POLITECNICO
MILANO 1863

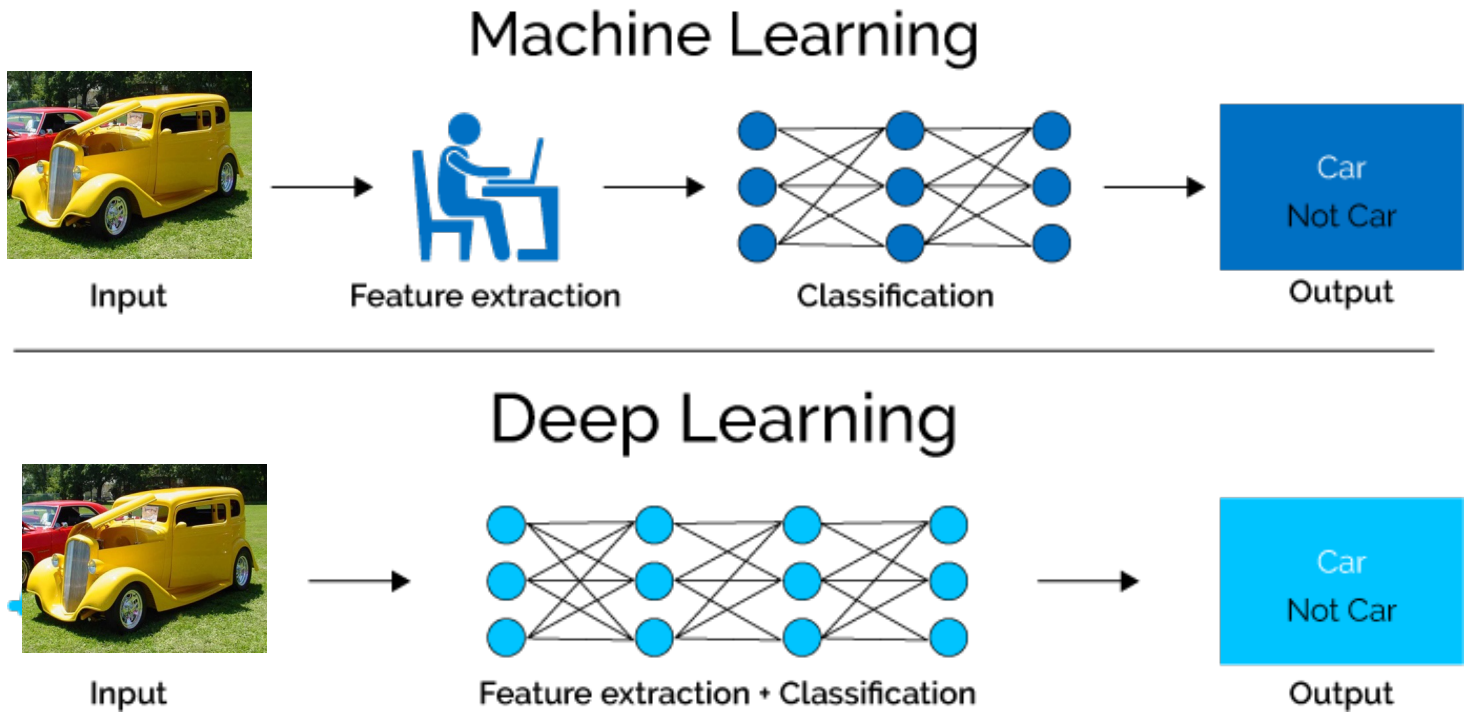
Hardware Architectures for Embedded and Edge AI

Prof Manuel Roveri – manuel.roveri@polimi.it

Lecture 5 – Deep Learning for Embedded and Edge AI

The basics of deep learning

Deep representation of knowledge



How does TinyML work?

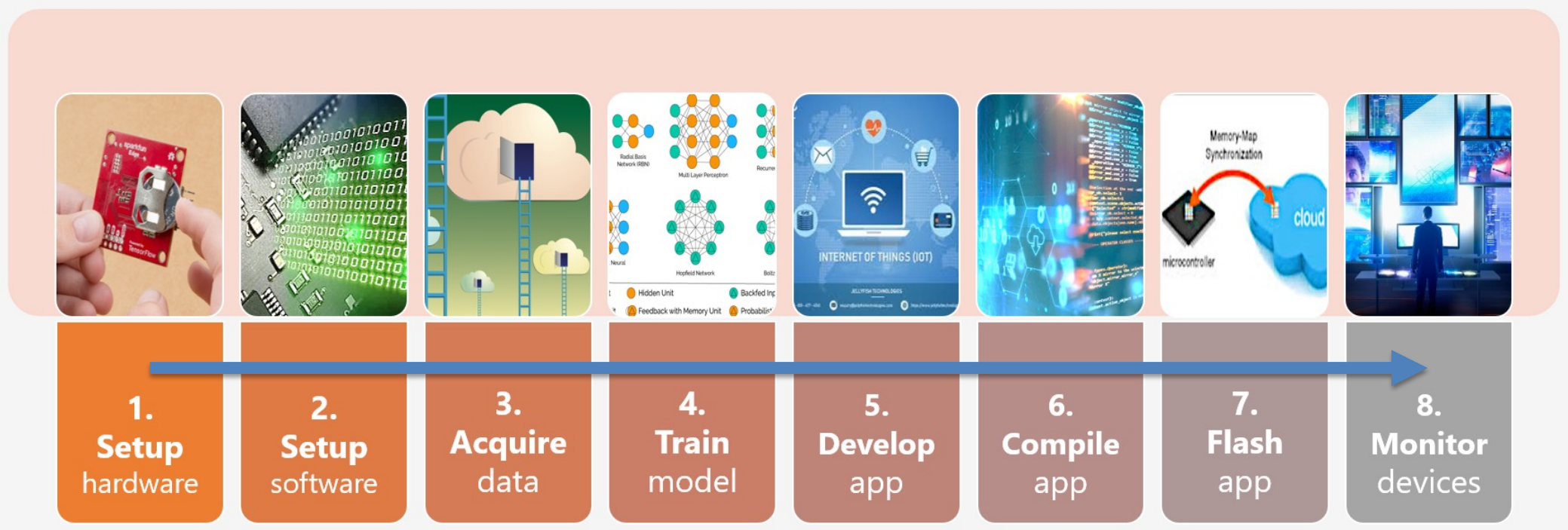


Image from Medium.com “Introducing cAInvas for TinyML devices”, July 2020

How does TinyML work?

