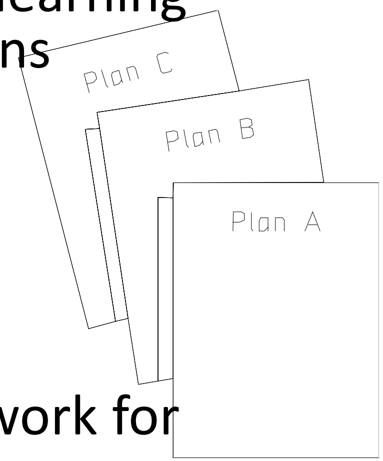
Mobile and Embedded Machine Learning

If you have the power to make someone happy, do it.
The world needs more of that.

Overview

- Objective
  - To understand the opportunities to apply machine learning and deep learning techniques for mobile applications

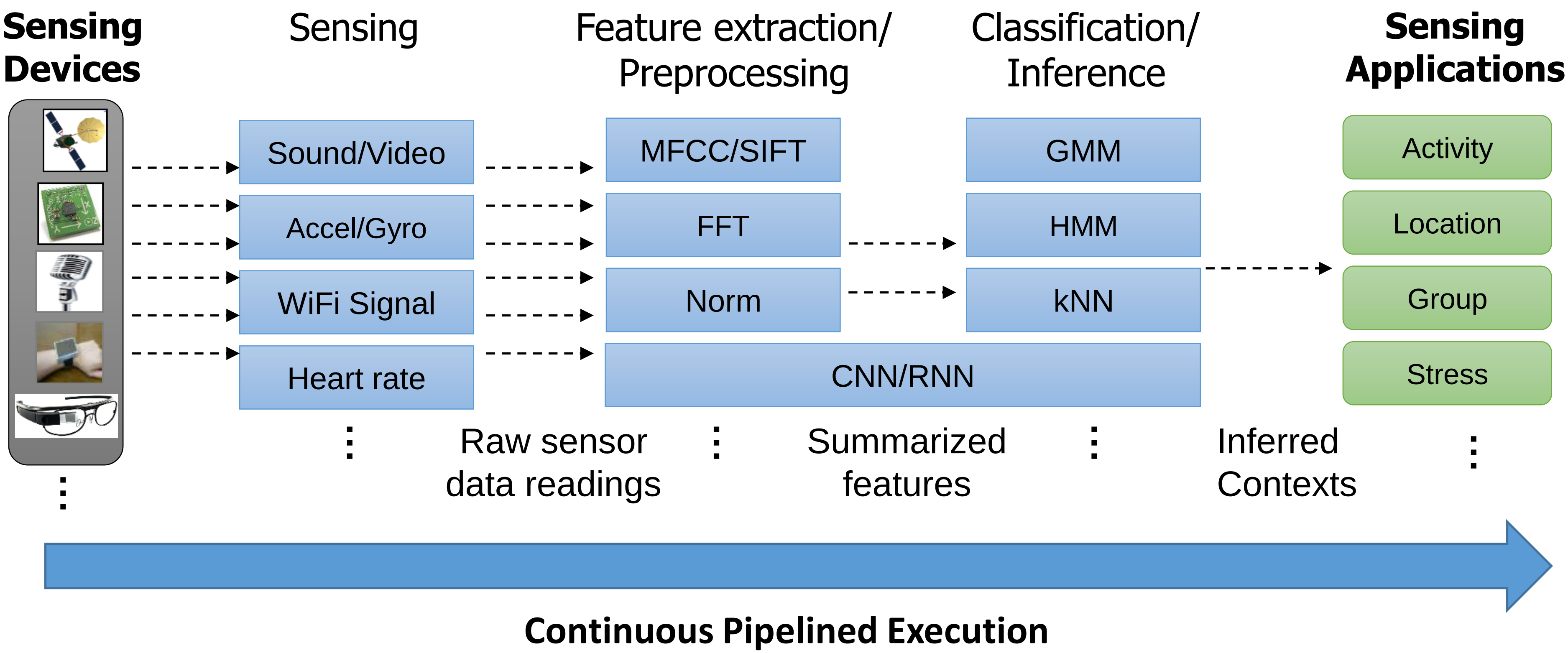


- Content
  - Machine learning for mobile and IoT applications
  - Intro to convolutional neural network
  - DeepMon: Mobile gpu-based deep learning framework for continuous vision applications

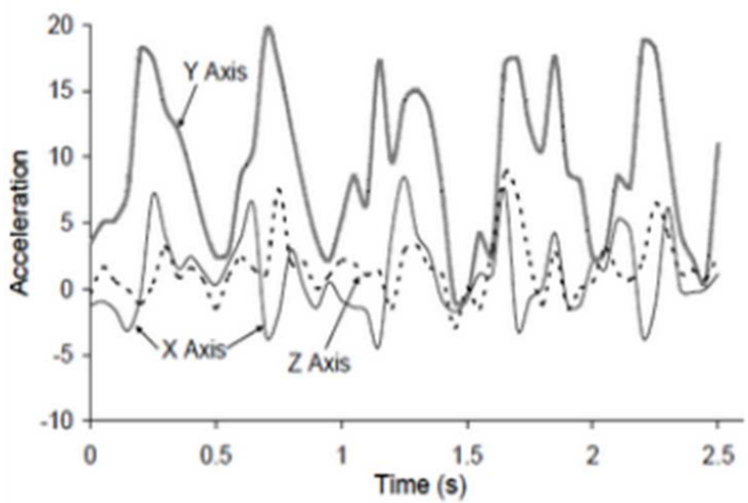
- After this module, you should be able to
  - Understand the basics of machine learning and deep learning techniques for mobile and IoT applications

Mobile and IoT Sensing

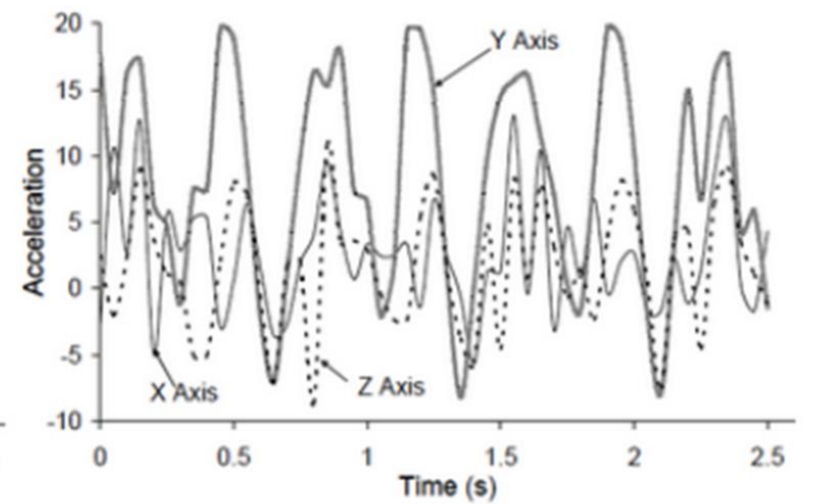
Continuous sensing and analytics of user activities, location, emotions, and surroundings with mobile/IoT/wearable devices



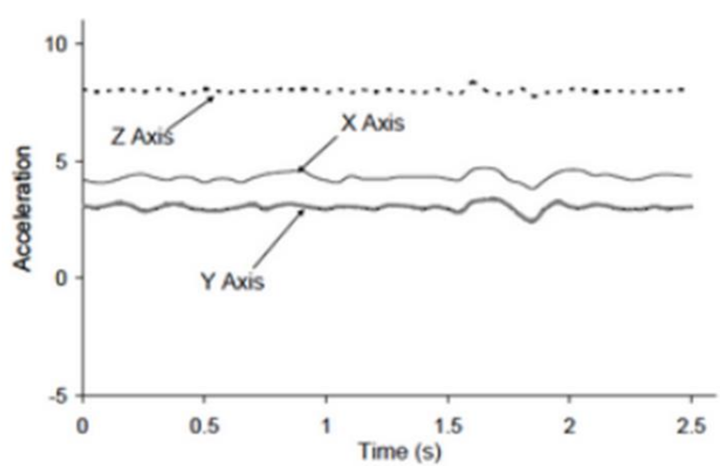
Revisit Activity Recognition



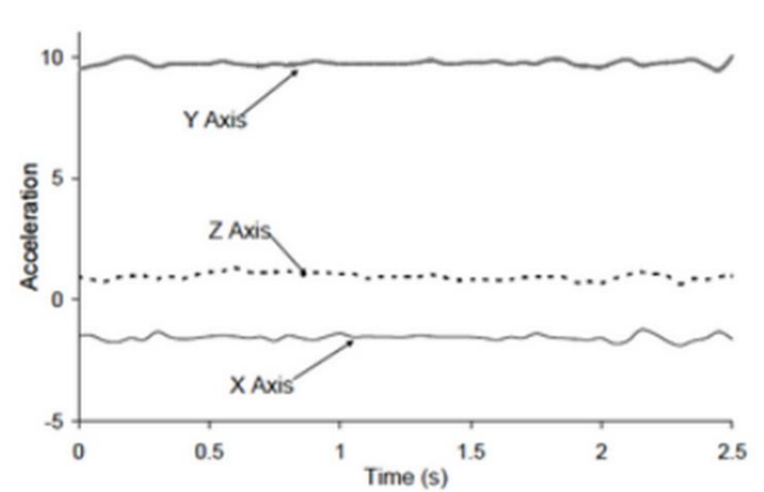
Waking



Jogging



Sitting



Standing

Simple Heuristic

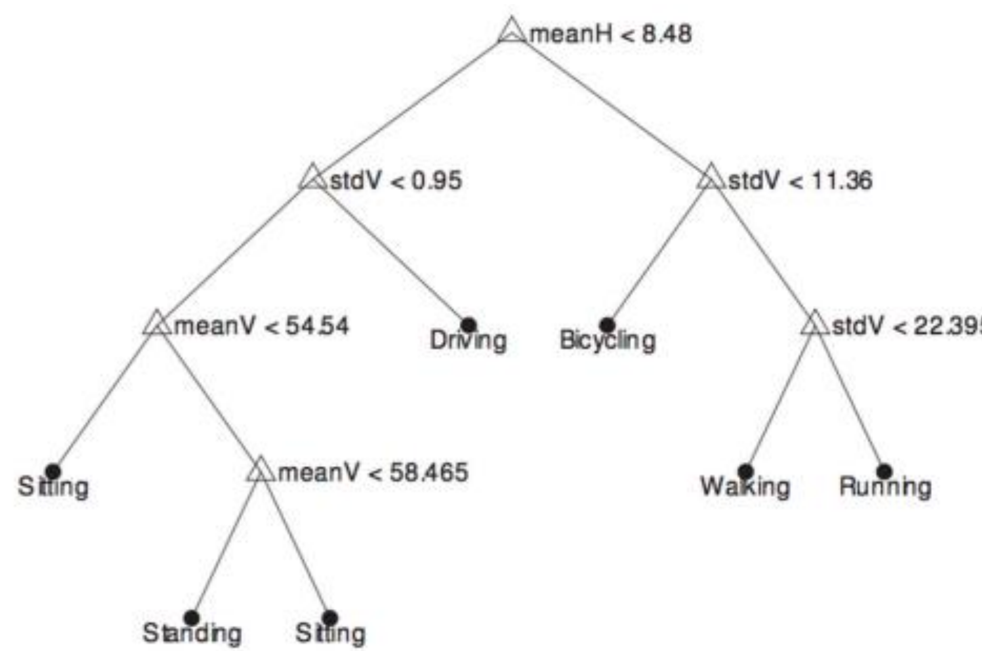
- If STDEV(y-axis samples) < $C_{Threshold1}$
  - If AVG(y-axis samples) > $C_{Threshold2}$
    - output standing
  - Else
    - output sitting
- Else
  - If FFT(y-axis samples) < $C_{Threshold3}$
    - output walking
  - Else
    - output jogging

Are We Good?

- How do we determine good features and good thresholds?
  - How do we know STDEV is better than MAX?
  - How do we know AVG is better than Median?
  - How do we know the right values for $C_{threshold}$?
- What if a user puts her phone in her bag, not in her front pocket?
  - The Y-axis of the phone is not anymore the major axis of movement.
- How do we solve these problems? A better heuristic?

Decision Tree

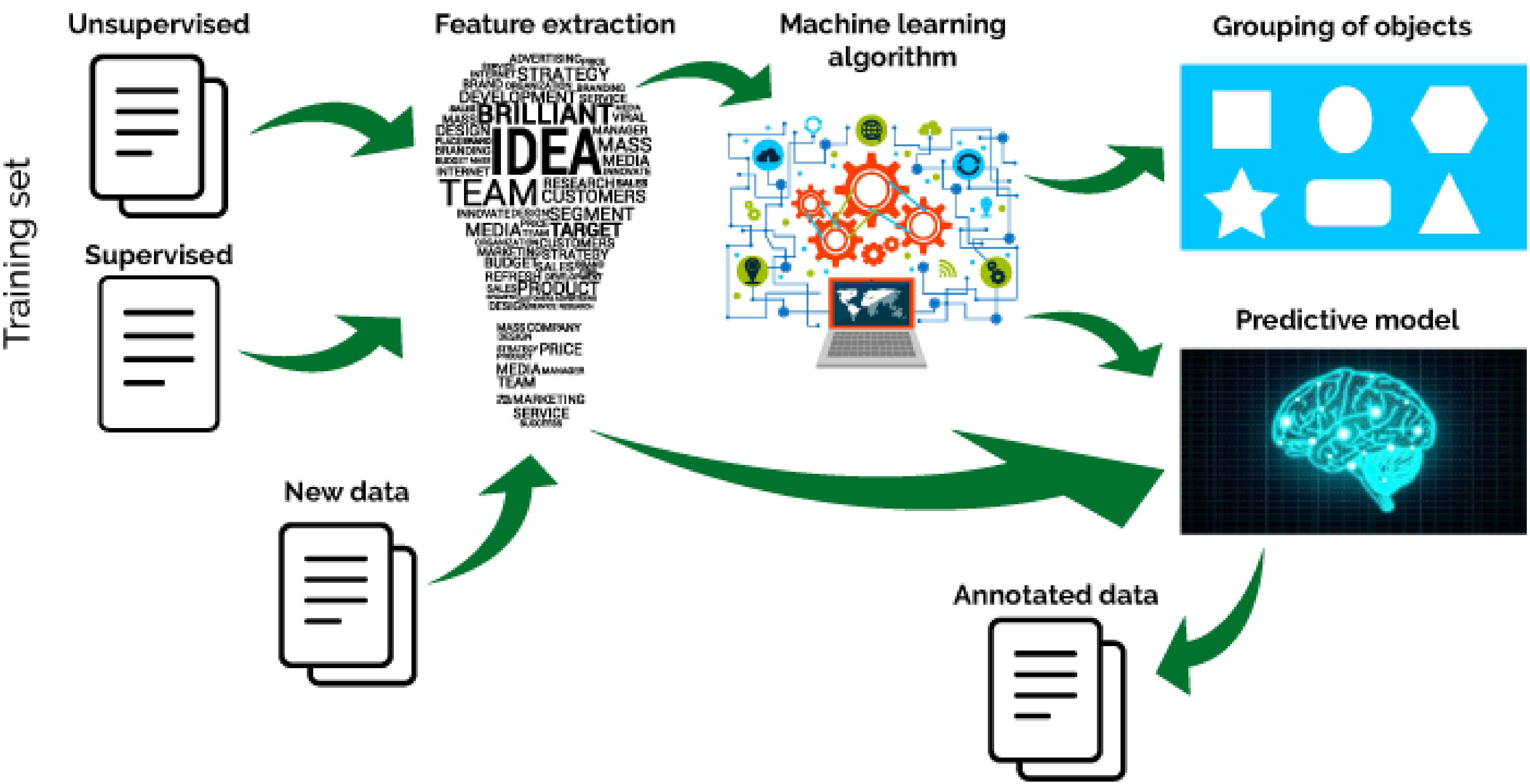
- A simple but effective ML classifier.
- This tree can be built by the C4.5 algorithm.
- Given sufficient training data, the algorithm can automatically determine the important features and their thresholds.



