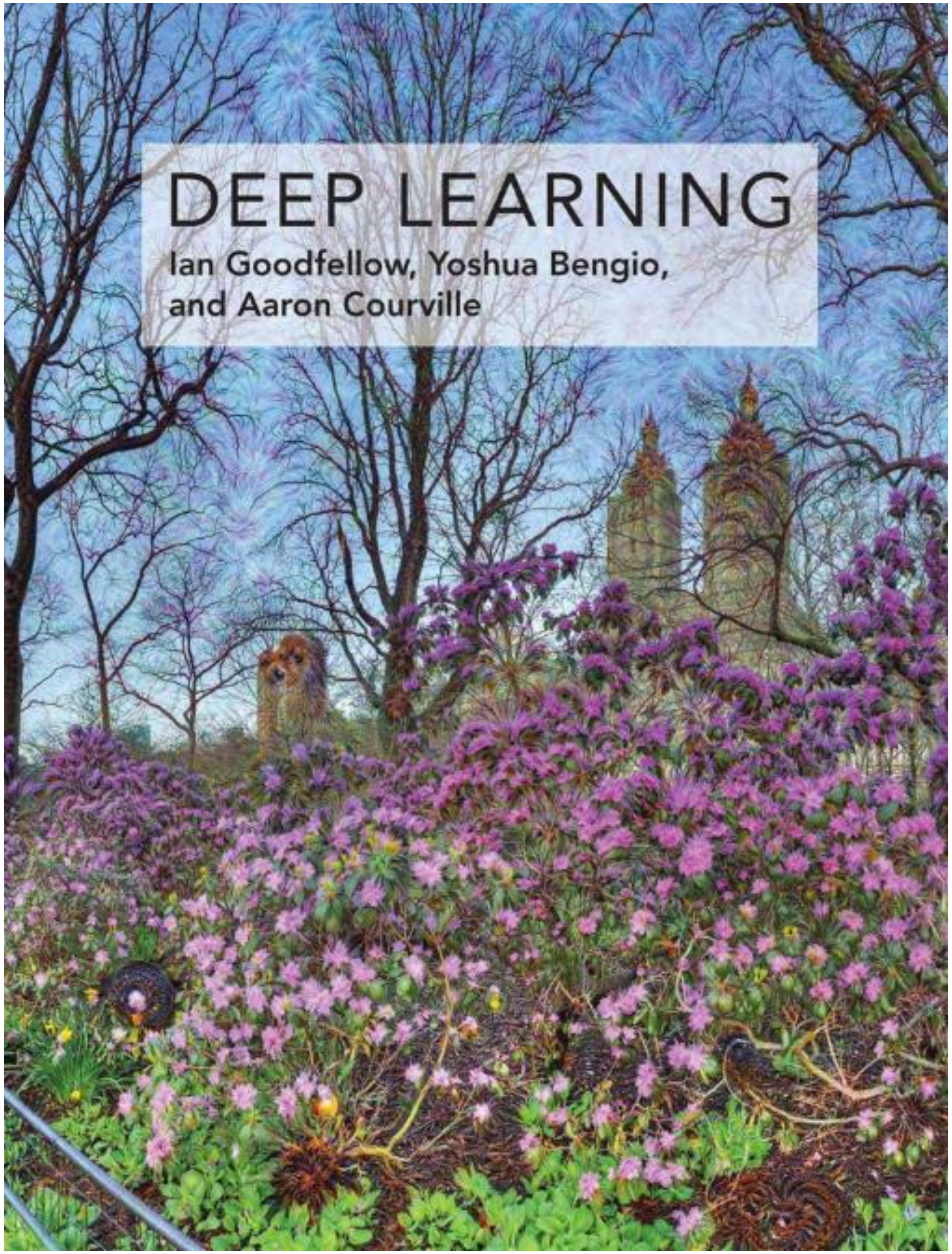
Fundamentals of Media Processing (Machine Learning Part)

Lecturer:
佐藤　真一（Prof. SATO　Shinichi）
<u>池畑　諭（Prof. IKEHATA　Satoshi）10/27, 11/10, 11/17, 11/24, 12/1, 12/8</u>
山岸　順一（Prof. Junichi Yamagishi）
児玉　和也（Prof. KODAMA　Kazuya）
孟　洋（Prof. MO　Hiroshi）



Chapter 1-9 (out of 20)

An introduction to a broad range of topics in deep learning, covering mathematical and conceptual background, deep learning techniques used in industry, and research perspectives.

- Due to my background, I will mainly talk about “image”
- I will introduce some applications beyond this book

Deep Learning

An MIT Press book in preparation

Ian Goodfellow, Yoshua Bengio and Aaron Courville

Book Exercises External Links

Lectures

We plan to offer lecture slides accompanying all chapters of this book. We currently offer slides for only some chapters. If you are a course instructor and have your own lecture slides that are relevant, feel free to contact us if you would like to have your slides linked or mirrored from this site.

1. Introduction
   - Presentation of Chapter 1, based on figures from the book [.key] [.pdf]
   - **Video** of lecture by Ian and discussion of Chapter 1 at a reading group in San Francisco organized by Alena Kruchkova
2. Linear Algebra [.key][.pdf]
3. Probability and Information Theory [.key][.pdf]
4. Numerical Computation [.key] [.pdf] [youtube]
5. Machine Learning Basics [.key] [.pdf]
6. Deep Feedforward Networks [.key] [.pdf]
   - Video (.flv) of a presentation by Ian and a group discussion at a reading group at Google organized by Chintan Kaur.
7. Regularization for Deep Learning [.pdf] [.key]
8. Optimization for Training Deep Models
   - **Gradient Descent and Structure of Neural Network Cost Functions** [.key] [.pdf]
     These slides describe how gradient descent behaves on different kinds of cost function surfaces. Intuition for the structure of the cost function can be built by examining a second-order Taylor series approximation of the cost function. This quadratic function can give rise to issues such as poor conditioning and saddle points. Visualization of neural network cost functions shows how these and some other geometric features of neural

Free copy of the book and useful materials are available at

https://www.deeplearningbook.org/lecture_slides.html

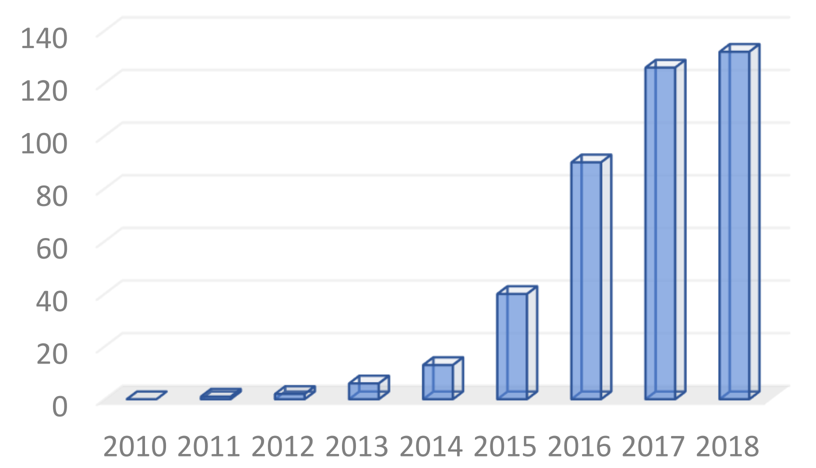
Schedule

10/27 (Today)	Introduction	Chap. 1
	probability, information theory, numerical computation	Chap. 2,3,4
11/10	Machine Learning Basics	Chap. 5
11/17, 11/24, 12/1	Deep Feedforward Networks	Chap. 6
	Regularization and Deep Learning	Chap. 7
	Optimization for Training Deep Models	Chap. 8
12/8	Convolutional Neural Networks	Chap. 9 and more

Convolutional Neural Networks

History of Convolutional Neural Networks

- 1990s, the neural network research group at AT & T developed a convolutional network for reading checks (LeCun1998)
- Several OCR and handwriting recognition systems based on CNN were deployed by Microsoft (Simard2003)
- AlexNet (2012) won the ImageNet object recognition challenge, and the current intensity of commercial interest in deep learning began



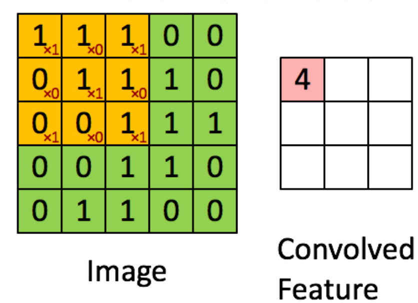
# of Papers with “Deep” in CVPR

Convolutional Neural Networks (1)

- ***Convolutional Neural networks*** (CNN; LeCun1989) are a neural network for processing data of gild-like structure. The major examples include ***image data***
- CNN are simply neural networks that use ***convolution*** in piece of general matrix multiplication in at least one of their layers. In general convolution, the kernel is **flipped**, but in neural networks, it does not matter since the kernel itself is learned
- The multichannel convolutional operations requires that *the input and output of the convolution have same channels* to make the convolution commutative; In reality, what CNN do is *cross correlation* rather than convolution

$$S(i,j) = (I * K)(i,j) = \sum_{m}\sum_{n} I(i+m, i+n)K(i,j)$$

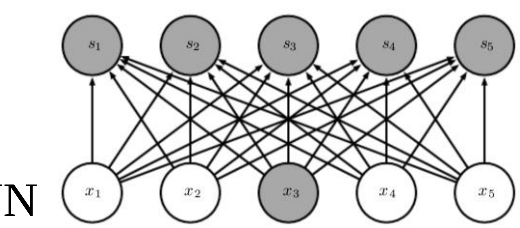
Cross correlation based on the commutative property in convolution

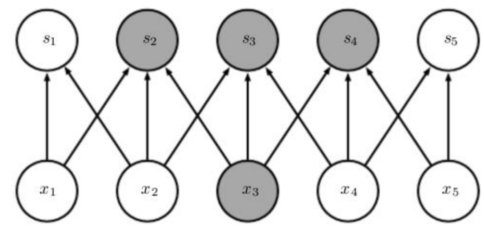


https://cdn-images-1.medium.com/max/1600/1*ZCjPUFrB6eHPRi4eyP6aaA.gif

Convolutional Neural Networks (2)

- CNN leverages three important ideas
  - Sparse interactions
    - Each unit is interacted with smaller number of units

Standard NN 



CNN

  - Parameter sharing
    - Traditional neural net→ dense multiplication ($y = W\boldsymbol{x}$)
    - We only need few parameters (# of kernel × size of kernel)
  - Equivariant to translation
    - $f\big(g(x)\big) = g(f(x))$
    - “translation then convolution” is same with “convolution, then translation”
    - CNN is not naturally equivalent to rotation and scale

Convolutional Neural Networks (3)