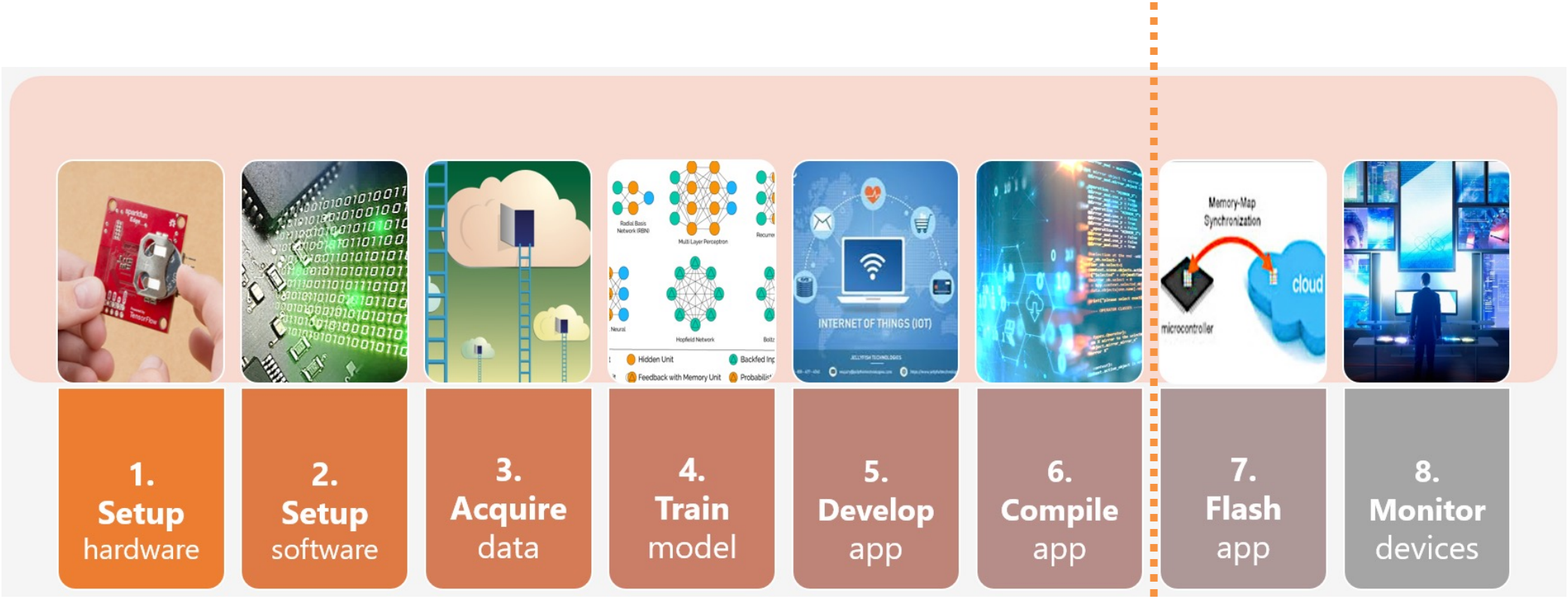
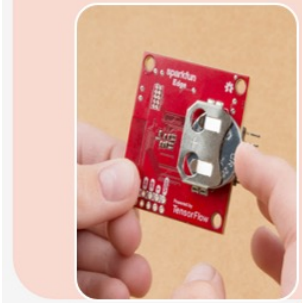


Image from Medium.com “Introducing cAInvas for TinyML devices”, July 2020

Open points?

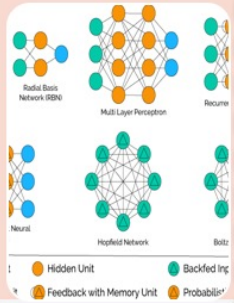
What about
Deep Learning Models ?



1. **Setup** hardware

2. **Setup** software

3. **Acquire** data



4. **Train** model

5. **Develop** app

6. **Compile** app

7. **Flash** app

8. **Monitor** devices

What about the on-device (incremental) learning?

A provocative question …

Is Tiny Deep Learning the new Deep Learning?

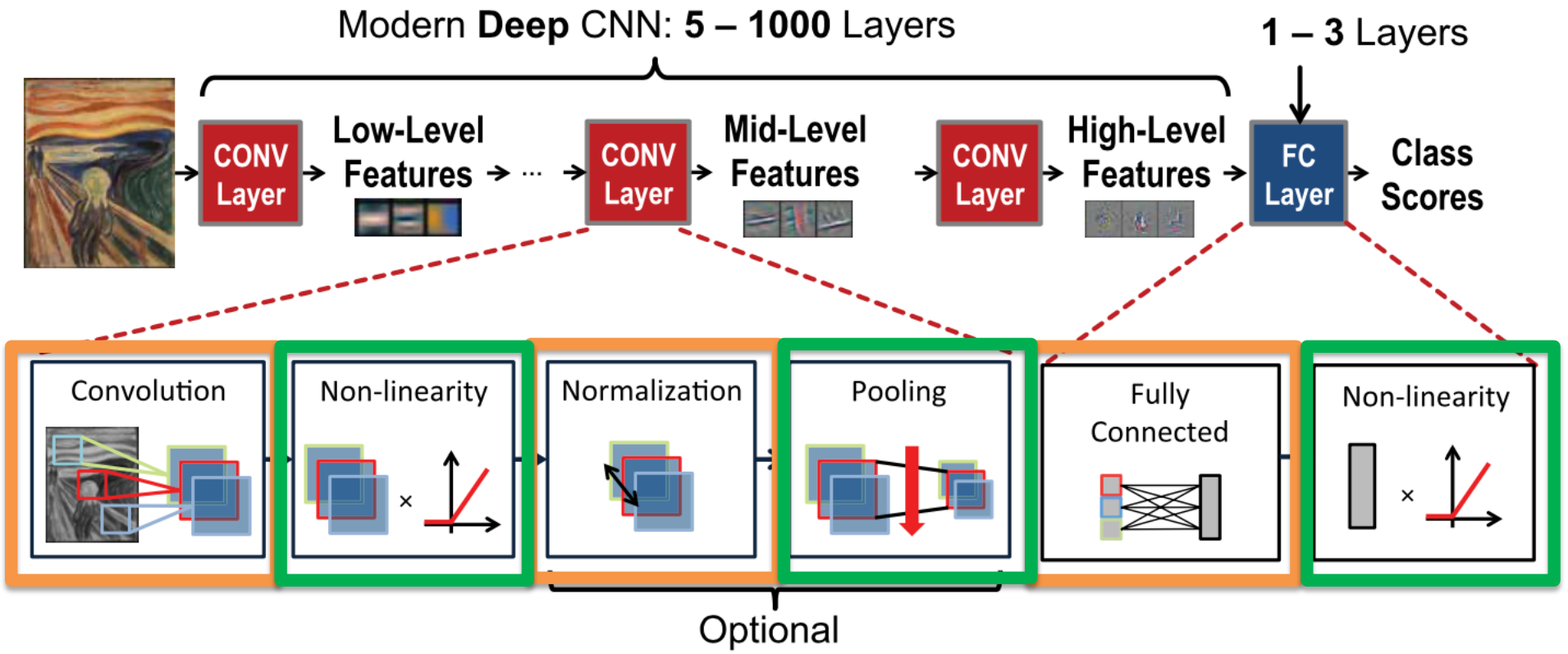
Deep Learning in 2020s

The 70s hardware…

What happened in the 2000s?

Which are the sources of complexity (memory footprint and computational demand) in deep learning models?