Other ML Techniques

- Naïve Bayes classifier
- Decision tree
- Random forest
- Support vector machine
- kNN algorithm
- Linear regression
- …

ML Techniques Flow

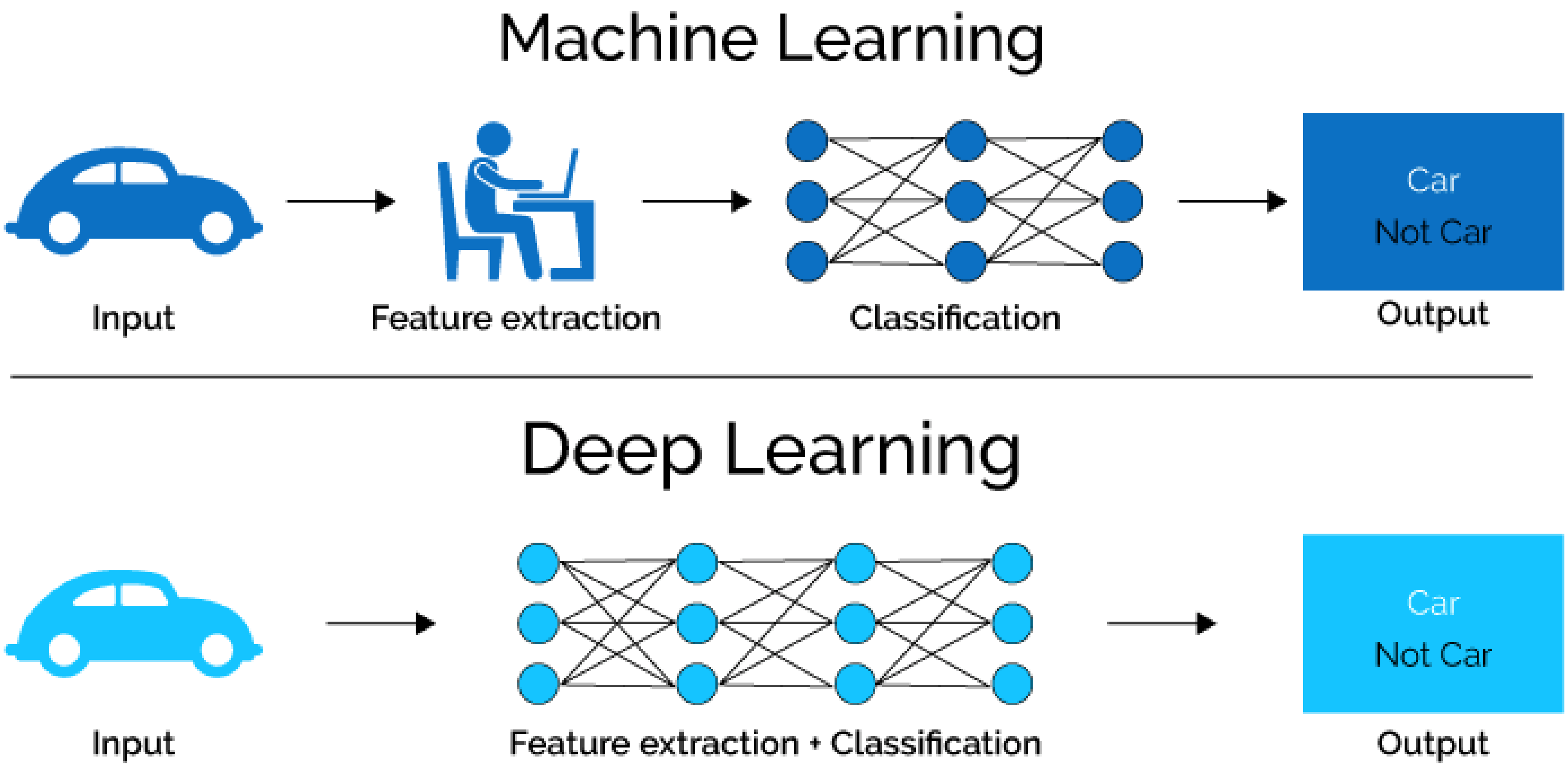


ML Techniques: Limitations

- Linear regression?
  - Why is it linear?
- Bayesian?
  - What is the prior?
- SVM?
  - What are the features?
- Decision tree?
  - What are the nodes/variables?
- KNN?
  - Cluster on what features?

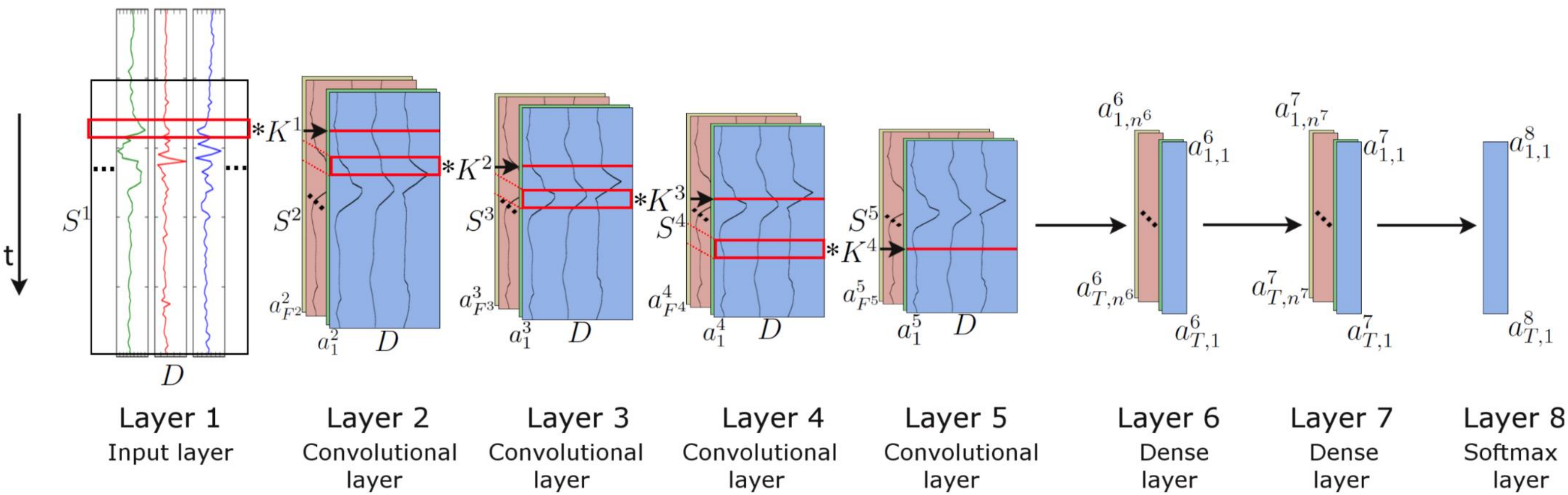
These methods do not suit well with very complex models.

Deep Learning



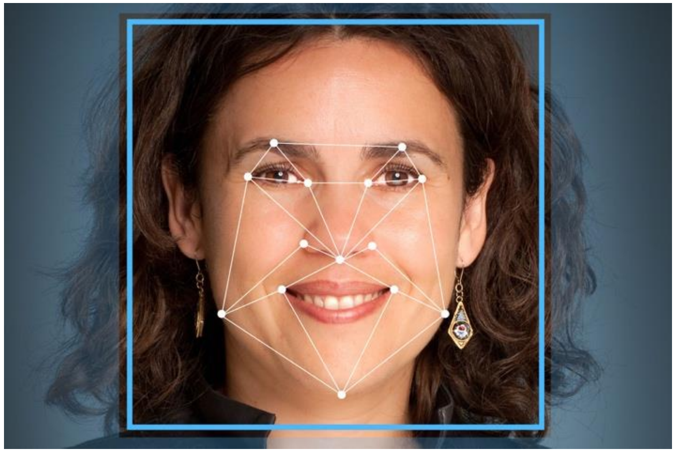
Deep Learning for Activity Recognition

- Example of applying a convolutional neural network



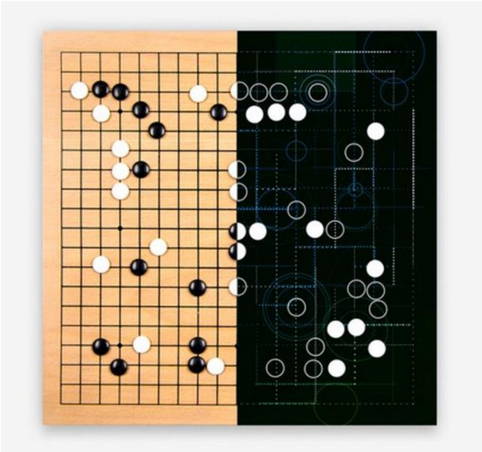
Deep Learning Applications

Self-Driving



Face Recognition

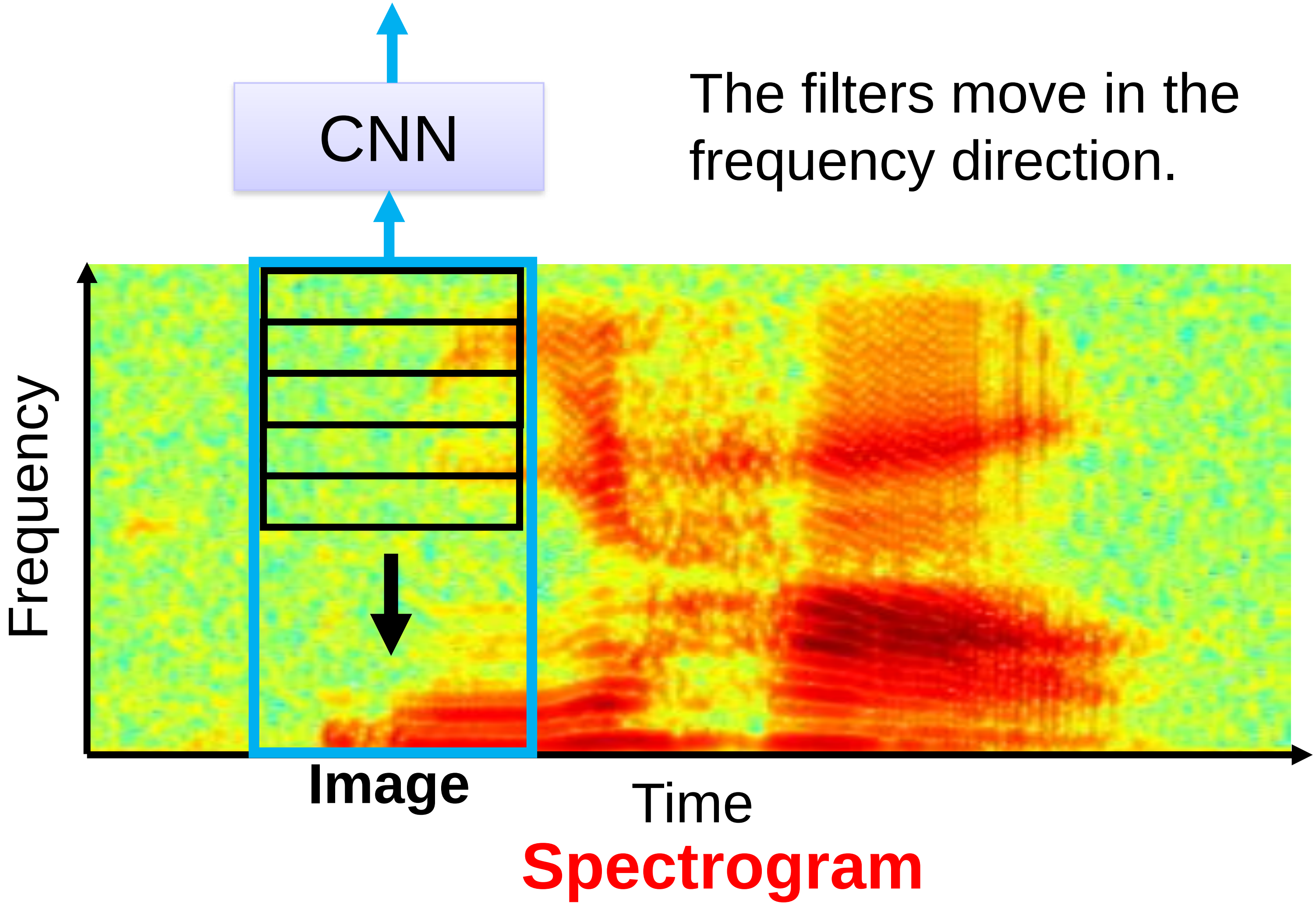
Speech Recognition



Play Go

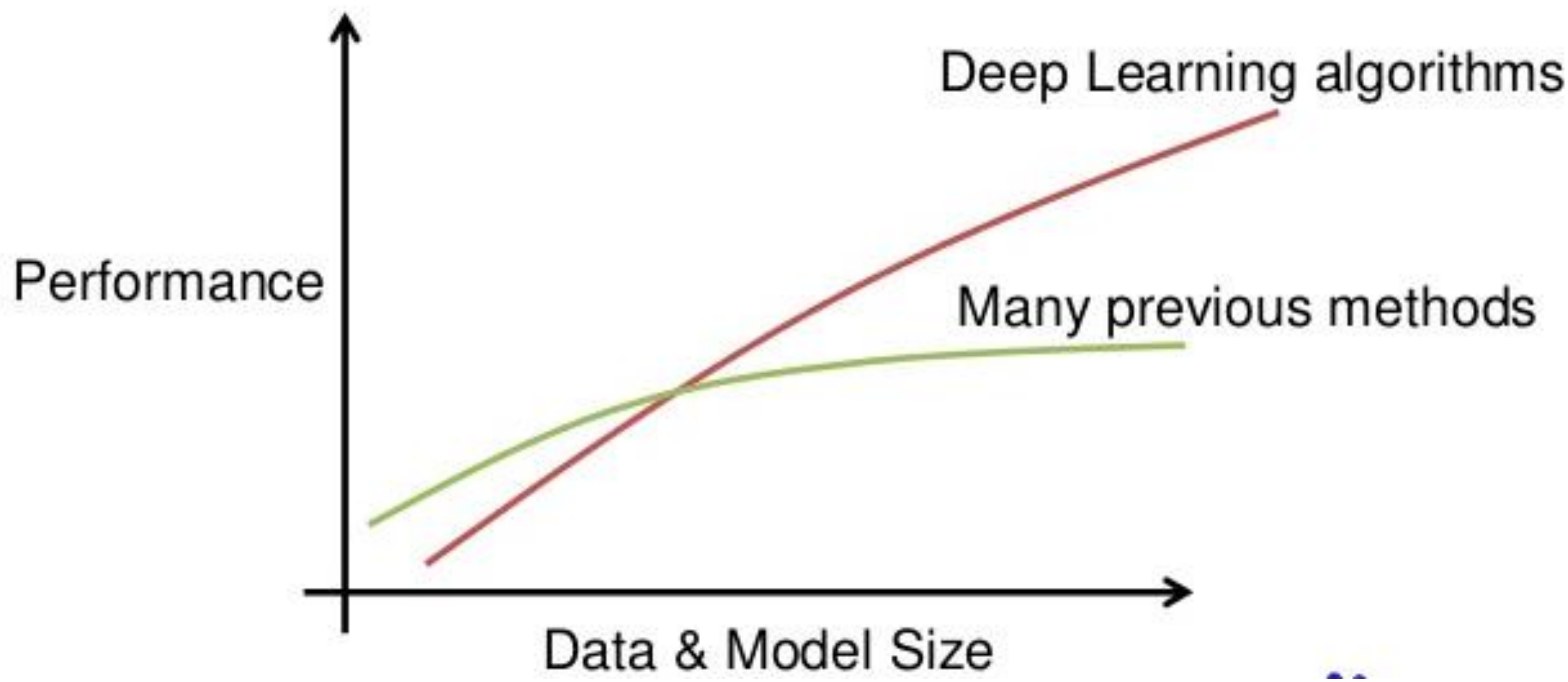
Deep Learning for Speech Recognition

The filters move in the frequency direction.



