CENG501 – Deep Learning

Week 4

Fall 2024

Sinan Kalkan

Dept. of Computer Engineering, METU

Per-parameter Methods: Adaptive Moments (Adam)

Previously on CENG501

- A variation of RMSprop + momentum
- Incorporates first & second order moments
- Bias correction needed to get rid of bias towards zero at initialization

Algorithm taken from:
Goodfellow et al., Deep Learning, 2016.

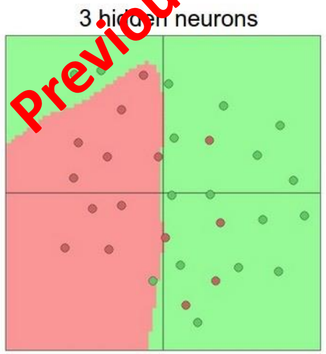
Algorithm 8.7 The Adam algorithm

Require: Step size ϵ (Suggested default: 0.001)
Require: Exponential decay rates for moment estimates, ρ_1 and ρ_2 in $[0, 1)$. (Suggested defaults: 0.9 and 0.999 respectively)
Require: Small constant δ used for numerical stabilization (Suggested default: 10^{-8})
Require: Initial parameters $\boldsymbol{\theta}$
  Initialize 1st and 2nd moment variables $\boldsymbol{s} = \mathbf{0}$, $\boldsymbol{r} = \mathbf{0}$
  Initialize time step $t = 0$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    $t \leftarrow t + 1$
    Update biased first moment estimate: $\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1 - \rho_1)\boldsymbol{g}$
    Update biased second moment estimate: $\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1 - \rho_2)\boldsymbol{g} \odot \boldsymbol{g}$
    Correct bias in first moment: $\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1-\rho_1^t}$
    Correct bias in second moment: $\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1-\rho_2^t}$
    Compute update: $\Delta \boldsymbol{\theta} = -\epsilon \frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}} + \delta}$ (operations applied element-wise)
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta \boldsymbol{\theta}$
  end while

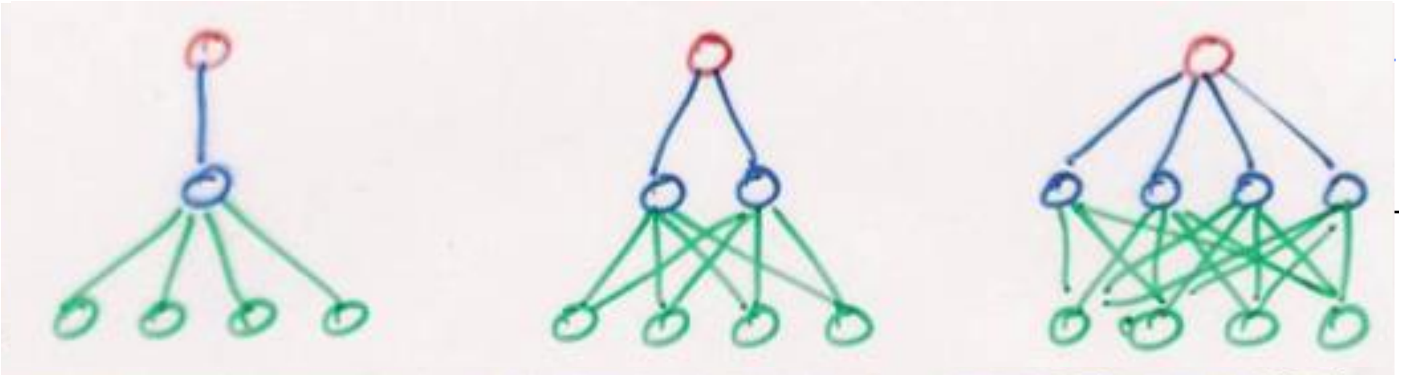
Previously on CENG501

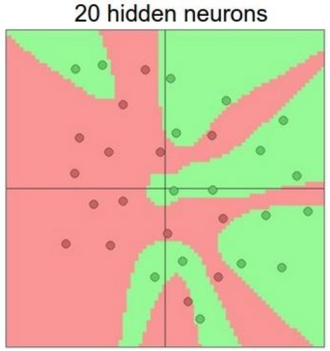
Model Complexity

- Models range in their flexibility to fit arbitrary data



https://cs231n.github.io/





https://cs231n.github.io/

simple model / **high bias**

low variance / **constrained**

small capacity may prevent it from representing all structure in data

complex model / **low bias**

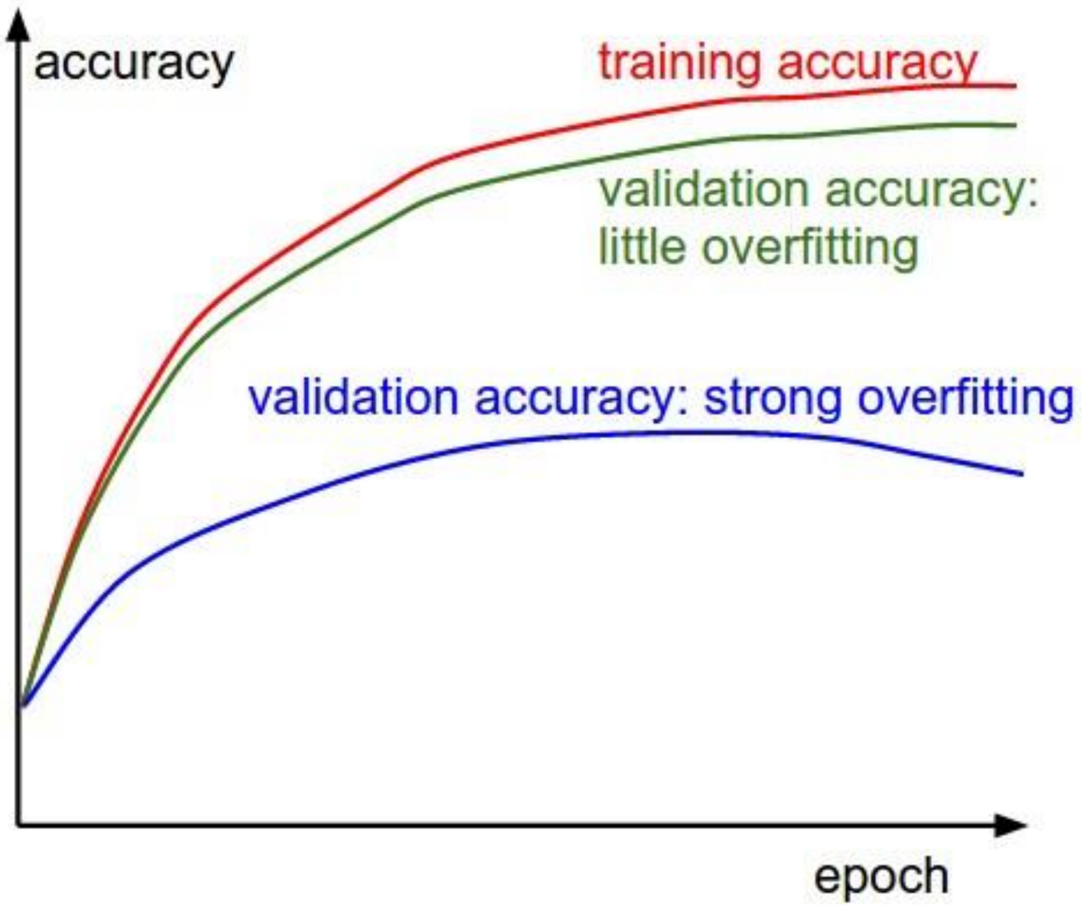
high variance / **unconstrained**

large capacity may allow it to memorize data and fail to capture regularities

Slide Credit: Michael Mozer

Previously on CENG501

How do you spot overfitting?



Previously on CENG501

Avoiding Overfitting

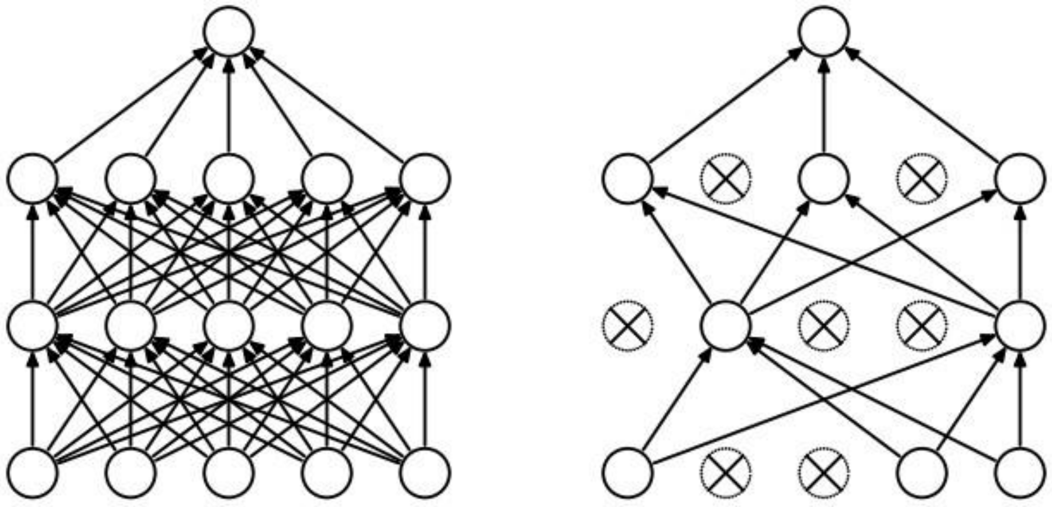
- Increase training set size
  - Make sure effective size is growing; redundancy doesn't help
- Incorporate domain-appropriate bias into model
  - Customize model to your problem
- Tune hyperparameters of model
  - number of layers, number of hidden units per layer, connectivity, etc.
- Regularization techniques

Slide Credit: Michael Mozer

Previously on CENG501

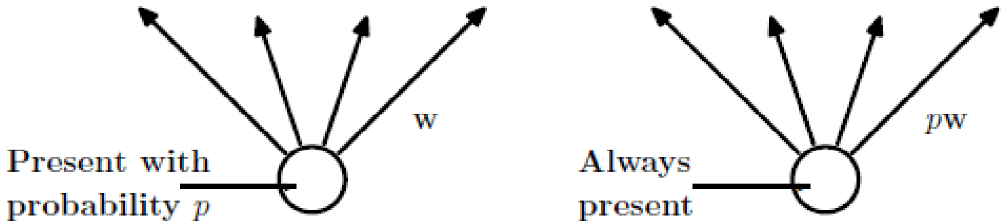
Regularization: Dropout

Fig: Srivastava et al., 2014



(a) Standard Neural Net (b) After applying dropout.

- Feed-forward only on active units
- Can be trained using SGD with mini-batch
  - Back propagate only “active” units.
- One issue:
  - Expected output x with dropout:
  - $E[x'] = \frac{1}{N}\sum_i (px_i + (1-p)0) = p\frac{1}{N}\sum_i x_i = pE[x]$
- To have the same scale at testing time (no dropout), multiply test-time activations with p.



(a) At training time (b) At test time

Fig: Srivastava et al., 2014

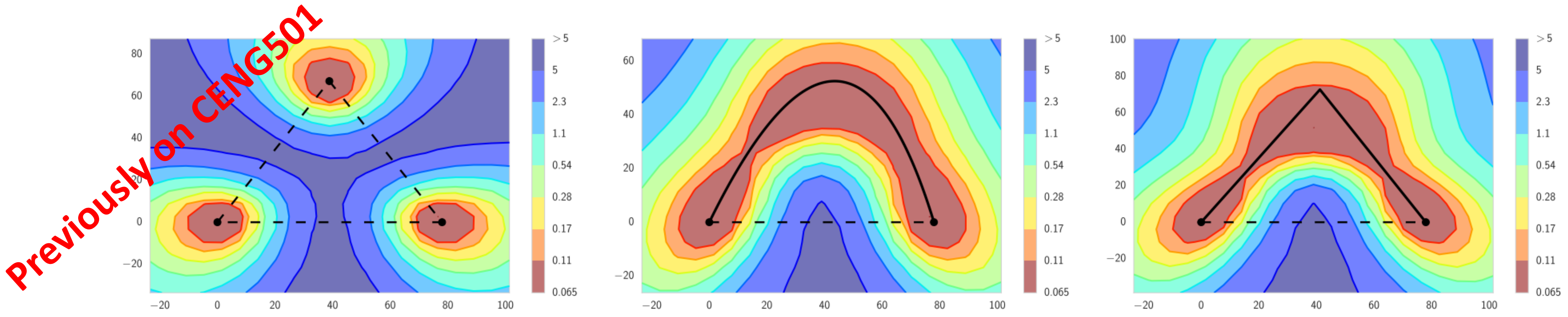


Figure 1: The ℓ_2-regularized cross-entropy train loss surface of a ResNet-164 on CIFAR-100, as a function of network weights in a two-dimensional subspace. In each panel, the horizontal axis is fixed and is attached to the optima of two independently trained networks. The vertical axis changes between panels as we change planes (defined in the main text). **Left:** Three optima for independently trained networks. **Middle** and **Right**: A quadratic Bezier curve, and a polygonal chain with one bend, connecting the lower two optima on the left panel along a path of near-constant loss. Notice that in each panel a direct linear path between each mode would incur high loss.

Garipov et al., “Loss Surfaces, Mode Connectivity, and Fast Ensembling of DNNs”, 2018.

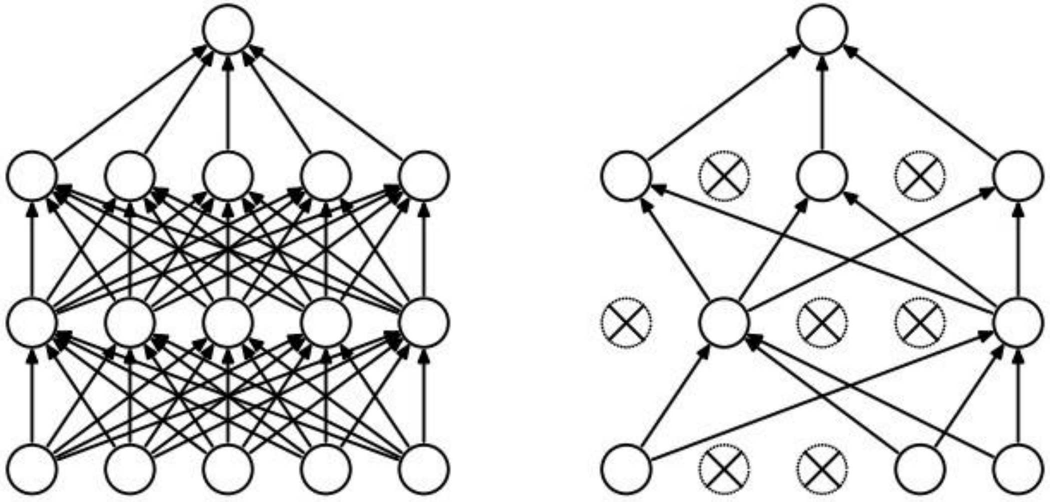
See also: Kuditipudi et al., “Explaining Landscape Connectivity of Low-cost Solutions for Multilayer Nets”, 2020.
- Explains this with noise stability, dropout stability.

Dropout as Ensemble Training Method

Previously on CENG501

"Dropout performs gradient descent on-line with respect to both the training examples and the ensemble of all possible subnetworks."

Fig: Srivastava et al., 2014



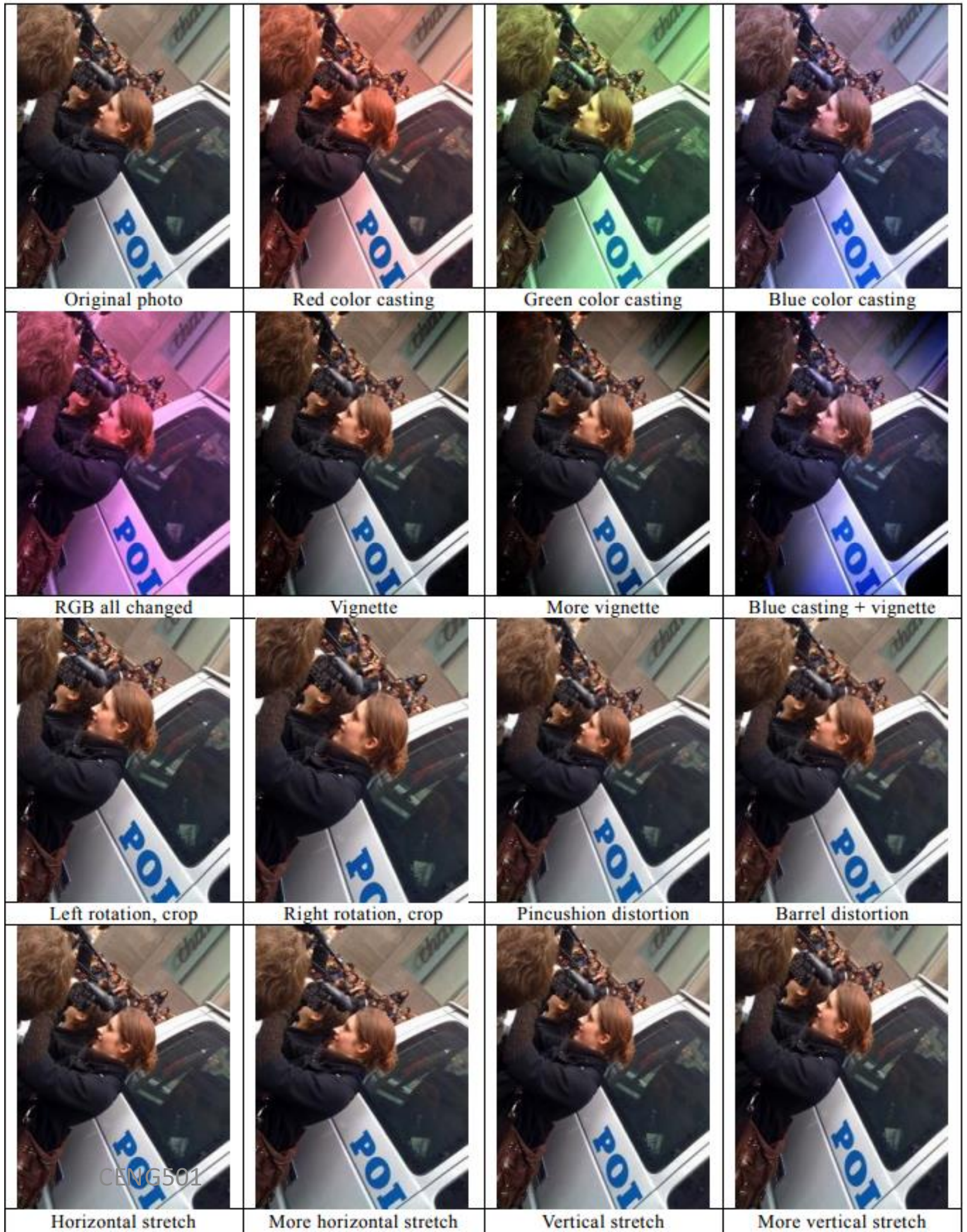
(a) Standard Neural Net

(b) After applying dropout.

Pierre Baldi and Peter J Sadowski. Understanding dropout. In Advances in neural information processing systems, pp. 2814–2822, 2013.

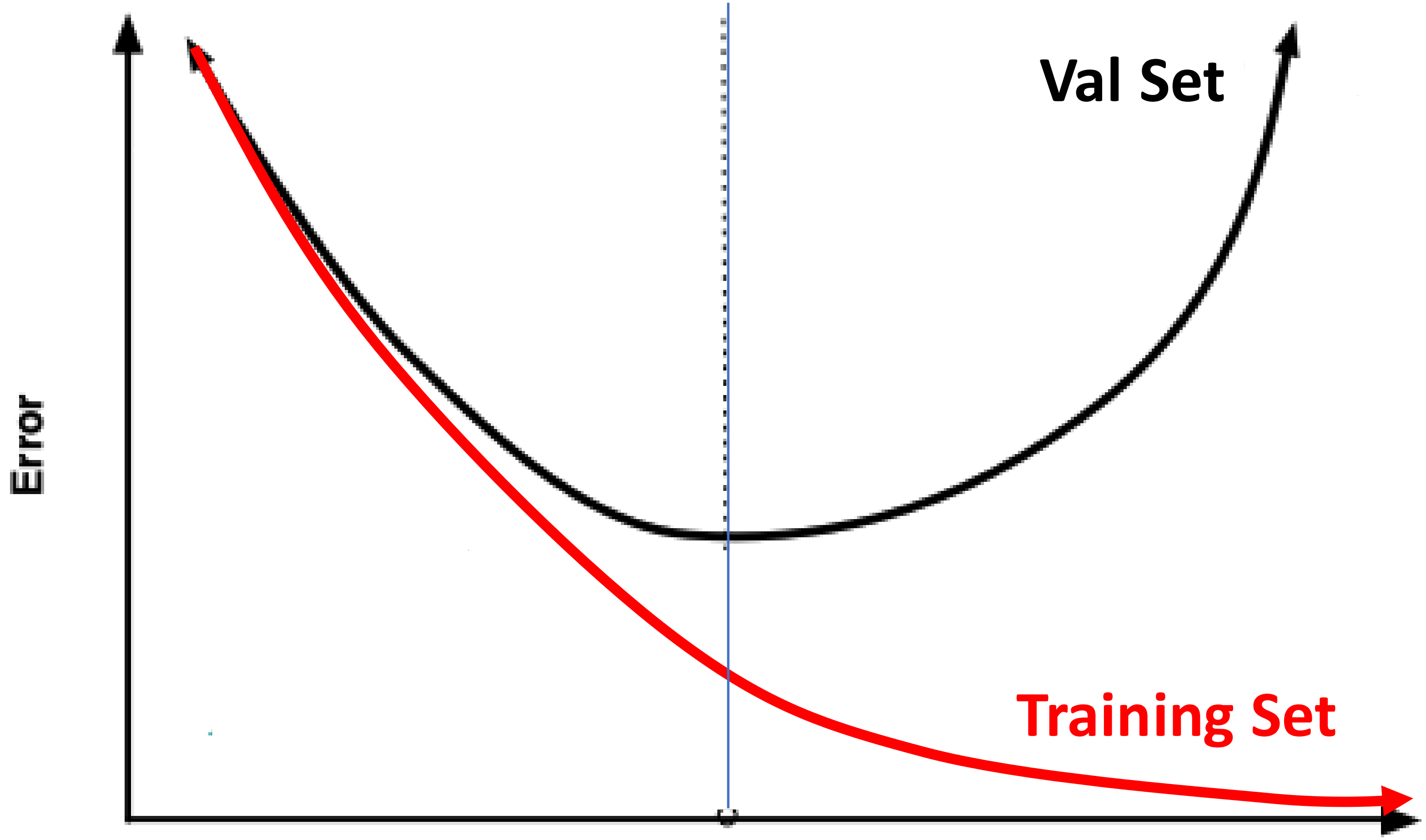
Data Augmentation

Previously on CENG501



http://blcv.pl/static//2018/02/27/demystifying-face-recognition-v-data-augmentation/

When to stop training



Slide Credit: Michael Mozer

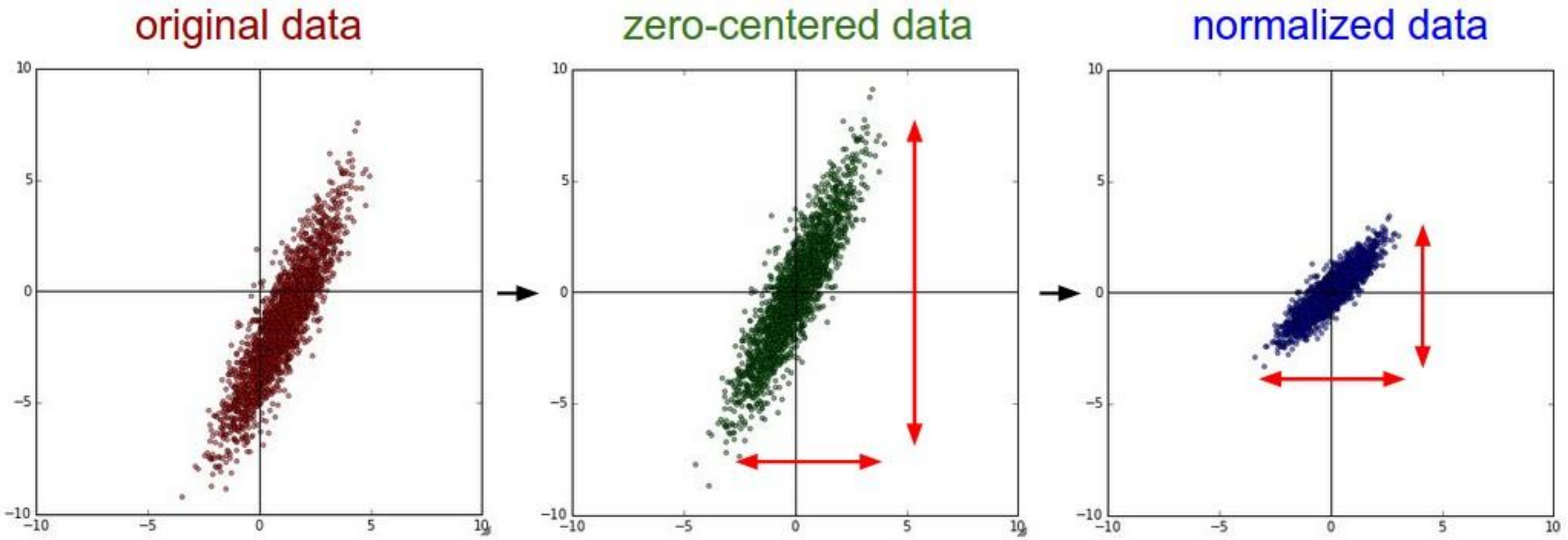
Previously on CENG501

Data Preprocessing: Normalization (or conditioning)

- Necessary if you believe that your dimensions have different scales
  - Might need to reduce this to give equal importance to each dimension
- Normalize each dimension by its std. dev. after mean subtraction:

$$x'_{ji} = x_{ji} - \mu_i$$
$$x''_{ji} = x'_{ji} / \sigma_i$$

- Effect: Make the dimensions have the same scale



http://cs231n.github.io/neural-networks-2/

Initial Weight Normalization

- Solution:
  - Get rid of n in $Var(s) = (n$ Var(w))Var(x)
- How?
  - Scale the initial weights by $\sqrt{n}$
  - Why? Because: $Var(aX) = a^2 Var(X)$

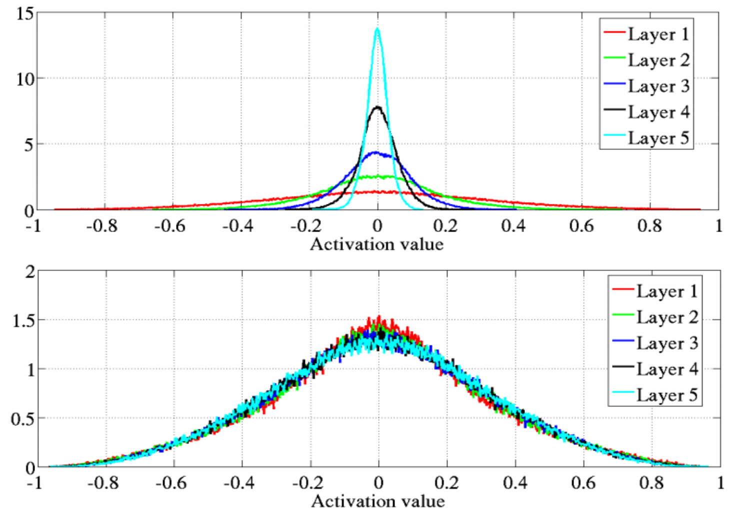


Figure 6: ***Activation values normalized histograms with hyperbolic tangent activation, with standard (top) vs normalized initialization (bottom). Top: 0-peak increases for higher layers.***

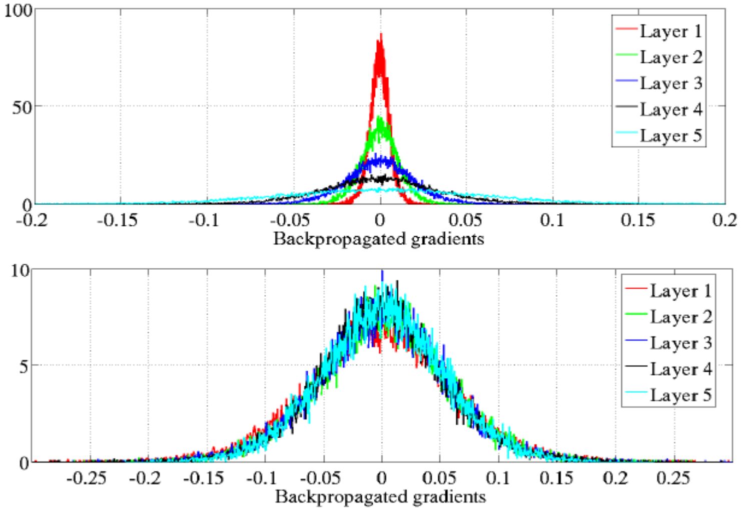


Figure 7: *Back-propagated gradients normalized histograms with hyperbolic tangent activation, with standard (top) vs normalized (bottom) initialization. Top: 0-peak decreases for higher layers.*

Figures: Glorot & Bengio, “Understanding the difficulty of training deep feedforward neural networks”, 2010.

- Standard Initialization (top plots in Figure 6 & 7):

$$w_i \sim U\left[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}\right]$$

which yields $n\, Var(w) = \frac{1}{3}$

because variance of $U[-r, r]$ is $\frac{r^2}{3}$ [1].

Xavier initialization for symmetric activation functions (Glorot & Bengio):

$$w_i \sim N\left(0, \frac{\sqrt{2}}{\sqrt{n_{in} + n_{out}}}\right)$$

With Uniform distribution:

$$w_i \sim U\left[-\frac{\sqrt{6}}{\sqrt{n_{in} + n_{out}}}, \frac{\sqrt{6}}{\sqrt{n_{in} + n_{out}}}\right]$$

[1] https://proofwiki.org/wiki/Variance_of_Continuous_Uniform_Distribution

Previously on CENG501

Alternative: Batch Normalization

- Normalization is differentiable
  - So, make it part of the model (not only at the beginning)
  - I.e., perform normalization during every step of processing
- More robust to initialization
- Shown to also regularize the network in some cases (dropping the need for dropout)
- Issue: How to normalize at test time?
  1. Store means and variances during training, or
  2. Calculate mean & variance over your test data
- PyTorch: use model.eval() in test time.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

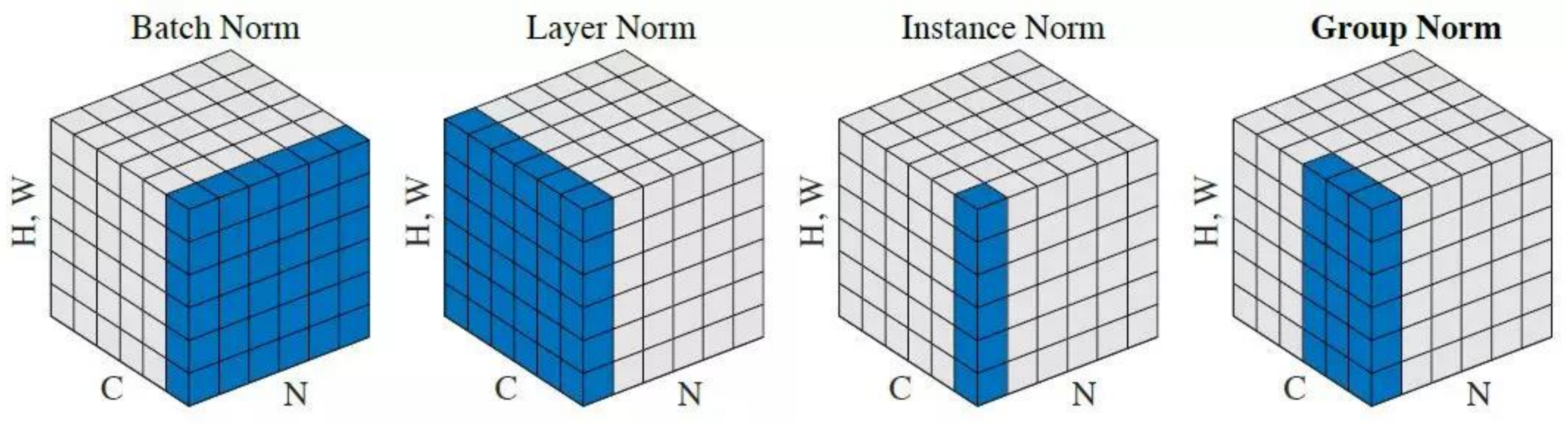
$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

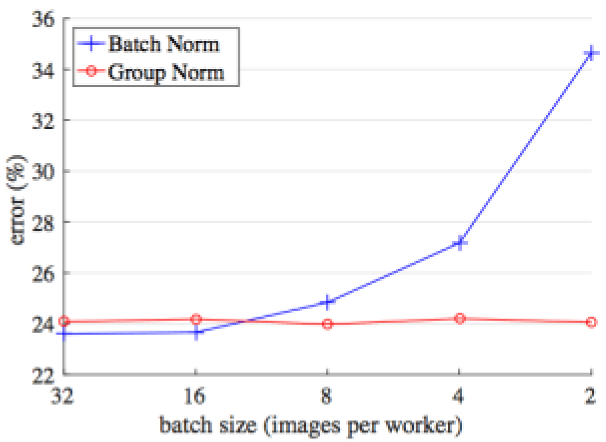
Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

Ioffe & Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift", 2015.

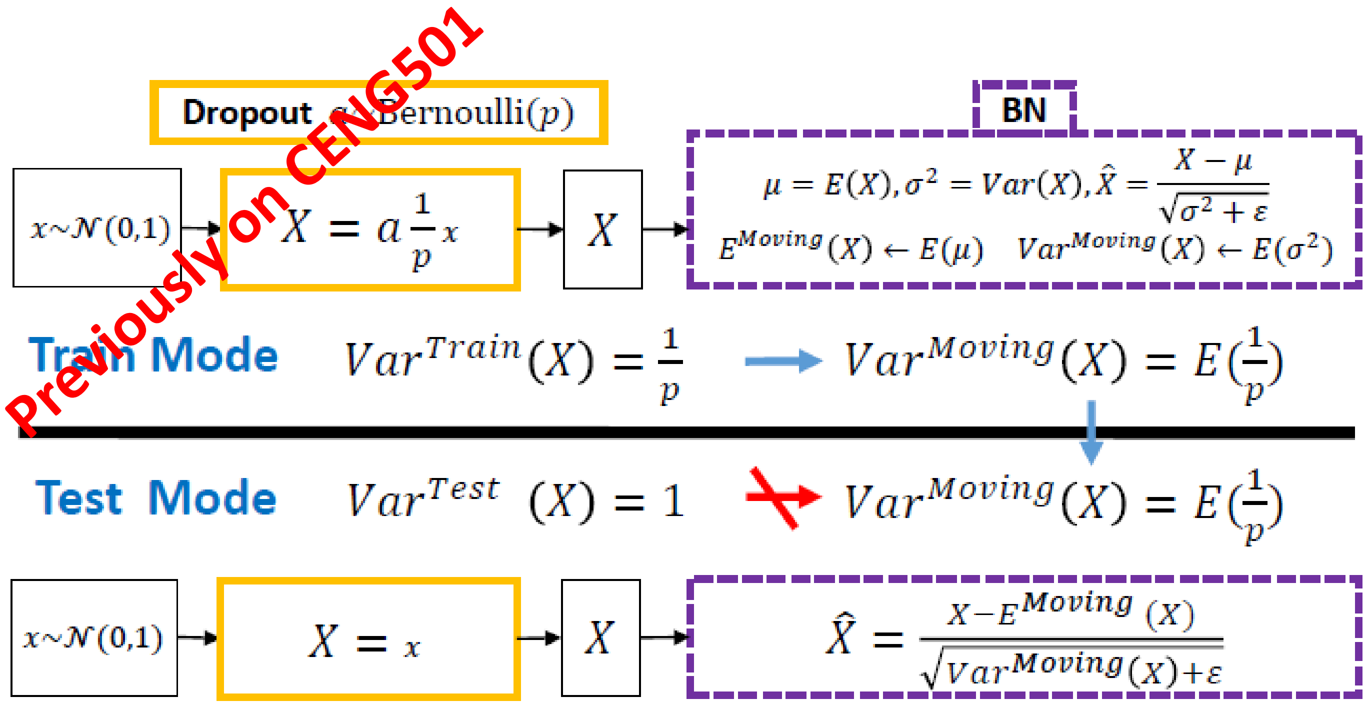
Alternative Normalizations

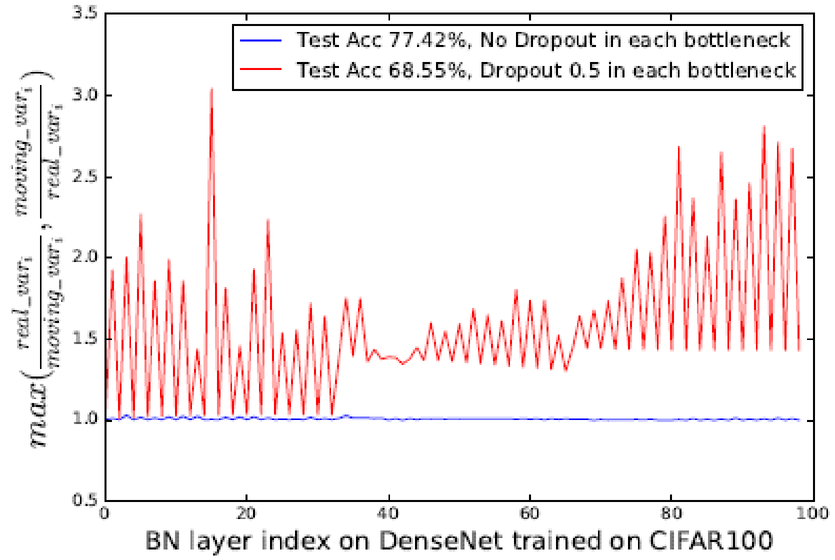
Previously on CENG501





https://medium.com/syncedreview/facebook-ai-proposes-group-normalization-alternative-to-batch-normalization-fb0699bffae7





Understanding the Disharmony between Dropout and Batch Normalization by Variance Shift

Xiang Li[1] Shuo Chen[1] Xiaolin Hu[2] Jian Yang[1]

2018

Since we get a clear knowledge about the disharmony between Dropout and BN, we can easily develop several approaches to combine them together, to see whether an extra improvement could be obtained. In this section, we introduce two possible solutions in modifying Dropout. One is to avoid the scaling on feature-map before every BN layer, by only applying Dropout after the last BN block. Another is to slightly modify the formula of Dropout and make it less sensitive to variance, which can alleviate the shift problem and stabilize the numerical behaviors.