Let's focus on Convolutional Neural Networks



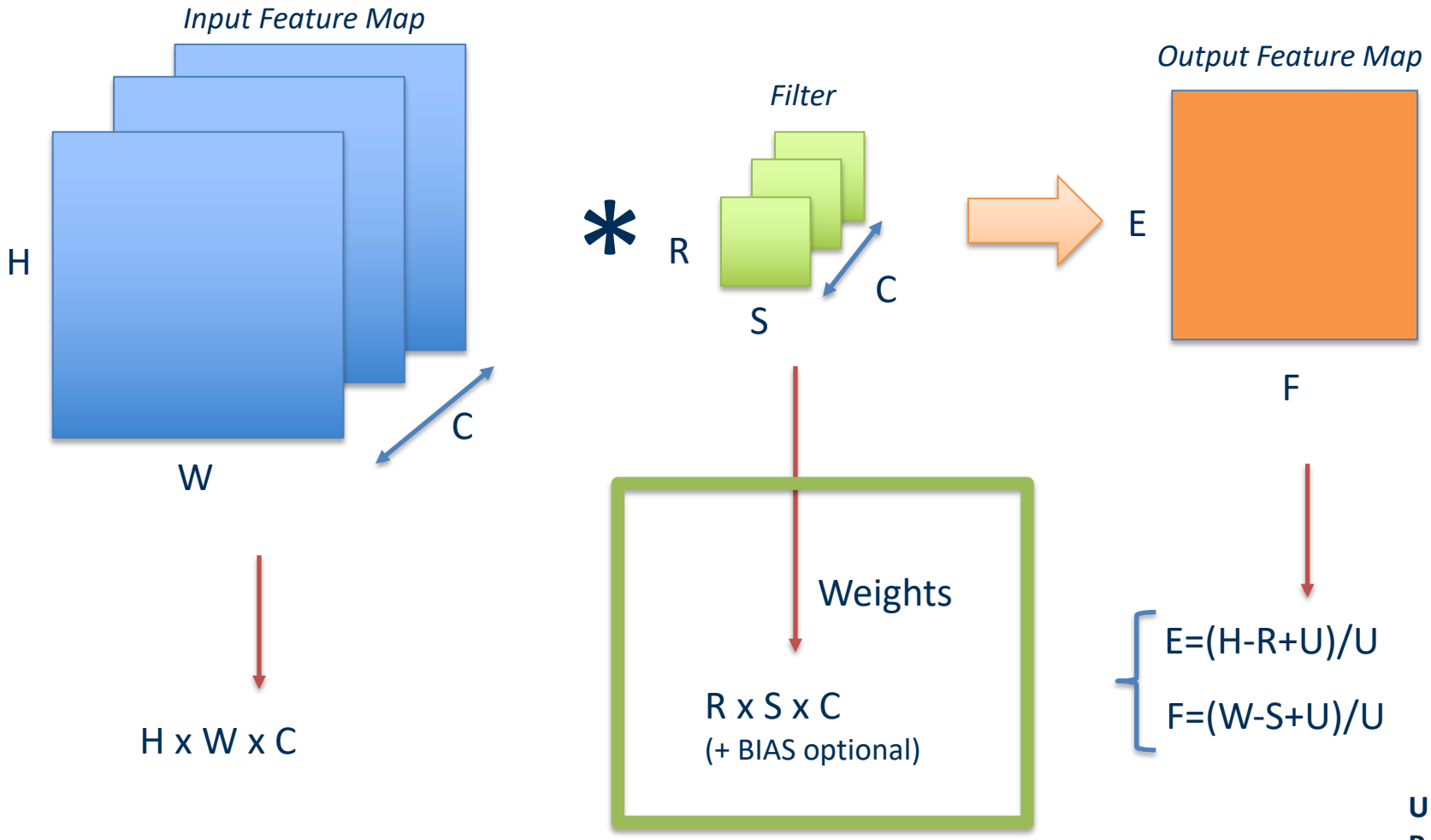
Processing layer with trainable parameters

Processing layer without trainable parameters

CONV and FC

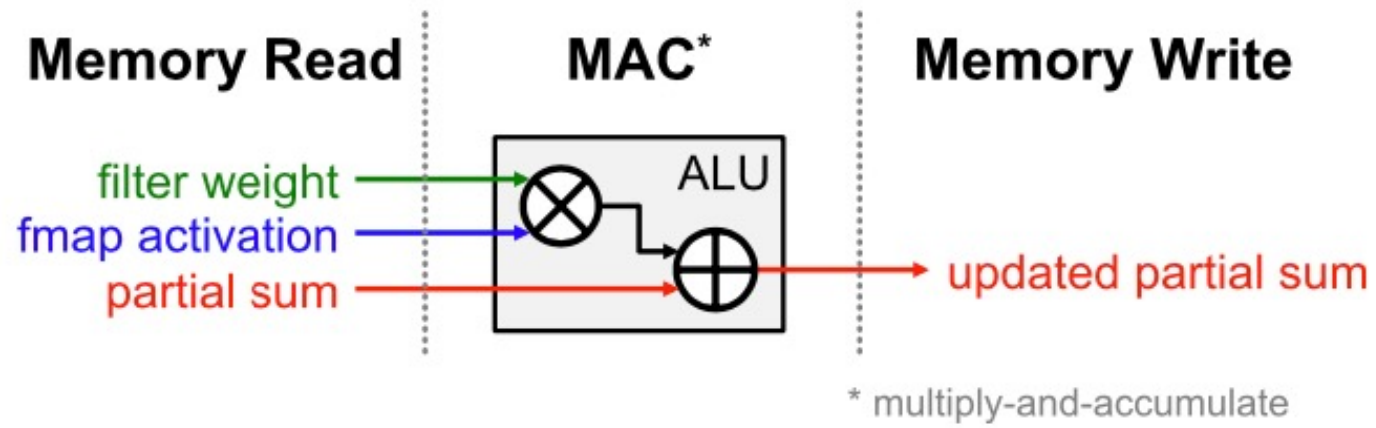
Picture taken from [1]

Convolutional layers (single filter): weights and FMs



What about the number of operations? Let's consider MAC

What is a Multiply And Accumulate (MAC)?



Let's consider a [3 x 3] filter W:

$$o = W[0] * I[0] + W[1] * I[1] + ...+ W[8] * I[8]$$

MAC

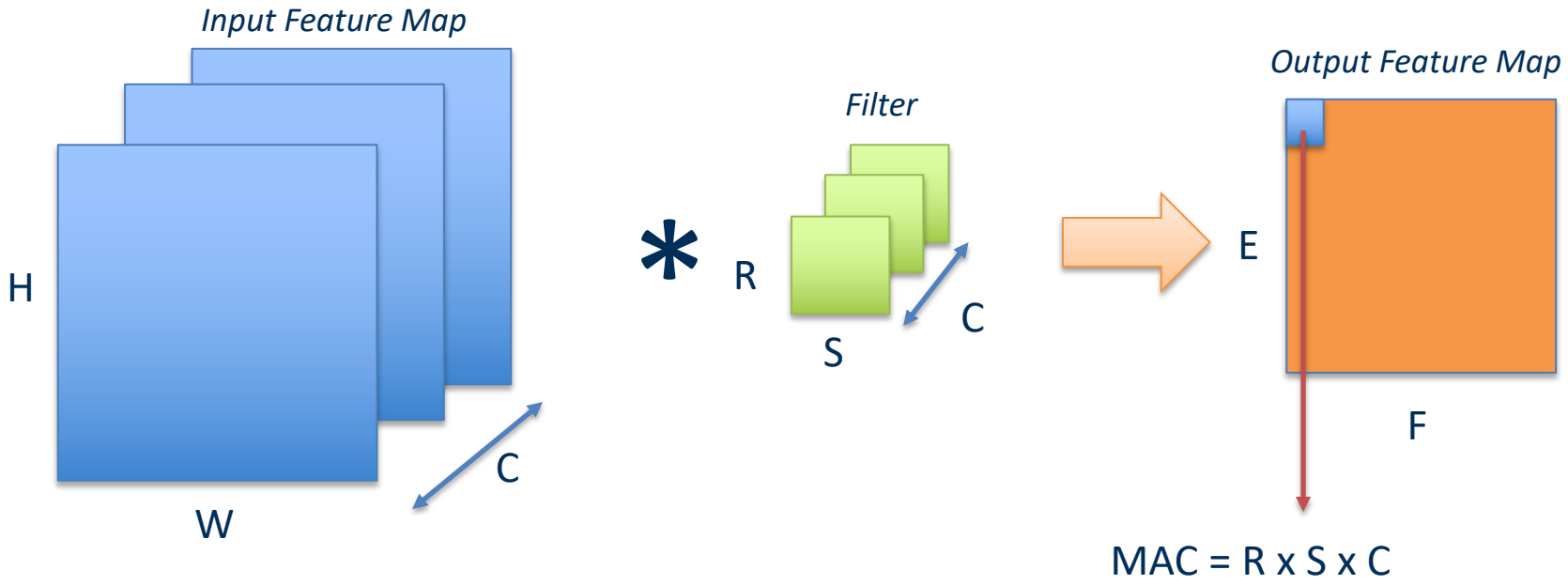
Total = 9 MAC

We have 9+(9-1) FLoating point OPerations (FLOPs) in total

(1MAC =≈ 2FLOPs)

Picture taken from [1]

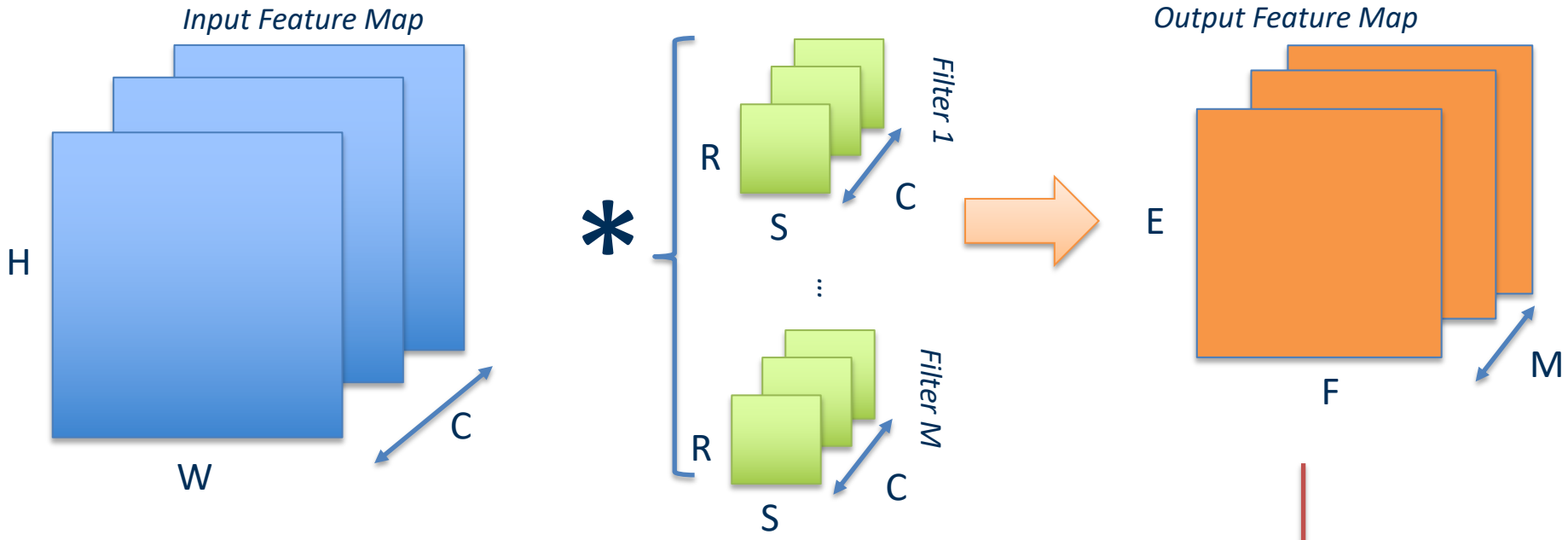
Convolutional layers (single filter): MAC



