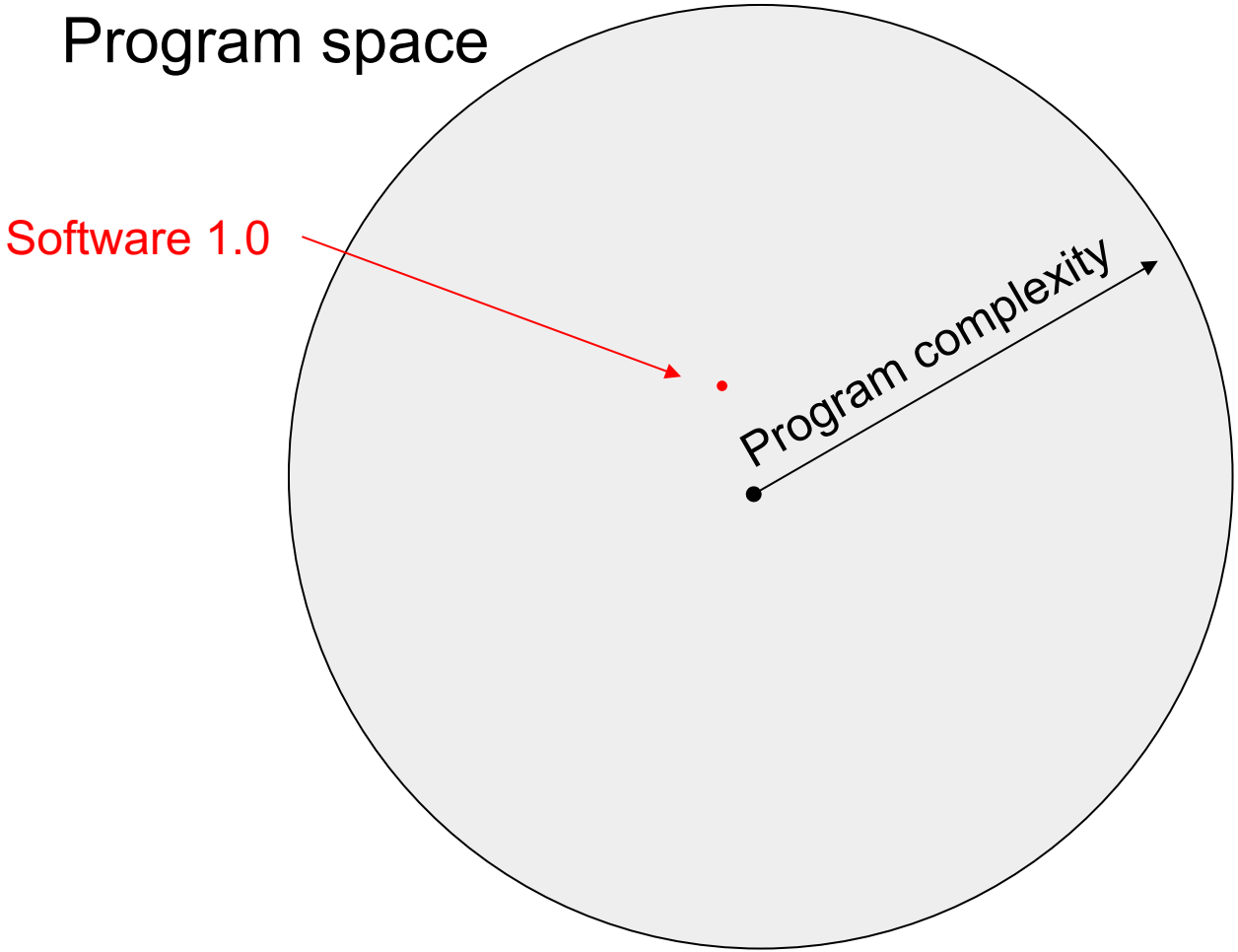


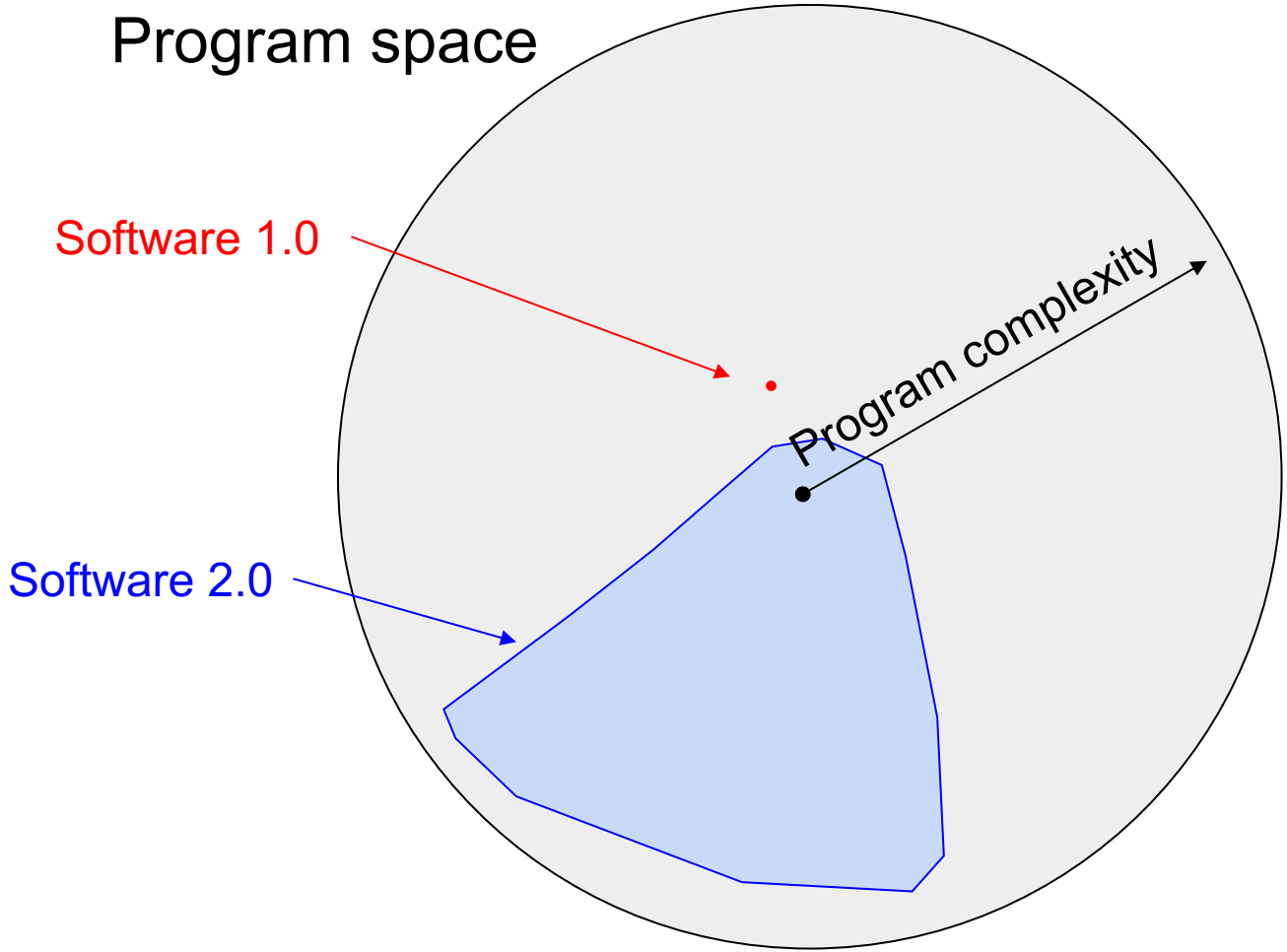
“Fill in the blanks programming”