Today

- CNNs
  - Drawbacks of MLPs
  - Benefits of convolution
  - Operations in CNNs

Administrative Notes

- Quiz #2
  - Upload the PDF on ODTUclass.
- Paper Selection
  - Feedback provided
  - Deadline: This Sunday

Convolutional Neural networks: MOTIVATION

Disadvantages of MLPs: Dimensionality

- The number of parameters in an MLP is high for practical problems
  - e.g., for grayscale images with 1000x1000 resolution, a fully-connected layer with 1000 neurons requires 10^9 parameters.
- The number of parameters in an MLP increases quadratically with an increase in input dimensionality
- For example, for a fully-connected layer with n_{in} input neurons and n_{out} output neurons:
  - Number of parameters: $n_{in} \times n_{out}$
  - Assuming proportional decrease in layer size, e.g. $n_{out} = n_{in}/10$, gives: $n_{in} \times n_{out} = n_{in}^2/10$
  - Increasing n_{in} by d yields a change of $\mathcal{O}(d^2)$.
- This is a problem because:
  - More parameters => larger model size & more computational complexity.
- Teaser for CNNs:
  - Input size does not affect model size (in general)

Disadvantages of MLPs: Curse of Dimensionality

- For conventional ML methods:
  - The number of required samples for obtaining small error increases exponentially with input dimensions

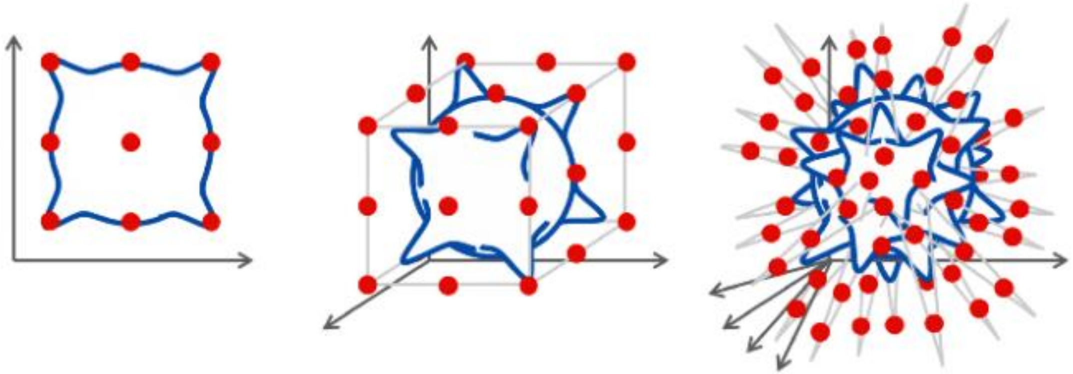


Illustration of the curse of dimensionality: in order to approximate a Lipschitz-continuous function composed of Gaussian kernels placed in the quadrants of a d-dimensional unit hypercube (blue) with error ε, one requires $\mathcal{O}(1/\varepsilon^d)$ samples (red points).

Figure: https://towardsdatascience.com/geometric-foundations-of-deep-learning-94cdd45b451d

- For deep networks:
  - This does not seem to be an issue for deep networks (see e.g. Poggio & Liao, 2018).

Poggio, T., & Liao, Q. (2018). Theory I: Deep networks and the curse of dimensionality. Bulletin of the Polish Academy of Sciences: Technical Sciences, (6).

Disadvantages of MLPs: Equivariance

- Vectorizing an image breaks patterns in consecutive pixels.
  - Shifting one pixel means a whole new vector
  - Makes learning more difficult
  - Requires more data to generalize

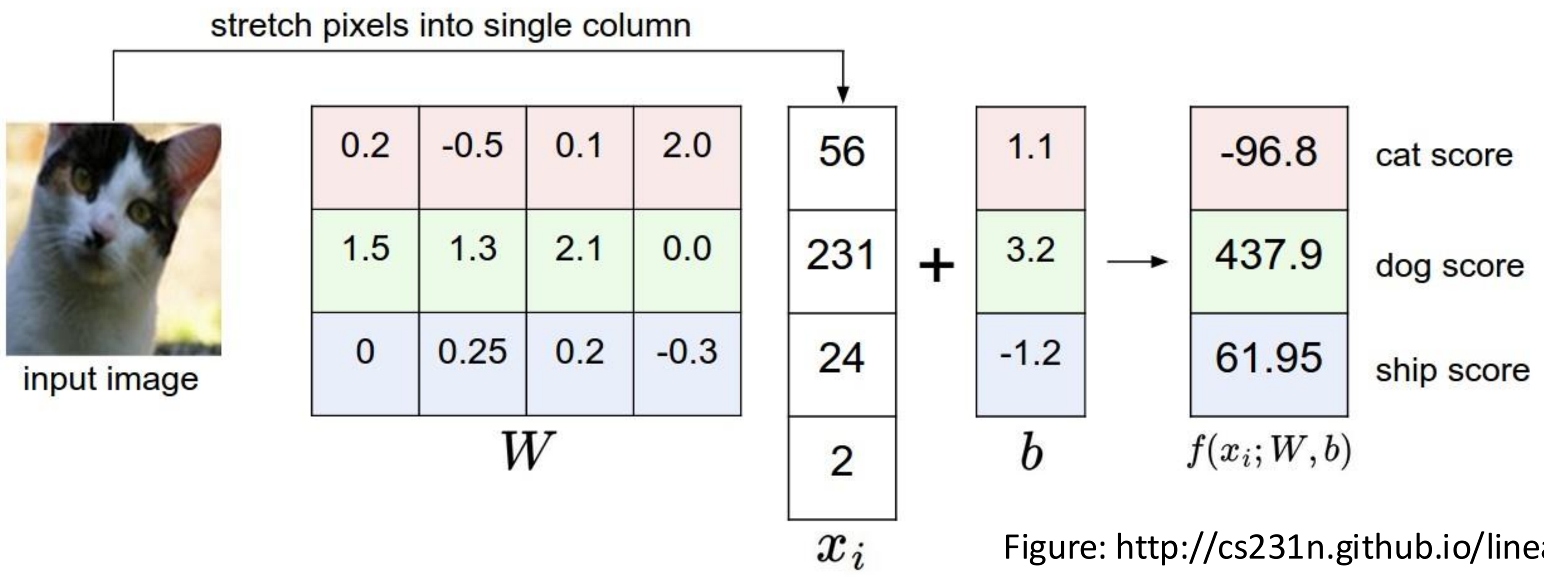
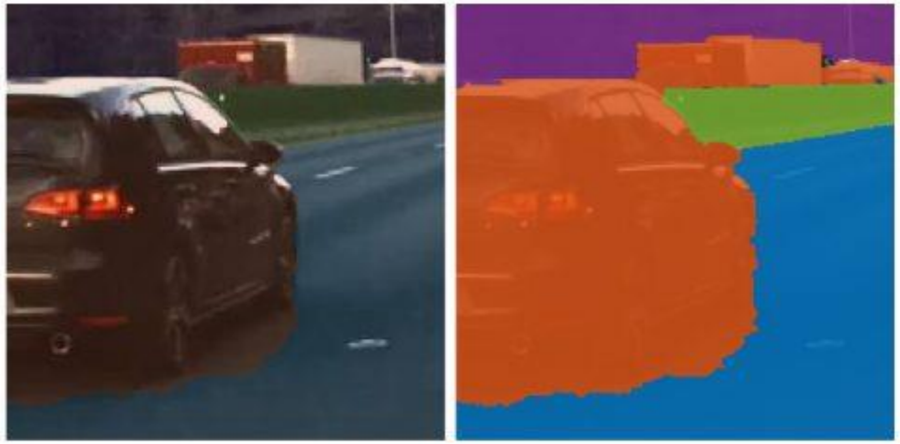


Figure: http://cs231n.github.io/linear-classify/

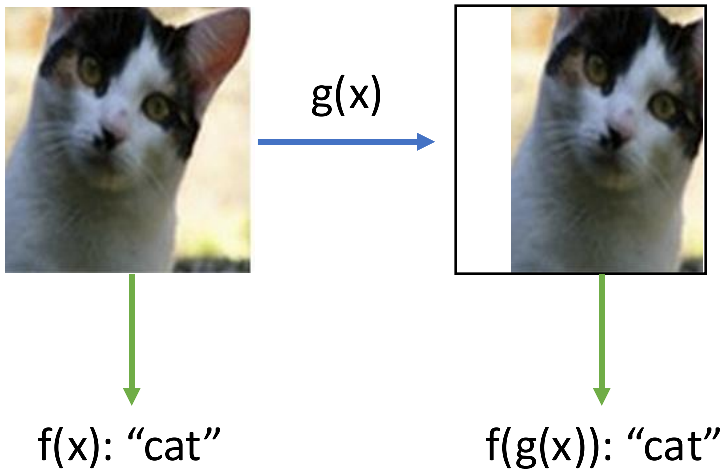
Equivariance vs. Invariance



https://www.mathworks.com/discovery/image-segmentation.html

- Equivariant problem: image segmentation.
  - f(g(x)) = g(f(x))

- Invariant problem: object recognition.
  - f(g(x)) = f(x)

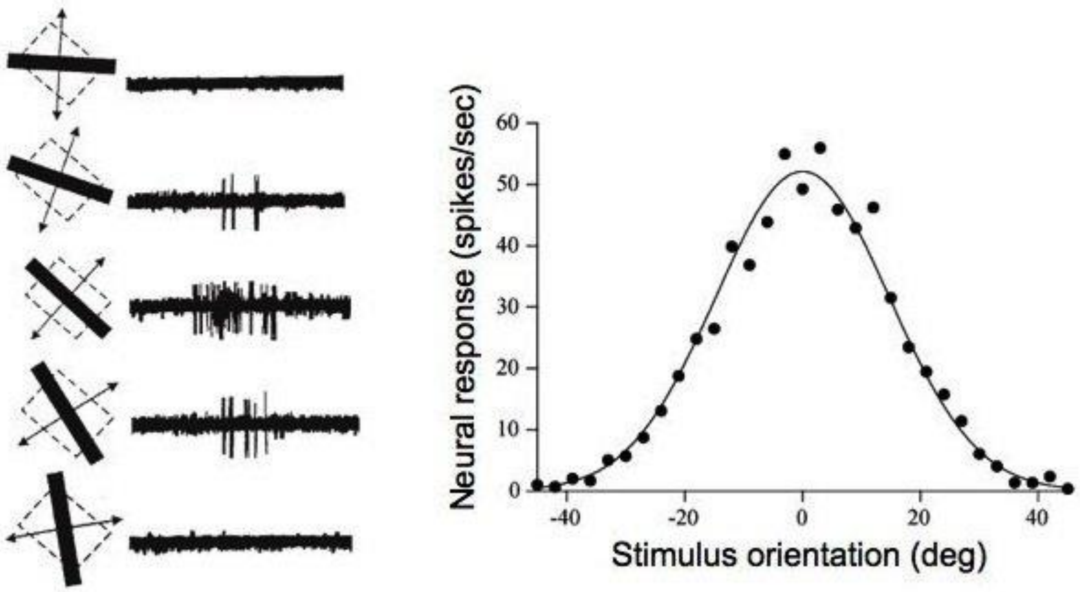


- Pooling provides invariance, convolution provides equivariance.

An Alternative to MLPs

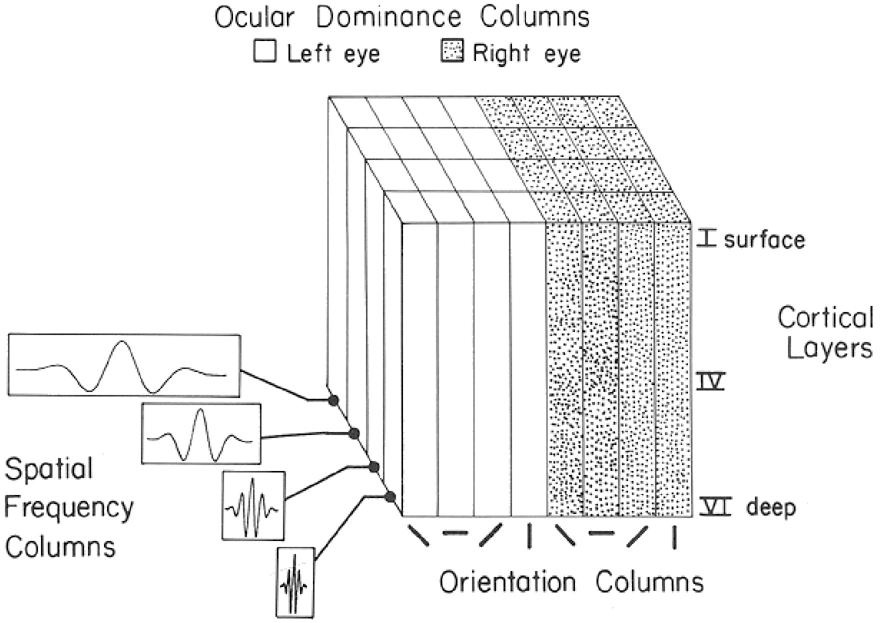
Solution (inspiration):

- Hubel & Wiesel: Brain neurons are not fully connected. They have local receptive fields



Hubel & Wiesel, 1968

CEN301



Model of Striate Module in Cats

http://fourier.eng.hmc.edu/e180/lectures/retina/node1.html

An Alternative to MLPs

Solution (inspiration):

- Hubel & Wiesel: Brain neurons are not fully connected. They have local receptive fields

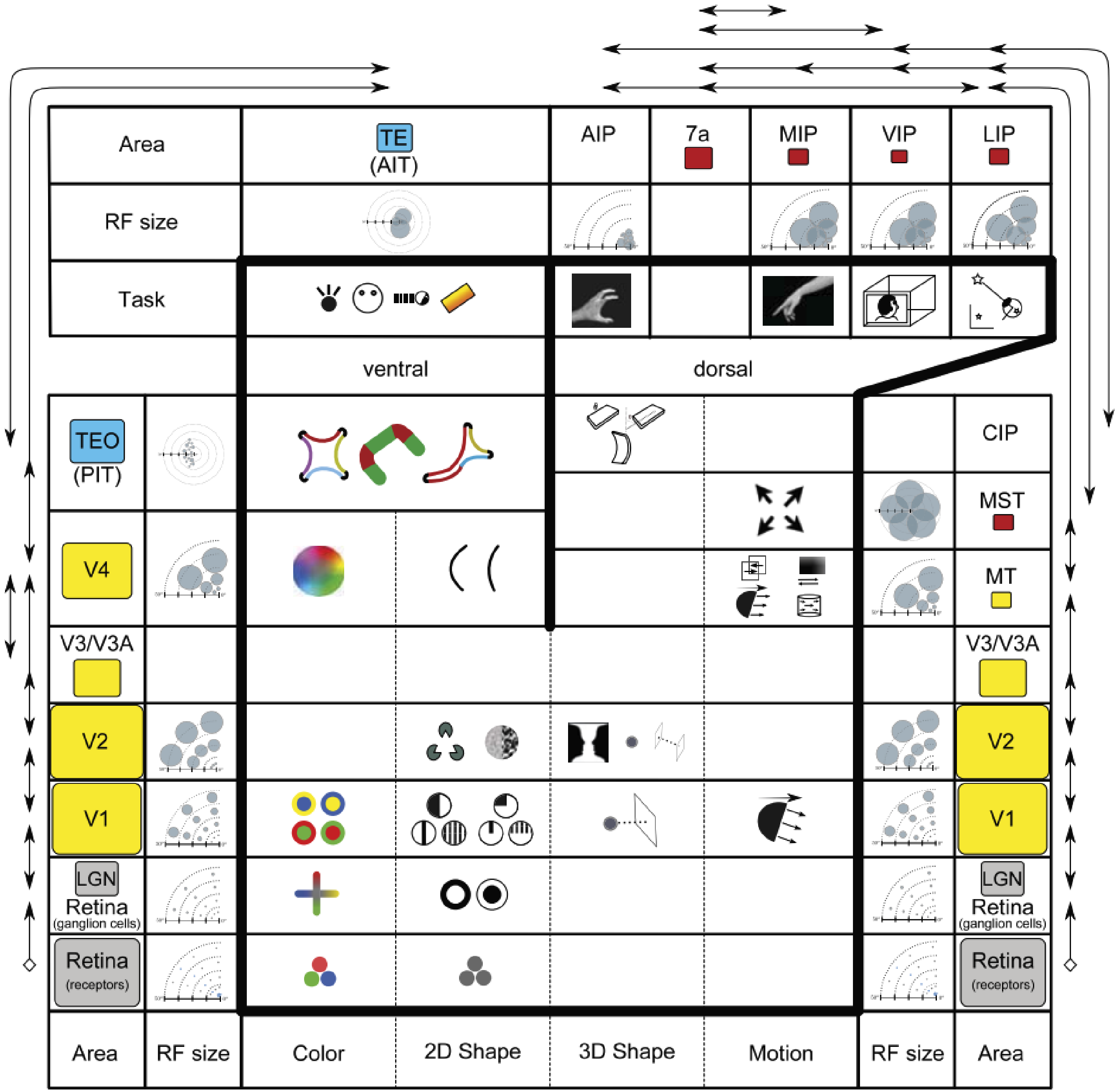


Figure: N. Krueger, P. Janssen, S. Kalkan, M. Lappe, A. Leonardis, J. Piater, A. J. Rodriguez-Sanchez, L. Wiskott, "Deep Hierarchies in the Primate Visual Cortex: What Can We Learn For Computer Vision?", IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), 2013.

An Alternative to MLPs

Solution: Neocognitron (Fukushima, 1979):

A neural network model unaffected by shift in position, applied to Japanese handwritten character recognition.

- S (simple) cells: local feature extraction.
- C (complex) cells: provide tolerance to deformation, e.g. shift.
- Self-organized learning method.

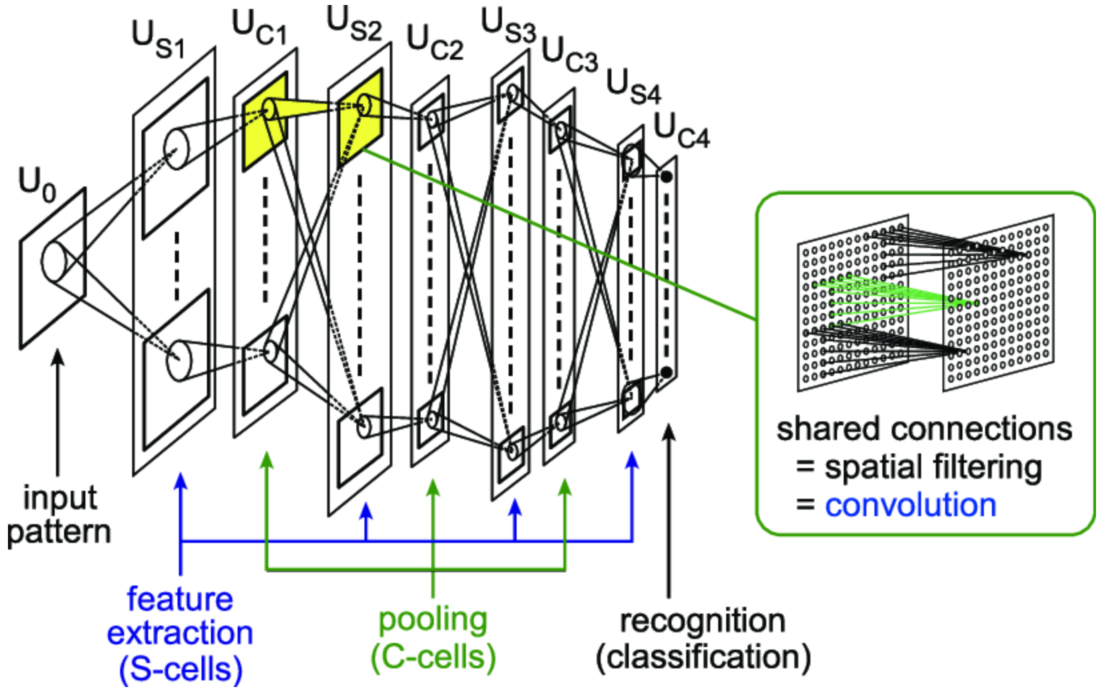


Figure: Fukushima (2019), Recent advances in the deep CNN neocognitron.

An Alternative to MLPs

Solution:

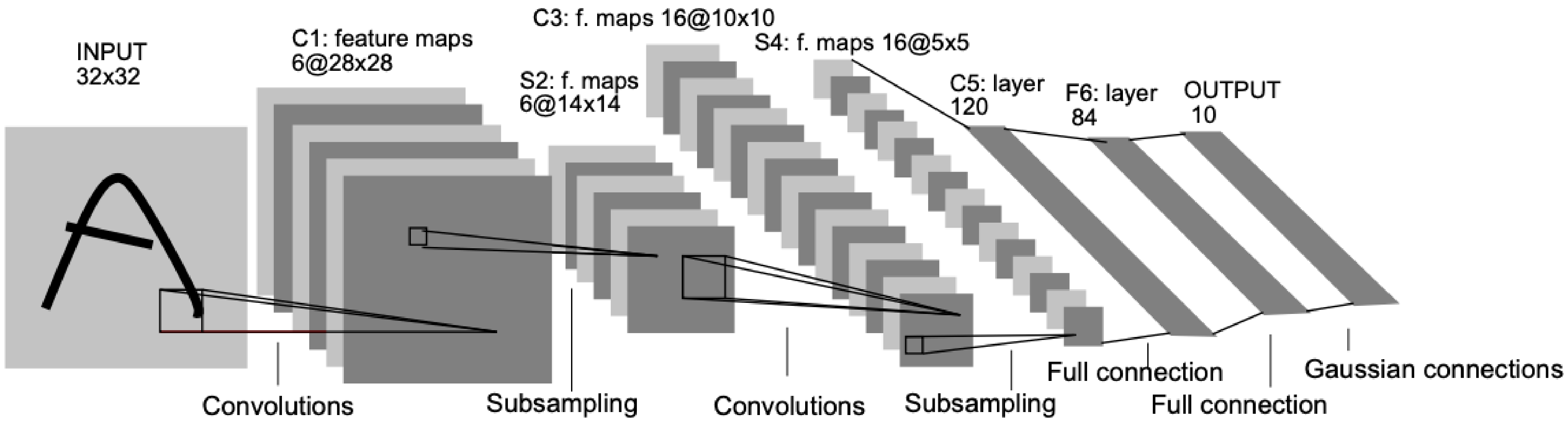
Neocognitron's self-organized learning method (Fukushima, 2019):

"For training intermediate layers of the neocognitron, the learning rule called AiS (Add-if-Silent) is used. Under the AiS rule, a new cell is generated and added to the network if all postsynaptic cells are silent in spite of non-silent presynaptic cells. The generated cell learns the activity of the presynaptic cells in one-shot. Once a cell is generated, its input connections do not change any more. Thus the training process is very simple and does not require time-consuming repetitive calculation."

An Alternative to MLPs

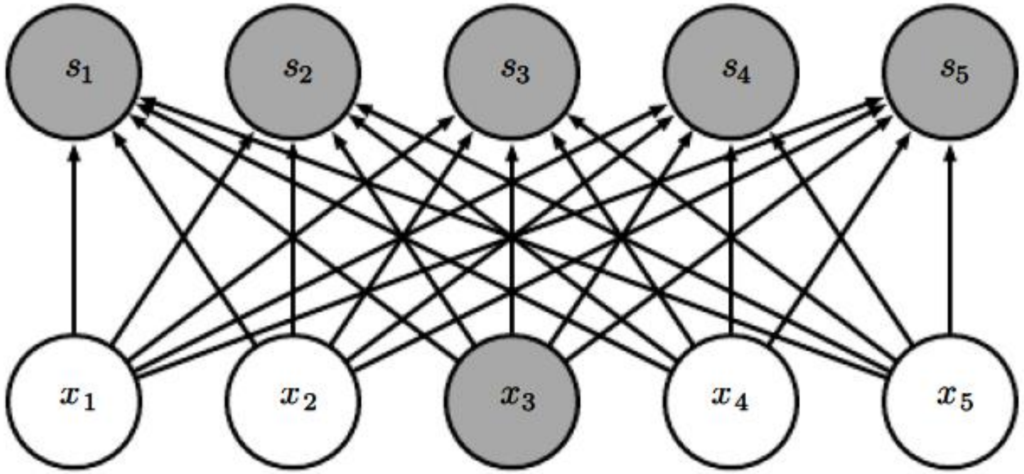
Solution: Convolutional Neural Networks (Lecun, 1998)

- Gradient descent
- Weights shared
- Document recognition



Lecun, 1998

CNNs: Underlying Principle



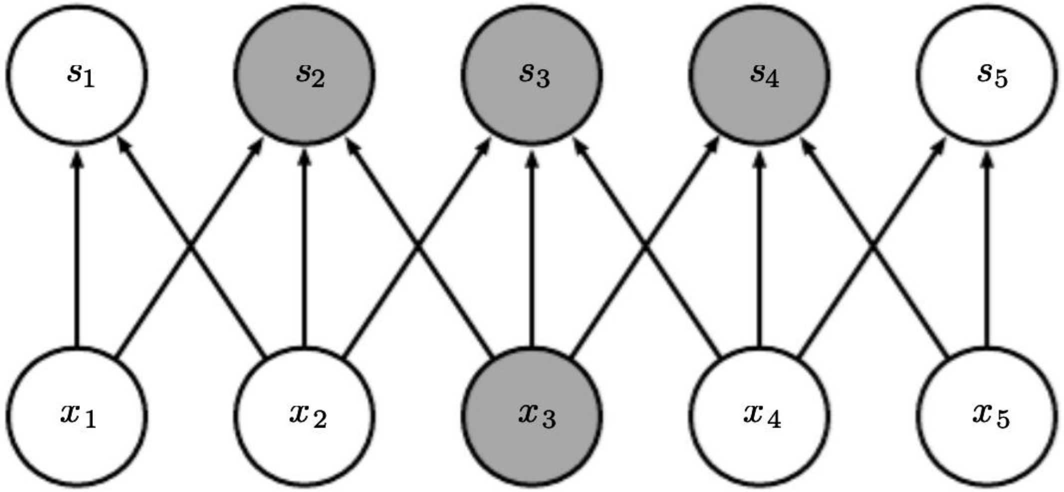
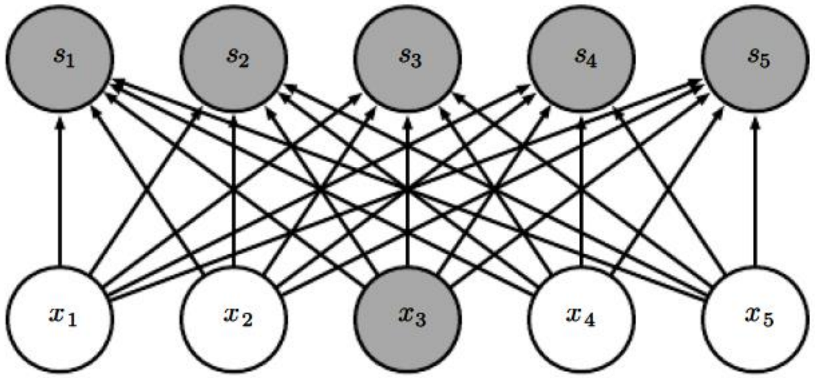


Figure: Goodfellow et al., “Deep Learning”, MIT Press, 2016.

CNNs vs. MLPs: Curse of Dimensionality

- A fully-connected network has too many parameters
  - On CIFAR-10:
    - Images have size 32x32x3 ➔ one neuron in hidden layer has 3072 weights!
  - With images of size 1024x1024x3 ➔ one neuron in hidden layer has 3,145,728 weights!
  - This explodes quickly if you increase the number of neurons & layers.
- Alternative: enforce local connectivity!



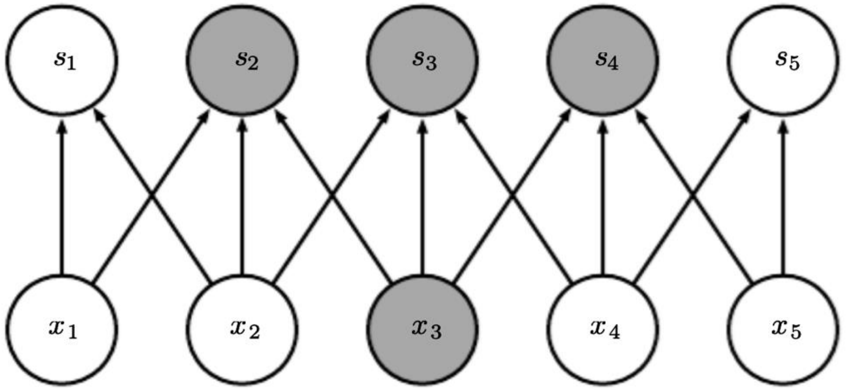
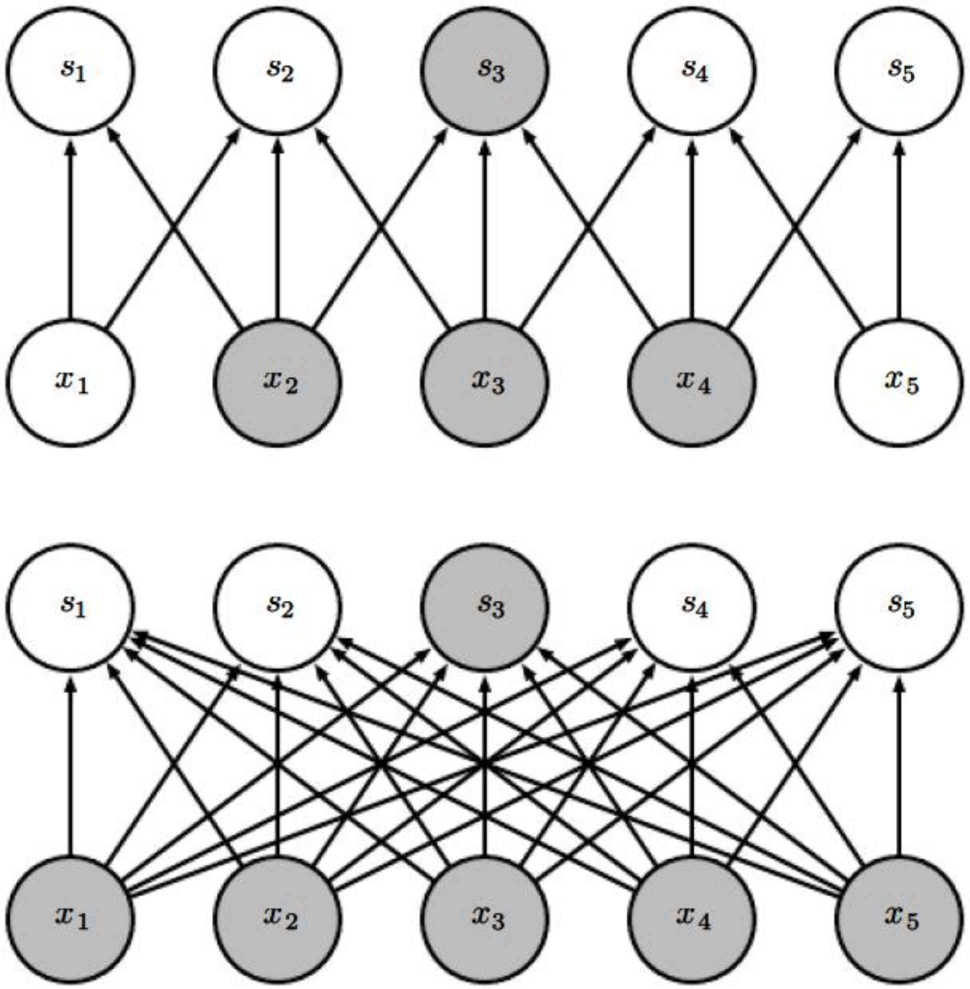


Figure: Goodfellow et al., “Deep Learning”, MIT Press, 2016.

CNNs vs. MLPs: Curse of Dimensionality



When things go deep, an output may depend on all or most of the input:

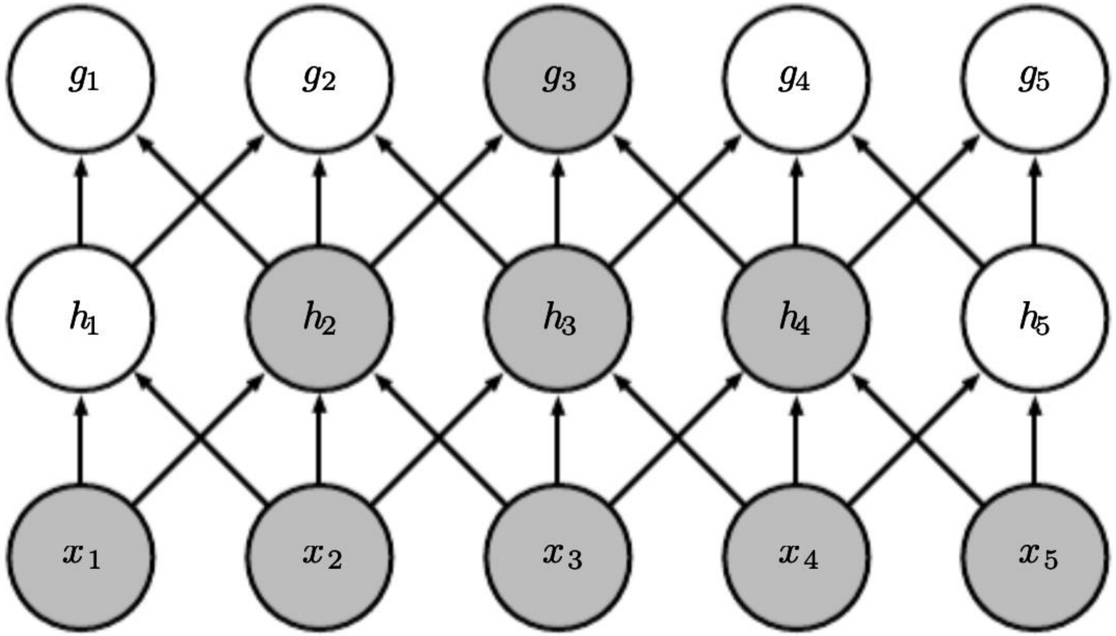


Figure: Goodfellow et al., “Deep Learning”, MIT Press, 2016.

