Session 2
Convolutional Neural networks

Breakthrough NN

Applications

Modern Convnets

Breakthrough of Neural network

Convolution neural network

Deep-learning

So far…

- Aside from the problems of training, generalization, architecture, forward/backward connections, we can see the major obstacle of traditional NN is way it “SEE” the input data.
- In the old time, most of the practical tasks that implement NN are linear, smaller input dimension (such weather forecast, polynomial fitting, error estimation of navigation) and it involves expert experience in hidden layers design (what should be useful as feature?)
- It is roughly 53 years of slow development from the first workable neural network (1957, perceptron) to the modern architecture (2010)
- How can we encode our four dimensional world information in a much effective way to the computer ?

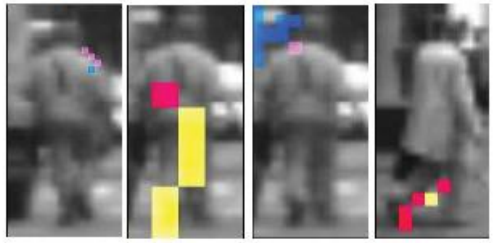
Importance of features in classical Machine learning

Traditional Machine Learning Flow

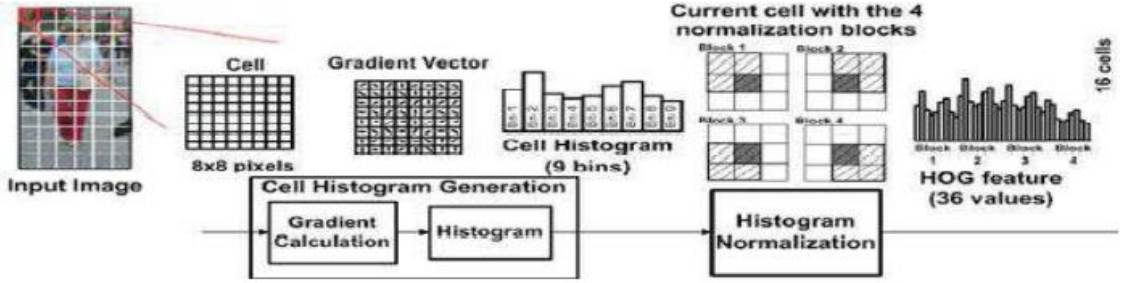
Examples of *hand-crafted* features

Haar features

Control-points features

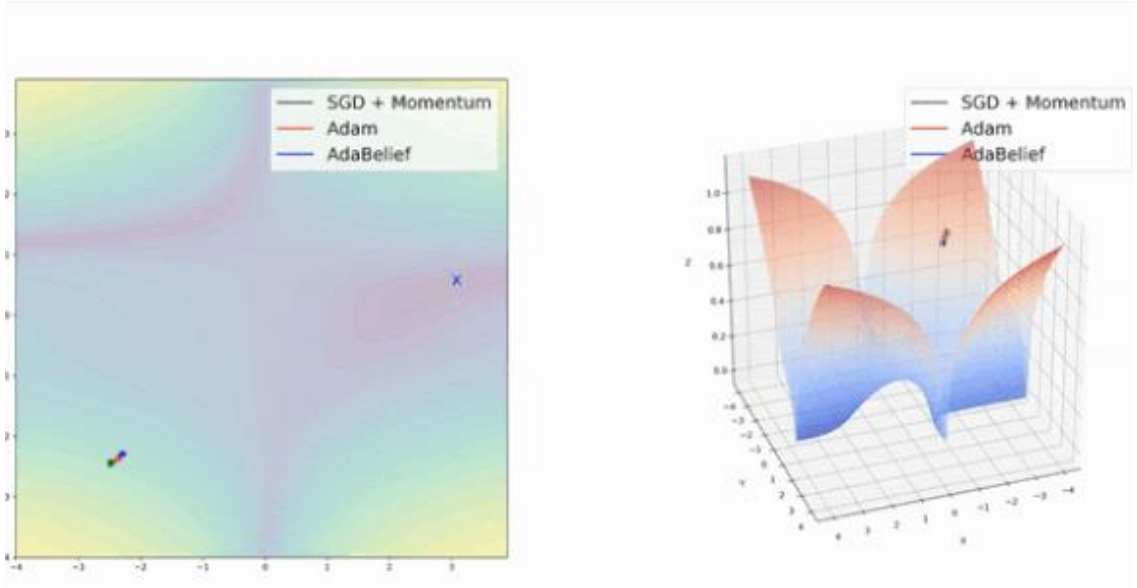


HoG (Histogram of Gradients)



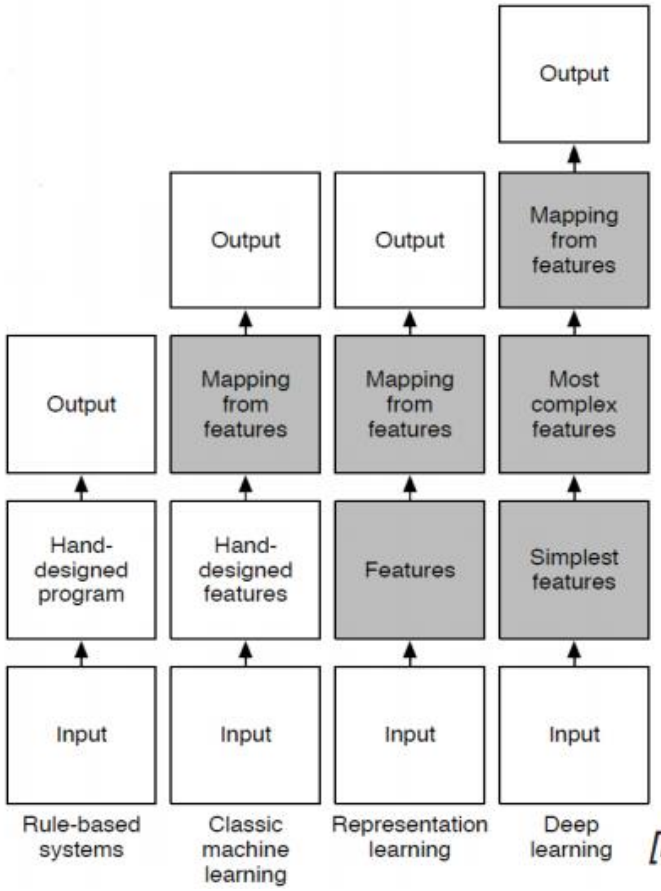
The turning point

- Convolution operation is proposed to solve the tasks especially in 2D or 3D images tasks➔ Section 2
- Recurrent network starts to have more mature architecture (LSTM, GRU) to “REMEMBER” and fuse information through time ➔ Section 4
- Reinforcement learning becomes more practical thanks to the computational power and the memory storage ➔ Section 5
- Aside from the type of neural network, deep learning succeeds to find several innovative ways (data augmentation, optimization algorithms) in training and it is on going research area



What is deep learning ?

Learning a *hierarchy* of increasingly abstract *representations*



Increasing level of abstraction

Each stage ~ trainable feature transform

Image recognition

Pixel → edge → texton → motif → part → object

Speech

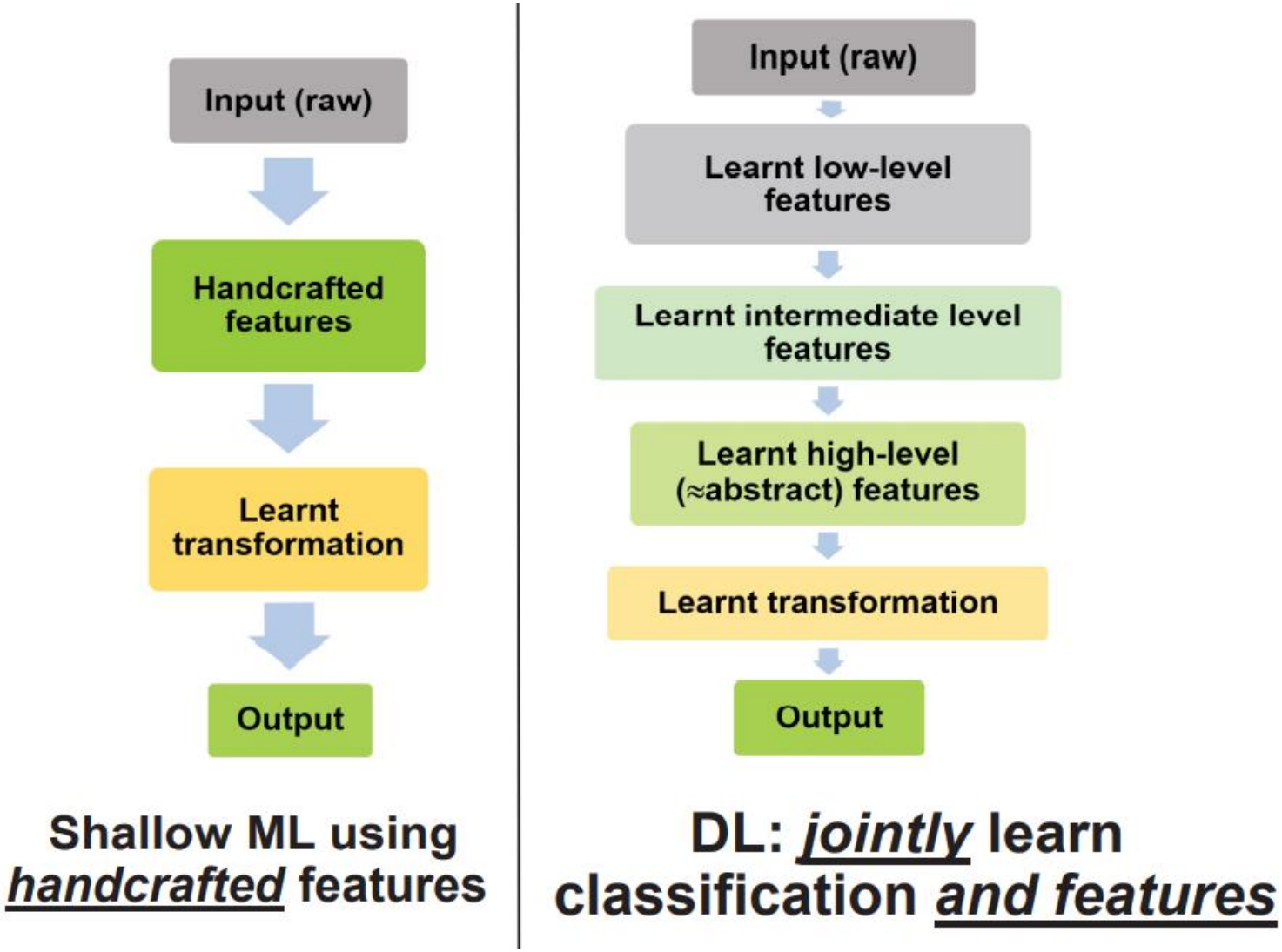
Sample → spectral band → … → phoneme → word

Text

Character → word → word group → clause → sentence → story

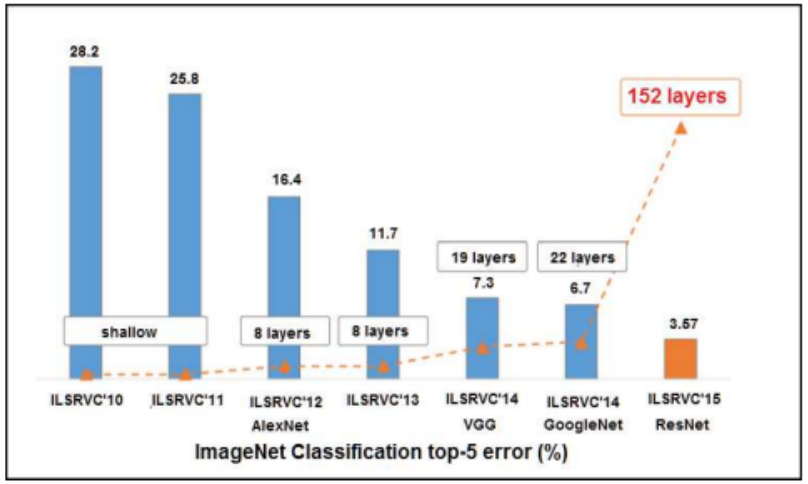
[Figure from Goodfellow]

DL v.s. classical ML



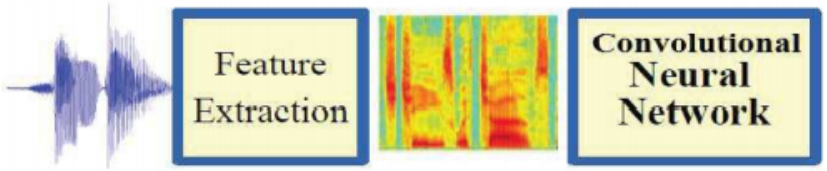
Deep learning recent breakthrough

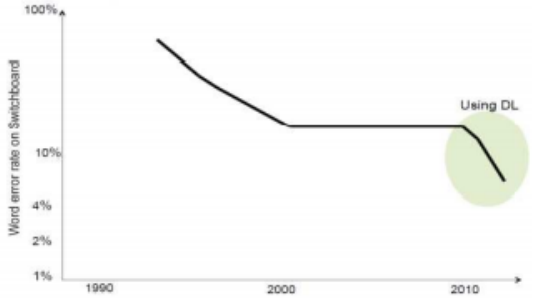
Very significant improvement over State-of-the-Art in Pattern Recognition / Image Semantic Analysis:



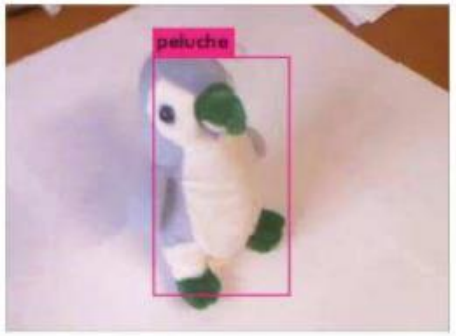
- **won many vision pattern recognition competitions (OCR, TSR, object categorization, facial expression,…)**
- **deployed in photo-tagging by Facebook, Google,Baidu,…**

Similar dramatic progress in Speech recognition + Natural Language Processing (NLP)



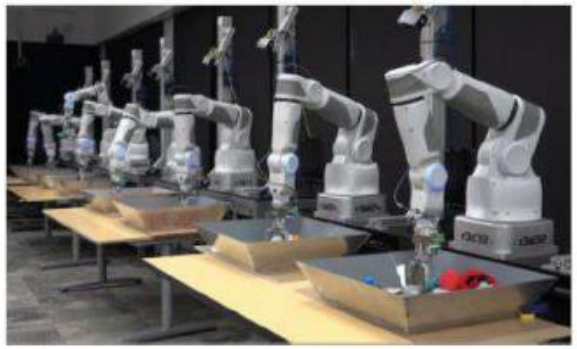


More applications



Object recognition

Scene analysis

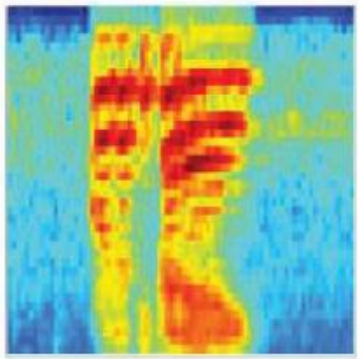


Robotics

Все счастливые семьи
похожи друг на друга,
каждая несчастливая семья
несчастлива по-своему.

Happy families are all alike.
Every unhappy family is
unhappy in its own way.

Language processing



Speech recognition

Medical diagnosis
& Bio-informatics

Why features?