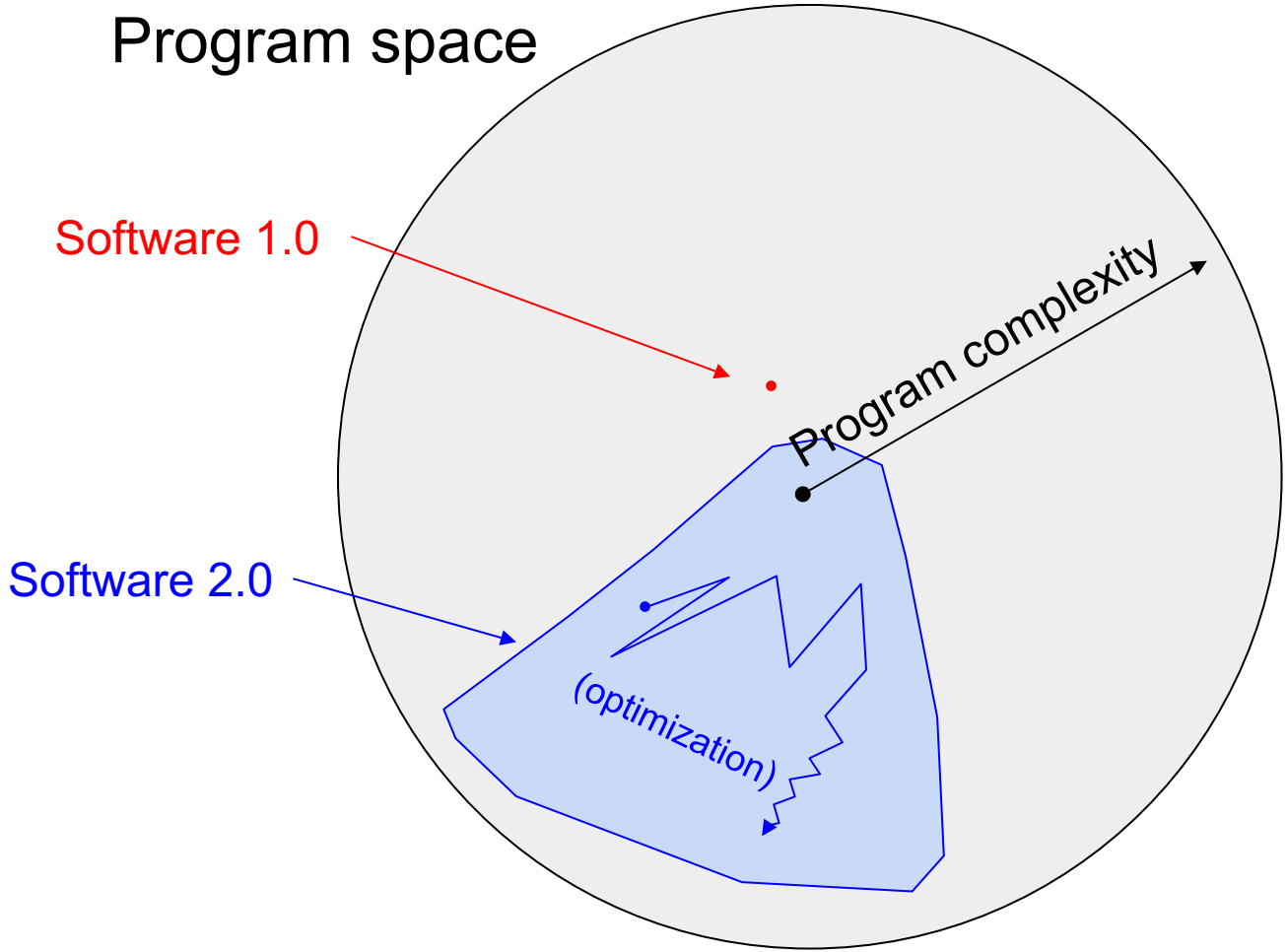
Requires much less domain expertise

1. Design a “code skeleton”
2. Measure performance

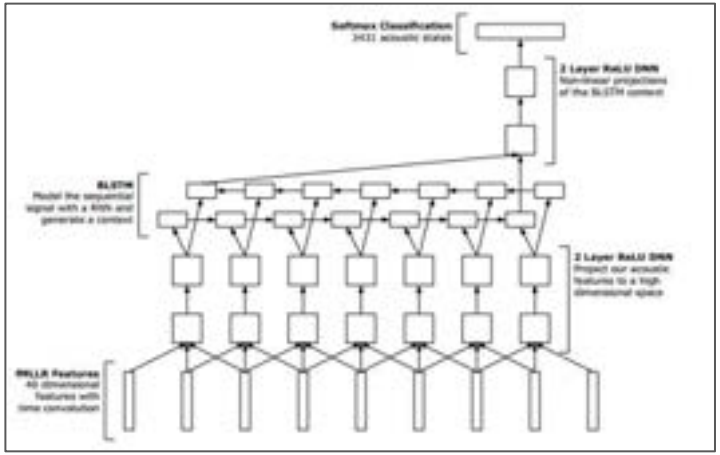
(automate)

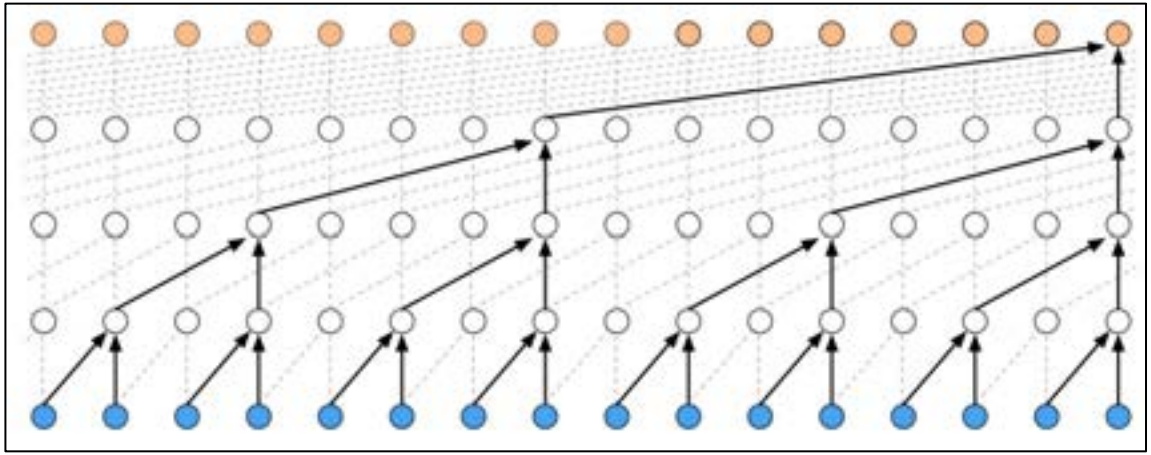

Program space
Software 1.0
Program complexity

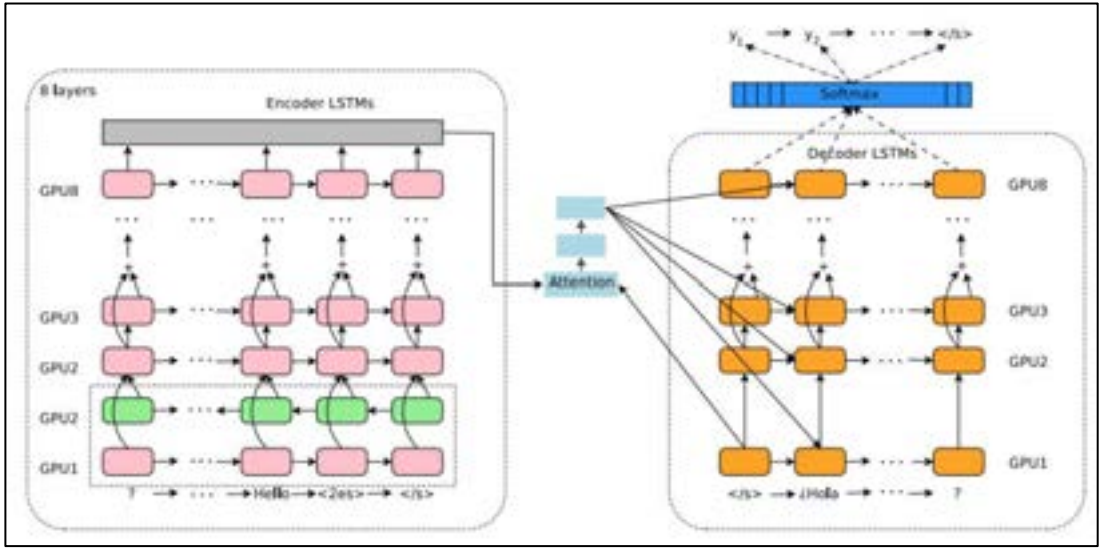

Program space
Software 1.0
Program complexity
Software 2.0


Program space
Software 1.0
Program complexity
Software 2.0
(optimization)


Andrej Karpathy
@karpathy
Gradient descent can write code better than you. I'm sorry.
3:56 PM - 4 Aug 2017
343 Retweets 1,161 Likes
72
343
1.2K
Add another Tweet
David Pfau @pfau · 5 Aug 2017
Replying to @karpathy
WHAT
GIF
18







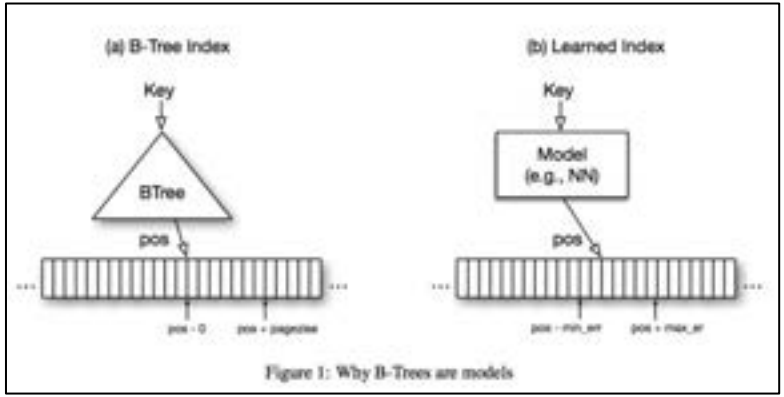


Figure 1: Why B-Trees are models

“One Model To Learn Them All”

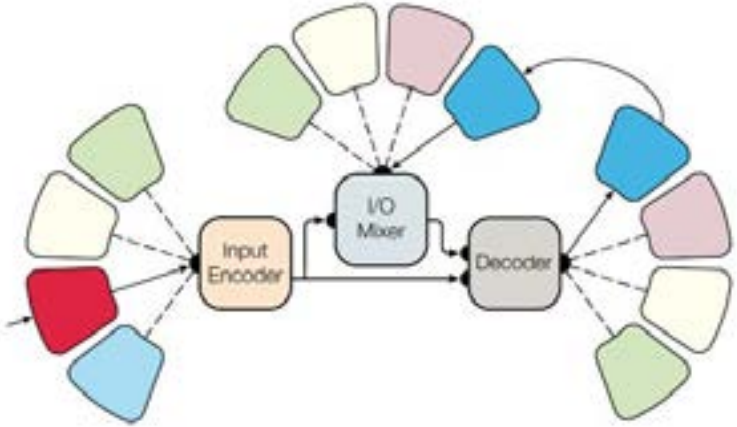


Figure 2: The MultiModel, with modality-nets, an encoder, and an autoregressive decoder.

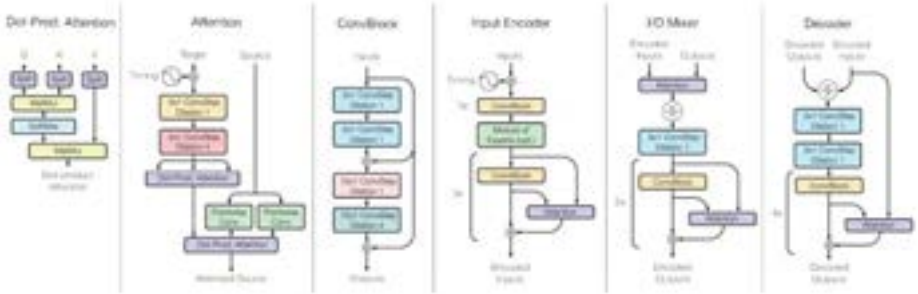


Figure 3: Architecture of the MultiModel; see text for details.

“single model is trained concurrently on ImageNet, multiple translation tasks, image captioning (COCO dataset), a speech recognition corpus, and an English parsing task”

(no need for datasets necessarily)

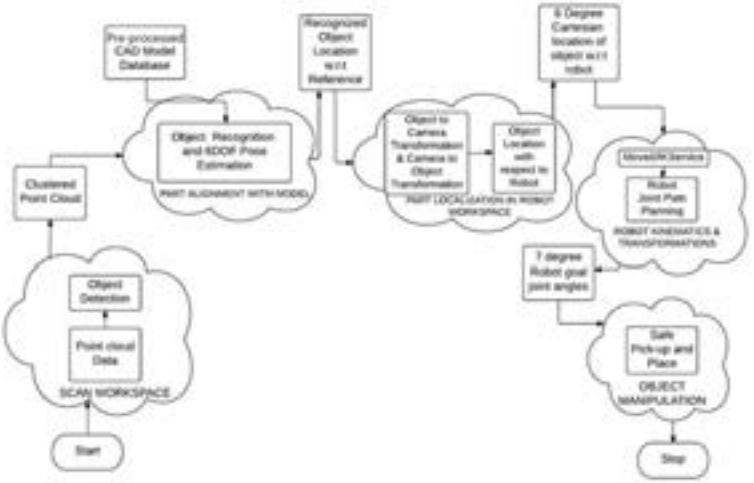
Other example members of the transition...