How Many Samples are Needed to Learn a Convolutional Neural Network?

Simon S. Du*[1], Yining Wang*[1], Xiyu Zhai[2], Sivaraman Balakrishnan[3], Ruslan Salakhutdinov[1], and Aarti Singh[1]

[1]Machine Learning Department, Carnegie Mellon University
[2]University of Cambridge
[3]Department of Statistics, Carnegie Mellon University

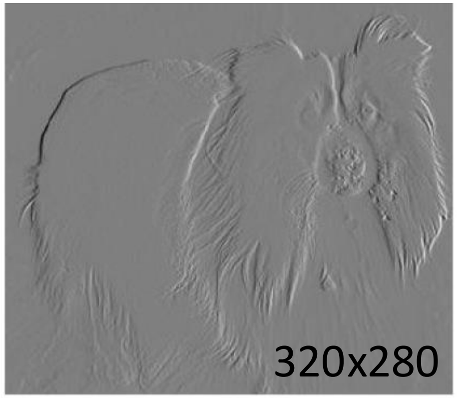
May 22, 2018

Abstract

A widespread folklore for explaining the success of convolutional neural network (CNN) is that CNN is a more compact representation than the fully connected neural network (FNN) and thus requires fewer samples for learning. We initiate the study of rigorously characterizing the sample complexity of learning convolutional neural networks. We show that for learning an m-dimensional convolutional filter with linear activation acting on a d-dimensional input, the sample complexity of achieving population prediction error of ϵ is $\widetilde{O}(m/\epsilon^2)$[1], whereas its FNN counterpart needs at least $\Omega(d/\epsilon^2)$ samples. Since $m \ll d$, this result demonstrates the advantage of using CNN. We further consider the sample complexity of learning a one-hidden-layer CNN with linear activation where both the m-dimensional convolutional filter and the r-dimensional output weights are unknown. For this model, we show the sample complexity is $\widetilde{O}\left((m+r)/\epsilon^2\right)$ when the ratio between the stride size and the filter size is a constant. For both models, we also present lower bounds showing our sample complexities are tight up to logarithmic factors. Our main tools for deriving these results are localized empirical process and a new lemma characterizing the convolutional structure. We believe these tools may inspire further developments in understanding CNN.

CNNs vs. MLPs: Curse of Dimensionality

- Parameter sharing
  - In regular ANN, each weight is independent
- In CNN, a layer might re-apply the same convolution and therefore, share the parameters of a convolution
  - Reduces storage and learning time



- For a neuron in the next layer:
  - With ANN: 320x280x320x280 multiplications
  - With CNN: 320x280x3x3 multiplications

Figure: Goodfellow et al., “Deep Learning”, MIT Press, 2016.

CNNs vs. MLPs: Equivariance

- Equivariant to translation
  - The output will be the same, just translated, since the weights are shared.

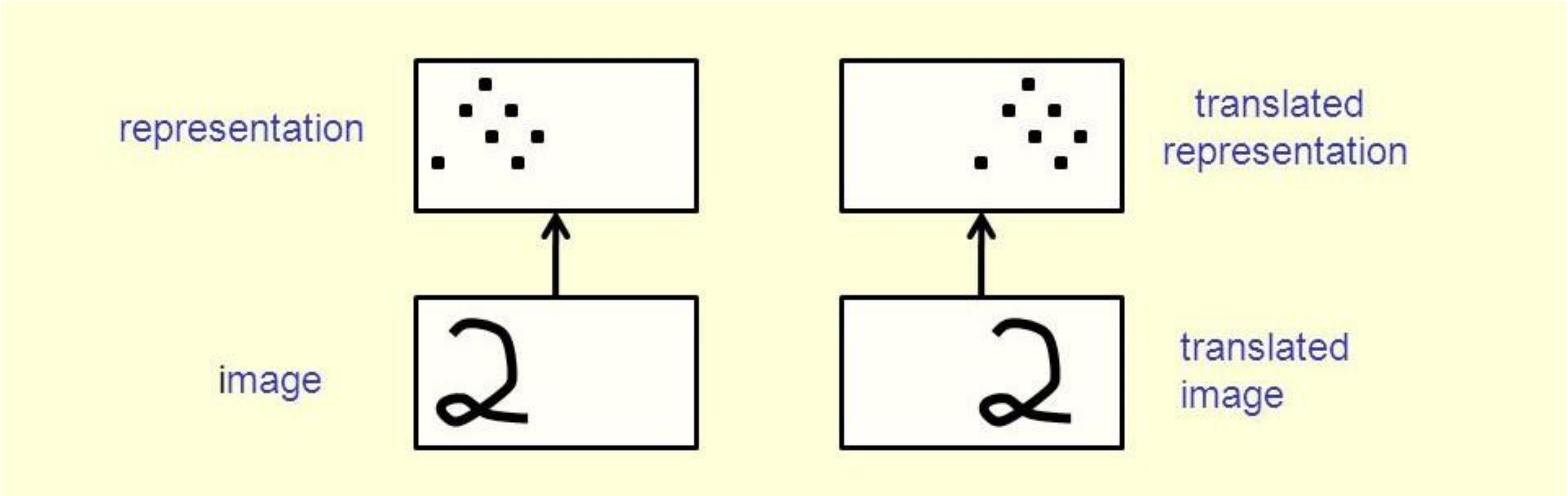
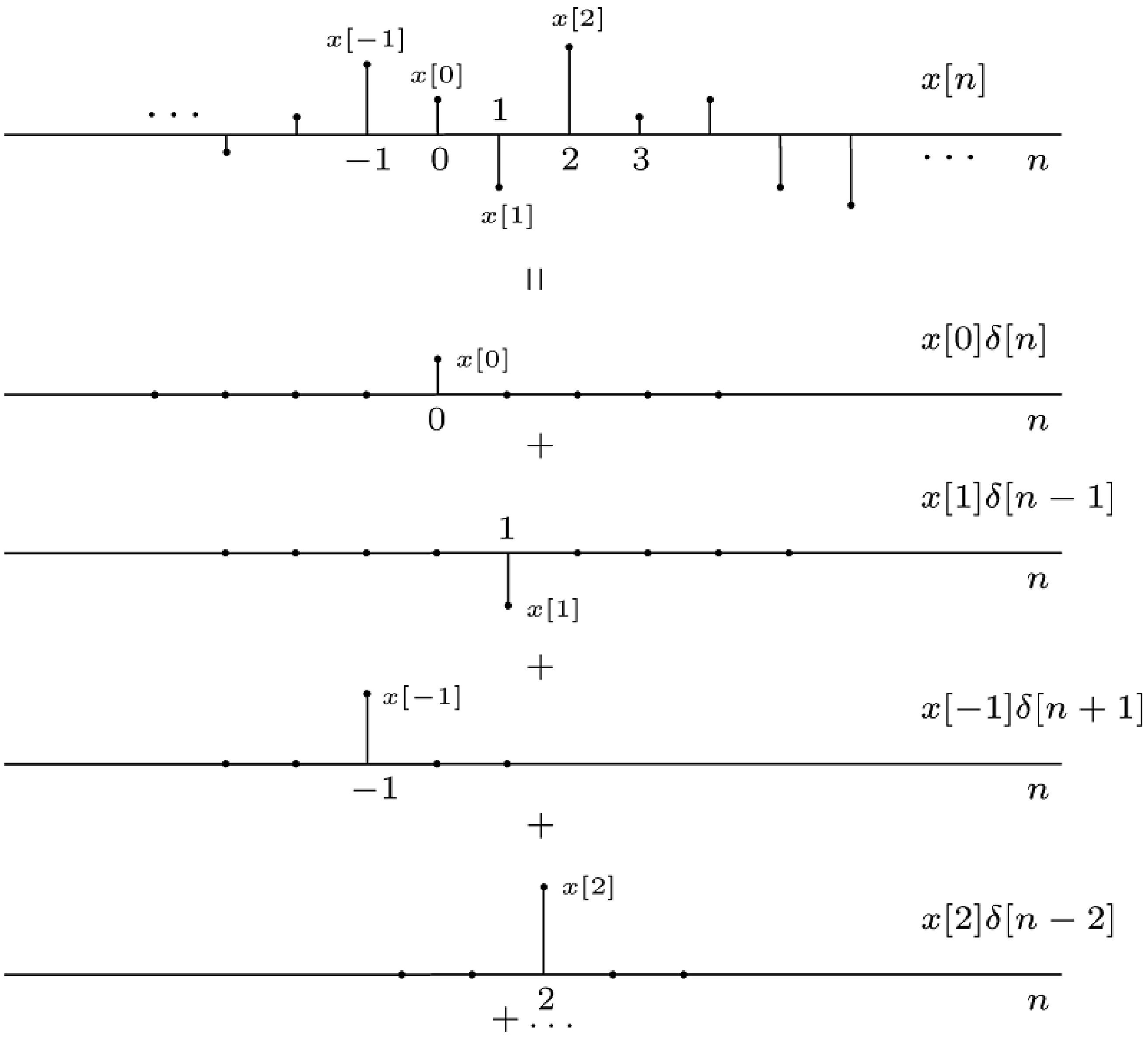


Figure: https://towardsdatascience.com/translational-invariance-vs-translational-equivariance-f9fbc8fca63a

- Not equivariant to scale or rotation.

A crash course on Convolution

Formulating Signals in Terms of Impulse Signal



Alan V. Oppenheim and Alan S. Willsky

Formulating Signals in Terms of Impulse Signal

$$x[n] = \ldots + x[-2]\delta[n+2] + x[-1]\delta[n+1] + x[0]\delta[n] + x[1]\delta[n-1] + \ldots$$

$$\Downarrow$$

Important to note the "-" sign

$$x[n] = \sum_{k=-\infty}^{+\infty} \underbrace{x[k]}_{\text{Coefficients}} \underbrace{\delta[n-k]}_{\text{Basic Signals}}$$

Alan V. Oppenheim and Alan S. Willsky

Unit Sample Response

- Now suppose the system is **LTI**, and define the *unit sample response* $h[n]$:

$$\delta[n] \longrightarrow h[n]$$

$$\Downarrow$$

From Time-Invariance:

$$\delta[n-k] \longrightarrow h[n-k]$$

From Linearity:

$$x[n] = \sum_{k=-\infty}^{+\infty} x[k]\,\delta[n-k] \longrightarrow y[n] = \underbrace{\sum_{k=-\infty}^{+\infty} x[k]\,h[n-k]}_{\textit{convolution sum}} = x[n] * h[n]$$

Alan V. Oppenheim and Alan S. Willsky

Conclusion

The output of *any* DT LTI System is a convolution of the input signal with the unit-sample response, *i.e.*

$$\text{Any DT LTI} \longleftrightarrow y[n] = x[n] * h[n]$$

$$= \sum_{k=-\infty}^{+\infty} x[k]\, h[n-k]$$

As a result, any DT LTI Systems are *completely characterized* by its unit sample response

Alan V. Oppenheim and Alan S. Willsky

Power of convolution

- Describe a “system” (or operation) with a very simple function (impulse response).
- Determine the output by convolving the input with the impulse response

Convolution

- Definition of continuous-time convolution

$$x(t) * h(t) = \int x(\tau)h(t-\tau)\, d\tau$$

$$h(\tau) \xrightarrow{\textit{Flip}} h(-\tau) \xrightarrow{\textit{Slide}} h(t-\tau) \xrightarrow{\textit{Multiply}}$$

$$x(\tau)h(t-\tau) \xrightarrow{\textit{Integrate}} \int_{-\infty}^{+\infty} x(\tau)h(t-\tau)d\tau$$

Alan V. Oppenheim and Alan S. Willsky

Convolution

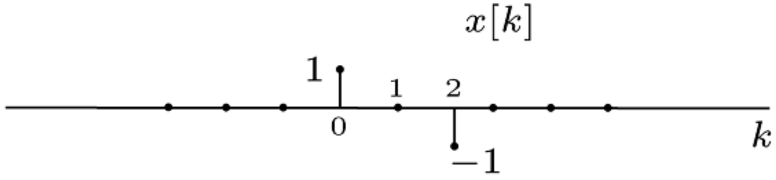
- Definition of discrete-time convolution

$$x[n] * h[n] = \sum x[k]h[n-k]$$

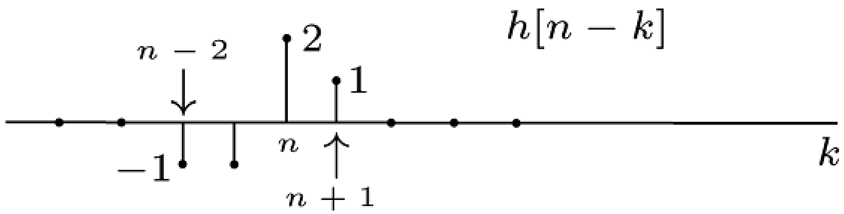
Choose the value of n and consider it fixed

$$y[n] = \sum_{k=-\infty}^{+\infty} x[k]h[n-k]$$

View as functions of k with n fixed



From $x[n]$ and $h[n]$ to $x[k]$ and $h[n\text{-}k]$



Alan V. Oppenheim and Alan S. Willsky

Discrete-time 2D Convolution

- For images, we need two-dimensional convolution:

$$s[i,j] = (I * K)[i,j] = \sum_m \sum_n I[m,n]K[i-m,j-n]$$

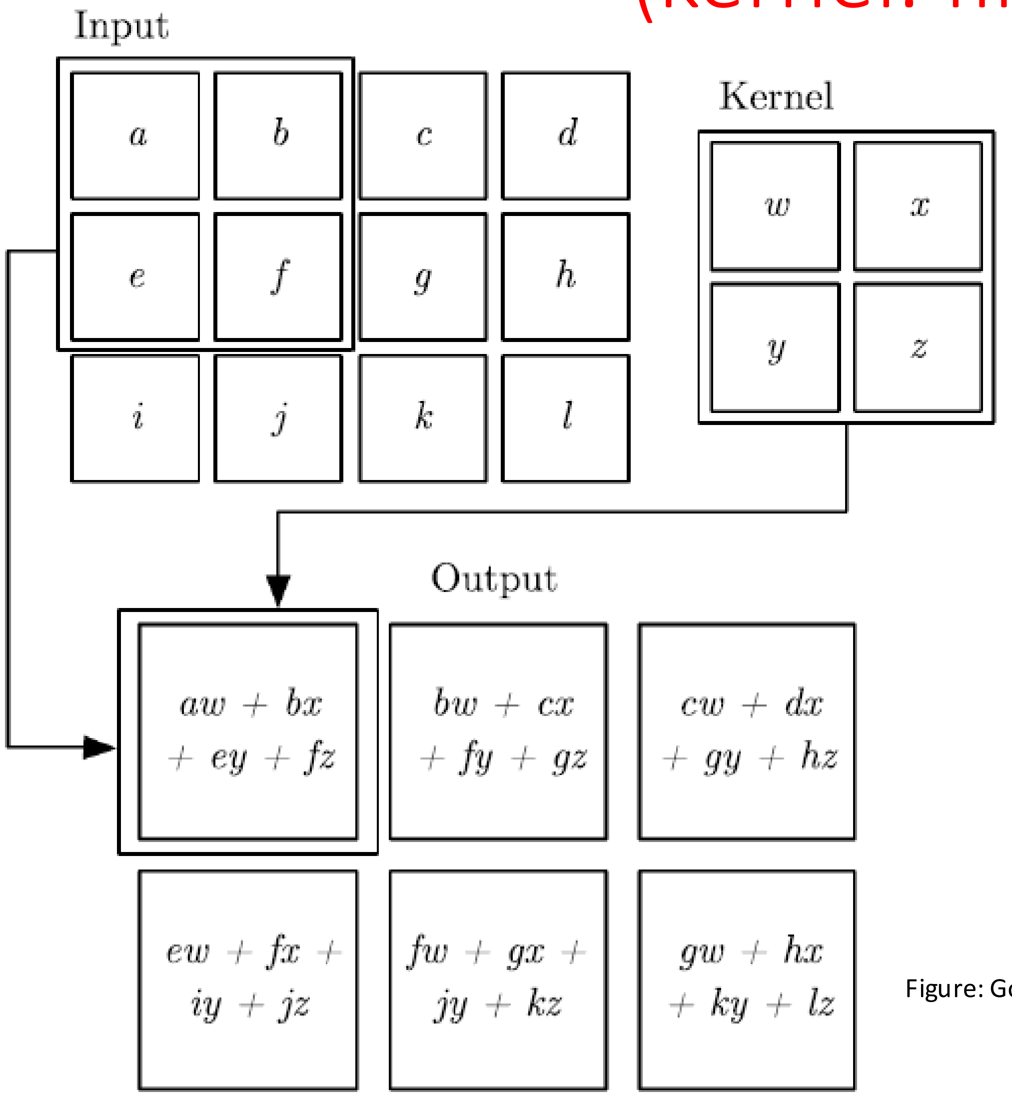
- These multi-dimensional arrays are called tensors
- We have commutative property:

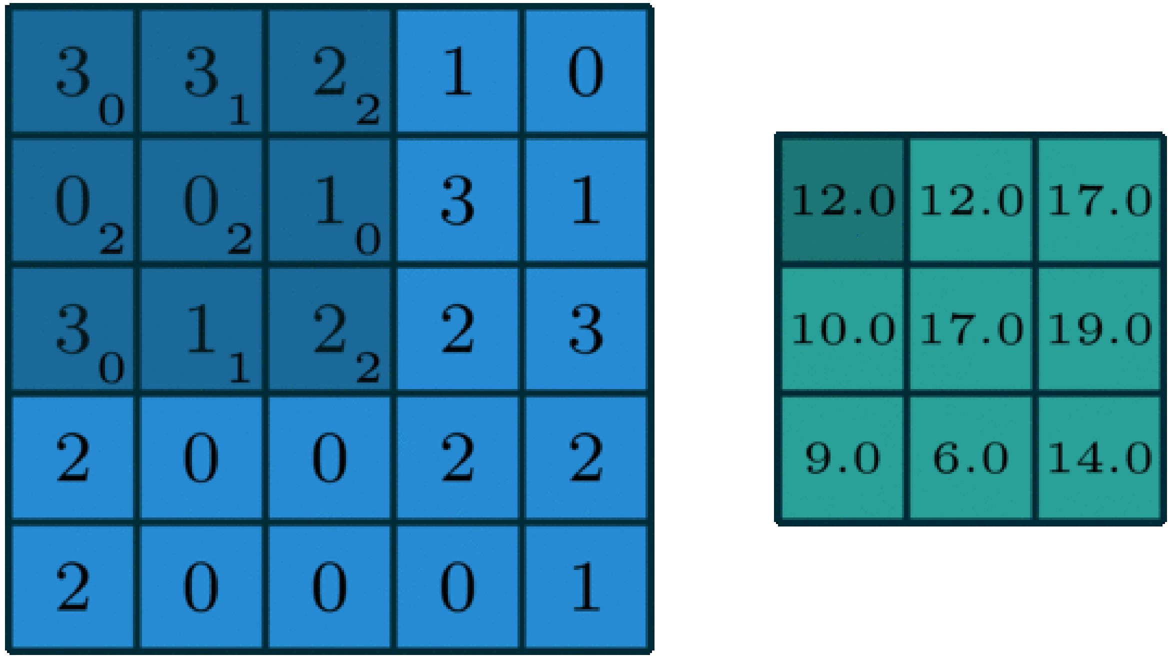
$$s[i,j] = (I * K)[i,j] = \sum_m \sum_n I[i-m,j-n]K[m,n]$$

- Instead of subtraction, we can also write (easy to drive by a change of variables). This is called cross-correlation:

$$s[i,j] = (I * K)[i,j] = \sum_m \sum_n I[i+m,j+n]K[m,n]$$

Example multi-dimensional convolution
(kernel: finite impulse response)





https://github.com/vdumoulin/conv_arithmetic

Figure: Goodfellow et al., “Deep Learning”, MIT Press, 2016.

What can filters do?
Rectangular filter

g[m,n]

$\otimes$

h[m,n]

$=$

f[m,n]

Slide: A. Torralba

What can filters do?
Rectangular filter

⊗ =

h[m,n]

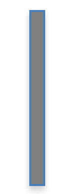
g[m,n]

f[m,n]

Slide: A. Torralba

What can filters do?
Rectangular filter

$\otimes$



h[m,n]

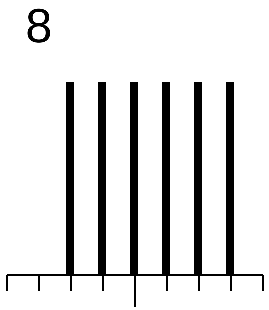
=

f[m,n]

g[m,n]

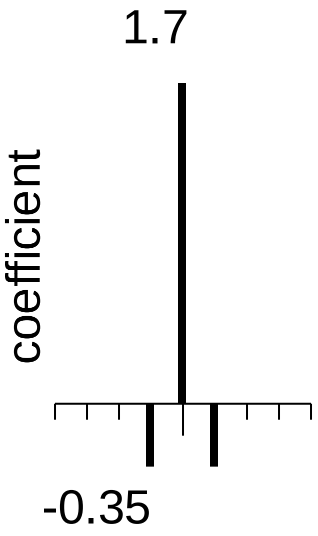
Slide: A. Torralba

What can filters do?
Sharpening filter

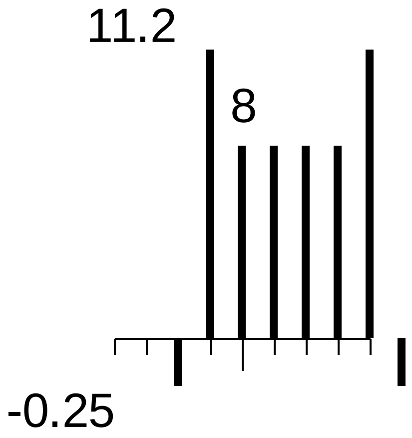


original

filter



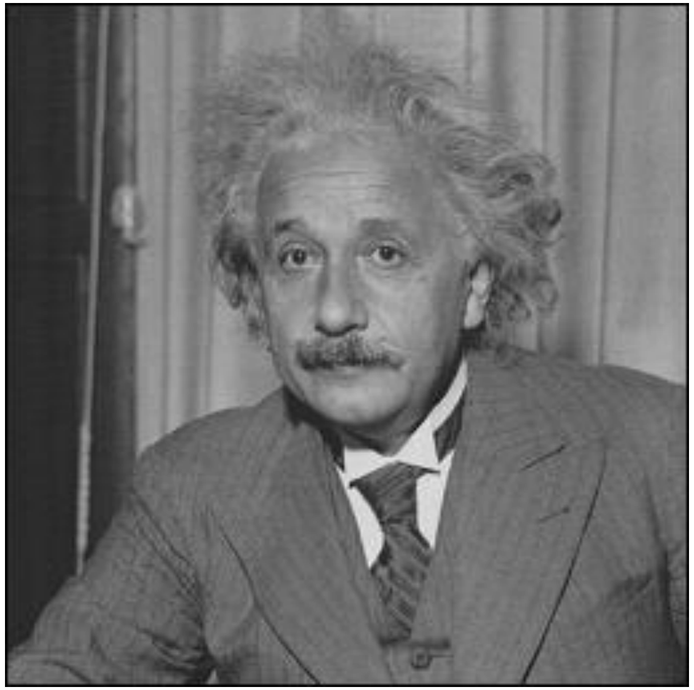
result



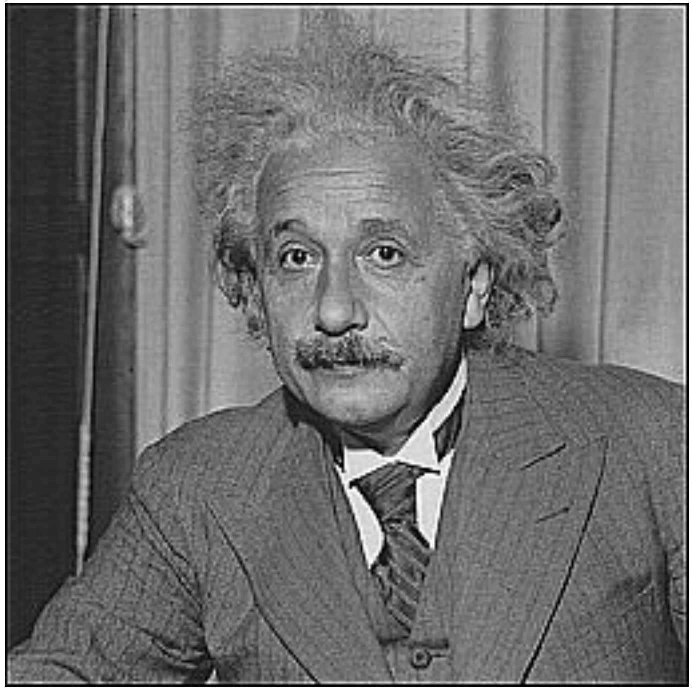
Sharpened (differences are accentuated; constant areas are left untouched).

Slide: A. Torralba

What can filters do? Sharpening filter



before

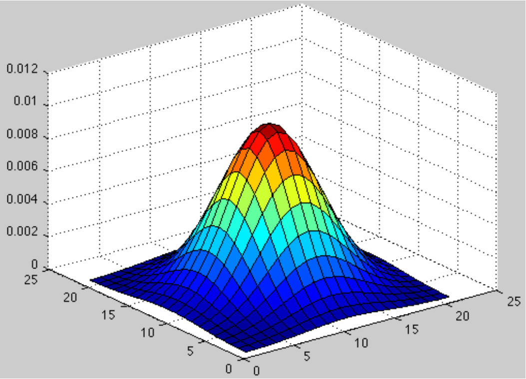


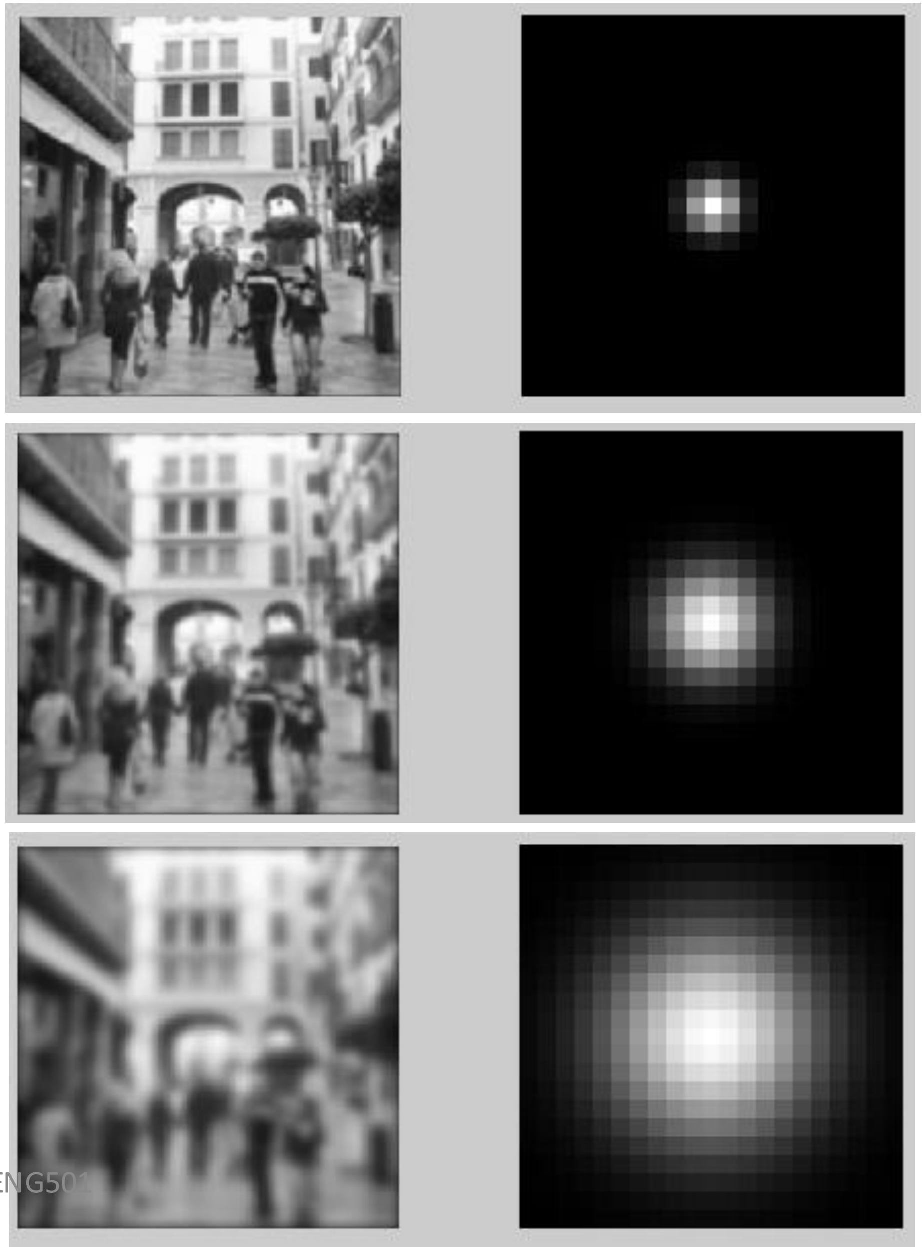
after

Slide: A. Torralba

What can filters do? Gaussian filter

$$G(x, y; \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$





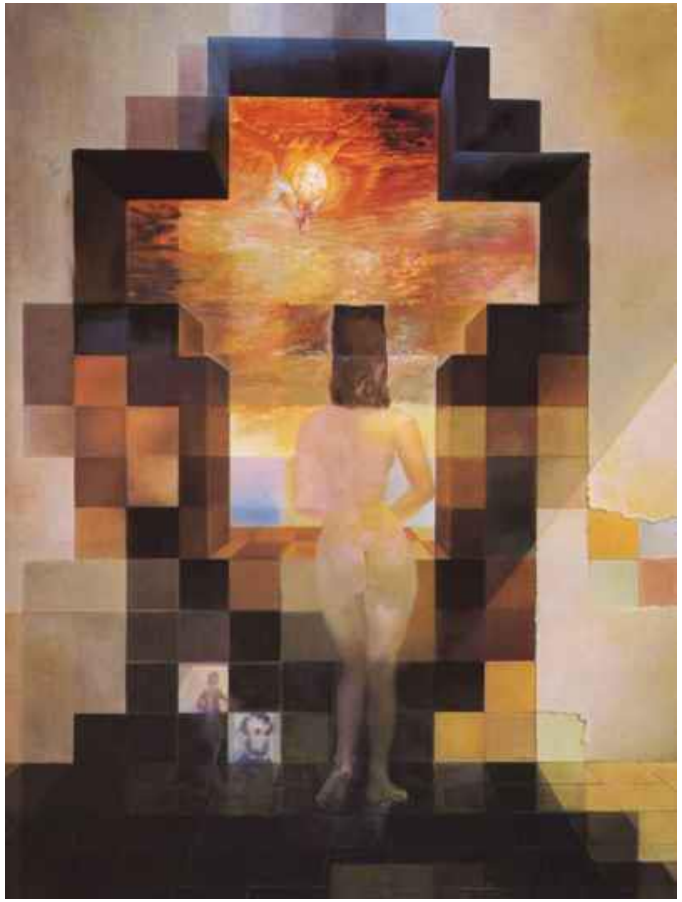
σ=1

σ=2

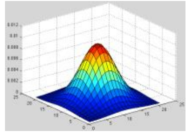
σ=4

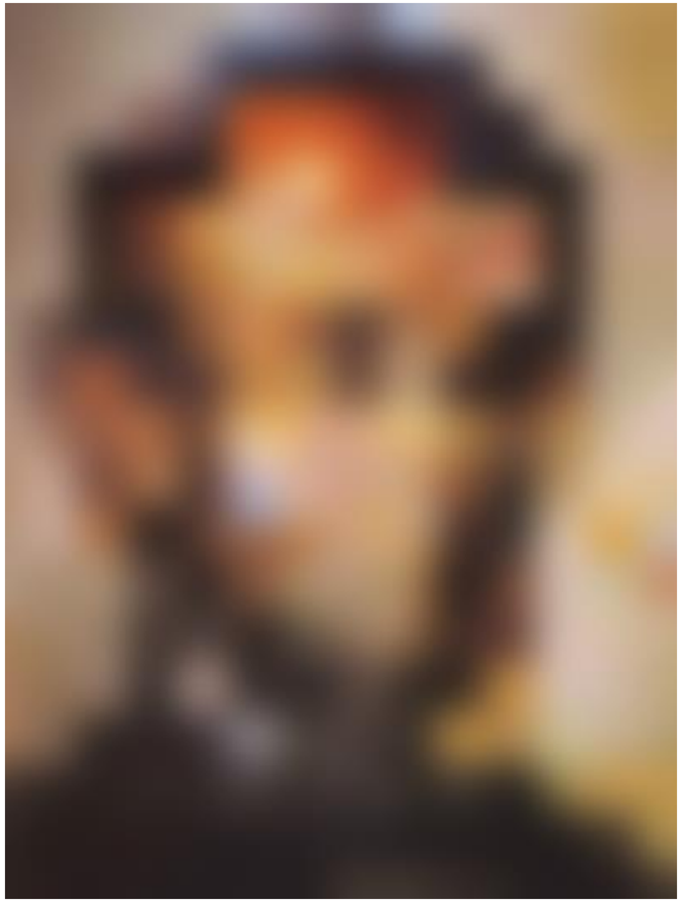
Slide: A. Torralba

Global to Local Analysis



Dali





Slide: A. Torralba

What can filters do?
[-1 1]

g[m,n]

$\otimes$

[-1, 1]

h[m,n]

=

f[m,n]

Slide: A. Torralba

What can filters do?
$[-1\ 1]^T$

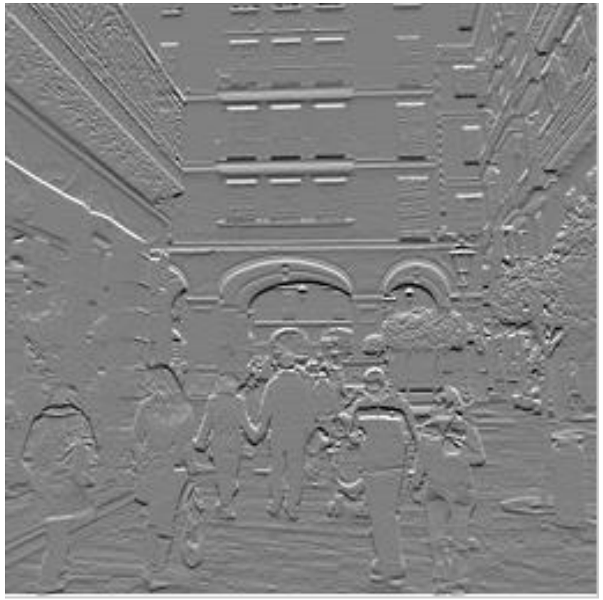
g[m,n]

$\otimes$

$[-1, 1]^T$

h[m,n]

=



f[m,n]

Slide: A. Torralba

Overview of CNN

CNN layers

- Operations in a CNN:
  - Convolution (in parallel) to produce pre-synaptic activations
  - Detector: Non-linear function
  - Pooling: A summary of a neighborhood

- Pooling of a region in a feature/activation map:
  - Max
  - Average
  - L2 norm
  - Weighted average acc. to the distance to the center
  - …