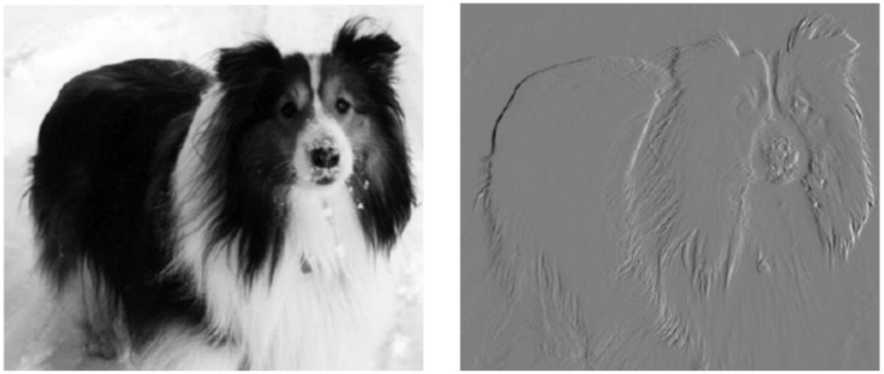
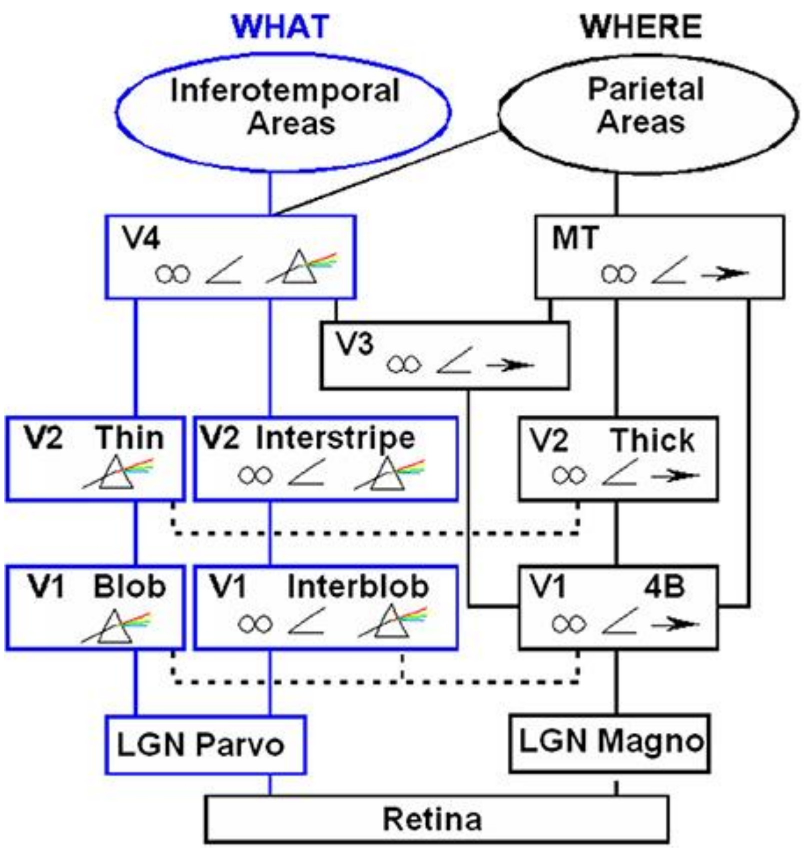
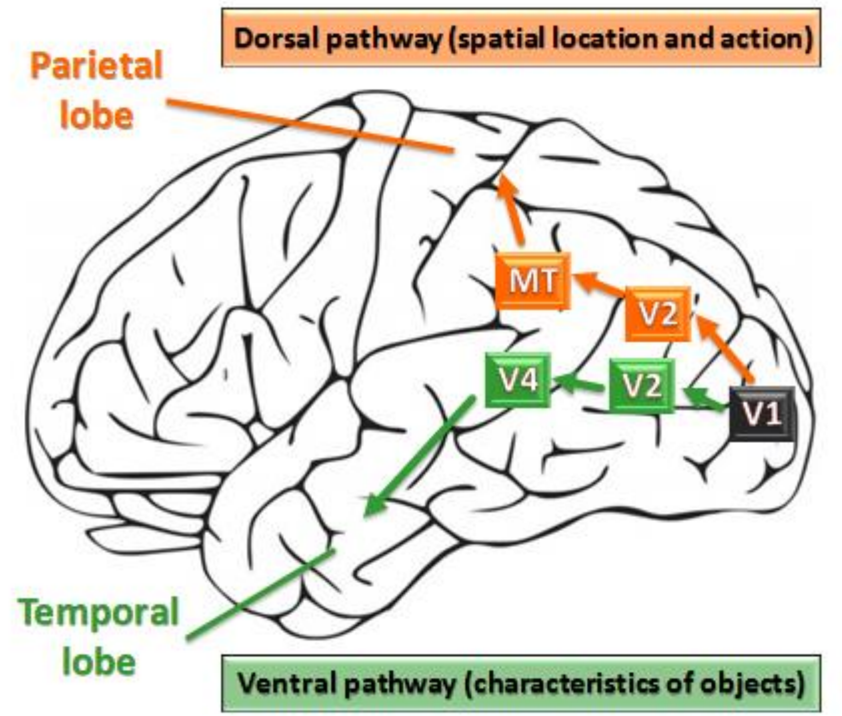


Figure 9.6: Efficiency of edge detection. The image on the right was formed by taking each pixel in the original image and subtracting the value of its neighboring pixel on the left. This shows the strength of all the vertically oriented edges in the input image, which can be a useful operation for object detection. Both images are 280 pixels tall. The input image is 320 pixels wide, while the output image is 319 pixels wide. This transformation can be described by a convolution kernel containing two elements, and requires $319 \times 280 \times 3 = 267,960$ floating-point operations (two multiplications and one addition per output pixel) to compute using convolution. To describe the same transformation with a matrix multiplication would take $320 \times 280 \times 319 \times 280$, or over eight billion, entries in the matrix, making convolution four billion times more efficient for representing this transformation. The straightforward matrix multiplication algorithm performs over sixteen billion floating point operations, making convolution roughly 60,000 times more efficient computationally. Of course, most of the entries of the matrix would be zero. If we stored only the nonzero entries of the matrix, then both matrix multiplication and convolution would require the same number of floating-point operations to compute. The matrix would still need to contain $2 \times 319 \times 280 = 178,640$ entries. Convolution is an extremely efficient way of describing transformations that apply the same linear transformation of a small local region across the entire input. Photo credit: Paula Goodfellow.

CNN and Neuroscience (1)



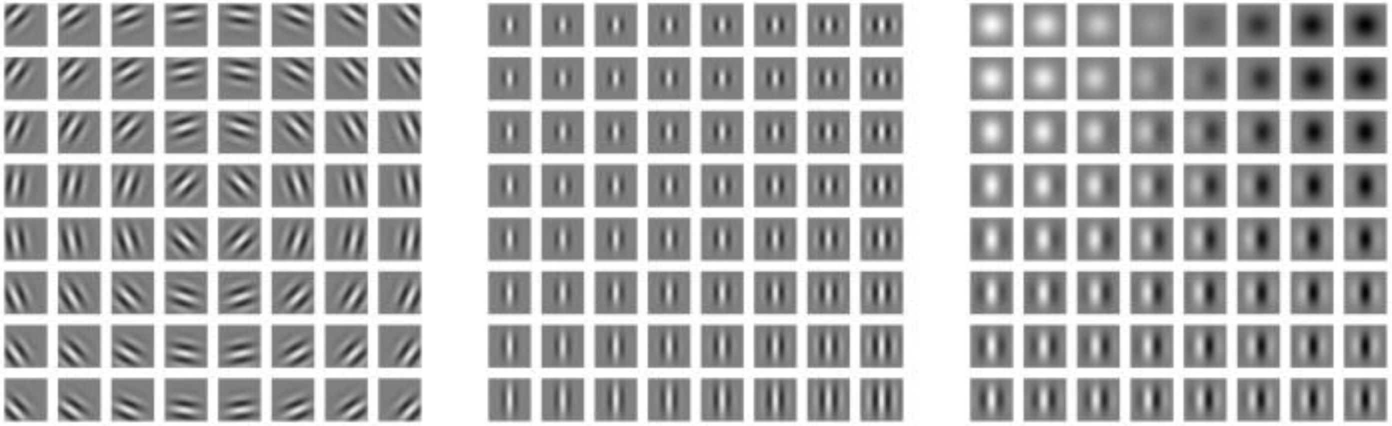
https://www.researchgate.net/figure/Schematic-diagram-of-anatomical-connections-and-neuronal-selectivities-of-early-visual_fig15_268228820

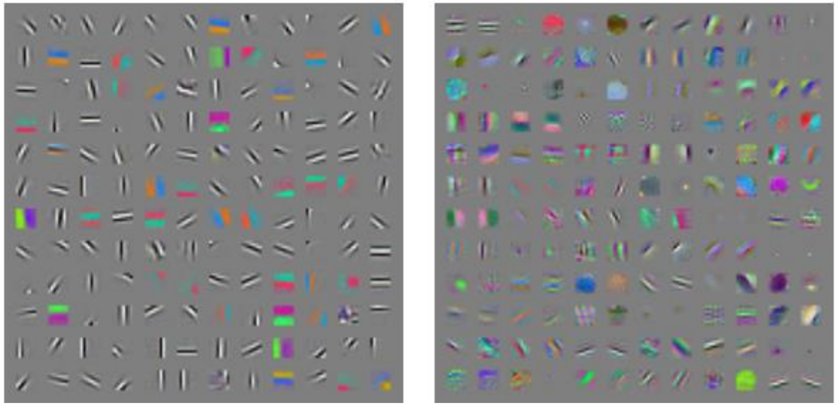


https://www.intechopen.com/books/visual-cortex-current-status-and-perspectives/adaptation-and-neuronal-network-in-visual-cortex

CNN and Neuroscience (2)

- V1 cells have weights that are described by Gabor functions that prefers the specific direction of edges





Feature maps learned by CNN

Example of the CNN

- Typical convolutional neural networks consist of ***convolution***, ***pooling***, and ***fully-connected*** layers

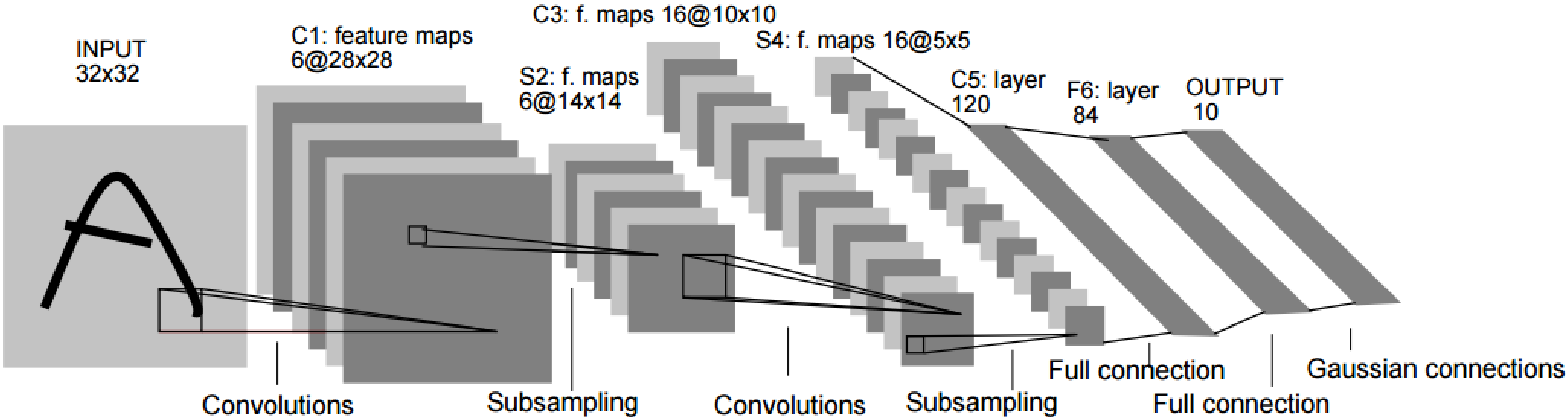


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

Pooling (1)

- A pooling function replaces the output of the net at a certain location with a summary statistic of the nearby outputs
  - ***Max pooling***: the maximum within a rectangular neighborhood
  - ***Average pooling***: the mean within a rectangular neighborhood

- Pooling encourages the network to learn the ***invariance*** to small translations of the input

- For many tasks, pooling is essential *for handling inputs of varying size* (varying the size of an offset between pooling regions so that the final output layer always receives the same number of summary statistics regardless of the input size)

12	20	30	0
8	12	2	0
34	70	37	4
112	100	25	12

20	30
112	37

Max pooling

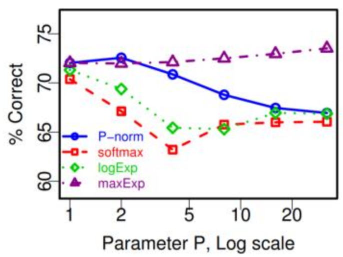
13	8
79	20

Average pooling

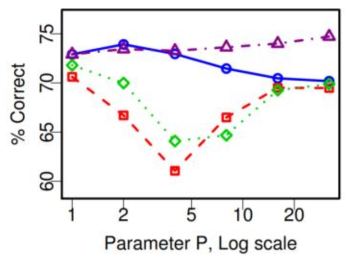
Pooling (2)

