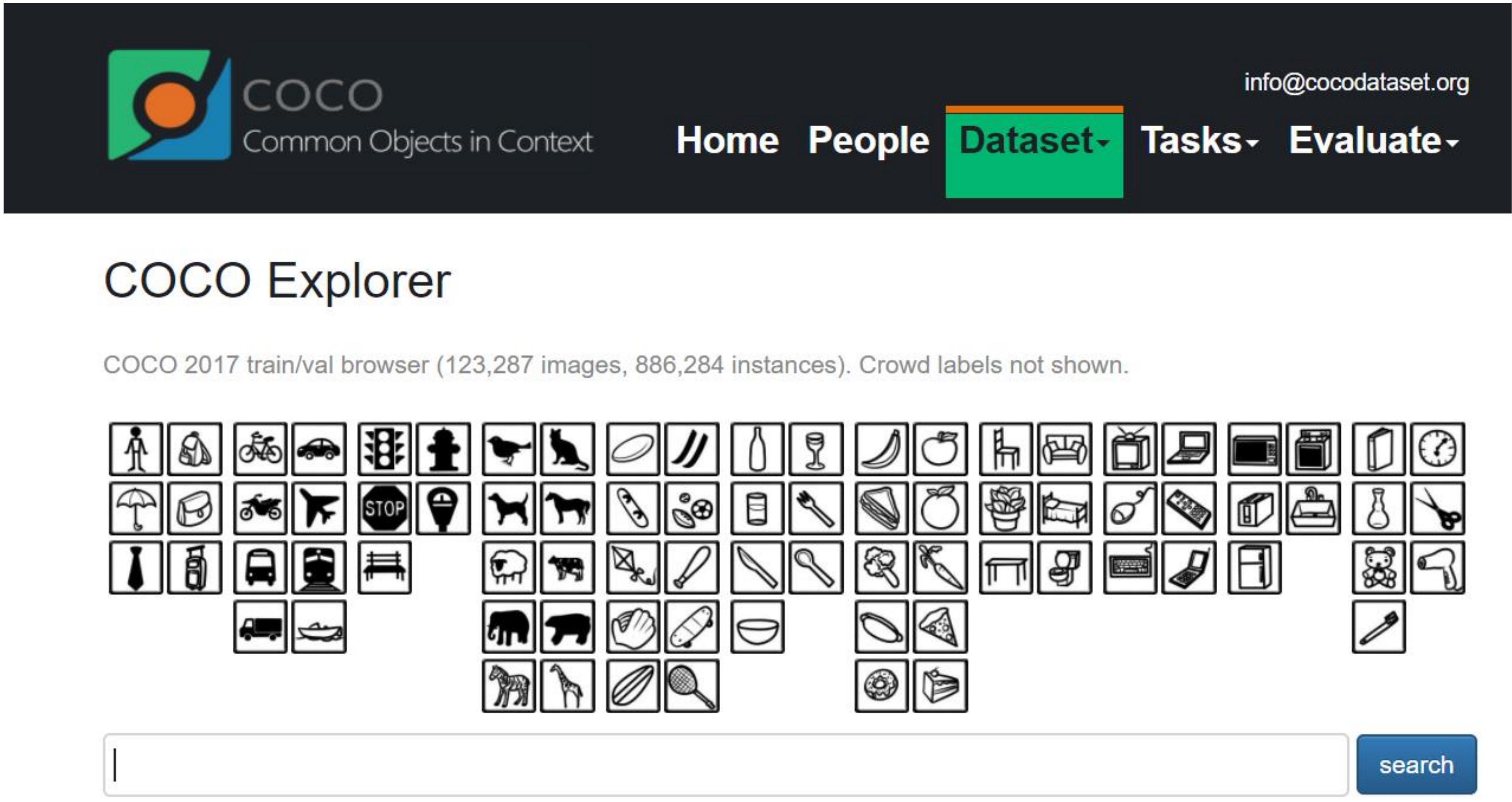
Segmentation
Low-cost/mobile networks
Newly emerging methods in deep learning
Deployment

Segmentation – Microsoft Coco



COCO Explorer

COCO 2017 train/val browser (123,287 images, 886,284 instances). Crowd labels not shown.

http://cocodataset.org/#explore

Segmentation

– Segmentation task is different in the sense that it requires predicting a class ***for each pixel*** of the input image.

– Classification needs to understand ***what*** is in the input (namely, the context).

– However, in order to predict what is in the input for each pixel, segmentation needs to recover not only *what* is in the input, but also ***where***.

Segmentation

– Fully Convolutional Networks (FCNs) are made up of locally connected layers and no dense (fully-connected) layer is used.

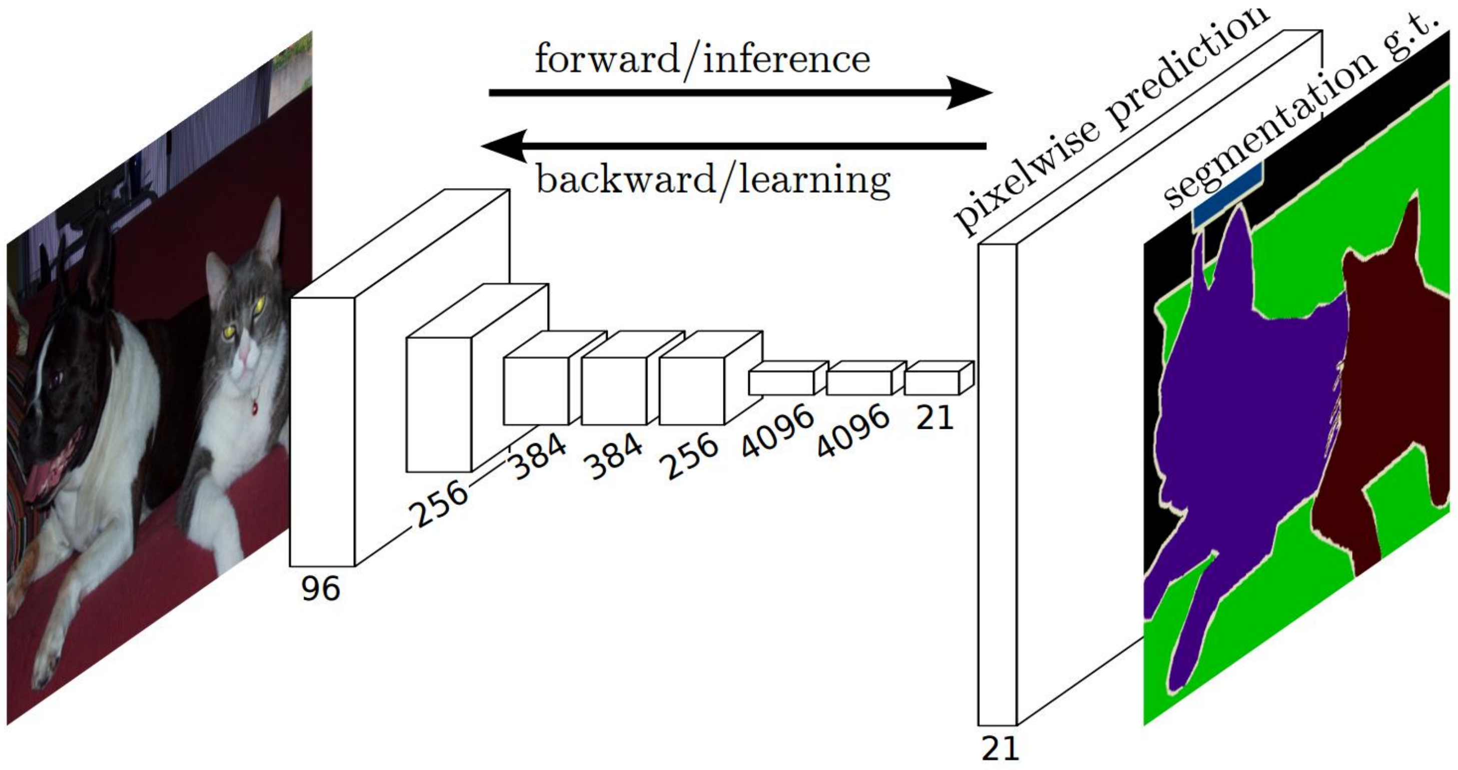
– A standard classification network such as VGG-16 can be converted to FCN using 1x1 convolutions.