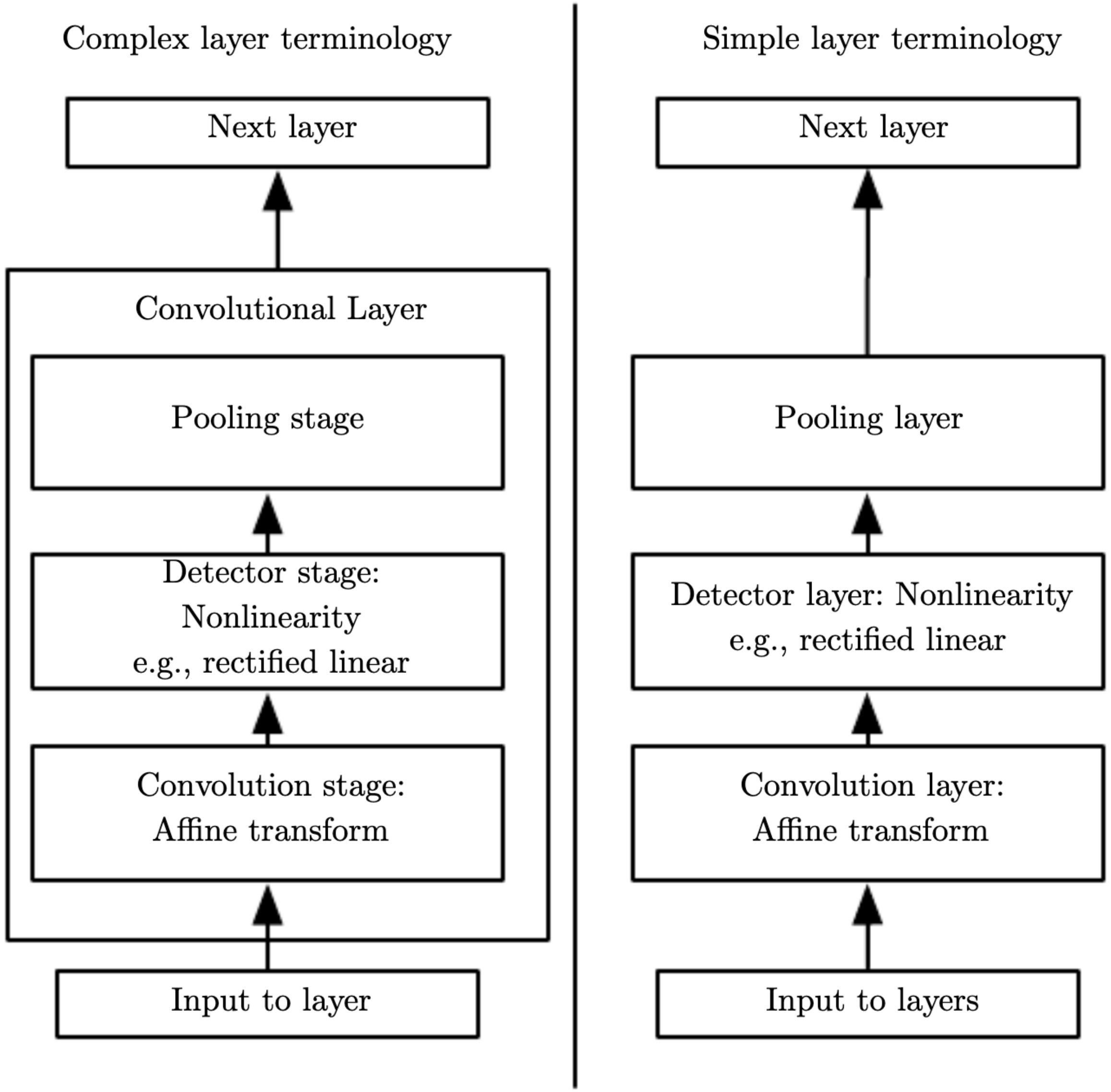
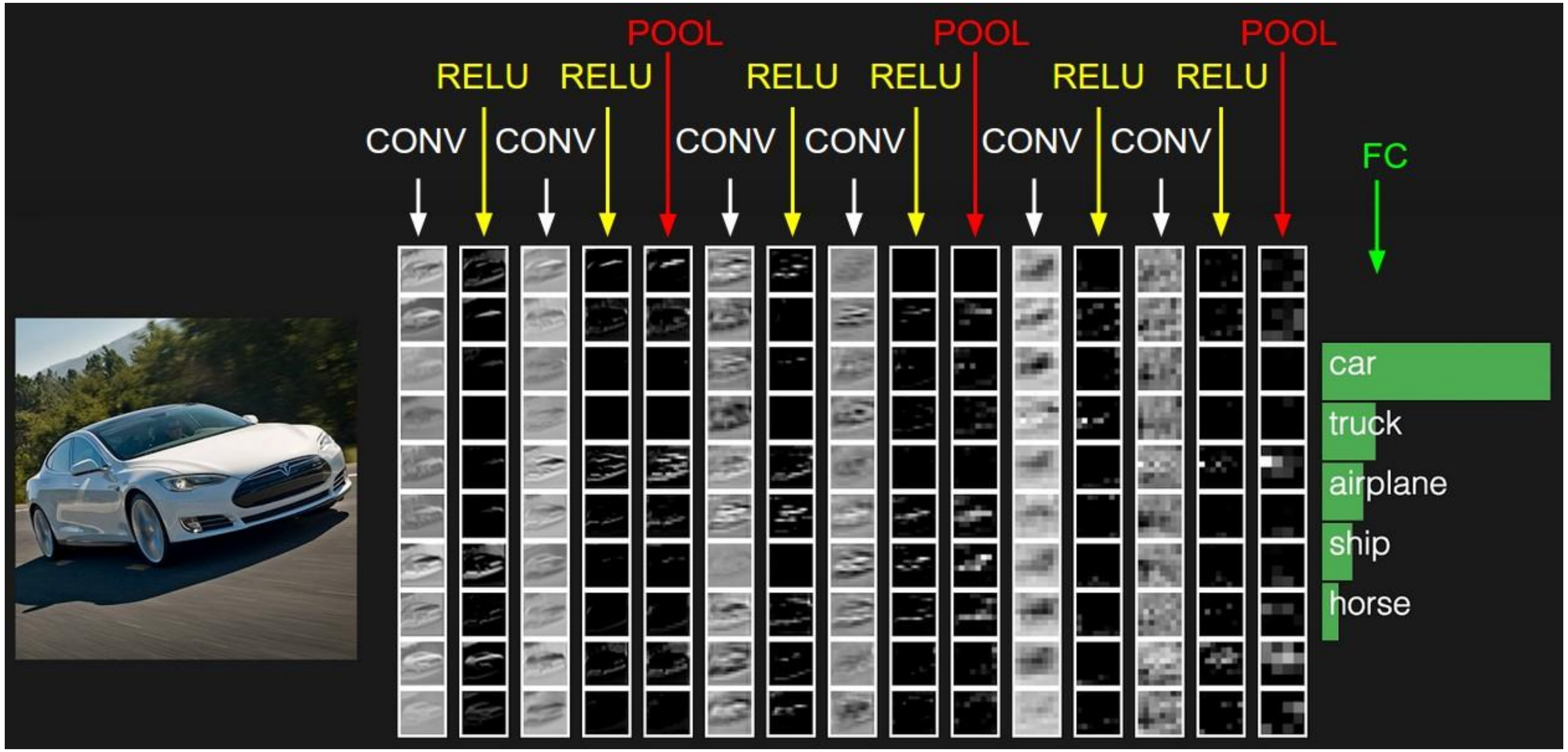


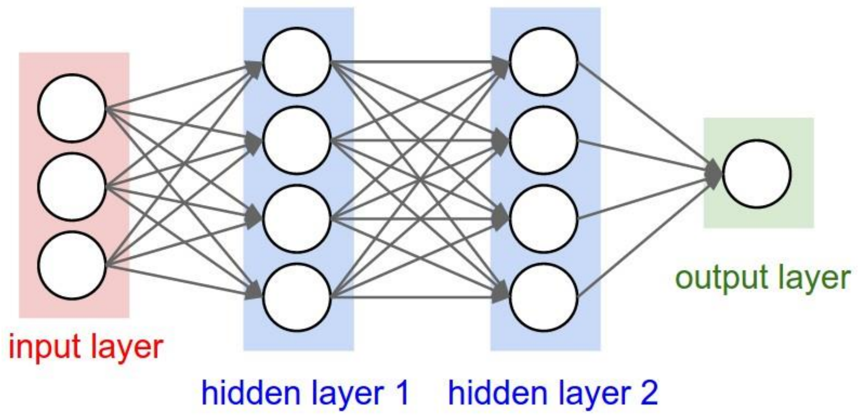
Figure: Goodfellow et al., "Deep Learning", MIT Press, 2016.

An example architecture

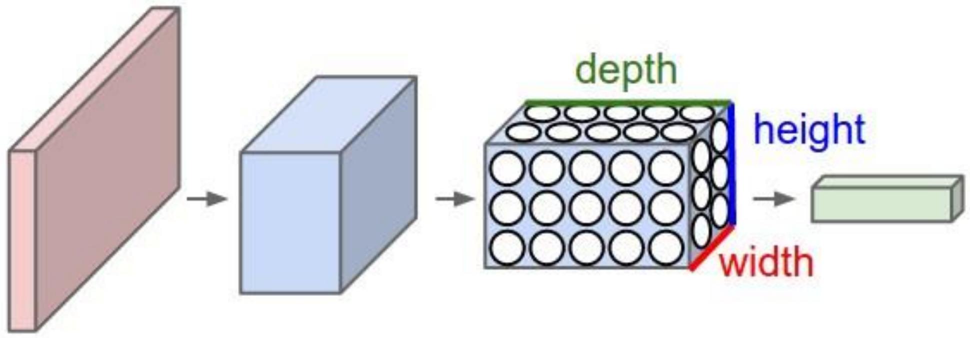


http://cs231n.github.io/convolutional-networks/

Regular ANN



CNN



http://cs231n.github.io/convolutional-networks/

OPERATIONS IN A CNN: Convolution

Convolution in CNN

- The weights correspond to the kernel
- The weights are shared in a channel (depth slice)
- We are effectively learning filters that respond to some part/entities/visual-cues etc.

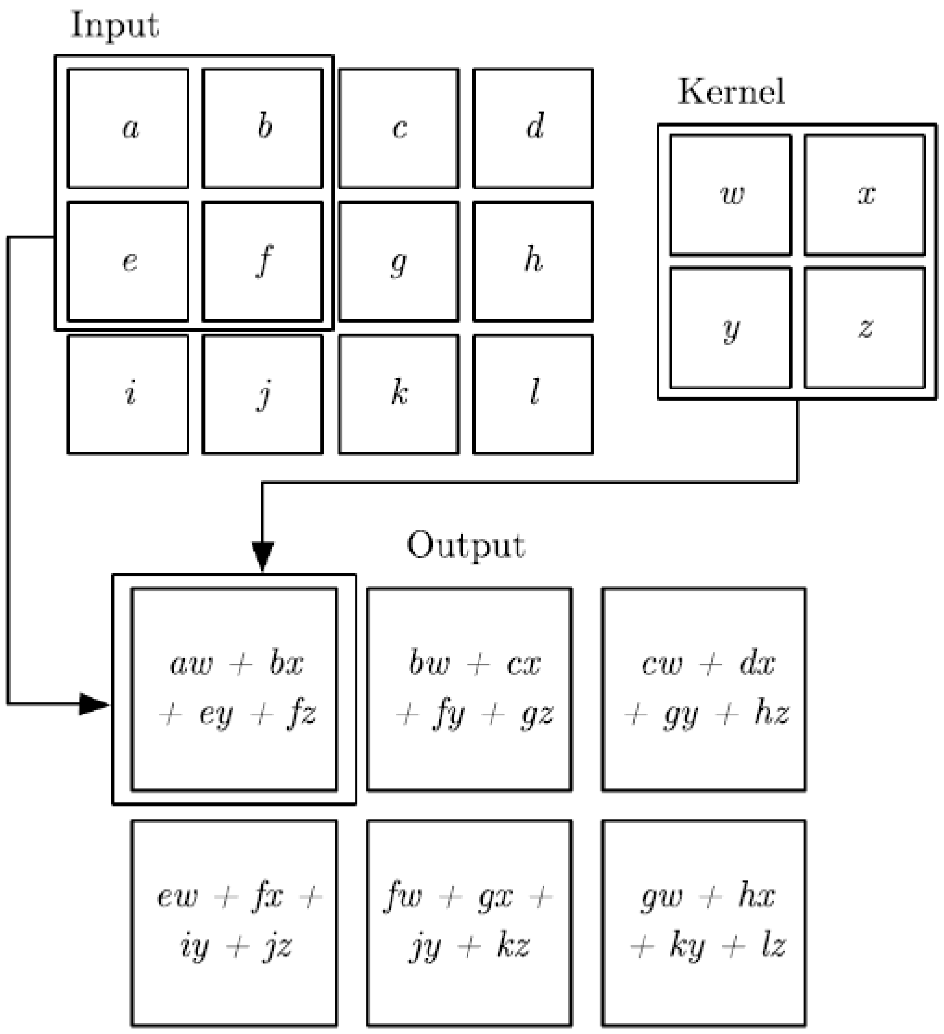


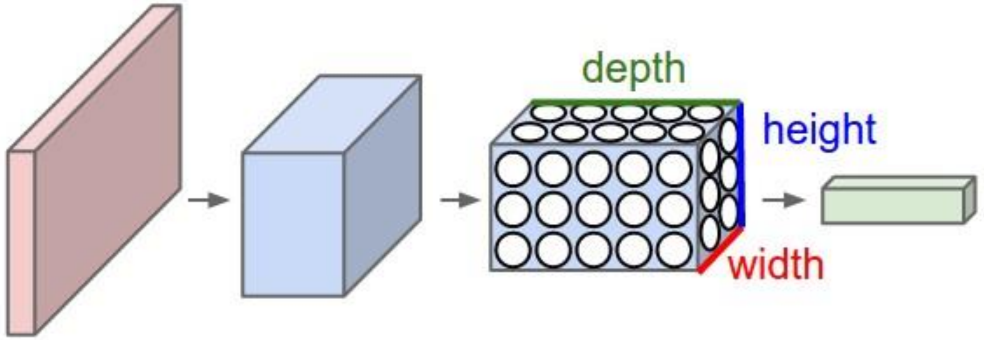
Figure: Goodfellow et al., “Deep Learning”, MIT Press, 2016.

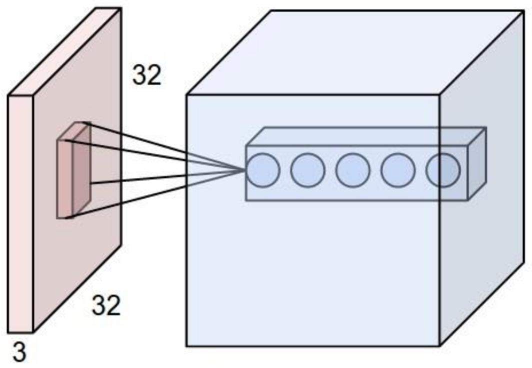
Local connectivity in CNN = Receptive fields

- Each neuron is connected to only a local neighborhood, i.e., receptive field
- The size of the receptive field ➔ another hyper-parameter.

Connectivity in CNN

- Local: The behavior of a neuron does not change other than being restricted to a subspace of the input.
- Each neuron is connected to slice of the previous layer
- A layer is actually a volume having a certain width x height and depth (or channel)
- A neuron is connected to a subspace of width x height but to all channels (depth)
- Example: CIFAR-10
  - Input: 32 x 32 x 3 (3 for RGB channels)
  - A neuron in the next layer with receptive field size 5x5 has input from a volume of 5x5x3.

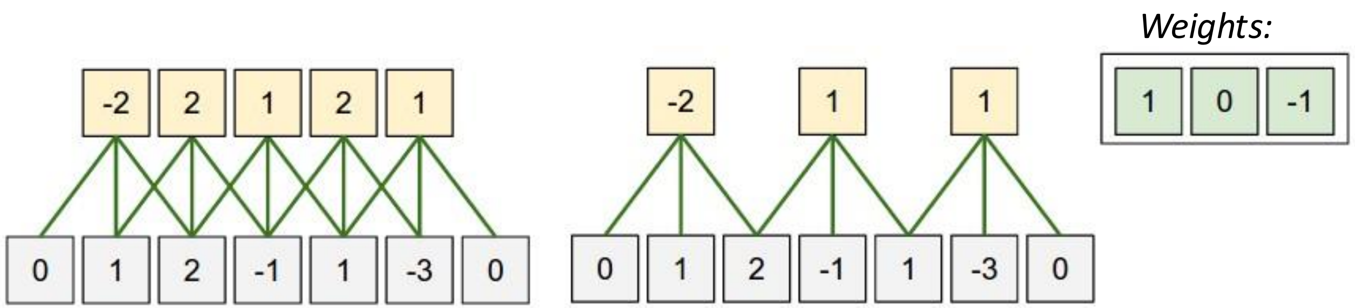




http://cs231n.github.io/convolutional-networks/

Important parameters

- Depth (number of channels)
  - We will have more neurons getting input from the same receptive field
  - This is similar to the hidden neurons with connections to the same input
  - These neurons learn to become selective to the presence of different signals in the same receptive field
- Stride
  - The amount of space between neighboring receptive fields
  - If it is small, RFs overlap more
  - It it is big, RFs overlap less
- How to handle the boundaries?
  1. Option 1: Don't process the boundaries. Only process pixels on which convolution window can be placed fully.
  2. Option 2: Zero-pad the input so that convolution can be performed at the boundary pixels.



http://cs231n.github.io/convolutional-networks/

Padding illustration

- Only convolution layers are shown.
- Top: no padding ➔ layers shrink in size.
- Bottom: zero padding ➔ layers keep their size fixed.

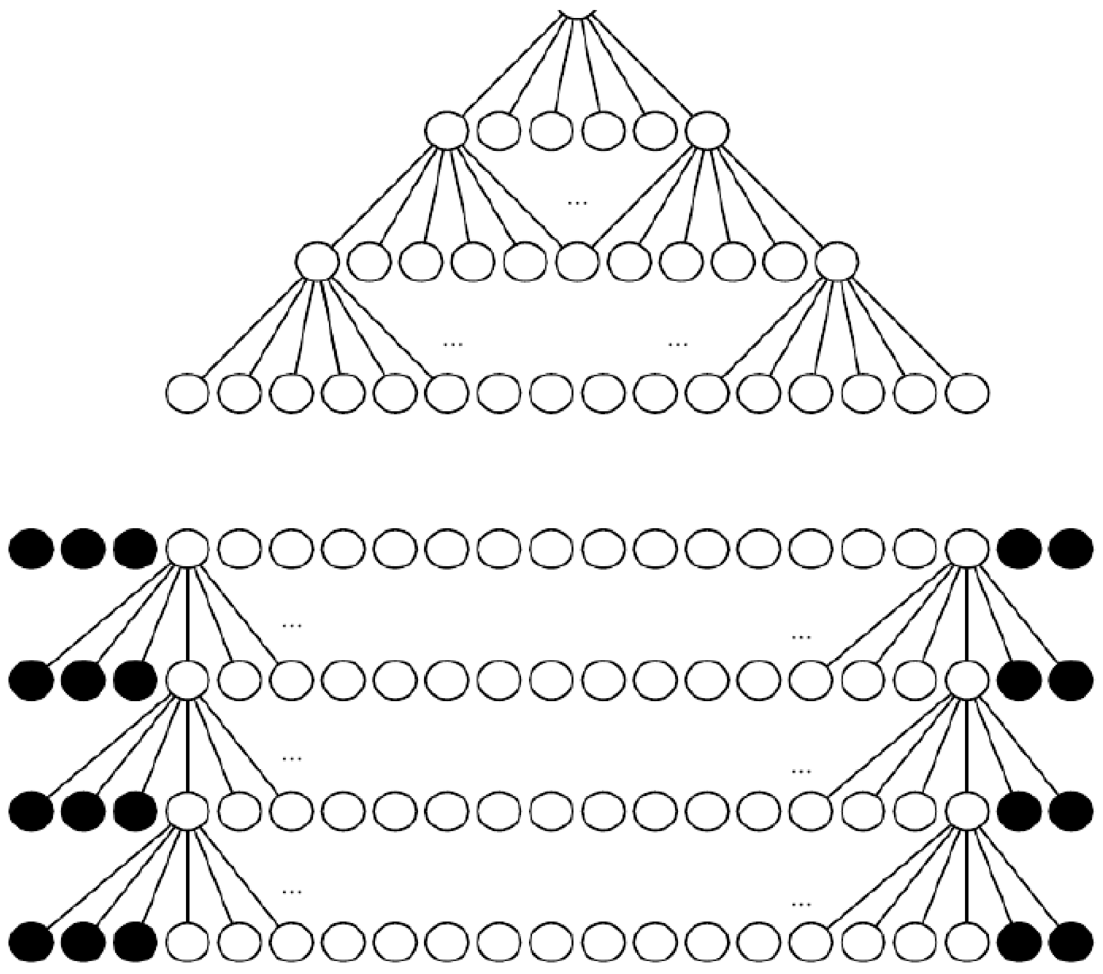


Figure 9.11: *The effect of zero padding on network size*: Consider a convolutional network with a kernel of width six at every layer. In this example, do not use any pooling, so only the convolution operation itself shrinks the network size. *Top)* In this convolutional network, we do not use any implicit zero padding. This causes the representation to shrink by five pixels at each layer. Starting from an input of sixteen pixels, we are only able to have three convolutional layers, and the last layer does not ever move the kernel, so arguably only two of the layers are truly convolutional. The rate of shrinking can be mitigated by using smaller kernels, but smaller kernels are less expressive and some shrinking is inevitable in this kind of architecture. *Bottom)* By adding five implicit zeroes to each layer, we prevent the representation from shrinking with depth. This allows us to make an arbitrarily deep convolutional network.

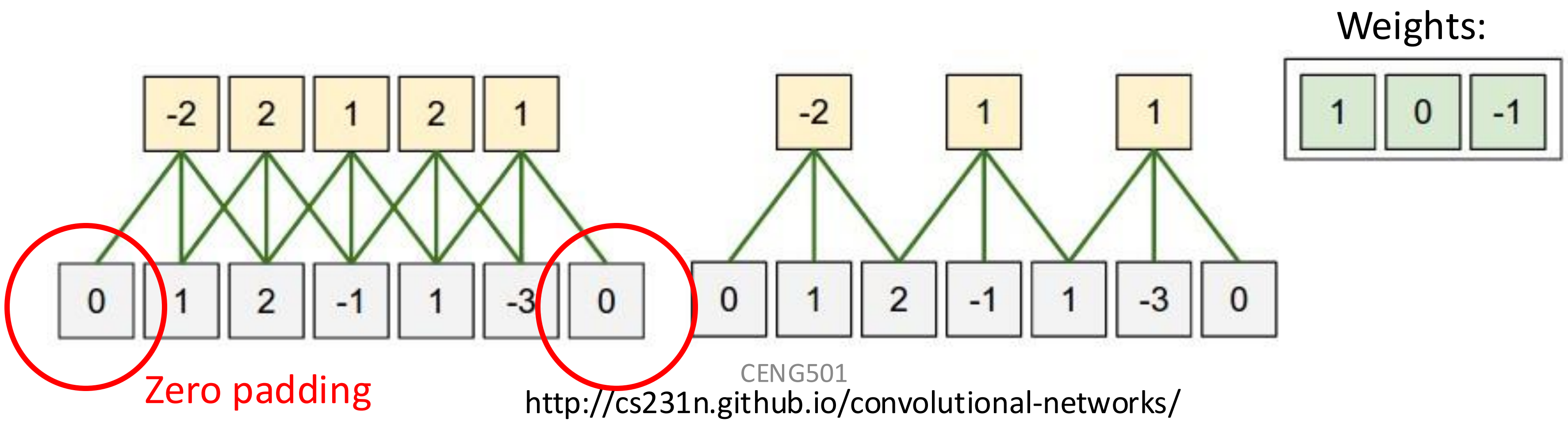
Figure: Goodfellow et al., “Deep Learning”, MIT Press, 2016.

Size of the next layer

- Along a dimension:
  - W: Size of the input
  - F: Size of the receptive field
  - S: Stride
  - P: Amount of zero-padding
- Then: the number of neurons as the output of a convolution layer:

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

- If this number is not an integer, your strides are incorrect and your neurons cannot tile nicely to cover the input volume



http://cs231n.github.io/convolutional-networks/

Size of the next layer

- Arranging these hyperparameters can be problematic
- Example:
- If W=10, P=0, and F=3, then

$$\frac{W - F + 2P}{S} + 1 = \frac{10 - 3 + 0}{S} + 1 = \frac{7}{S} + 1$$

i.e., S cannot be an integer other than 1 or 7.

- Zero-padding is your friend here.

Real example – AlexNet (Krizhevsky et al., 2012)

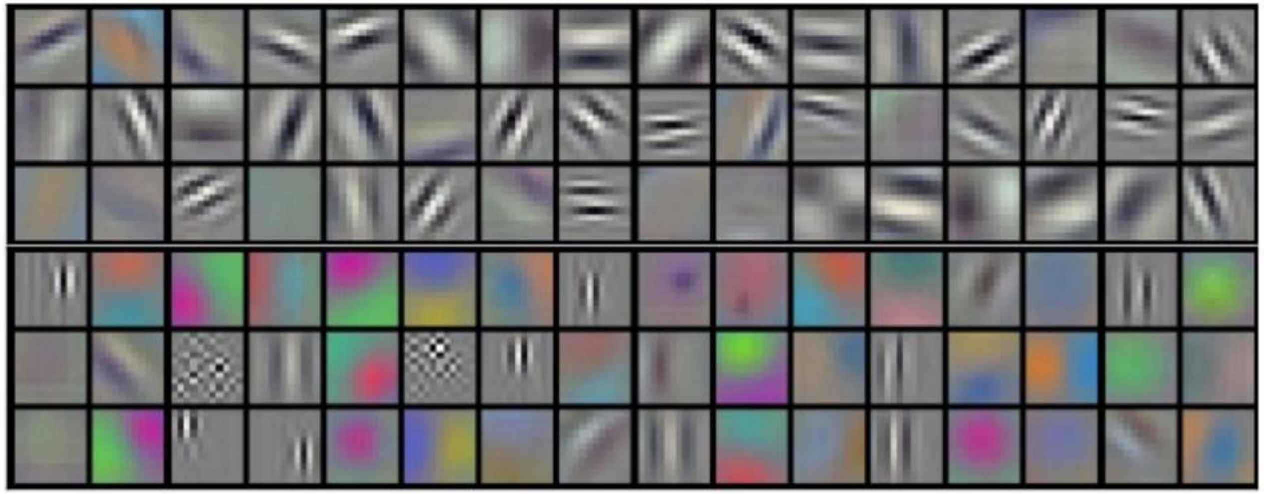
- Image size: 227×227×3
- W=227, F=11, S=4, P=0 ➔ $\frac{227-11}{S} + 1 = 55$

  (55 => the width of the convolution layer)
- Convolution layer: 55×55×96 neurons

  (96: the depth, the number of channels)
- Therefore, the first layer has 55×55×96 = 290,400 neurons
  - Each has 11×11×3 receptive field ➔ 363 weights and 1 bias
  - Then, 290,400×364 = 105,705,600 parameters just for the first convolution layer (if there were no weight sharing)
  - With weight sharing: 96 x 364 = 34,944

Real example –
AlexNet (Krizhevsky et al., 2012)

- However, we can share the parameters
  - For each channel (slice of depth), have the same set of weights
  - If 96 channels, this means 96 different set of weights
  - Then, 96×364 = 34,944 parameters
  - 364 weights shared by 55×55 neurons in each channel



Example filters learned by Krizhevsky et al. Each of the 96 filters shown here is of size [11x11x3], and each one is shared by the 55*55 neurons in one depth slice. Notice that the parameter sharing assumption is relatively reasonable: If detecting a horizontal edge is important at some location in the image, it should intuitively be useful at some other location as well due to the translationally-invariant structure of images. There is therefore no need to relearn to detect a horizontal edge at every one of the 55*55 distinct locations in the Conv layer output volume.

http://cs231n.github.io/convolutional-networks/

More on connectivity

Small RF & Stacking

- E.g., 3 CONV layers of 3x3 RFs
- Pros:
  - Same extent for these example figures
  - With non-linearity added on 2nd and 3rd layers ➔ More expressive! More representational capacity!
  - Less parameters:

    3 layers x [(3 x 3 x C) x C] = 27CxC
- Cons?

Large RF & Single Layer

- 7x7 RFs of single CONV layer
- Pros?
- Cons:
  - One layer => Linear capacity
  - More parameters:

    (7x7xC)xC = 49CxC

So, we prefer a stack of small filter sizes against big ones

http://cs231n.github.io/convolutional-networks/

Implementation Details: NumPy example

- Suppose input is X of shape (11,11,4)
- Depth slice at depth d (i.e., channel d): `X[:,:,d]`
- Depth column at position (x,y): `X[x,y,:]`
- F: 5, P:0 (no padding), S=2
  - Output volume (V) width, height = (11-5+0)/2+1 = 4
- Example computation for some neurons in first channel:

```
V[0,0,0] = np.sum(X[:5,:5,:] * W0) + b0
V[1,0,0] = np.sum(X[2:7,:5,:] * W0) + b0
V[2,0,0] = np.sum(X[4:9,:5,:] * W0) + b0
V[3,0,0] = np.sum(X[6:11,:5,:] * W0) + b0
```

- Note that this is just along one dimension (x)

http://cs231n.github.io/convolutional-networks/

Implementation Details: NumPy example

- A second activation map (channel):

```
V[0,0,1] = np.sum(X[:5,:5,:] * W1) + b1
V[1,0,1] = np.sum(X[2:7,:5,:] * W1) + b1
V[2,0,1] = np.sum(X[4:9,:5,:] * W1) + b1
V[3,0,1] = np.sum(X[6:11,:5,:] * W1) + b1
V[0,1,1] = np.sum(X[:5,2:7,:] * W1) + b1
V[2,3,1] = np.sum(X[4:9,6:11,:] * W1) + b1
```

(example of going along y) — for `V[0,1,1]`

(or along both) — for `V[2,3,1]`

http://cs231n.github.io/convolutional-networks/

Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

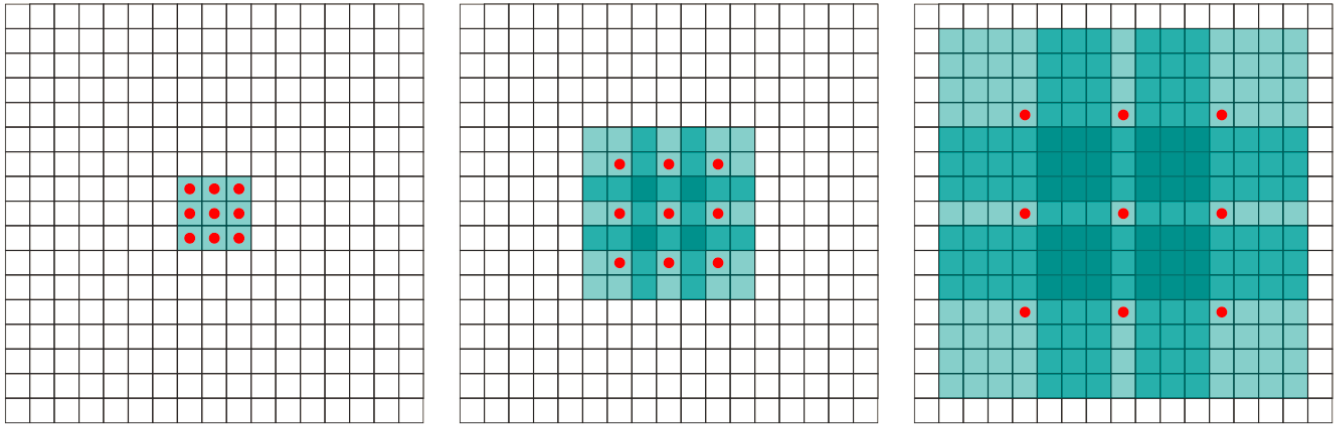
http://cs231n.github.io/convolutional-networks/

Types of Convolution: Unshared convolution

- In some cases, sharing the weights does not make sense
  - When?
- Different parts of the input might require different types of processing/features
- In such a case, we just have a network with local connectivity
- E.g., a face.
  - Features are not repeated across the space.

Types of Convolution: Dilated (Atrous) Convolution

Purpose: Increase effective receptive field size without increasing parameters.



(a) (b) (c)

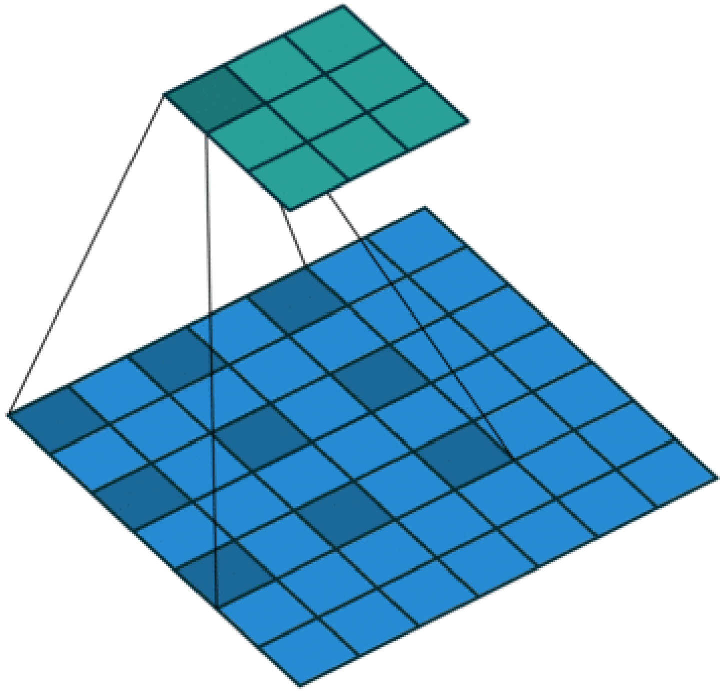
Figure 1: Systematic dilation supports exponential expansion of the receptive field without loss of resolution or coverage. (a) F_1 is produced from F_0 by a 1-dilated convolution; each element in F_1 has a receptive field of 3×3. (b) F_2 is produced from F_1 by a 2-dilated convolution; each element in F_2 has a receptive field of 7×7. (c) F_3 is produced from F_2 by a 4-dilated convolution; each element in F_3 has a receptive field of 15×15. The number of parameters associated with each layer is identical. The receptive field grows exponentially while the number of parameters grows linearly.

Published as a conference paper at ICLR 2016

MULTI-SCALE CONTEXT AGGREGATION BY DILATED CONVOLUTIONS

Fisher Yu
Princeton University

Vladlen Koltun
Intel Labs



https://github.com/vdumoulin/conv_arithmetic

Types of Convolution: Transposed Convolution

Purpose: Increasing layer width+height (upsampling).

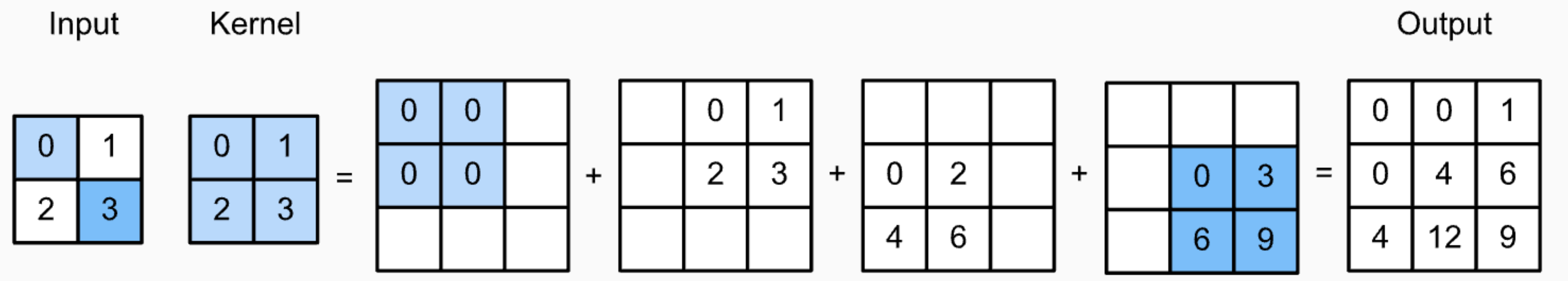


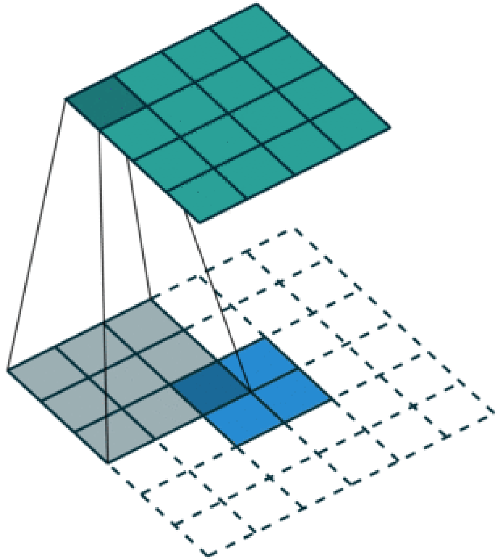
Fig. 13.10.1 Transposed convolution layer with a 2×2 kernel.