- Boureau2010 (mentioned in image classification task):

"Depending on the data and features, either max or average pooling may perform best. The optimal pooling type for a given classification problem may be neither max nor average pooling"

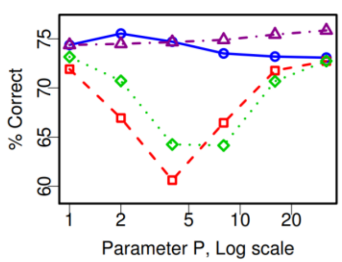


(a) 128 codewords

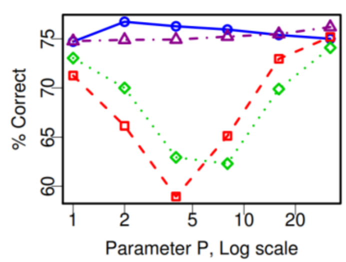


(b) 256 codewords

Average → Max



(c) 512 codewords



(d) 1024 codewords

Smallest cardinality		1024	512	256
Caltech 101	Avg, One	$\mathbf{32.4 \pm 1.1}$	31.3 ± 1.0	28.6 ± 1.1
	Avg, Joint		31.9 ± 1.2	32.1 ± 1.2
	Max, One	31.7 ± 1.4	32.7 ± 1.3	30.4 ± 2.3
	Max, Joint		34.4 ± 0.7	$\mathbf{35.8 \pm 0.9}$
	SM, One	37.9 ± 0.6	40.5 ± 0.7	$\mathbf{42.0 \pm 1.4}$
	SM, Joint		39.4 ± 1.3	40.6 ± 0.8
15 Scenes	Avg, One	$\mathbf{69.8 \pm 0.7}$	68.7 ± 0.8	66.3 ± 0.7
	Avg, Joint		69.6 ± 0.7	69.2 ± 1.0
	Max, One	63.5 ± 0.6	64.8 ± 0.7	64.3 ± 0.4
	Max, Joint		65.4 ± 0.6	$\mathbf{67.1 \pm 0.6}$
	SM, One	67.2 ± 0.8	70.4 ± 0.7	$\mathbf{72.6 \pm 0.7}$
	SM, Joint		69.2 ± 0.7	70.7 ± 0.7

Pooling (3)

- An infinitely strong prior places zero probability on some parameters and says these parameter values are completely forbidden. We can imagine CNN as being similar to a fully connected net but with an infinitely strong prior over its weights (e.g., translation invariance) and without some priors in standard neural network (e.g., ***permutation invariance***)

- *Convolution and pooling can cause underfitting*. If a task relies on preserving precise spatial information, then using pooling on all features can increase the training error. Some CNN therefore uses pooling on some specific channels (Szegedy2014) in order to get both highly invariant features and features that will not underfit when the translation invariance prior is incorrect

Variance of the Basic Convolution Function (1)

- The convolution function used in CNN and the standard discrete convolution operation is usually different
  - The convolution in CNN is an operation that consists of many applications of convolution in parallel to extract many kinds of features at many locations
  - The input and output are grid of vector-valued observations (i.e., 3-D tensors; e.g., RGB image)

- ***Stride***: We may want to skip over some positions of the kernel to reduce the computational cost. We can define a downsampled convolution function with stride as

$$Z_{i,j,k} = \sum_{l,m,n} V_{l,j+m-1,k+n-1} K_{i,l,m,n}$$

i: output channel j: offset of rows
l: input channel k: offset of columns

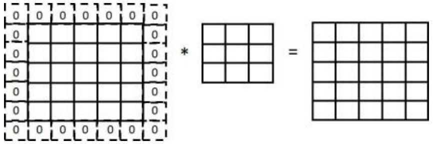
$$Z_{i,j,k} = \sum_{l,m,n} V_{l,(j-1)*s+m,(k-1)*s+n} K_{i,l,m,n}$$

Downsampled convolution with stride (s)

It is more common to firstly apply conv with stride = 1 and then apply pooling with e.g., stride= 2 rather than applying strided convolution (not for the computational cost)

Variance of the Basic Convolution Function (2)

- To avoid shrink of the output size after the convolution, we can do ***zero padding*** of the input V to make it wider



  - *valid convolution*: The output size shrinks
  - *same convolution*: The output size is same with the input

- ***Tiled convolution*** (Gregor2010): offers a compromise between a convolutional layer and a locally connected layer (learning a separate set of weights at every spatial location to emphasize the local information). We learn a set of kernels that we rotate through as we move through space, which implies that we use different kernels at different locations. (# params, standard < tiled < full)

- To back-propagate the convolution layer, we can simply see the convolution operation as a (sparse) matrix multiplication. As for the bias, it is typical to have one bias per channel of the output and share it across all locations (for tiled convolution, across same tiling patterns as the kernels)

Learning A Simple Convolutional Neural Networks

- Suppose we want to train a convolutional network that incorporates strided convolution of kernel stack K applied to multichannel image V with stride s
- Suppose the loss function is $J(V, K)$.
  - During the back propagation, we will receive a tensor G such that $G_{i,j,k} = \frac{\partial}{\partial Z_{i,j,k}} J(V, K)$ (Z is the output of the convolution).
  - To train the network we need to compute the derivatives with respect to the weights in the kernel:

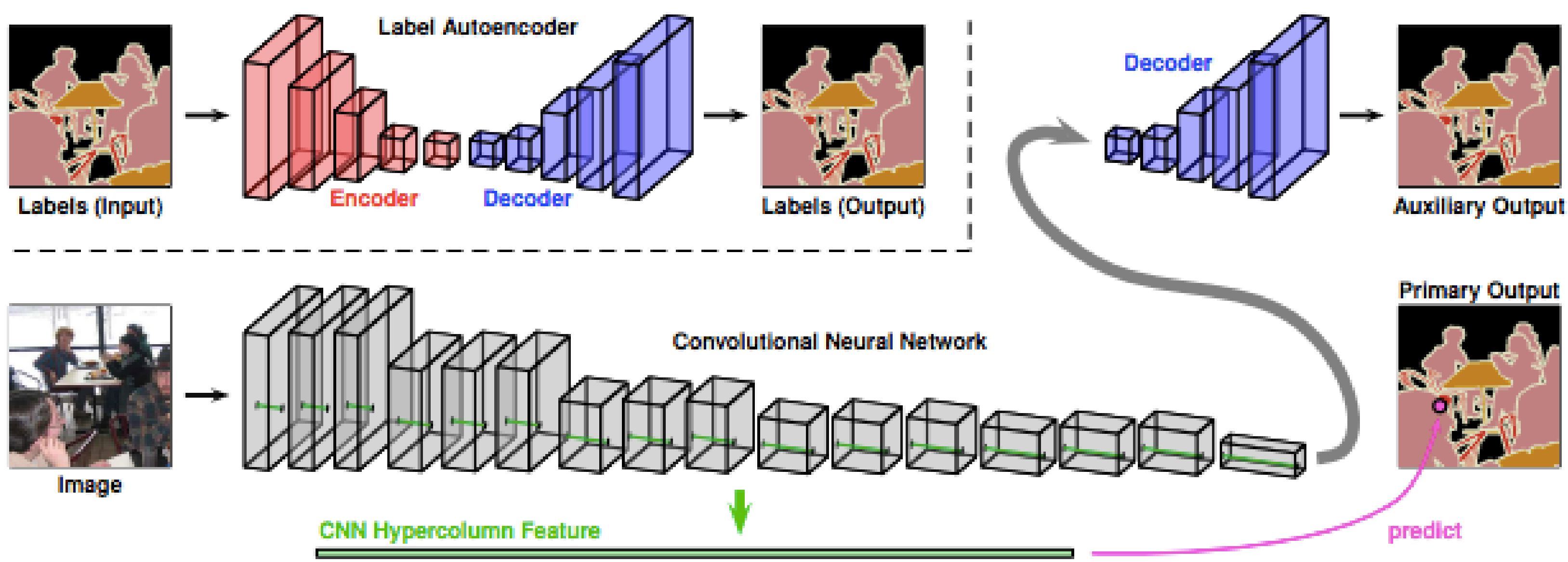
$$\frac{\partial}{\partial K_{i,j,k,l}} J(V, K) = \sum_{m,n} G_{i,m,n} V_{j,(m-1)*s+k,(n-1)*s+l}$$

  - We may need to compute the gradient with respect to the hidden layer V,

$$\frac{\partial}{\partial V_{i,j,k}} J(V, K) = \sum_{l,m \text{ (s.t.}(l-1)*s+m=j)} \sum_{n,p \text{ (s.t.}(n-1)*s+p=k} K_{q,i,m,p} G_{q,l,n}$$

Structured Outputs

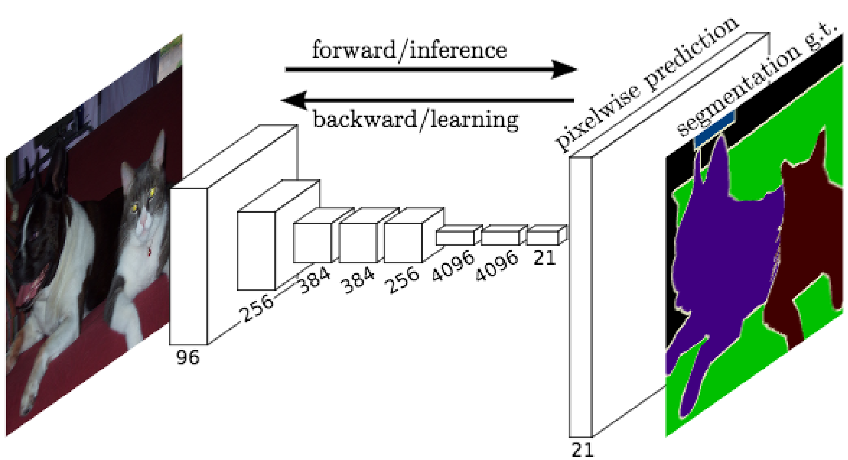
- Convolutional networks can be used to output a high-dimensional ***structured object*** (e.g., semantic segmentation)
- The issue is the output dimension can be smaller then input due to the pooling layers with large stride. To overcome this issue:
  a. Produce an initial guess at low resolution, then refine it using graphical model such as CRF/MRF
  b. Use ***upsampling/unpooling layer*** to increase the output size



Data Types

- One advantage to fully-convolutional neural networks is that they can process inputs with varying size of images in training/test data (note that valid for only spatial variation)

- If we put the dense layer with convolution layer (e.g., for assigning label to an entire image), we need some additional design steps, like inserting a pooling layer whose pooling regions scale in size proportional to the size of the input to maintain a fixed number of pooled outputs



Long et al., “Fully convolutional networks for semantic segmentation”, In CVPR2015

	Single channel	Multichannel
1-D	Audio waveform: The axis we convolve over corresponds to time. We discretize time and measure the amplitude of the waveform once per time step.	Skeleton animation data: Animations of 3-D computer-rendered characters are generated by altering the pose of a “skeleton” over time. At each point in time, the pose of the character is described by a specification of the angles of each of the joints in the character’s skeleton. Each channel in the data we feed to the convolutional model represents the angle about one axis of one joint.
2-D	Audio data that has been preprocessed with a Fourier transform: We can transform the audio waveform into a 2-D tensor with different rows corresponding to different frequencies and different columns corresponding to different points in time. Using convolution in the time makes the model equivariant to shifts in time. Using convolution across the frequency axis makes the model equivariant to frequency, so that the same melody played in a different octave produces the same representation but at a different height in the network’s output.	Color image data: One channel contains the red pixels, one the green pixels, and one the blue pixels. The convolution kernel moves over both the horizontal and the vertical axes of the image, conferring translation equivariance in both directions.
3-D	Volumetric data: A common source of this kind of data is medical imaging technology, such as CT scans.	Color video data: One axis corresponds to time, one to the height of the video frame, and one to the width of the video frame.

