CSE 152: Computer Vision

Hao Su

Lecture 9: Convolutional Neural Network and Learning

Recap: Bias and Variance

- Bias – error caused because the model lacks the ability to represent the (complex) concept

- Variance – error caused because the learning algorithm overreacts to small changes (noise) in the training data

TotalLoss = Bias + Variance (+ noise)

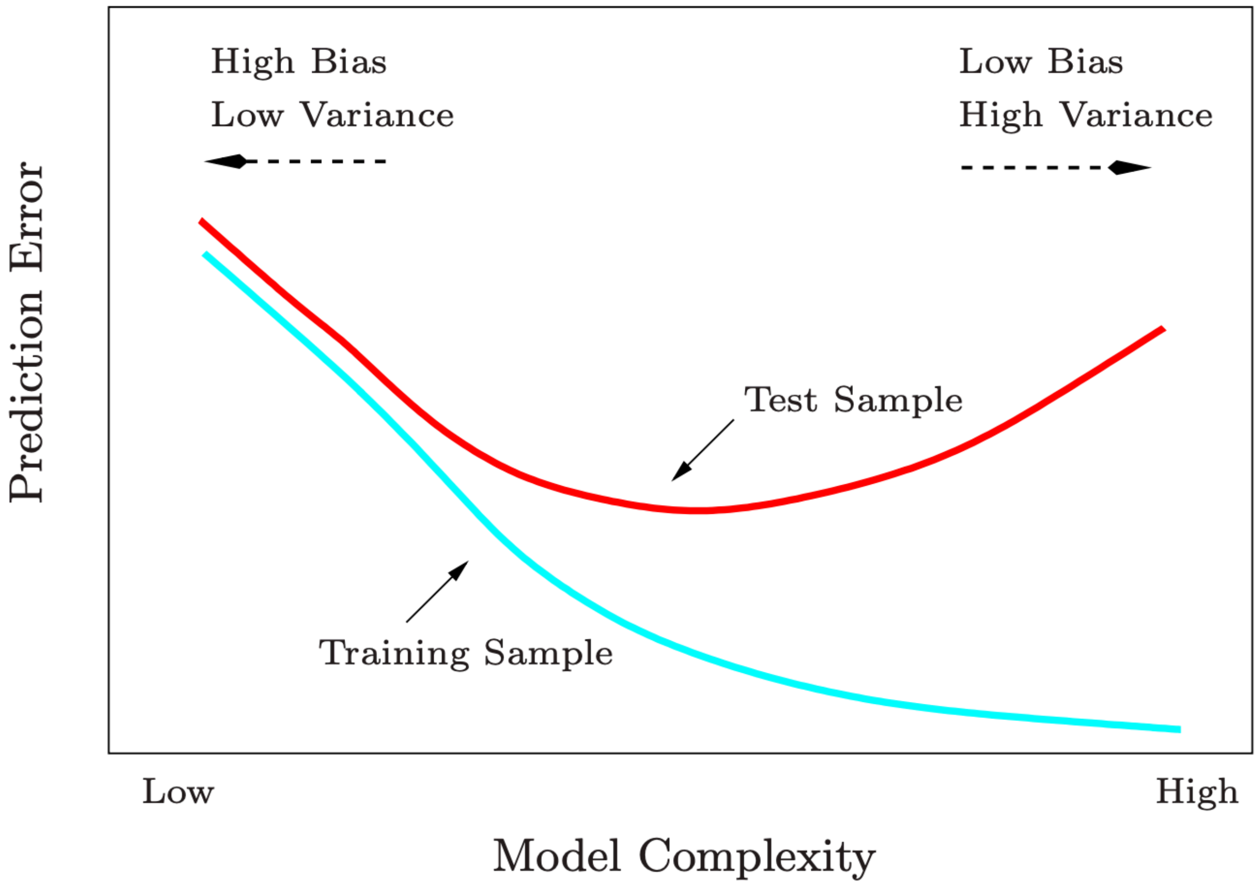


FIGURE 2.11. *Test and training error as a function of model complexity.*

Credit: Elements of Statistical Learning, Second edition

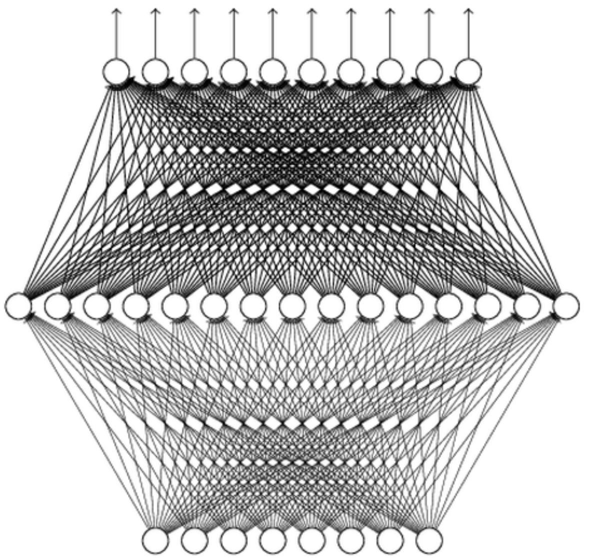
Recap: Universality Theorem

Any continuous function f

$$f : R^N \rightarrow R^M$$

Can be realized by a network with one hidden layer

(given **enough** hidden neurons)



Reference for the reason: http://neuralnetworksanddeeplearning.com/chap4.html

Recap: Universality is Not Enough

- Neural network has very high capacity (millions of parameters)

- By our basic knowledge of bias-variance tradeoff, so many parameters should imply very low bias, and very high variance. The test loss may not be small.

- Many efforts of deep learning are about mitigating overfitting!