- Real data examples for a given task are usually not spread everywhere in input space, but rather clustered on a low-dimension « manifold »
- Example: Face images of 1000x1000 pixels

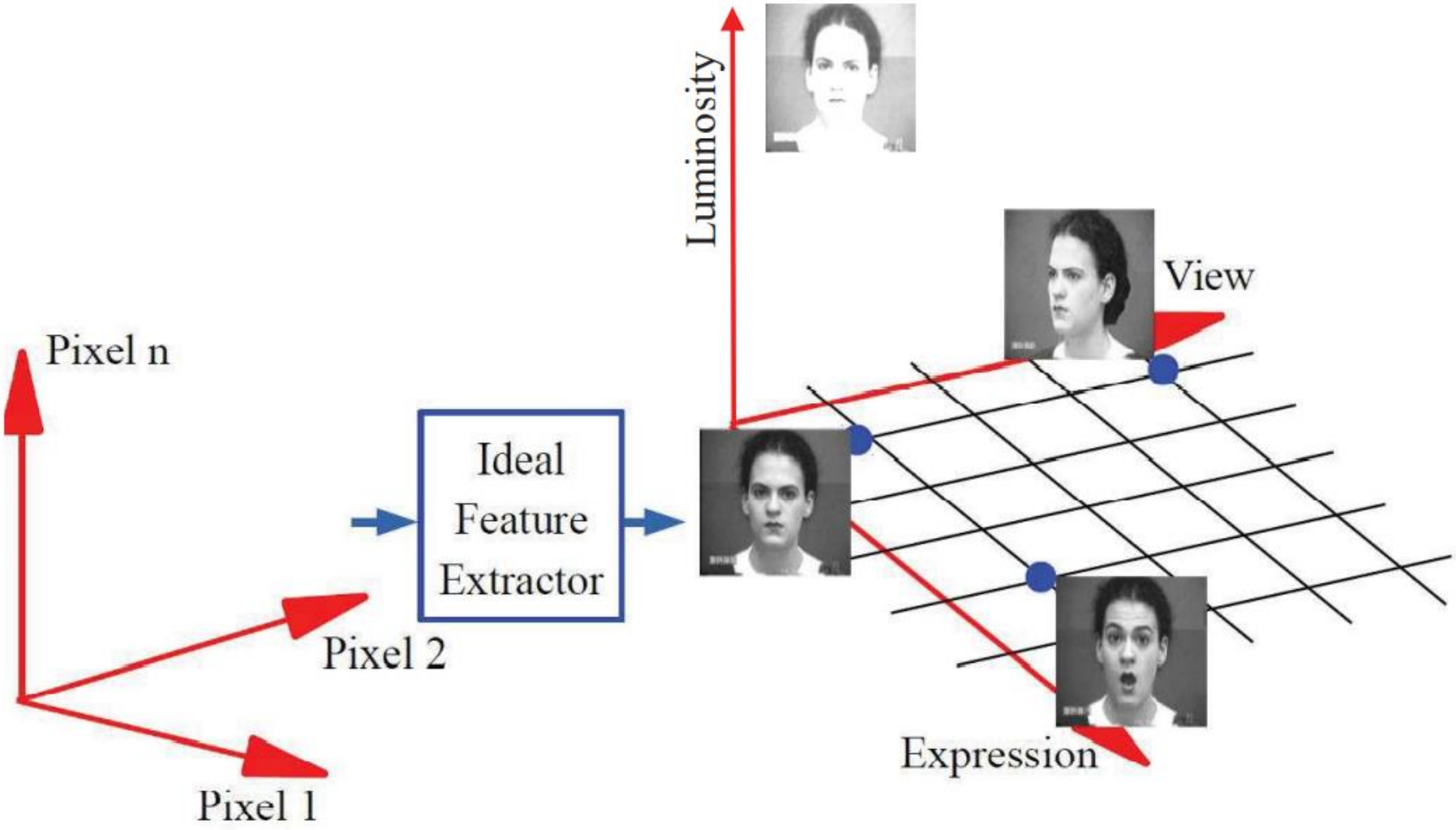
➔« raw » examples are vectors in R^1000000 !!

- BUT:
  - position = 3 cartesian coord
  - orientation 3 Euler angles
  - 50 muscles in face
  - Luminosity, color

  ➔Set of all images of ONE person has ≤ 69 dim

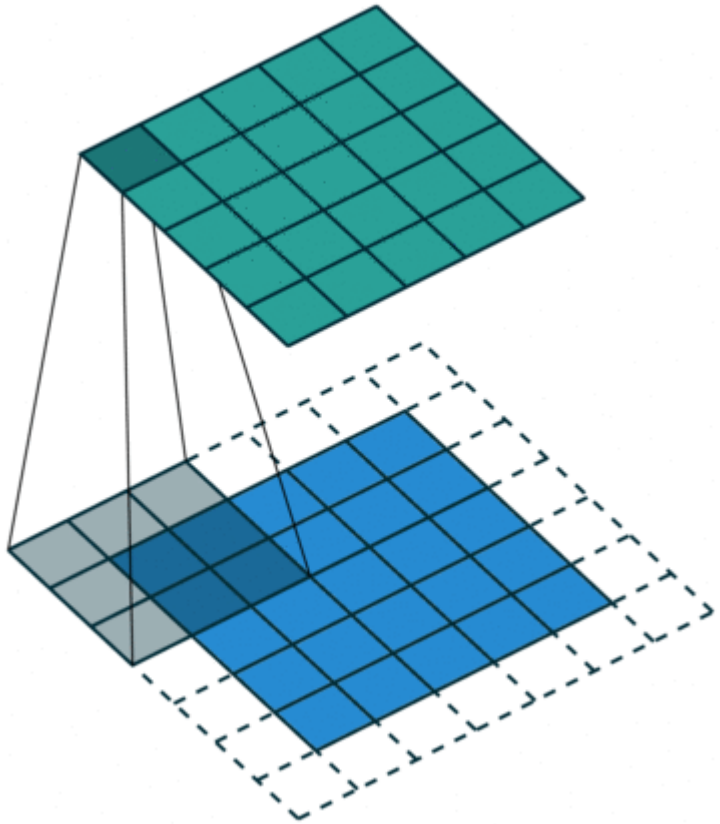
**This is why the powerful convolution operation is introduced!

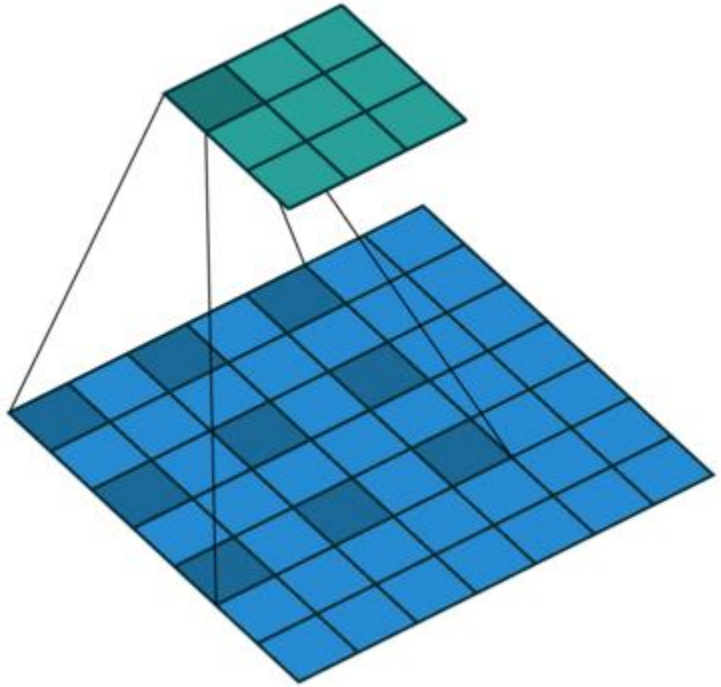
Ideal features



Key innovations in feature learning

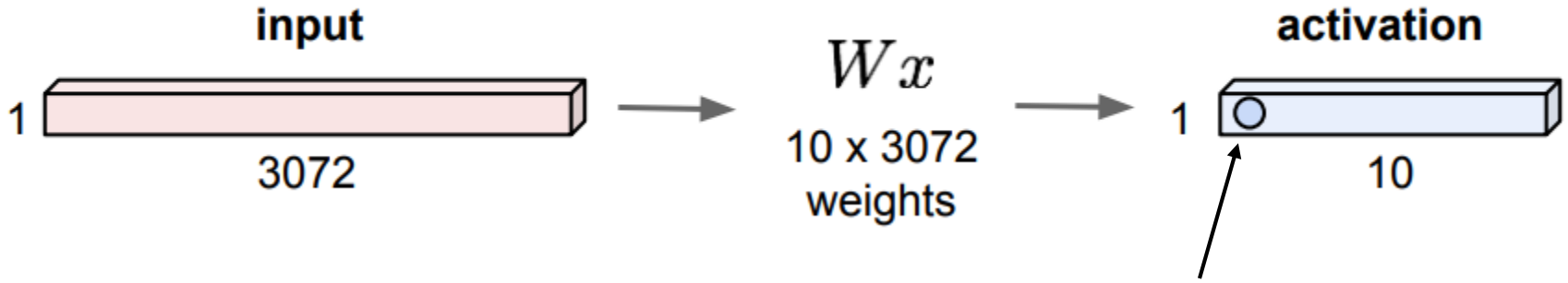
- Convolution operation: fused the pixels' information spatially
- Atrous convolution (Dilated convolution)





Fully connected layer

32x32x3 image -> stretch to 3072x1

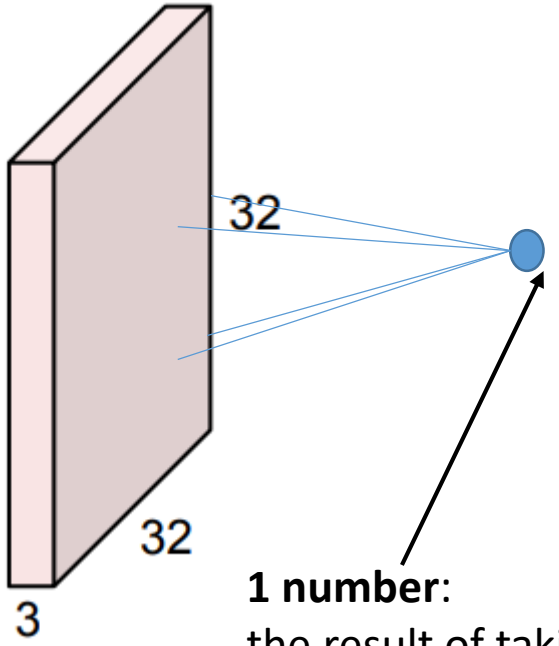


1 number:
the result of taking dot product between a row of W and the input (a 3072-dimensional dot product

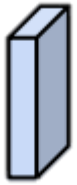
Convolution layer

Filters always extend the full depth of the input volume

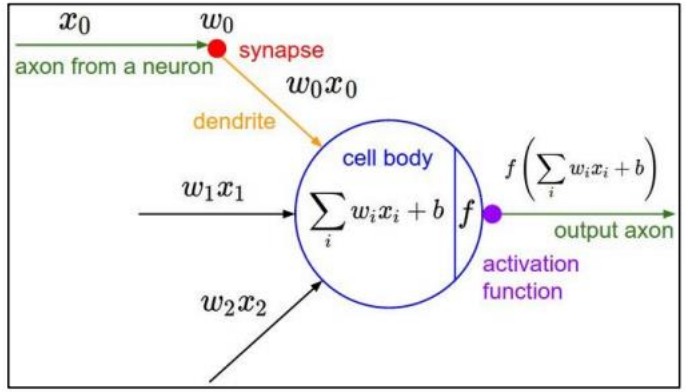
32x32x3 image (x)->
preserve spatial structure



5x5x3 filter (w)



Convolve the filter with the image i.e. "slide over the image spatially, computing dot products"



It's just a neuron with local connectivity...

1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image (i.e. 5*5*3 = 75-dimensional dot product + bias): $y = \phi(w^T x + b)$ where ϕ is the activation function

The output size of convolution

- There are three hyper parameters control the size of the output volume: the depth, stride and zero-padding
  - Depth(D): It corresponds to the number of filters would like to use, each learning to look for something different in the input.
  - Stride(S): the way we slide the filter. When the stride is 1 then we move the filter one pixel at a time. When the stride is N then the filter jump N pixels at a time. This will produce smaller output volumes spatially.
  - Padding(P): this number pad the input volume with zeros around the border. This is a nice feature that allow us to control the spatial size of the output volumes
- Formula for the calculating the output volume:

$$\frac{W - F + 2P}{S} + 1$$

Input volume: W, kernel size:F

Summary of convolution layer

- Acceps a volume of $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - Their spatial extent F (kernel size)
  - The stride S
  - The amount of zero padding P
- Produces a volume of size $W_2 \times H_2 \times D_2$ where
  - $W_2 = \frac{W_1 - F + 2P}{S} + 1$
  - $H_2 = \frac{H_1 - F + 2P}{S} + 1$
  - $D_2 = K$

Common settings:

K = (powers of 2, e.g. 32, 64, 128, 512)
- F = 3, S = 1, P = 1
- F = 5, S = 1, P = 2
- F = 5, S = 2, P = ? (whatever fits)
- F = 1, S = 1, P = 0

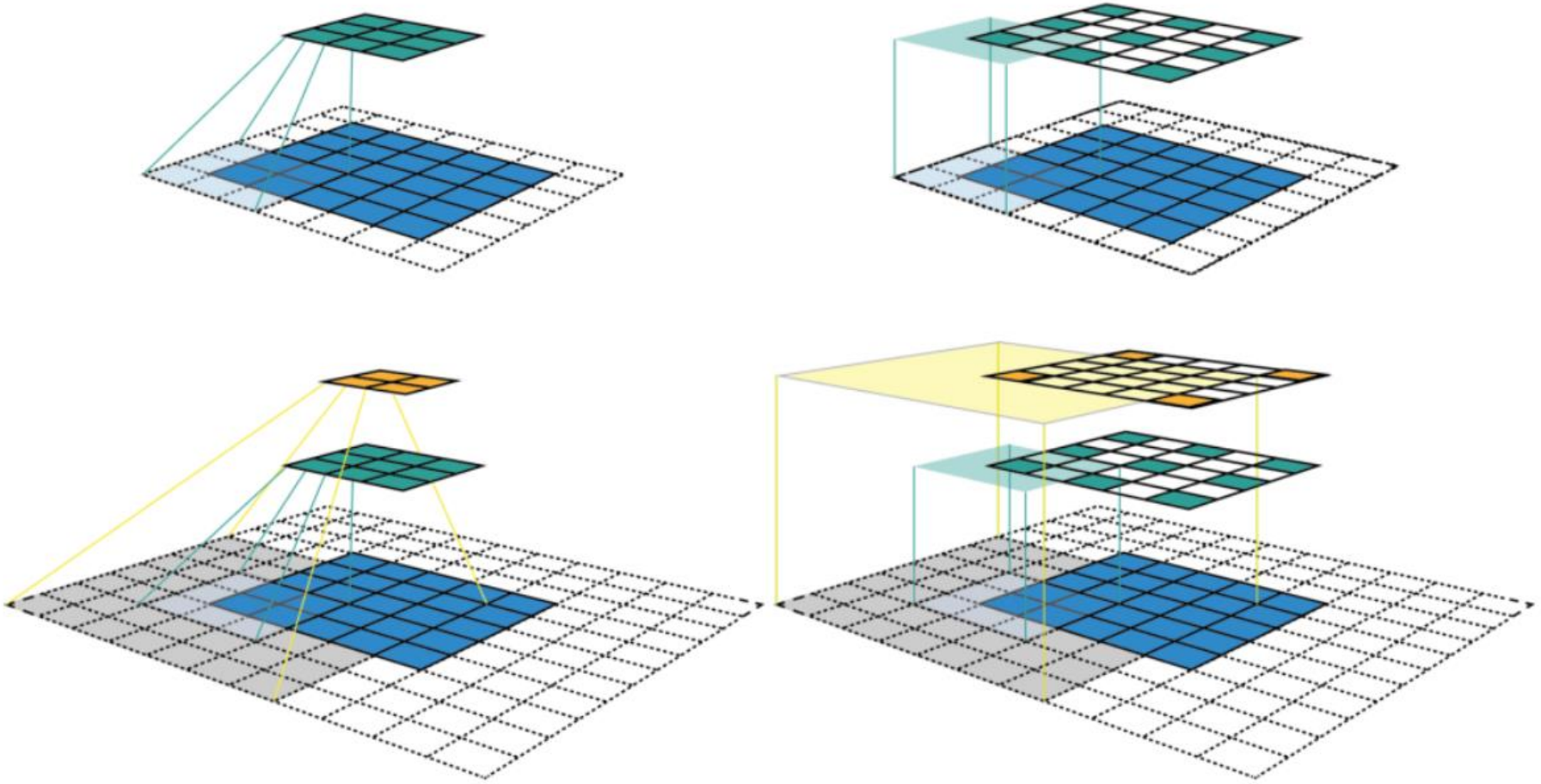
But wait! Why there is 1x1 convolution layers?