– This network produces class presence heat map in low resolution.

– The upsampling of these low resolution semantic feature maps is done using transposed convolutions (initialized with bilinear interpolation filters).

Long, J., Shelhamer, E. and Darrell, T., 2015. Fully convolutional networks for semantic segmentation. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 3431-3440).

Segmentation



To obtain a segmentation map (output), segmentation networks usually have 2 parts:

- **Downsampling path**: capture semantic/contextual information - extract and interpret the context (*what*).
- **Upsampling path**: recover spatial information- to enable precise localization (*where*).
- To fully recover the fine-grained spatial information lost in the pooling layers, skip connections are used.

Segmentation

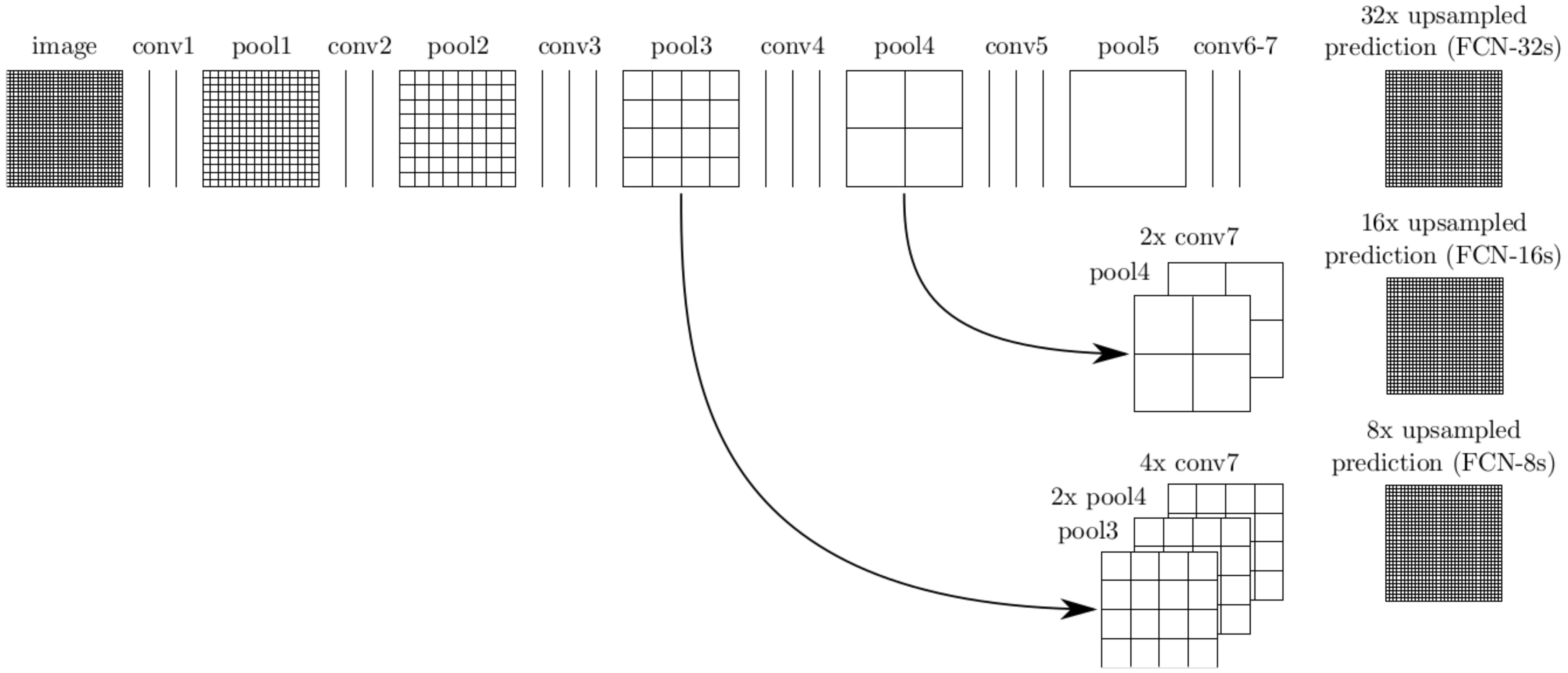
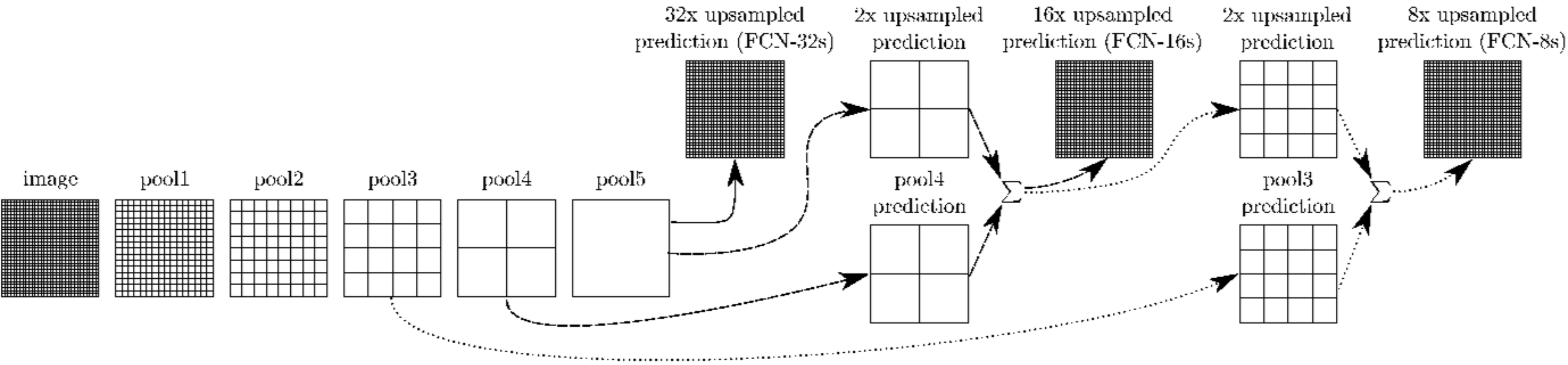


image
conv1
pool1
conv2
pool2
conv3
pool3
conv4
pool4
conv5
pool5
conv6-7
32x upsampled prediction (FCN-32s)
2x conv7
pool4
16x upsampled prediction (FCN-16s)
4x conv7
2x pool4
pool3
8x upsampled prediction (FCN-8s)

There are three variants: FCN-32, FCN-16 and FCN-8 (different spatial precision levels). Convolutional layers are represented as vertical lines between pooling layers, which explicitly show the relative size of the feature maps.

Long, J., Shelhamer, E. and Darrell, T., 2015. Fully convolutional networks for semantic segmentation. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 3431-3440).