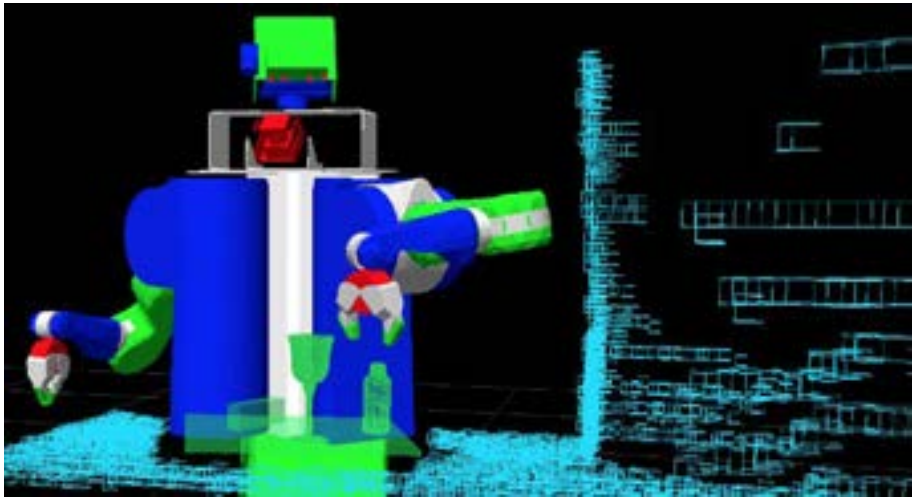
```
 1 # gcc -O3                    1 # STOKE
 2                              2
 3 movq rsi, r9                 3 shlq 32, rcx
 4 movl ecx, ecx                4 movl edx, edx
 5 shrq 32, rsi                 5 xorq rdx, rcx
 6 andl 0xffffffff, r9d         6 movq rcx, rax
 7 movq rcx, rax                7 mulq rsi
 8 movl edx, edx                8 addq r8, rdi
 9 imulq r9, rax                9 adcq 0, rdx
10 imulq rdx, r9               10 addq rdi, rax
11 imulq rsi, rdx              11 adcq 0, rdx
12 imulq rsi, rcx              12 movq rdx, r8
13 addq rdx, rax               13 movq rax, rdi
14 jae .L0
15 movabsq 0x100000000, rdx
16 addq rdx, rcx
17 .L0:
18 movq rax, rsi
19 movq rax, rdx
20 shrq 32, rsi
21 salq 32, rdx
22 addq rsi, rcx
23 addq r9, rdx
24 adcq 0, rcx
25 addq r8, rdx
26 adcq 0, rcx
27 addq rdi, rdx
28 adcq 0, rcx
29 movq rcx, r8
30 movq rdx, rdi
```

Figure 4.1: Montgomery multiplication kernel from the OpenSSL RSA library. Compilations shown for gcc -O3 (left) and a STOKE (right).

STOCHASTIC PROGRAM OPTIMIZATION FOR x86 64 BINARIES
PhD thesis of Eric Schkufza, 2015

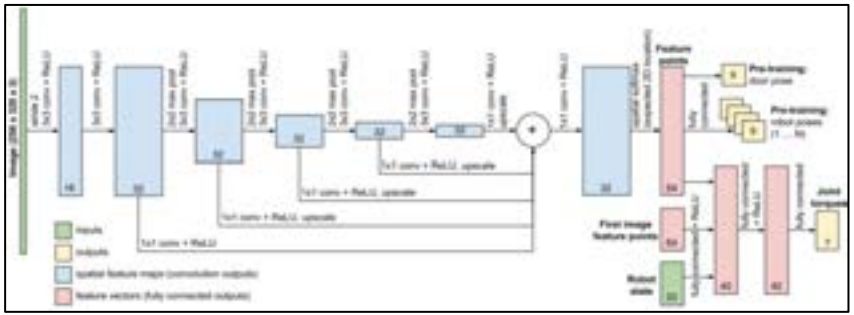
Robotics





2016+

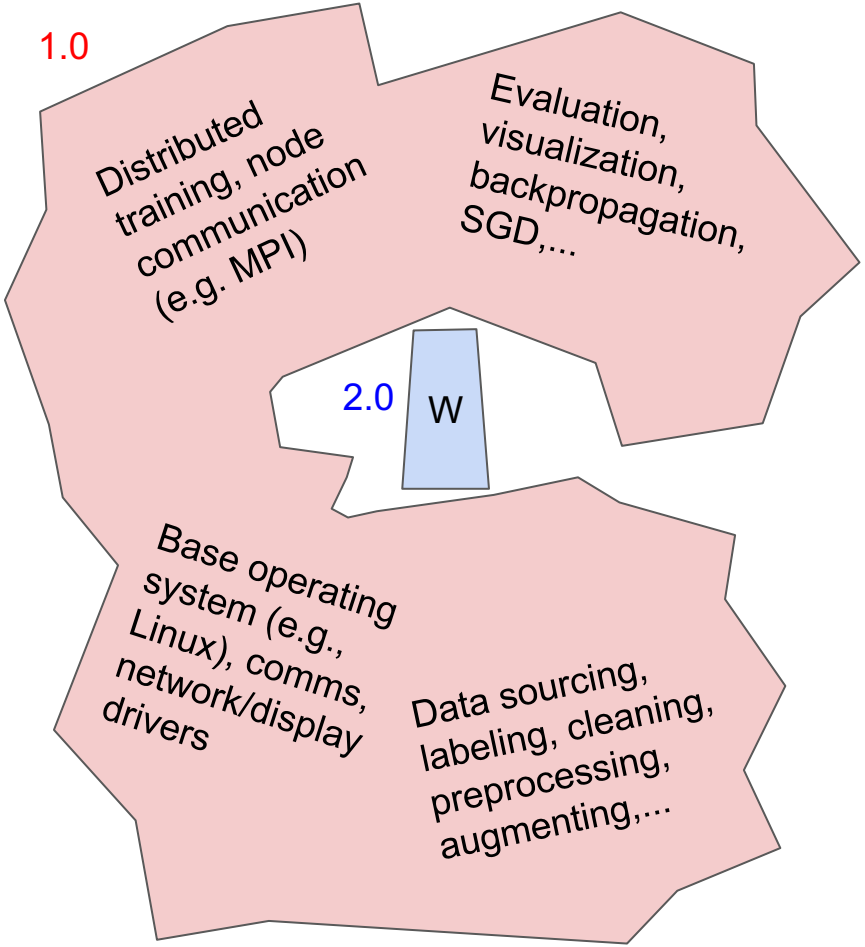
Google robot arm farm



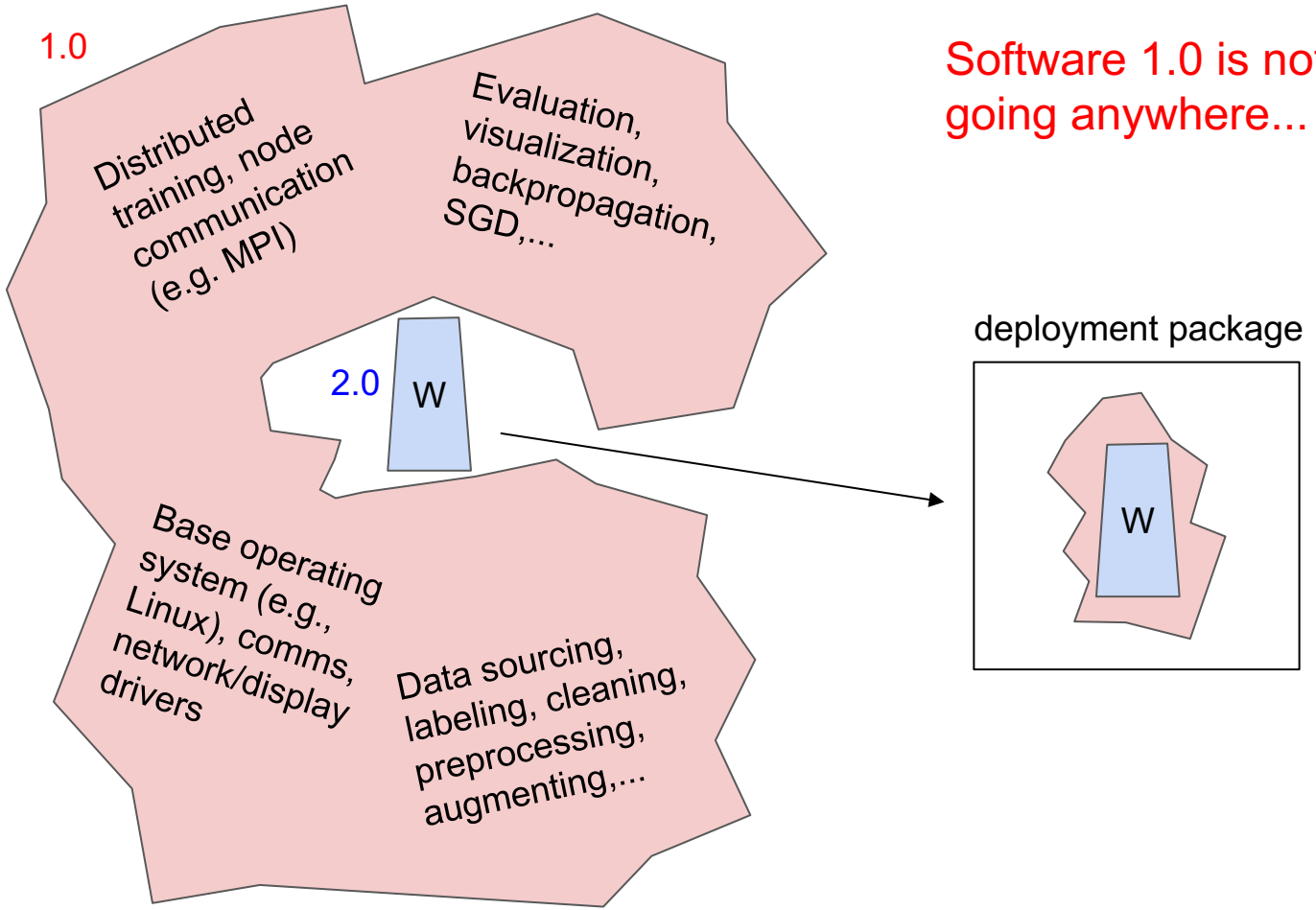
Neural Net: Image to torques

*ASTERISK :)

Fully Software 1.0

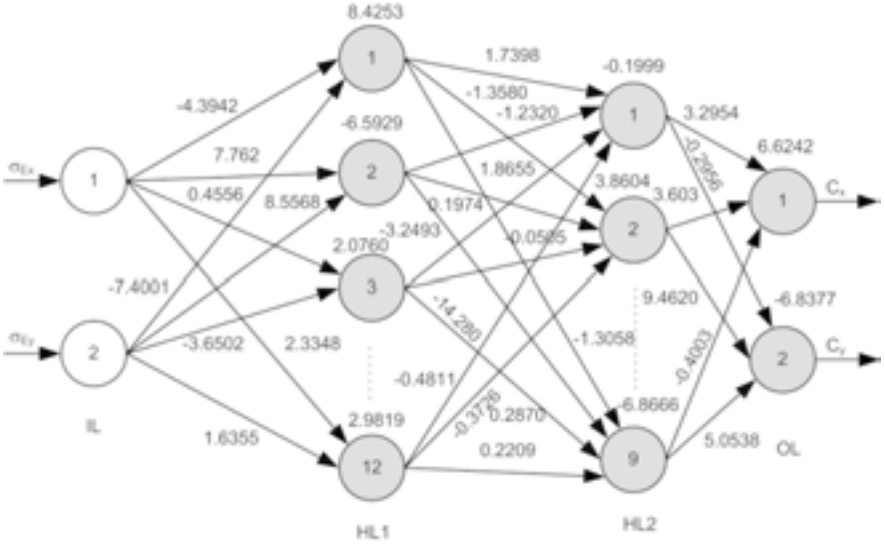


Software 1.0 is not going anywhere...