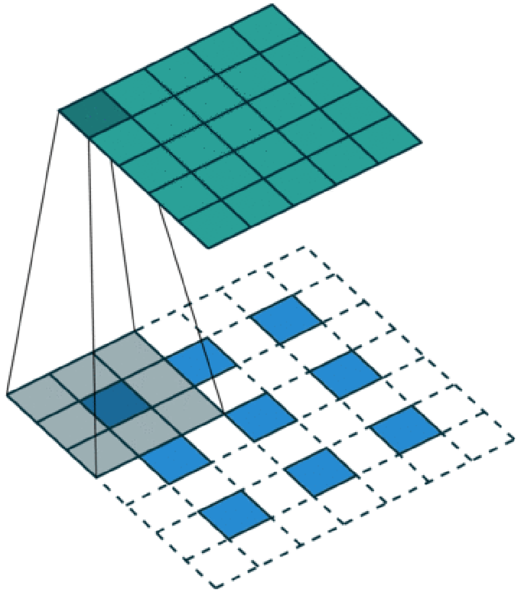
Figure: https://d2l.ai/chapter_computer-vision/transposed-conv.html

The size of the output:

- Regular convolution: $O = \frac{W-F+2\times P}{S} + 1$
- Transpose convolution: $W = (O-1) \times S + F - 2 \times P$

Types of Convolution: Upsampling with Padding or Dilation

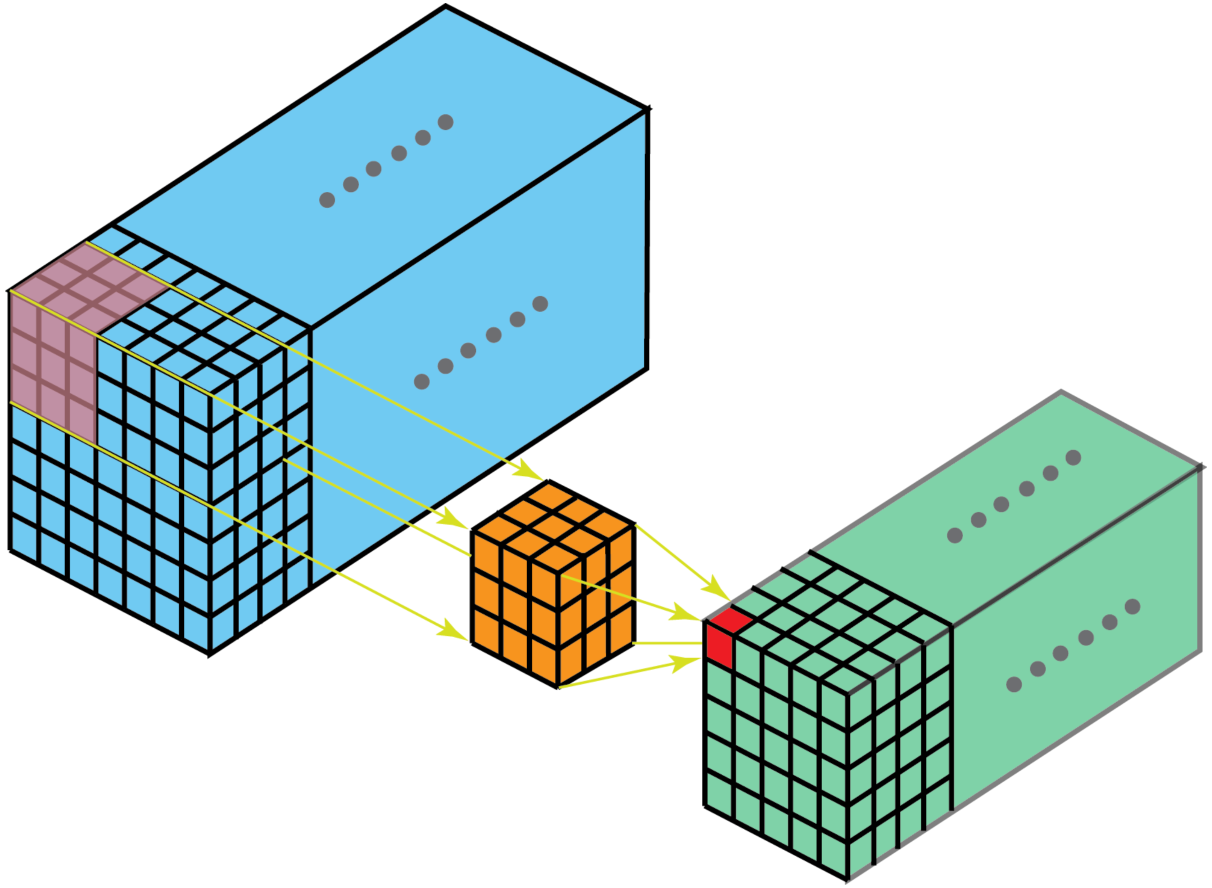




https://github.com/vdumoulin/conv_arithmetic

Types of Convolution: 3D Convolution

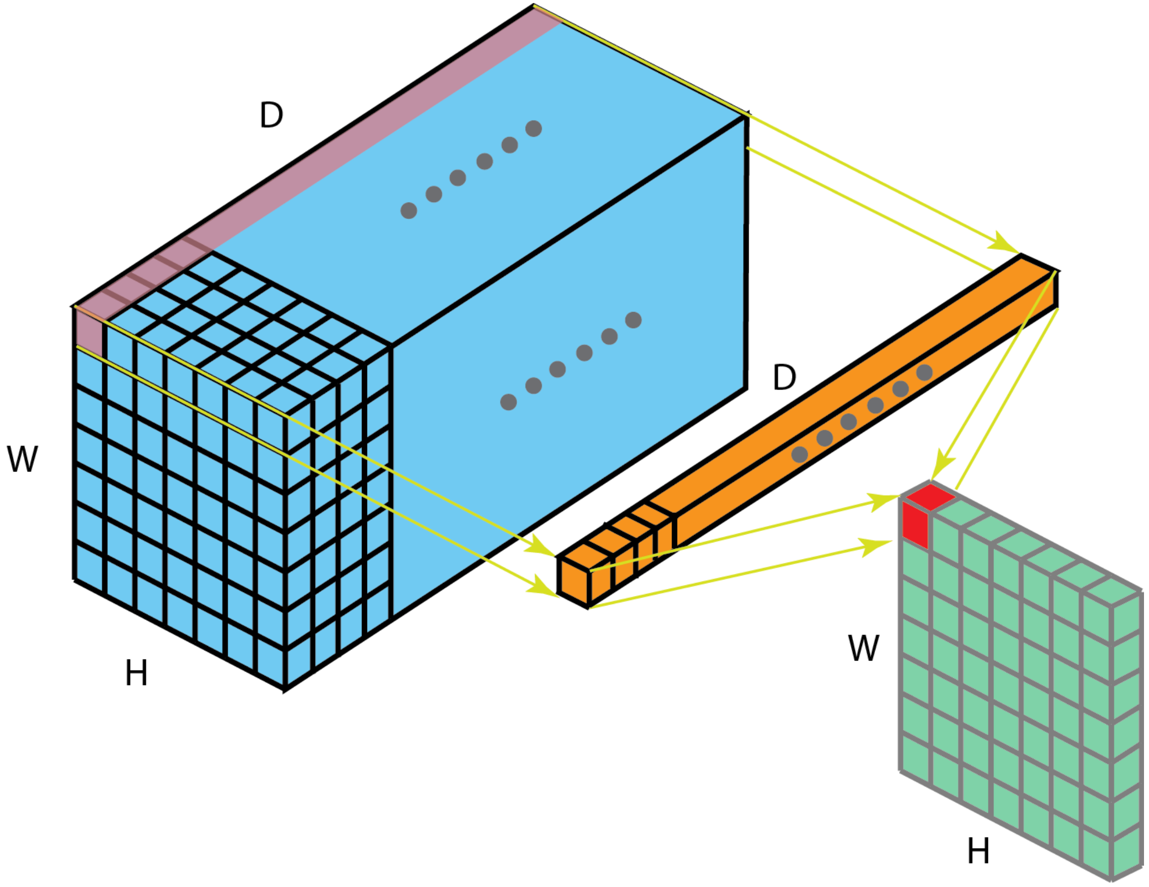
Purpose: Work with 3D data, e.g. learn spatial + temporal representations for videos.



https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215

Types of Convolution: 1x1 Convolution

Purpose: Reduce number of channels.

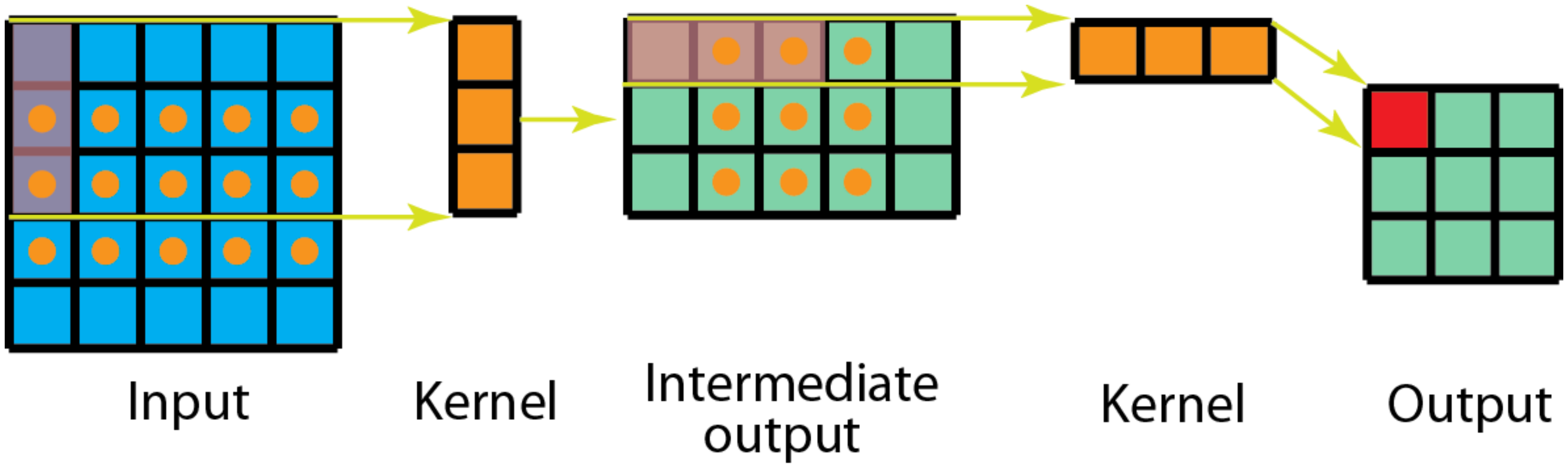


https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215

Types of Convolution: Separable Convolution

Purpose: Reduce number of parameters and multiplications.

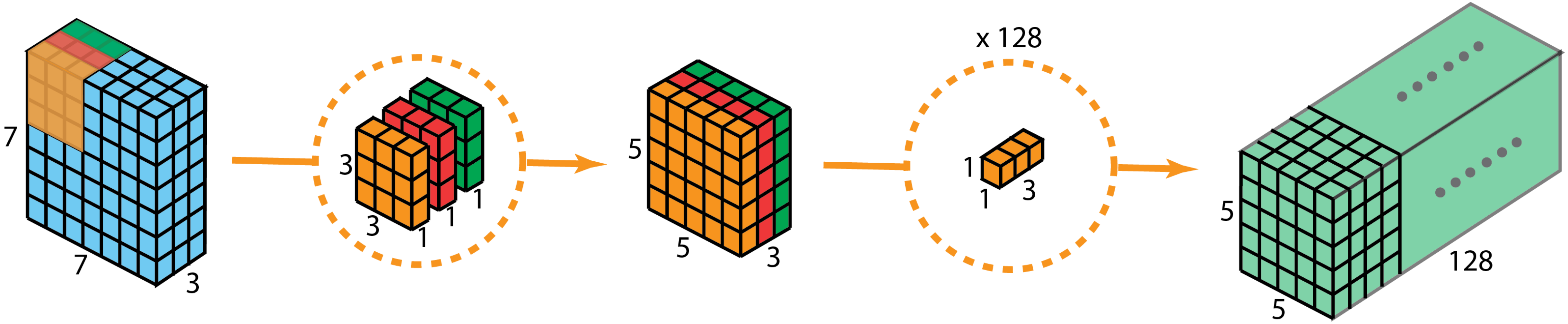
$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \times \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$$



https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215

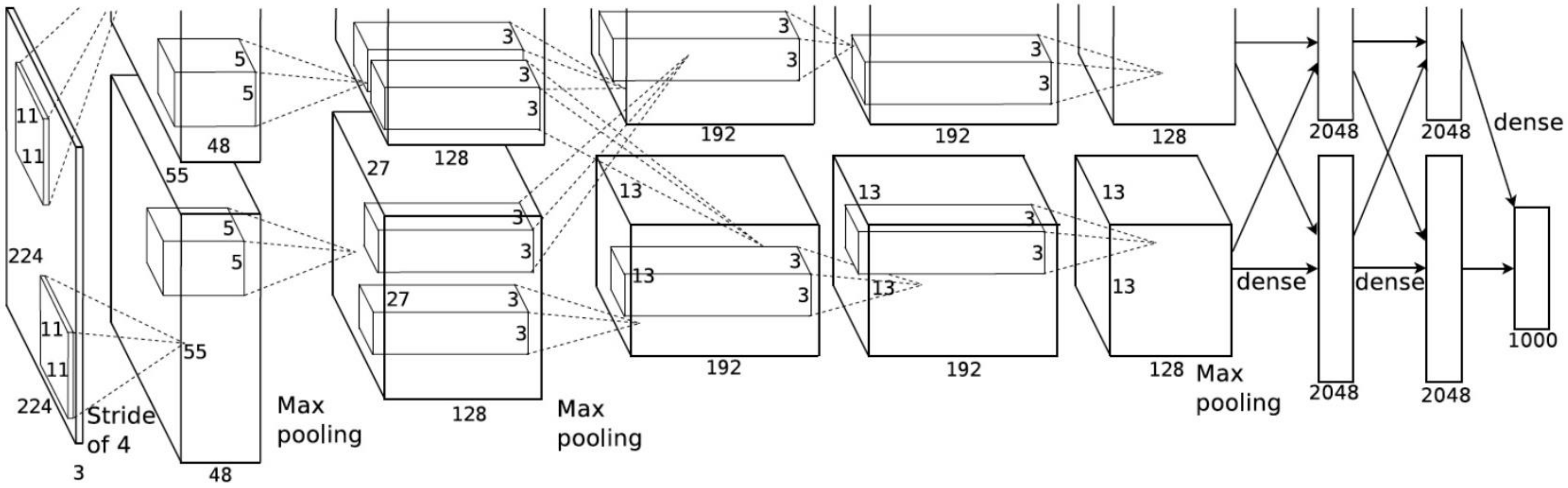
Types of Convolution: Depth-wise Separable Convolution

Purpose: Reduce number of parameters and multiplications.

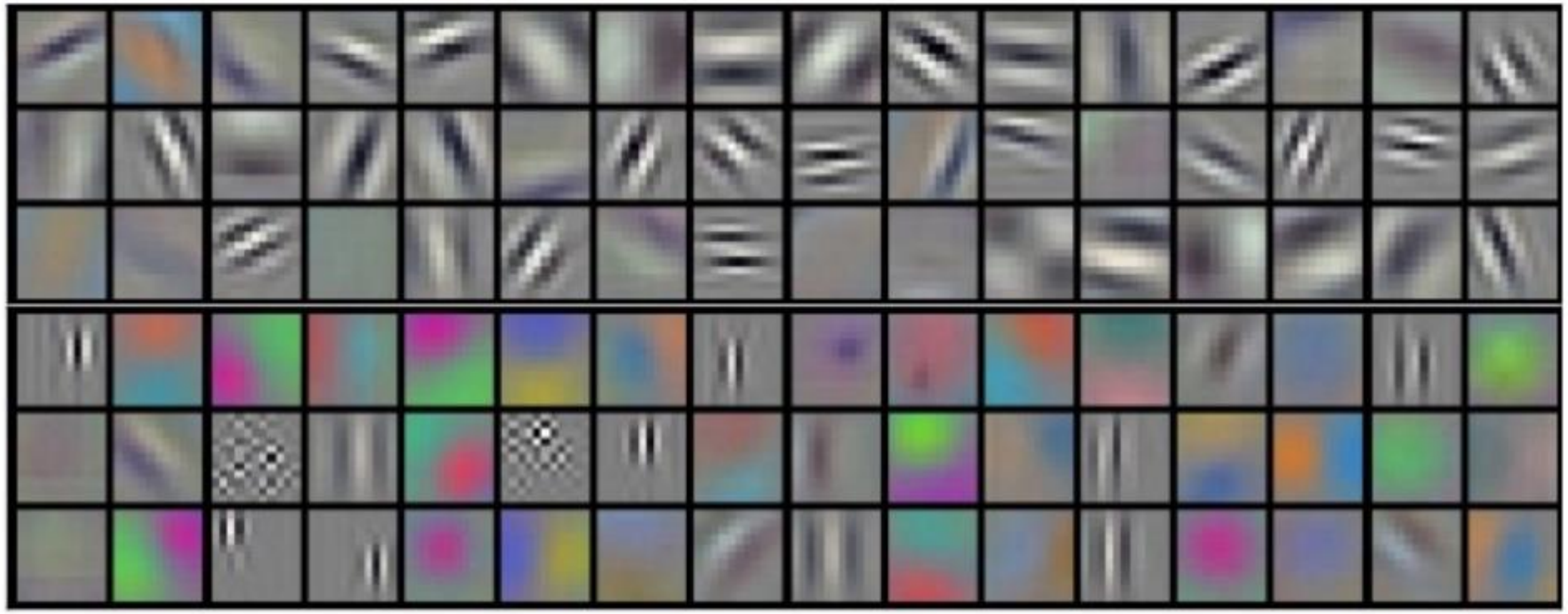


https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215

Types of Convolution: Group Convolution

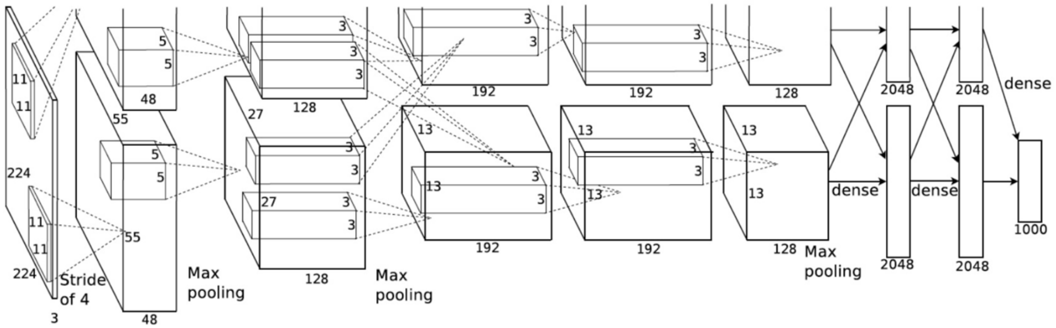


AlexNet (Krizhevsky et al.)

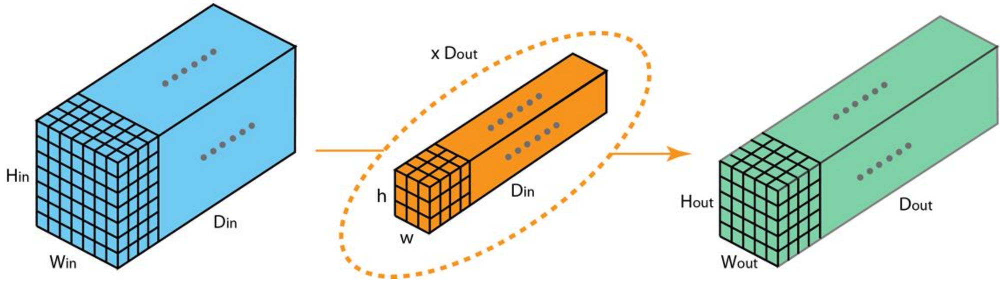


Types of Convolution: Group Convolution

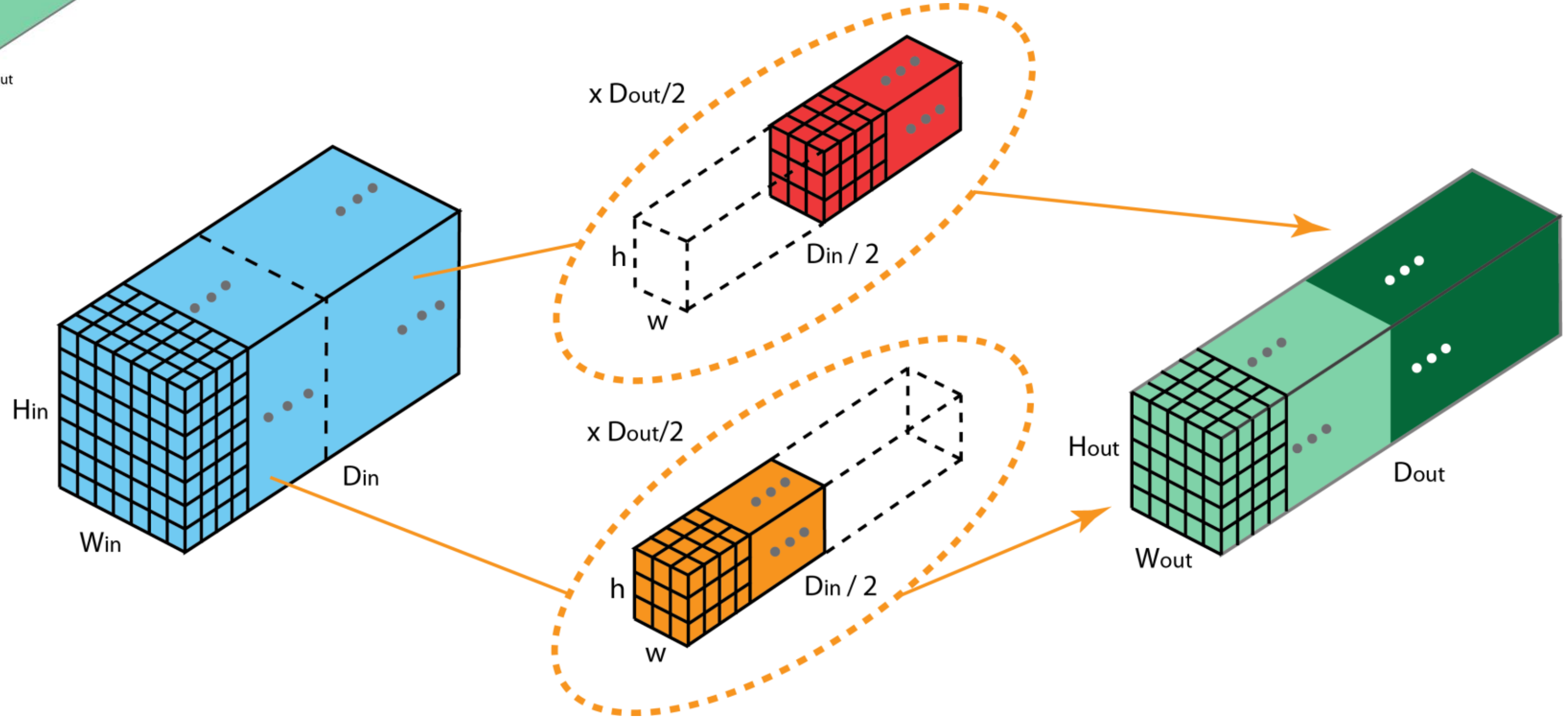
Purpose: Reduce number of parameters and multiplications.



AlexNet



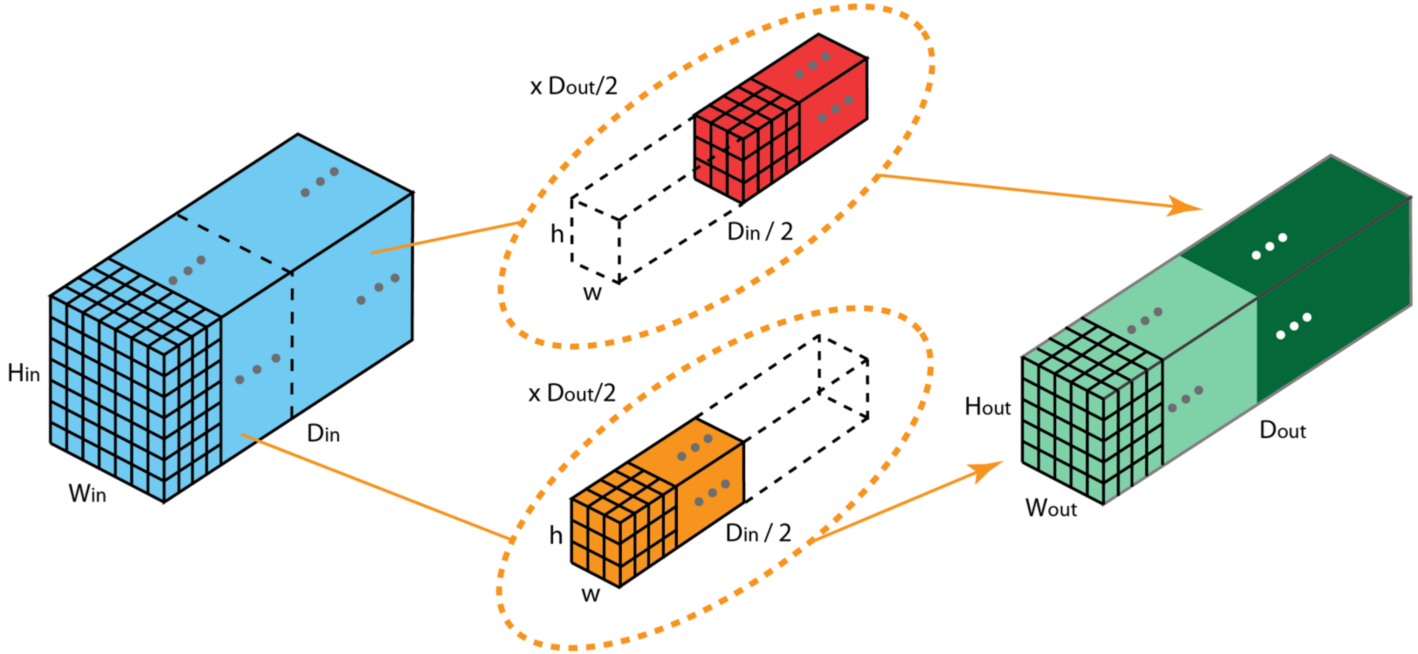
Normal Convolution



https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215

Types of Convolution: Group Convolution

- Benefits:
  - Efficiency in training (distribute groups to different GPUs)
  - Decrease in # of parameters as the # of groups increases
  - Better performance?



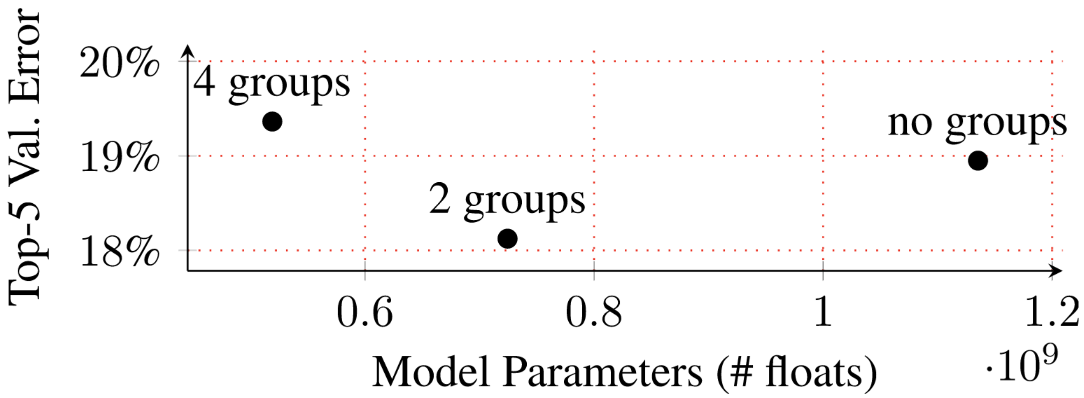


Figure: https://blog.yani.ai/filter-group-tutorial/

https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215

Types of Convolution: Deformable Convolution

Dai, J., Qi, H., Xiong, Y., Li, Y., Zhang, G., Hu, H., & Wei, Y. (2017). Deformable convolutional networks. ICCV.

Purpose: Flexible receptive field.

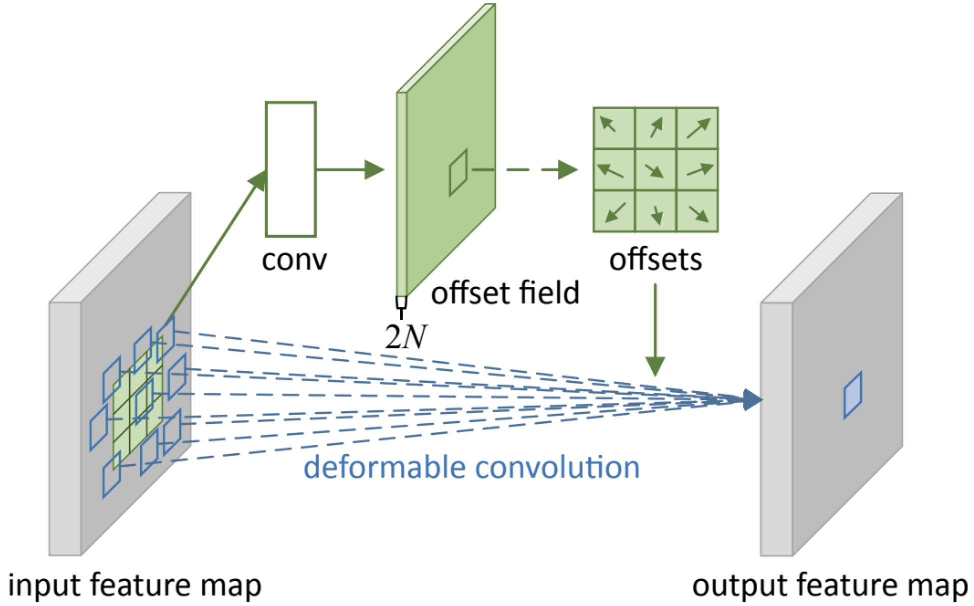


Figure 2: Illustration of 3×3 deformable convolution.

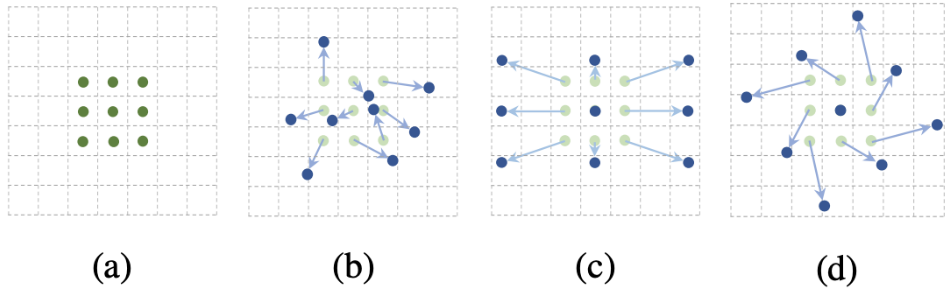
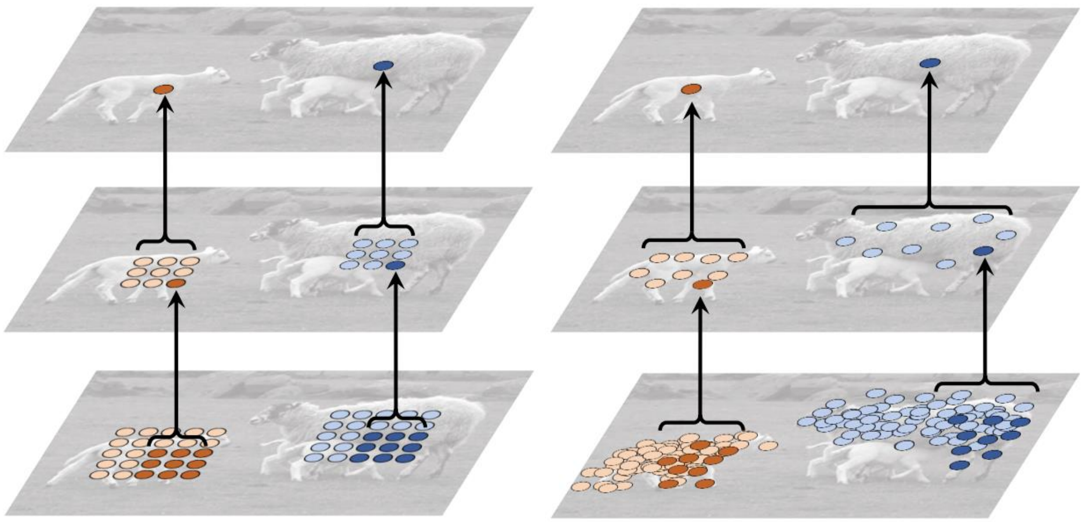


Figure 1: Illustration of the sampling locations in 3×3 standard and deformable convolutions. (a) regular sampling grid (green points) of standard convolution. (b) deformed sampling locations (dark blue points) with augmented offsets (light blue arrows) in deformable convolution. (c)(d) are special cases of (b), showing that the deformable convolution generalizes various transformations for scale, (anisotropic) aspect ratio and rotation.



(a) standard convolution (b) deformable convolution

Types of Convolution: Deformable Convolution

Dai, J., Qi, H., Xiong, Y., Li, Y., Zhang, G., Hu, H., & Wei, Y. (2017). Deformable convolutional networks. ICCV.

Bilinear interpolation for x(p).

$$\mathcal{R} = \{(-1,-1), (-1,0), \dots, (0,1), (1,1)\}$$

defines a 3×3 kernel with dilation 1.

For each location $\mathbf{p}_0$ on the output feature map $\mathbf{y}$, we have

$$\mathbf{y}(\mathbf{p}_0) = \sum_{\mathbf{p}_n \in \mathcal{R}} \mathbf{w}(\mathbf{p}_n) \cdot \mathbf{x}(\mathbf{p}_0 + \mathbf{p}_n), \tag{1}$$

where $\mathbf{p}_n$ enumerates the locations in $\mathcal{R}$.

In deformable convolution, the regular grid $\mathcal{R}$ is augmented with offsets $\{\Delta\mathbf{p}_n | n = 1, ..., N\}$, where $N = |\mathcal{R}|$. Eq. (1) becomes

$$\mathbf{y}(\mathbf{p}_0) = \sum_{\mathbf{p}_n \in \mathcal{R}} \mathbf{w}(\mathbf{p}_n) \cdot \mathbf{x}(\mathbf{p}_0 + \mathbf{p}_n + \Delta\mathbf{p}_n). \tag{2}$$

Now, the sampling is on the irregular and offset locations $\mathbf{p}_n + \Delta\mathbf{p}_n$. As the offset $\Delta\mathbf{p}_n$ is typically fractional, Eq. (2) is implemented via bilinear interpolation as

$$\mathbf{x}(\mathbf{p}) = \sum_{\mathbf{q}} G(\mathbf{q}, \mathbf{p}) \cdot \mathbf{x}(\mathbf{q}), \tag{3}$$

where $\mathbf{p}$ denotes an arbitrary (fractional) location ($\mathbf{p} = \mathbf{p}_0 + \mathbf{p}_n + \Delta\mathbf{p}_n$ for Eq. (2)), $\mathbf{q}$ enumerates all integral spatial locations in the feature map $\mathbf{x}$, and $G(\cdot,\cdot)$ is the bilinear interpolation kernel. Note that G is two dimensional. It is separated into two one dimensional kernels as

$$G(\mathbf{q}, \mathbf{p}) = g(q_x, p_x) \cdot g(q_y, p_y), \tag{4}$$

where $g(a, b) = max(0, 1 - |a - b|)$. Eq. (3) is fast to compute as $G(\mathbf{q}, \mathbf{p})$ is non-zero only for a few $\mathbf{q}$s.