Segmentation



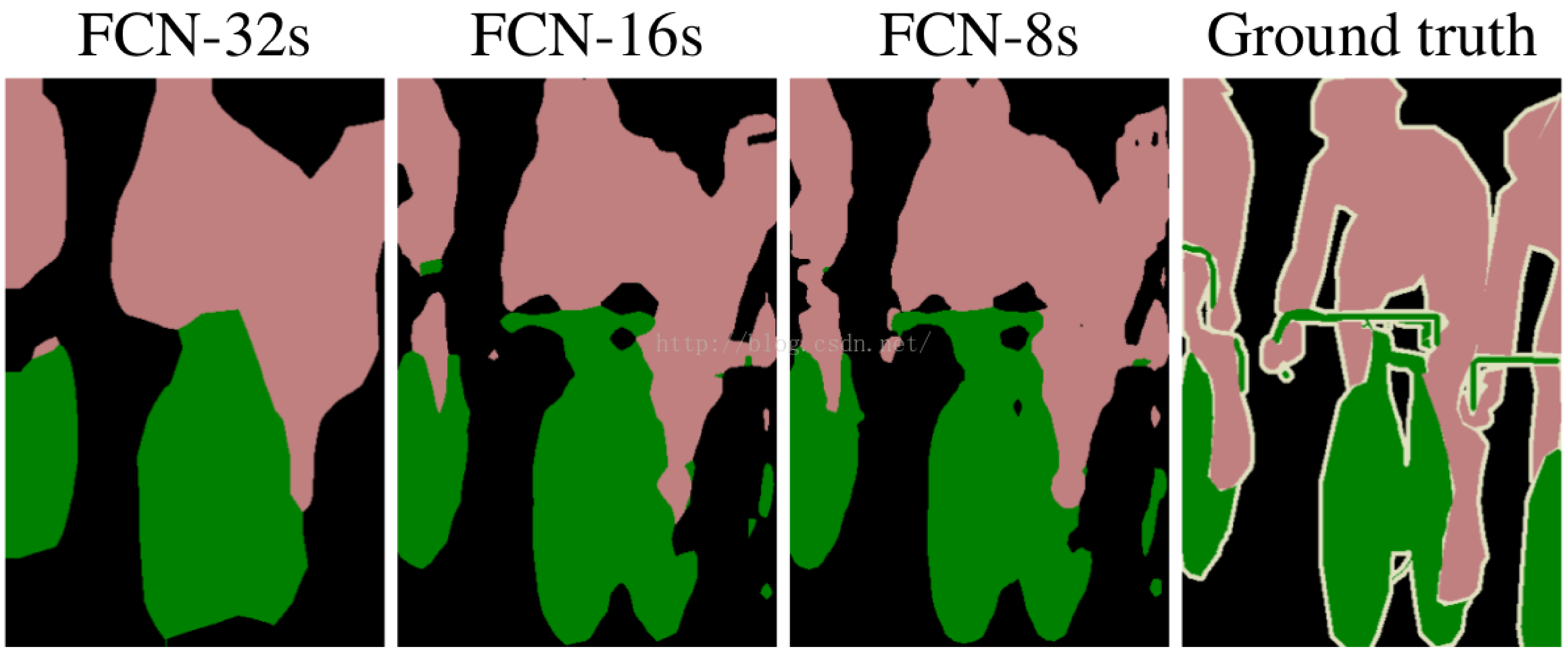
Variants share the same downsampling path, but differ in their respective upsampling paths.

FCN-32 : Directly produces the segmentation map from *conv7*, by using a transposed convolution layer with stride 32.

FCN-16 : Sums the 2x upsampled prediction from *conv7* (using a transposed convolution with stride 2) with *pool4* and then produces the segmentation map, by using a transposed convolution layer with stride 16 on top of that.

FCN-8 : Sums the 2x upsampled *conv7* (with a stride 2 transposed convolution) with *pool4*, upsamples them with a stride 2 transposed convolution and sums them with *pool3*, and applies a transposed convolution layer with stride 8 on the resulting feature maps to obtain the segmentation map.

Segmentation



- The upsampling paths of the FCN variants are different, since they use different skip connection layers and strides for the last convolution, yielding different segmentations.
- Combining layers that have different precision helps retrieving fine-grained spatial information, as well as coarse contextual information.

Spatial Pyramid Pooling

- Spatial Pyramid Pooling (SPP) is a technique which allows Convolutional Neural Networks (CNNs) to use input images of any size, not fixed size images as most architectures do.
  - However, there is a lower bound for the size of the input image

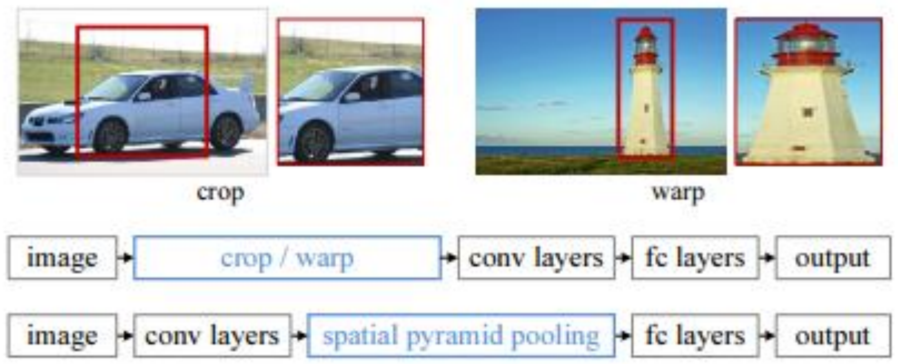


Figure 1: Top: cropping or warping to fit a fixed size. Middle: a conventional CNN. Bottom: our spatial pyramid pooling network structure.

Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition, Kaiming He and Xiangyu Zhang and Shaoqing Ren and Jian Sun, arXiv e-Print archive - 2014

Spatial Pyramid Pooling

- Convolutional layers operate on any size, but fully connected layers need fixed-size inputs, to solve this problem:
    - Add a new SPP layer on top of the last convolutional layer, before the fully connected layer
    - Use an approach similar to bag of words (BoW), but maintain the spatial information. The BoW approach is used for text classification, where the order of the words is discarded and only the number of occurrences is kept.
    - The SPP layer operates on each feature map independently.
    - The output of the SPP layer is of dimension k x M, where k is the number of feature maps the SPP layer got as input and M is the number of bins.

Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition, Kaiming He and Xiangyu Zhang and Shaoqing Ren and Jian Sun, arXiv e-Print archive - 2014

Spatial Pyramid Pooling

- Example: We could use spatial pyramid pooling with 21 bins:
- 1 bin which is the max of the complete feature map
- 4 bins which divide the image into 4 regions of equal size (depending on the input size) and rectangular shape. Each bin gets the max of its region.
- 16 bins which divide the image into 16 regions of equal size (depending on the input size) and rectangular shape. Each bin gets the max of its region.

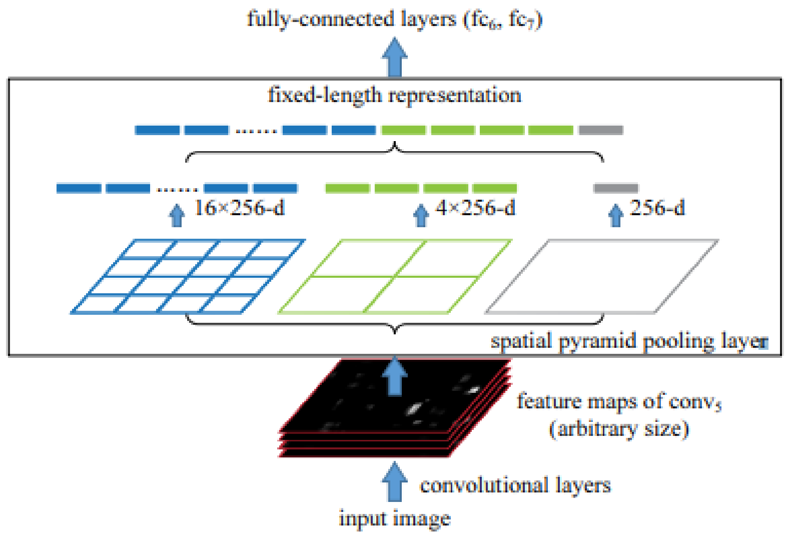
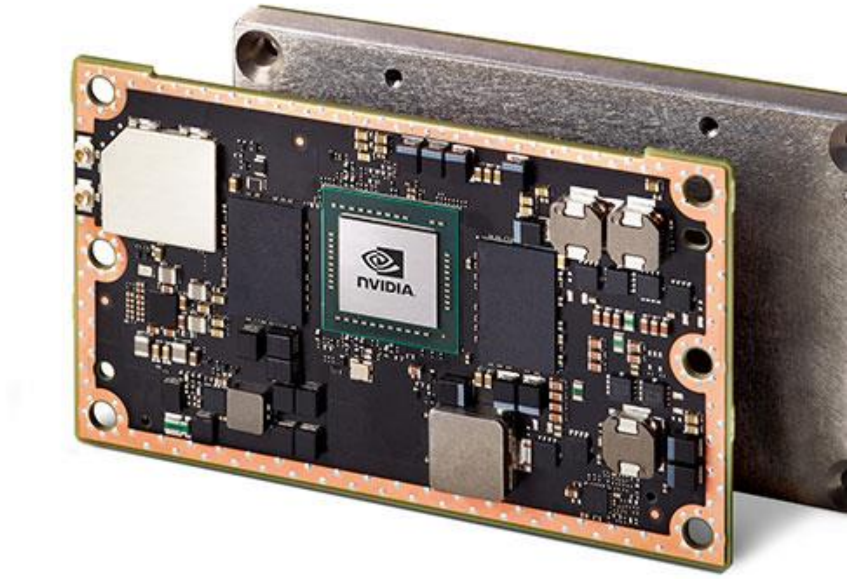