Obtaining Kernels without Supervised Training

- The forward/backward propagation for the supervised training of CNN is time consuming. One way to reduce the cost of convolutional neural network training is to use features that are not trained in a supervised fashion

- One is to initialize them randomly (e.g., Jarrett2009), another is to design them by hand (e.g., edge detector). Finally, one can learn the kernels with an unsupervised criterion (e.g., Coates2011 applied k-means clustering to small image patches then use each learned centroid as convolution kernel)

- A ***greedy layer-wise pretraining*** (e.g., Lee2009) train the first layer in isolation, then extract all features from the first layer only once then train the second layer in isolation and so on.
- *Today, it is common to learn the CNN in purely supervised manner*

Preprocessing

- In computer vision applications, images should be standardized so that their pixels all lie in the same reasonable range (e.g., [0,1]). Mixing different ranges results in failure. The common procedure is to subtract the mean from each image and divide it by std (***global contrast normalization***) or do it per local region (***local contrast normalization***). The result is the image of zero-mean and one-std
- The images should have the same aspect ratio (generally square) achieved by clopping and scaling

Input image GCN LCN

Design of the Hyperparameters in CNN

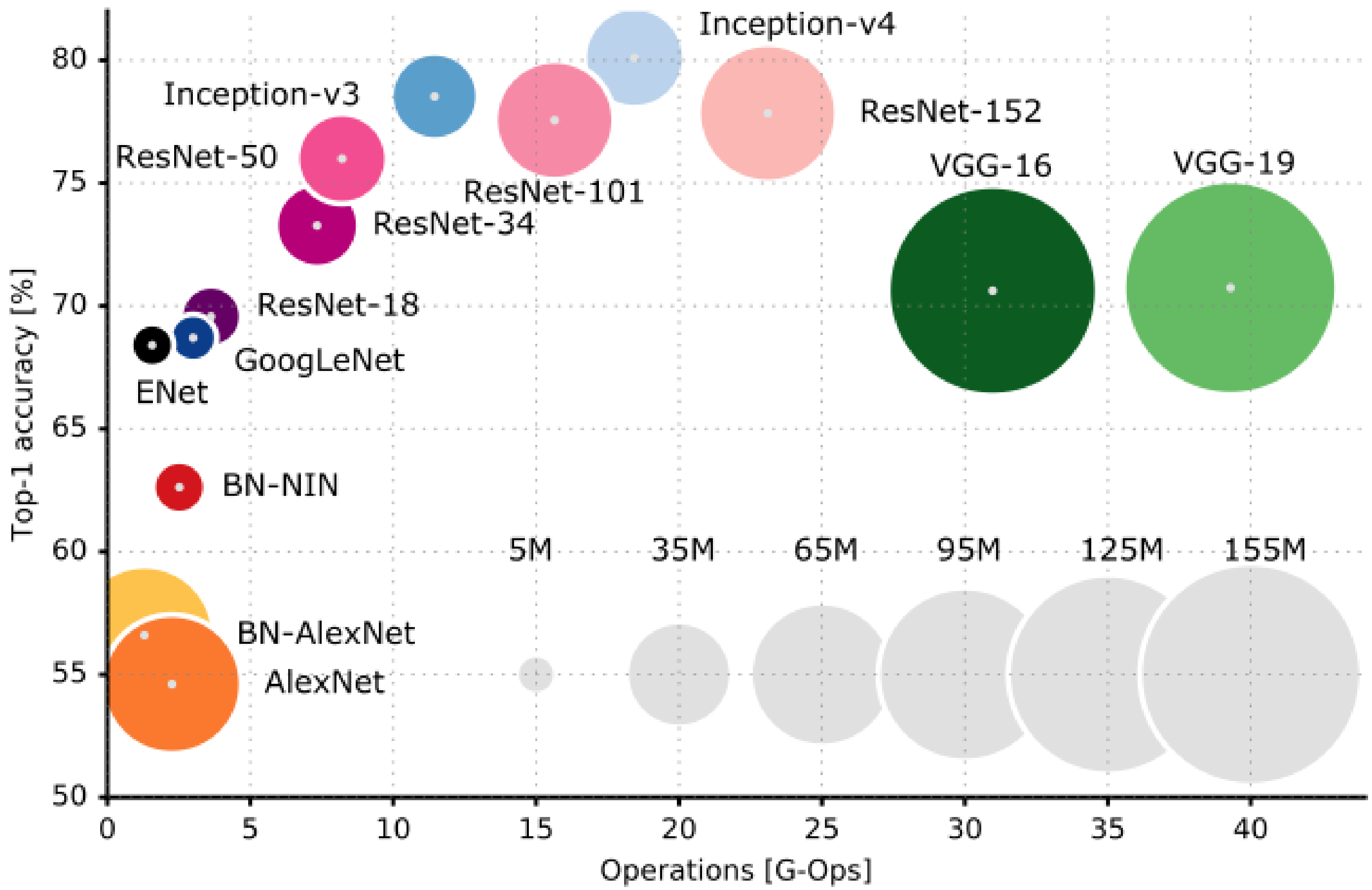
Hyperparameter	Increases capacity when...	Reason	Caveats
Number of hidden units	increased	Increasing the number of hidden units increases the representational capacity of the model.	Increasing the number of hidden units increases both the time and memory cost of essentially every operation on the model.
Learning rate	tuned optimally	An improper learning rate, whether too high or too low, results in a model with low effective capacity due to optimization failure.	
Convolution kernel width	increased	Increasing the kernel width increases the number of parameters in the model.	A wider kernel results in a narrower output dimension, reducing model capacity unless you use implicit zero padding to reduce this effect. Wider kernels require more memory for parameter storage and increase runtime, but a narrower output reduces memory cost.
Implicit zero padding	increased	Adding implicit zeros before convolution keeps the representation size large.	Increases time and memory cost of most operations.
Weight decay coefficient	decreased	Decreasing the weight decay coefficient frees the model parameters to become larger.	
Dropout rate	decreased	Dropping units less often gives the units more opportunities to "conspire" with each other to fit the training set.	

Table 11.1: The effect of various hyperparameters on model capacity.

Applications of CNN

Image Classification

- ImageNet Large Scale Visual Recognition Competition (ILSVRC): 1.2M for training, 150K for test.
- Object localization for 1000 categories, object detection for 200 categories, object detection from video for 30 categories




/data/ILSVRC2015/Data/VID/val/ILSVRC2015_val_00024002/000027.JPEG ID:4726
23:giant panda 0.936
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

LeNet and Its Variance

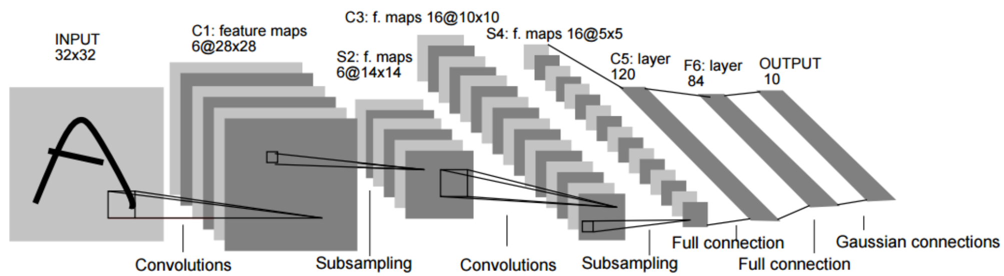
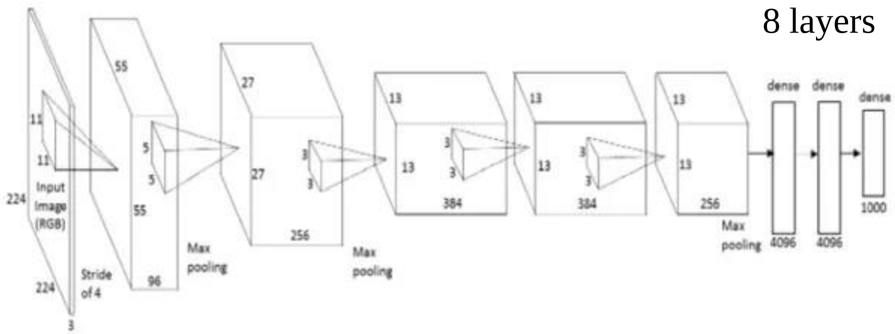


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

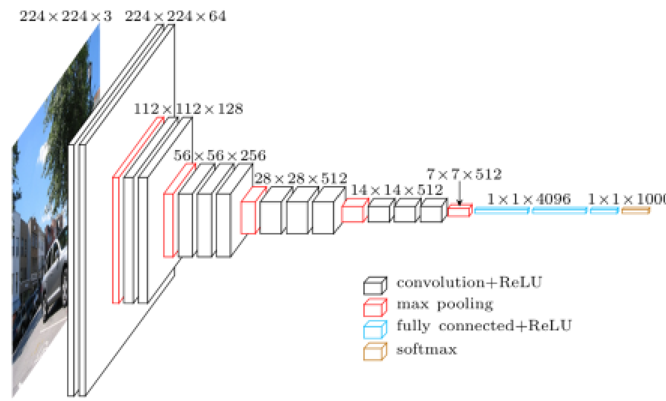
LeNet (LeCun1998)

8 layers



Alex-Net (Krizhevsky2012)

8 layers



VGG-Net (Simonyan2014)

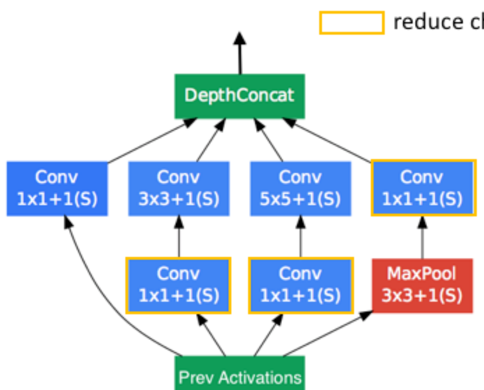
16 layers

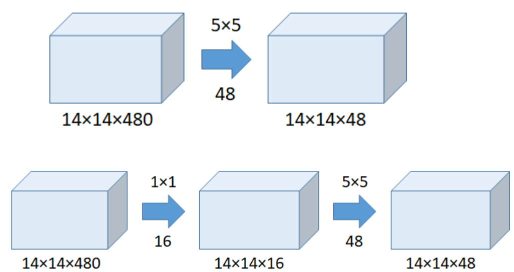
- Deeper, with more filters per layer, max pooling, dropout, data augmentation, ReLu activation, SGD with momentum (9% improvement in accuracy from the last year's challenge)

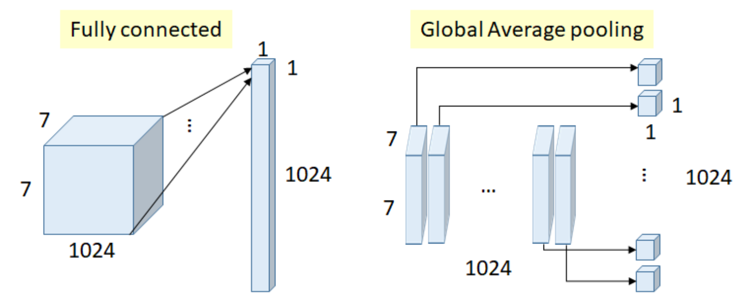
- 2nd ranked at ILSVRC2014
- Only using 3x3 convolution
- Similar to AlexNet but more filters
- The pretrained weight is publicly available