Machine Learning vs. Deep Learning

- Deep learning: the more data, the higher accuracy



Introduction to Convolutional Neural Network (CNN)

Convolutional Neural Network

- CNN is a feed-forward network that can extract topological properties from an image.
- Like almost every other neural networks they are trained with a version of the back-propagation algorithm.
- Convolutional Neural Networks are designed to recognize visual patterns directly from pixel images with minimal preprocessing.
- They can recognize patterns with extreme variability (such as handwritten characters).

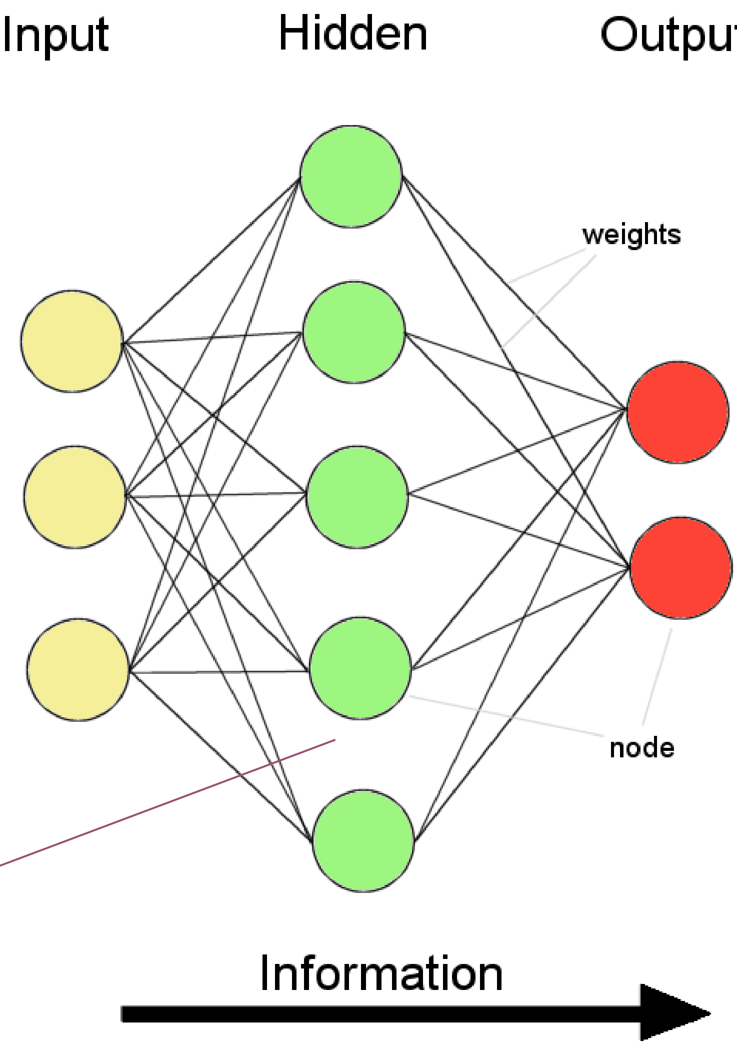
Feed-Forward Networks

Information flow is unidirectional

- Data is presented to *Input layer*
- Passed on to *Hidden Layer*
- Passed on to *Output layer*

Information is distributed

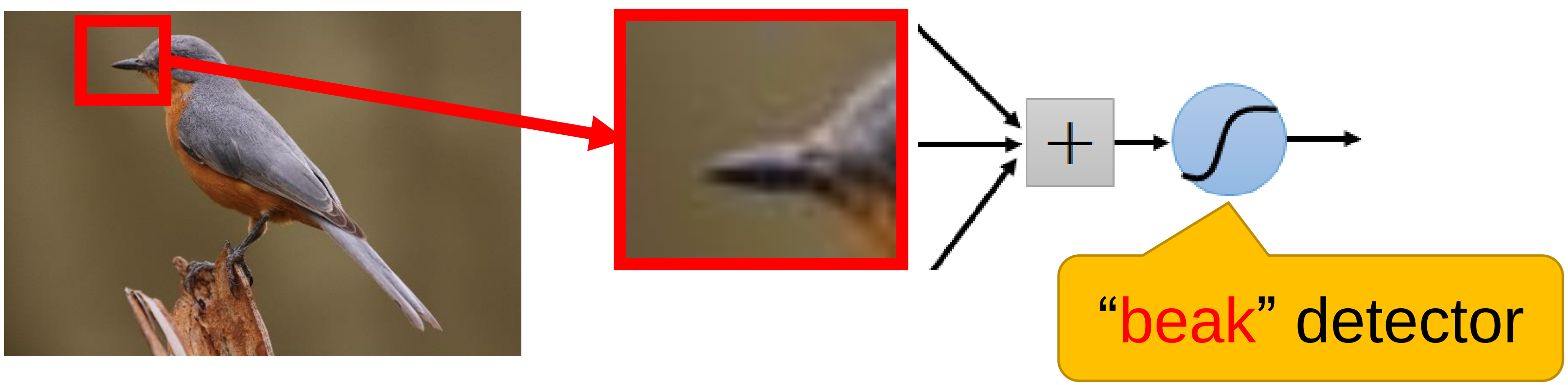
Information processing is parallel



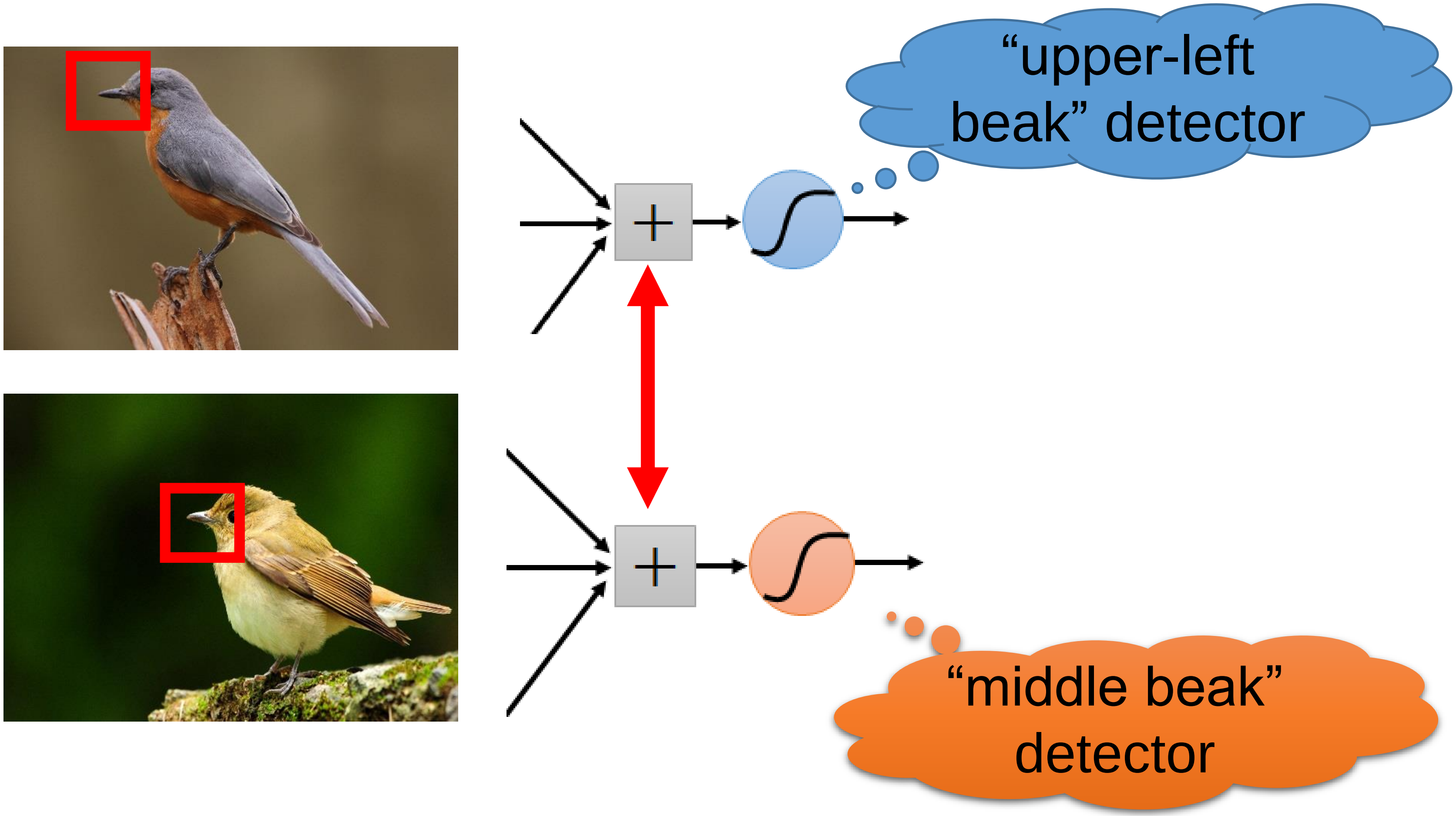
Internal representation
(interpretation) of data

Identifying a Bird in an Image

- Let's assume "beak" is unique to birds.
- "beak" exists in a small sub-region of an image.

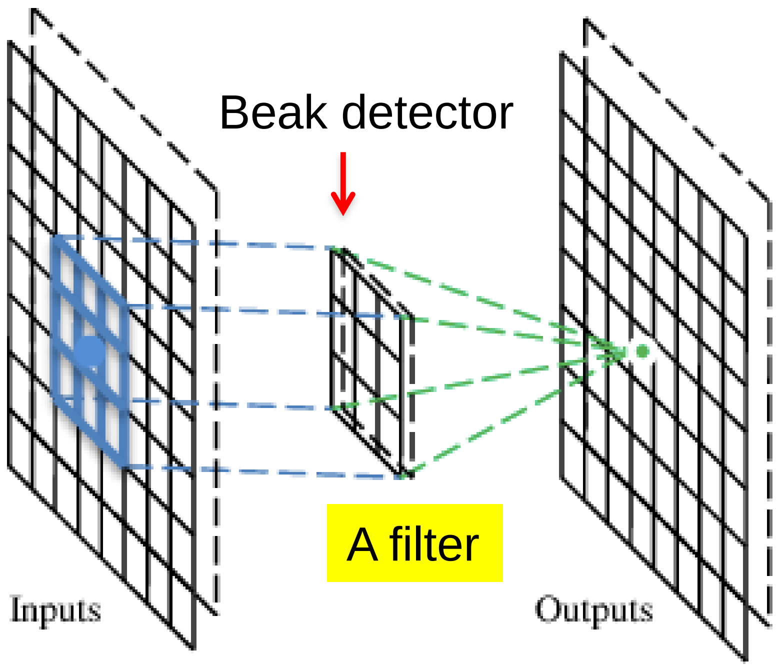


“Beak” in Different Parts of Images



Convolutional Layer

A Convolutional Neural Network (CNN) is a neural network with “convolutional layers”, which has a number of filters that does convolutional operation.



Convolution

These are the network parameters to be learned.

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2

⋮ ⋮

Each filter detects
a small pattern (3 x 3).