fully-connected layers (fc_6, fc_7)
fixed-length representation
16×256-d
4×256-d
256-d
spatial pyramid pooling layer
feature maps of $conv_5$ (arbitrary size)
convolutional layers
input image

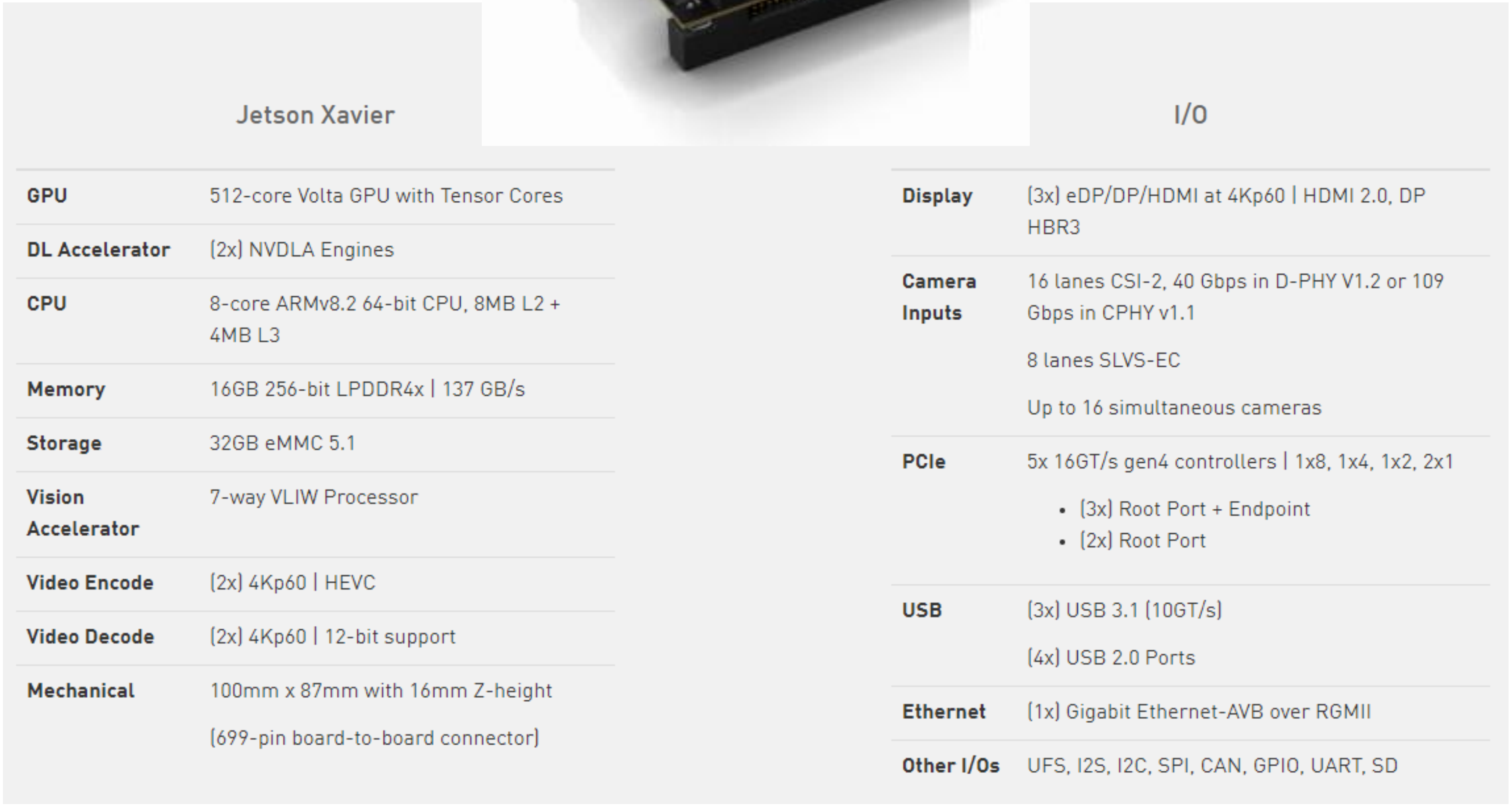
Figure 3: A network structure with a **spatial pyramid pooling layer**. Here 256 is the filter number of the $conv_5$ layer, and $conv_5$ is the last convolutional layer.

CNNs on Mobile Devices : NVIDIA Jetson



	Jetson TX2	Jetson TX1
GPU	NVIDIA Pascal™, 256 CUDA cores	NVIDIA Maxwell ™, 256 CUDA cores
CPU	HMP Dual Denver 2/2 MB L2 + Quad ARM® A57/2 MB L2	Quad ARM® A57/2 MB L2
Video	4K x 2K 60 Hz Encode (HEVC) 4K x 2K 60 Hz Decode (12-Bit Support)	4K x 2K 30 Hz Encode (HEVC) 4K x 2K 60 Hz Decode (10-Bit Support)
Memory	8 GB 128 bit LPDDR4 59.7 GB/s	4 GB 64 bit LPDDR4 25.6 GB/s
Display	2x DSI, 2x DP 1.2 / HDMI 2.0 / eDP 1.4	2x DSI, 1x eDP 1.4 / DP 1.2 / HDMI
CSI	Up to 6 Cameras (2 Lane) CSI2 D-PHY 1.2 (2.5 Gbps/Lane)	Up to 6 Cameras (2 Lane) CSI2 D-PHY 1.1 (1.5 Gbps/Lane)
PCIE	Gen 2 | 1x4 + 1x1 OR 2x1 + 1x2	Gen 2 | 1x4 + 1x1
Data Storage	32 GB eMMC, SDIO, SATA	16 GB eMMC, SDIO, SATA
Other	CAN, UART, SPI, I2C, I2S, GPIOs	UART, SPI, I2C, I2S, GPIOs
USB	USB 3.0 + USB 2.0	
Connectivity	1 Gigabit Ethernet, 802.11ac WLAN, Bluetooth	
Mechanical	50 mm x 87 mm (400-Pin Compatible Board-to-Board Connector)	