GoogLeNet and Inception Module

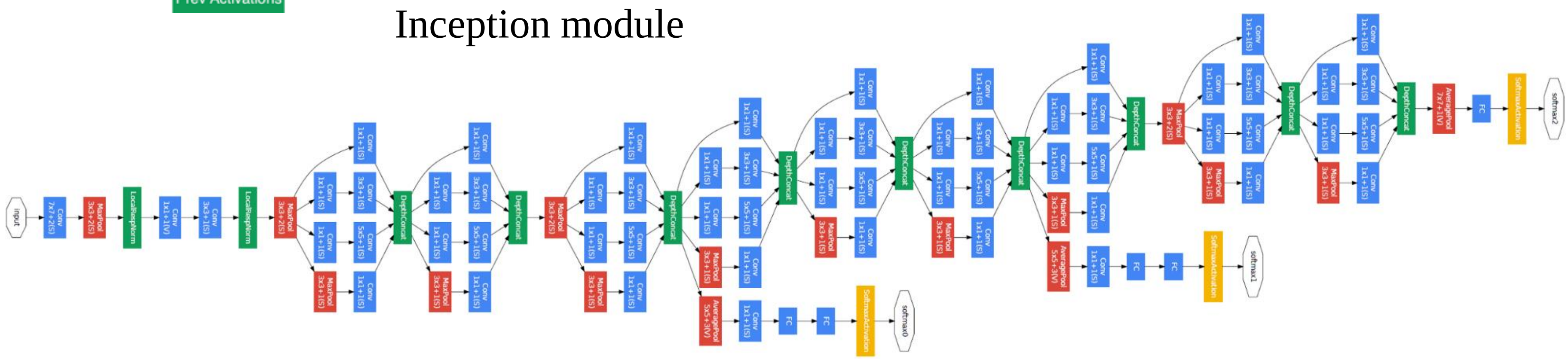
- ***Google-Net*** (Segedy2014): Won ILSVRC2014 (22 Layers)
- 1x1 convolution is used as a dimension reduction
- ***Global average pooling*** is introduced by averaging feature map from 7x7 to 1x1 to remove the weights for FCN layers







Inception module



https://medium.com/coinmonks/paper-review-of-googlenet-inception-v1-winner-of-ilsvlc-2014-image-classification-c2b3565a64e7

Deep Residual Networks

- ***ResNet*** (He2015): Won ILSVRC2015 (152 layers)
- Basic concept is "More Layers is Better"
- To avoid vanishing gradient problem, the residual function $H(x) = F(x) + x$ is introduced which allows the gradient being rapidly propagated through the network when applying backprop

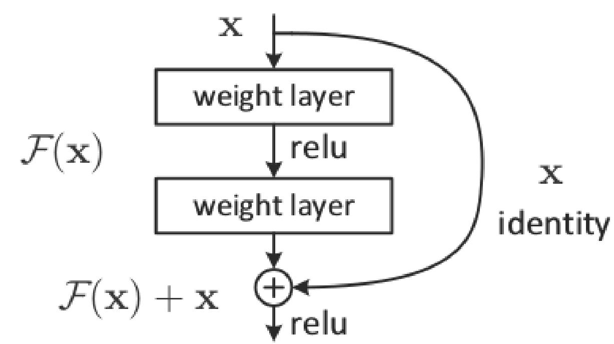
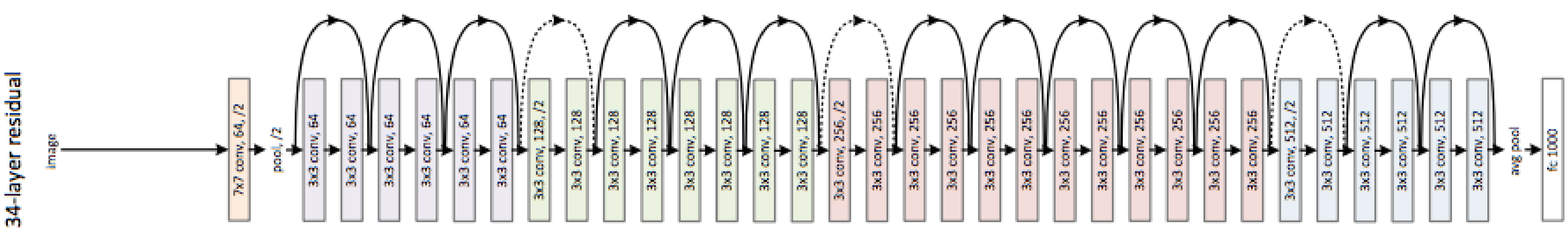
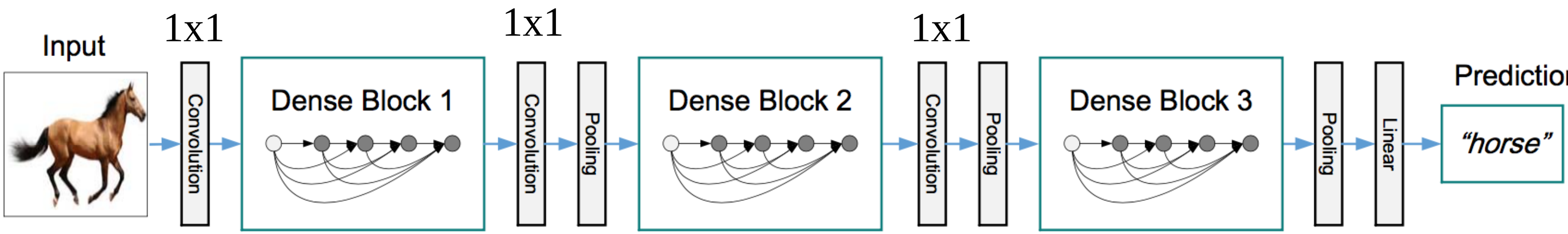


Figure 2. Residual learning: a building block.



Densely Connected Convolutional Networks

- ***DenseNets*** (Huang2017): introduces direct connections between any two layers with the same feature-map size. The idea behind is "it may be useful to reference feature maps from earlier in the network"



- DenseNets require substantially fewer parameters and less computation to achieve state-of-the-art performance

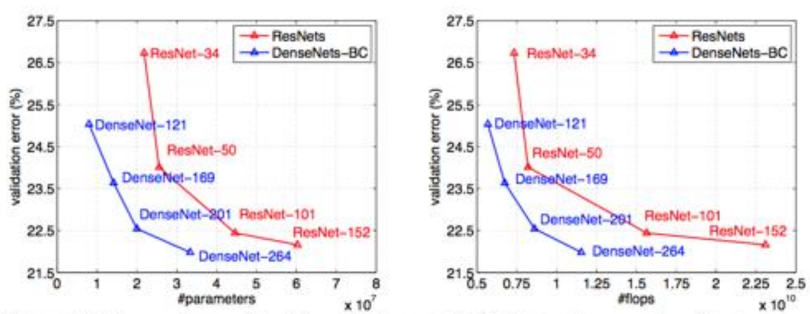
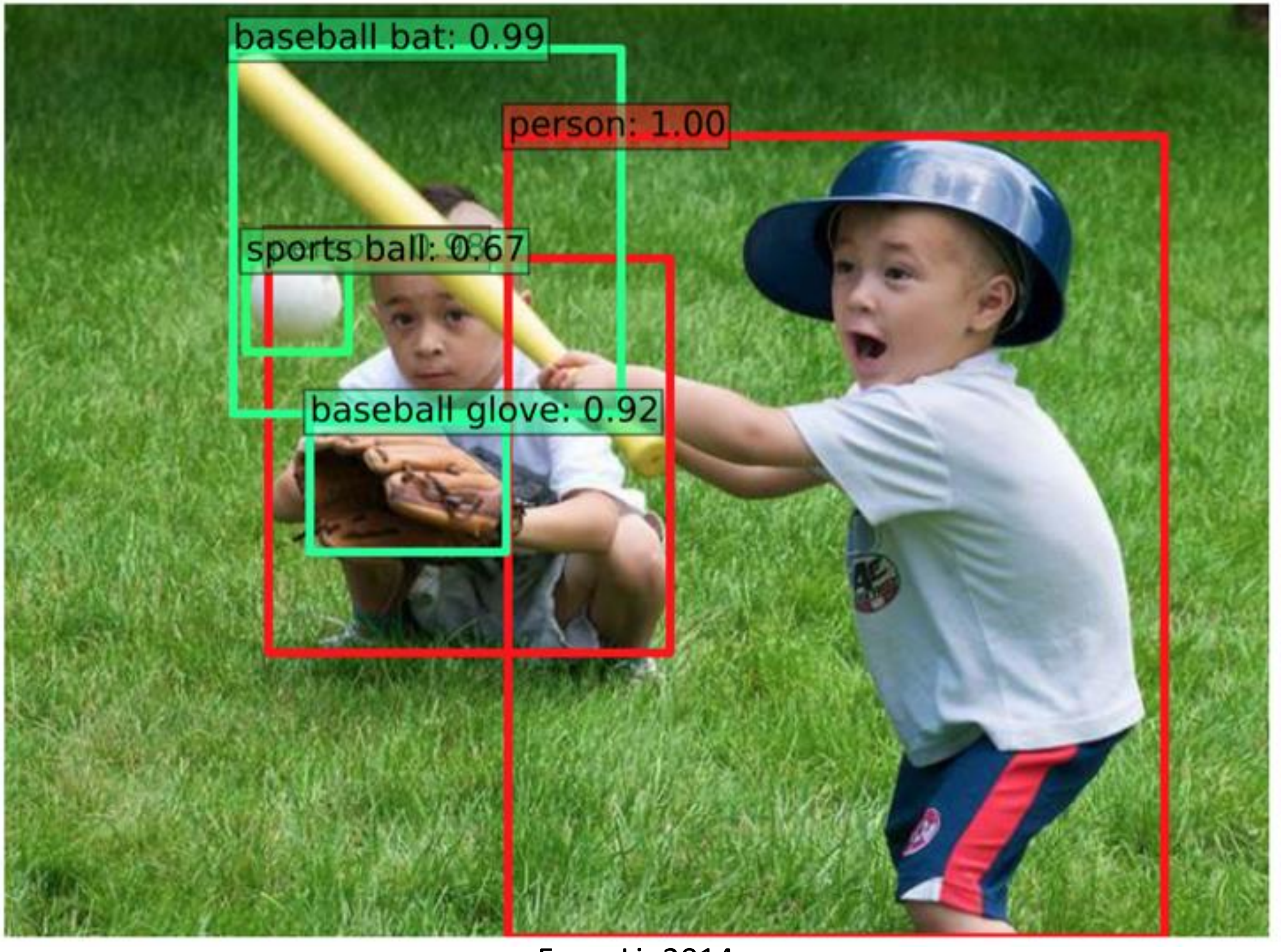


Figure 3: Comparison of the DenseNets and ResNets top-1 error rates (single-crop testing) on the ImageNet validation dataset as a function of learned parameters (*left*) and FLOPs during test-time (*right*).