Total number of MAC operations: E x F x R x S x C

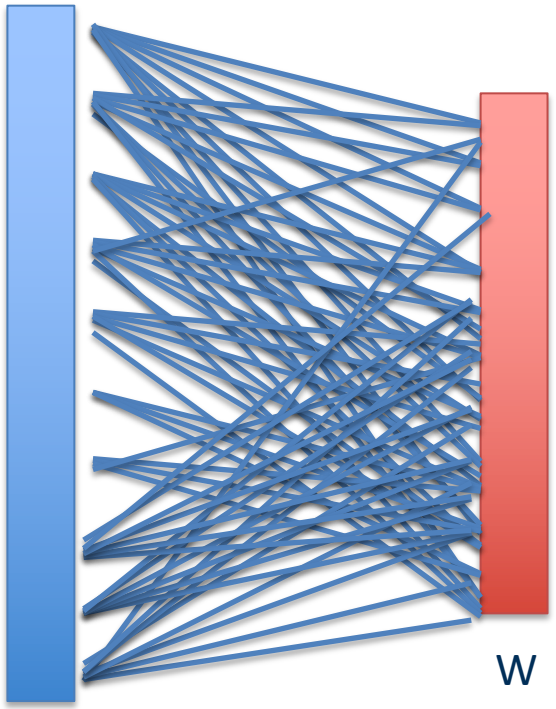
We are here neglecting the bias

Convolutional layers (multiple filters): weights, FPs, MAC



The total number of MAC is E x F x R x S x C x M

Fully Connected Layer: weights and MAC

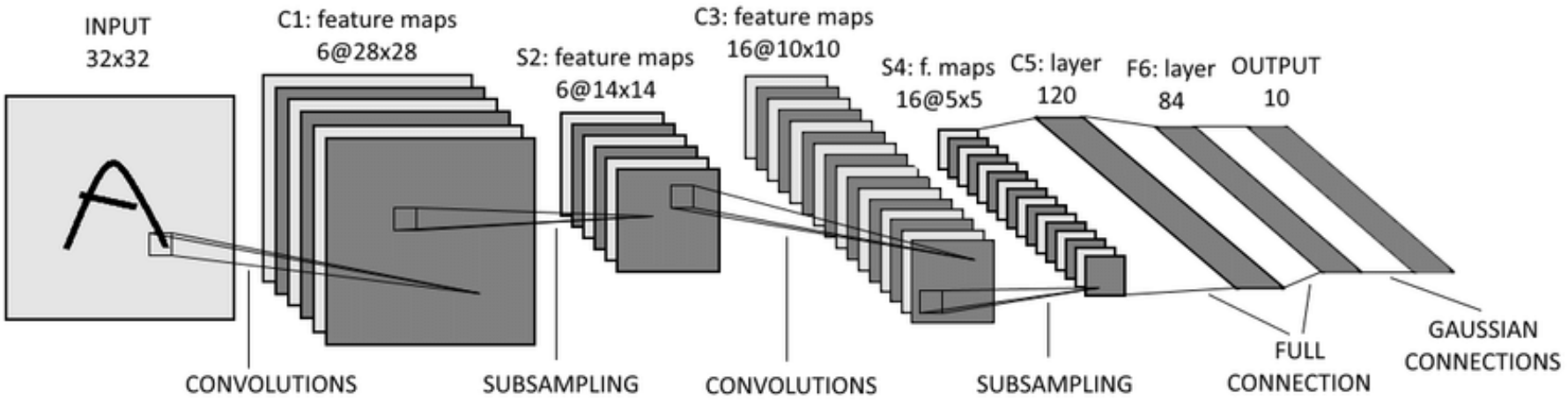


Weights

H x W + W
(bias)

The total number of MAC is H x W

Let's consider a simple CNN: the Le-Net (1989)



Picture taken from [2]

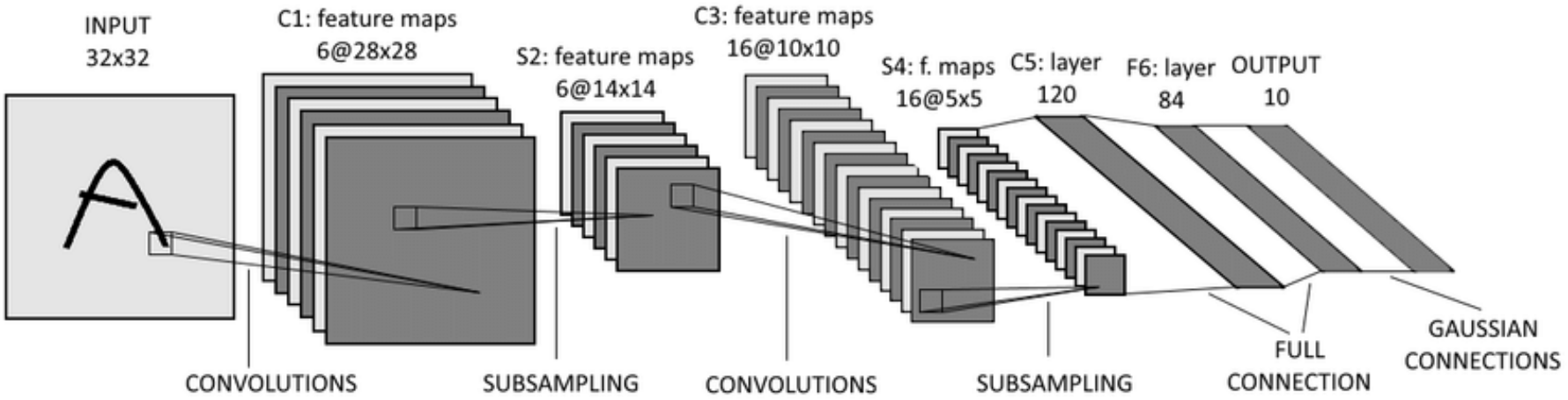
Le-Net: the Tensorflow implementation and summary

```
model = models.Sequential()
model.add(layers.Conv2D(6, 5, activation='tanh',
input_shape=x_train.shape[1:]))
model.add(layers.AveragePooling2D(2))
model.add(layers.Activation('sigmoid'))
model.add(layers.Conv2D(16, 5, activation='tanh'))
model.add(layers.AveragePooling2D(2))
model.add(layers.Activation('sigmoid'))
model.add(layers.Conv2D(120, 5, activation='tanh'))
model.add(layers.Flatten())
model.add(layers.Dense(84, activation='tanh'))
model.add(layers.Dense(10, activation='softmax'))
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 28, 28, 6)         156
_________________________________________________________________
average_pooling2d (AveragePo (None, 14, 14, 6)         0
_________________________________________________________________
activation (Activation)      (None, 14, 14, 6)         0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 10, 10, 16)        2416
_________________________________________________________________
average_pooling2d_1 (Average (None, 5, 5, 16)          0
_________________________________________________________________
activation_1 (Activation)    (None, 5, 5, 16)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 1, 1, 120)         48120
_________________________________________________________________
flatten (Flatten)            (None, 120)               0
_________________________________________________________________
dense (Dense)                (None, 84)                10164
_________________________________________________________________
dense_1 (Dense)              (None, 10)                850
=================================================================
Total params: 61,706
Trainable params: 61,706
Non-trainable params: 0
```

Picture taken from [2]