Embedded Devices: Jetson Xavier

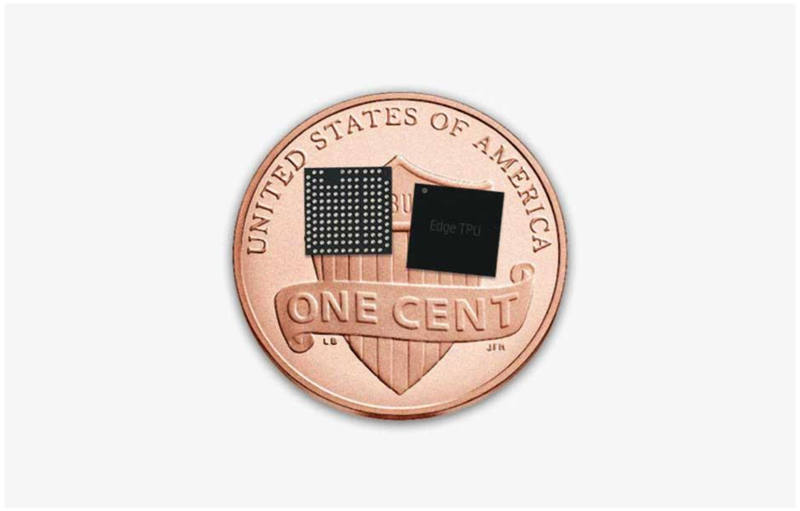


Jetson Xavier	
GPU	512-core Volta GPU with Tensor Cores
DL Accelerator	(2x) NVDLA Engines
CPU	8-core ARMv8.2 64-bit CPU, 8MB L2 + 4MB L3
Memory	16GB 256-bit LPDDR4x | 137 GB/s
Storage	32GB eMMC 5.1
Vision Accelerator	7-way VLIW Processor
Video Encode	(2x) 4Kp60 | HEVC
Video Decode	(2x) 4Kp60 | 12-bit support
Mechanical	100mm x 87mm with 16mm Z-height (699-pin board-to-board connector)

I/O	
Display	(3x) eDP/DP/HDMI at 4Kp60 | HDMI 2.0, DP HBR3
Camera Inputs	16 lanes CSI-2, 40 Gbps in D-PHY V1.2 or 109 Gbps in CPHY v1.1 8 lanes SLVS-EC Up to 16 simultaneous cameras
PCIe	5x 16GT/s gen4 controllers | 1x8, 1x4, 1x2, 2x1 • (3x) Root Port + Endpoint • (2x) Root Port
USB	(3x) USB 3.1 (10GT/s) (4x) USB 2.0 Ports
Ethernet	(1x) Gigabit Ethernet-AVB over RGMII
Other I/Os	UFS, I2S, I2C, SPI, CAN, GPIO, UART, SD

CNNs on Mobile Devices – Edge TPU

- Edge TPU: a small ASIC designed by Google that provides high performance ML inferencing for low-power devices.
- It can execute state-of-the-art mobile vision models such as MobileNet V2 at 100+ fps, in a power efficient manner.
- Supports Tensorflow Lite



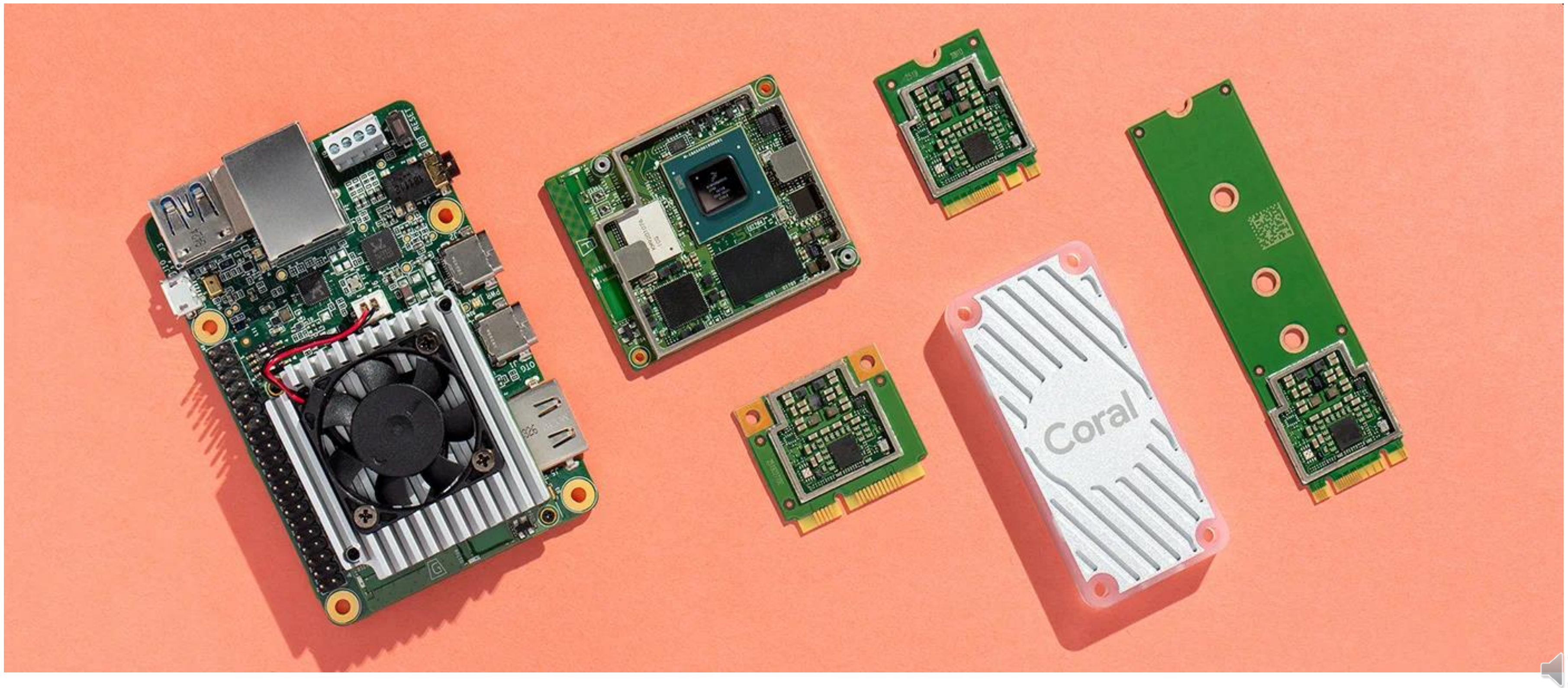
Two Edge TPU chips on the head of a US penny

USB Accelerator with Edge TPU

https://coral.withgoogle.com/tutorials/edgetpu-faq/

Edge TPU – Coral Toolkit

- Coral is a complete toolkit to build products with local AI.
- Prototyping devices include a single-board computer and USB accessory
- Production-ready devices include a system-on-module and PCIe module.



MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications

– Designed for mobile and embedded vision applications.

– MobileNets are based on a streamlined architecture that uses depthwise separable convolutions to build light weight deep neural networks.

– Introduces two simple global hyperparameters that efficiently trade off between latency and accuracy.

– These hyper-parameters allow the model builder to choose the right sized model for their application based on the constraints of the problem.

MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications

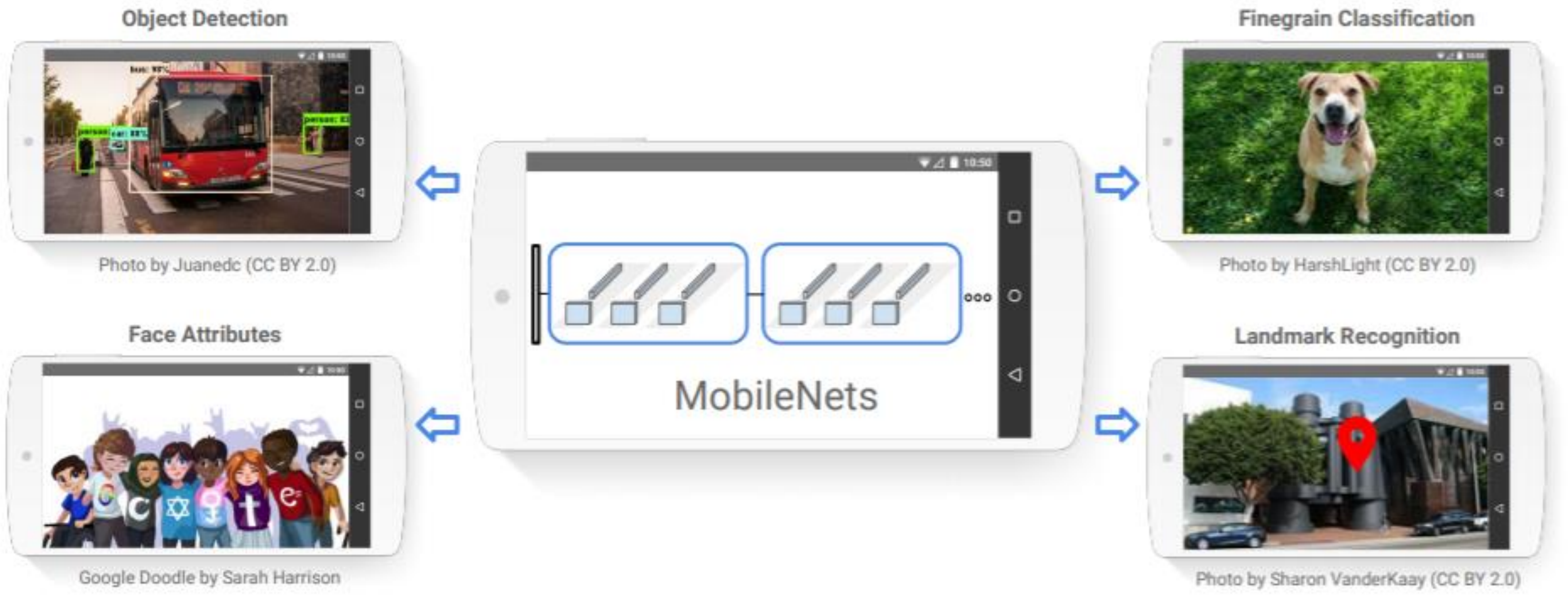
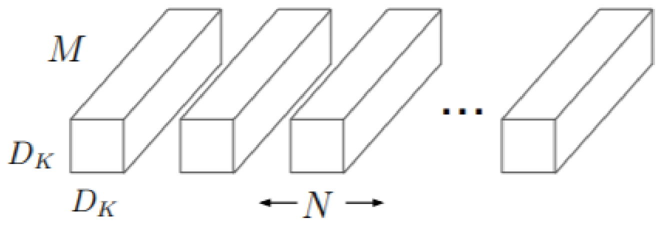

Object Detection
Photo by Juanedc (CC BY 2.0)
Face Attributes
Google Doodle by Sarah Harrison
MobileNets
Finegrain Classification
Photo by HarshLight (CC BY 2.0)
Landmark Recognition
Photo by Sharon VanderKaay (CC BY 2.0)

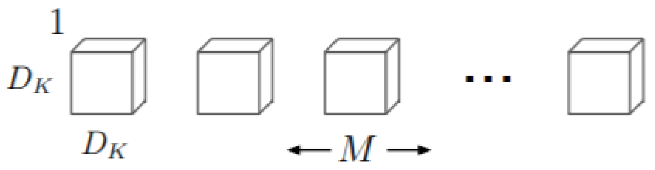
Figure 1. MobileNet models can be applied to various recognition tasks for efficient on device intelligence.

Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M. and Adam, H., 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*.

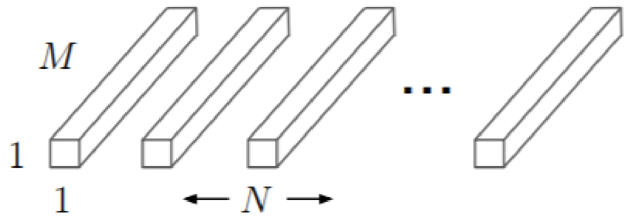
MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications



(a) Standard Convolution Filters



(b) Depthwise Convolutional Filters



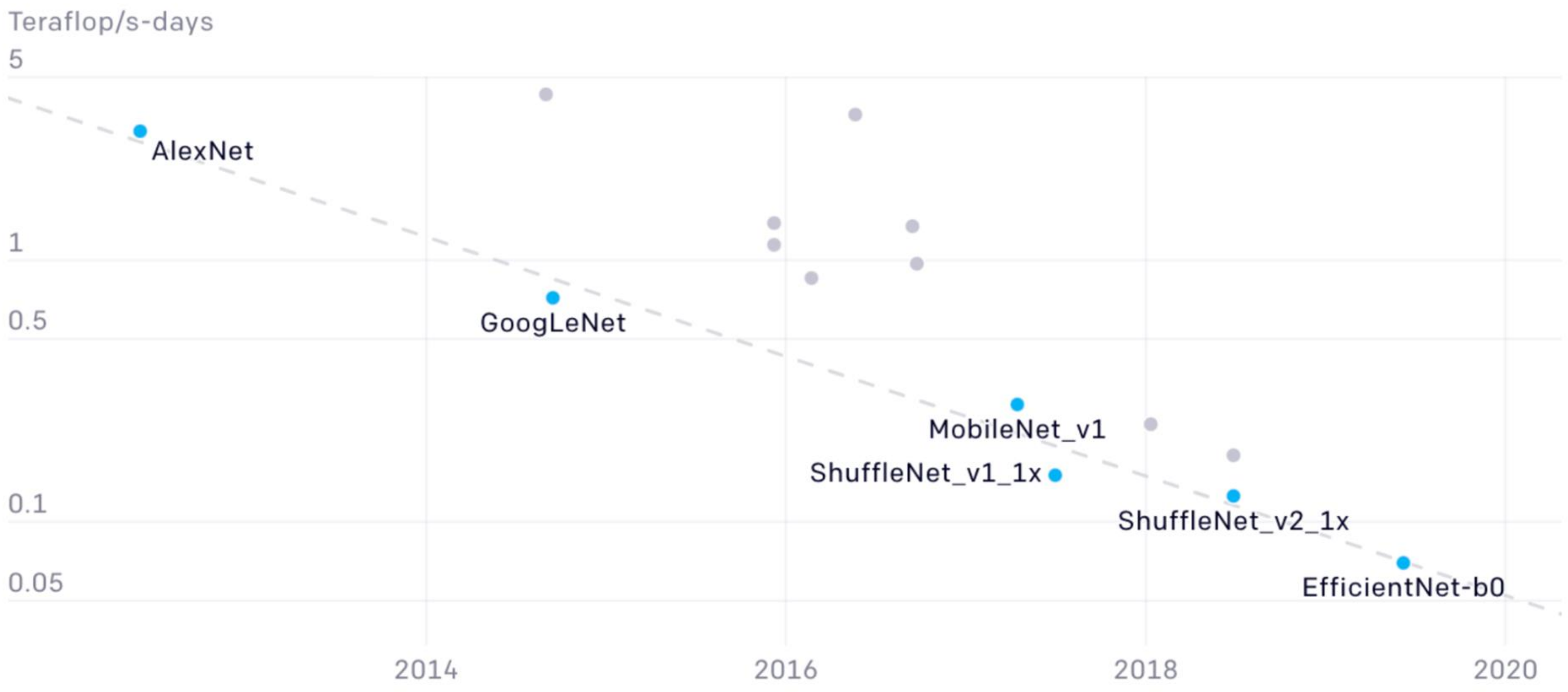
(c) 1×1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

The standard convolutional filters in (a) are replaced by two layers: depthwise convolution in (b) and pointwise convolution in (c) to build a depthwise separable filter.

Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M. and Adam, H., 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*.

EfficientNet

44x less compute required to get to AlexNet performance 7 years later



Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding

“Our method first prunes the network by learning only the important connections. Next, we quantize the weights to enforce weight sharing, finally, we apply Huffman coding. After the first two steps we retrain the network to fine tune the remaining connections and the quantized centroids
Reduces the storage requirement of neural networks by 35x to 49x without affecting their accuracy.”

https://arxiv.org/abs/1510.00149

ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design

“Currently, the neural network architecture design is mostly guided by the indirect metric of computation complexity, i.e., FLOPs. However, the direct metric, e.g., speed, also depends on the other factors such as memory access cost and platform characterics. Thus, this work proposes to evaluate the direct metric on the target platform, beyond only considering FLOPs. Based on a series of controlled experiments, this work derives several practical guidelines for efficient network design. Accordingly, a new architecture is presented, called ShuffleNet V2. Comprehensive ablation experiments verify that our model is the state-of-the-art in terms of speed and accuracy tradeoff.”

https://arxiv.org/abs/1807.11164

The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks

“Neural network pruning techniques can reduce the parameter counts of trained networks by over 90%, decreasing storage requirements and improving computational performance of inference without compromising accuracy. However, contemporary experience is that the sparse architectures produced by pruning are difficult to train from the start, which would similarly improve training performance. We find that a standard pruning technique naturally uncovers subnetworks whose initializations made them capable of training effectively.