Object Detection (1)

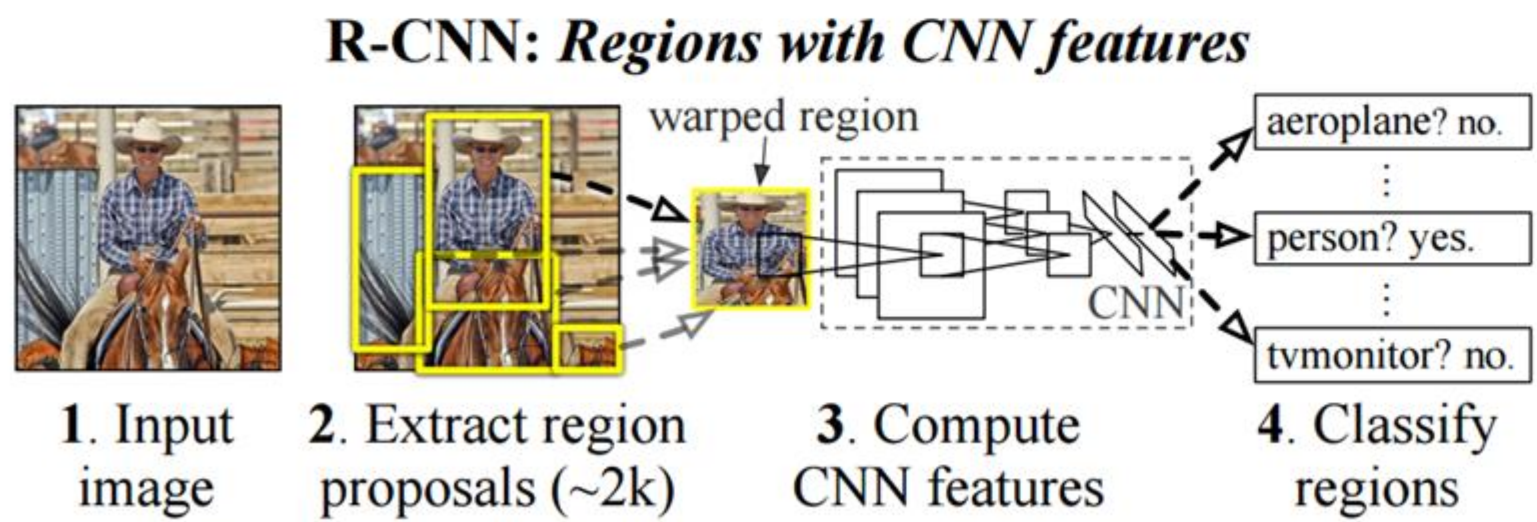
- Object detection task in the context of deep neural networks asks "where the object is" as well as "what is the object"



From Liu2014

Object Detection (2)

- ***Regions with Convolutional Neural Networks (R-CNN;*** Girshick2013): (a) extract region proposals (b) where CNN is applied for extracting features, (c) which are then classified using SVM, (d) then bounding box regression is applied
- The original R-CNN introduces ***selective search*** (hierarchical grouping) for region extraction: (a) initial candidate regions, (b) use greedy algorithm to merge similar regions into larger one

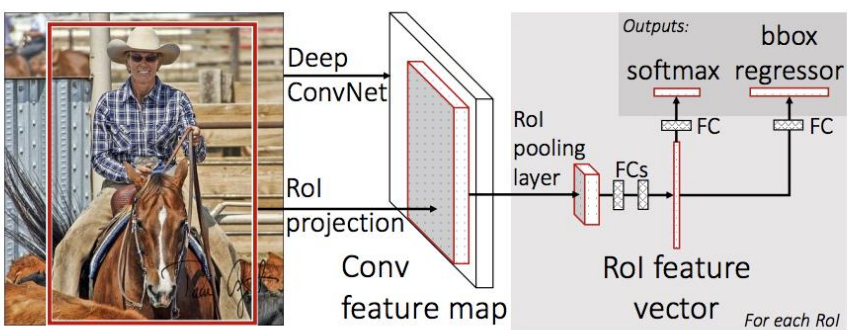


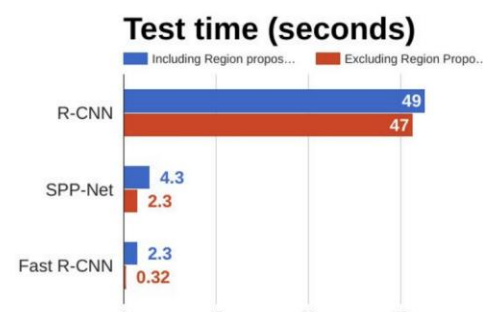
R-CNN workflow

Object Detection (3)

- Problems in R-CNN:
  - It trains classification/bounding box regression *independently*, therefore it takes large time to train the network and cannot be real time in test (47sec for one image)

- ***Fast R-CNN*** (Girshick2015): (a) Firstly, extracting features from *an entire image* and then using RoI projection to extract features at each region. (b) classification/bounding box regression are trained simultaneously using the multi-task loss

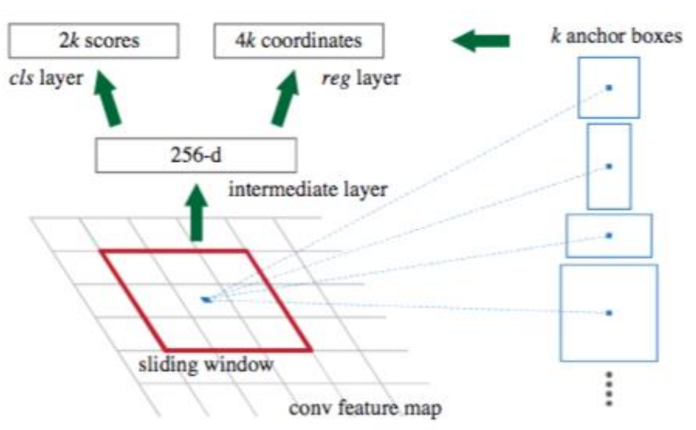


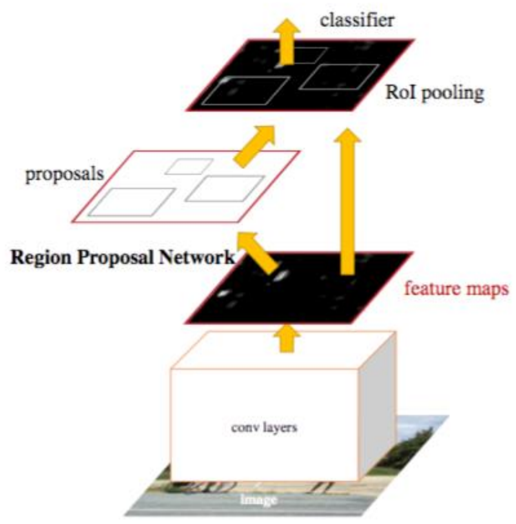


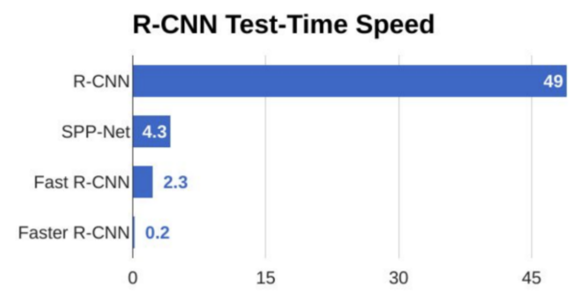
Object Detection (4)

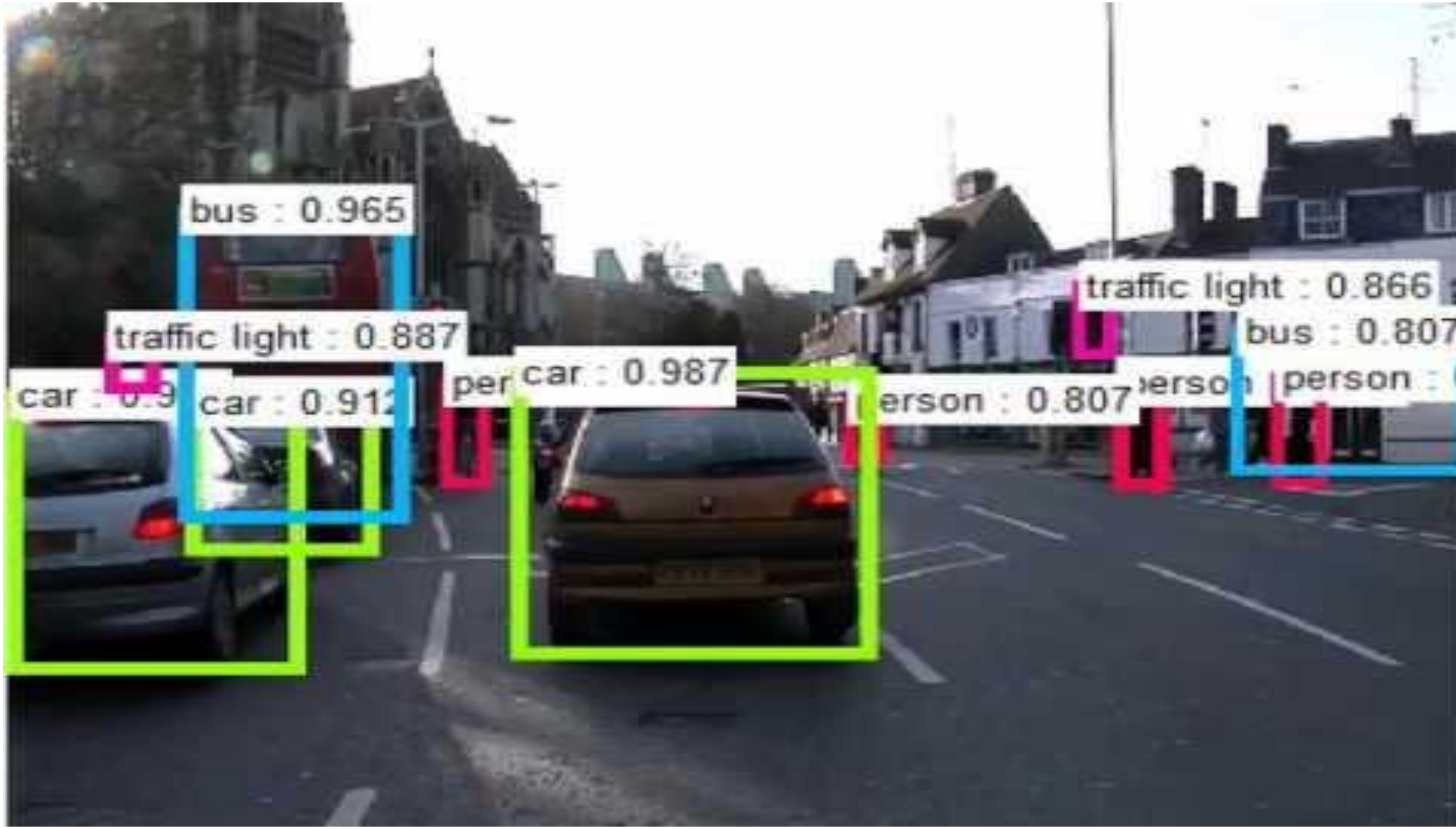
- Problems in Fast R-CNN:
  - It still requires the time-consuming region proposal extraction, therefore the framework is not actually end-to-end

- ***Faster R-CNN*** (Ren2015): introduced the ***region proposal network*** to learn the extraction of region proposal which allows an end-to-end learning (5fps on GPU)









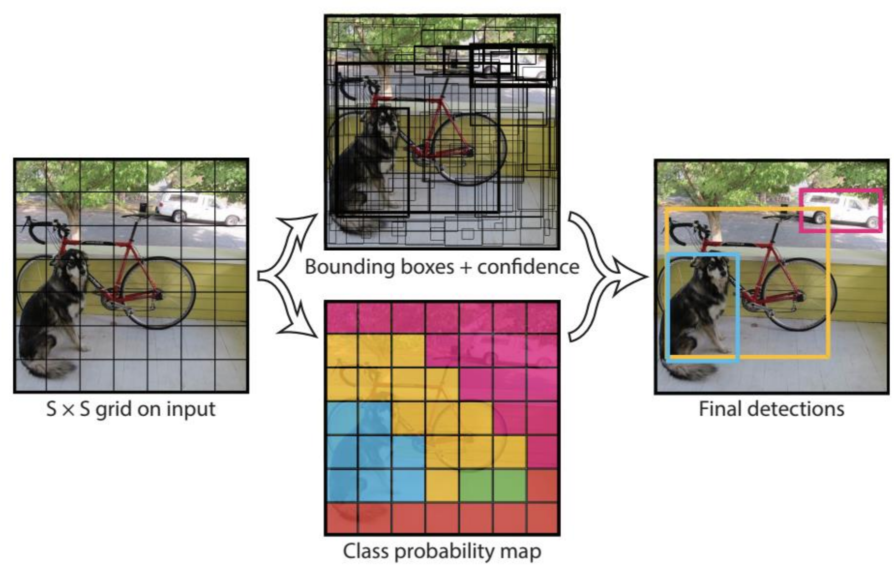
Object detection in the wild by Faster R-CNN + ResNet-101

https://www.youtube.com/watch?v=WZmSMkK9VuA

Object Detection (5)