Convolution

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

Dot product → 3 -1

6 x 6 image

Convolution

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

If stride=2

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

3 -3

Convolution

stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

3	-1	-3	-1
-3	1	0	-3
-3	-3	0	1
3	-2	-2	-1

Convolution

-1	1	-1
-1	1	-1
-1	1	-1

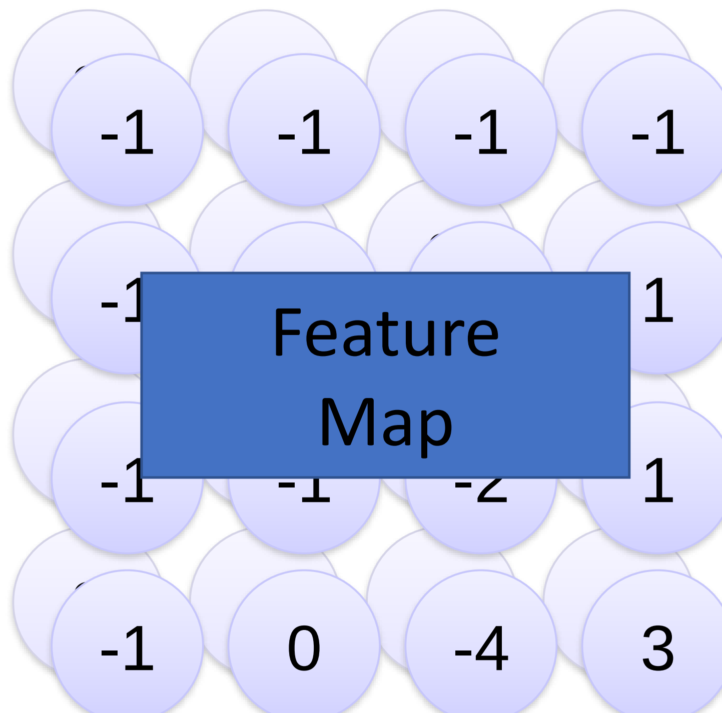
Filter 2

stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

Repeat this for each filter



Two 4 x 4 images
Forming 2 x 4 x 4 matrix

Color Image: RGB 3 Channels

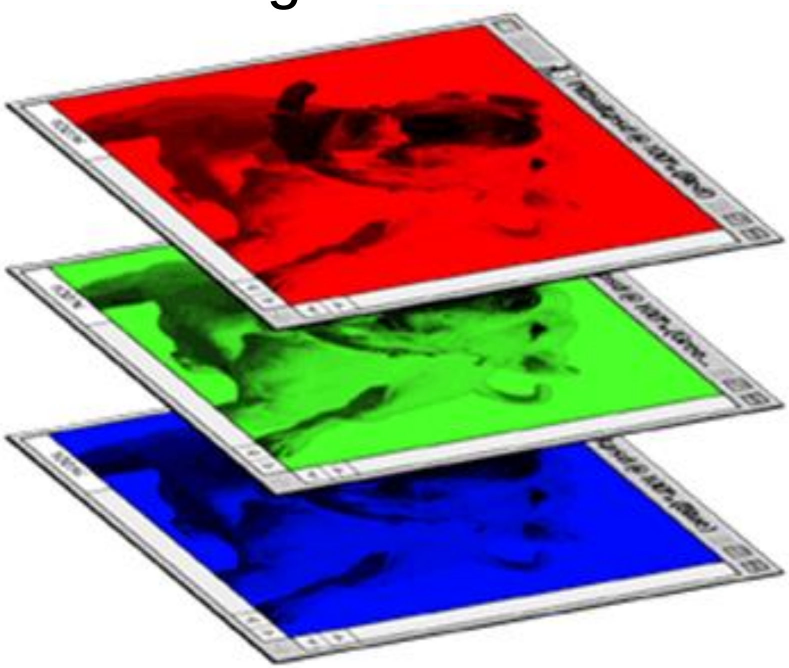
Filter 1

1	-1	-1
-1	1	-1
-1	-1	1

Filter 2

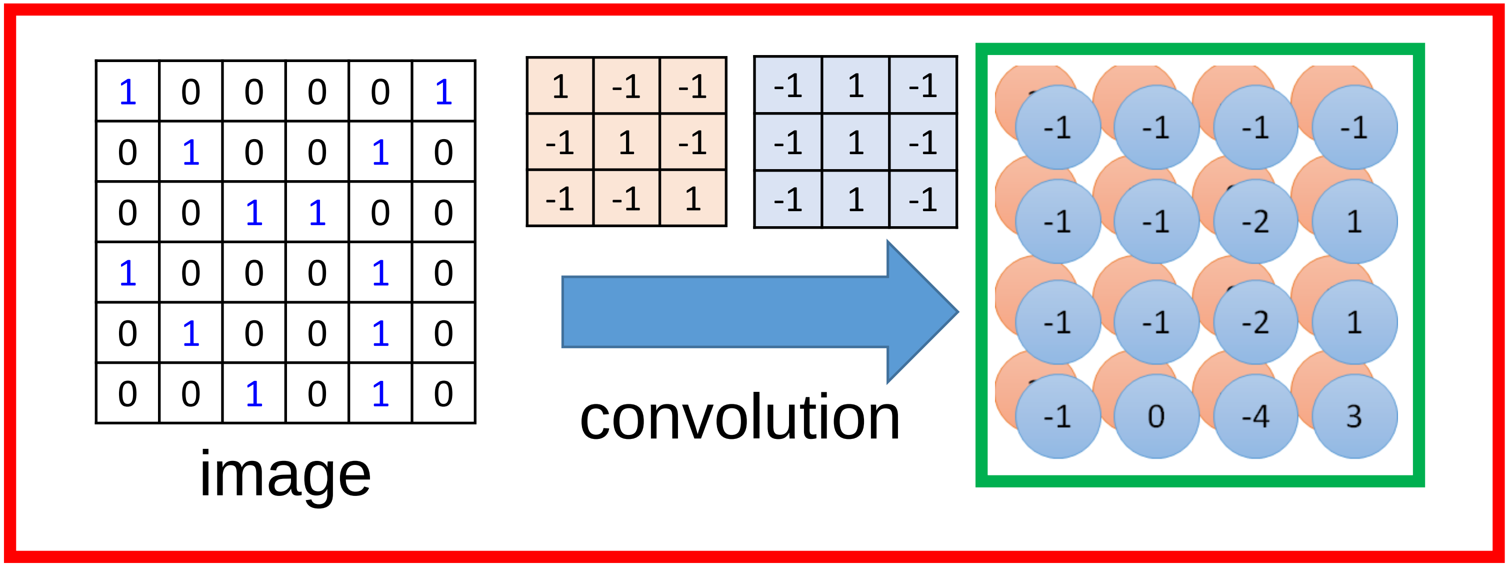
-1	1	-1
-1	1	-1
-1	1	-1

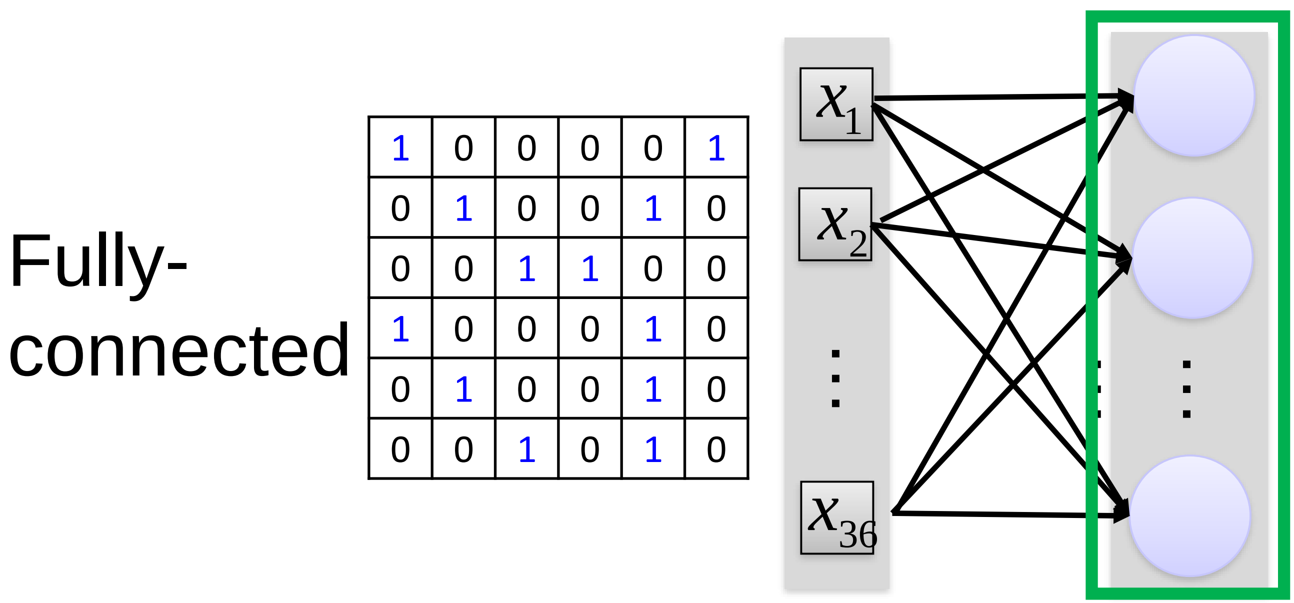
Color image

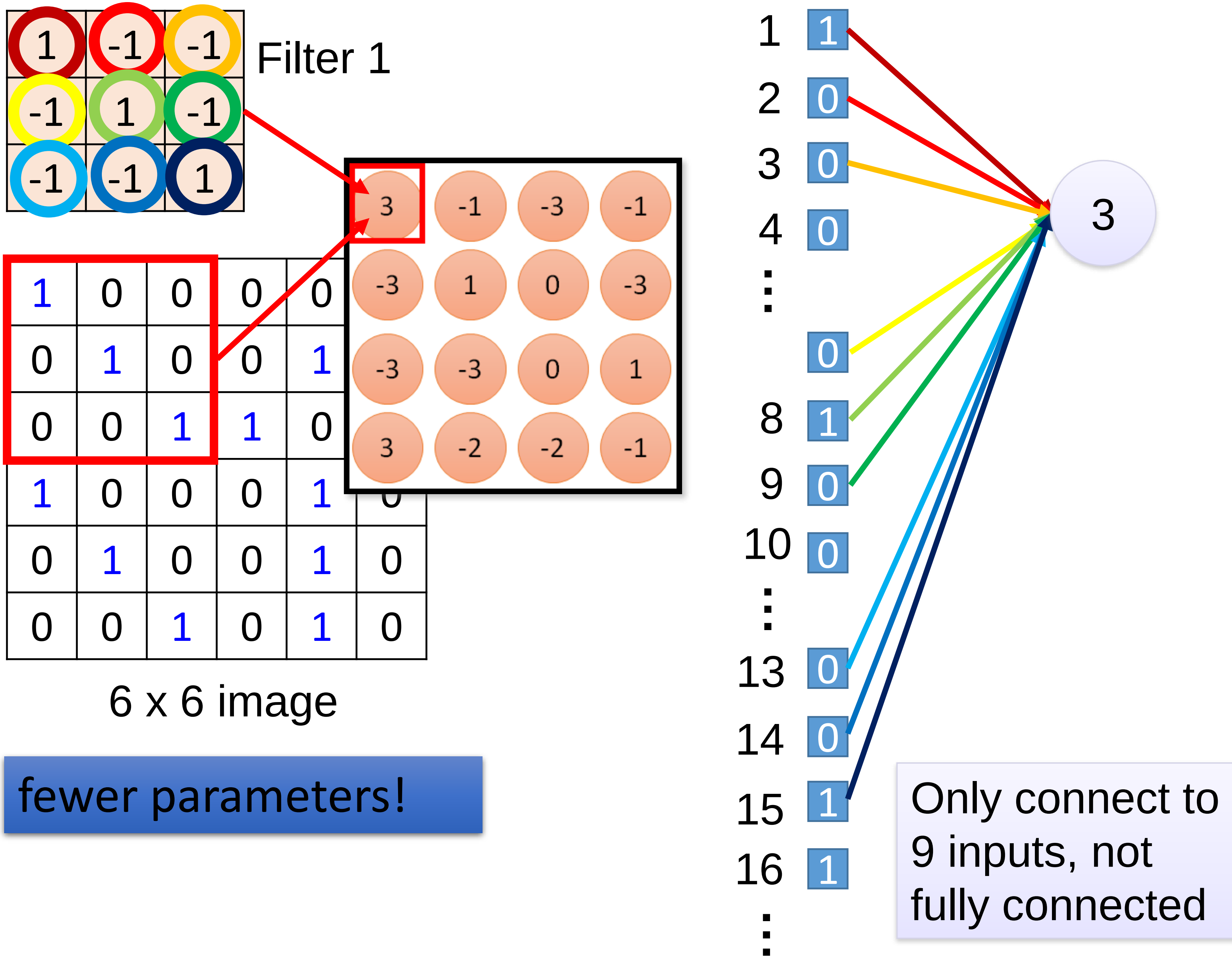


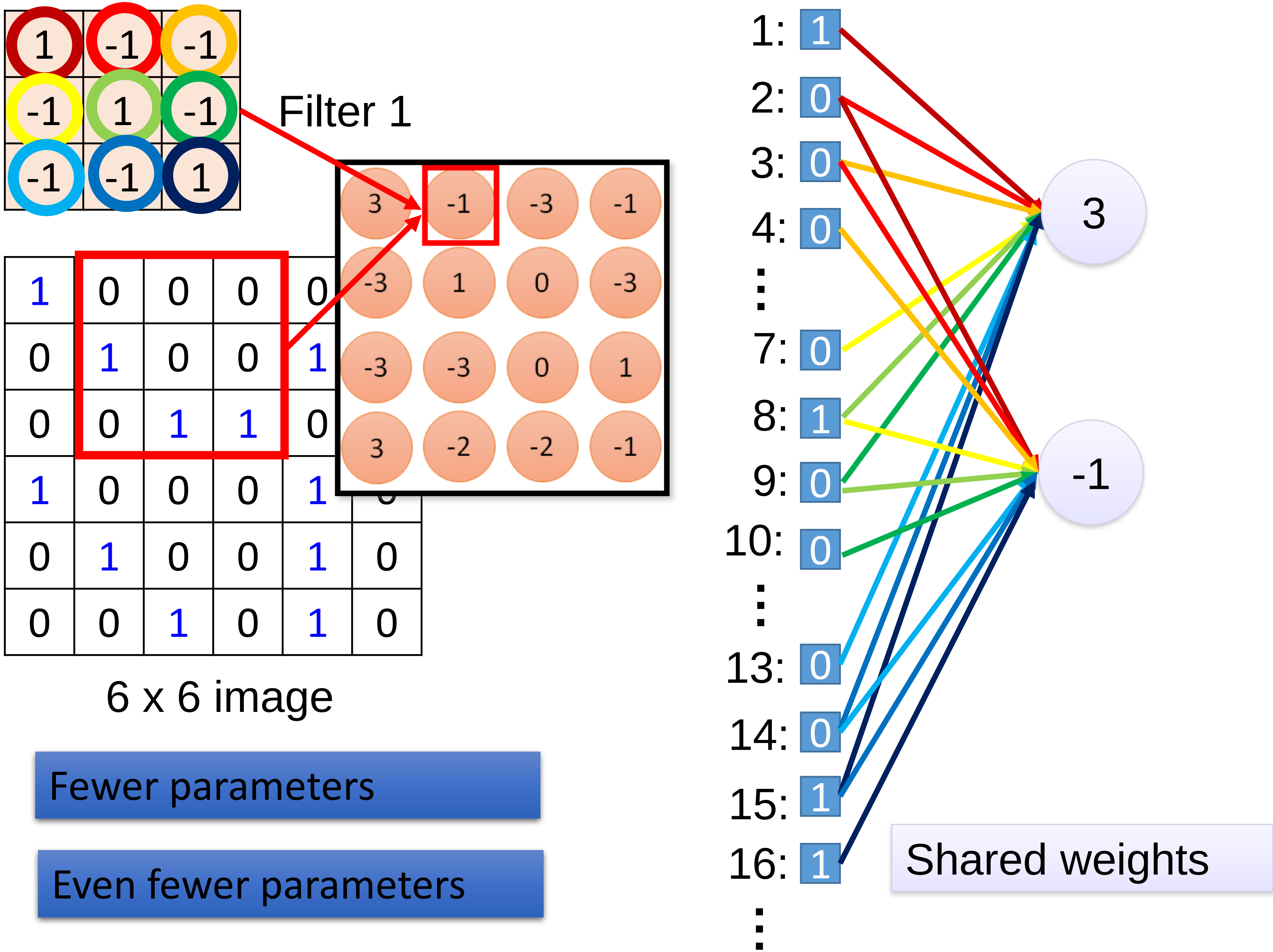
1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

How to Form a Feed Forward Network



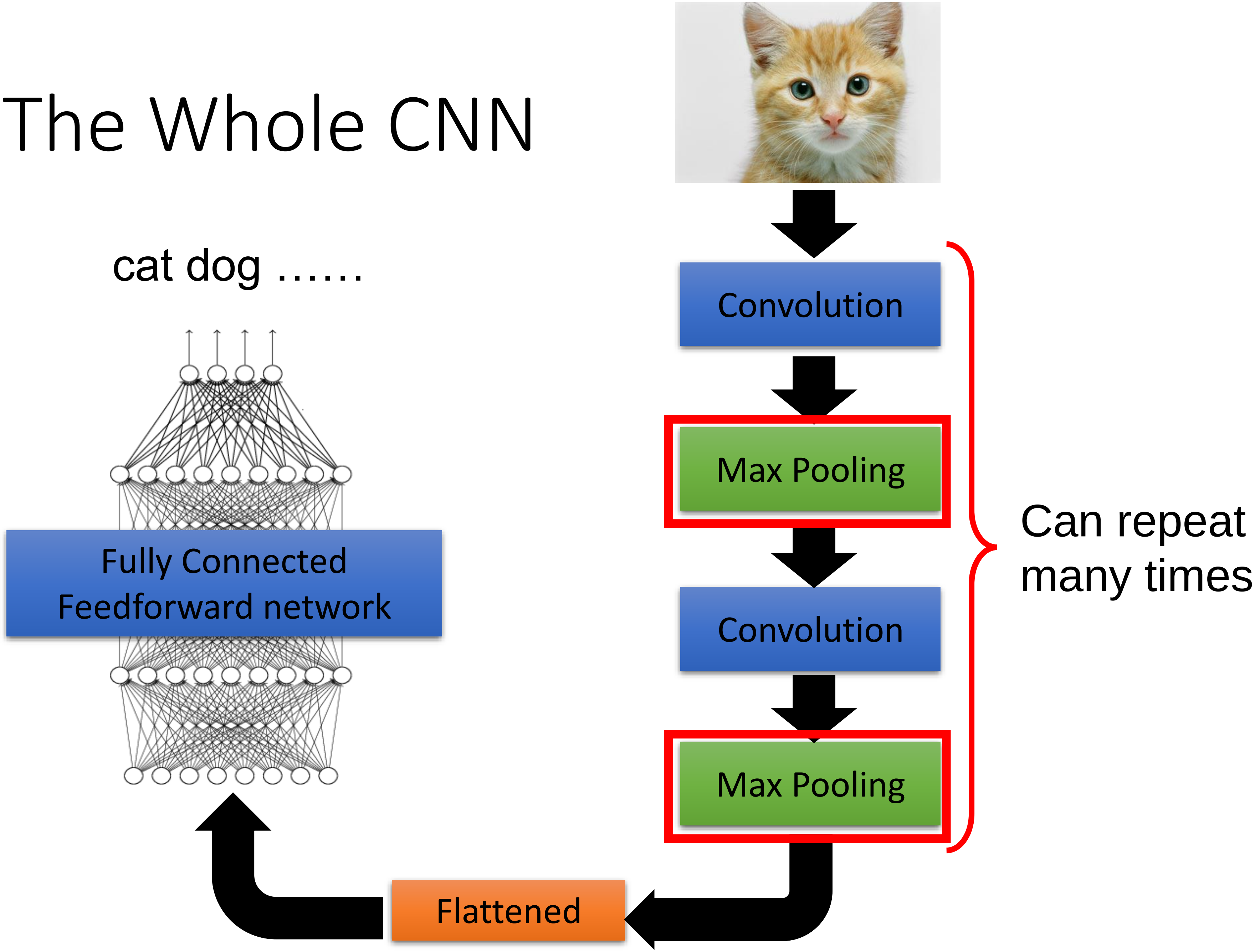



Filter 1
6 x 6 image
fewer parameters!
Only connect to 9 inputs, not fully connected
3



1 -1 -1
-1 1 -1
-1 -1 1
Filter 1
3 -1 -3 -1
-3 1 0 -3
-3 -3 0 1
3 -2 -2 -1
1 0 0 0 0 1
0 1 0 0 1 0
0 0 1 1 0 0
1 0 0 0 1 0
0 1 0 0 1 0
0 0 1 0 1 0
6 x 6 image
Fewer parameters
Even fewer parameters
1: 1
2: 0
3: 0
4: 0
7: 0
8: 1
9: 0
10: 0
13: 0
14: 0
15: 1
16: 1
3
-1
Shared weights