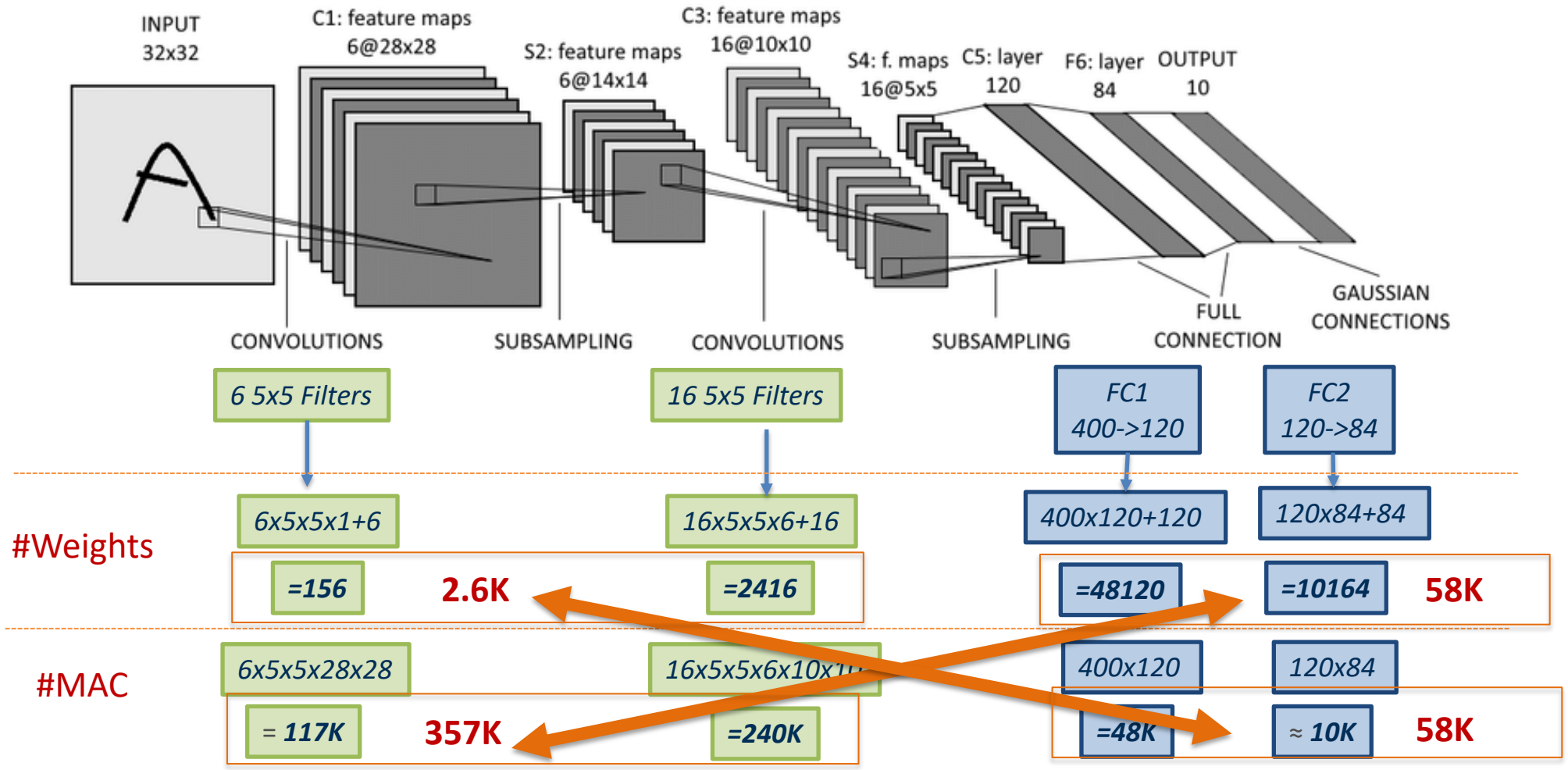
Le-Net: weights and MAC



	6 5x5 Filters	*16 5x5 Filters*	*FC1 400->120*	*FC2 120->84*
#Weights	*6x5x5x1+6*	*16x5x5x6+16*	*400x120+120*	*120x84+84*
	=156	***=2416***	***=48120***	***=10164***
#MAC	*6x5x5x28x28*	*16x5x5x6x10x10*	*400x120*	*120x84*
	= 117K	***=240K***	***=48K***	***≈ 10K***

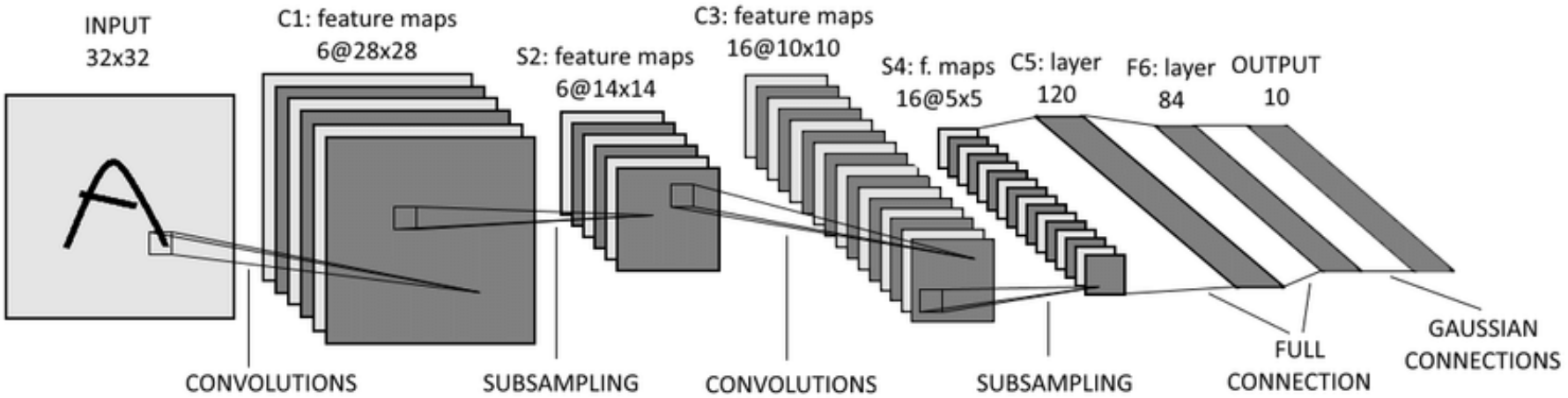
Picture taken from [2]

Le-Net: weights and MAC



Picture taken from [2]

Le-Net: memory occupation of the weights



6 5x5 Filters → *6x5x5x1+6* → *=156*

16 5x5 Filters → *16x5x5x6+16* → *=2416*

2.6K

FC1 400->120 → *400x120+120* → *=48120*

FC2 120->84 → *120x84+84* → *=10164*

58K

Weights

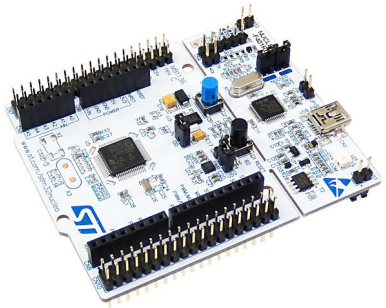
Memory occupation = 60.6K x 4 Byte =≈ 242 KB

Picture taken from [2]

Can we port our LeNet on a IoT unit with 256 KB of RAM?

242KB for the weights

10KB for the firmware

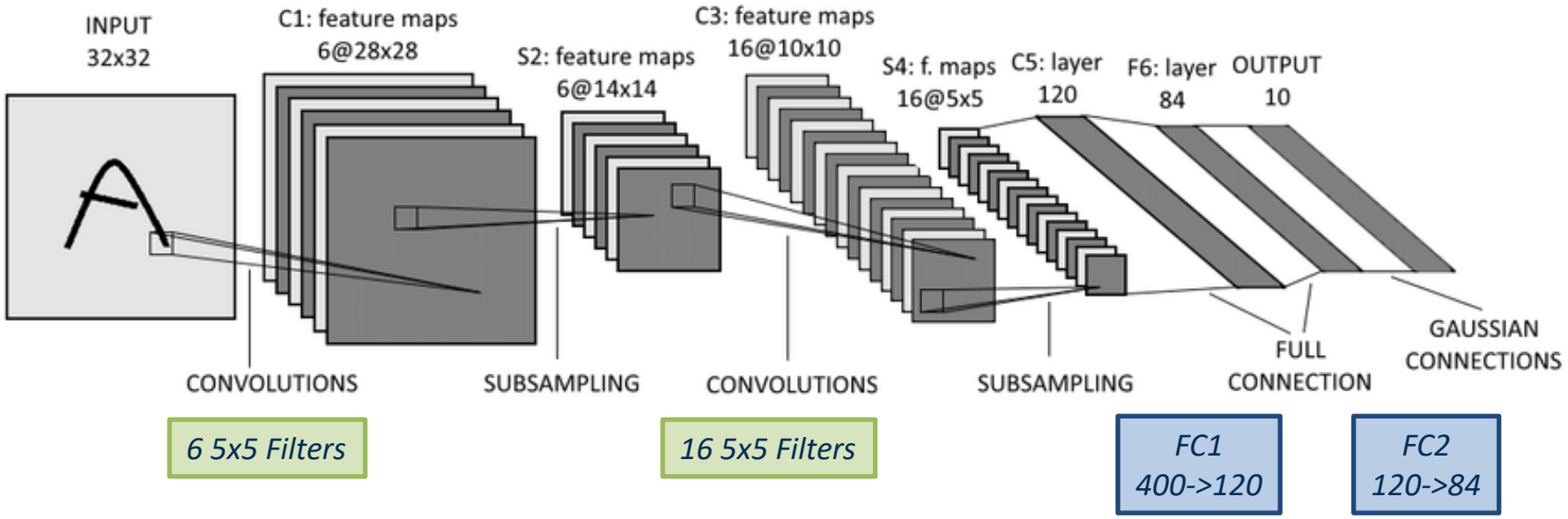


STM32F429[2]	180	512 K to 2056 K	256
STM32F427[2]	180	1024 K to 2056 K	256

What are we missing?