This is helpful and make sense because 1x1 convolution performs a K-dimensional dot product. It is used very often to reduce the depth

Receptive field

- The ***receptive field*** is perhaps one of the most important concepts in Convolutional Neural Networks
- *The **receptive field** is defined as the region in the input space that a particular CNN's feature is looking at (i.e. be affected by)*
- A receptive field of a feature can be described by its **center location** and its **size**
  - Number of output feature: $W_{out} = \frac{W_{in} - K + 2P}{S} + 1$
  - Jump size $\mathrm{J_{out}} = \mathrm{J_{in}} * \mathrm{s}$
  - Receptive field size $r_{out} = r_{in} + (K - 1) * \mathrm{J_{in}}$

Receptive field demo

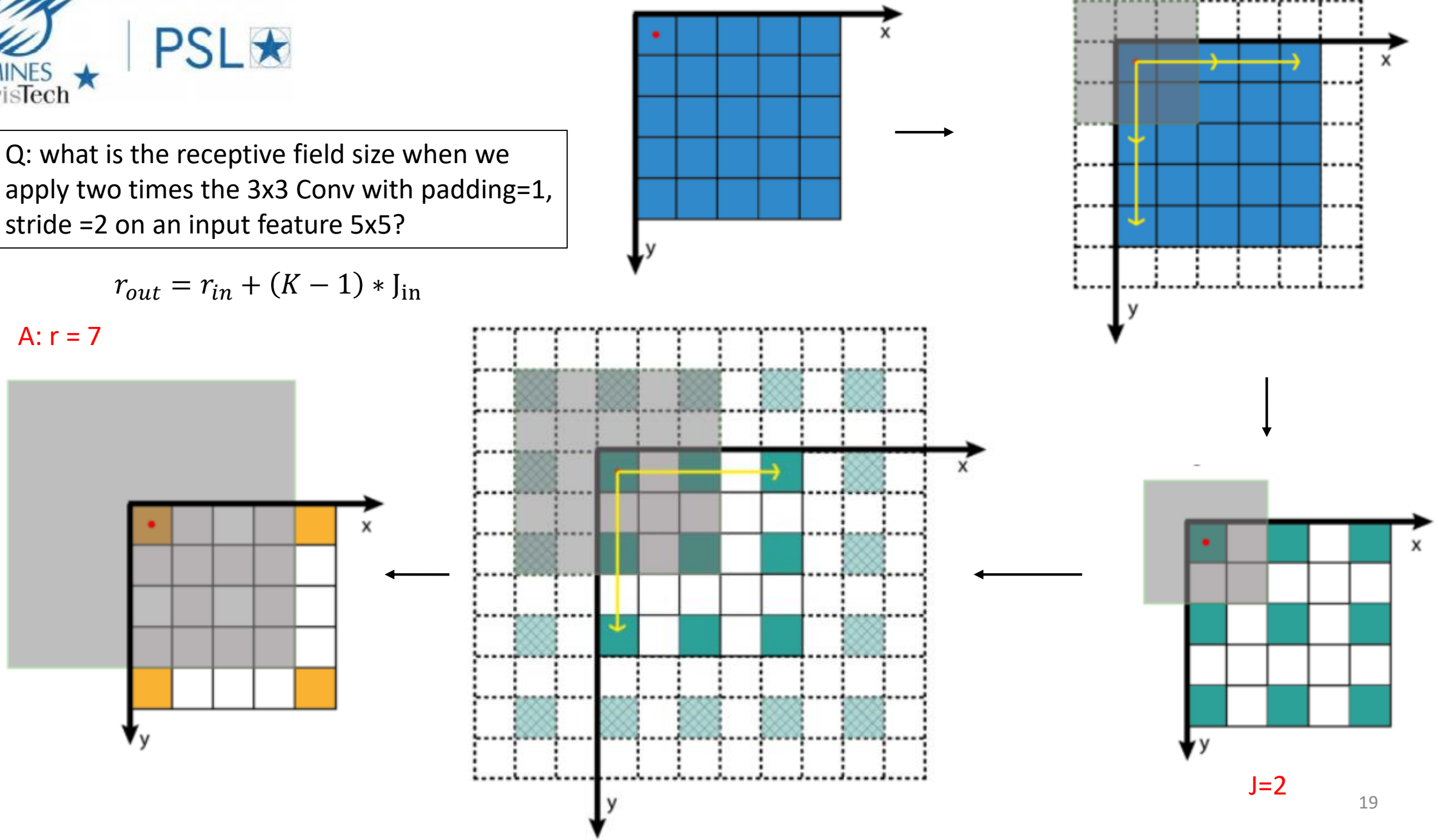


From [https://medium.com/mlreview/a-guide-to-receptive-field-arithmetic-for-convolutional-neural-networks-e0f514068807]

Q: what is the receptive field size when we apply two times the 3x3 Conv with padding=1, stride =2 on an input feature 5x5?

$$r_{out} = r_{in} + (K-1) * \mathrm{J_{in}}$$

A: r = 7



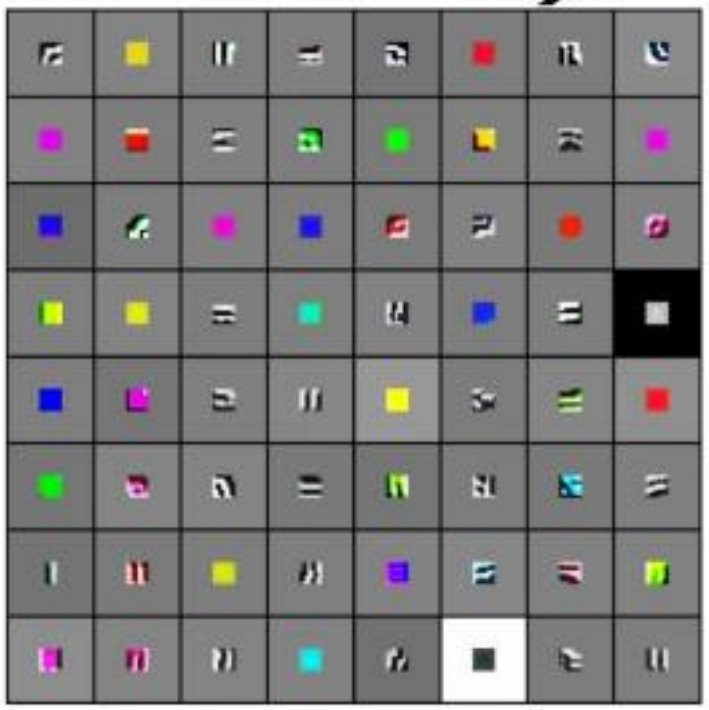
J=2

Preview

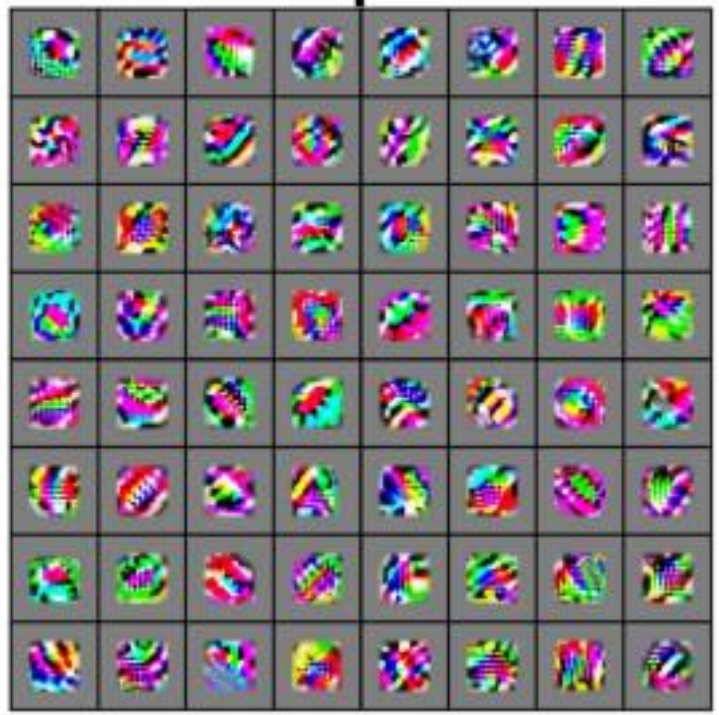
[Zeiler and Fergus 2013]

Visualization of VGG-16 by Lane McIntosh. VGG-16 architecture from [Simonyan and Zisserman 2014].

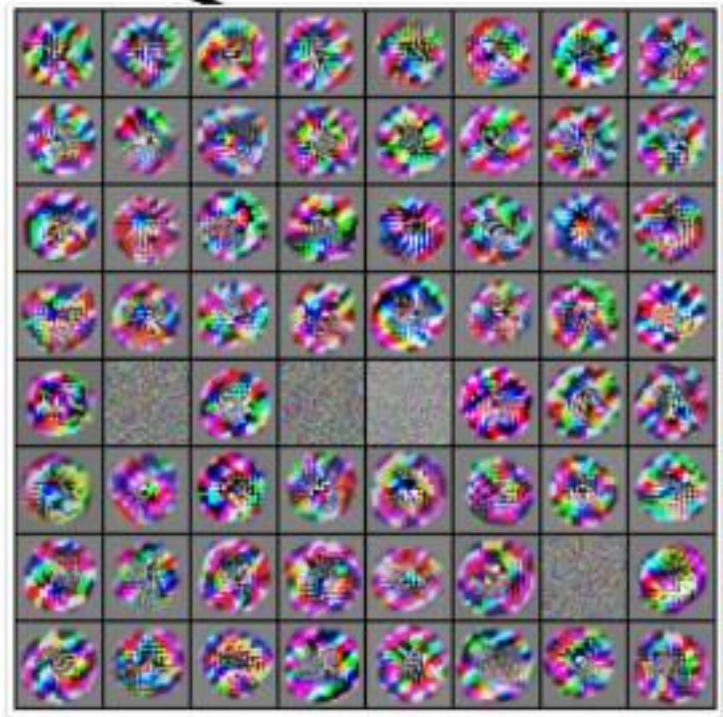
Low-level features → Mid-level features → High-level features → Linearly separable classifier



VGG-16 Conv1_1



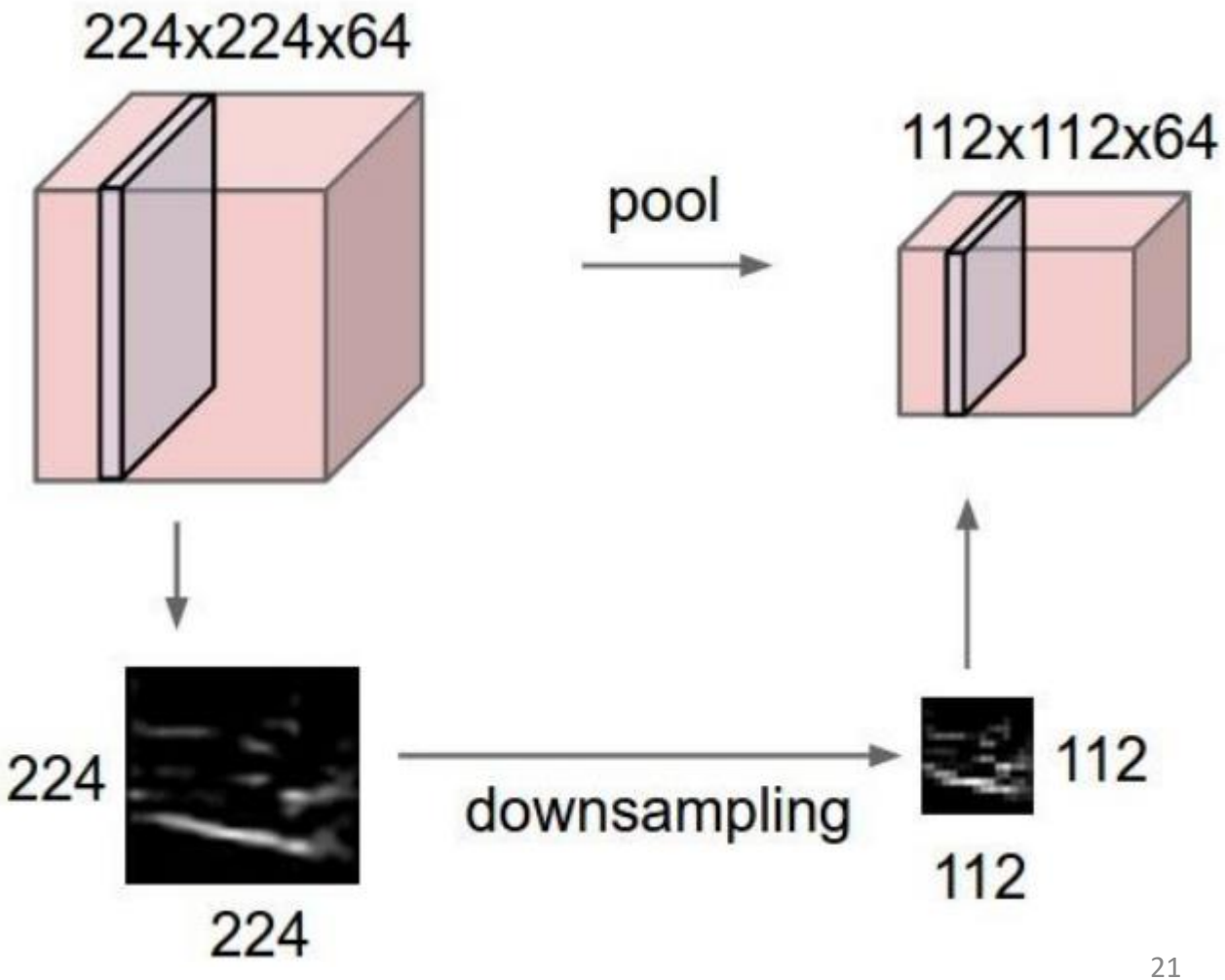
VGG-16 Conv3_2



VGG-16 Conv5_3

Pooling layer

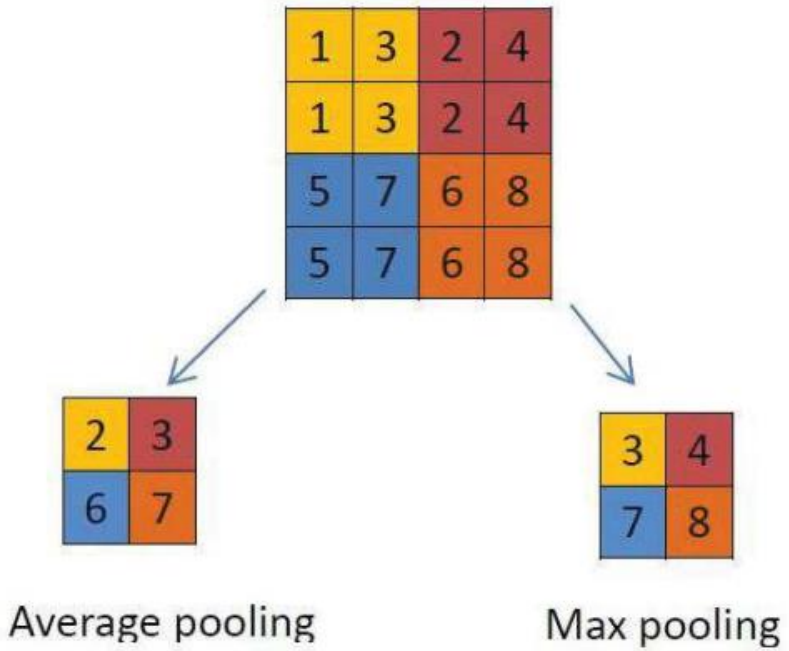
- makes the representations smaller and more manageable
- operates over each activation ma independently
- Noise reduction
- Small-translation invariance
- Small-scaling invariance



Pooling layers algorithm details

Parameters:

- **pooling size (often 2x2)**
- **pooling stride (usually = pooling_size)**
- **Pooling operation: *max*, average, Lp,…**

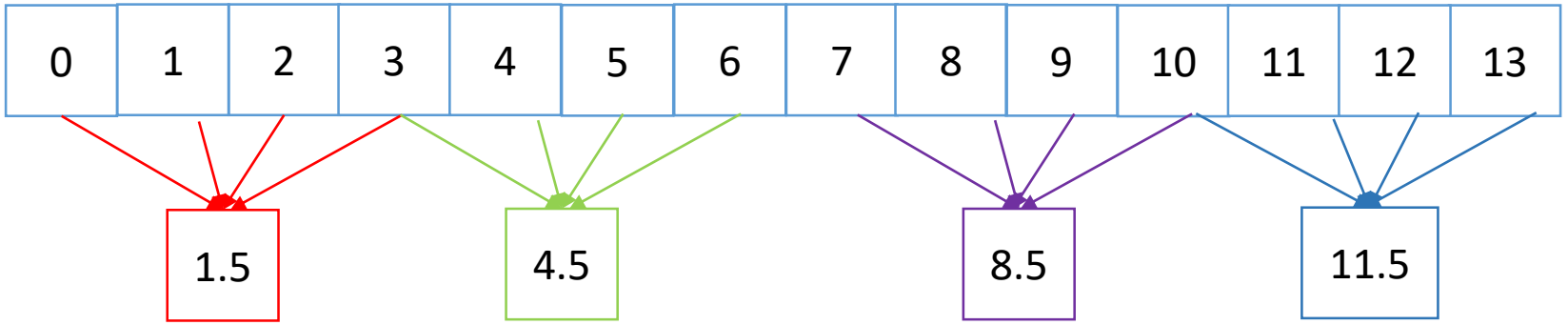


Example: 2x2 pooling, stride 2

Global average Pooling

```
def torch_pool(inputs, target_size):
   start_points = (torch.arange(target_size,
dtype=torch.float32) * (inputs.size(-1) /
target_size)).long()
   end_points = ((torch.arange(target_size,
dtype=torch.float32)+1) * (inputs.size(-1) /
target_size)).ceil().long()
   pooled = []
   for idx in range(target_size):
      pooled.append(torch.mean(inputs[:, :,
start_points[idx]:end_points[idx]], dim=-1,
keepdim=False))
      pooled = torch.cat(pooled, -1)
return pooled
```

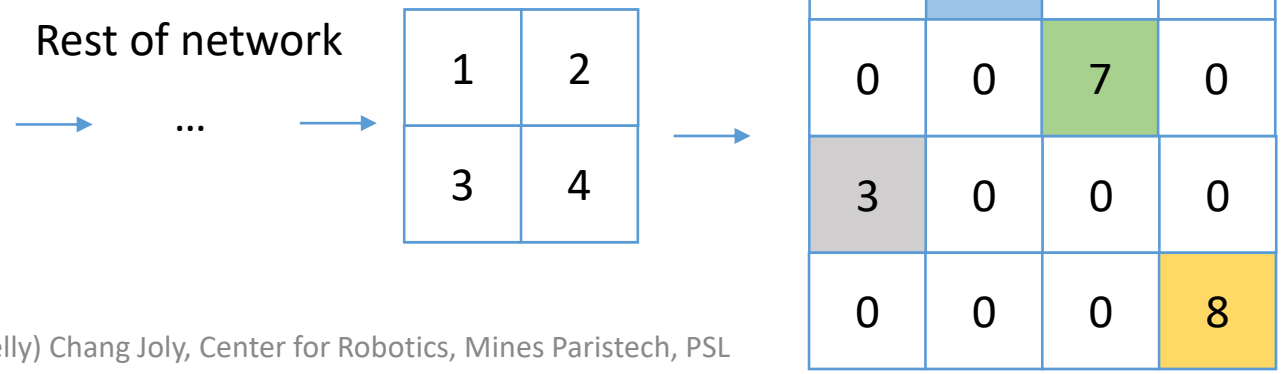
- Global Average Pooling (Lin, et al, 2013)
  - Replace fully connected layers
  - More native to the convolution structure
  - Categories confidence maps
  - No parameters to optimize and thus avoiding overfitting
  - It sums out the spatial information ➔ robust to spatial translations of input
  - Example: torch.nn.AdaptiveAvgPool2d(a,4) with output_size = 4 and input feature map a a.shape = (1,14). Kernel size = (input_size+target_size-1) // target_size = 4



Lin, M., Chen, Q., & Yan, S. (2014). Network in network. *2nd International Conference on Learning Representations, ICLR 2014 - Conference Track Proceedings*.