The Whole CNN



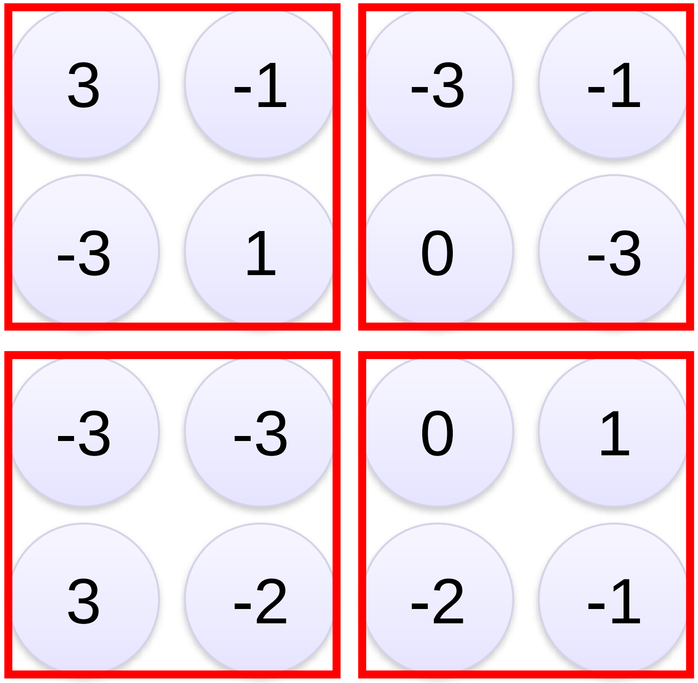
Max Pooling

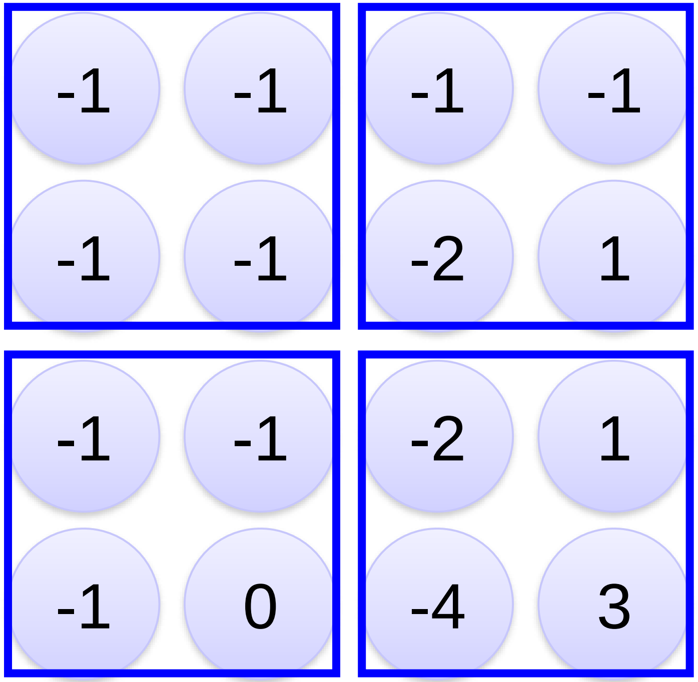
1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2





Why Pooling

- Subsampling pixels will not change the object

bird

Subsampling

bird

We can subsample the pixels to make image smaller

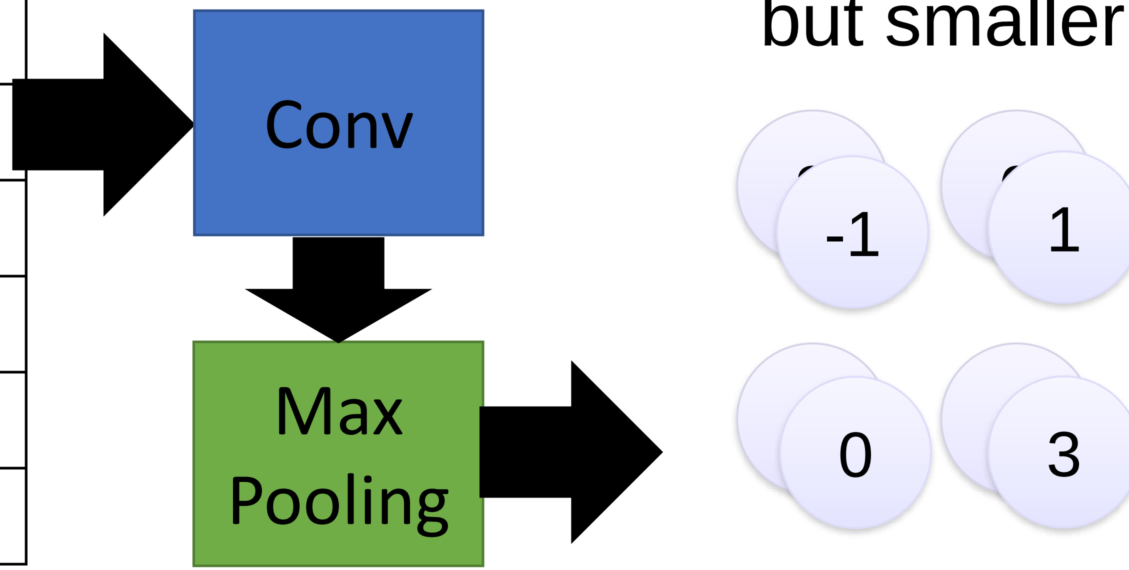
fewer parameters to characterize the image

Max Pooling

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

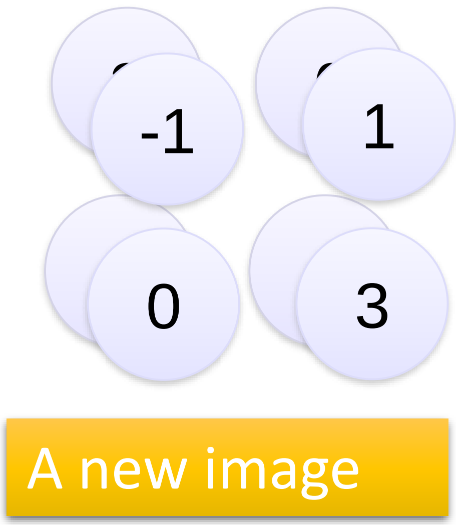
New image
but smaller

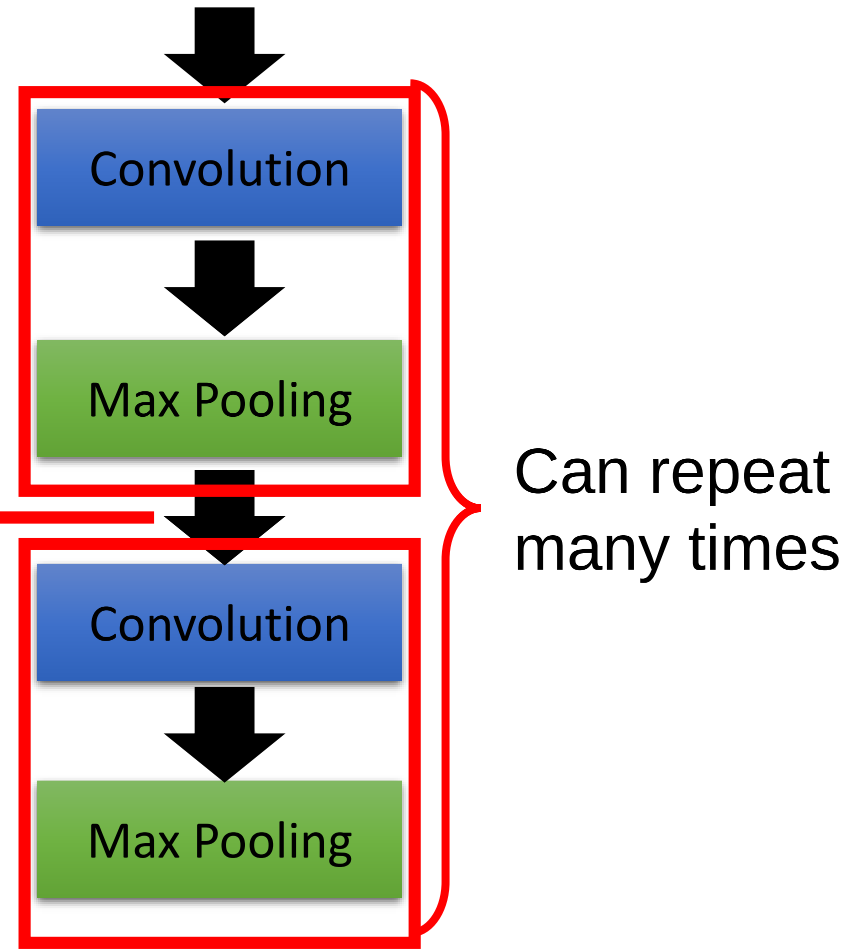


2 x 2 image

Each filter
is a channel

The Whole CNN

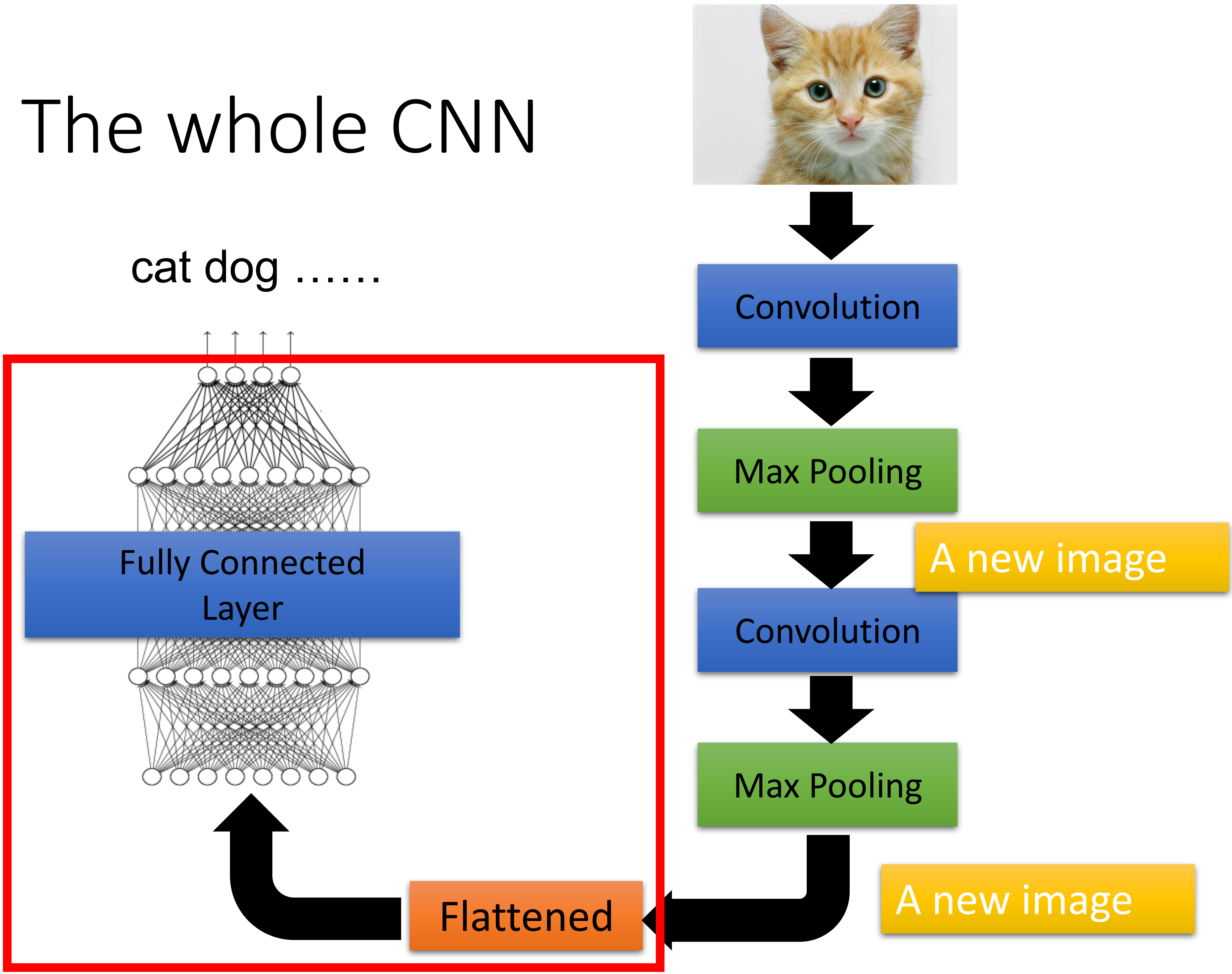




Smaller than the original image

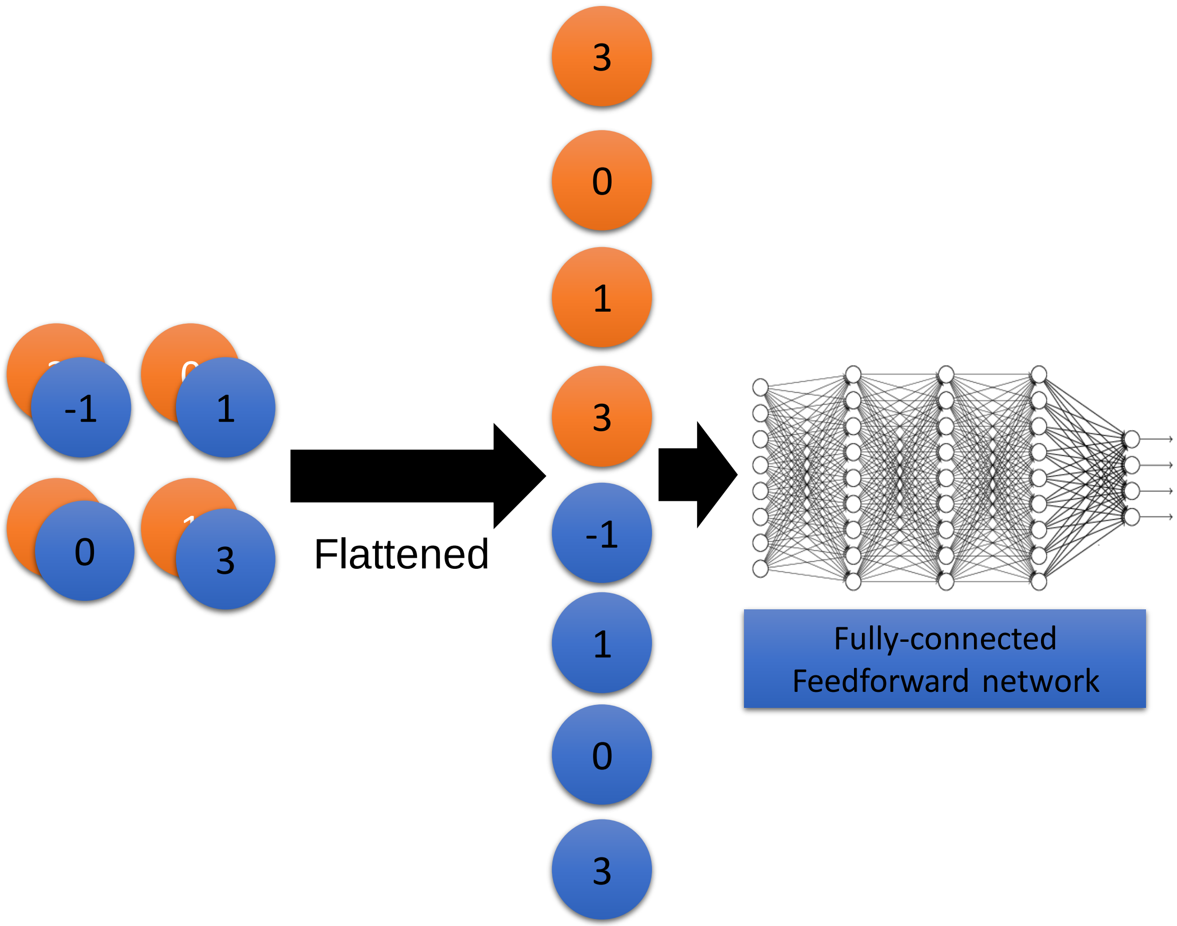
The number of channels is the number of filters

The whole CNN



Fully Connected Layer

Conceptually, this can be understood as the voting process to see which input values contribute more to the output.



cat dog ……

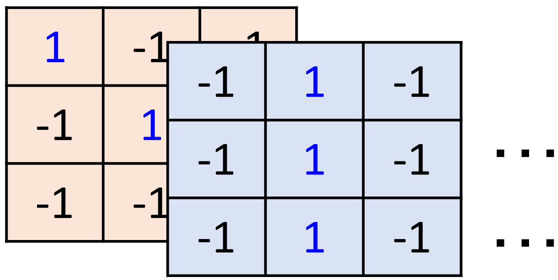
Tools and APIs

- Tensorflow (https://www.tensorflow.org/)
  - Tensorflow light for Mobile and IoT
- PyTorch (https://pytorch.org)
- Caffe2 (https://caffe2.ai)
- Keras (https://keras.io/)

CNN in Keras

Only modified the ***network structure*** and ***input format (vector -> 3-D tensor)***

```
model2.add( Convolution2D( 25,3,3,
            input_shape=(28,28,1)) )
```



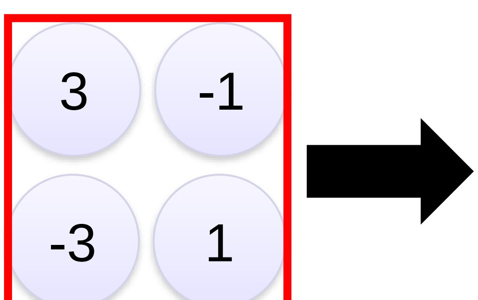
There are 25 3x3 filters.