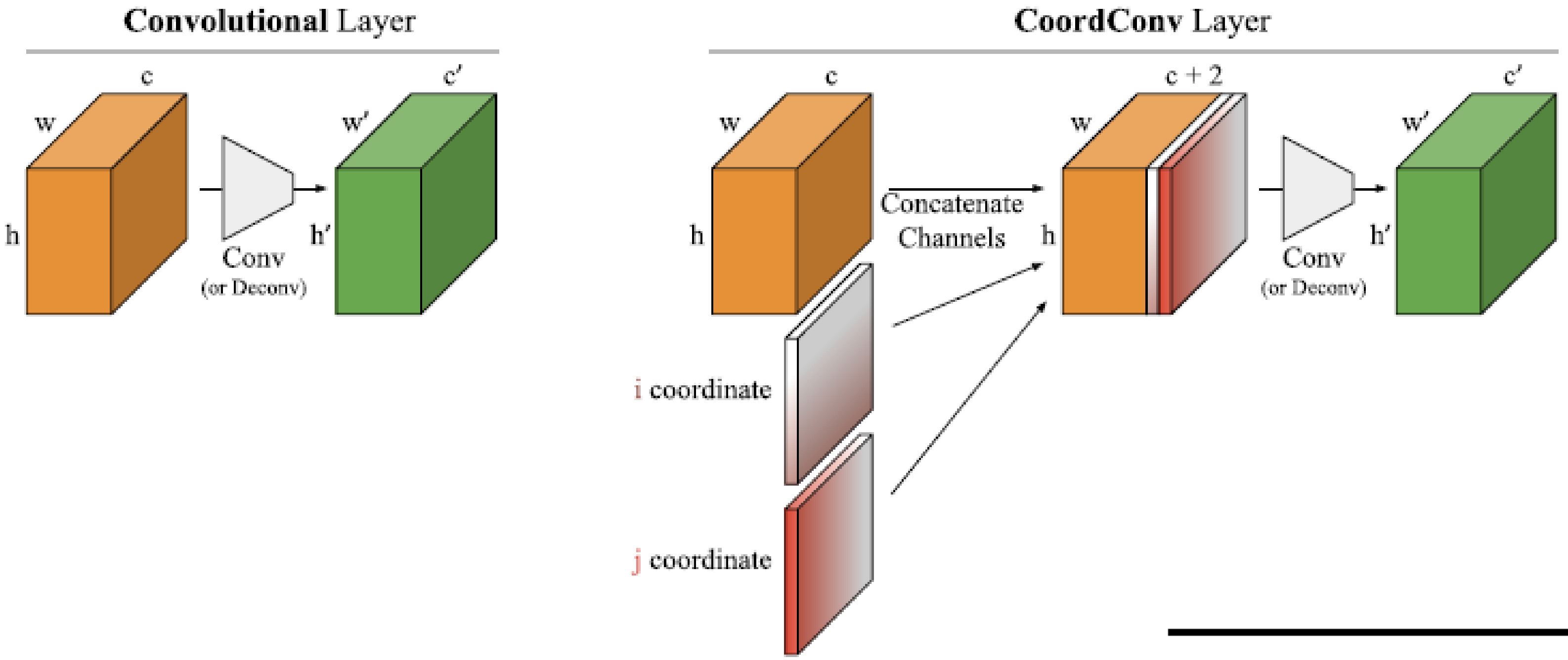
In the deformable convolution Eq. (2), the gradient w.r.t. the offset $\Delta\mathbf{p}_n$ is computed as

$$\begin{aligned}\frac{\partial \mathbf{y}(\mathbf{p}_0)}{\partial \Delta\mathbf{p}_n} &= \sum_{\mathbf{p}_n \in \mathcal{R}} \mathbf{w}(\mathbf{p}_n) \cdot \frac{\partial \mathbf{x}(\mathbf{p}_0 + \mathbf{p}_n + \Delta\mathbf{p}_n)}{\partial \Delta\mathbf{p}_n} \\ &= \sum_{\mathbf{p}_n \in \mathcal{R}} \left[\mathbf{w}(\mathbf{p}_n) \cdot \sum_{\mathbf{q}} \frac{\partial G(\mathbf{q}, \mathbf{p}_0 + \mathbf{p}_n + \Delta\mathbf{p}_n)}{\partial \Delta\mathbf{p}_n} \mathbf{x}(\mathbf{q}) \right],\end{aligned} \tag{7}$$

where the term $\frac{\partial G(\mathbf{q}, \mathbf{p}_0 + \mathbf{p}_n + \Delta\mathbf{p}_n)}{\partial \Delta\mathbf{p}_n}$ can be derived from Eq. (4). Note that the offset $\Delta\mathbf{p}_n$ is 2D and we use $\partial\Delta\mathbf{p}_n$ to denote $\partial\Delta p_n^x$ and $\partial\Delta p_n^y$ for simplicity.

Types of Convolution: Position-sensitive convolution

- Learn to use position information when necessary



An intriguing failing of convolutional neural networks and the CoordConv solution 2018

Rosanne Liu[1] rosanne@uber.com
Joel Lehman[1] joel.lehman@uber.com
Piero Molino[1] piero@uber.com
Felipe Petroski Such[1] felipe.such@uber.com
Eric Frank[1] mysterefrank@uber.com
Alex Sergeev[2] asergeev@uber.com
Jason Yosinski[1] yosinski@uber.com

Convolution demos & tutorials

- https://github.com/vdumoulin/conv_arithmetic
- http://cs231n.github.io/assets/conv-demo/index.html
- https://ezyang.github.io/convolution-visualizer/index.html
- https://ikhlestov.github.io/pages/machine-learning/convolutions-types/
- https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215

OPERATIONS IN A CNN: Pooling

Pooling

Remember the motivation for CNNs:
S (simple) cells: local feature extraction.
C (complex) cells: provide tolerance to deformation, e.g. shift.

- Apply an operation on the “detector” results to combine or to summarize the answers of a set of units.
  - Applied to **each channel (depth slice) independently**
  - The operation has to be differentiable of course.
- Alternatives:
  - Maximum
  - Sum
  - Average
  - Weighted average with distance from the value of the center pixel
  - L2 norm
  - Second-order statistics?
  - …
- Different problems may perform better with different pooling methods
- Pooling can be overlapping or non-overlapping

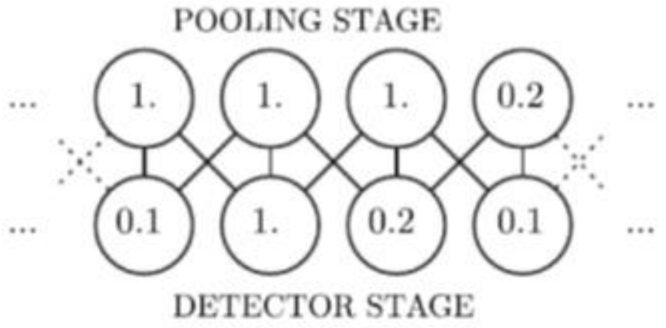


Figure: Goodfellow et al., “Deep Learning”, MIT Press, 2016.

http://cs231n.github.io/convolutional-networks/

Pooling

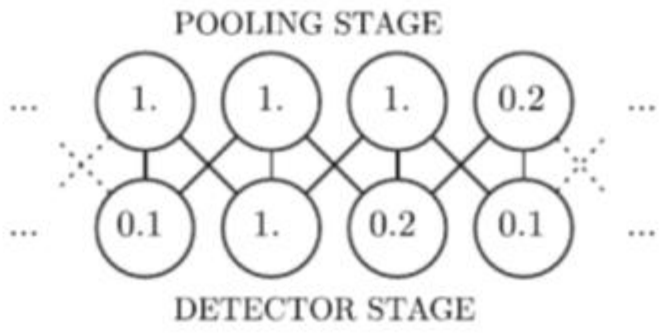
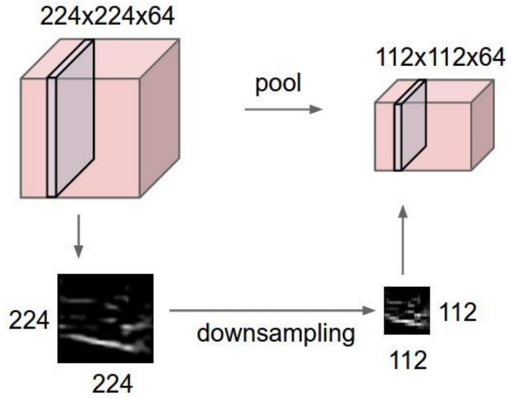
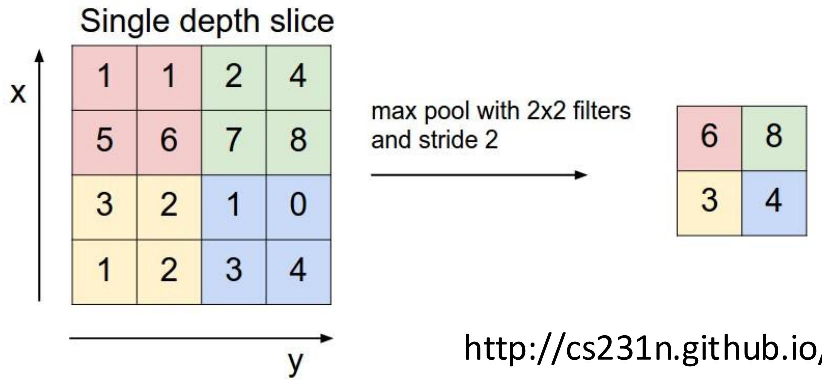


Figure: Goodfellow et al., “Deep Learning”, MIT Press, 2016.

- Example
  - Pooling layer with filters of size 2x2
  - With stride = 2
  - Discards 75% of the activations
  - Depth dimension remains unchanged
- Max pooling with F=3, S=2 or F=2, S=2 are quite common.
  - Pooling with bigger receptive field sizes can be destructive
- Avg pooling is an obsolete choice. Max pooling is shown to work better in practice.

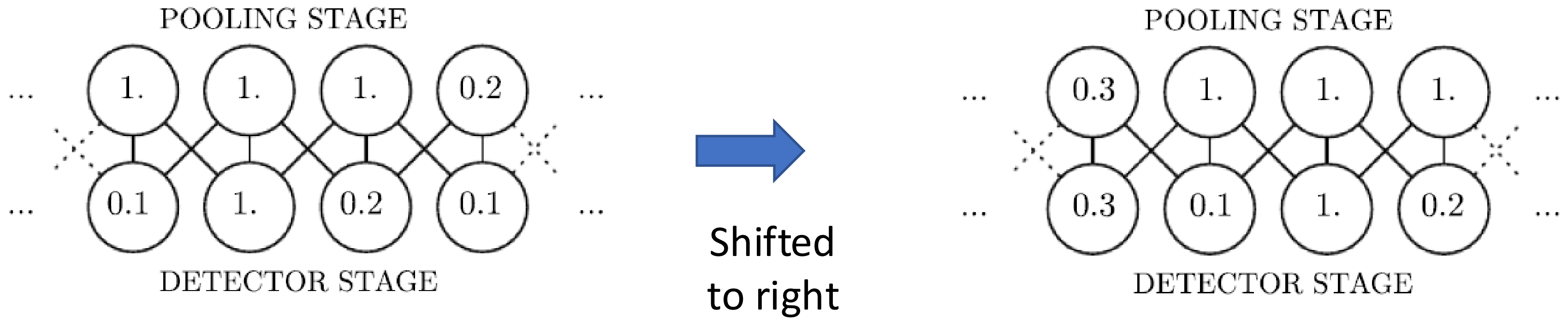




http://cs231n.github.io/convolutional-networks/

Pooling

- Pooling provides invariance to small translation.



Figures: Goodfellow et al., “Deep Learning”, MIT Press, 2016.

- If you pool over different convolution operators, you can gain invariance to different transformations.

Pooling can downsample

- Especially needed when to produce an output with fixed-length on varying length input.

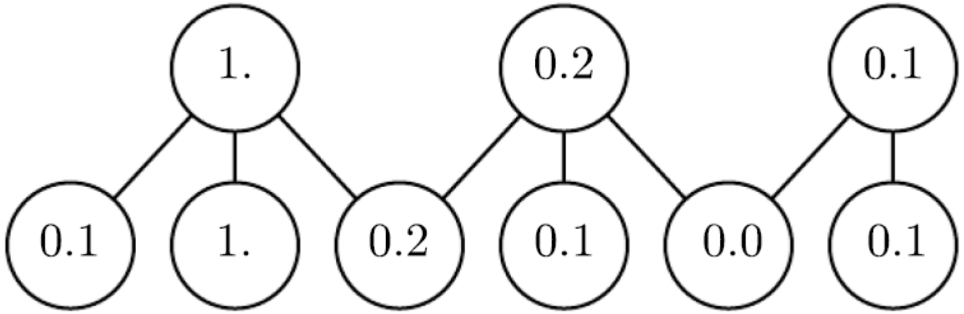


Figure: Goodfellow et al., “Deep Learning”, MIT Press, 2016.

- If you want to use the network on images of varying size, you can arrange this with pooling (with the help of convolutional layers)

CNNs without pooling

- *"Striving for Simplicity: The All Convolutional Net* proposes to *discard the pooling layer* *in favor of architecture that only consists of repeated CONV layers*. *To reduce the size of the representation they suggest using larger stride in CONV layer once in a while."*

CIFAR-10 classification error

Model	Error (%)	# parameters
without data augmentation		
Model A	12.47%	≈ 0.9 M
Strided-CNN-A	13.46%	≈ 0.9 M
ConvPool-CNN-A	10.21%	≈ 1.28 M
ALL-CNN-A	10.30%	≈ 1.28 M
Model B	10.20%	≈ 1 M
Strided-CNN-B	10.98%	≈ 1 M
ConvPool-CNN-B	9.33%	≈ 1.35 M
ALL-CNN-B	9.10%	≈ 1.35 M
Model C	9.74%	≈ 1.3 M
Strided-CNN-C	10.19%	≈ 1.3 M
ConvPool-CNN-C	9.31%	≈ 1.4 M
ALL-CNN-C	9.08%	≈ 1.4 M

(ALL-CNN: No pooling)

https://arxiv.org/pdf/1412.6806.pdf

http://cs231n.github.io/convolutional-networks/

Summary: Convolution & pooling

- Provide strong bias on the model and the solution
- They directly affect the overall performance of the system

OPERATIONS IN A CNN: nonlinearity

Non-linearity

- Sigmoid
- Tanh
- ReLU and its variants
  - The common choice
  - Faster
  - Easier (in backpropagation etc.)
  - Avoids saturation issues
- …

OPERATIONS IN A CNN: normalization

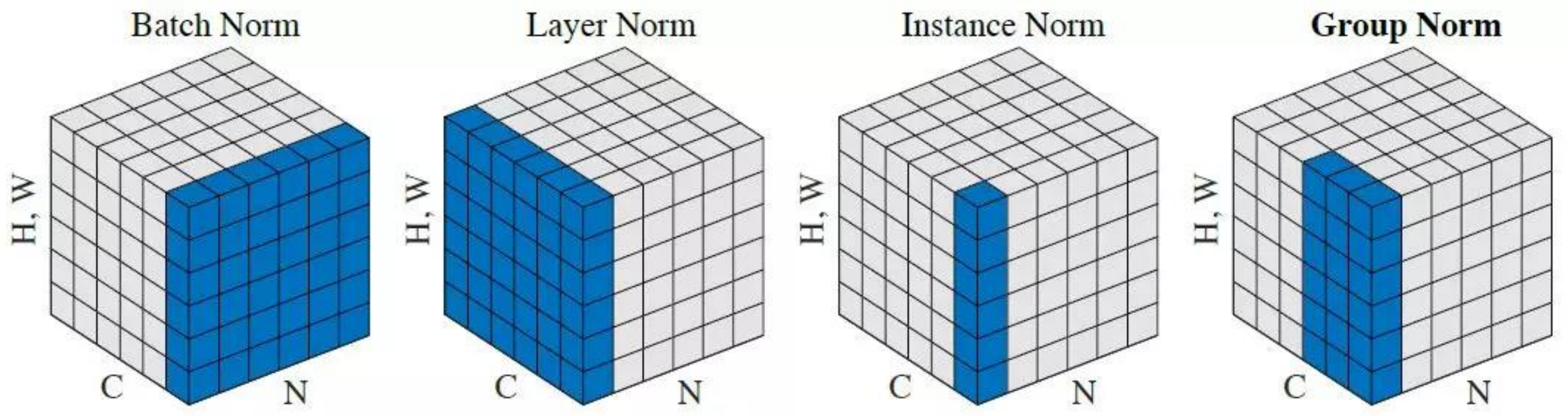
- From Krizhevsky et al. (2012):

generalization. Denoting by $a^i_{x,y}$ the activity of a neuron computed by applying kernel i at position (x, y) and then applying the ReLU nonlinearity, the response-normalized activity $b^i_{x,y}$ is given by the expression

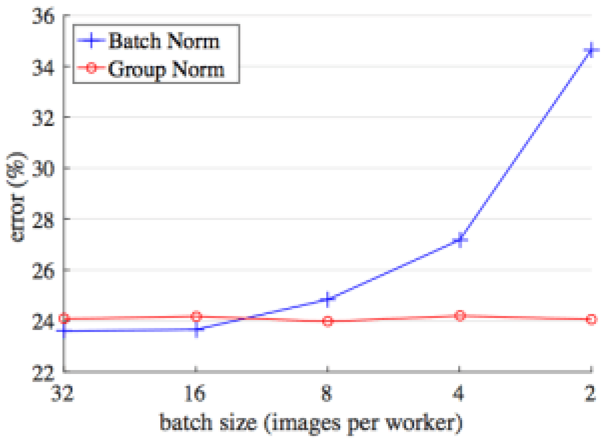
$$b^i_{x,y} = a^i_{x,y} \bigg/ \left(k + \alpha \sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)} (a^j_{x,y})^2 \right)^{\beta}$$

where the sum runs over n "adjacent" kernel maps at the same spatial position, and N is the total number of kernels in the layer. The ordering of the kernel maps is of course arbitrary and determined before training begins. This sort of response normalization implements a form of lateral inhibition inspired by the type found in real neurons, creating competition for big activities amongst neuron outputs computed using different kernels. The constants k, n, α, and β are hyper-parameters whose values are determined using a validation set; we used $k = 2$, $n = 5$, $\alpha = 10^{-4}$, and $\beta = 0.75$. We

Normalization



For each channel independently.

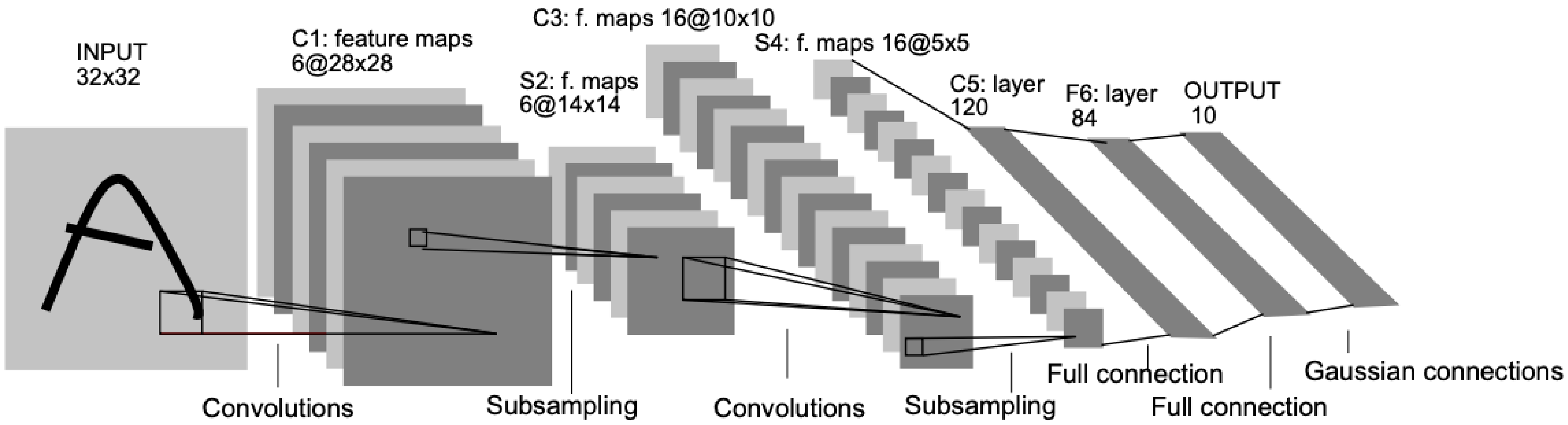


https://medium.com/syncedreview/facebook-ai-proposes-group-normalization-alternative-to-batch-normalization-fb0699bffae7

OPERATIONS IN A CNN: fully connected layer

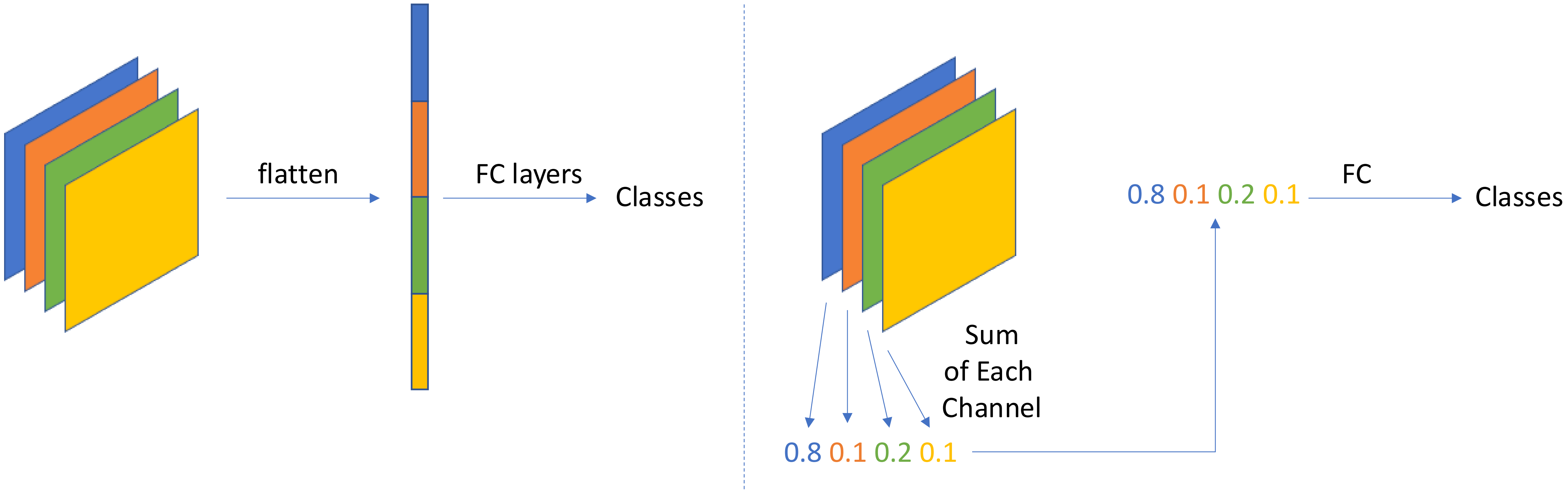
Fully-connected layer

- At the top of the network for mapping the feature responses to output labels
- Full connectivity
- Can be many layers
- Various activation functions can be used



Alternative to FC: Global Average Pooling

"Network In Network", https://arxiv.org/pdf/1312.4400.pdf



Alternative to FC: Global Average Pooling

"Network In Network", https://arxiv.org/pdf/1312.4400.pdf

- We have n feature maps: $f_1, \ldots, f_n$.
- Global average pooling is then:
$$\bar{f_i} = \sum_{x,y} f_i(x,y)$$
- Classification scores are obtained by:
$$S_c = \sum_i w_i^c \bar{f_i}$$

- Advantages:
  - No parameters, hence significant improvement in terms of overfitting problem.
  - Forces the feature maps to capture confidence maps.
  - It is more suitable to the nature of CNNs.
  - Provides invariance to spatial transformations.

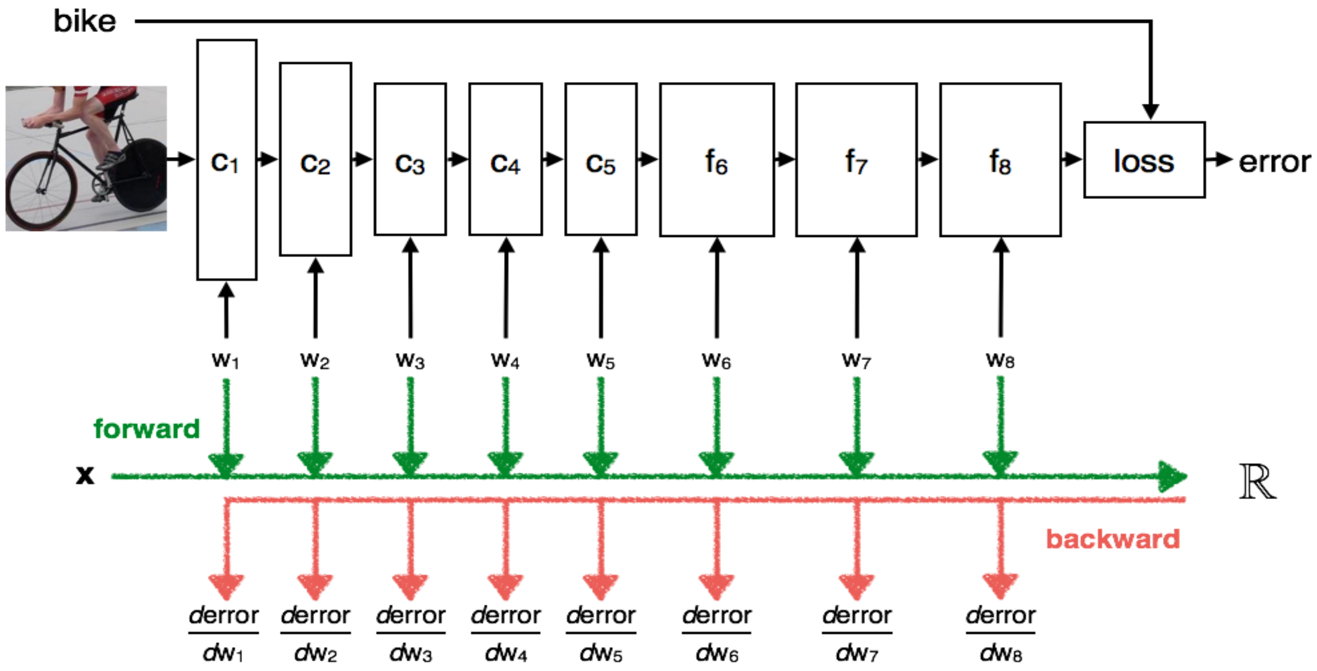
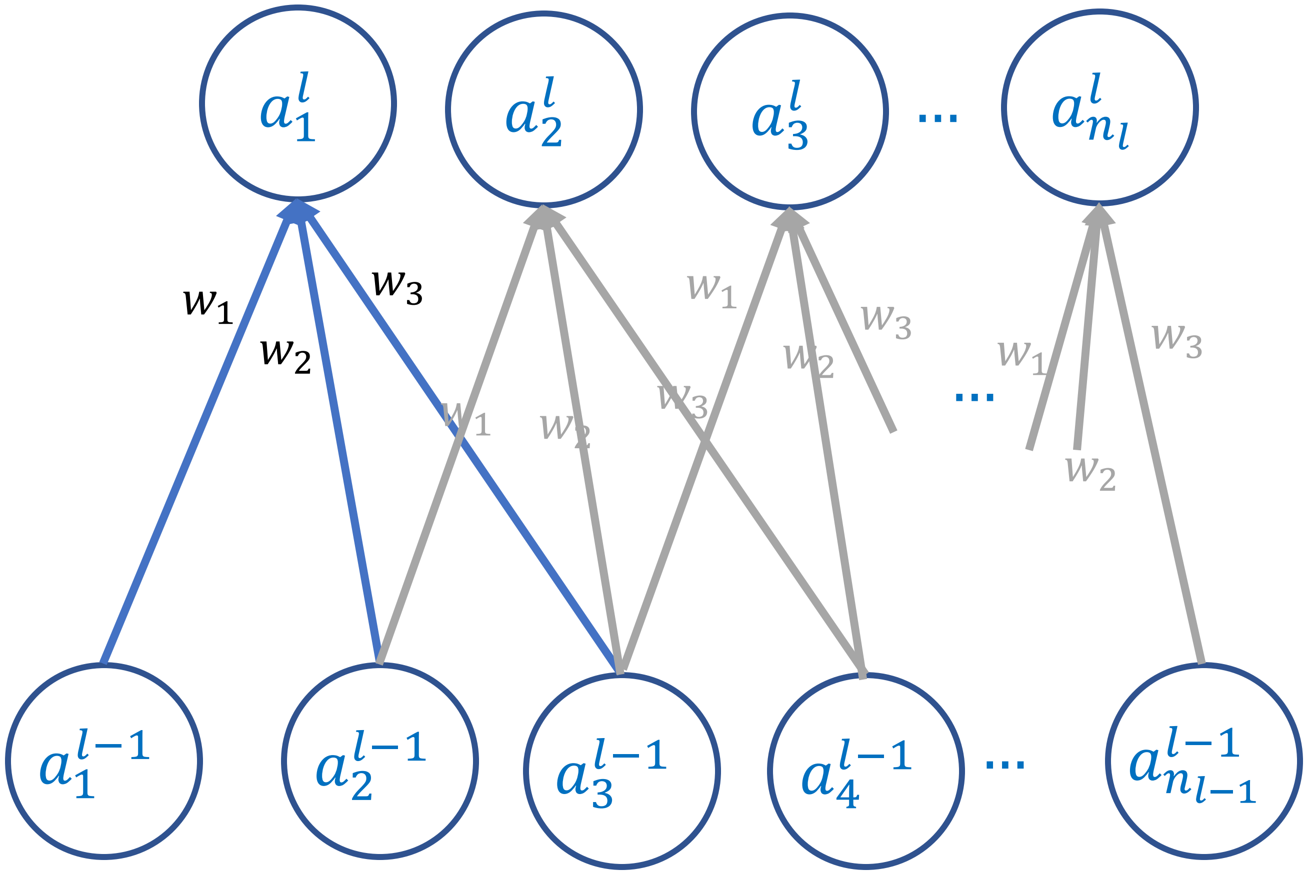


Fig: http://www.robots.ox.ac.uk/~vgg/practicals/cnn/

Training a CNN

Feed-forward through convolution



$$a_i^l = \sigma\left(net_i^l\right)$$

$$net_i^l = \sum_{j=1}^{F} w_j \cdot a_{i+j-1}^{l-1}$$

For example:

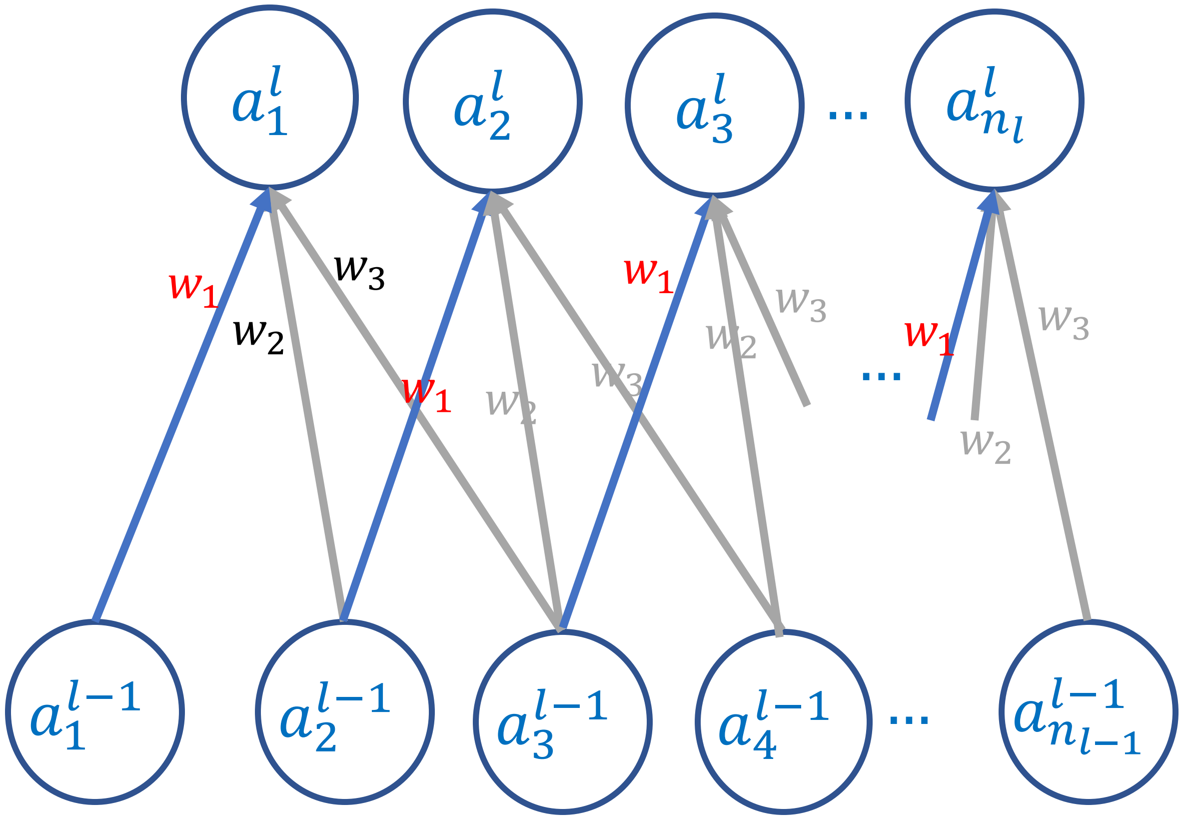
$$net_1^l = w_1 a_1^{l-1} + w_2 a_2^{l-1} + w_3 a_3^{l-1}$$

Backpropagation through convolution

Feedforward:

$$a_i^l = \sigma(net_i^l)$$

$$net_i^l = \sum_{j=1}^{F} w_j \cdot a_{i+j-1}^{l-1}$$



Gradient wrt. weights:

$$\frac{\partial L}{\partial w_k} = ?$$

$$= \frac{\partial L}{\partial a_1^l}\frac{\partial a_1^l}{\partial w_k} + \frac{\partial L}{\partial a_2^l}\frac{\partial a_2^l}{\partial w_k} \dots$$

$$= \sum_i \frac{\partial L}{\partial a_i^l}\frac{\partial a_i^l}{\partial w_k}$$

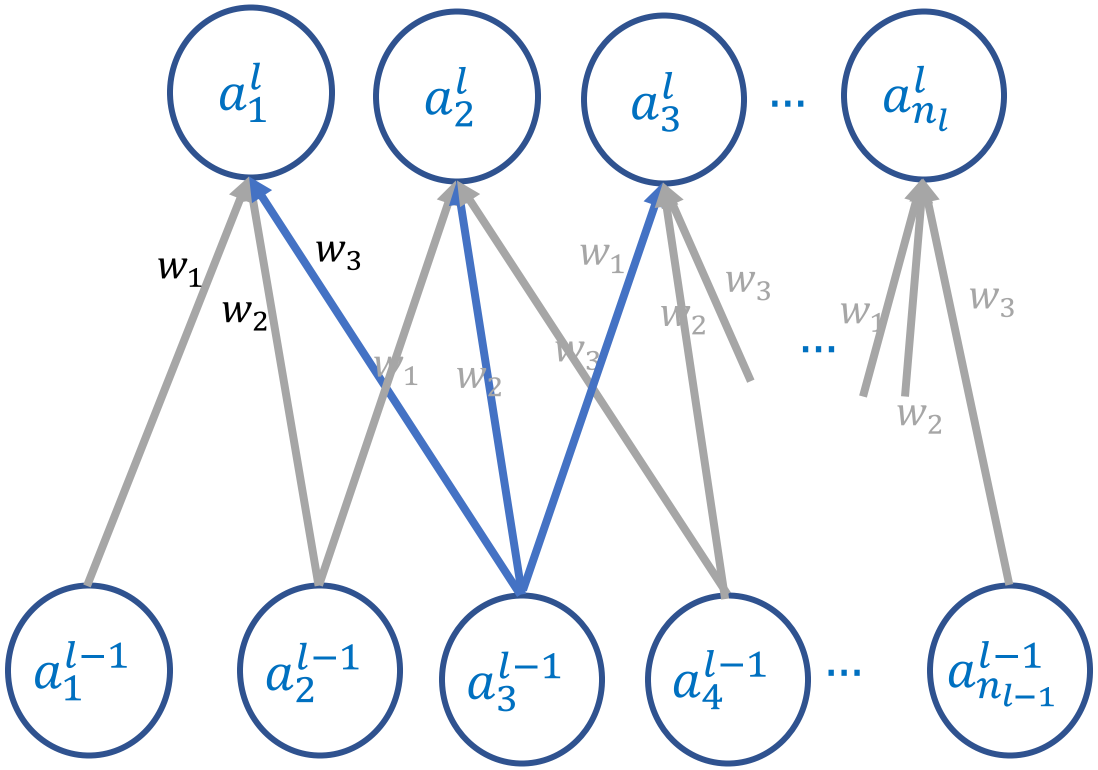
$$= \sum_i \frac{\partial L}{\partial a_i^l}\frac{\partial a_i^l}{\partial net_i^l}\frac{\partial net_i^l}{\partial w_k}$$