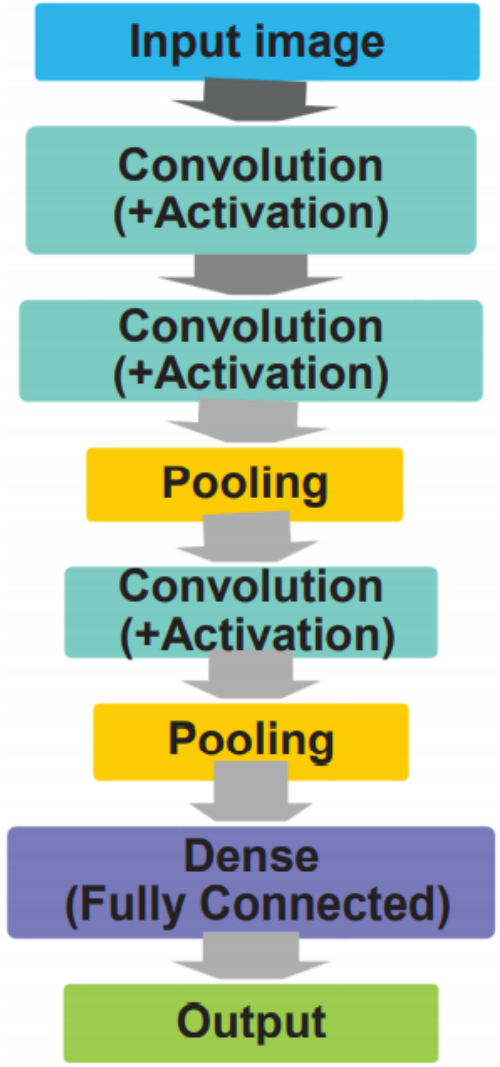
2D exmaple

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25



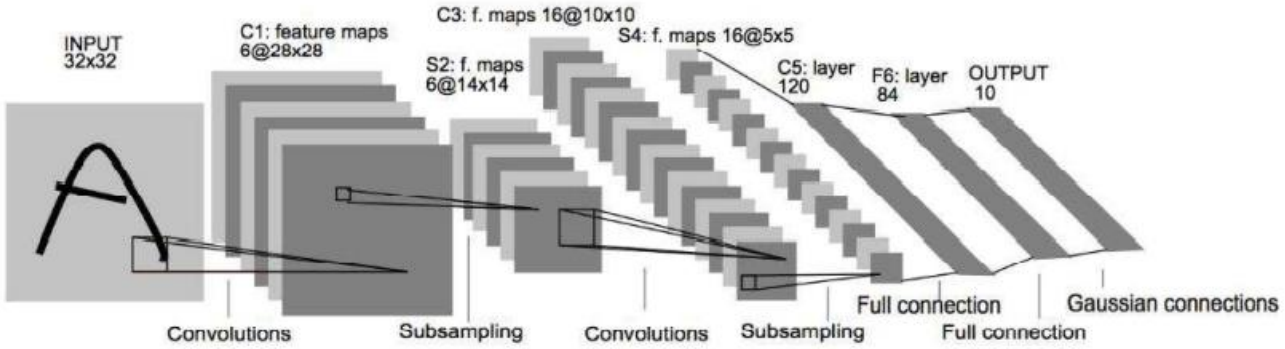
Summary of ConvNet

Succession of Convolution (+ optional activation) layers and Pooling layers, which extract the hierarchy of features, followed by dense (fully connected) layer(s) for final classification

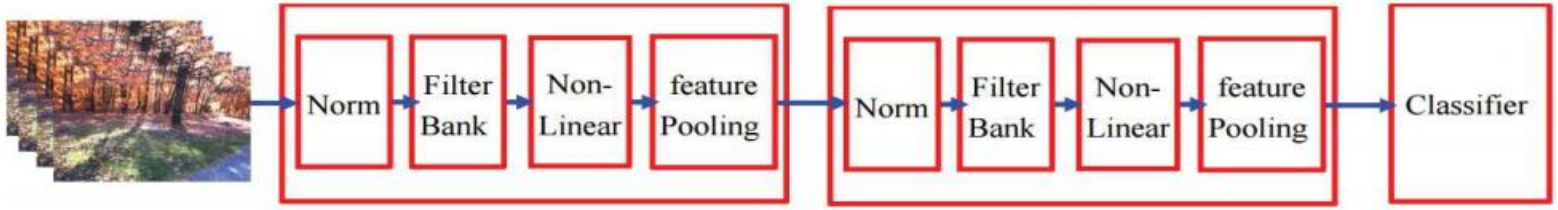


Convolution neural network (ConvNet)

- Proposed in 1998 by Yann LeCun (french prof.@ NYU, now also AI research director of Facebook). However, it took a long time to be popular

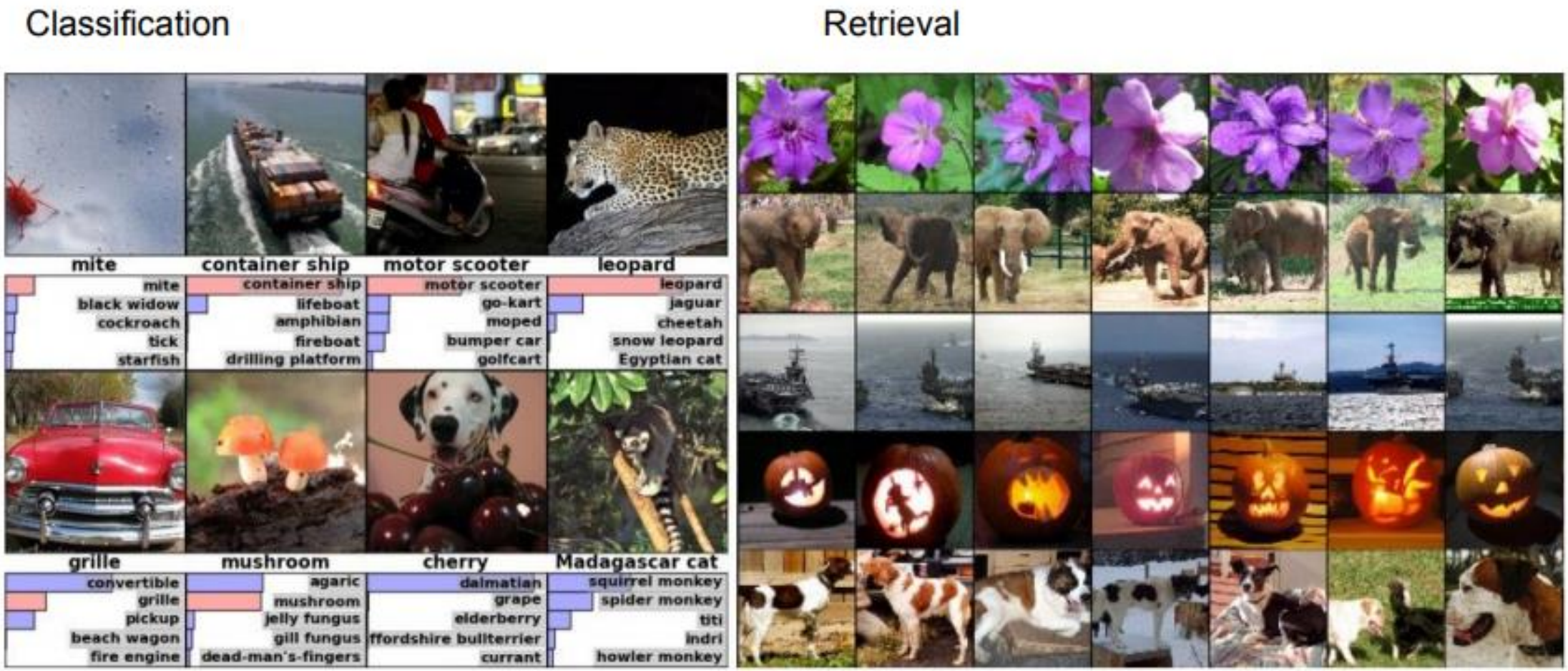


CNN called LeNet by Yann LeCun (1998)



- **For inputs with correlated dims (2D *image*, 1D signal,…)**
- **Supervised learning**

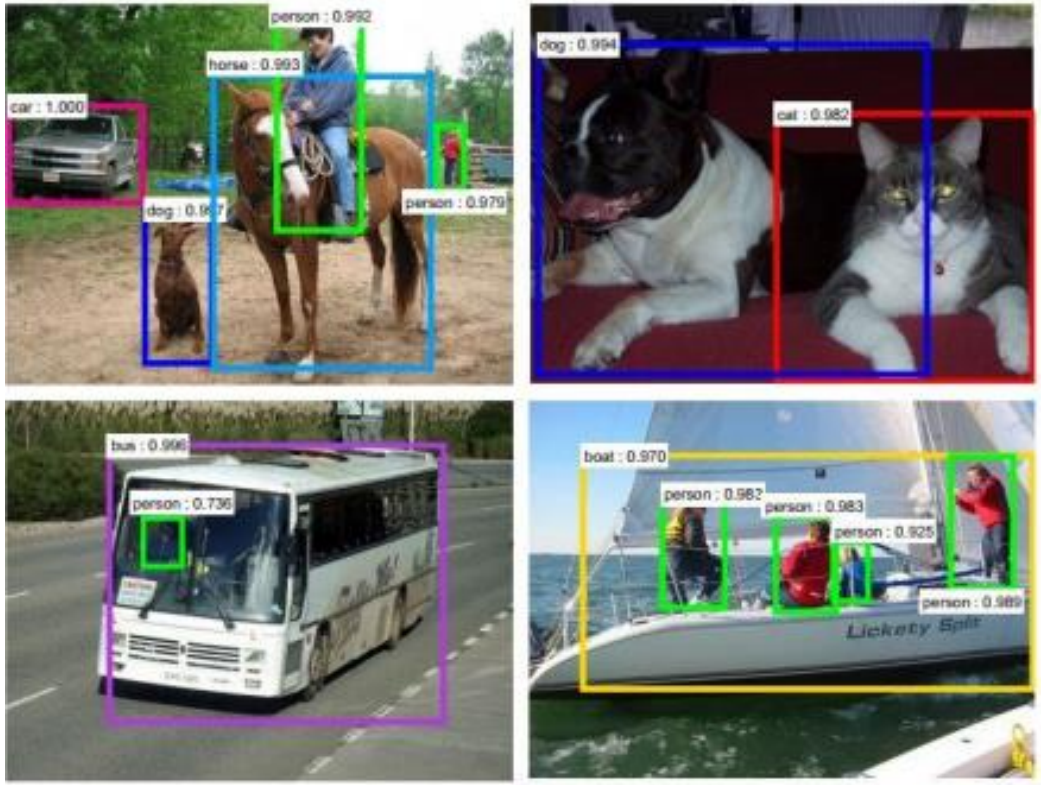
ConvNet is everywhere



Figures copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

ConvNet is everywhere

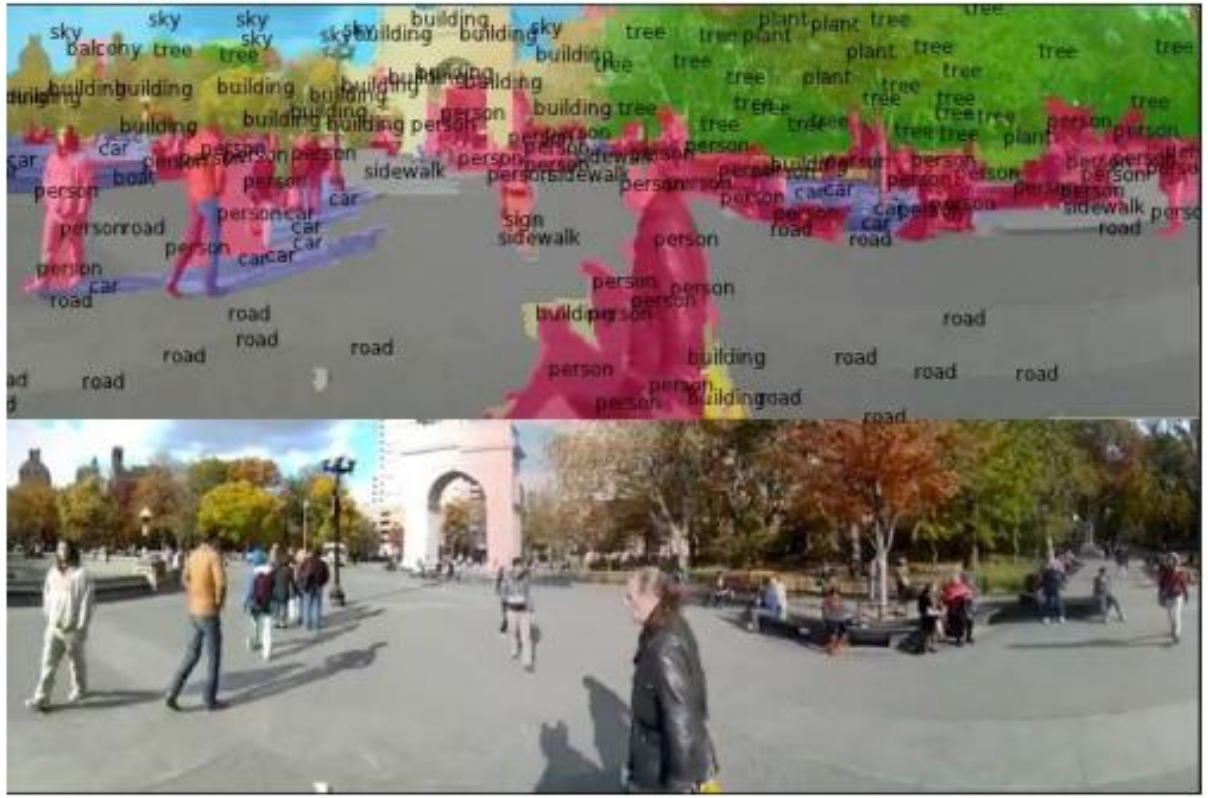
Detection



Figures copyright Shaoqing Ren, Kaiming He, Ross Girschick, Jian Sun, 2015. Reproduced with permission.

[Faster R-CNN: Ren, He, Girshick, Sun 2015]

Segmentation



Figures copyright Clement Farabet, 2012. Reproduced with permission.