Address Overfitting for NN

- Use larger training data set

- Design better network architecture

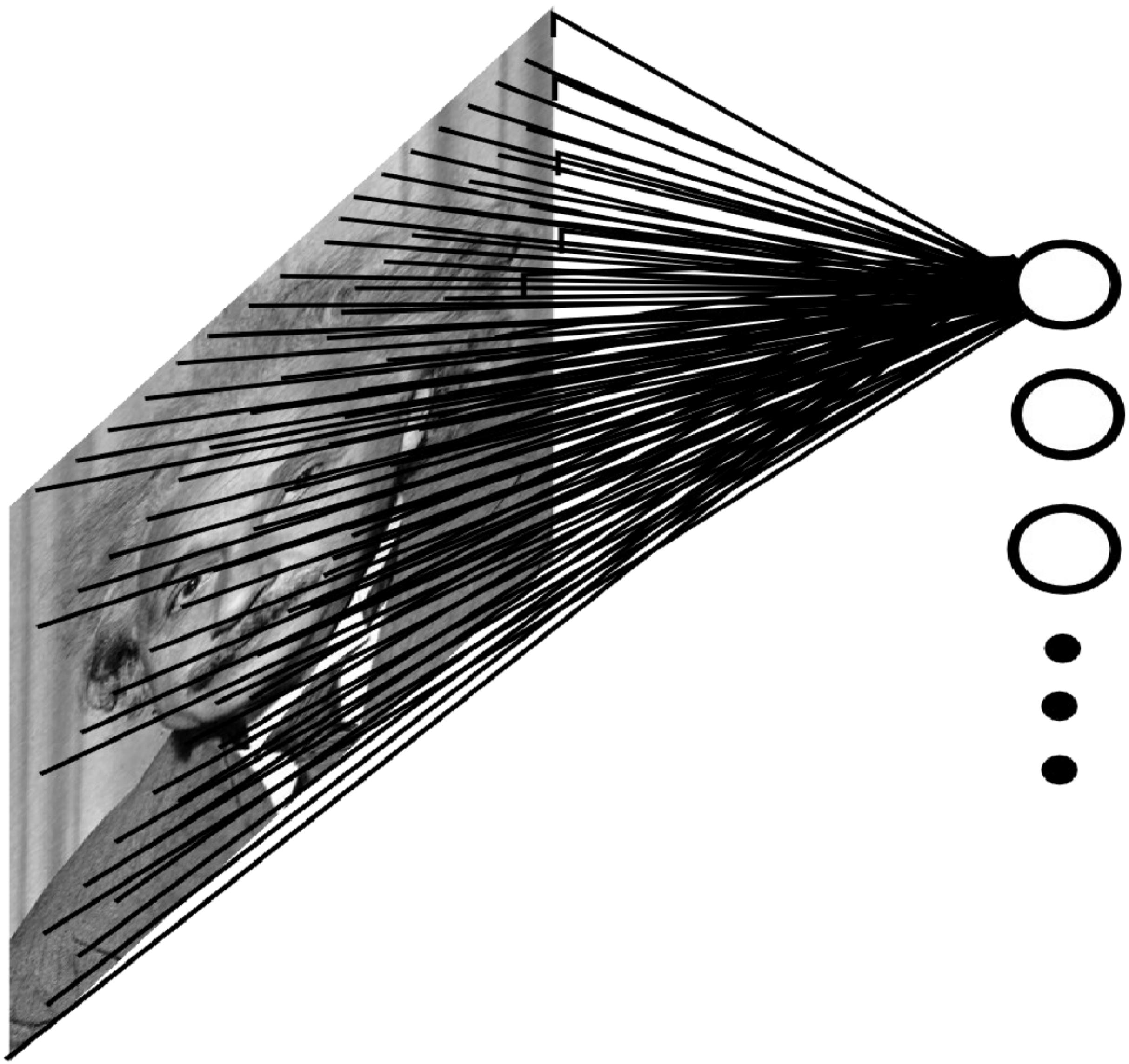
Address Overfitting for NN