Backpropagation through convolution

Feedforward:

$$a_i^l = \sigma(net_i^l)$$

$$net_i^l = \sum_{j=1}^{F} w_j \cdot a_{i+j-1}^{l-1}$$



Gradient wrt. input layer:

$$\frac{\partial L}{\partial a_3^{l-1}} = ?$$

$$= \frac{\partial L}{\partial a_1^l}\frac{\partial a_1^l}{\partial net_1^l}\frac{\partial net_1^l}{\partial a_3^{l-1}} + \frac{\partial L}{\partial a_2^l}\frac{\partial a_2^l}{\partial net_2^l}\frac{\partial net_2^l}{\partial a_3^{l-1}} + \frac{\partial L}{\partial a_2^l}\frac{\partial a_2^l}{\partial net_2^l}\frac{\partial net_2^l}{\partial a_3^{l-1}}$$

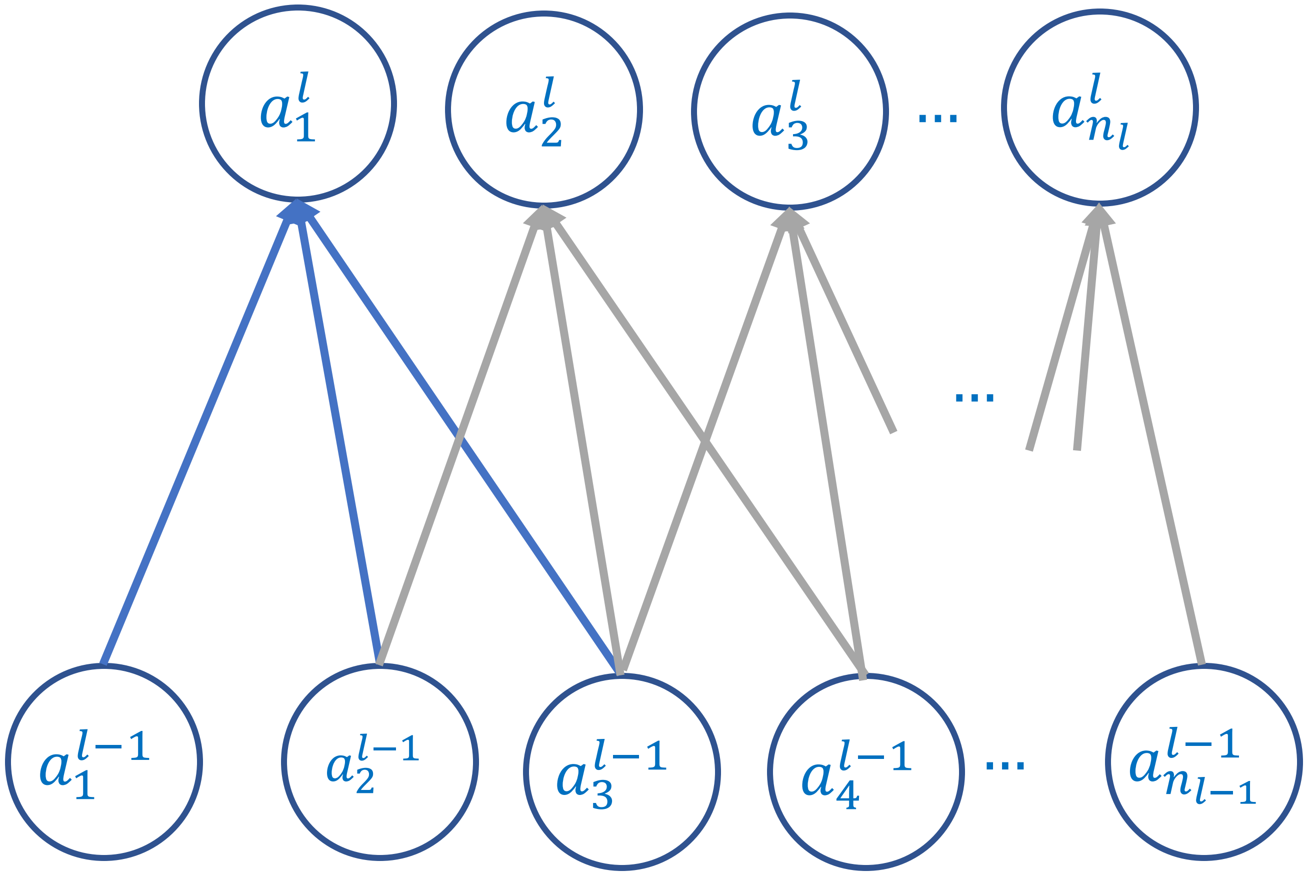
$$= \frac{\partial L}{\partial net_1^l} w_3 + \frac{\partial L}{\partial net_2^l} w_2 + \frac{\partial L}{\partial net_3^l} w_1$$

This is also convolution!

In general:

$$\frac{\partial L}{\partial a_i^{l-1}} = \sum_{j=1} \frac{\partial L}{\partial net_{i-j+1}^l} w_j$$

Feed-forward through pooling



$$a_i^l = \max \left\{a_{i+j-1}^{l-1}\right\}_{j=1}^{F}$$

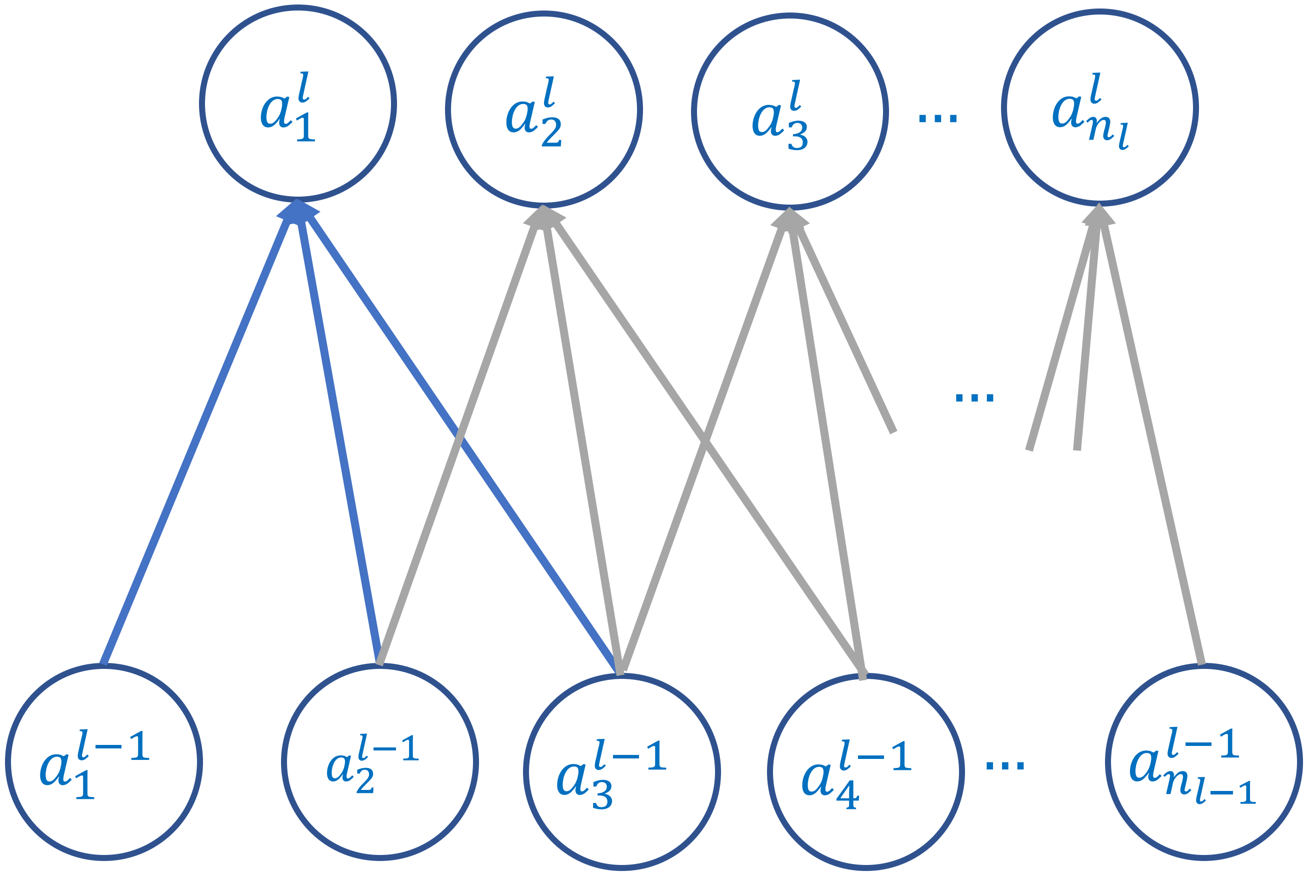
For example:

$$net_1^l = \max\left\{a_1^{l-1}, a_2^{l-1}, a_3^{l-1}\right\}$$

Backpropagation through pooling

Feedforward:

$$a_i^l = \max\{a_{i+j-1}^{l-1}\}_{j=1}^{F}$$



Using derivative of max:

$$\frac{\partial L}{\partial a_i^{l-1}} = \frac{\partial L}{\partial net_k^l}\frac{\partial net_k^l}{\partial a_i^{l-1}}$$

$$= \begin{cases} \dfrac{\partial L}{\partial net_k^l}, & a_i^{l-1} \text{ is max} \\ 0, & \text{otherwise} \end{cases}$$

This requires that we save the index of the max activation (sometimes also called *the switches*) so that gradient "routing" is handled efficiently during backpropagation.

Backpropagation

- Backpropagation through non-linearity and fully-connected layers are straight-forward

Designing CNN Architectures

A Blueprint for CNNs

`INPUT -> [[CONV -> RELU]*N -> POOL?]*M -> [FC -> RELU]*K -> FC`

where the `*` indicates repetition, and the `POOL?` indicates an optional pooling layer. Moreover, `N >= 0` (and usually `N <= 3`), `M >= 0`, `K >= 0` (and usually `K < 3`). For example, here are some common ConvNet architectures you may see that follow this pattern:

- `INPUT -> FC`, implements a linear classifier. Here `N = M = K = 0`.
- `INPUT -> CONV -> RELU -> FC`
- `INPUT -> [CONV -> RELU -> POOL]*2 -> FC -> RELU -> FC`. Here we see that there is a single CONV layer between every POOL layer.
- `INPUT -> [CONV -> RELU -> CONV -> RELU -> POOL]*3 -> [FC -> RELU]*2 -> FC` Here we see two CONV layers stacked before every POOL layer. This is generally a good idea for larger and deeper networks, because multiple stacked CONV layers can develop more complex features of the input volume before the destructive pooling operation.

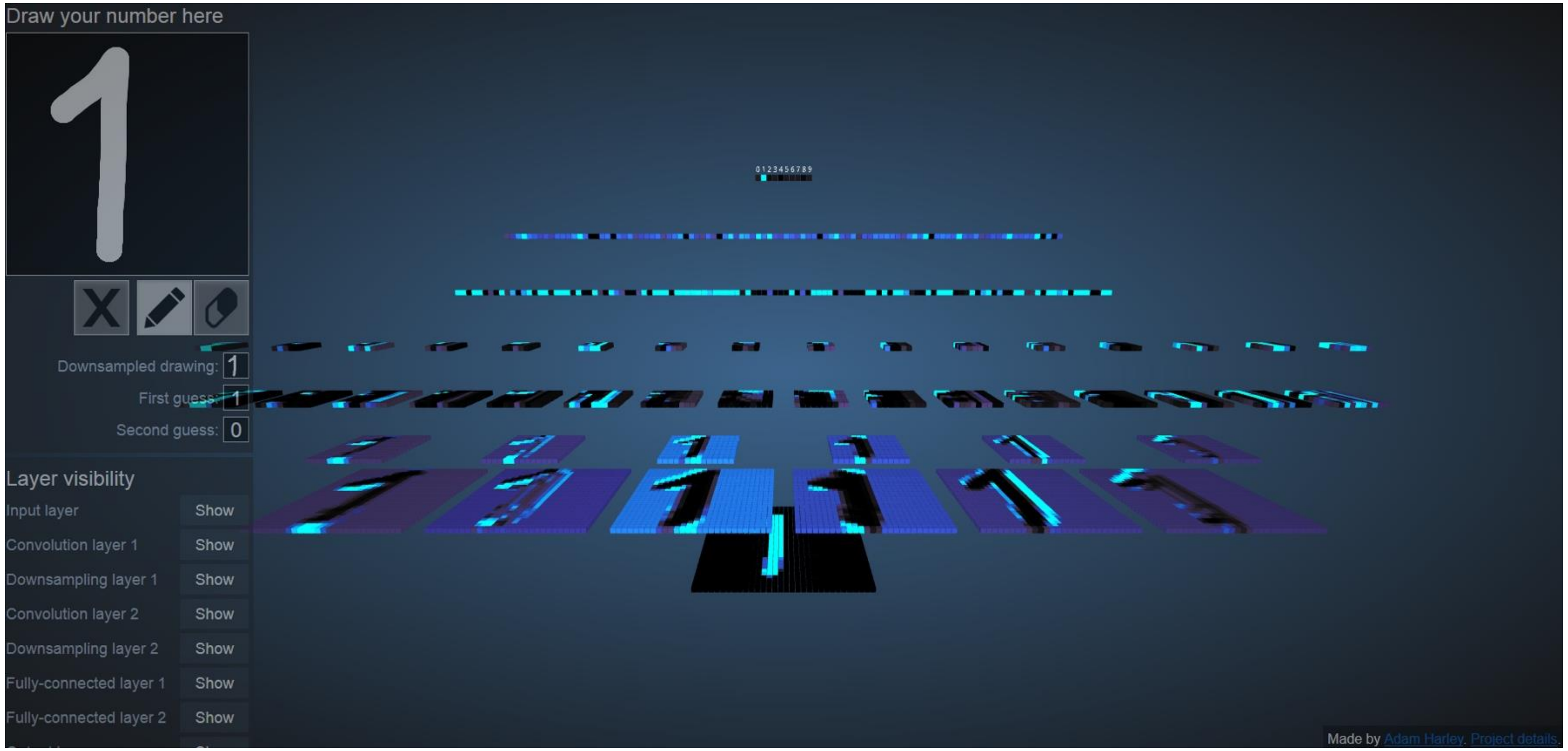
http://cs231n.github.io/convolutional-networks/

Demo

https://poloclub.github.io/cnn-explainer/

The following doesn't work, try cnn-explainer instead

http://scs.ryerson.ca/~aharley/vis/conv/



Fully Convolutional Networks (FCNs)

Long, J., Shelhamer, E., & Darrell, T. (2015). Fully convolutional networks for semantic segmentation. CVPR.

- Fully-connected layers limit the input size
- Use convolution, especially 1x1 convolution to reduce channels and layer size

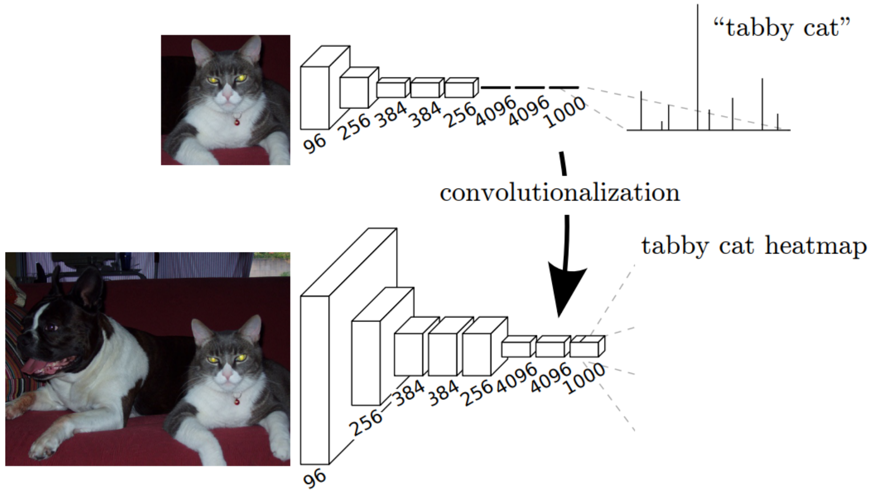


Figure 2. Transforming fully connected layers into convolution layers enables a classification net to output a heatmap. Adding layers and a spatial loss (as in Figure 1) produces an efficient machine for end-to-end dense learning.

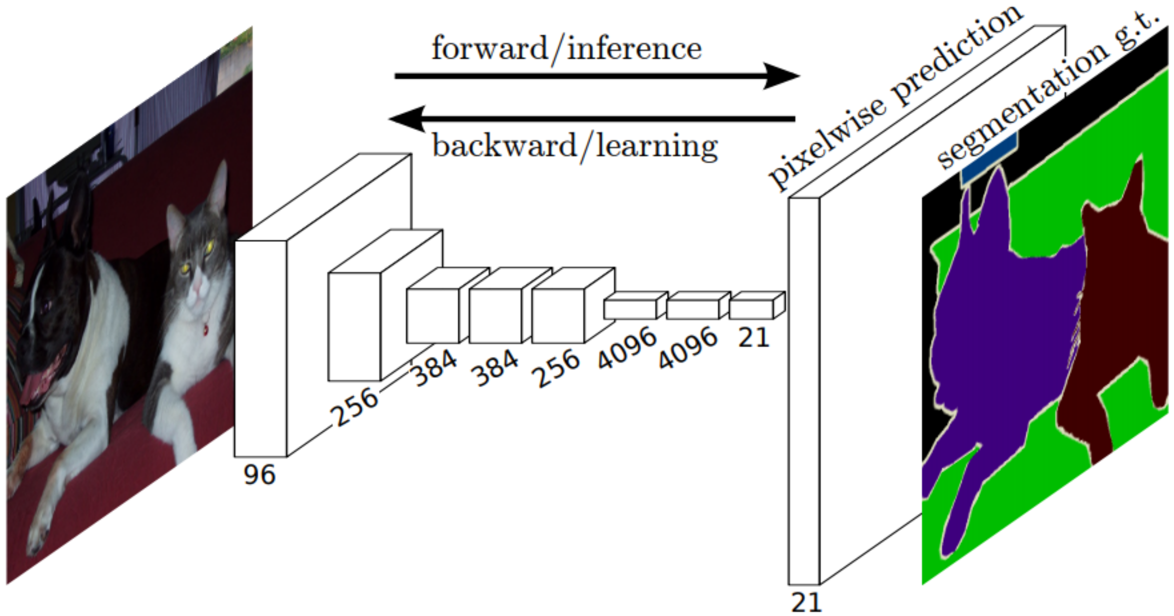


Figure 1. Fully convolutional networks can efficiently learn to make dense predictions for per-pixel tasks like semantic segmentation.

General rules of thumb: The input layer

- The size of the input layer should be divisible by 2 many times
  - Hopefully a power of 2
- E.g.,
  - 32 (e.g. CIFAR-10),
  - 64,
  - 96 (e.g. STL-10), or
  - 224 (e.g. common ImageNet ConvNets),
  - 384, and 512 etc.

General rules of thumb: The conv layer

- Small filters with stride 1
- Usually zero-padding applied to keep the input size unchanged
- In general, for a certain F, if you choose

$$P = (F - 1)/2,$$

the input size is preserved (for S=1):

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

- Number of filters:
  - A convolution channel is more expensive compared to fully-connected layer.
  - We should keep this as small as possible.

General rules of thumb: The pooling layer

- Commonly,
  - F=2 with S=2
  - Or: F=3 with S=2

- Bigger F or S is very destructive

Taking care of downsampling

- At some point(s) in the network, we need to reduce the size
- If conv layers do not downsize, then only pooling layers take care of downsampling
- If conv layers also downsize, you need to be careful about strides etc. so that
  - (i) the dimension requirements of all layers are satisfied and
  - (ii) all layers tile up properly.
- S=1 seems to work well in practice
- However, for bigger input volumes, you may try bigger strides

Trade-offs in architecture

- Between filter size and number of layers (depth)
  - Keep the layer widths fixed.
  - *"When the time complexity is roughly the same, the deeper networks with smaller filters show better results than the shallower networks with larger filters."*
- Between layer width and number of layers (depth)
  - Keep the size of the filters fixed.
  - *"We find that increasing the depth leads to considerable gains, even the width needs to be properly reduced."*
- Between filter size and layer width
  - Keep the number of layers (depth) fixed.
  - No significant difference

Convolutional Neural Networks at Constrained Time Cost

Kaiming He Jian Sun
Microsoft Research
{kahe, jiansun}@microsoft.com

4.4. Is Deeper Always Better?

The above results have shown the priority of depth for improving accuracy. With the above trade-offs, we can have a much deeper model if we further decrease width/filter sizes and increase depth. However, in experiments we find that the accuracy is stagnant or even reduced in some of our very deep attempts. There are two possible explanations: (1) the width/filter sizes are reduced overly and may harm the accuracy, or (2) overly increasing the depth will degrade the accuracy even if the other factors are not traded. To understand the main reason, *in this subsection we do not constrain the time complexity* but solely increase the depth without other changes.

Memory

Main sources of memory load:

- Activation maps:
  - Training: They need to be kept during training so that backpropagation can be performed
  - Testing: No need to keep the activations of earlier layers
- Parameters:
  - The weights, their gradients and also another copy if momentum is used
- Data:
  - The originals + their augmentations
- If all these don’t fit into memory,
  - Load your data batch by batch from disk
  - Decrease the size of your batches

Memory constraints

- Using smaller RFs with more layers means more memory since you need to store more activation maps

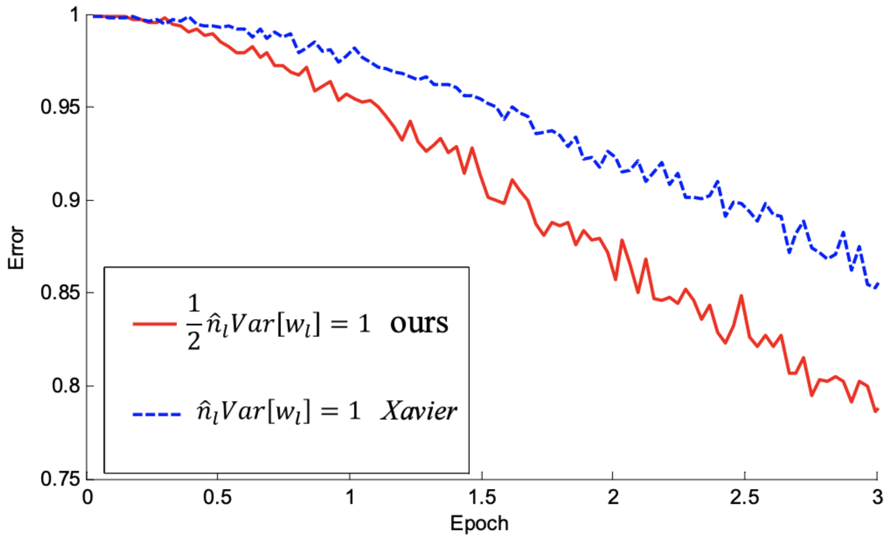
- In such memory-scarce cases,
  - the first layer may use bigger RFs with S>1
  - information loss from the input volume may be less critical than the following layers

- E.g., AlexNet uses RFs of 11x11 and S = 4 for the first layer.

How to initialize the weights?

- Option 1: randomly
  - E.g. using He initialization (check Week 8 slides)
  - This has been shown to work nicely in the literature

- Option 2:
  - Train/obtain the "filters" elsewhere and use them as the weights
  - Unsupervised pre-training using image patches (windows)
  - Avoids full feedforward and backward pass, allows the search to start from a better position
  - You may even skip training the convolutional layers



He et al., "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification", 2015.

Convolutional Clustering for Unsupervised Learning

Aysegul Dundar, Jonghoon Jin, and Eugenio Culurciello
Purdue University, West Lafayette, IN 47907, USA
{adundar,jhjin,euge}@purdue.edu

Table 3: Classification error on MNIST.

(a) Algorithms that learn the filters unsupervised.

Algorithm	600	1000	3000	All
Zhao et al. (2015) (auto-encoder)	8.4%	6.40%	4.76%	-
Rifai et al. (2011) (constractive auto-encoder)	6.3%	4.77%	3.22%	1.14%
This work (2 layers + multi dict.)	2.8%	2.5%	1.4%	0.5%

(b) Supervised and semi-supervised algorithms.

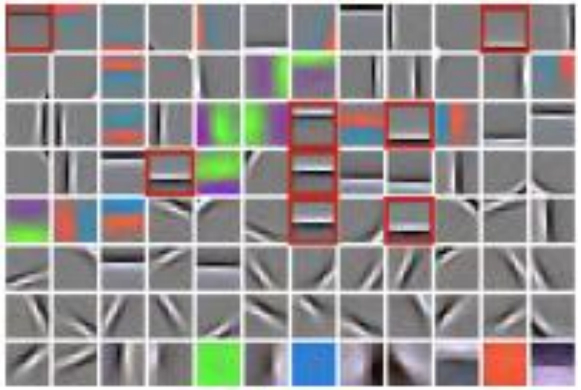
Algorithm	600	1000	3000	All
LeCun et al. (1998) (convnet)	7.68%	6.45%	3.35%	
Lee (2013) (psuedo-label)	5.03%	3.46%	2.69%	-
Zhao et al. (2015) (semi-supervised auto-encoder)	3.31%	2.83%	2.10%	0.71%
Kingma et al. (2014) (generative models)	2.59%	2.40%	2.18%	0.96%
Rasmus et al. (2015) (semi-supervised ladder)	-	1.0%	-	-

3.1 Learning Filters with k-means

Our method for learning filters is based on the k-means algorithm. The classic k-means algorithm finds cluster centroids that minimize the distance between points in the Euclidean space. In this context, the points are randomly extracted image patches and the centroids are the filters that will be used to encode images. From this perspective, k-means algorithm learns a dictionary $D \in \mathbb{R}^{n \times k}$ from the data vector $w^{(i)} \in \mathbb{R}^n$ for $i = 1, 2, ..., m$. The algorithm finds the dictionary as follows:

$$\begin{aligned} s_j^{(i)} &:= \begin{cases} {D^{(j)}}^T w^{(i)} & \text{if } j = \operatorname{argmax}_l \left| {D^{(l)}}^T w^{(i)} \right|, \\ 0 & \text{otherwise,} \end{cases} \\ D &:= W S^T + D, \\ D^{(j)} &:= \frac{D^{(j)}}{||D^{(j)}||_2}, \end{aligned} \tag{1}$$

where $s^{(i)} \in \mathbb{R}^k$ is the code vector associated with the input $w^{(i)}$, and $D^{(j)}$ is the j'th column of the dictionary D. The matrices $W \in \mathbb{R}^{n \times m}$ and $S \in \mathbb{R}^{k \times m}$ have the columns $w^{(i)}$ and $s^{(i)}$, respectively. $w^{(i)}$'s are randomly extracted patches from input images that have the same dimension as the dictionary vectors, $D^{(j)}$.



(a) k-means

(b) convolutional k-means

Transfer learning:
using a trained CNN & fine-tuning

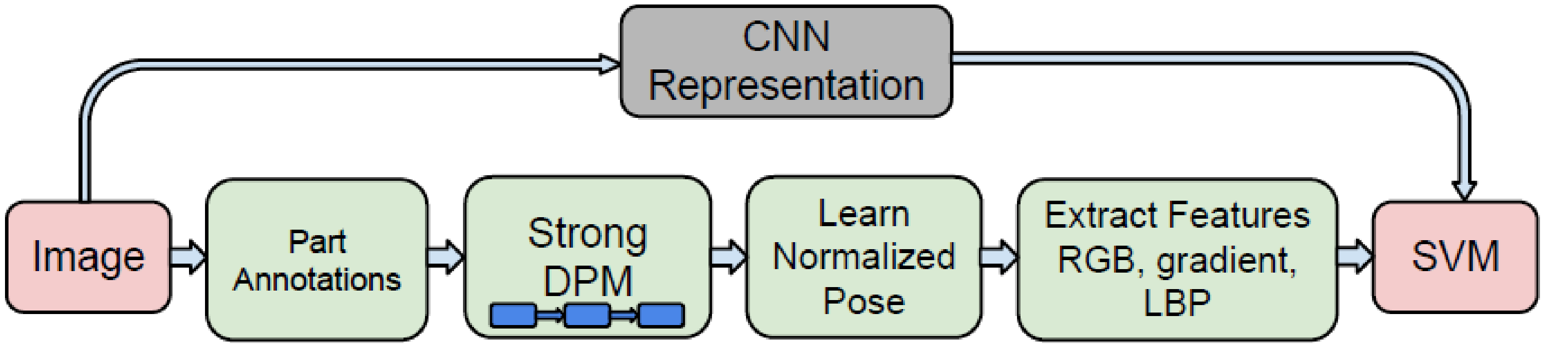
Using trained CNN

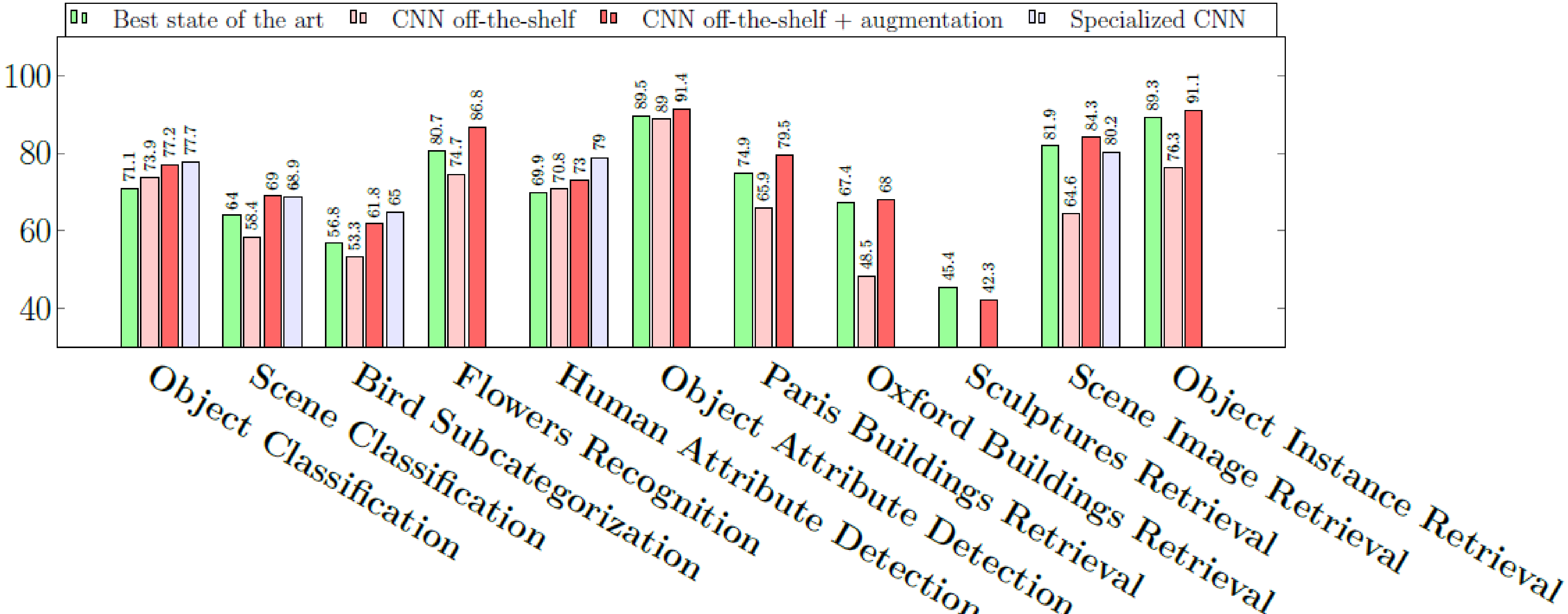
- Also called transfer learning
  - Rare to design and train a CNN from scratch!
- Take a trained CNN, e.g., AlexNet
  - Use a trained CNN as a feature detector:
    - Remove the last fully-connected layer
    - The activations of the remaining layer are called CNN codes
    - This yields a 4096 dimensional feature vector for AlexNet
    - Now, add a fully-connected layer for your problem and train a linear classifier on your dataset.
  - Alternatively, fine-tune the whole network with your new layer and outputs
    - You may limit updating only to the last layers because earlier layers are generic, and quite dataset independent
- Pre-trained CNNs

CNN Features off-the-shelf: an Astounding Baseline for Recognition

Ali Sharif Razavian Hossein Azizpour Josephine Sullivan Stefan Carlsson
CVAP, KTH (Royal Institute of Technology)
Stockholm, Sweden
{razavian,azizpour,sullivan,stefanc}@csc.kth.se

2014





Finetuning

1. If the new dataset is small and similar to the original dataset used to train the CNN:
   - Finetuning the whole network may lead to overfitting
   - Just train the newly added layer

2. If the new dataset is big and similar to the original dataset:
   - The more, the merrier: go ahead and train the whole network

3. If the new dataset is small and different from the original dataset:
   - Not a good idea to train the whole network
   - However, add your new layer not to the top of the network, since those parts are very dataset (problem) specific
   - Add your layer to earlier parts of the network

4. If the new dataset is big and different from the original dataset:
   - We can "finetune" the whole network
   - This amounts to a new training problem by initializing the weights with those of another network

More on finetuning

- You cannot change the architecture of the trained network (e.g., remove layers) arbitrarily

- The sizes of the layers can be varied
  - For convolution & pooling layers, this is straightforward
  - For the fully-connected layers: you can convert the fully-connected layers to convolution layers, which makes it size-independent.

- You should use small learning rates while fine-tuning

See also:

Preprint release. Full citation: Yosinski J, Clune J, Bengio Y, and Lipson H. *How transferable are features in deep neural networks?* In Advances in Neural Information Processing Systems 27 (NIPS '14), NIPS Foundation, 2014.

How transferable are features in deep neural networks?