[Farabet et al., 2012]

ConvNet is everywhere

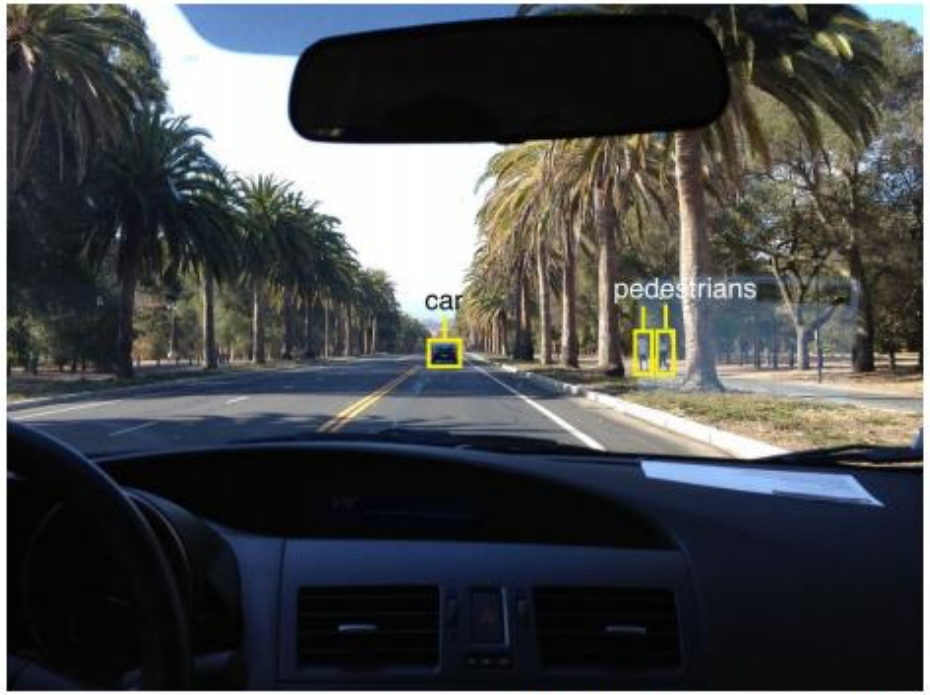
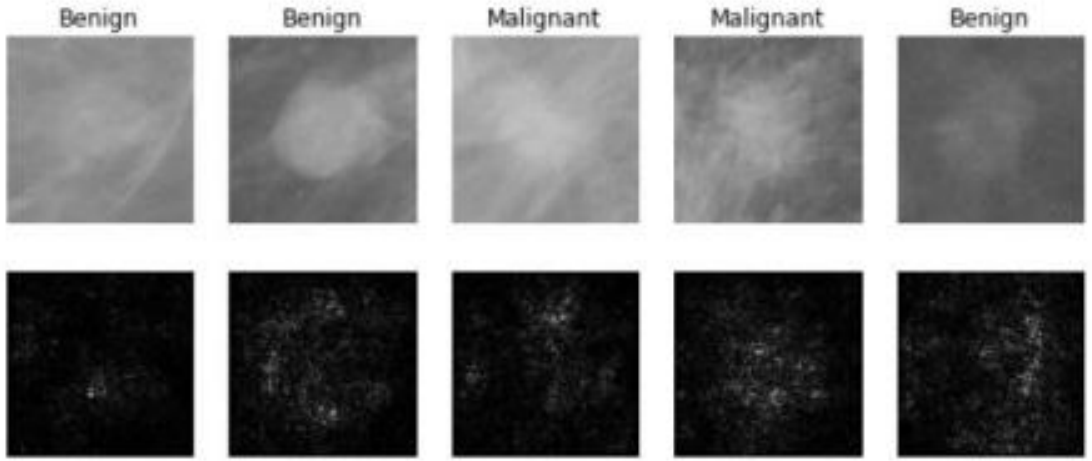


Photo by Lane McIntosh. Copyright CS231n 2017.

self-driving cars



[Levy et al. 2016]

Figure copyright Levy et al. 2016.
Reproduced with permission.

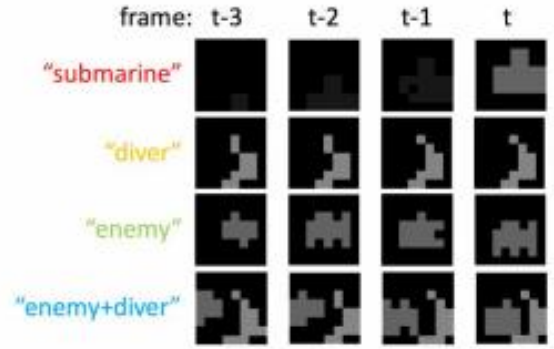
[Dieleman et al. 2014]

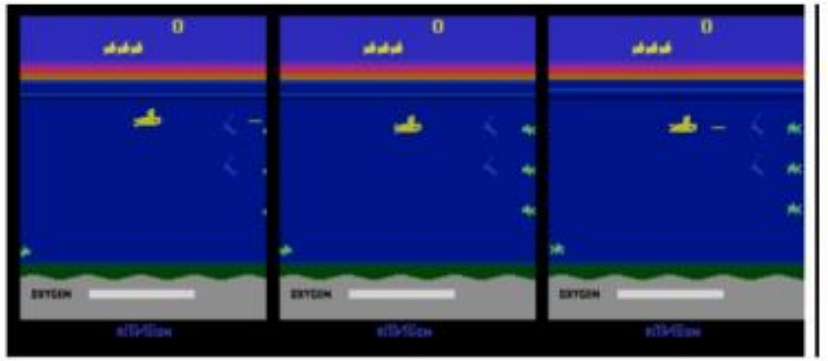
From left to right: public domain by NASA, usage permitted by ESA/Hubble, public domain by NASA, and public domain.

ConvNet is everywhere

Images are examples of pose estimation, not actually from Toshev & Szegedy 2014. Copyright Lane McIntosh.

[Toshev, Szegedy 2014]





[Guo et al. 2014]

Figures copyright Xiaoxiao Guo, Satinder Singh, Honglak Lee, Richard Lewis, and Xiaoshi Wang, 2014. Reproduced with permission.

ConvNet is everywhere

No errors

A white teddy bear sitting in the grass

Minor errors

A man in a baseball uniform throwing a ball

Somewhat related

A woman is holding a cat in her hand

A man riding a wave on top of a surfboard

A cat sitting on a suitcase on the floor

A woman standing on a beach holding a surfboard

Image Captioning

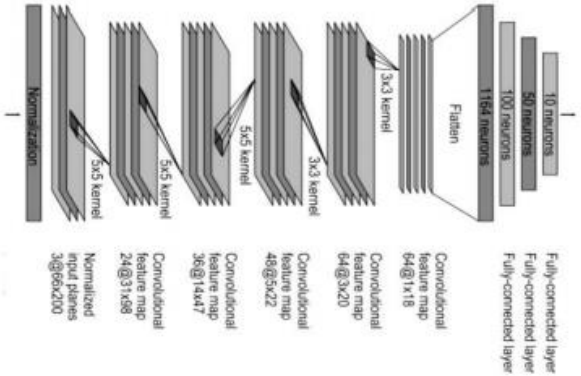
[Vinyals et al., 2015]
[Karpathy and Fei-Fei, 2015]

All images are CC0 Public domain:
https://pixabay.com/en/luggage-antique-cat-1643010/
https://pixabay.com/en/teddy-plush-bears-cute-teddy-bear-1623436/
https://pixabay.com/en/surf-wave-summer-sport-litoral-1668716/
https://pixabay.com/en/woman-female-model-portrait-adult-983967/
https://pixabay.com/en/handstand-lake-meditation-496008/
https://pixabay.com/en/baseball-player-shortstop-infield-1045263/

Captions generated by Justin Johnson using Neuraltalk2

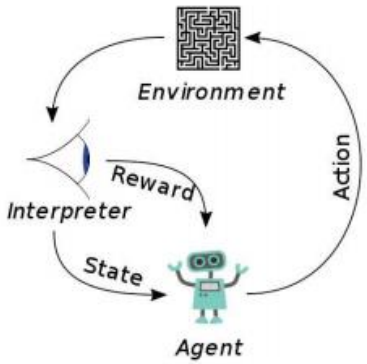
ConvNet is everywhere

ConvNet input:
Cylindrical projection of fisheye camera



ConvNet output: steering angle

Imitation Learning from Human driving on real data



End-to-end driving via Deep *Reinforcement* Learning
[thèse CIFRE Valeo/MINES-ParisTech en cours]

Other image-based applications

- Image classification
- Visual object detection and categorization
- Semantic segmentation of images
- Tracking of human and cars

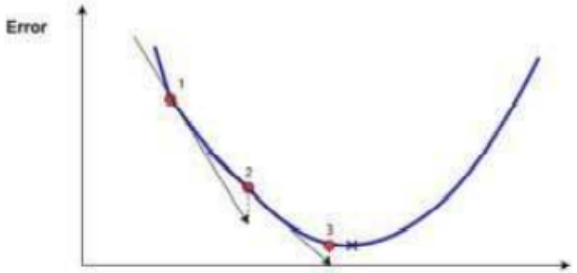
Overview of training

- One time step
  - activation functions, preprocessing, weight initialization, regularization, gradient checking
- Training dynamics
  - babysitting the learning process, parameter updates, hyperparameter optimization
- Evaluation
  - Model ensembles

ConvNet training

All successive layers of a convNet forms a Deep neural network (with weigh-sharing inside each conv. Layer, and specific pooling layers).

Training = optimizing values of weights&biases
Method used = gradient descent



➔ Stochastic Gradient Descent (SGD), using *back-propagation*:

- **Input 1 (or a few) random training sample(s)**
- **Propagate**
- **Calculate error (loss)**
- **Back-propagate through all layers from end to input, to compute gradient**
- **Update convolution filter weights**

They are the same like we face in machining learning problem

Datasets for image classification

Open source and public image dataset

Dataset	Total images	Categories	Images per category	Object per image	Size	Started year	Challenges competition
PASCAL VOC (Everinghame et al. 2010, 2015)	11540	20	303~4087	2.4	470x380	2005	VOC(2007~2012)
ImageNet (Deng et al. 2009)	14 millions+	21841		1.5	500x400	2009	ILSVRC(2010~2017)
MS COCO (Lin et al. 2014)	328000+	91		7.3	640x460	2014	MS COCO(2015~2018)
Open Images (Kuznetsova et al. 2018)	9 millions+	6000+		8.3	varied	2017	OICOD2018

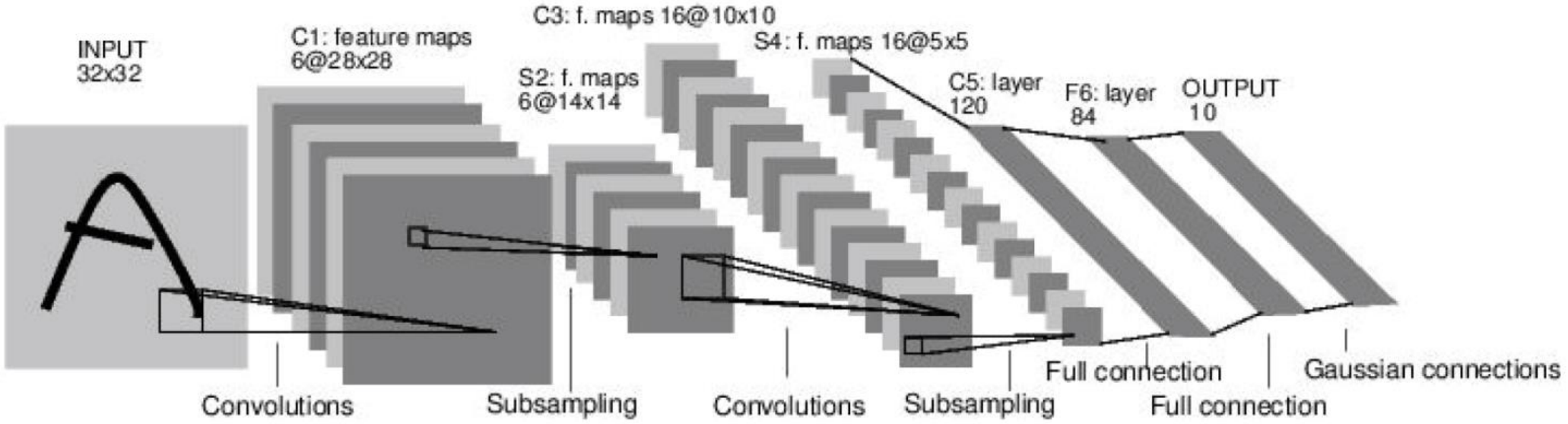
Revised from [Liu, L., Ouyang, W., Wang, X. *et al.* Deep Learning for Generic Object Detection: A Survey. *Int J Comput Vis* **128,** 261–318 (2020)]

Convolution Neural Network Architecture

Popular CNN architecture

- LeNet: 1st successful applications of ConvNets, by Yann LeCun in 1990's. Used to read zip codes, digits, etc.
- AlexNet: Beginning of ConvNet "buzz": largely outperformed competitors in ImageNet_ILSVRC2012 challenge. Architecture similar to LeNet (but deeper+larger, and some chained ConvLayers before Pooling). 60 M parameters !
- GoogLeNet: ILSVRC 2014 winner, developed by Google. Introduced an Inception Module, + AveragePooling instead of FullyConnected layer at output. Dramatic reduction of number of parameters (5M, compared to AlexNet with 60M).
- VGGNet: Runner-up in ILSVRC 2014. Very deep (16 CONV/FC layers) ➔140M parameters !!
- ResNet: ILSVRC 2015, "Residual Network" introducing "skip" connections. Currently ~ SoA in convNet. Very long training but fast execution.

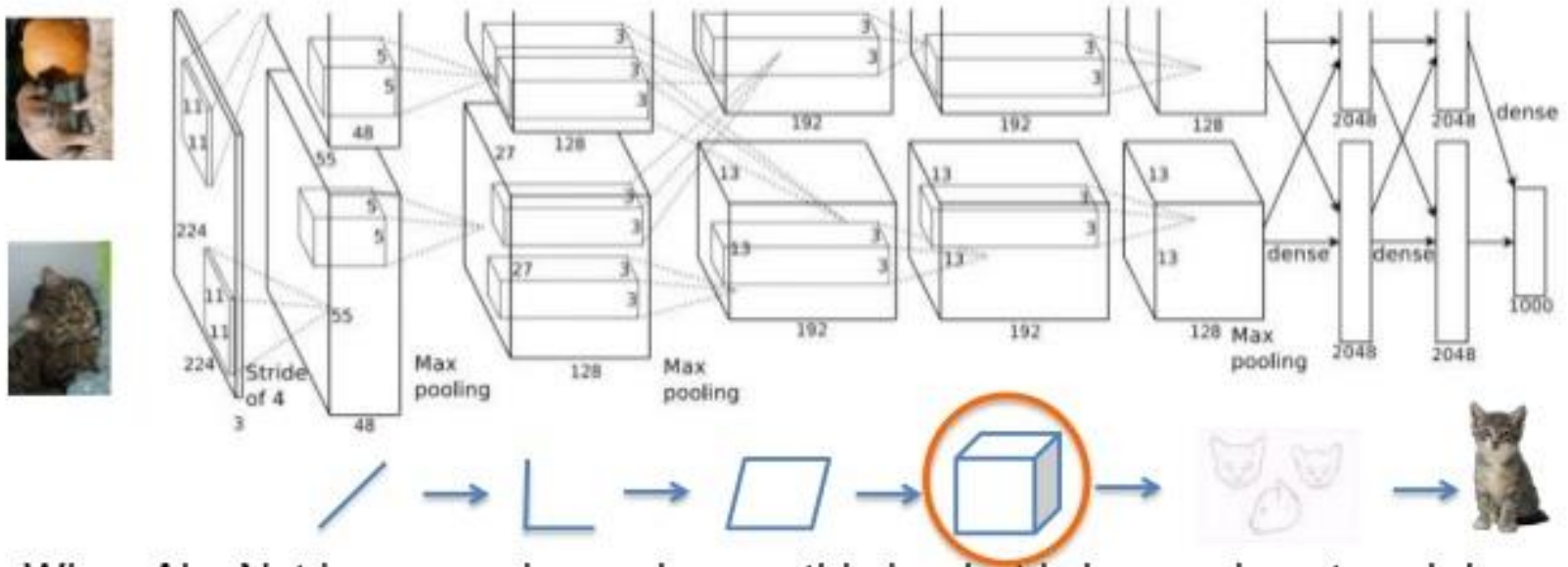
LeNet-5 architecture



Conv filters were 5x4, applied at stride 1 (C1)
Subsampling (Pooling) layers were 2x2 applied at stride 2
Finally there are 3 fully connected layers
[Conv-Pool-Conv-Pool-Conv-FC]

AlexNet architecture

- AlexNet (2012 winner) is still used today even though there are more accurate networks available, because of its relative simple structure and small depth. It is widely used in computer vision



When AlexNet is processing an image, this is what is happening at each layer.