Based on these results, we articulate the "lottery ticket hypothesis:" dense, randomly-initialized, feed-forward networks contain subnetworks ("winning tickets") that - when trained in isolation - reach test accuracy comparable to the original network in a similar number of iterations. ”

https://arxiv.org/abs/1803.03635

The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks

“The winning tickets we find have won the initialization lottery: their connections have initial weights that make training particularly effective.

We present an algorithm to identify winning tickets and a series of experiments that support the lottery ticket hypothesis and the importance of these fortuitous initializations. We consistently find winning tickets that are less than 10-20% of the size of several fully-connected and convolutional feed-forward architectures for MNIST and CIFAR10. Above this size, the winning tickets that we find learn faster than the original network and reach higher test accuracy.”

https://arxiv.org/abs/1803.03635

What's Hidden in a Randomly Weighted Neural Network?

“Training a neural network is synonymous with learning the values of the weights. In contrast, we demonstrate that randomly weighted neural networks contain subnetworks which achieve impressive performance **without ever training the weight values**. Hidden in a randomly weighted Wide ResNet-50 we show that there is a subnetwork (with random weights) that is smaller than, but matches the performance of a ResNet-34 trained on ImageNet.

Not only do these "untrained subnetworks" exist, but we provide an algorithm to effectively find them. We empirically show that as randomly weighted neural networks with fixed weights grow wider and deeper, an "untrained subnetwork" approaches a network with learned weights in accuracy.”

https://arxiv.org/abs/1911.13299

What's Hidden in a Randomly Weighted Neural Network?

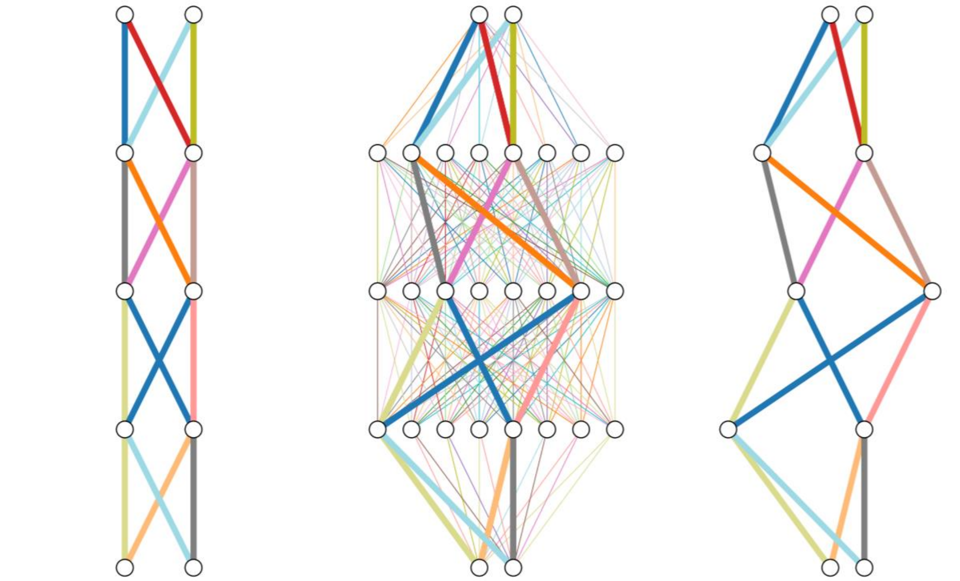

A neural network τ which achieves good performance
Randomly initialized neural network N
A subnetwork τ' of N

Figure 1. If a neural network with random weights (center) is sufficiently overparameterized, it will contain a subnetwork (right) that perform as well as a trained neural network (left) with the same number of parameters.

https://arxiv.org/abs/1911.13299

Two minute papers Youtube Channel

https://www.youtube.com/channel/UCbfYPyITQ-7l4upoX8nvctg

Web → https://cg.tuwien.ac.at/~zsolnai/

Patreon → https://www.patreon.com/TwoMinutePapers

Facebook → https://www.facebook.com/TwoMinutePapers

Twitter → https://twitter.com/karoly_zsolnai

Self Supervised Learning (SSL)

Self-supervised learning (or self-supervision) is a relatively recent learning technique where the training data is autonomously labelled.

It is still supervised learning, but the datasets do not need to be manually labelled by a human,

They can e.g. be labelled by finding and exploiting the relations (or correlations) between different input signals (that is, input coming from different sensor modalities).

https://ai.stackexchange.com/questions/10623/what-is-self-supervised-learning-in-machine-learning

Self Supervised Learning (SSL)

For example, consider a robot which is equipped with a proximity sensor (which is a short-range sensor capable of detecting objects in front of the robot at short distances) and a camera (which is long-range sensor, but which does not provide a direct way of detecting objects).

Consider now the task of detecting objects in front of the robot at longer ranges than the range the proximity sensor allows. In general, we could train a CNN to achieve that.

However, to train such CNN, in supervised learning, we would first need a labelled dataset, which contains labelled images (or videos), where the labels could e.g. be "object in the image" or "no object in the image".

In supervised learning, this dataset would need to be manually labelled by a human, which clearly would require a lot of work.

https://ai.stackexchange.com/questions/10623/what-is-self-supervised-learning-in-machine-learning

Self Supervised Learning (SSL)

To overcome this issue, we can use a self-supervised learning approach. In this example, the basic idea is to associate the output of the proximity sensors at a time step t′>t with the output of the camera at time step t (a smaller time step than t′).

At time *t′*, the robot is at position (*x′*,*y′*). At time step *t′*, the output of the proximity sensor will e.g. be "object in front of the robot" or "no object in front of the robot".

Without loss of generality, suppose that the output of the proximity sensor at *t′*>*t* is "no object in front of the robot", then the label associated with the output of the camera (an image frame) at time *t* will be "no object in front of the robot".

For more details: Learning to Perceive Long-Range Obstacles Using Self-Supervision from Short-Range Sensors

M Nava, J Guzzi, RO Chavez-Garcia, LM Gambardella, A Giusti, IEEE Robotics and Automation Letters 4 (2), 1279-1286, 2019

https://ai.stackexchange.com/questions/10623/what-is-self-supervised-learning-in-machine-learning

Self-supervised vs. supervised learning

Self-supervised Learning is supervised learning because its goal is to learn a function from pairs of inputs and labeled outputs.

Explicit use of labeled input-outputs pairs in self-supervised learning is not needed. Instead, correlations, embedded metadata, or domain knowledge available within the input is implicitly and autonomously extracted from the data and used as supervisory signals.

Like supervised learning, self-supervised learning has use cases in regression and classification.

https://hackernoon.com/self-supervised-learning-gets-us-closer-to-autonomous-learning-be77e6c86b5a

Self-supervised vs. unsupervised learning

Self-supervised learning is like unsupervised Learning because the system learns without using explicitly-provided labels.

It is different from unsupervised learning because we are not learning the inherent structure of data.

Self-supervised learning, unlike unsupervised learning, is not centered around clustering and grouping, dimensionality reduction, recommendation engines, density estimation, or anomaly detection.

https://hackernoon.com/self-supervised-learning-gets-us-closer-to-autonomous-learning-be77e6c86b5a

Semi-supervised learning

Semi-supervised learning is a class of machine learning tasks and techniques that also make use of unlabeled data for training – typically a small amount of labeled data with a large amount of unlabeled data.