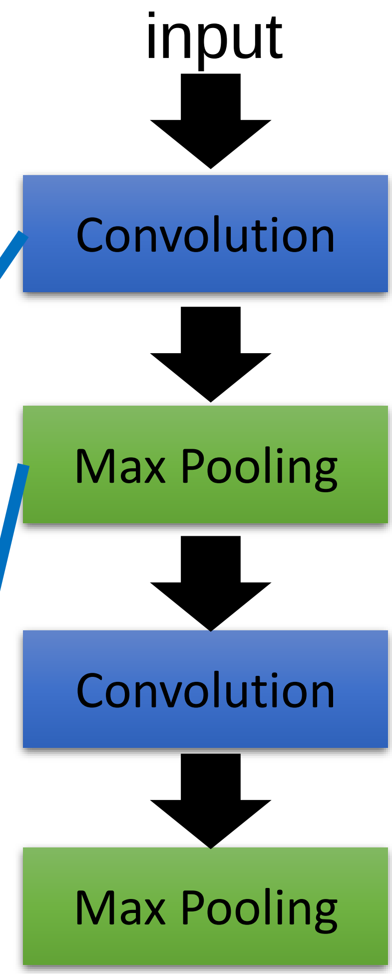
Input_shape = (28 , 28 , 1)

28 x 28 pixels

1: black/white, 3: RGB

```
model2.add(MaxPooling2D((2,2)))
```





CNN in Keras

Only modified the ***network structure*** and ***input format (vector -> 3-D array)***

Input

1 x 28 x 28

```
model2.add( Convolution2D( 25,3,3,
        input_shape=(28,28,1)) )
```

Convolution

How many parameters for each filter? 9

25 x 26 x 26

```
model2.add(MaxPooling2D((2,2)))
```

Max Pooling

25 x 13 x 13

```
model2.add(Convolution2D(50,3,3))
```

Convolution

How many parameters for each filter? 225= 25x9

50 x 11 x 11

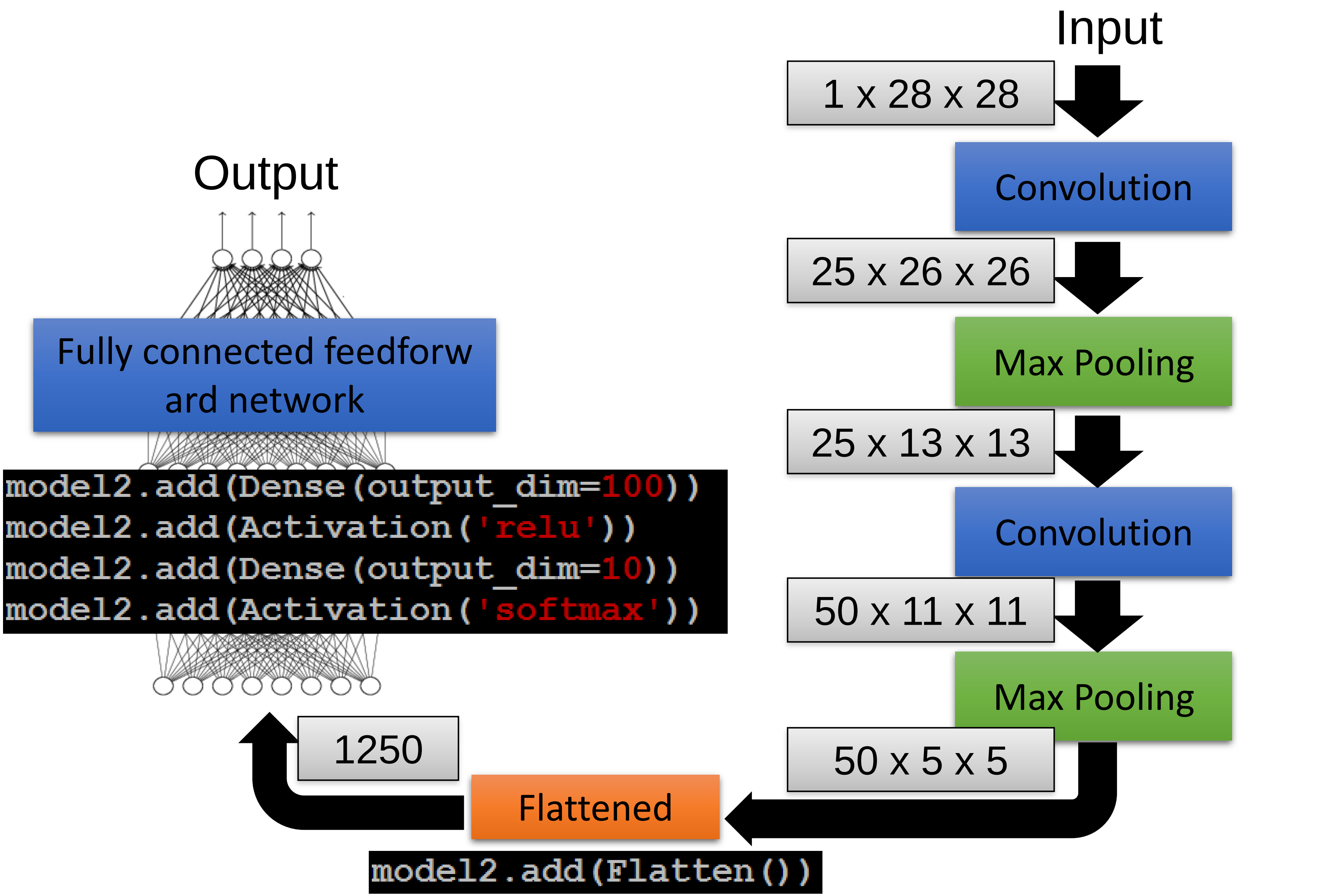
```
model2.add(MaxPooling2D((2,2)))
```

Max Pooling

50 x 5 x 5

CNN in Keras

Only modified the ***network structure*** and ***input format (vector -> 3-D array)***



AlphaGo

19 x 19 matrix

Neural Network

Next move

(19 x 19 positions)

Black: 1

white: -1

none: 0

Fully-connected feedforward network can be used

But CNN performs much better

AlphaGo’s policy network

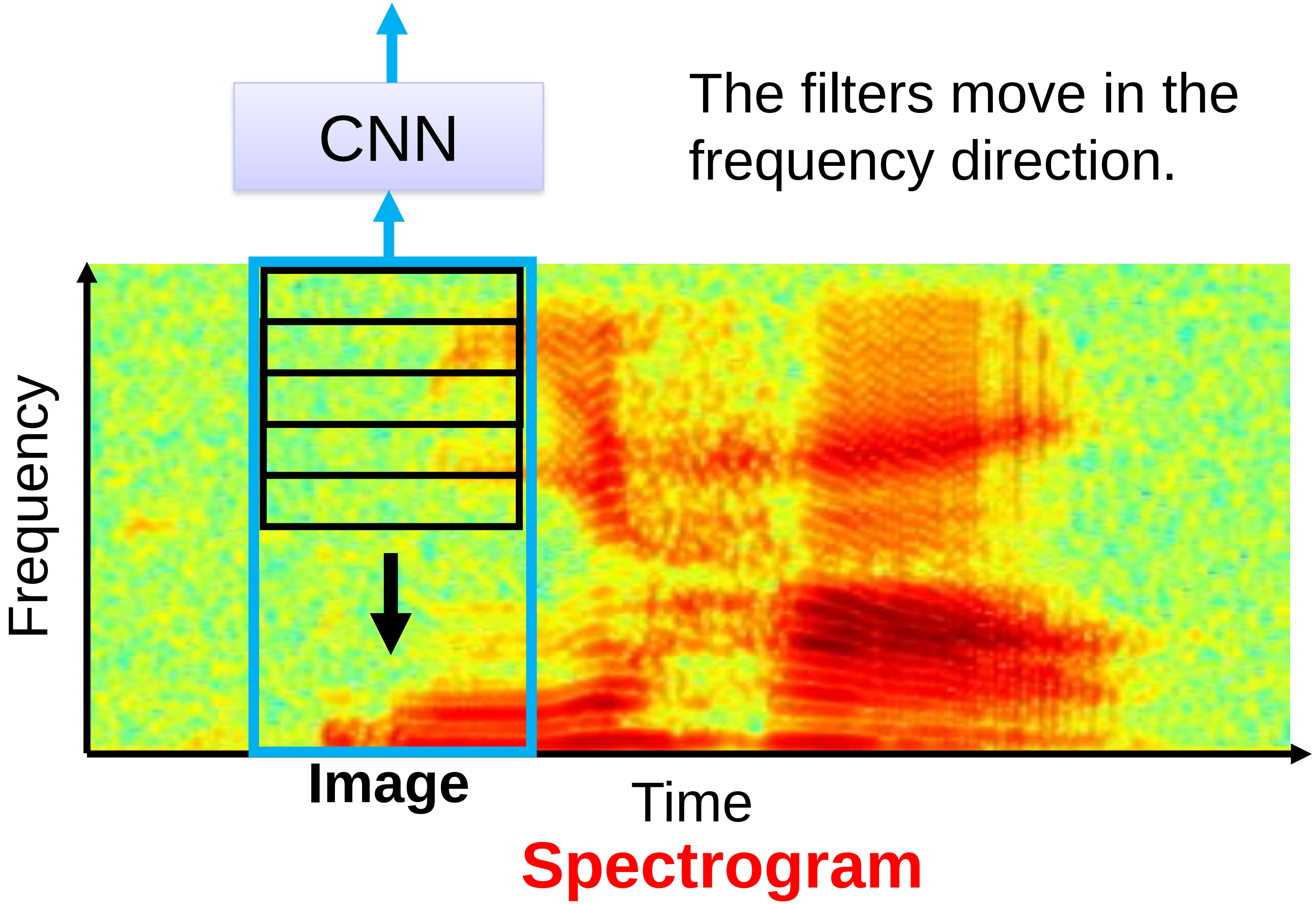
The following is quotation from their Nature article:

Note: AlphaGo does not use Max Pooling.

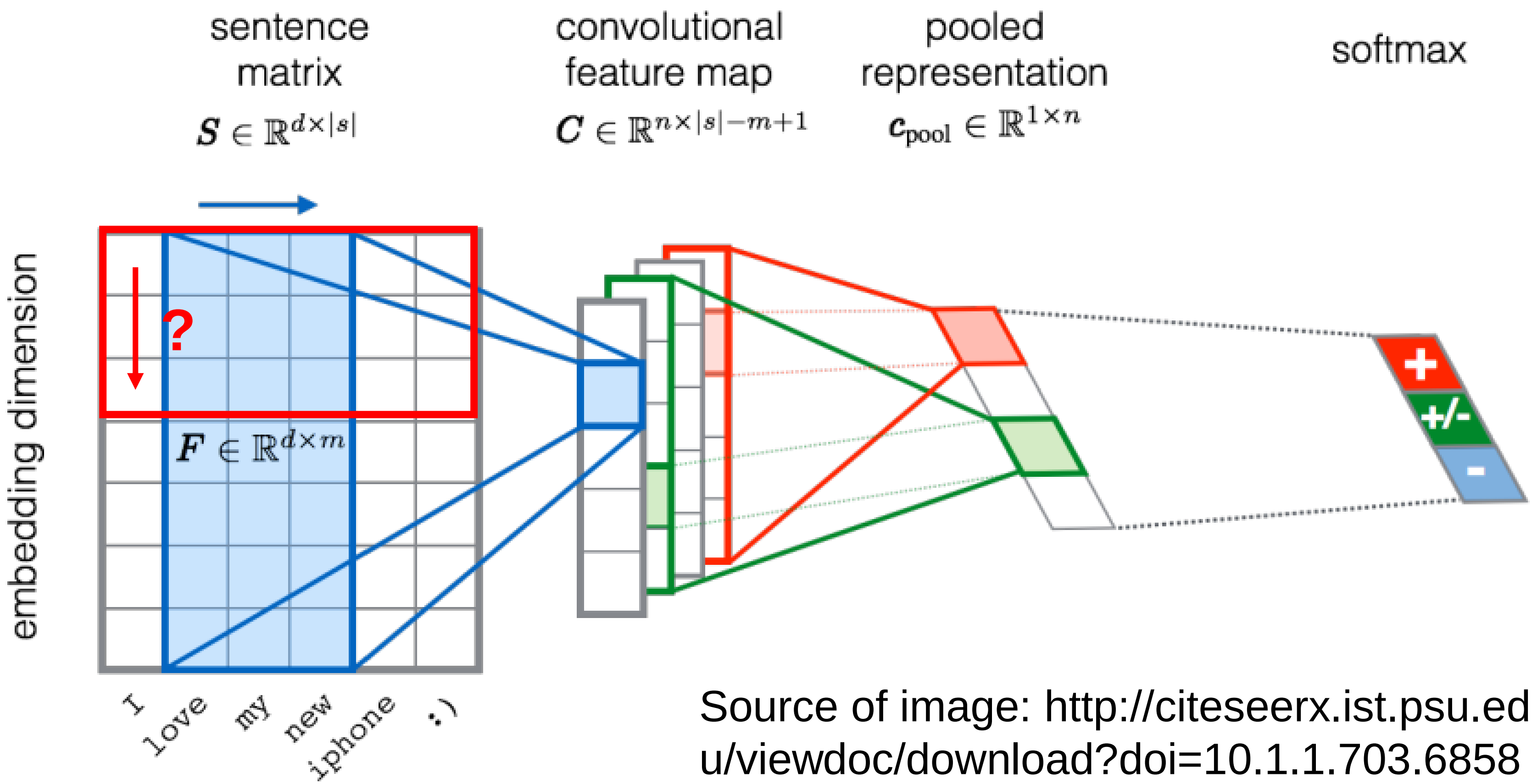
Neural network architecture. The input to the policy network is a $19 \times 19 \times 48$ image stack consisting of 48 feature planes. The first hidden layer zero pads the input into a 23×23 image, then convolves k filters of kernel size 5×5 with stride 1 with the input image and applies a rectifier nonlinearity. Each of the subsequent hidden layers 2 to 12 zero pads the respective previous hidden layer into a 21×21 image, then convolves k filters of kernel size 3×3 with stride 1, again followed by a rectifier nonlinearity. The final layer convolves 1 filter of kernel size 1×1 with stride 1, with a different bias for each position, and applies a softmax function. The match version of AlphaGo used $k = 192$ filters; Fig. 2b and Extended Data Table 3 additionally show the results of training with $k = 128$, 256 and 384 filters.