Details/Retrospectives:
- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- SGD Momentum 0.9
- Learning rate 1e-2, reduced by 10 manually when val accuracy plateaus
- L2 weight decay 5e-4
- 7 CNN ensemble: 18.2% -> 15.4%

ILSVRC winners

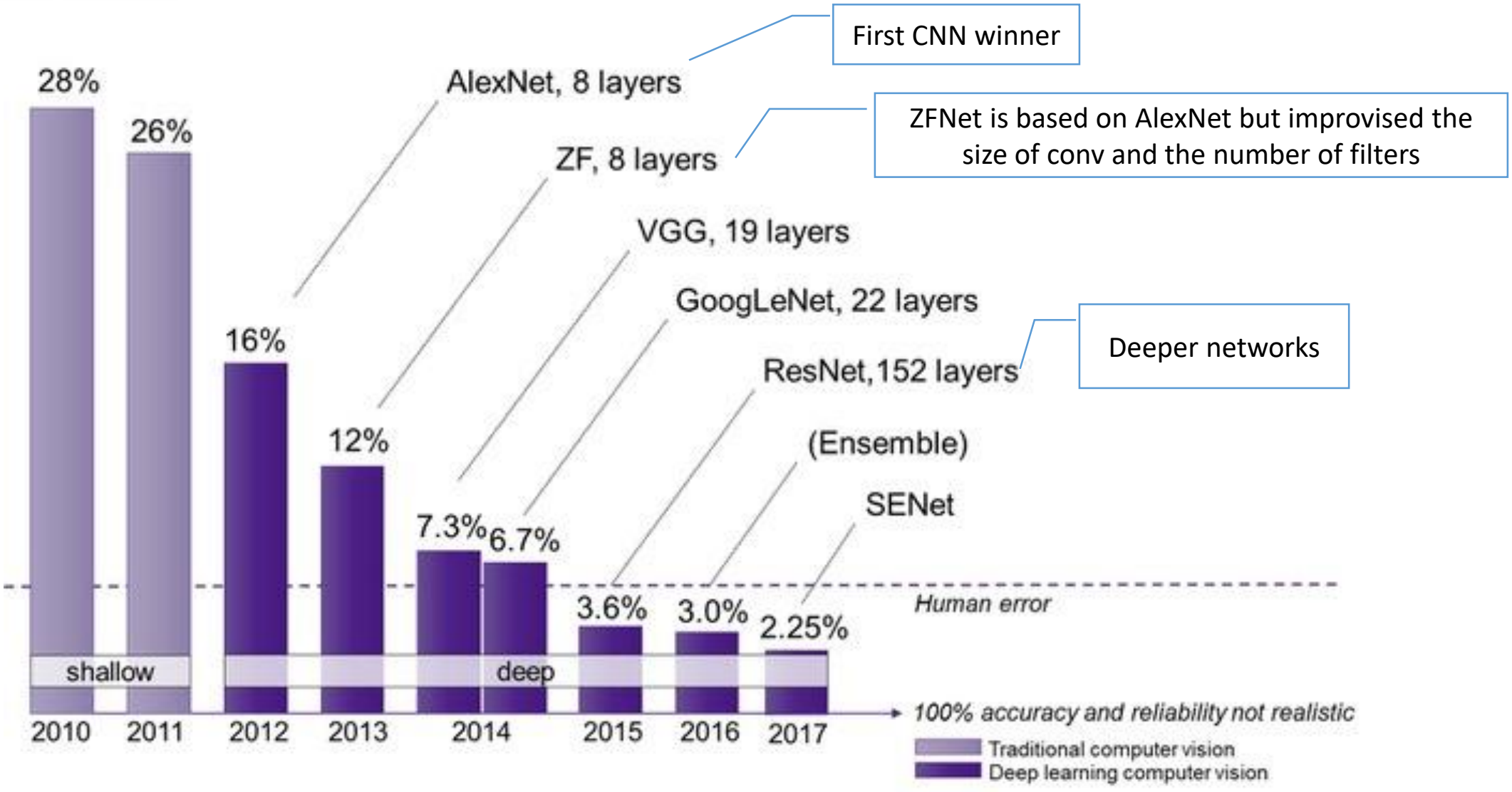
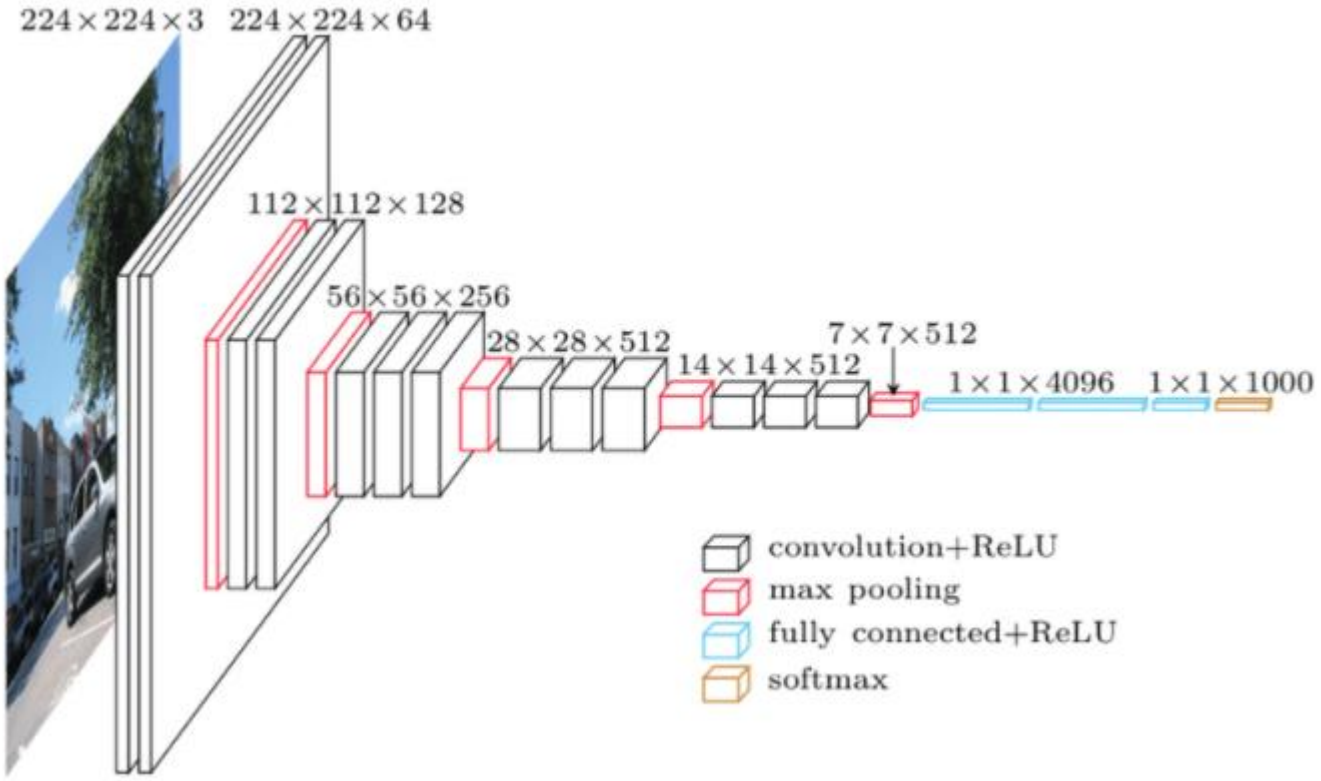


Figure copyright Gordon Cooper, 2019

VGGNet architecture



- VGG16 is a 16-layer neural network, not counting the max pooling layer and the softmax layer. Hence, it is known as VGG16.
- VGG19 consists of 19 layers.
- A pre-trained model is available in Keras for both Theano and TensorFlow backends.

VGGNet architecture

Case Study: VGGNet

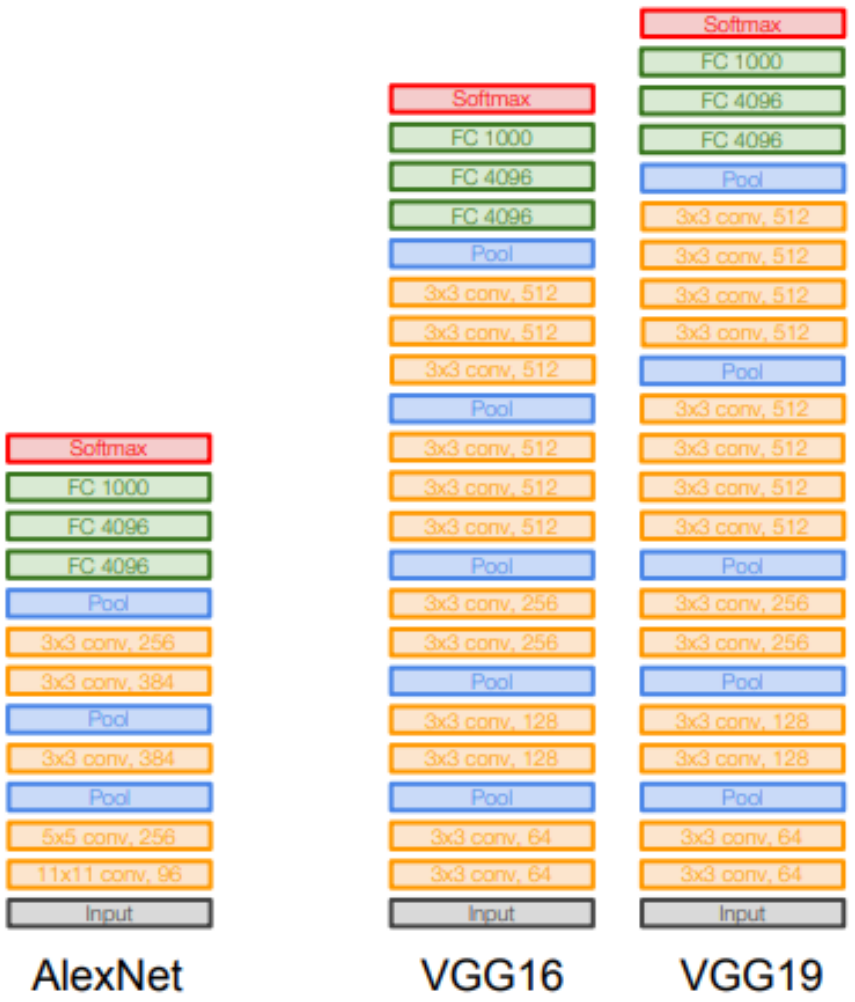
[Simonyan and Zisserman, 2014]

Small filters, Deeper networks

8 layers (AlexNet)
-> 16 - 19 layers (VGG16Net)

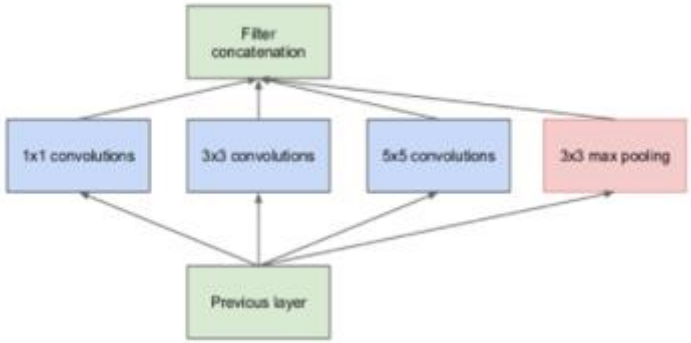
Only 3x3 CONV stride 1, pad 1
and 2x2 MAX POOL stride 2

11.7% top 5 error in ILSVRC'13
(ZFNet)
-> 7.3% top 5 error in ILSVRC'14

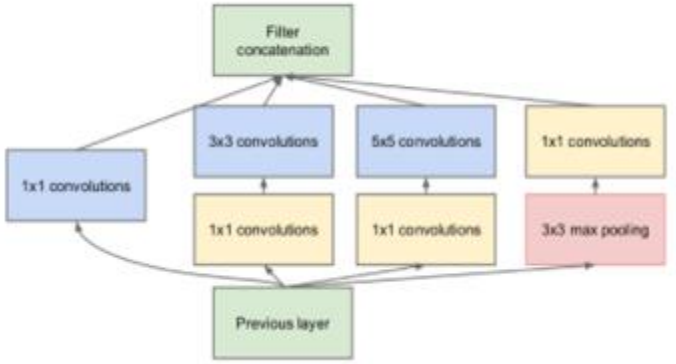


GoogLeNet architecture (2014)

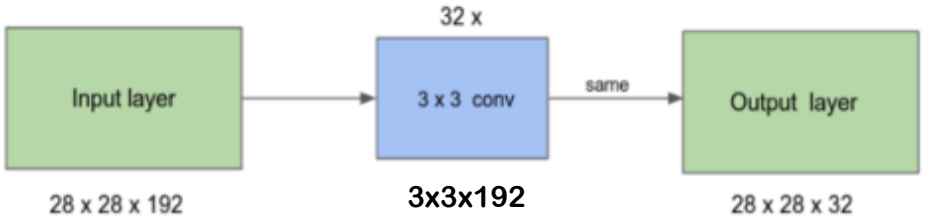
- In most of the architectures, the intuition is not clear why and when perform pooling and convolutional operation
  - AlexNet:1 conv+1 pooling
  - VGGNet: 3 convs+1 max pooling
- Inception model: a good local network topology (network within a network) and then stack these modules on top of each other
- It uses all the operations at the same time.It computes multiple kernels of different size over the same input map in parallel, concatenating their results into a single output.
- A pre-trained model is available in Keras for both Theano and TensorFlow backends.



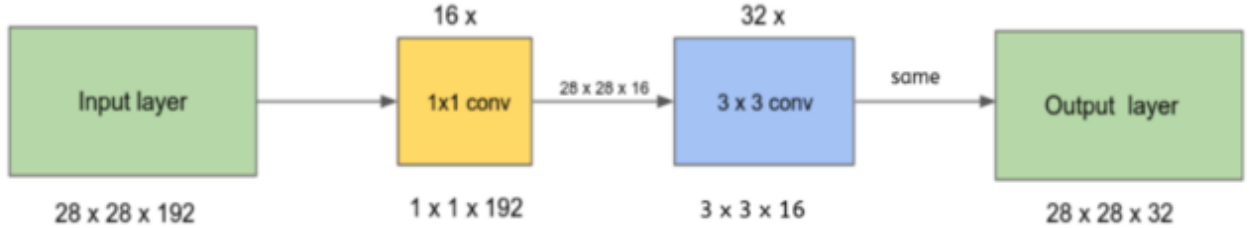
Q: What is the problem with this?



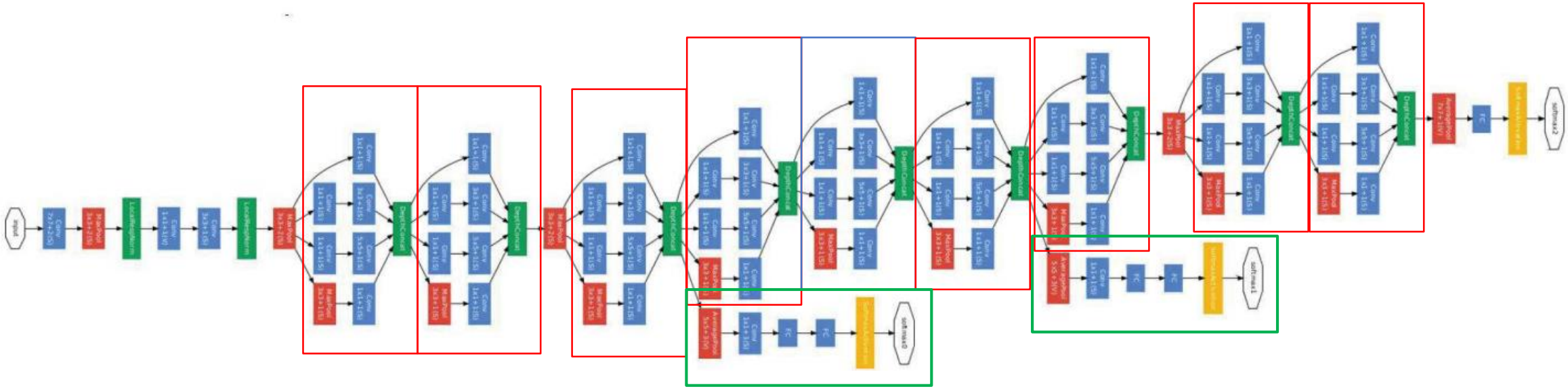
Inception module with dimension reduction



((28 x 28 x 3 x 3) x 192) x 32 ≃**43 M parameters**



((28 x 28 x 1 x 1) x 192) x 16 + ((28 x 28 x 3 x 3) x 16) x 32 ≃**5.4 M parameters**