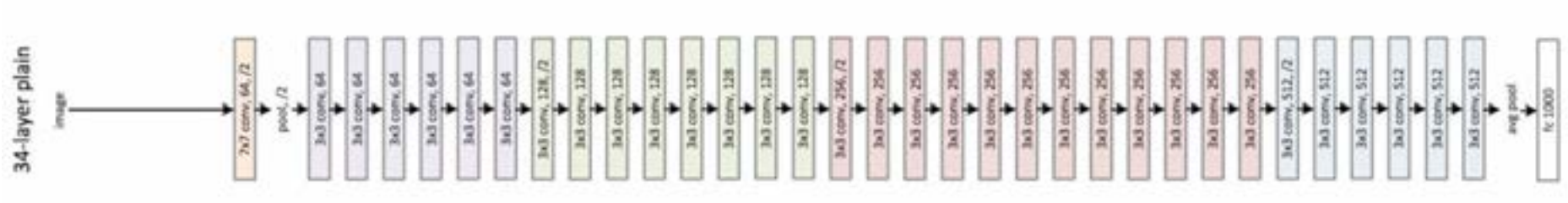


Software 1.0 is not going anywhere...

The benefits of Software 2.0



Computationally homogeneous

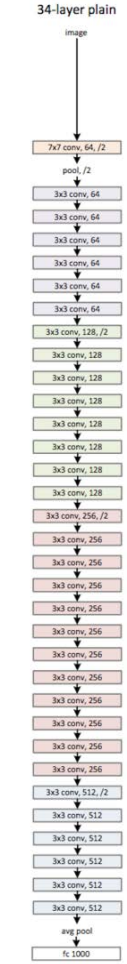


Hardware-friendly

Constant running time and memory use

```
int main( int argc, const char* argv[] ) {

    CvCapture* capture = cvCaptureFromCAM(CV_CAP_ANY);
    IplImage *img = 0;
    IplImage *yuv = 0;

    Configuration * config = new Configuration(true); // setting red and blue detection on.
    config->setDetect(config->RED,true);
    config->setDetect(config->BLUE,true);

    ColoredObjectExtractor * coe = new ColoredObjectExtractor(config); // creating detector

    ColoredObject *coloredObjects,*biggestObject;
    int color,size,i;

    while ( 1 ) {

        img = cvQueryFrame( capture ); // getting image from camera
        if (yuv==0) yuv=cvCreateImage(cvSize(img->width,img->height), IPL_DEPTH_8U, 3);
        cvCvtColor(img,yuv,CV_RGB2YCrCb); // converting to YUV color space

        coe->setImage(yuv);
        coe->detect(); // running detection

        // printing information about all colored object detected
        cout << "\n\n-----------------------------------------------\n";
        for(color=0;color<config->NBCOLORS;color++){
            if (config->isDetecting[color]){
                size = coe->getSize(color);
                cout << "detecting " << size << " " << config->printColors[color] << " objects\n";
                if (size>0){
                    biggestObject = coe->getBiggest(color);
                    coloredObjects = coe->getColoredObjects(color);
                    for (i=0;i<size;i++){
                        if (&coloredObjects[i] == biggestObject) cout<<"(biggest) ";
                        cout << "area\t"<<coloredObjects[i].area<<"\tx\t"<<coloredObjects[i].x<<"\ty\t"<<coloredObjects[i].y<<"\n";
                    }
                }
            }
        }
        cout << "-----------------------------------------------\n";

        if ( (cvWaitKey(10) & 255) == 27 ) break;
    }

    cvReleaseImage(&yuv);
    cvReleaseImage(&img);
    delete coe;
    delete config;
}
```

vs.

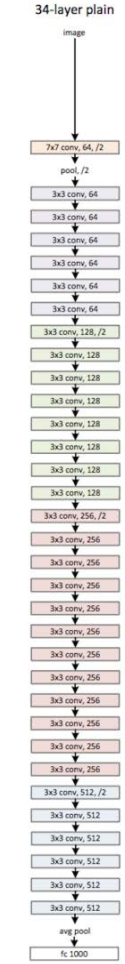


Agile

“I’d like code with the same functionality but I’d like it to run faster, even if it means slightly worse results”

```
int main( int argc, const char* argv[] ) {

    CvCapture* capture = cvCaptureFromCAM(CV_CAP_ANY);
    IplImage *img = 0;
    IplImage *yuv = 0;

    Configuration * config = new Configuration(true); // setting red and blue detection on.
    config->setDetect(config->RED,true);
    config->setDetect(config->BLUE,true);

    ColoredObjectExtractor * coe = new ColoredObjectExtractor(config); // creating detector

    ColoredObject *coloredObjects,*biggestObject;
    int color,size,i;

    while ( 1 ) {

        img = cvQueryFrame( capture ); // getting image from camera
        if (yuv==0) yuv=cvCreateImage(cvSize(img->width,img->height), IPL_DEPTH_8U, 3);
        cvCvtColor(img,yuv,CV_RGB2YCrCb); // converting to YUV color space

        coe->setImage(yuv);
        coe->detect(); // running detection

        // printing information about all colored object detected
        cout << "\n\n-----------------------------------------------\n";
        for(color=0;color<config->NBCOLORS;color++){
            if (config->isDetecting[color]){
                size = coe->getSize(color);
                cout << "detecting " << size << " " << config->printColors[color] << " objects\n";
                if (size>0){
                    biggestObject = coe->getBiggest(color);
                    coloredObjects = coe->getColoredObjects(color);
                    for (i=0;i<size;i++){
                        if (&coloredObjects[i] == biggestObject) cout<<"(biggest) ";
                        cout << "area\t"<<coloredObjects[i].area<<"\tx\t"<<coloredObjects[i].x<<"\ty\t"<<coloredObjects[i].y<<"\n";
                    }
                }
            }
        }
        cout << "-----------------------------------------------\n";

        if ( (cvWaitKey(10) & 255) == 27 ) break;
    }

    cvReleaseImage(&yuv);
    cvReleaseImage(&img);
    delete coe;
    delete config;
}
```

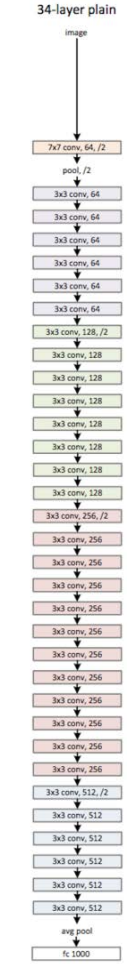
vs.



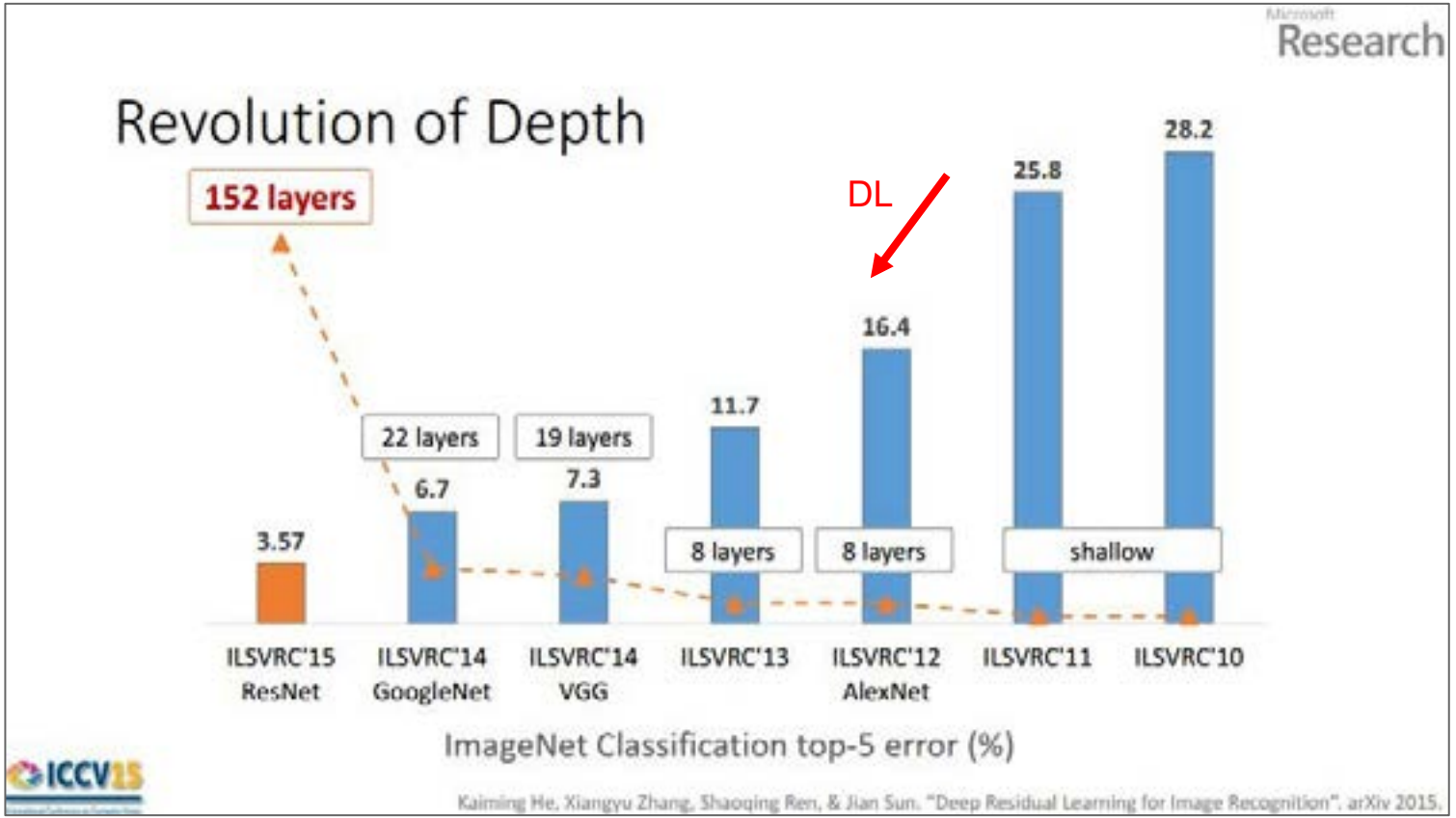
Finetuning

```
int main( int argc, const char* argv[] ) {

    CvCapture* capture = cvCaptureFromCAM(CV_CAP_ANY);
    IplImage *img = 0;
    IplImage *yuv = 0;

    Configuration * config = new Configuration(true); // setting red and blue detection on.
    config->setDetect(config->RED,true);
    config->setDetect(config->BLUE,true);

    ColoredObjectExtractor * coe = new ColoredObjectExtractor(config); // creating detector

    ColoredObject *coloredObjects,*biggestObject;
    int color,size,i;

    while ( 1 ) {

        img = cvQueryFrame( capture ); // getting image from camera
        if (yuv==0) yuv=cvCreateImage(cvSize(img->width,img->height), IPL_DEPTH_8U, 3);
        cvCvtColor(img,yuv,CV_RGB2YCrCb); // converting to YUV color space

        coe->setImage(yuv);
        coe->detect(); // running detection

        // printing information about all colored object detected
        cout << "\n\n-----------------------------------------------\n";
        for(color=0;color<config->NBCOLORS;color++){
            if (config->isDetecting[color]){
                size = coe->getSize(color);
                cout << "detecting " << size << " " << config->printColors[color] << " objects\n";
                if (size>0){
                    biggestObject = coe->getBiggest(color);
                    coloredObjects = coe->getColoredObjects(color);
                    for (i=0;i<size;i++){
                        if (&coloredObjects[i] == biggestObject) cout<<"(biggest) ";
                        cout << "area\t"<<coloredObjects[i].area<<"\tx\t"<<coloredObjects[i].x<<"\ty\t"<<coloredObjects[i].y<<"\n";
                    }
                }
            }
        }
        cout << "-----------------------------------------------\n";

        if ( (cvWaitKey(10) & 255) == 27 ) break;
    }

    cvReleaseImage(&yuv);
    cvReleaseImage(&img);
    delete coe;
    delete config;
}
```