CNN in speech recognition

The filters move in the frequency direction.



CNN in text classification



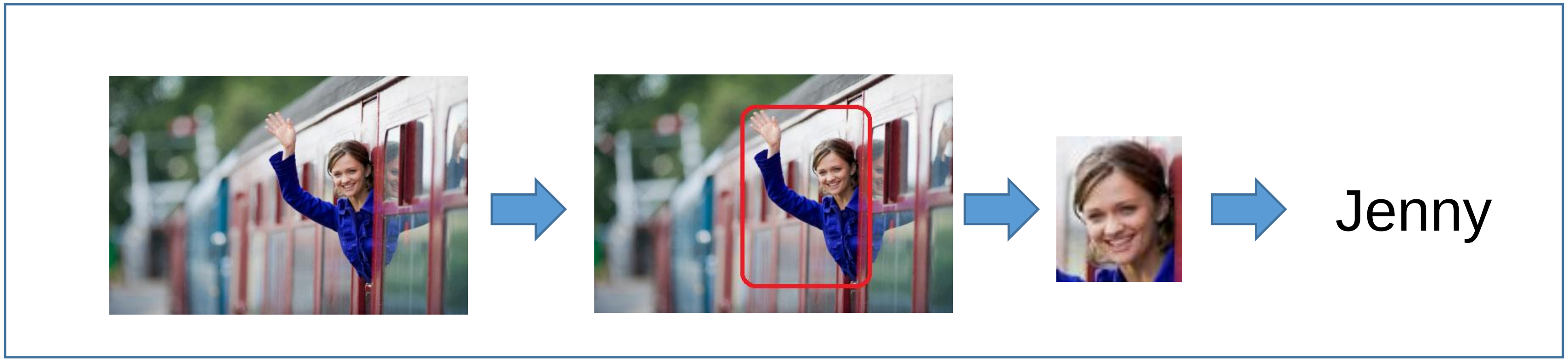
Source of image: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.703.6858&rep=rep1&type=pdf

DeepMon: Mobile GPU-based Deep Learning Framework for Continuous Vision Applications

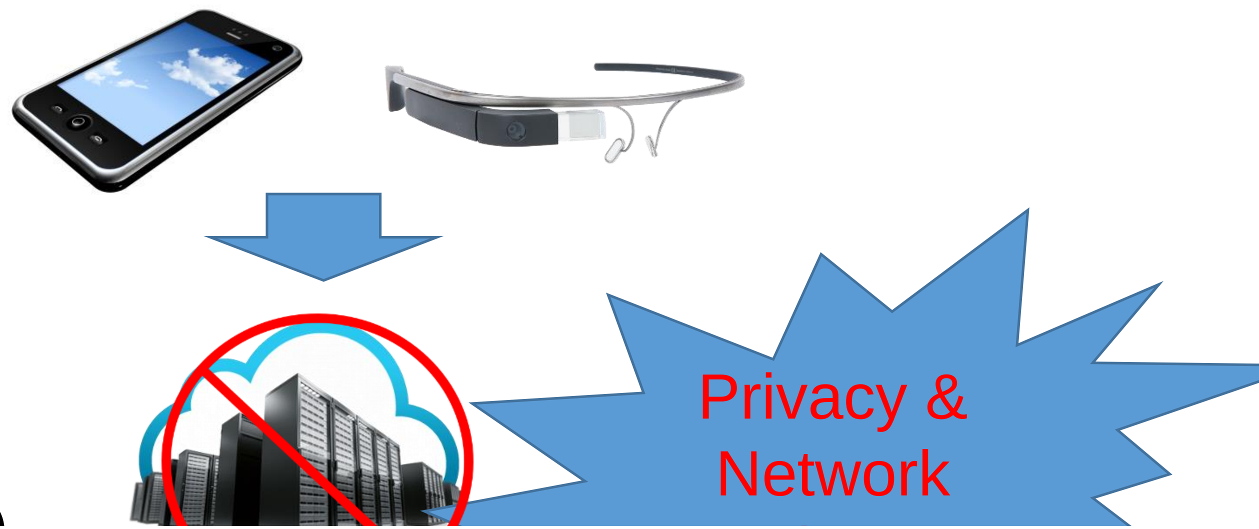
ACM MobiSys 2017

Continuous Vision Applications

Conventional Processing Flow



Capture frames
& Process
(with DNN models
such as YoLo)



Process frames
(with DNN models)

Research Question: Can we support fully-disconnected DNN-based inference purely on the mobile device?

DeepMon: Mobile Deep Learning System

- Supports low-latency execution of CNNs on commodity mobile devices using mobile GPUs
  - OpenCL/Vulkan enabled devices
- Supports multiple GPU architectures & mobile OSs
  - Mali, Adreno, PowerVR (to be supported)
- Supports existing trained models
  - Multiple frameworks (Caffe, Matconvnet, Yolo, Darknet)
- Available today- https://github.com/JC1DA/deepmon

Challenge 1: Mobile GPU is Weak!

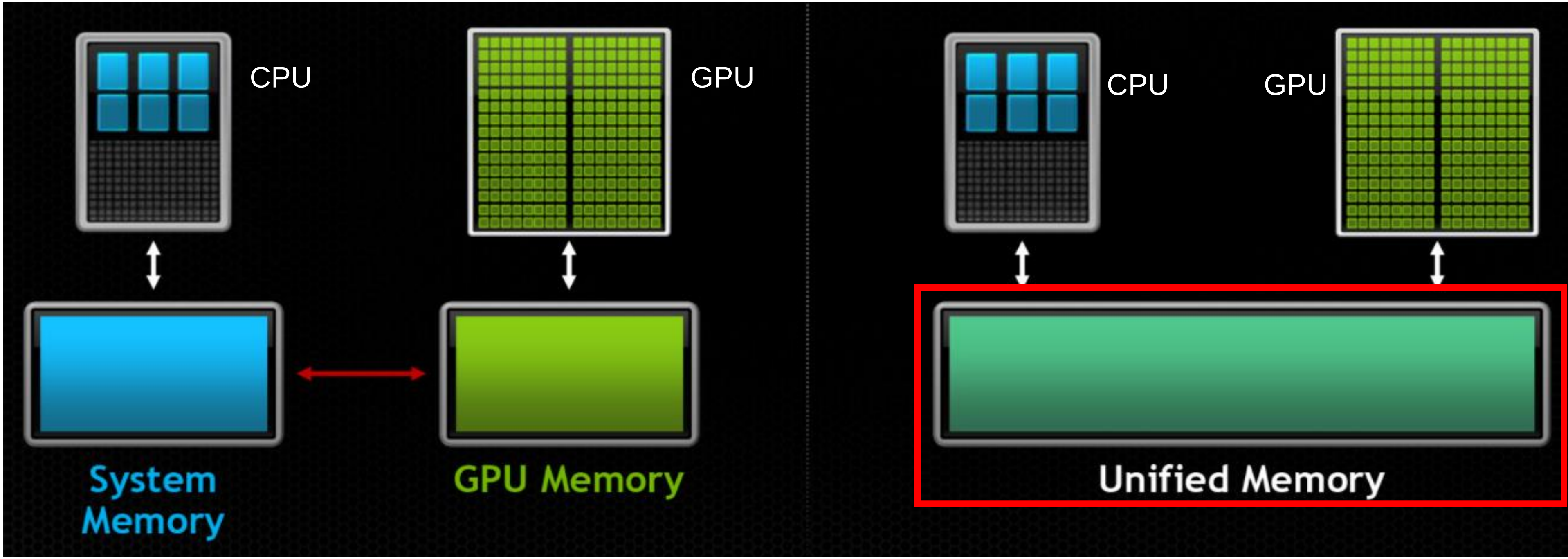
DNNs can well run on desktop GPUs, but…

	Nvidia GTX 980	Adreno 330 (Samsung Note 4)	Ratio (Nvidia / Adreno)
Number of ALUs (CUDA cores)	2048	128	16x
Memory Band[illegible]	[illegible]	[illegible]	17.5x
Peak Per[illegible] ps)	[illegible]	166.5	27.6x
CNN Execution Time VGG-16 (ms)	238.59 (Caffe)	6315	~26.5x

Too high latency to support continues vision.

Challenge 2: Architecture is Different!

- Existing GPU-based optimizations won't simply work!



Desktop GPU **Mobile GPU**

http://cdn.wccftech.com/wp-content/uploads/2014/03/NVIDIA-Maxwell-Unified-Virtual-Memory.jpg

Identifying Latency Bottleneck

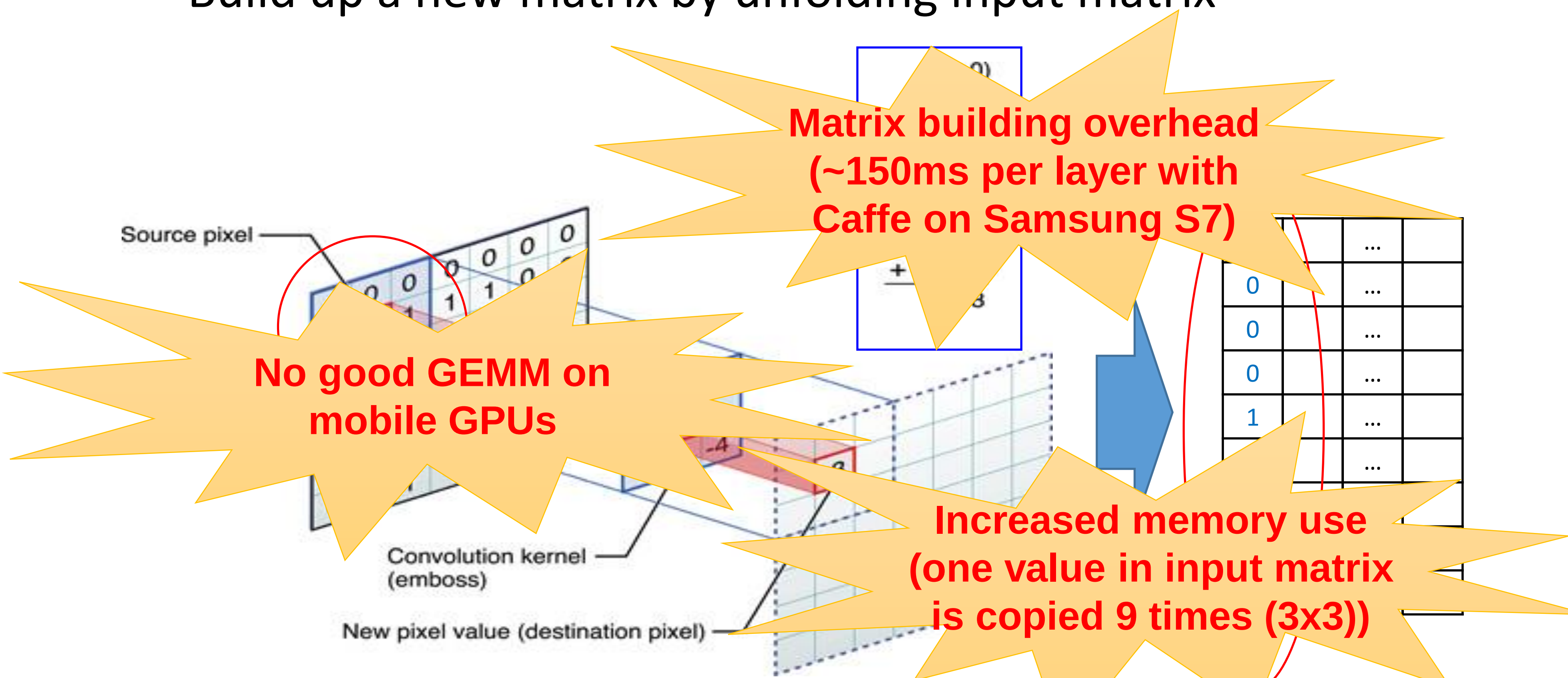
- Convolutional Neural Network
- Device: Samsung Galaxy Note 4
- Implementation: naïve CPU

Model	Conv. (ms)	FC. (ms)	Pooling (ms)	Total (ms)
VGG-F	8072	1079	26	9177
VGG-M	19521	2122	156	21800
VGG-16	213371	2408	882	21662

~90% time consumed by Convolutional Layers

Problem 1: High Memory Use

- Fast convolution operations on desktop GPU
  - Use optimized general matrix multiplication (GEMM)
  - Build up a new matrix by unfolding input matrix



https://i.stack.imgur.com/GvsBA.jpg

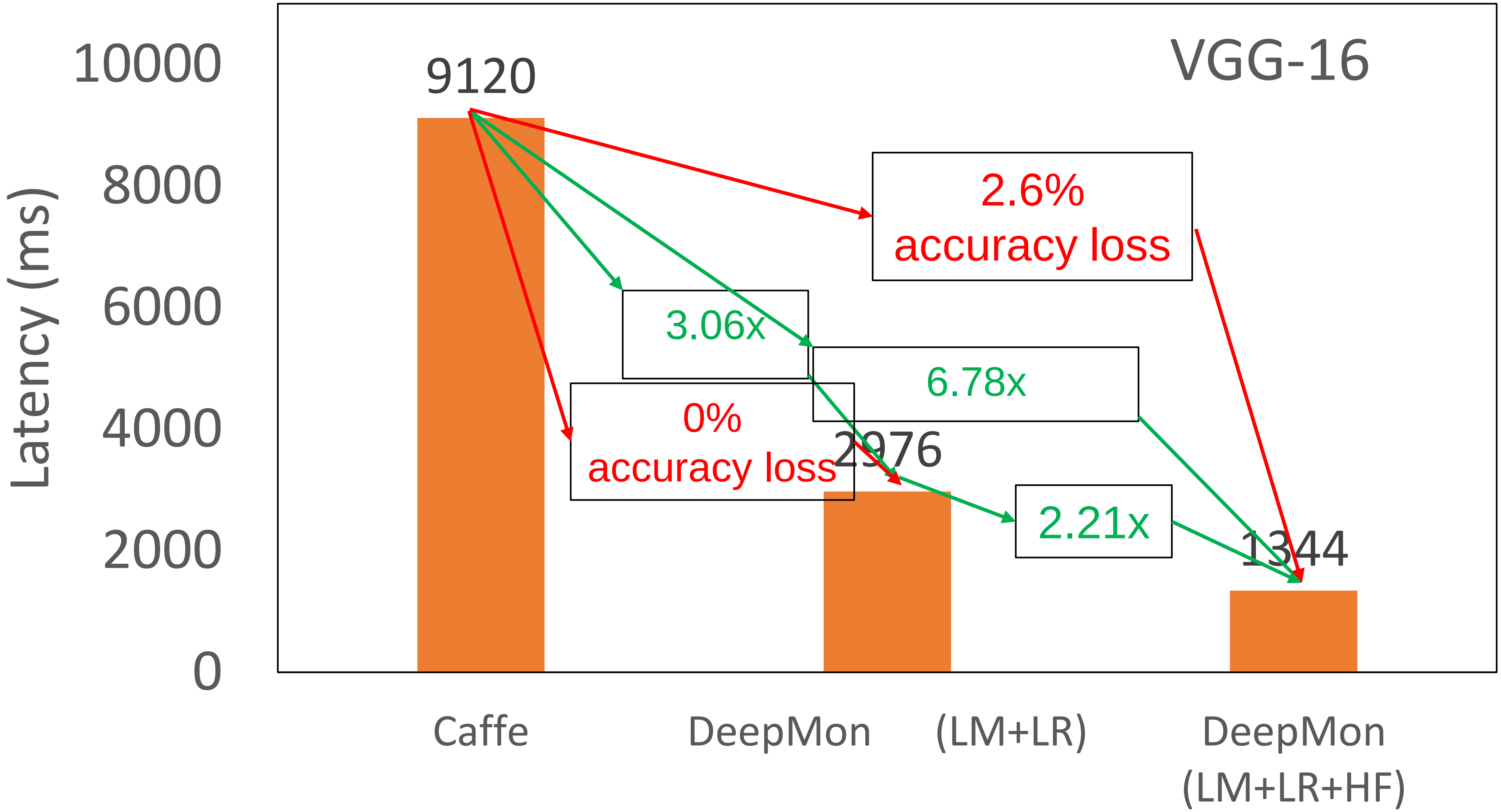
Solution 1: mGPU-Aware Optimizations

- Do convolution operations directly on input
  - No matrix building overhead
  - Less memory consumption