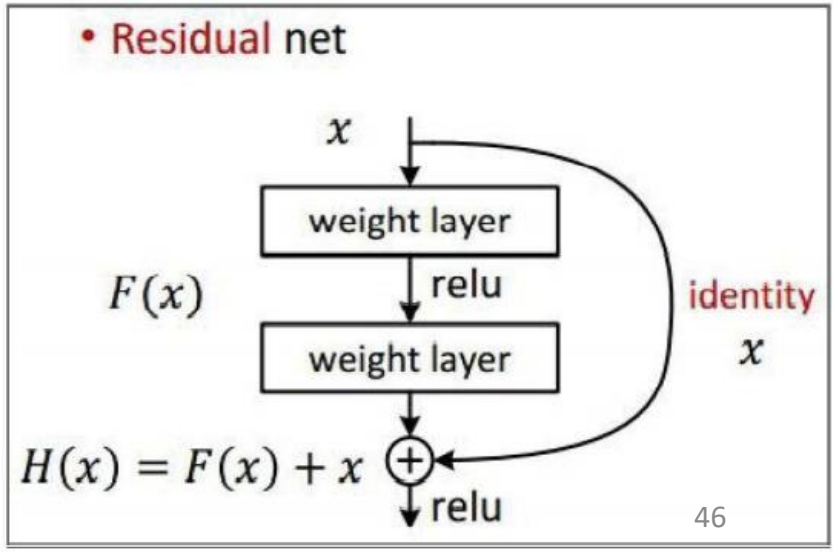


- Without expensive FC layers at the end. Instead, it use Global average pooling
- 9 inception modules
- 2 auxiliaries softmax layer (AvgPool – 1x1 Conv – FC-FC-Softmax)
  - Their role is to push the network toward its goal and helps to ensure that the intermediate features are good enough for the network to learn
  - It turns out that softmax0 and sofmax1 gives regularization effect.
  - During training, their loss gets added to the total loss with a discount weight (the losses of the auxiliary classifiers were weighted by 0.3).
  - During inference, they are discarded
- 22 total layers with weights

ResNet architecture (2015)

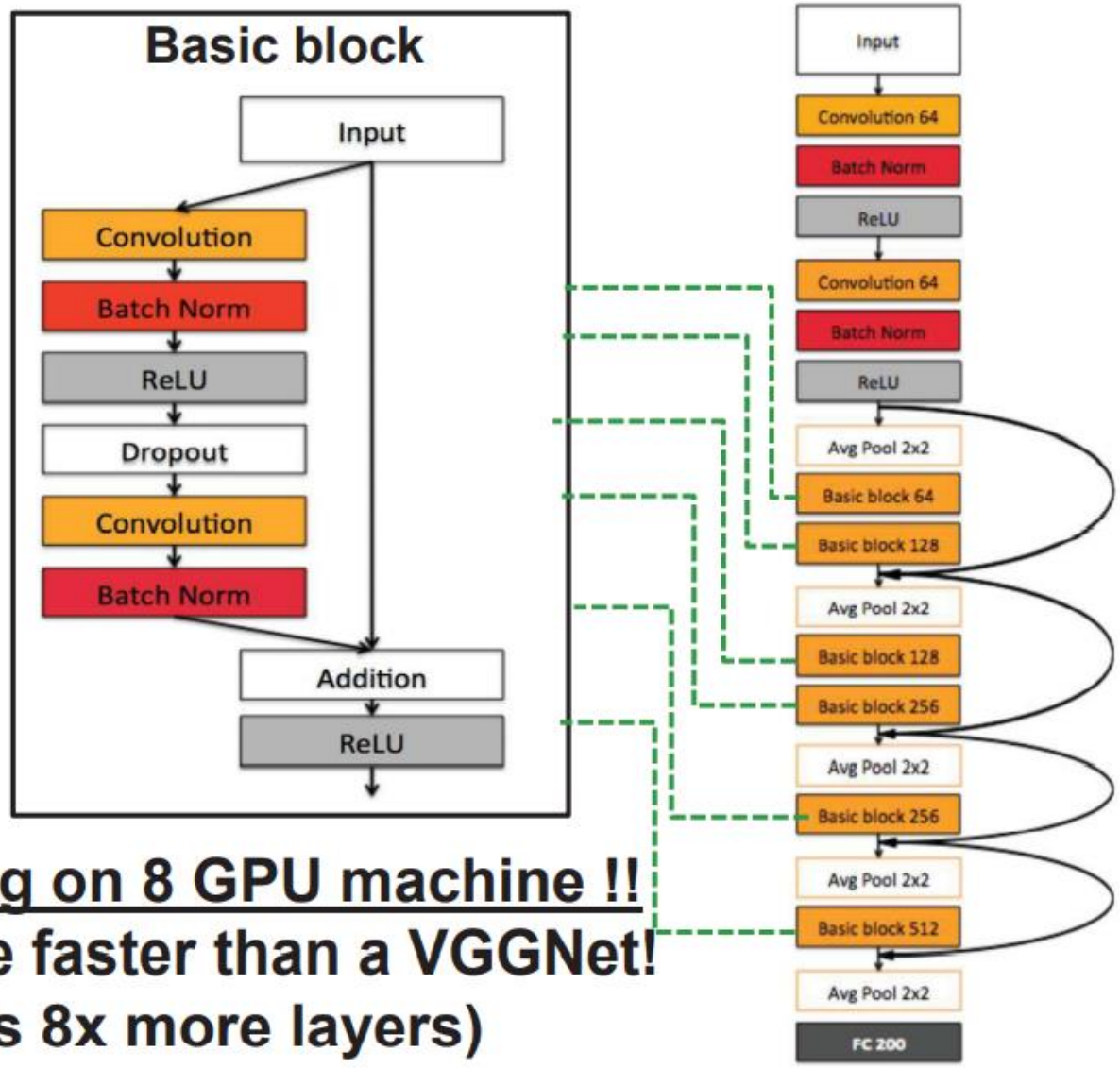
- After a certain depth, adding additional layers to feed-forward convNets results in a higher training error and higher validation error. When adding layers, performance increases only up to a certain depth, and then it rapidly decreases.
- In the **ResNet** (**Residual Network**) paper, the authors argued that this underfitting is unlikely due to the vanishing gradient problem, because this happens even when using the batch normalization technique.
- A new concept called **residual block** is proposed that the ResNet added connections that can skip layers
- Hypothesis: the problem is an optimization problem,

deeper models are harder to optimize



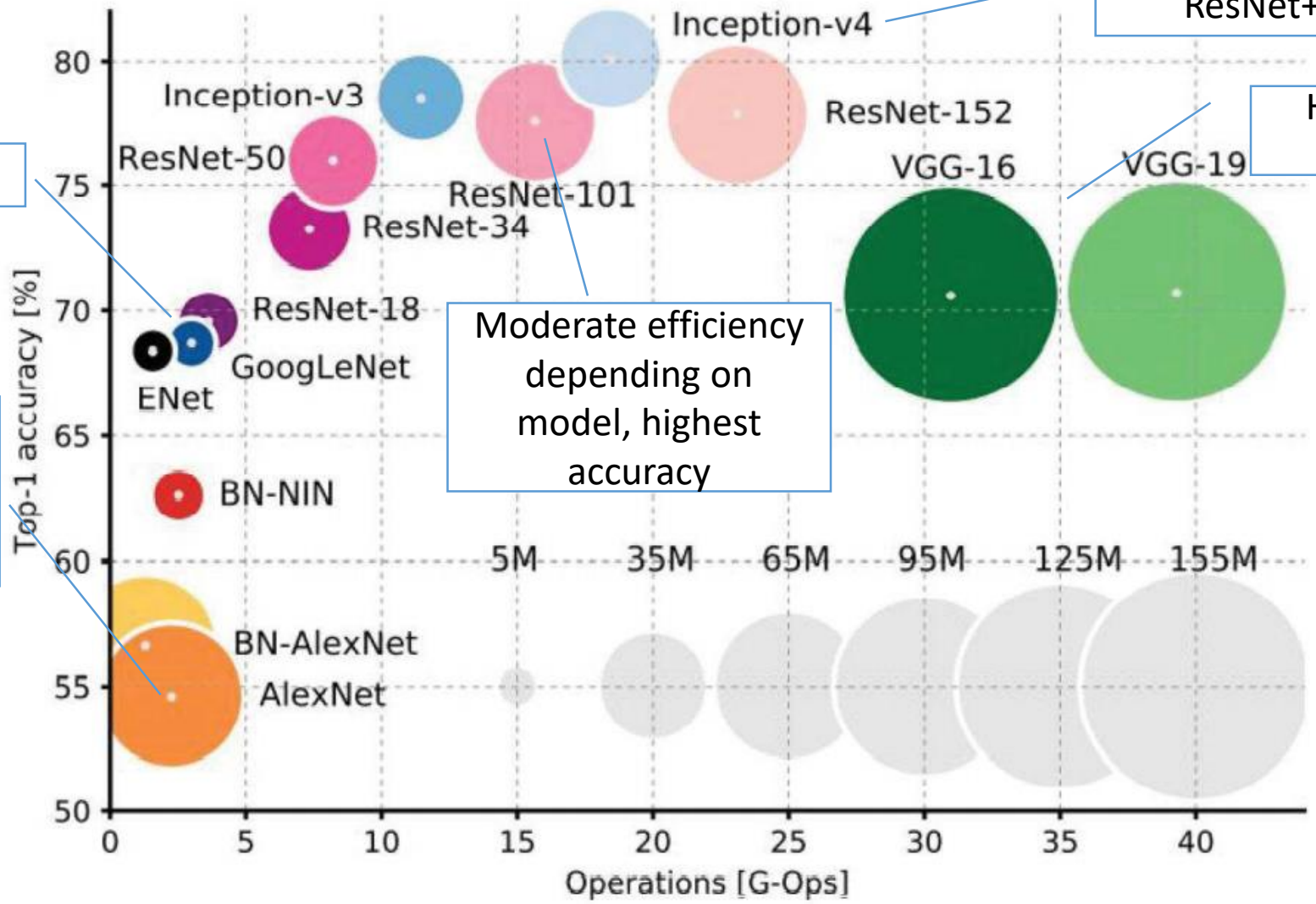
ResNet global architecture

- **152-layer module for ImageNet**
- **Swept all classification and detection competitions in ILSVRC15 and COCO15**



- **2-3 weeks of training on 8 GPU machine !!**
- **However, at runtime faster than a VGGNet! (even though it has 8x more layers)**

Performance comparison



ResNet+Inception!

Highest memory, most operation

Most efficient

Moderate efficiency depending on model, highest accuracy

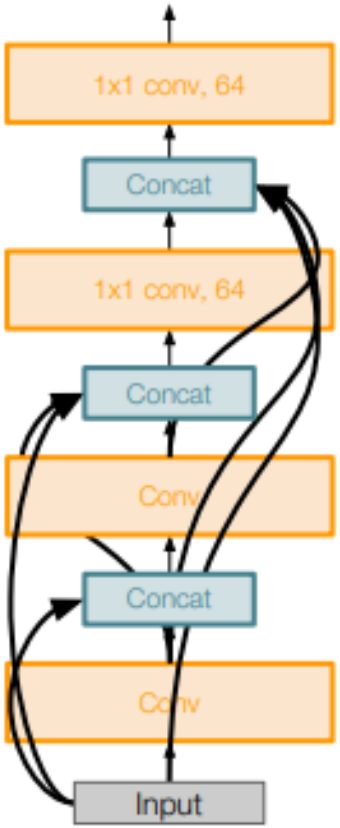
Smaller compute, still memory heavy, lower accuracy

Plots based on Convnets before 2015

Summary: CNN architecture

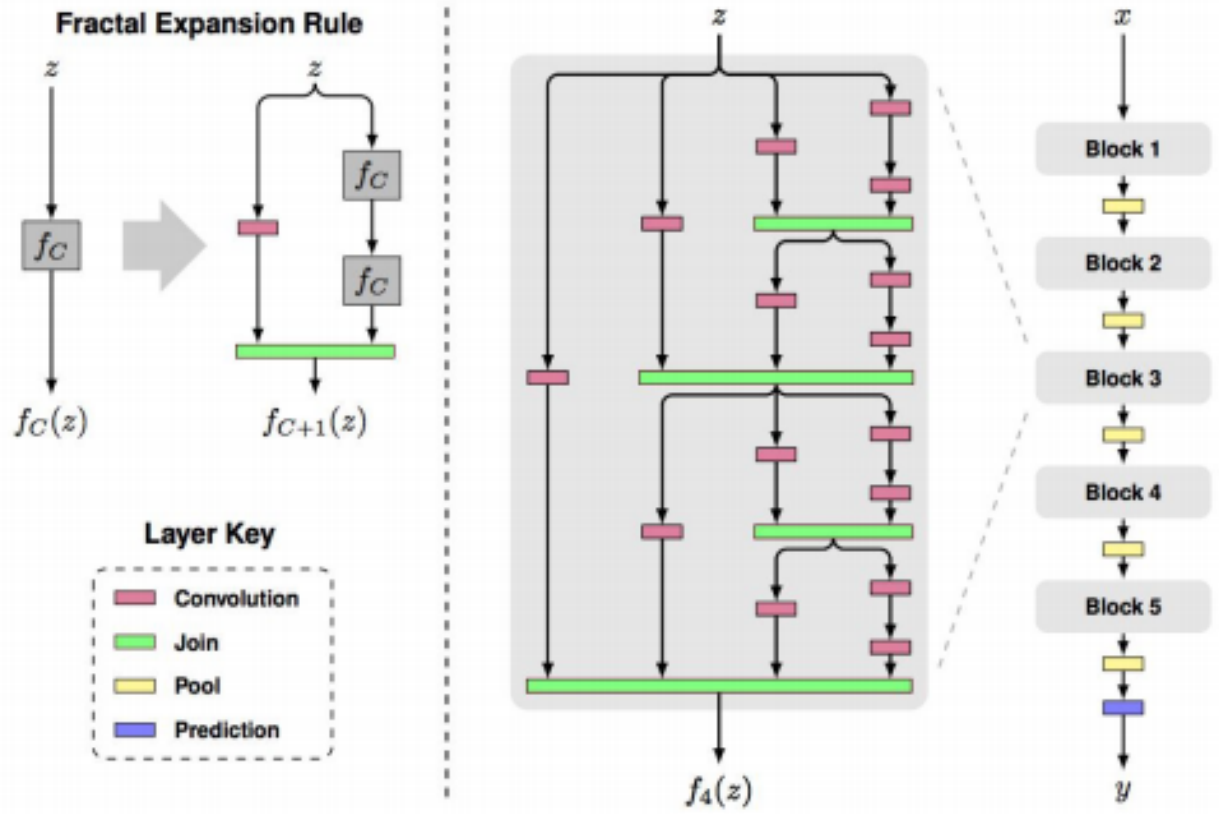
- VGG, GoogLeNet, ResNet all in wide use, available in model zoos
- ResNet current best default
- Trend towards extremely deep networks
- Significant research centers around design of layer / skip connections and improving gradient flow
- Even more recent trend towards examining necessity of depth vs. width and residual connections
- Other good architectures:
  - NiN [Lin et al. 2014]: (Network in Network. Insert MLP in between Conv. layer
  - Wide ResNet [azgoruyko et al. 2016]: User wider residual blocks (Fxk filters) is better
  - ResNeXT [Xie et al. 2016]: Wider residual blocks like inception module
  - Stochastic Depth [Huang et al. 2016]:randomly drop a subset of layers during training
  - DenseNet [Huang et al. 2017]: Dense blocks connected to every other layer in feedforward fashion
  - FractalNet [Larsson et al. 2017]: fusion of shallow and deep networks
  - SqueezeNet [Landola et al. 2017]: Fire modules consisting of squeeze and expand layers to increase the efficiency.
  - EfficientNet [M. Tan, and Q.Le, 2019]: compound the optimal search of scale by width, depth and resolution

DenseNet



Dense Block

FractalNet

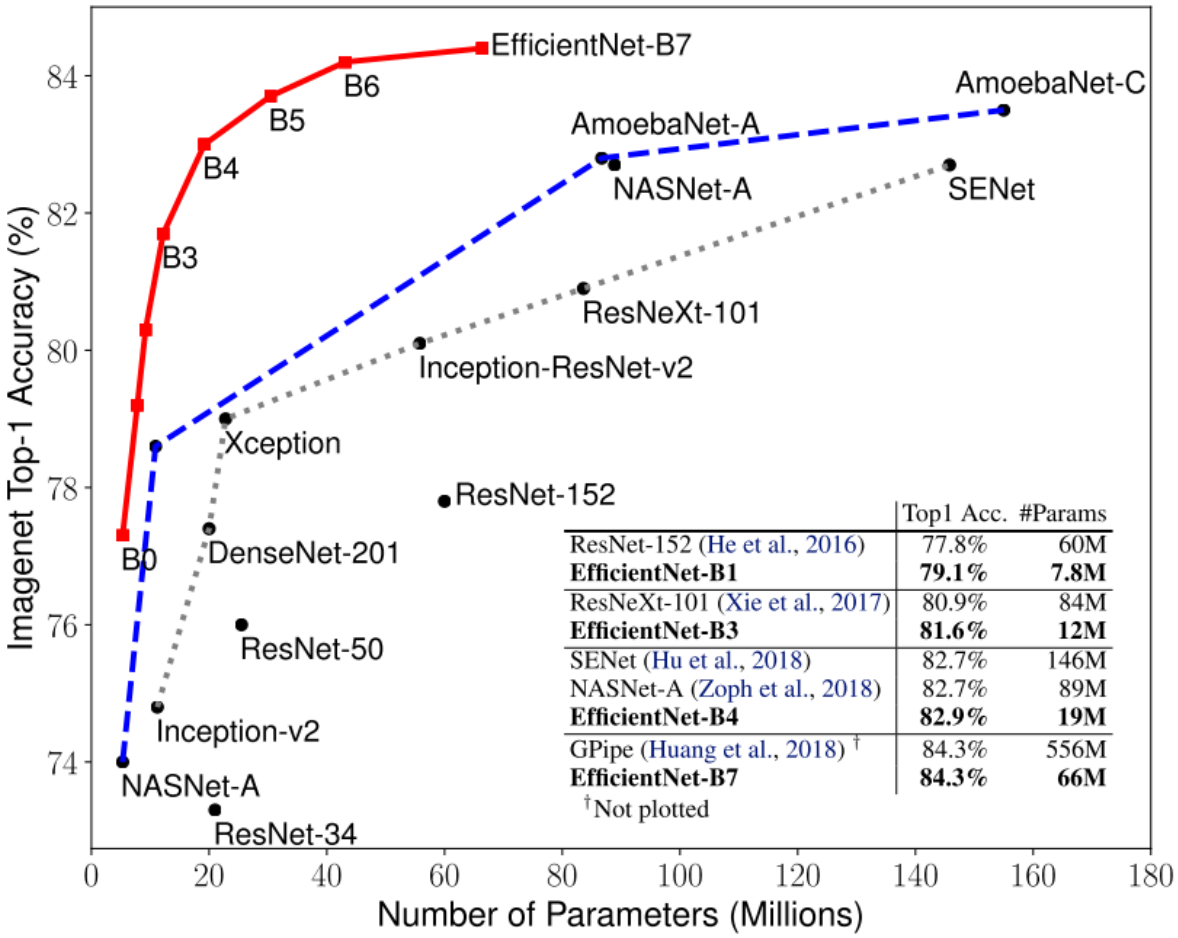


Figures copyright Larsson et al., 2017. Reproduced with permission.