- ***YOLO*** (You Only Look Once; Redmon2016): unlike previous algorithms that are "proposal extraction + classification", YOLO uses a single CNN to predicts the bounding boxes and the class probabilities for the box (use information outside the local region)
- YOLO takes an image and split it into grid, within each of the grid bounding boxes are taken. The bounding boxes whose class probability is above a threshold is selected to locate the object
- 2x faster but less accurate than Faster R-CNN. It is also weak for small objects

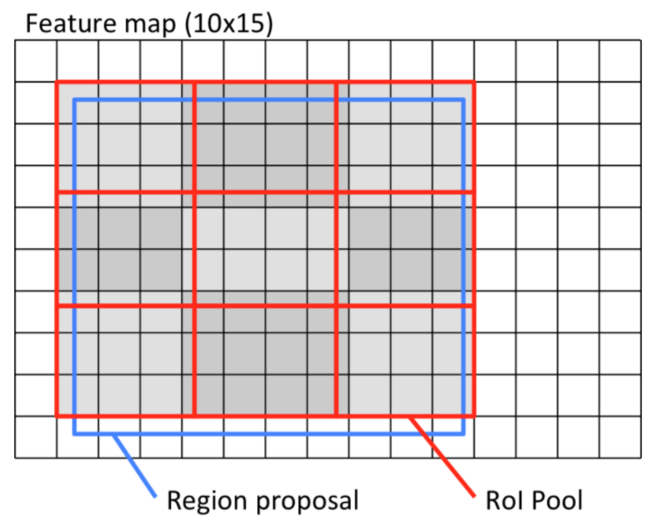
YOLO v2: 2017/12 w/ BN 224px -> 448px
YOLO v3: 2018/04 w/ multi-class, New arc.
YOLO v4: 2020/04 implemented by different people
YOLO v5: 2030/06 implemented by different people

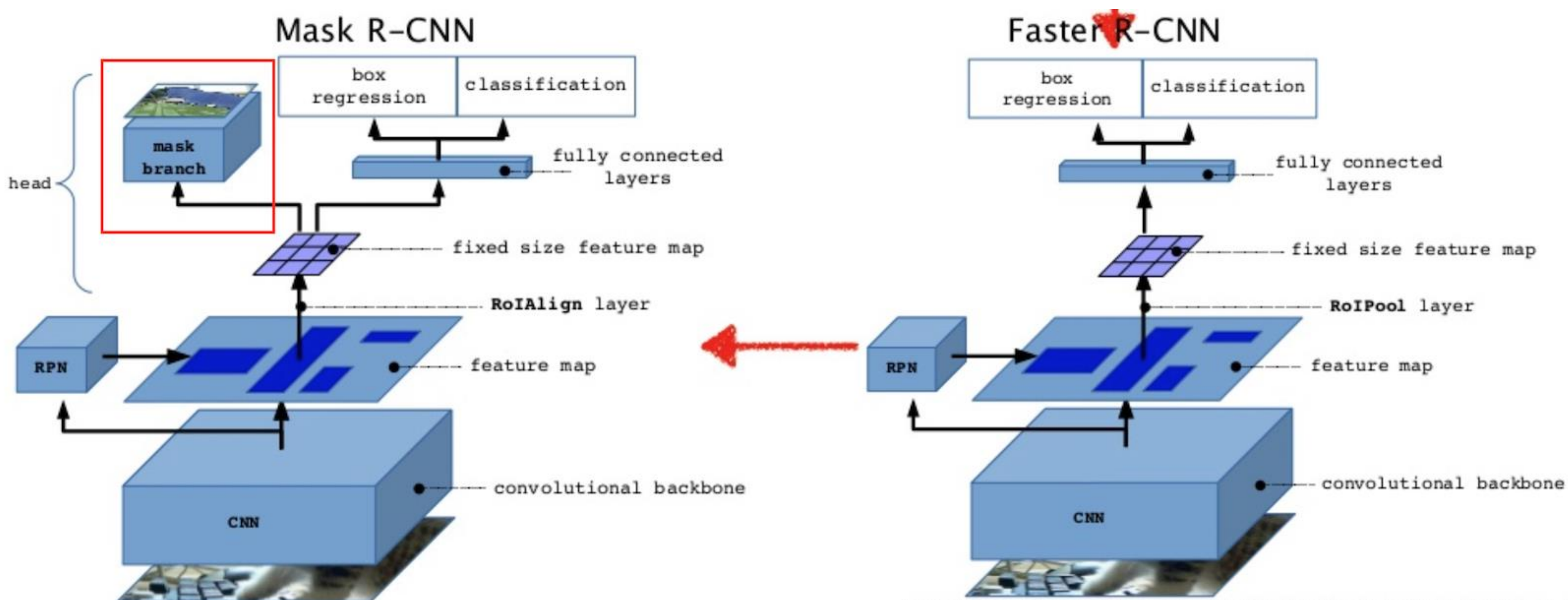


https://www.youtube.com/watch?v=V4P_ptn2FF4

Object Detection (6)

- Mask R-CNN (He2017):
  - Solved the "subpixel shift" problem in Faster R-CNN by bilinear interpolation
  - Can also predict object masks



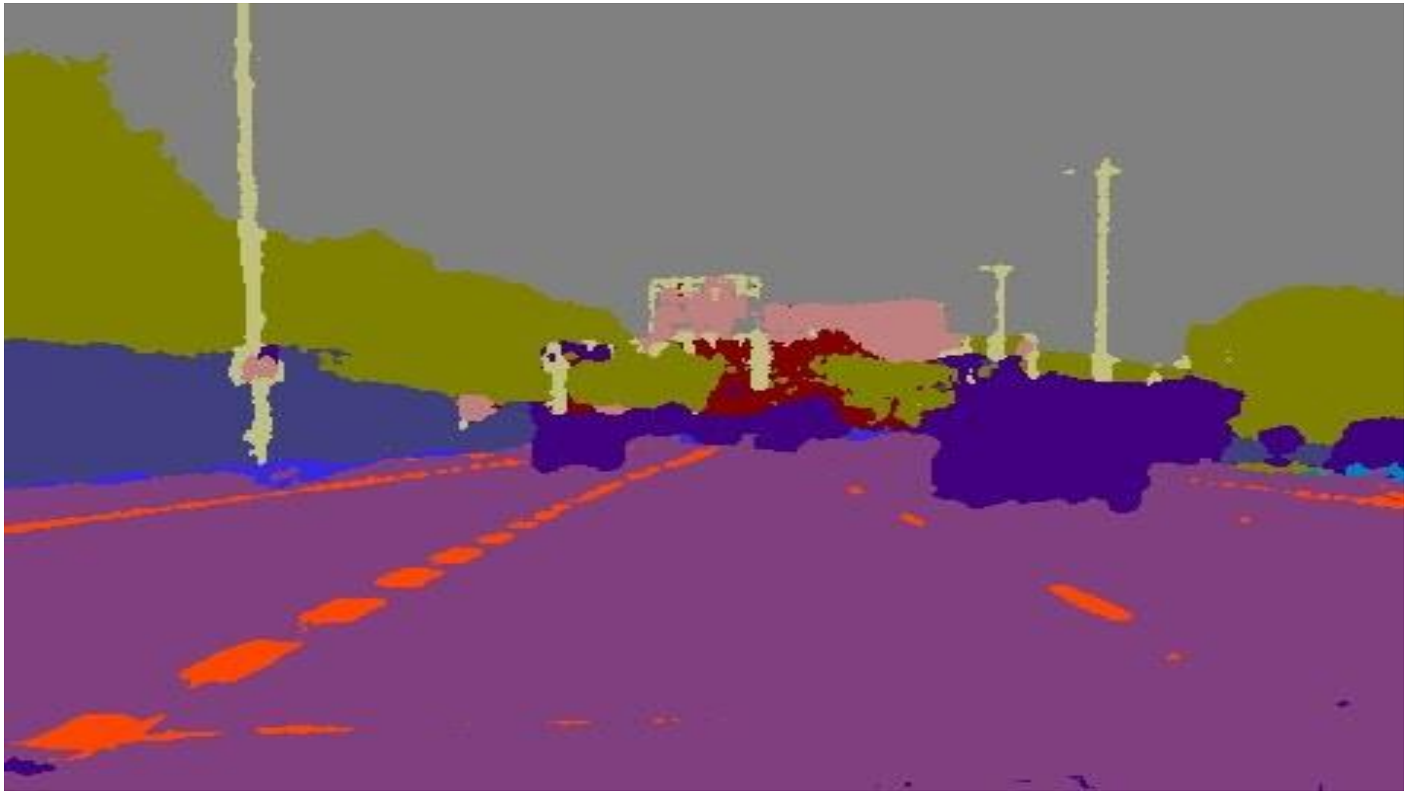


Curtis Kim, kakao, https://www.slideshare.net/IldooKim/deep-object-detectors-1-20166


YOLOv2 COCO
YOLOv3 COCO
Mask RCNN COCO

Semantic Segmentation (1)

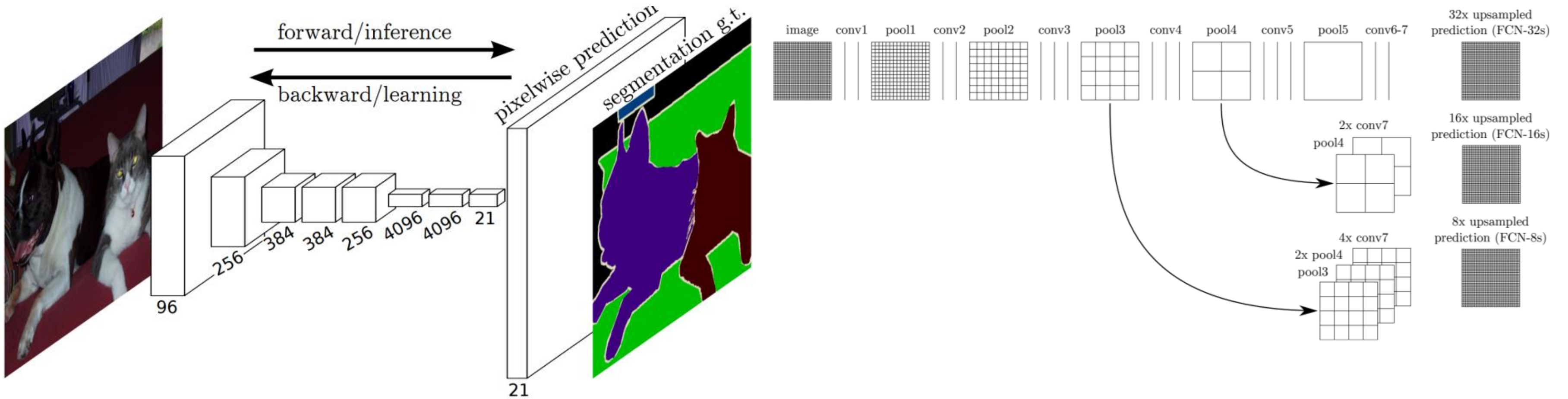
- ***Semantic segmentation*** task is to predict a *pixel-wise* instance label corresponding to an input image or vide frames
- VOC2012 and MSCOCO are important benchmark datasets
- Unlike other CNN tasks, the output is *structured (e.g,. image)*



SegNet: https://www.youtube.com/watch?v=CxanE_W46ts

Semantic Segmentation (2)

- The standard strategy is to use the ***encoder-decoder*** architecture



FCN (Long2014)