vs.



It works very well.

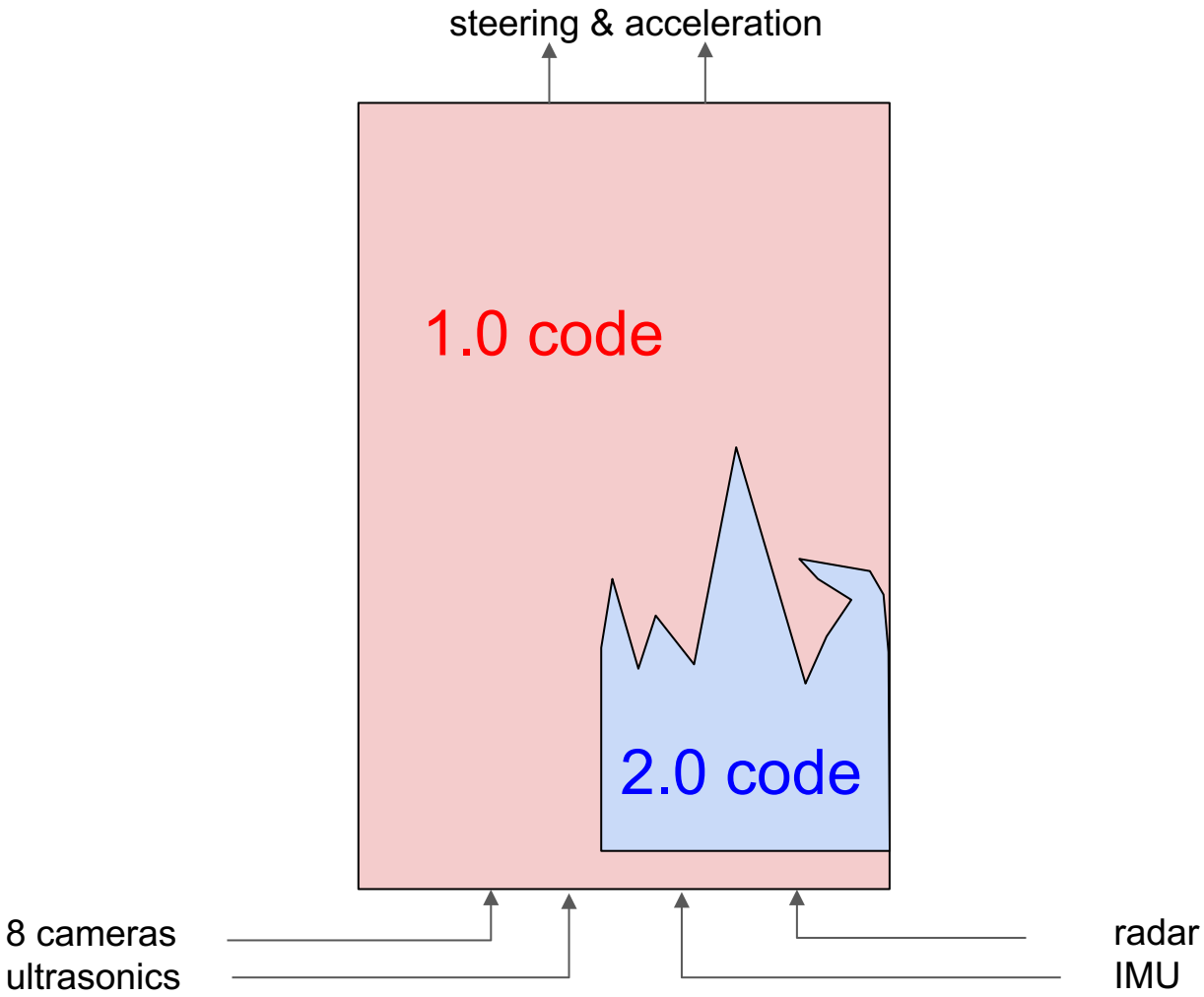


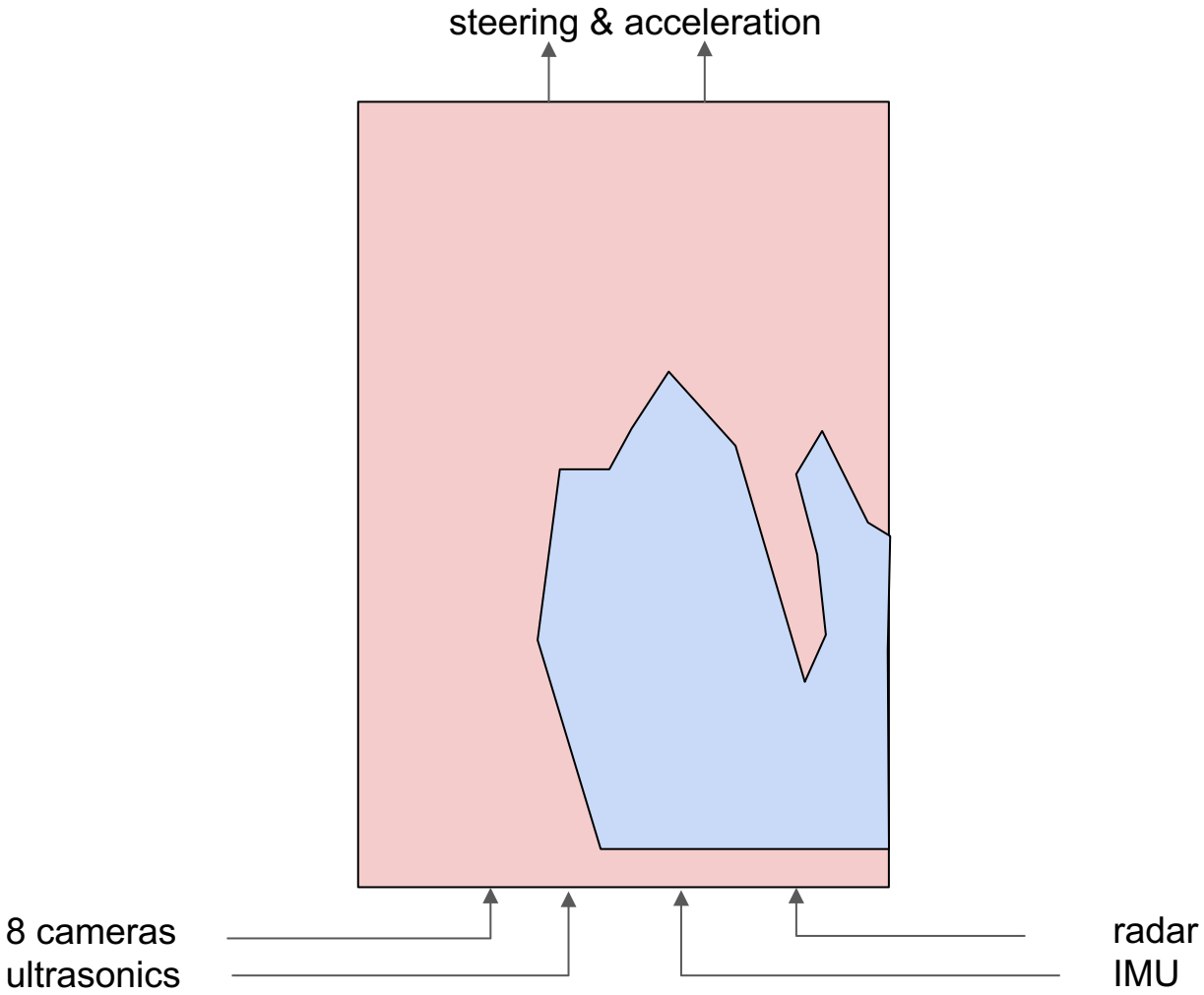
(slide from Kaiming He's recent presentation)