Current layer receives signal from all the previous layers

Newest ConvNets (after 2019~)

- Structural ideas
  - EfficientNet (2019, Google Reasearch, Brain Team): optimal architecture search that uniformly scales all dimensions of depth/width/resolution. ➔ Top-1 accuracy 84.3% ImageNet
  - NFNets (2021): faster than EfficientNets, no normalization ➔ Top-1 accuracy 86.5% ImageNet



	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**79.1%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.6%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**82.9%**	**19M**
GPipe (Huang et al., 2018) †	84.3%	556M
EfficientNet-B7	**84.3%**	**66M**

†Not plotted

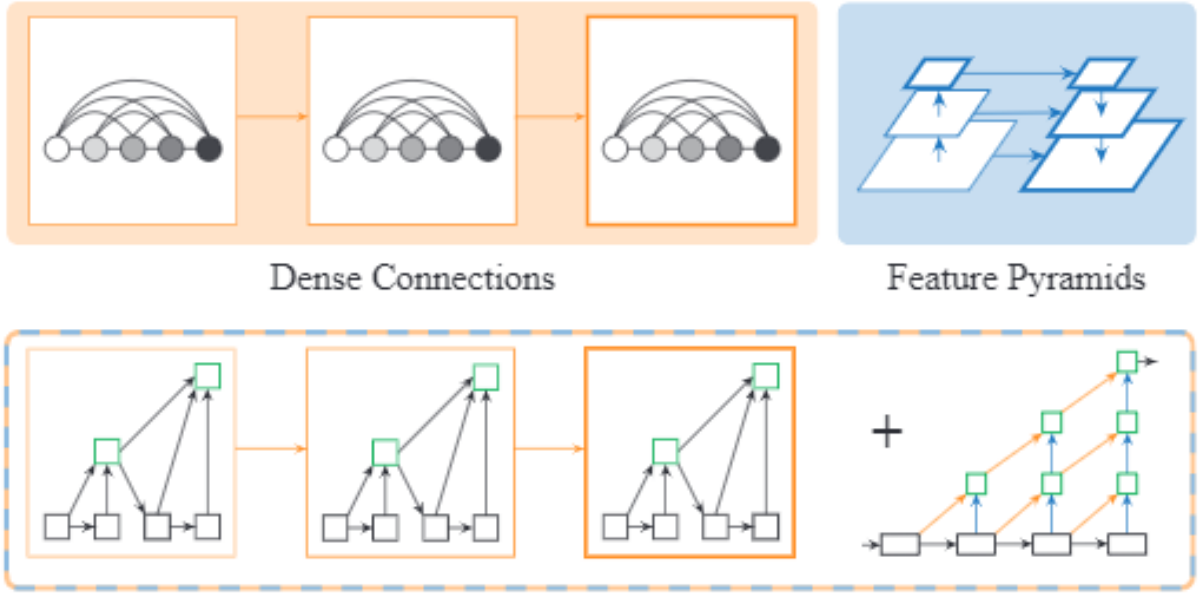
Tan, M., & Le, Q. V. (2019). EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks.

Other popular networks

- High way networks [R. K. Srivastava, K. Greff, and J. Schmidhuber. High- way networks. In NIPS, 2015]
- ResNeXT [S. Xie, R. Girshick, P. Doll´ar, Z. Tu, and K. He. Ag- gregated residual transformations for deep neural net- works. In CVPR, 2017]
- Networks-in-Networks [M. Lin, Q. Chen, and S. Yan. Network in network. In ICLR, 2014] demonstrated channel mixing as a technique to fuse features, control dimensionality, and go deeper.

Deep layer aggregation [2020]

- DLA unifies the spatial and semantic fusion to better encompass and extend densely connected networks and feature pyramid networks with hierarchical and iterative skip connections that deepen the representation and refine resolution



Original paper test it with convolution backbone, we can find deformable convolution version with DLA in FairMOT which is used to track human.

Conclusion of DLA

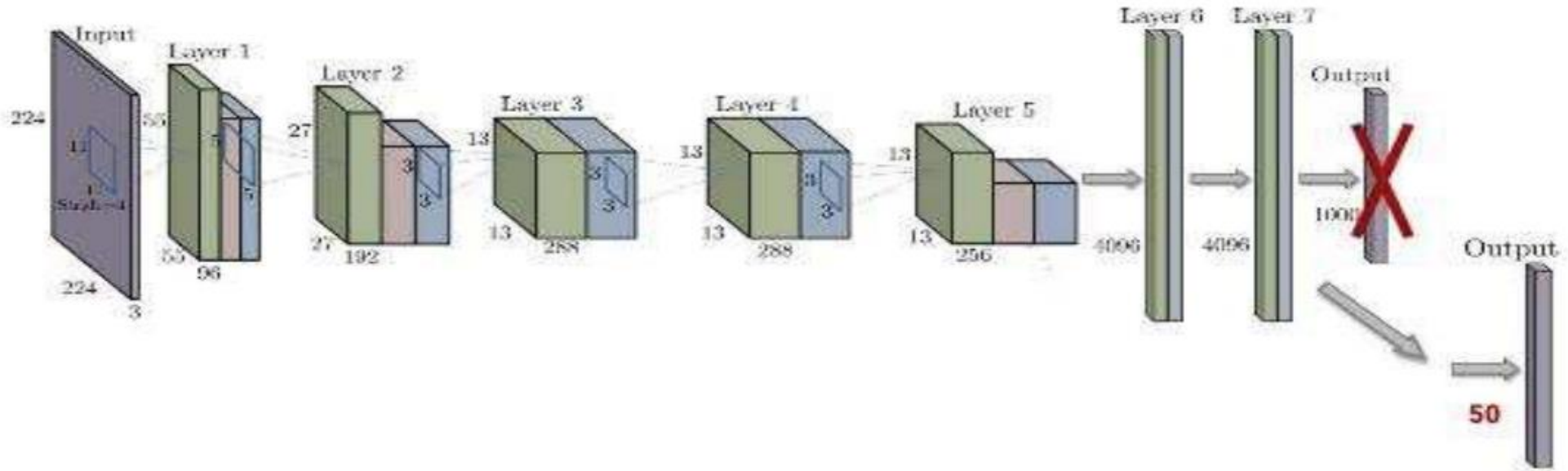
- For the most part these architectures derive from innovations in connectivity: skipping, gating, branching, and aggregating
- Aggregation is a decisive aspect of architecture, and as the number of modules multiply their connectivity is made all the more important.
- By relating architectures for aggregating channels, scales, and resolutions we identified the need for deeper aggregation, and addressed it by iterative deep aggregation and hierarchical deep aggregation.
- Our models are more accurate and make more efficient use of parameters and computation than baseline networks. Our aggregation extensions improve on dominant architectures like residual and densely connected networks.
- Bridging the gaps of architecture makes better use of layers in aggregate

Programming environments

- TensorFlow https://www.tensorflow.org
- KERAS https://keras.io
  - Python front-end APIs mapped either on Tensor-Flow or Theano back-end
- PyTorch https://pytorch.org/
- Caffe http://caffe.berkeleyvision.org/
  - C++ library, hooks from Python ➔ notebooks
- Theano http://www.deeplearning.net/software/theano/
- Lasagne http://lasagne.readthedocs.io
  - lightweight library to build+train neural nets in Theano

All of them handle transparent use of GPU, and most of them are used in Python code/notebook

Transfer learning



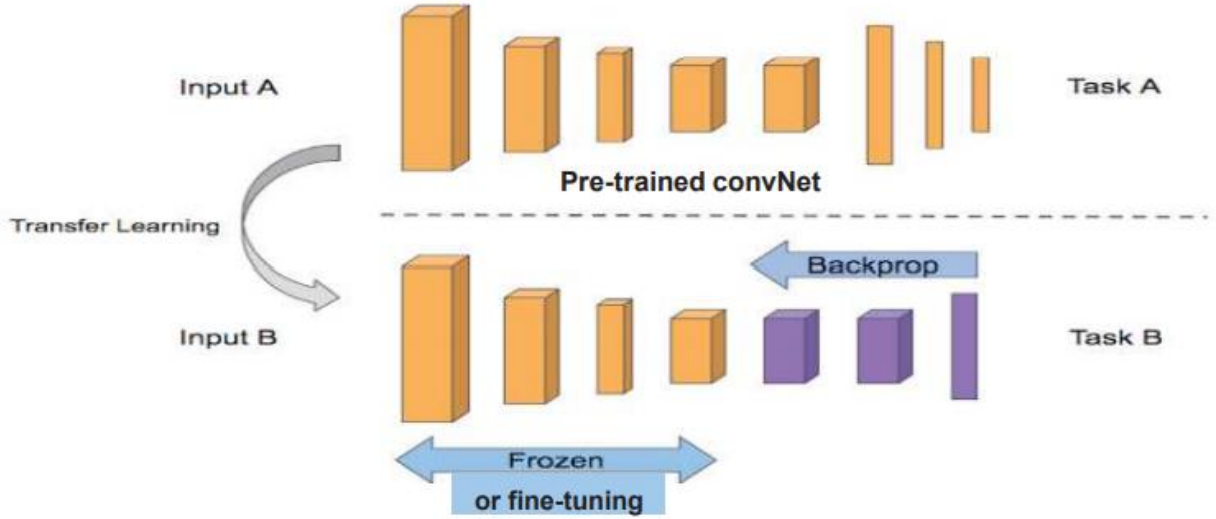
By removing *last layer(s)* (those for classification) of a convNet trained on ImageNet, one obtains a *transformation of any input image into a semi-abstract representation,* which can be used for learning SOMETHING ELSE (« transfer learning »):

- **either by just using learnt representation as features**
- **or by creating new convNet output and perform learning of new output layers + fine-tuning of re-used layers**

Transfer learning and fine-tuning

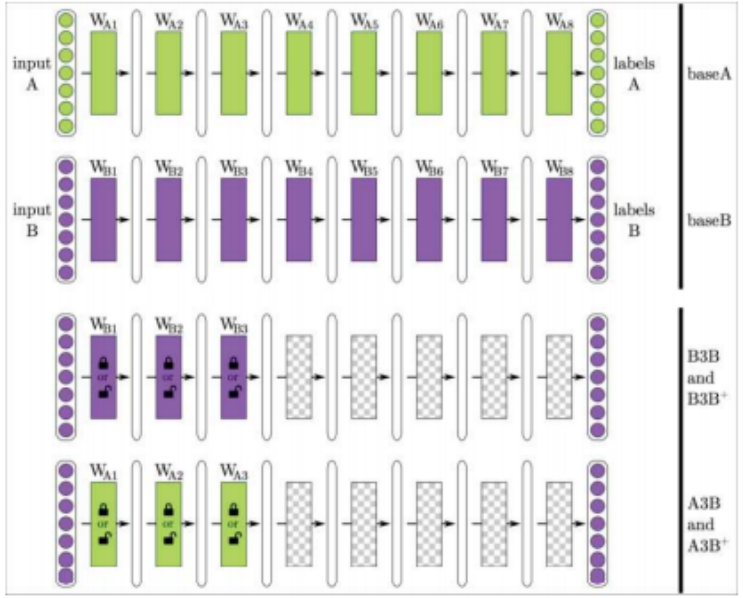
- **SoA convNets trained on ImageNet are <u>image CLASSIFIERS</u> for one object per image**
- **Many object categories can be irrelevant (e.g. boat in a office)**

➔ **For each application, models are usually obtained from state-of-the-art ConvNets pre-trained on ImageNet (winners of yearly challenge, eg: AlexNet, VGG, Inception, ResNet, etc…)**

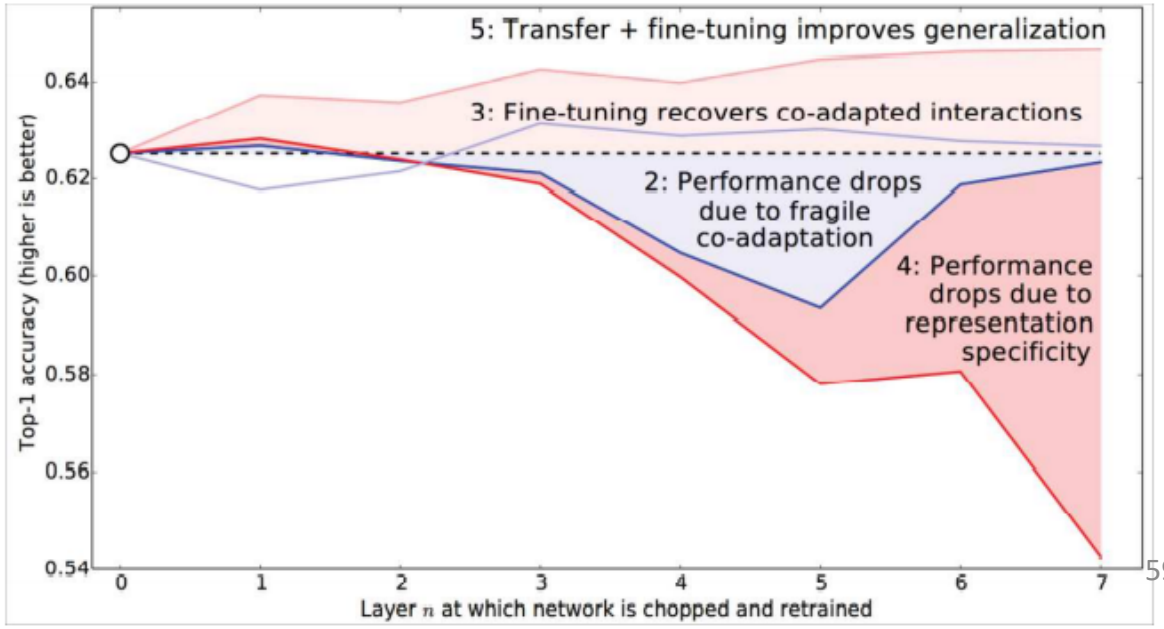


➔ **Adaptation is performed by <u>*Transfer Learning*</u>, ie modification+training of last layers and/or fine-tuning of pre-trained weights of lower layers**

Transfer-Learning even improves performances



[Yosinski, Clune, Bengio, Lipson, "How transferable are features in deep neural networks?", ICML'2014]



Some transfer-learning applications

- Learning on simulated synthetic images + fine-tuning on real-world images
- Recognition/classification for OTHER categories or classes
- Training an objects detector (or a semantic segmenter)
- Precise localization (position+bearing) = PoseNet
- Human posture estimation = openPose
- End-to-end driving (imitation Learning)
- 3D informations (depth map) from monovision!

Summary on ConvNets & Deep-Learning

- Proven advantage of learning features empirically from data
- Large ConvNets require huge amounts of labelled examples data for training
- Current research/progresses = finding efficient global architecture of ConvNets
- Enormous potential of TRANSFER-LEARNING on small datasets for restricted/specialized problems
- ConvNets also for multivariate time-series (1D temporal convolutions) and for 3D data (3D conv on voxels, etc…)
- ConvNets can potentially infer from image ANYTHING for which information is in the image (3D, movement, planning, …)