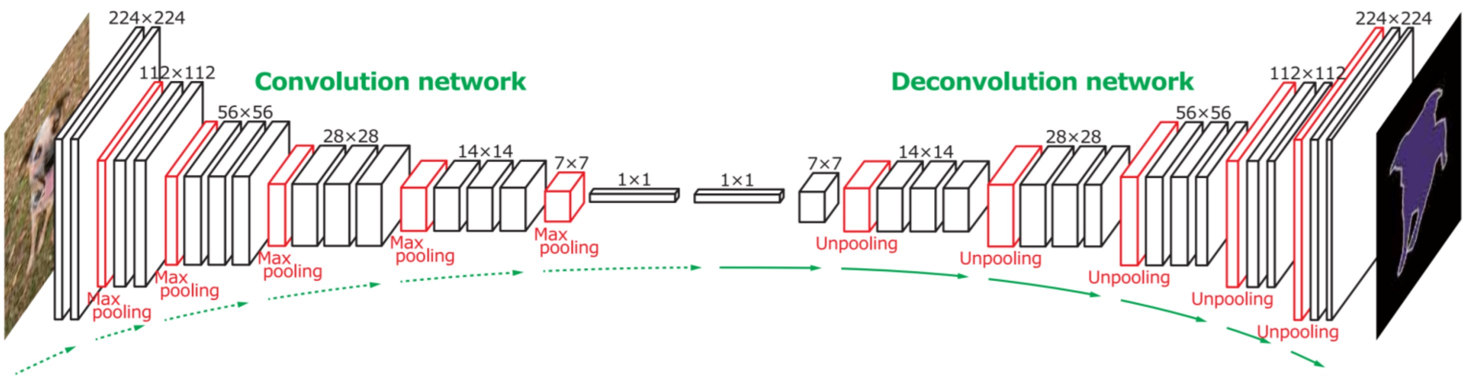


Figure 2. Overall architecture of the proposed network. On top of the convolution network based on VGG 16-layer net, we put a multi-layer deconvolution network to generate the accurate segmentation map of an input proposal. Given a feature representation obtained from the convolution network, dense pixel-wise class prediction map is constructed through multiple series of unpooling, deconvolution and rectification operations.

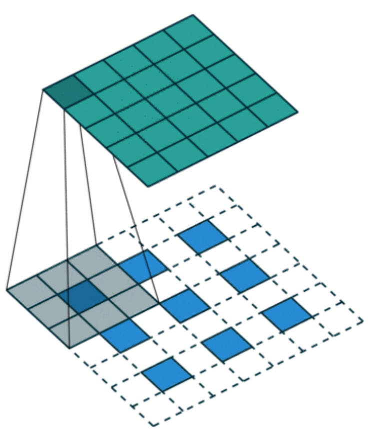
DeconvNet (Noh2015)

No skip connection

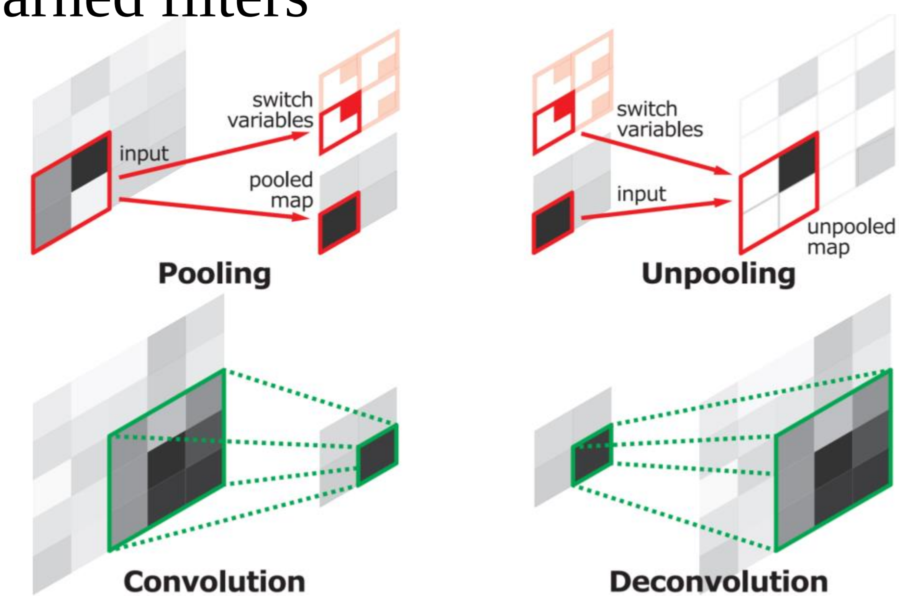
Semantic Segmentation (3)

- ***Unpooling***: the reverse operation of max pooling. It recodes the locations of maximum activations selected during pooling operation in switch variables, which are employed to place each activation back to its original pooled location

- ***Deconvolution (transposed convolution)***: densify the sparse activations obtained by unpooling through convolution-like operations with multiple learned filters

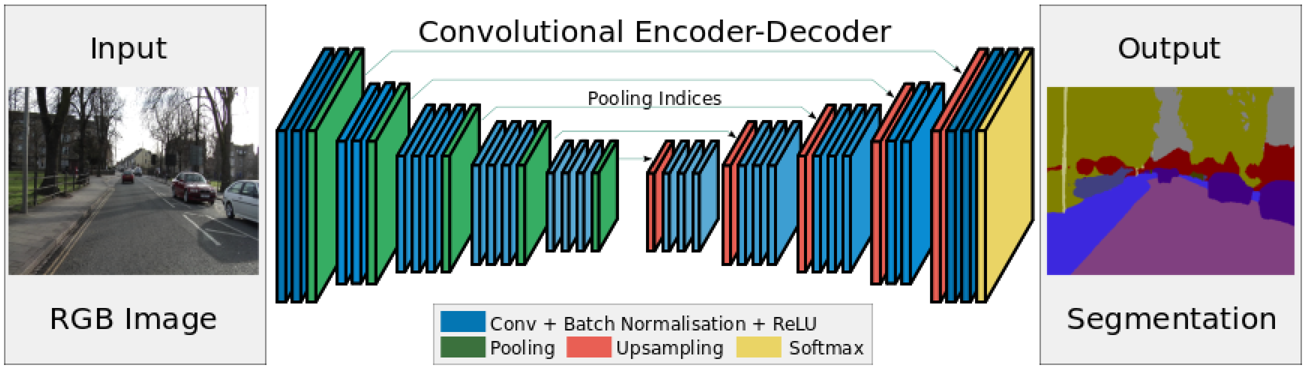


Deconvolution operation

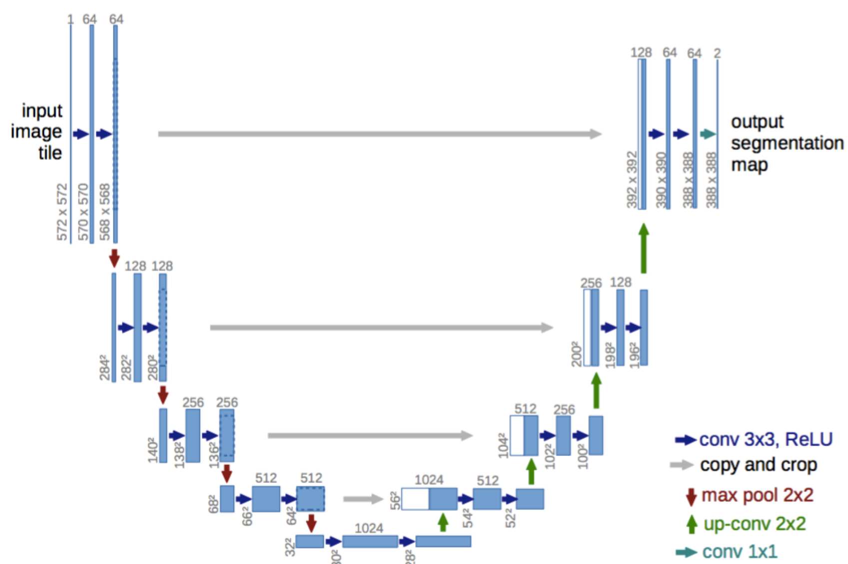


Semantic Segmentation (5)

- ***Skip connection*** is a very powerful tool to keep the original resolution and propagate loss effectively in back propagation



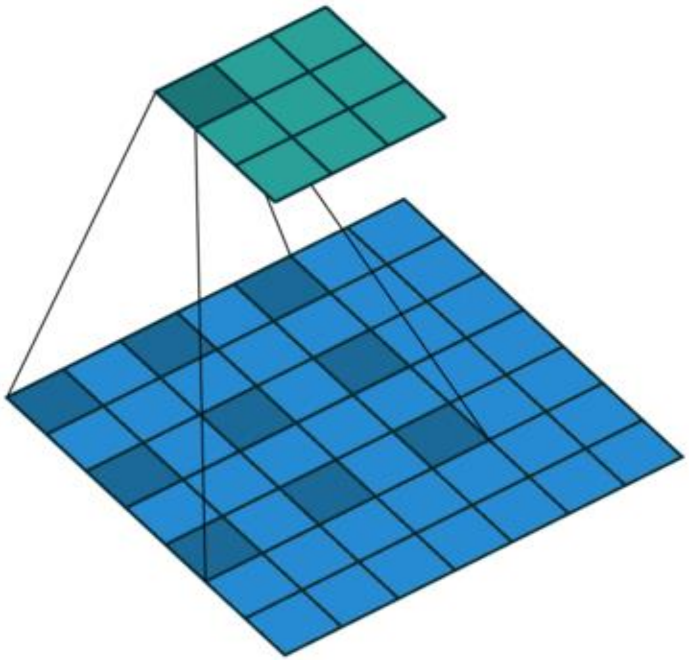
Seg-Net (Badrinarayanan2015)   w/ class balancing

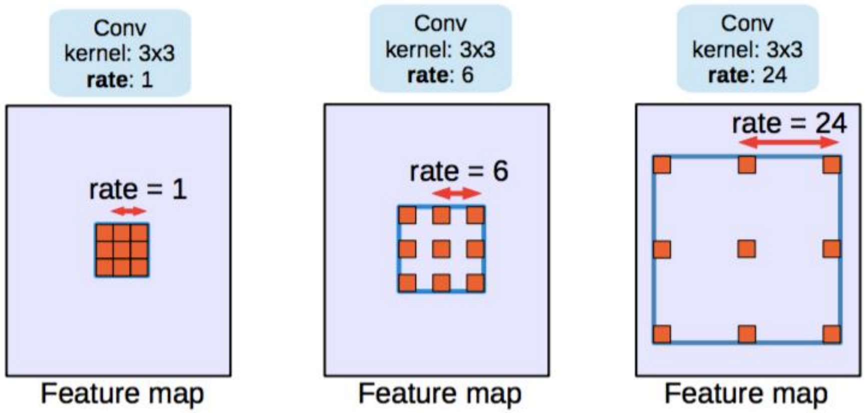


U-Net (Ronneberger2015) w/ weighted loss on boundary

Semantic Segmentation (6)

- Other than the encoder-decoder like net, we can use the ***dilated convolution*** (Yu2015) *without using pooling* to keep the original resolution





Layer	1	2	3	4	5	6	7	8
Convolution	3×3	3×3	3×3	3×3	3×3	3×3	3×3	1×1
Dilation	1	1	2	4	8	16	1	1
Truncation	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No
Receptive field	3×3	5×5	9×9	17×17	33×33	65×65	67×67	67×67
Output channels								
Basic	C	C	C	C	C	C	C	C
Large	$2C$	$2C$	$4C$	$8C$	$16C$	$32C$	$32C$	C

https://github.com/vdumoulin/conv_arithmetic

Other Topics

- CNN on 3-D Data (point cloud, voxels, meshes)
  - Differentiable Renderer, Single image 3-D reconstruction
- Graph Neural Networks
  - Holistic vision
- Self-supervised / Unsupervised Learning, Domain Adaptation
  - Deep learning with few training samples
- Generative Networks (e.g., GAN, Transformer)
  - Deep fake, StyleGAN
- Multi-modal Neural Networks (e.g., Image + Text)
  - Deep image captioning
- Deep Reinforcement Learning
  - e.g., AlphaGO
- <u>Recurrent Neural Networks (next weak)</u>