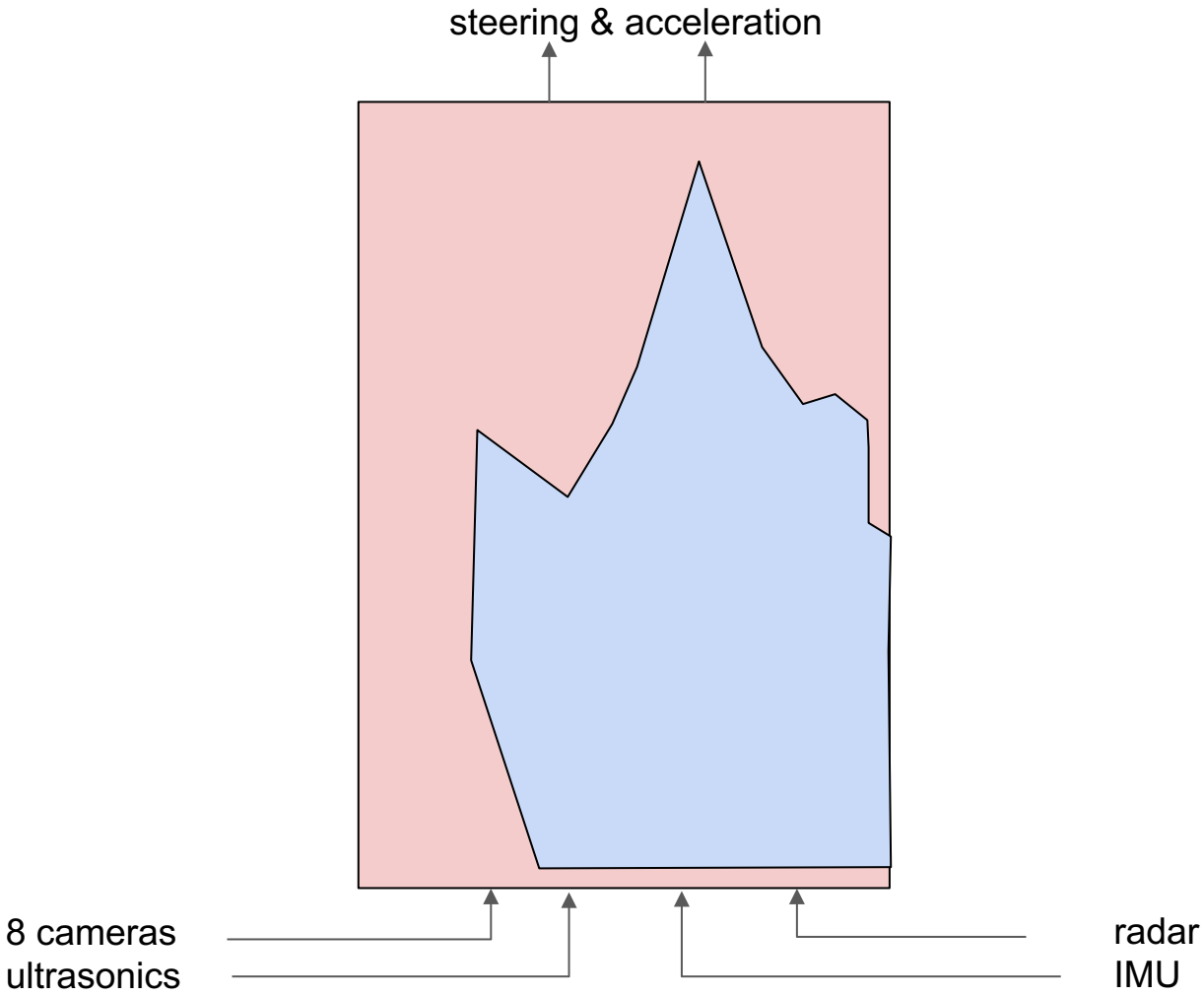
TESLA

Largest deployment of robots in the world (0.25M)

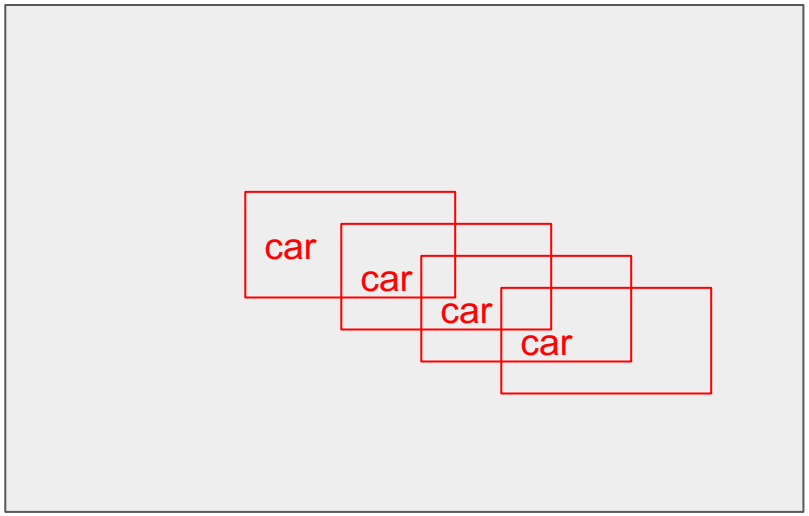
Make them autonomous.


steering & acceleration
1.0 code
2.0 code
8 cameras
ultrasonics
radar
IMU


steering & acceleration
8 cameras
ultrasonics
radar
IMU


steering & acceleration
8 cameras
ultrasonics
radar
IMU

Example: parked cars

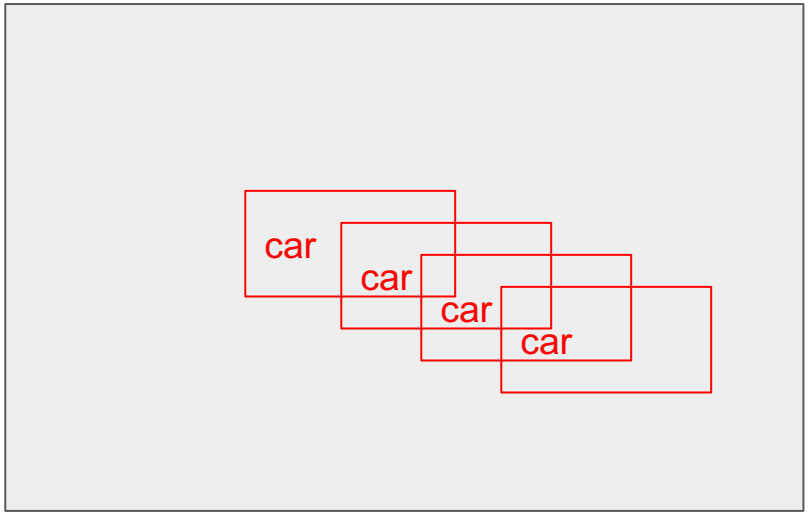


Parked if:

Tracked bounding box does not move more than 20 pixels over last 3 seconds AND is in a neighboring lane, AND...

(brittle rules on highly abstracted representation)

Example: parked cars



Parked if:

Tracked bounding box does not move more than 20 pixels over last 3 seconds AND is in a neighboring lane, AND...

(brittle rules on highly abstracted representation)

Parked if:

Neural network says so,
based on a lot of labeled data.

Programming with the 2.0 stack

If optimization is doing most of the coding, what are the humans doing?

- Design and develop cool algorithms
- Analyze running times

Algorithm 1 GreedyPlus Algorithm

```
1: alignment ← null
2: EAWeight ← user input
3: while ∃u, v s.t. [(u ∈ D1 and v ∈ D2) or (u ∈ B1 and v ∈ B2)] and u, v ∉ alignment do
4:   bestPair ← null
5:   for all u, v do
6:     if Score(u, v) + EA(u, v) > Score(bestPair) + EA(bestPair) then
7:       bestPair ← (u, v)
8:     end if
9:     alignment ← alignment + bestPair
10:  end for
11: end while
12:
13: EA(u,v) {
14: EAScore ← 0
15: for all ⟨w, x⟩ ∈ alignment do
16:   if (u, w) ∈ E1 and (v, x) ∈ E2 then
17:     EAScore ← EAScore + EAWeight
18:   end if
19: end for
20: return EAScore
21: }
```

If optimization is doing most of the coding, what are the humans doing?

- Design and develop cool algorithms
- Analyze running times

Algorithm 1 GreedyPlus Algorithm

```
1: alignment ← null
2: EAWeight ← user input
3: while ∃u, v s.t. [(u ∈ D1 and v ∈ D2) or (u ∈ B1 and v ∈ B2)] and u, v ∉ alignment do
4:   bestPair ← null
5:   for all u, v do
6:     if Score(u, v) + EA(u, v) > Score(bestPair) + EA(bestPair) then
7:       bestPair ← (u, v)
8:     end if
9:     alignment ← alignment + bestPair
10:  end for
11: end while
12:
13: EA(u,v) {
14: EAScore ← 0
15: for all ⟨w, x⟩ ∈ alignment do
16:   if (u, w) ∈ E1 and (v, x) ∈ E2 then
17:     EAScore ← EAScore + EAWeight
18:   end if
19: end for
20: return EAScore
21: }
```

1. Label

If optimization is doing most of the coding, what are the humans doing?

- Design and develop cool algorithms
- Analyze running times

Algorithm 1 GreedyPlus Algorithm

```
1: alignment ← null
2: EAWeight ← user input
3: while ∃u, v s.t. [(u ∈ D1 and v ∈ D2) or (u ∈ B1 and v ∈ B2)] and u, v ∉ alignment do
4:   bestPair ← null
5:   for all u, v do
6:     if Score(u, v) + EA(u, v) > Score(bestPair) + EA(bestPair) then
7:       bestPair ← (u, v)
8:     end if
9:     alignment ← alignment + bestPair
10:  end for
11: end while
12:
13: EA(u,v) {
14: EAScore ← 0
15: for all ⟨w, x⟩ ∈ alignment do
16:   if (u, w) ∈ E1 and (v, x) ∈ E2 then
17:     EAScore ← EAScore + EAWeight
18:   end if
19: end for
20: return EAScore
21: }
```