What are we missing?

What about the feature maps ???



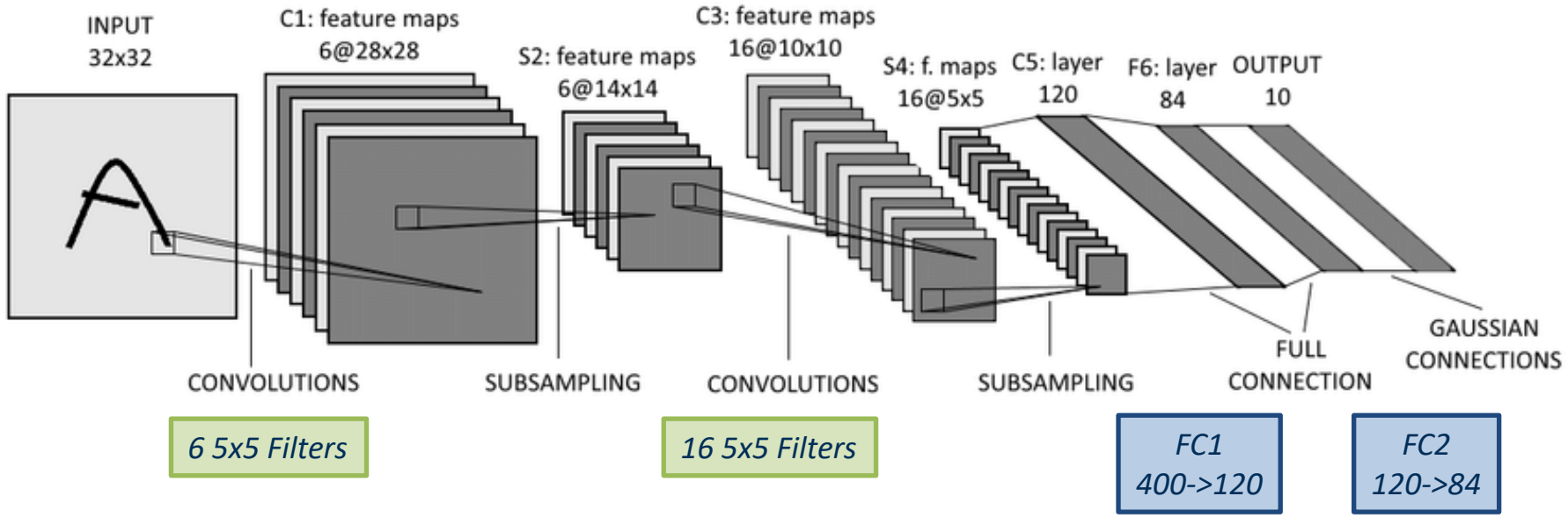
Weight Memory occupation = 60.6K x 4 Byte = 242 KB

Picture taken from [2]

Le-Net: memory occupation of the feature maps

=≈ *37KB*

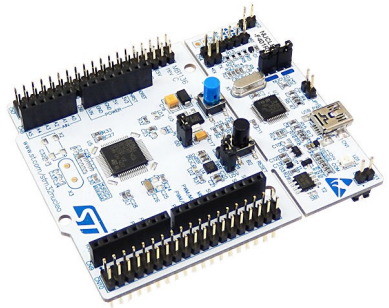
4KB *19KB* *5KB* *6KB* *1.6KB* *0.5KB* *0.3KB* *…KB*



Weight Memory occupation = 60.6K x 4 Byte = 240 KB

Picture taken from [2]

Can we port our LeNet on a IoT unit with 384 KB of RAM?



Product lines	F_{CPU} (MHz)	Flash (Kbytes)	RAM (KB)
STM32F469[2]	180	512 K to 2056 K	384