Semi-supervised learning falls between unsupervised learning (without any labeled training data) and supervised learning (with completely labeled training data).

Many machine-learning researchers have found that unlabeled data, when used in conjunction with a small amount of labeled data, can produce considerable improvement in learning accuracy.

https://www.wikiwand.com/en/Semi-supervised_learning

Self-supervised vs. semi-supervised learning

Combination of labeled and unlabeled data is used to train semi-supervised learning algorithms, where smaller amounts of labeled data in conjunction with large amounts of unlabeled data can speed up learning tasks.

Self-supervised learning is different as systems learn entirely without using explicitly-provided labels.

https://hackernoon.com/self-supervised-learning-gets-us-closer-to-autonomous-learning-be77e6c86b5a

Main Limitations to current DL systems

The supervised and reinforcement learning paradigms seem to require enormously more samples or trials than humans and animals to learn a new task.

One solution to this may be self-supervised learning (SSL) in which a system is not trained for a particular task, but is trained to capture the regularity of the input by observation, through a prediction or a reconstruction criterion.

This SSL paradigm has been astonishingly successful in NLP: good representations of text are learned by getting a large transformer network to predict missing words in a text. Similar success in images and video are starting to appear.

"in AI, the next revolution will not be supervised". Leaders in computer vision have been saying very similar things including Alyosha Efros (who coined the above slogan a few years ago) and Jitendra Malik (who says "labeled samples are the opium of the AI researcher").

So the idea the deep learning should go beyond supervised and reinforcement learning towards more animal-like self-supervised learning has been very, very widely shared among prominent researchers in the AI community for years. It is not some sort of new epiphany.

Yann Lecun, Chief AI Scientist at Facebook, Prof. at NewYork University

Main Limitations to current DL systems

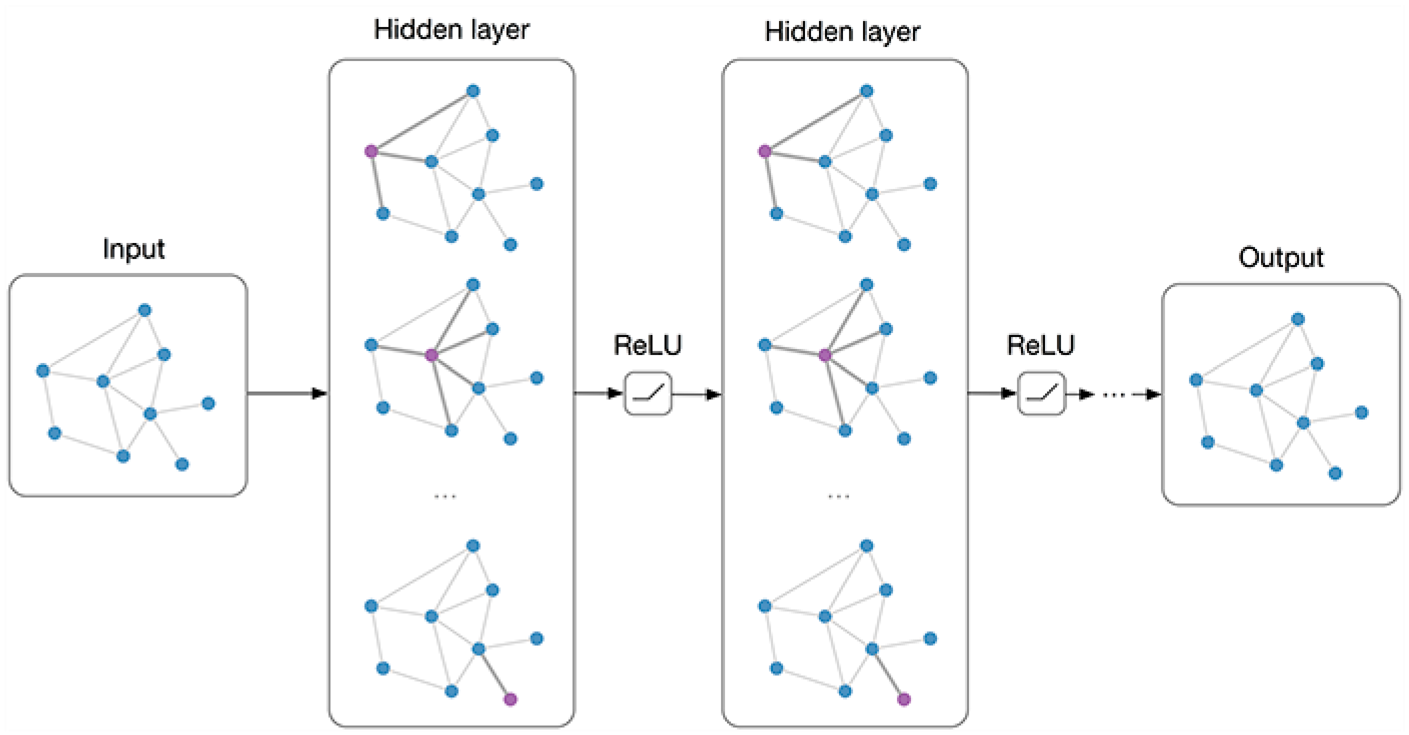
The second challenge is to get machines to learn to reason and plan, not just to perceive and to act reactively.

The challenge here is to get learning machines to be able to perform long chains of reasoning. This is a major research program for a large number of people at FAIR, Google Brain, DeepMind, AI2 and many other non-academic and academic research outfits.

The question is not ignored. The question is not recent either (there are several papers on how to approach this over the last 3 decades). Perhaps the most interesting developments in this direction started with multiplicative interactions (aka "attention"), leading to memory-augmented network, leading to multihead-self-attention modules, leading to transformer networks. There is fascinating recent work on dynamically-generated networks for reasoning tasks.

Yann Lecun, Chief AI Scientist at Facebook, Prof. at NewYork University

Graph Convolutional Networks (GCN)



https://tkipf.github.io/graph-convolutional-networks/

Graph Convolutional Networks (GCN)

Many important real-world datasets come in the form of graphs or networks: social networks, knowledge graphs, protein-interaction networks, the World Wide Web, etc.

Generalizing well-established neural models like RNNs or CNNs to work on arbitrarily structured graphs is a challenging problem. Some recent papers introduce problem-specific specialized architectures (e.g. Duvenaud et al., NIPS 2015; Li et al., ICLR 2016; Jain et al., CVPR 2016), others make use of graph convolutions known from spectral graph theory[1] (Bruna et al., ICLR 2014; Henaff et al., 2015) to define parameterized filters that are used in a multi-layer neural network model, akin to "classical" CNNs.

https://tkipf.github.io/graph-convolutional-networks/

Graph Convolutional Networks (GCN)

Graph convolutional network (GCN) is a neural network that operates on graphs.

Given a graph $G = (V, E)$, a GCN takes as input an $N \times F^o$ input feature matrix, $\boldsymbol{X}$, where

N is the number of nodes and

F^o is the number of input features for each node, and

an $N \times N$ matrix representation of the graph structure such as the adjacency matrix $\boldsymbol{A}$ of G.

https://towardsdatascience.com/how-to-do-deep-learning-on-graphs-with-graph-convolutional-networks-7d2250723780

Graph Convolutional Networks (GCN)

A hidden layer in the GCN can thus be written as

$\boldsymbol{H^i} = f(\boldsymbol{H}^{i-1}, \boldsymbol{A}))$

where $\boldsymbol{H^0} = \boldsymbol{X}$ and f is a propagation.

Each layer $\boldsymbol{H^i}$ corresponds to an $N \times F^{i}$ feature matrix where each row is a feature representation of a node.

At each layer, these features are aggregated to form the next layer's features using the propagation rule f. In this way, features become increasingly more abstract at each consecutive layer. In this framework, variants of GCN differ only in the choice of propagation rule f .

https://towardsdatascience.com/how-to-do-deep-learning-on-graphs-with-graph-convolutional-networks-7d2250723780