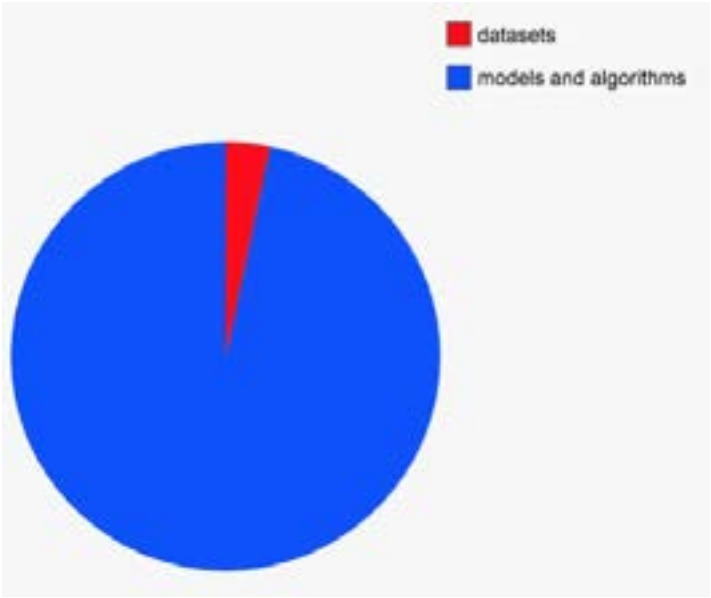
1. Label

2. Maintain surrounding “dataset infrastructure”

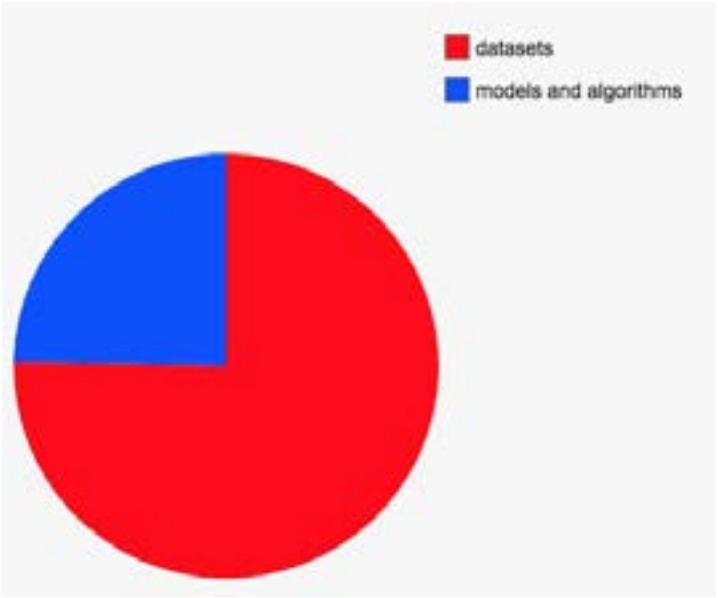
- Flag labeler disagreements, keep stats on labelers, “escalation” features
- Identify “interesting” data to label
- Clean existing data
- Visualize datasets

Amount of lost sleep over...

PhD



Tesla



Lesson learned the hard way #1:

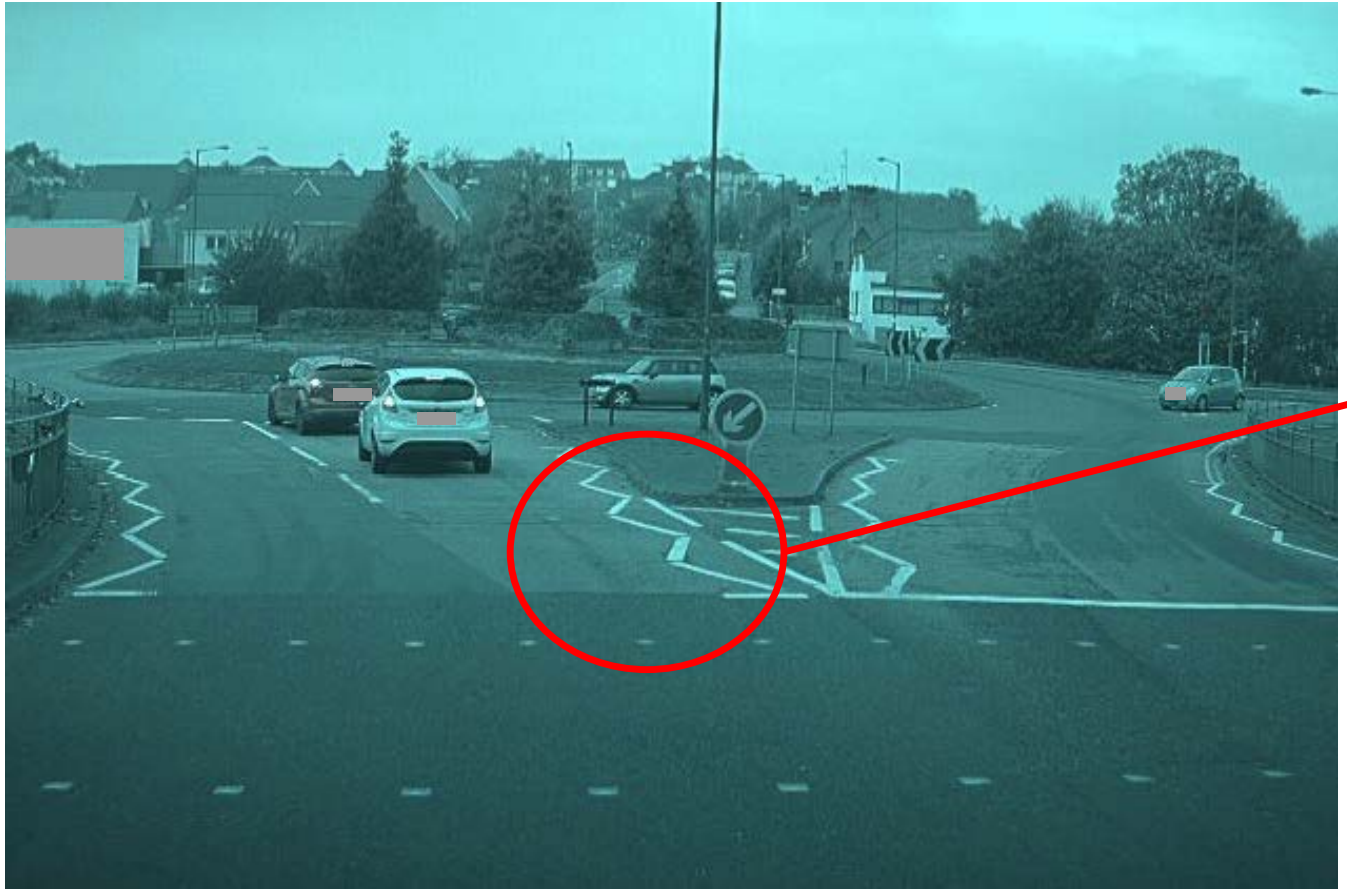
Data labeling is highly non-trivial

“Label lane lines”

“Label lane lines”

“Label lane lines”

(Philosophical conundrums)



How do you annotate lane lines when they do this?

“Label lane lines”

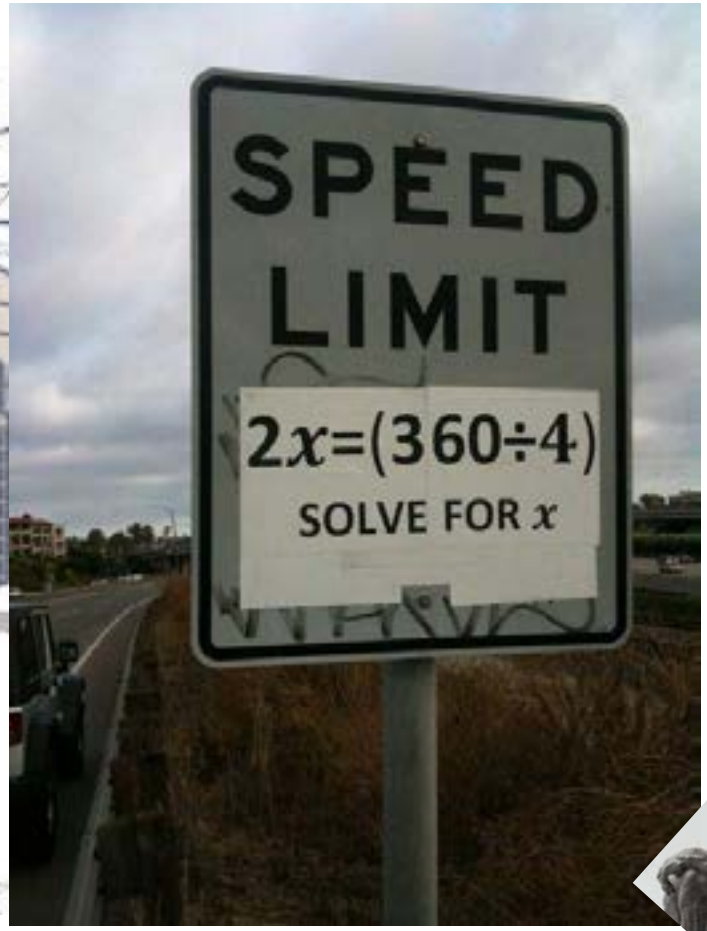
“Label lane lines”

(Philosophical conundrums)

“Is that one car, four cars, two cars?”

(Philosophical conundrums)

(Philosophical conundrums)



Lesson learned the hard way #2:

Chasing Label/Data Imbalances is non-trivial

car

90% of all vehicles

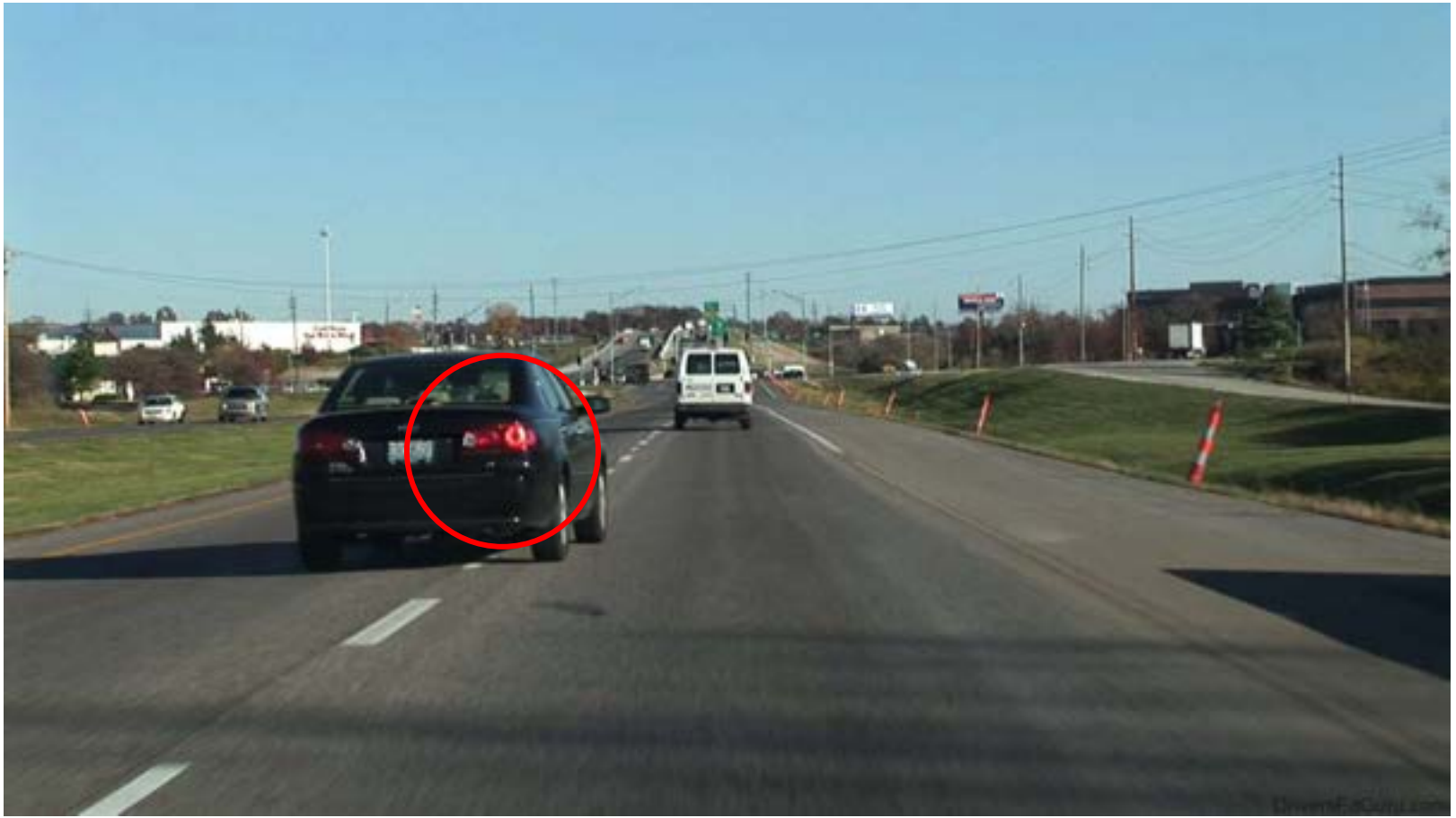
trolley

1e-3% of all vehicles

10% of all signs

1e-4% of all signs

Right blinker on



Orange traffic light

90%+ of data

1e-3% of data

1e-3% of data

Lesson learned the hard way #3:

Labeling is an iterative process

Example: Autowiper

1. Collect labels
2. Train a model
3. Deploy the model

Example: Autowiper

1. Collect labels
2. Train a model
3. Deploy the model

Will it Wipe? Tesla Autowiper Extreme Test!

26,553 views

497 175 SHARE

Will it Wipe? Tesla Autowiper Extreme Test!

26,553 views

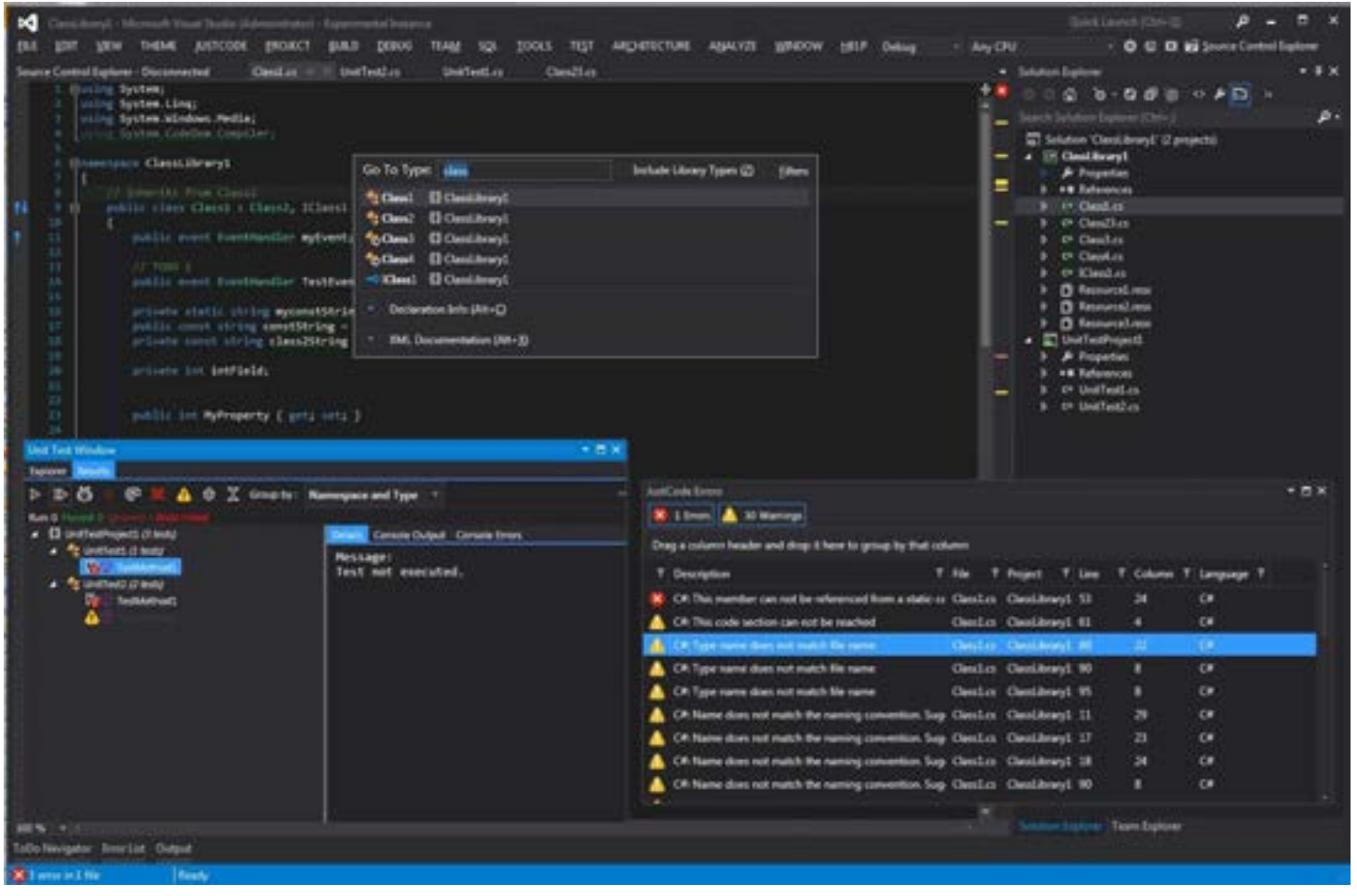
497 179 SHARE

Lesson learned the hard way overall:

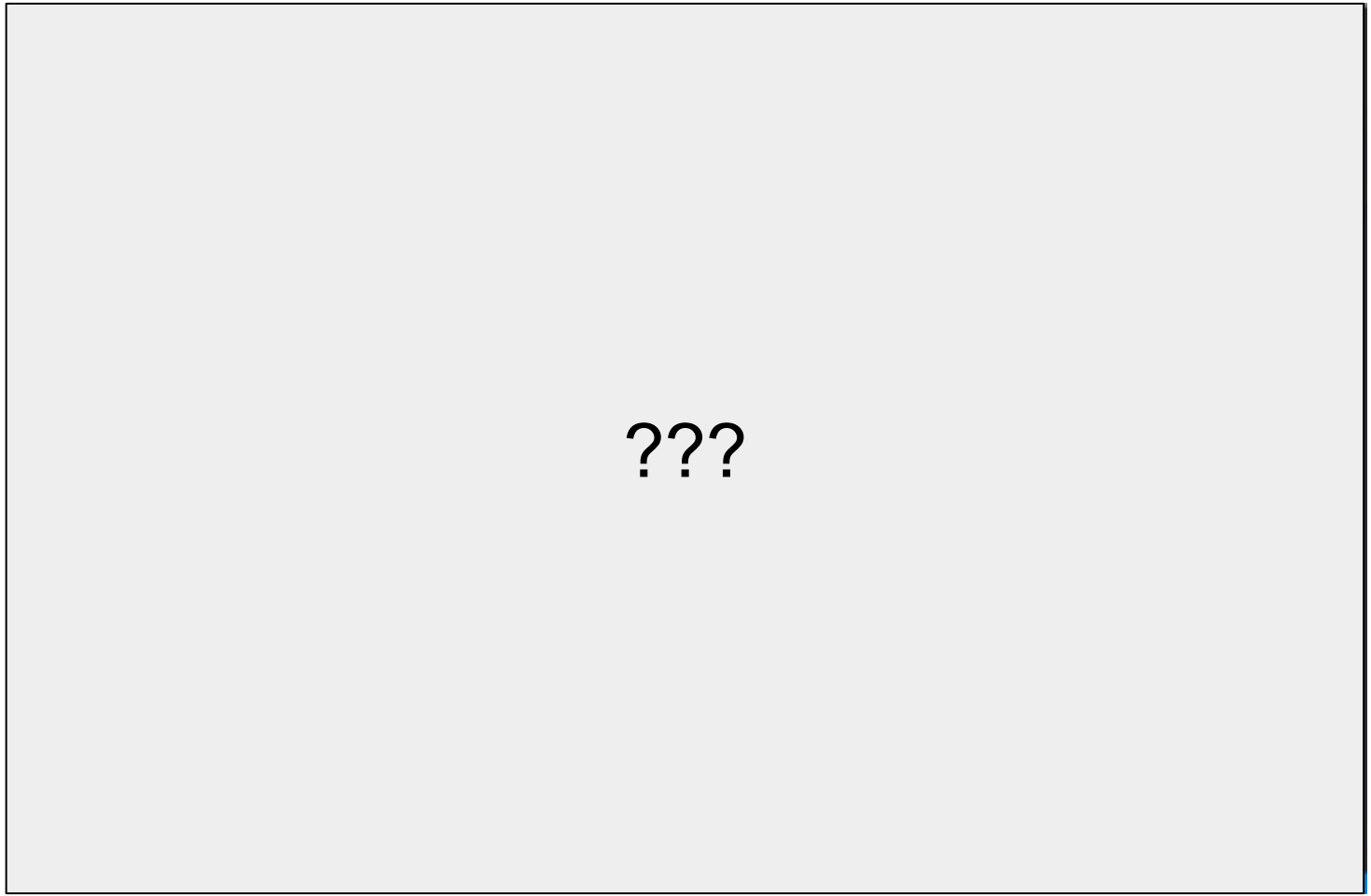
The toolchain for the 2.0 stack does not yet exist.

(and few people realize it’s a thing)

1.0 IDEs



2.0 IDEs



2.0 IDEs

- Show a full inventory/stats of the current dataset
- Create / edit annotation layers for any datapoint
- Flag, escalate & resolve discrepancies in multiple labels
- Flag & escalate datapoints that are likely to be mislabeled
- Display predictions on an arbitrary set of test datapoints
- Autosuggest datapoints that should be labeled
- ...


SPACEX
TESLA

The sky's the limit

Thank you!