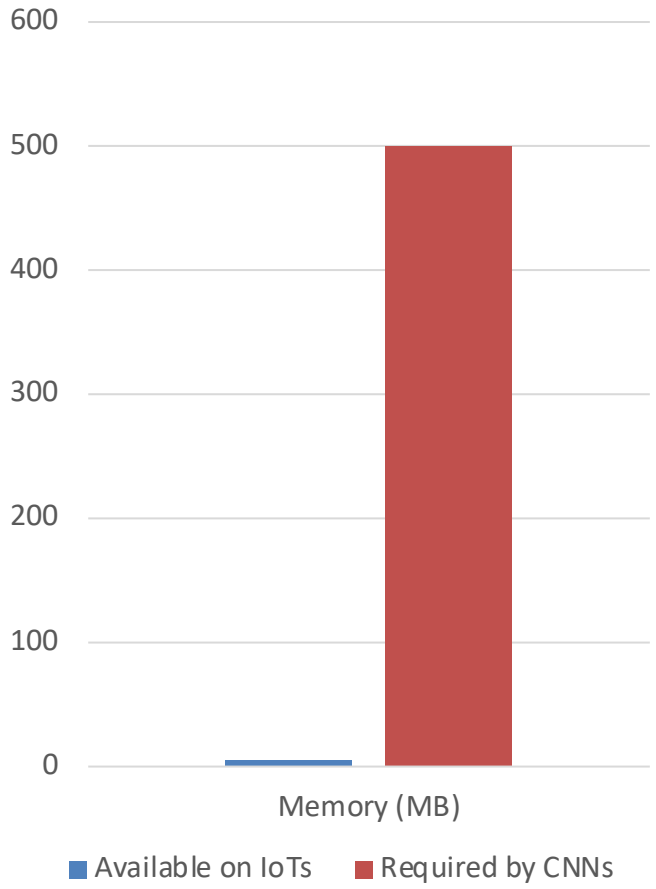
CNNs memory and computational demand

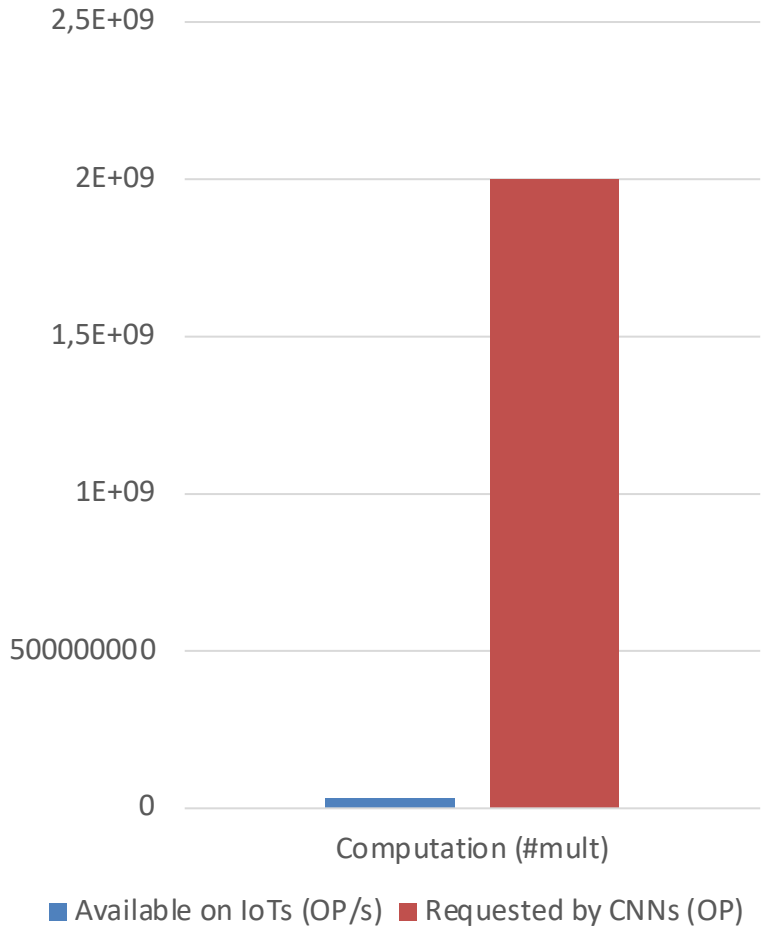
Metrics	AlexNet	Overfeat fast	VGG 16	GoogLeNet v1	ResNet 50
Top-5 error†	16.4	14.2	7.4	6.7	5.3
Top-5 error (single crop)†	19.8	17.0	8.8	10.7	7.0
Input Size	227×227	231×231	224×224	224×224	224×224
# of CONV Layers	5	5	13	57	53
Depth in # of CONV Layers	5	5	13	21	49
Filter Sizes	3,5,11	3,5,11	3	1,3,5,7	1,3,7
# of Channels	3-256	3-1024	3-512	3-832	3-2048
# of Filters	96-384	96-1024	64-512	16-384	64-2048
Stride	1,4	1,4	1	1,2	1,2
Weights	2.3M	16M	14.7M	6.0M	23.5M
MACs	666M	2.67G	15.3G	1.43G	3.86G
# of FC Layers	3	3	3	1	1
Filter Sizes	1,6	1,6,12	1,7	1	1
# of Channels	256-4096	1024-4096	512-4096	1024	2048
# of Filters	1000-4096	1000-4096	1000-4096	1000	1000
Weights	58.6M	130M	124M	1M	2M
MACs	58.6M	130M	124M	1M	2M
Total Weights	61M	146M	138M	7M	25.5M
Total MACs	724M	2.8G	15.5G	1.43G	3.9G
Pretrained Model Website	[57, 58]	n/a	[57–59]	[57–59]	[57–59]

†Accuracy is Measured Based on Top-5 Error on ImageNet [14].

Picture taken from [1]

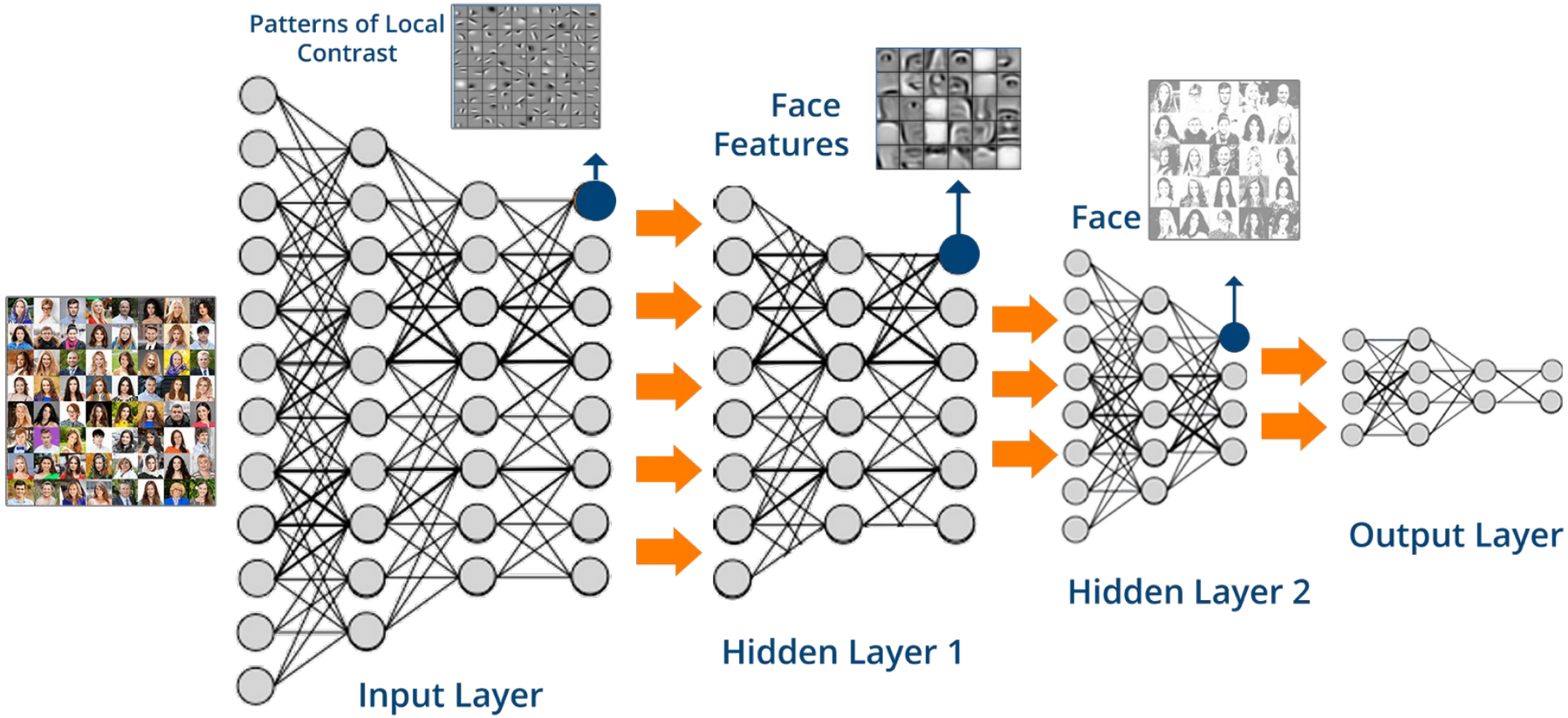
What about available memory and computation in IoT units?





How can we match these extremes?

Approximate Deep Learning for IoT: Tiny Deep Learning



Approximate Deep Learning for IoT: Tiny Deep Learning

Re-design of the CNN architecture → Approximate Computing → Embedded System Code Optimization