Jason Yosinski,[1] **Jeff Clune,**[2] **Yoshua Bengio,**[3] **and Hod Lipson**[4]
[1] Dept. Computer Science, Cornell University
[2] Dept. Computer Science, University of Wyoming
[3] Dept. Computer Science & Operations Research, University of Montreal
[4] Dept. Mechanical & Aerospace Engineering, Cornell University

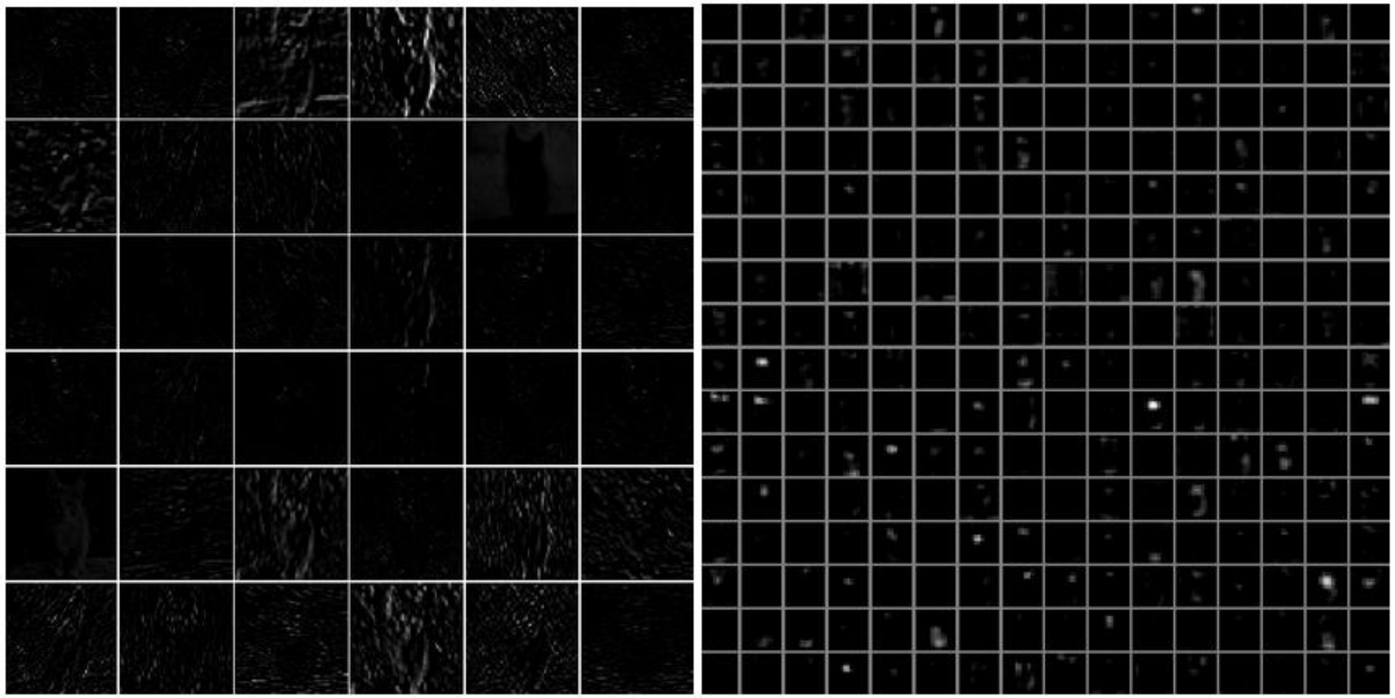
Visualizing and Understanding CNNs

Many different mechanisms

- Visualize layer activations
- Visualize the weights (i.e., filters)
- Visualize examples that maximally activate a neuron
- Visualize a 2D embedding of the inputs based on their CNN codes
- Occlude parts of the window and see how the prediction is affected
- Data gradients

Visualize activations during training

- Activations are dense at the beginning.
  - They should get sparser during training.
- If some activation maps are all zero for many inputs, dying neuron problem => high learning rate in the case of ReLUs.

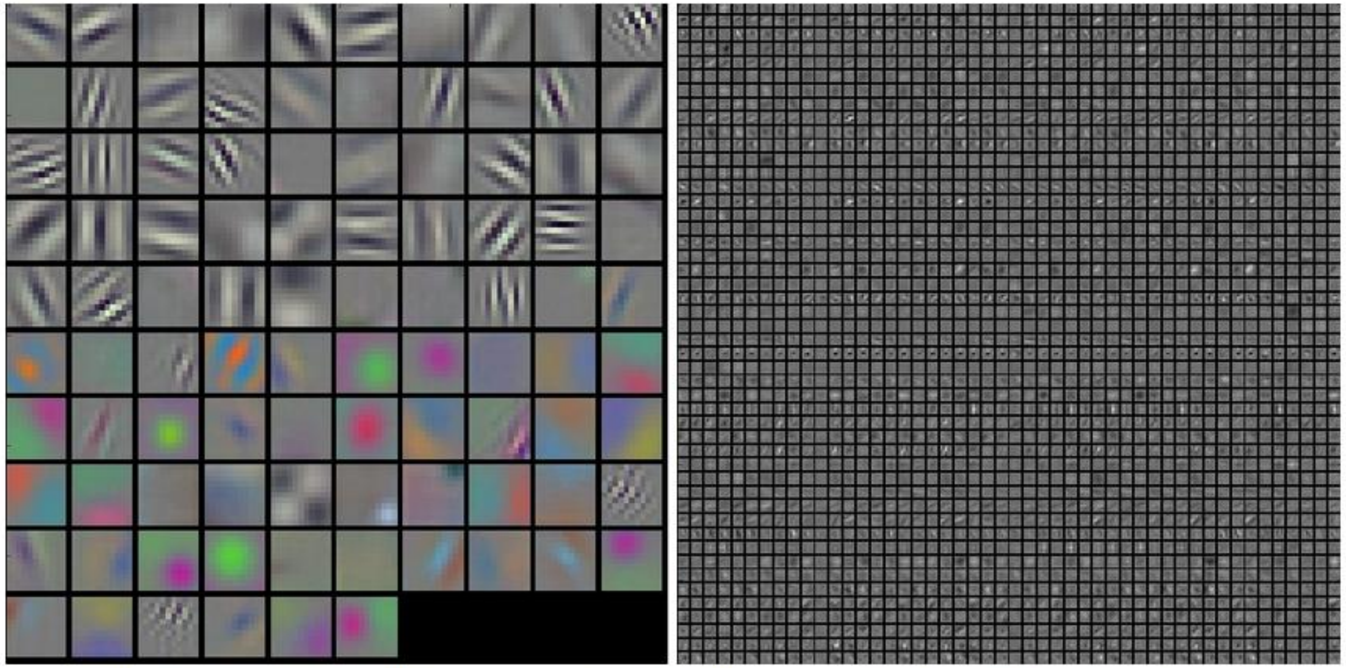


Typical-looking activations on the first CONV layer (left), and the 5th CONV layer (right) of a trained AlexNet looking at a picture of a cat. Every box shows an activation map corresponding to some filter. Notice that the activations are sparse (most values are zero, in this visualization shown in black) and mostly local.

http://cs231n.github.io/convolutional-networks/

Visualize the weights

- We can directly look at the filters of all layers
- First layer is easier to interpret
- Filters shouldn't look noisy



Typical-looking filters on the first CONV layer (left), and the 2nd CONV layer (right) of a trained AlexNet. Notice that the first-layer weights are very nice and smooth, indicating nicely converged network. The color/grayscale features are clustered because the AlexNet contains two separate streams of processing, and an apparent consequence of this architecture is that one stream develops high-frequency grayscale features and the other low-frequency color features. The 2nd CONV layer weights are not as interpretable, but it is apparent that they are still smooth, well-formed, and absent of noisy patterns.

http://cs231n.github.io/convolutional-networks/

Visualize the inputs that maximally activate a neuron

- Keep track of which images activate a neuron most

Rich feature hierarchies for accurate object detection and semantic segmentation
Tech report (v5)

Ross Girshick   Jeff Donahue   Trevor Darrell   Jitendra Malik
UC Berkeley
{rbg,jdonahue,trevor,malik}@eecs.berkeley.edu

Maximally activating images for some POOL5 (5th pool layer) neurons of an AlexNet. The activation values and the receptive field of the particular neuron are shown in white. (In particular, note that the POOL5 neurons are a function of a relatively large portion of the input image!) It can be seen that some neurons are responsive to upper bodies, text, or specular highlights.

http://cs231n.github.io/convolutional-networks/

Embed the codes in a lower-dimensional space

- Place images into a 2D space such that images which produce similar CNN codes are placed close.
- You can use, e.g., t-Distributed Stochastic Neighbor Embedding (t-SNE)

t-SNE embedding of a set of images based on their CNN codes. Images that are nearby each other are also close in the CNN representation space, which implies that the CNN "sees" them as being very similar. Notice that the similarities are more often class-based and semantic rather than pixel and color-based. For more details on how this visualization was produced the associated code, and more related visualizations at different scales refer to t-SNE visualization of CNN codes.

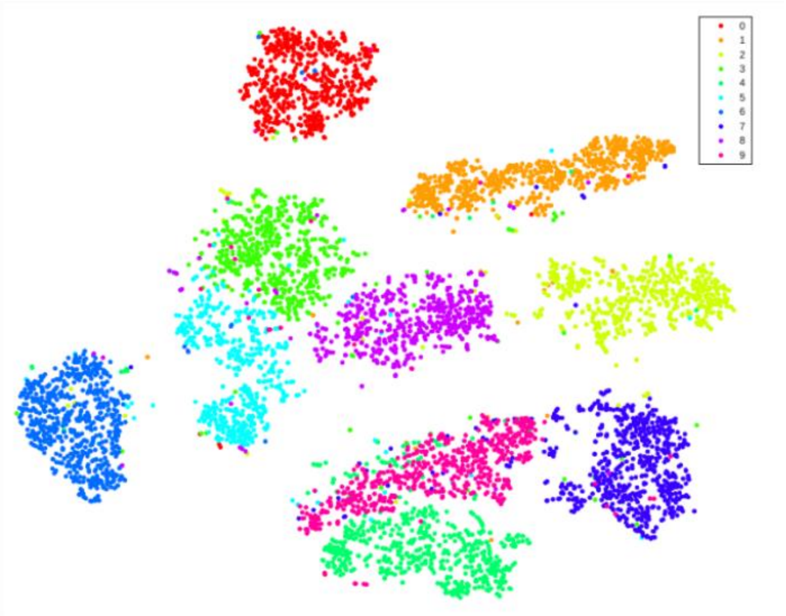


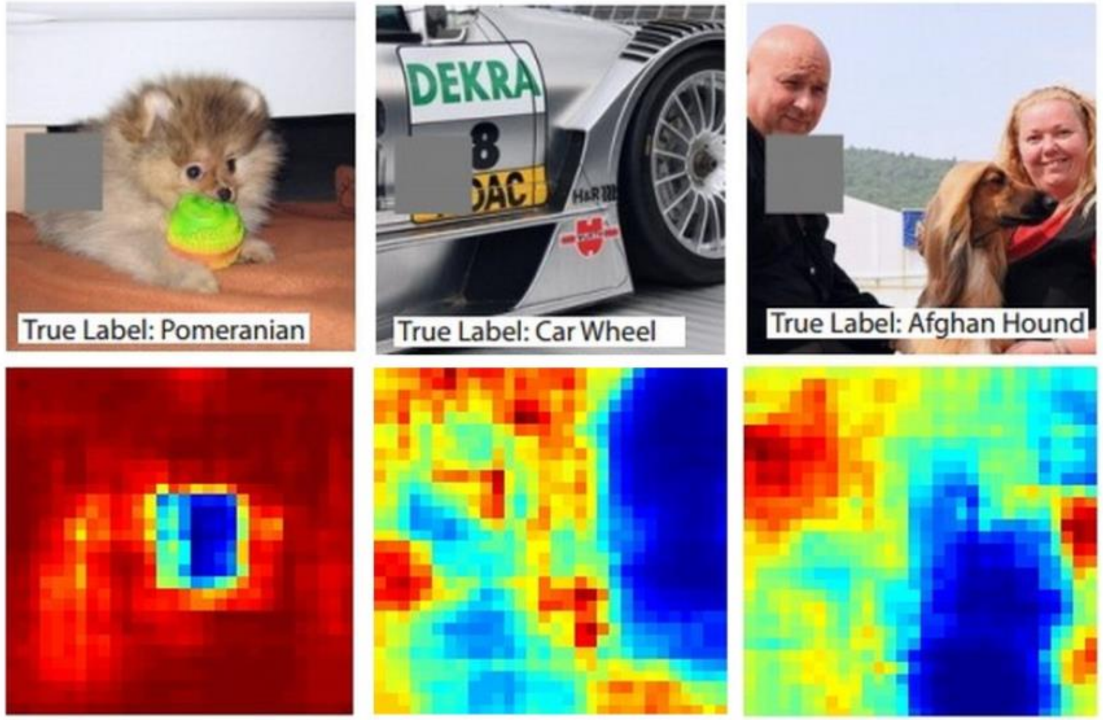
Figure 1 : Illustration of t-SNE on MNIST dataset

Figure: Laurens van der Maaten and Geoffrey Hinton

http://cs231n.github.io/convolutional-networks/

Occlude parts of the image

- Slide an "occlusion window" over the image
- For each occluded image, determine the class prediction confidence/probability.



Three input images (top). Notice that the occluder region is shown in grey. As we slide the occluder over the image we record the probability of the correct class and then visualize it as a heatmap (shown below each image). For instance, in the left-most image we see that the probability of Pomeranian plummets when the occluder covers the face of the dog, giving us some level of confidence that the dog's face is primarily responsible for the high classification score. Conversely, zeroing out other parts of the image is seen to have relatively negligible impact.

http://cs231n.github.io/convolutional-networks/

Data gradients

- Generate an image that maximizes the class score.

More formally, let $S_c(I)$ be the score of the class c, computed by the classification layer of the ConvNet for an image I. We would like to find an L_2-regularised image, such that the score S_c is high:

$$\arg\max_I S_c(I) - \lambda \|I\|_2^2, \tag{1}$$

where λ is the regularisation parameter. A locally-optimal I can be found by the back-propagation

- Use: Gradient ascent!

Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps

Karen Simonyan   Andrea Vedaldi   Andrew Zisserman
Visual Geometry Group, University of Oxford
{karen,vedaldi,az}@robots.ox.ac.uk

2014

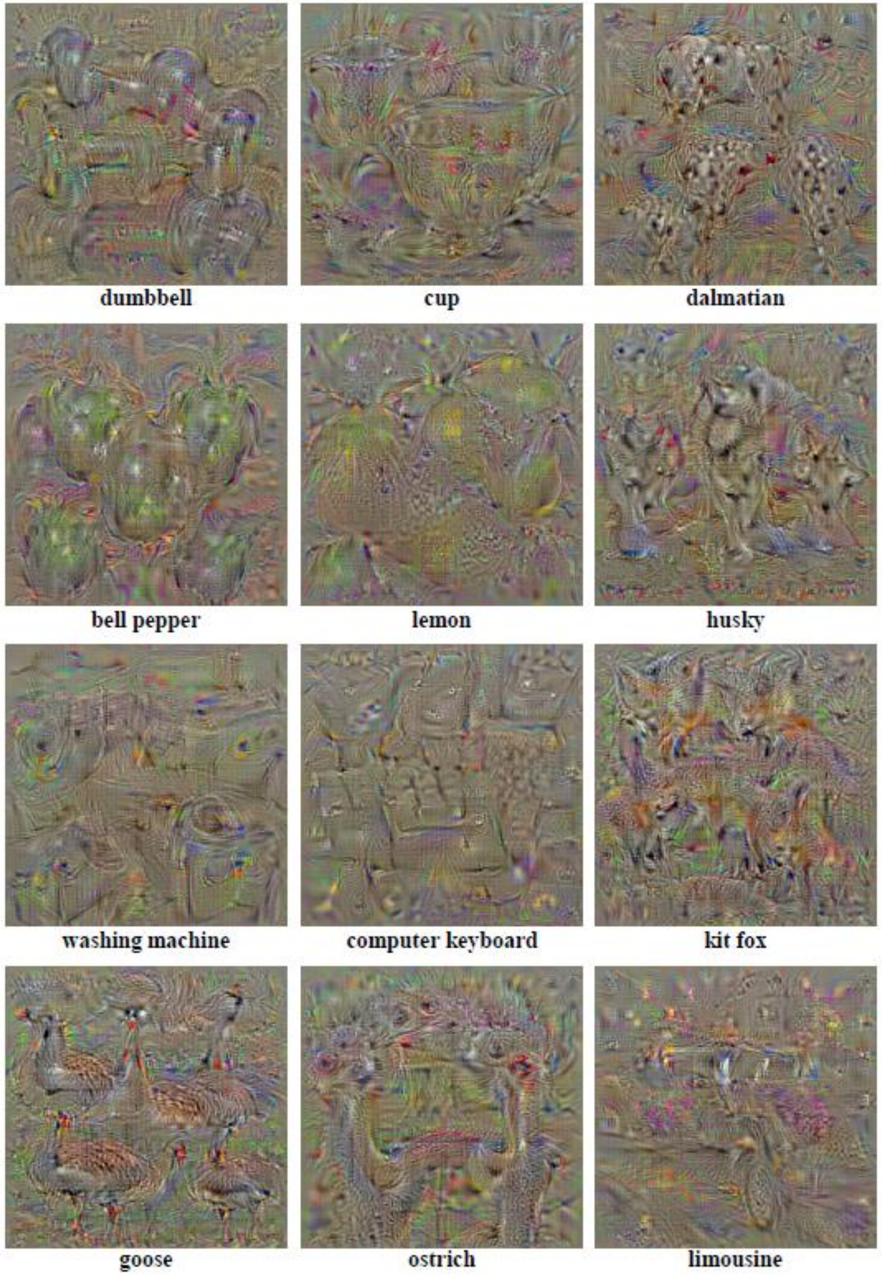


Figure 1: Numerically computed images, illustrating the class appearance models, learnt by a ConvNet, trained on ILSVRC-2013. Note how different aspects of class appearance are captured in a single image. Better viewed in colour.

Data gradients

- The gradient with respect to the input is high for pixels which are on the object

We start with a motivational example. Consider the linear score model for the class c:

$$S_c(I) = w_c^T I + b_c, \tag{2}$$

where the image I is represented in the vectorised (one-dimensional) form, and w_c and b_c are respectively the weight vector and the bias of the model. In this case, it is easy to see that the magnitude of elements of w defines the importance of the corresponding pixels of I for the class c.

In the case of deep ConvNets, the class score $S_c(I)$ is a highly non-linear function of I, so the reasoning of the previous paragraph can not be immediately applied. However, given an image I_0, we can approximate $S_c(I)$ with a linear function in the neighbourhood of I_0 by computing the first-order Taylor expansion:

$$S_c(I) \approx w^T I + b, \tag{3}$$

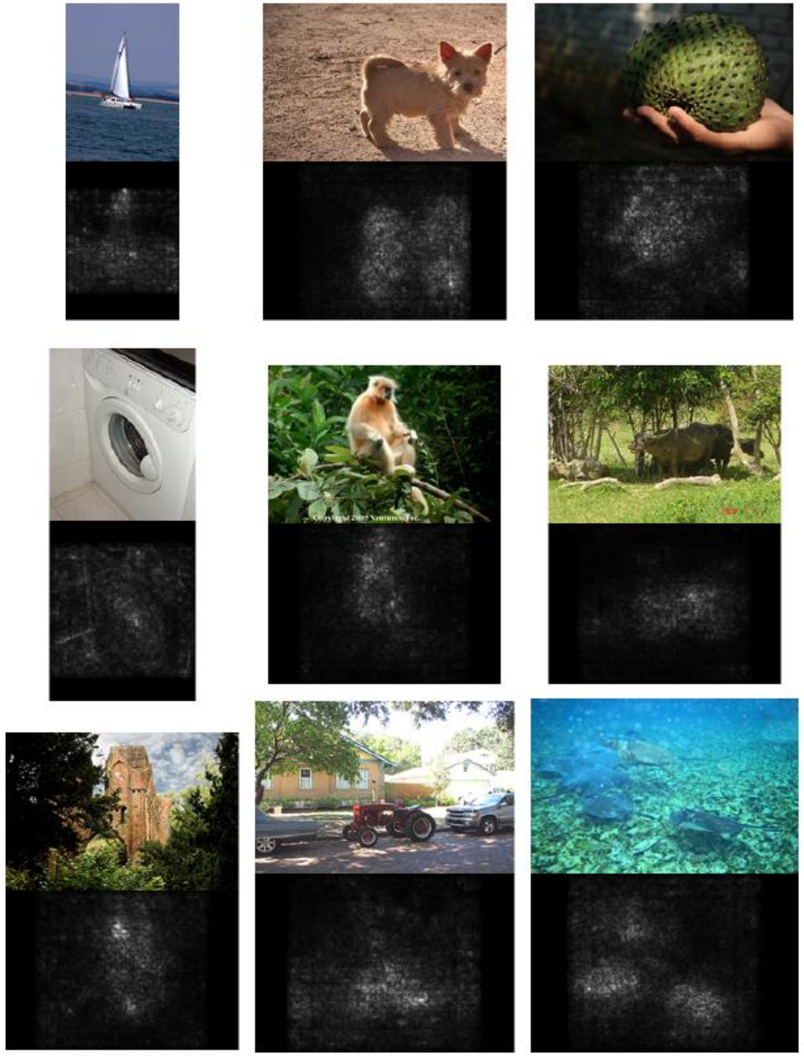
where w is the derivative of S_c with respect to the image I at the point (image) I_0:

$$w = \left. \frac{\partial S_c}{\partial I} \right|_{I_0}. \tag{4}$$

Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps

Karen Simonyan   Andrea Vedaldi   Andrew Zisserman
Visual Geometry Group, University of Oxford
{karen,vedaldi,az}@robots.ox.ac.uk

2014



Class Activation Maps

- Weighted combination of the feature maps before GAP:

$$M(x,y) = \sum_k w_k^c f_k(x,y)$$

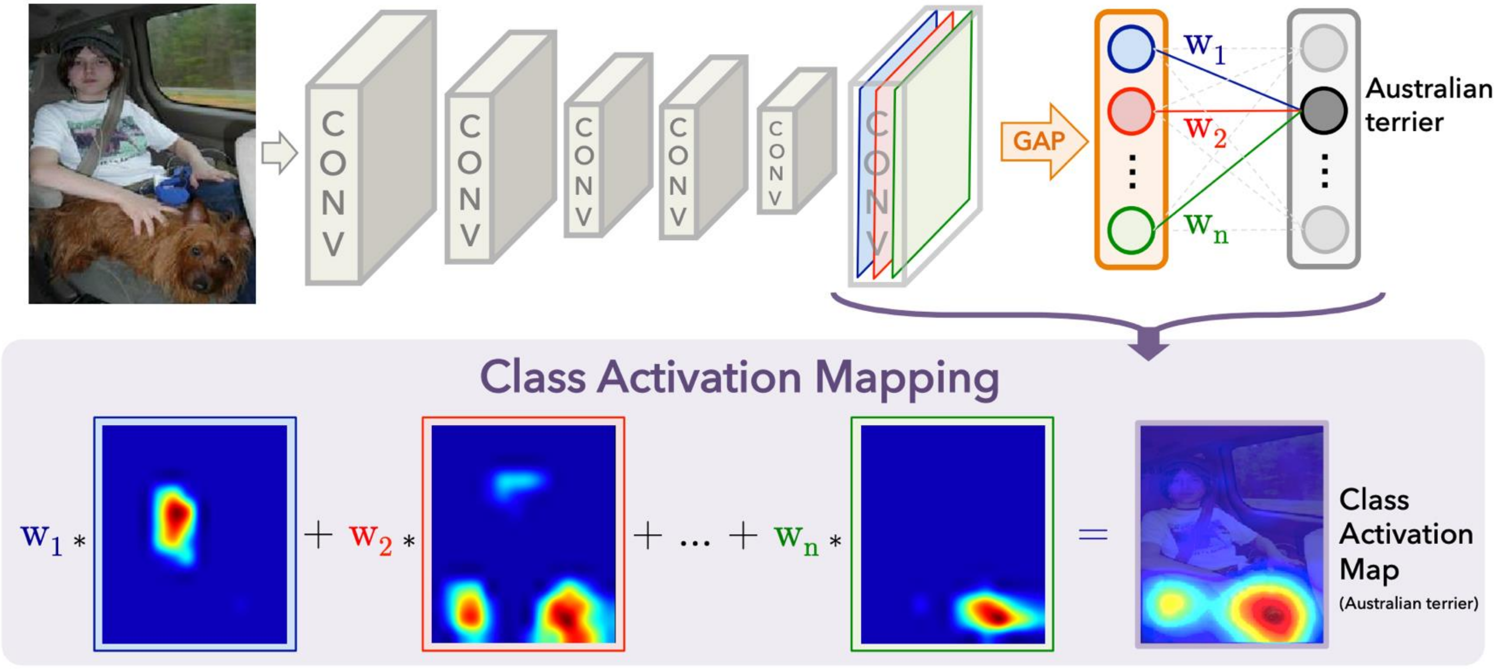


Figure 2. Class Activation Mapping: the predicted class score is mapped back to the previous convolutional layer to generate the class activation maps (CAMs). The CAM highlights the class-specific discriminative regions.

B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Learning deep features for discriminative localization," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 2921–2929.

Class Activation Maps

- GradCAM:

$$\alpha_k^c = \sum_{x,y} \frac{\partial S_c}{\partial f_k(x,y)}$$

$$M^c(x,y) = ReLU\left(\sum_k \alpha_k^c \, f_k(x,y)\right)$$

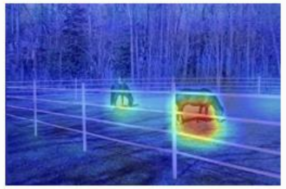
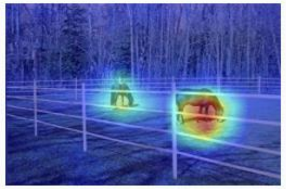
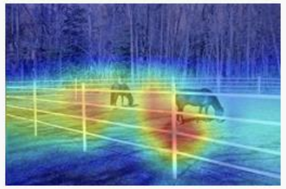
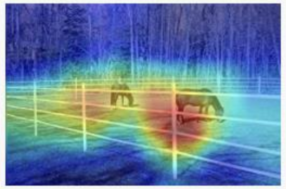
Network	Image	GradCAM	GradCAM++
VGG16			
Resnet50			

Figure: https://pypi.org/project/grad-cam/

R. R. Selvaraju, A. Das, R. Vedantam, M. Cogswell, D. Parikh, and D. Batra, “Grad-cam: Why did you say that? visual explanations from deep networks via gradient-based localization,” arXiv preprint arXiv:1610.02391, 2016.

Chattopadhay, A., Sarkar, A., Howlader, P., & Balasubramanian, V. N. (2018, March). Grad-cam++: Generalized gradient-based visual explanations for deep convolutional networks. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)* (pp. 839-847). IEEE.

Feature inversion

- Learns to reconstruct an image from its representation

This section introduces our method to compute an approximate inverse of an image representation. This is formulated as the problem of finding an image whose representation best matches the one given [34]. Formally, given a representation function $\Phi : \mathbb{R}^{H \times W \times C} \to \mathbb{R}^d$ and a representation $\Phi_0 = \Phi(\mathbf{x}_0)$ to be inverted, reconstruction finds the image $\mathbf{x} \in \mathbb{R}^{H \times W \times C}$ that minimizes the objective:

$$\mathbf{x}^* = \operatorname*{argmin}_{\mathbf{x} \in \mathbb{R}^{H \times W \times C}} \ell(\Phi(\mathbf{x}), \Phi_0) + \lambda \mathcal{R}(\mathbf{x}) \tag{1}$$

where the loss ℓ compares the image representation $\Phi(\mathbf{x})$ to the target one Φ_0 and $\mathcal{R} : \mathbb{R}^{H \times W \times C} \to \mathbb{R}$ is a regulariser capturing a *natural image prior*.

- Regularization term here is the key factor, e.g. a combination of the two terms:

$$\mathcal{R}_\alpha(\mathbf{x}) = \|\mathbf{x}\|_\alpha^\alpha \qquad \mathcal{R}_{V^\beta}(\mathbf{x}) = \sum_{i,j} \left((x_{i,j+1} - x_{ij})^2 + (x_{i+1,j} - x_{ij})^2 \right)^{\frac{\beta}{2}}$$

Understanding Deep Image Representations by Inverting Them

Aravindh Mahendran
University of Oxford

Andrea Vedaldi
University of Oxford

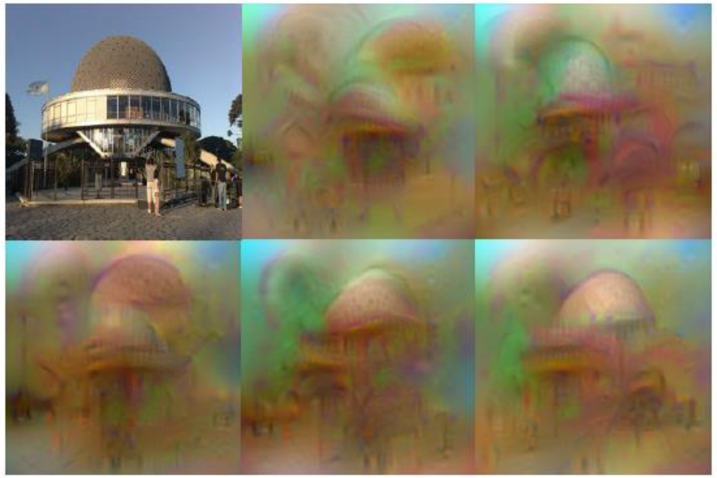


Figure 1. **What is encoded by a CNN?** The figure shows five possible reconstructions of the reference image obtained from the 1,000-dimensional code extracted at the penultimate layer of a reference CNN[13] (before the softmax is applied) trained on the ImageNet data. From the viewpoint of the model, all these images are practically equivalent. This image is best viewed in color/screen.

Feature inversion with perceptual losses

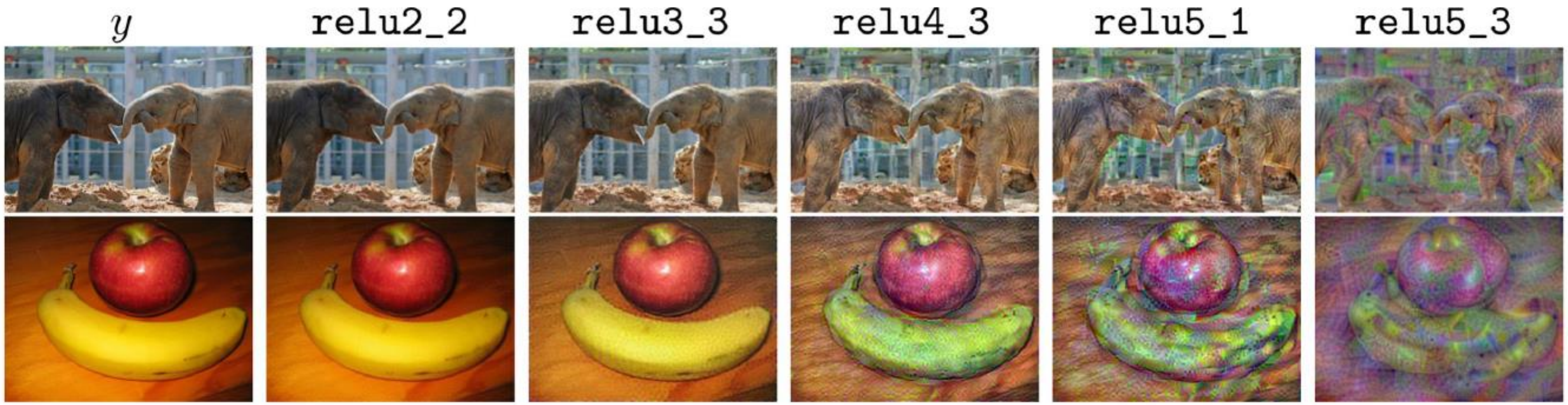


Figure from Johnson, Alahi, and Fei-Fei, “Perceptual Losses for Real-Time Style Transfer and Super-Resolution”, ECCV 2016.

Visualization distill.pub

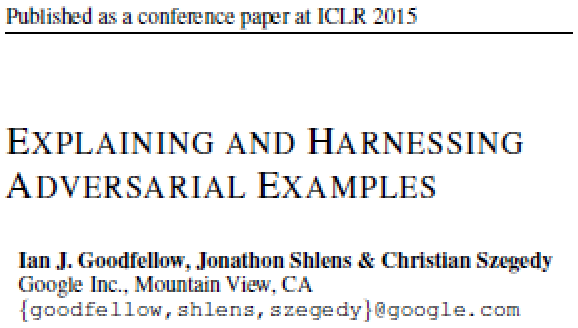
https://distill.pub/2017/feature-visualization/

https://distill.pub/2018/building-blocks/

Fooling ConvNets

- Given an image I labeled as l_1, find minimum "r" (noise) such that $I + r$ is classified as a different label, l_2.
- I.e., minimize:

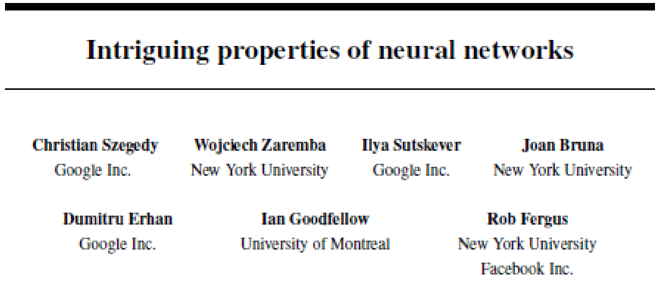
$$\arg\min_{r} loss(I + r, l_2) + c|r|$$



Published as a conference paper at ICLR 2015

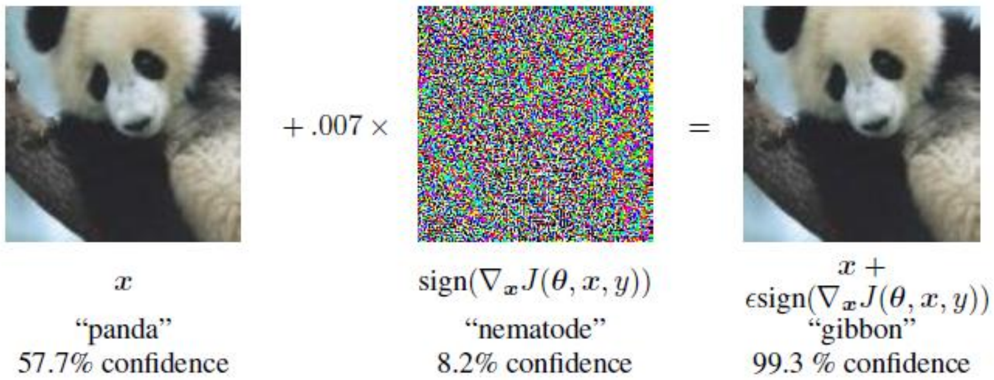
EXPLAINING AND HARNESSING ADVERSARIAL EXAMPLES

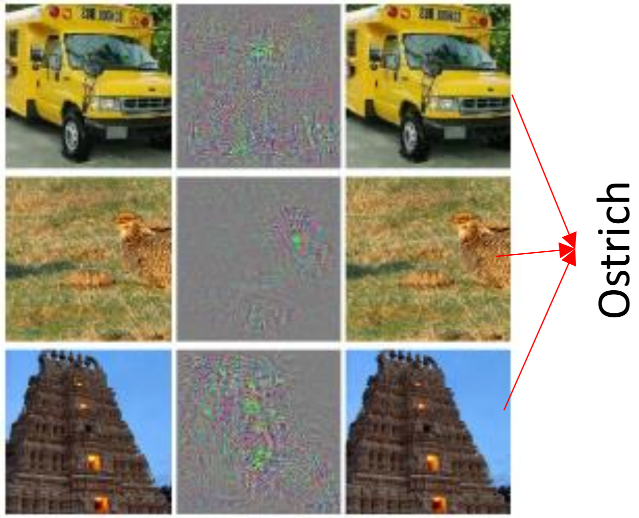
Ian J. Goodfellow, Jonathon Shlens & Christian Szegedy
Google Inc., Mountain View, CA
{goodfellow,shlens,szegedy}@google.com



Intriguing properties of neural networks

Christian Szegedy, Google Inc.
Wojciech Zaremba, New York University
Ilya Sutskever, Google Inc.
Joan Bruna, New York University
Dumitru Erhan, Google Inc.
Ian Goodfellow, University of Montreal
Rob Fergus, New York University, Facebook Inc.





More on adversarial examples

- How to classify adversarial examples correctly?
  - You need to train your network against them!
  - That is very expensive and training against all kinds of adversarial examples is not possible
  - However, training against adversarial examples increases accuracy on non-adversarial examples as well.
- They are still an unsolved issue in neural networks
- Adversarial examples are problems of any learning method
- See I. Goodfellow for more on adversarial examples:
  - http://www.kdnuggets.com/2015/07/deep-learning-adversarial-examples-misconceptions.html

There Is No Free Lunch In Adversarial Robustness
(But There Are Unexpected Benefits)

Dimitris Tsipras*
MIT
tsipras@mit.edu

Shibani Santurkar*
MIT
shibani@mit.edu

Logan Engstrom
MIT
engstrom@mit.edu

Alexander Turner
MIT
turneram@mit.edu

Aleksander Mądry
MIT
madry@mit.edu

2018

- "We provide a new understanding of the fundamental nature of adversarially robust classifiers and how they differ from standard models. In particular, we show that there provably exists a trade-off between the standard accuracy of a model and its robustness to adversarial perturbations. We demonstrate an intriguing phenomenon at the root of this tension: a certain dichotomy between "robust" and "non-robust" features. We show that while robustness comes at a price, it also has some surprising benefits. Robust models turn out to have interpretable gradients and feature representations that align unusually well with salient data characteristics. In fact, they yield striking feature interpolations that have thus far been possible to obtain only using generative models such as GANs."