- Do mGPU-aware Optimizations
  - Leverage local memory (high performance cache inside GPU) to reduce memory reading
    - Store reusable convolutional kernels inside the local memory
    - It will be shared across multiple threads
  - Layout the input data to enable fast vector addition/ multiplication on mGPUs
    - The data in vectors need to be consecutively stored in the memory.
  - Use half floating point (32 bits → 16 bits)

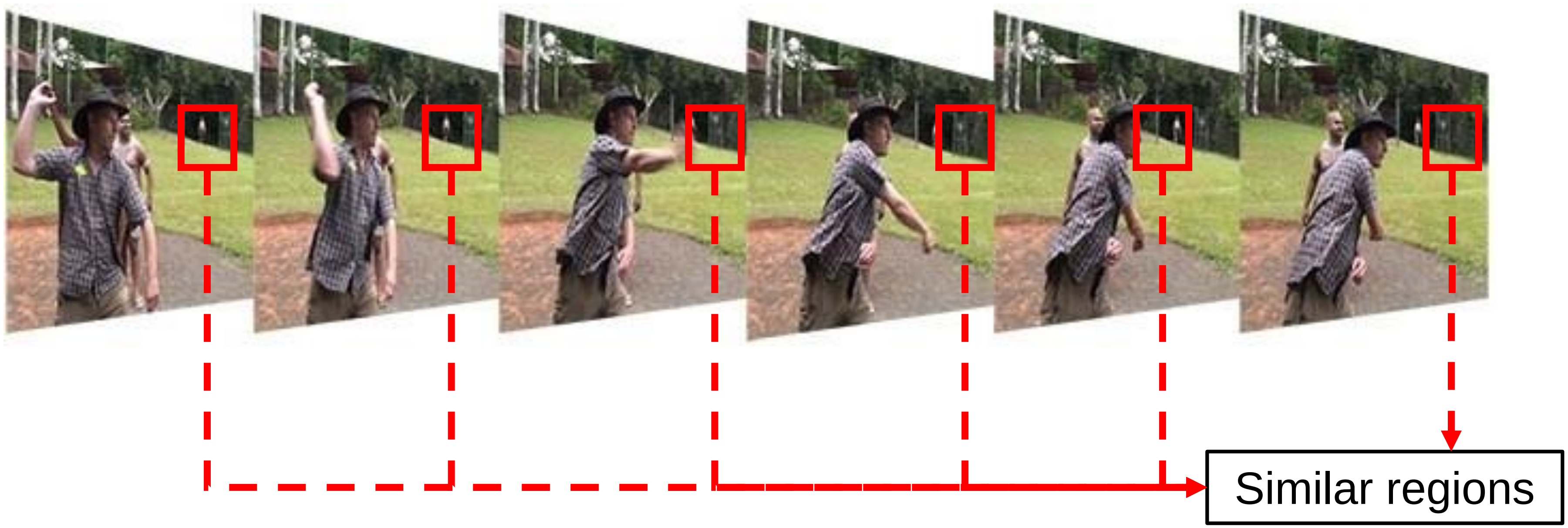
Impact of Memory Optimization

measured on Samsung s7 (Mali T770)



LM: Local Memory – LR: Layout Redesign – HF: Half Floating point

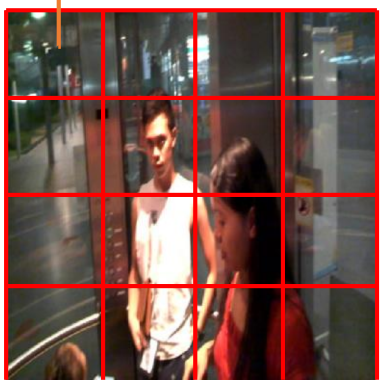
Problem 2: Redundant Computation

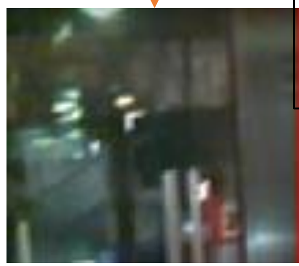


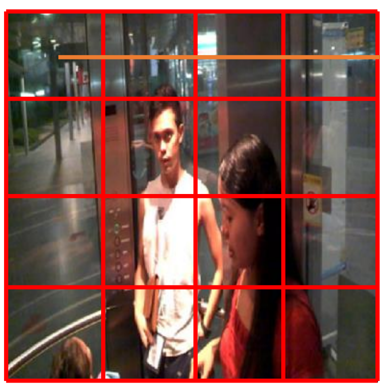
- Background in continuous video frames tends to be static.
- Independent processing of each frame is redundant.

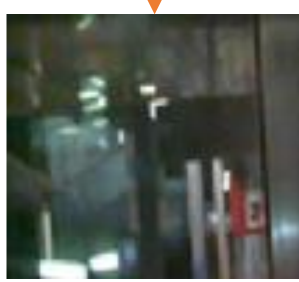
Key idea: Can we reuse the intermediate results of previous convolutional layer computation for similar regions?

Solution 2: Convolutional Caching









- Light-weight & accurate comparisons required!
- SIFT-based approach did not work.

Histogram-Based Sub-Image Comparison

Reusable?

yes

Reuse cached Conv. Op. results

no

Perform Conv. Op.

Cache Manager

- 16 bins color histogram
- Chi square distance
- If distance < 0.005, block is reusable

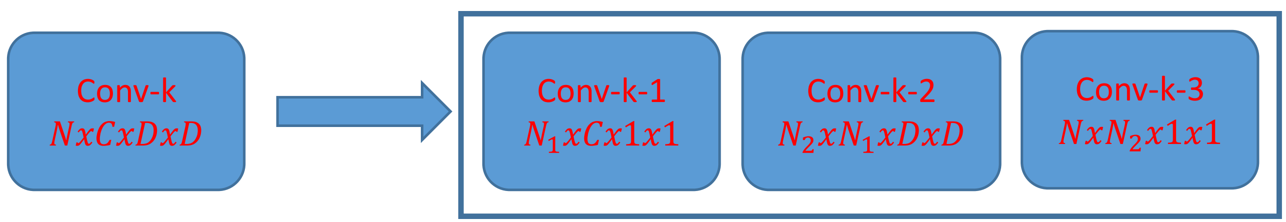
Solution 3: Decomposition

- To decompose large convolutional layer into a sequence of several smaller ones so computation cost can be reduced

- Tucker-2 decomposition
  - Decompose a convolutional layer into 3 parts
    - 2 with filter size of (1x1)
      - 1st layer acts as dimension reduction -> reduce computational cost
      - 2nd layer acts as dimension restoration -> guarantee output size equal to output size of original convolutation layer
    - 1 with original filter size
      - have lower number of input/output channels -> reduce computational cost

Tucker-2 Decomposition

- N: number of filters
- C: number of input channels
- D: filter size (D ≥ 3)
- Input: (HxWxC)

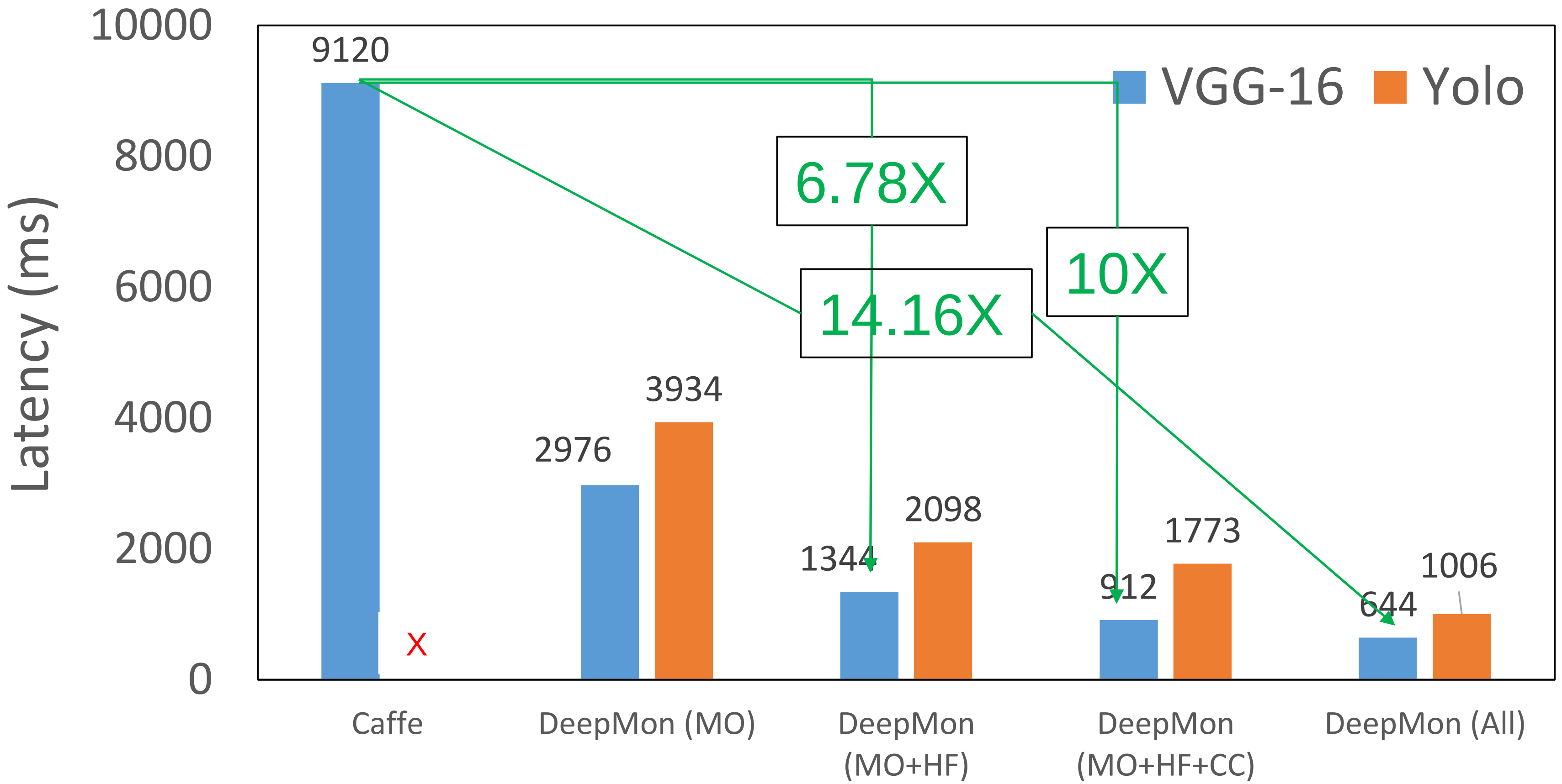


Total computation: $HxWxNxCxDxD$

$HxWxN_1xC$ $HxWxN_2xN_1xDxD$ $HxWxNxN_2$

Speedup: $\frac{NCD^2}{N_1C+N_1N_2D^2+NN_2}$

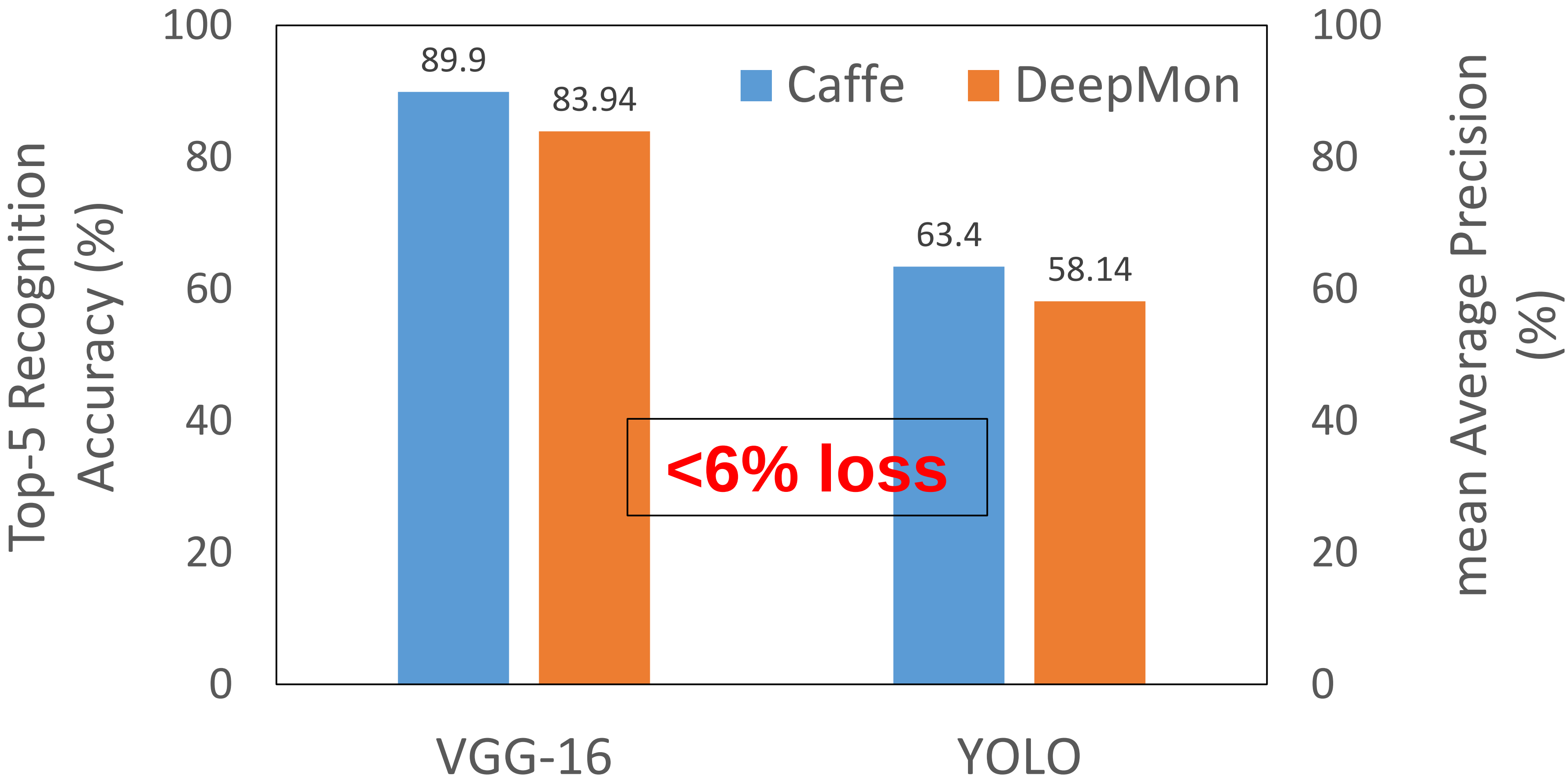
DeepMon Performance: Latency



MO: Memory Opt.   HF: Half Floating point   CC: Convolutional Caching

Dataset: UCF-101 (13K+ short video clips)

DeepMon Performance: Accuracy



All Optimization techniques are applied for DeepMon.
Dataset: (1) ILSVRC2012 for VGG-VeryDeep-16,
(2) the Pascal VOC 2007 for YOLO

Conclusion

- DeepMon is an easy to use framework
  - Supports existing deep learning models
  - Supports commodity mobile devices & OS's
  - Supports various optimizations to reduce latency
    - Memory loading optimizations
    - Convolutional caching
    - Decomposition

- Achieve speedup of 14x over Caffe with minimal accuracy loss (<6%)

- DeepMon implementation in OpenCL/Vulkan
  https://github.com/JC1DA/deepmon