- Use larger training data set

- **Design better network architecture**

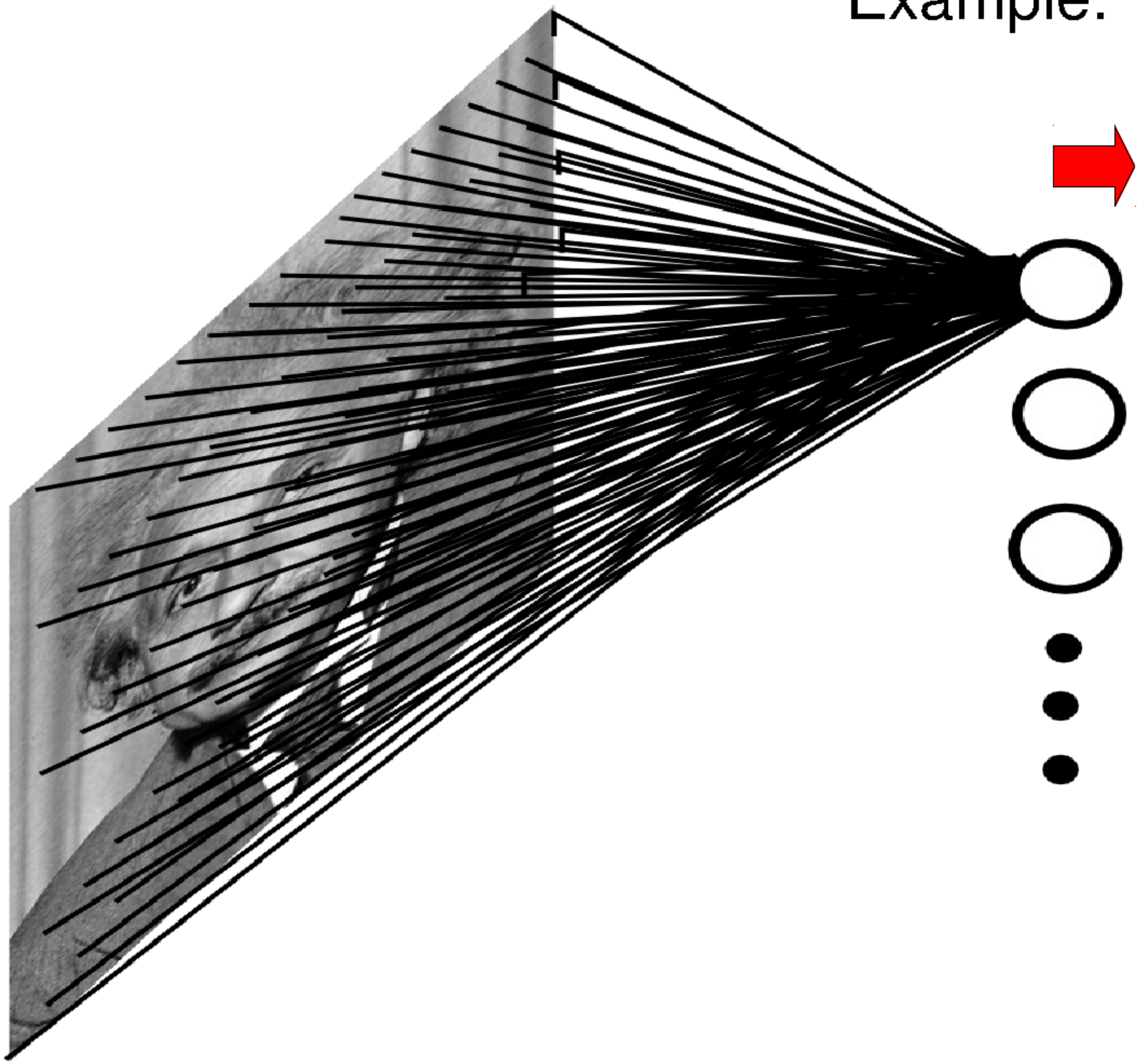
Convolutional Neural Network

Images as input to neural networks



Ranzato

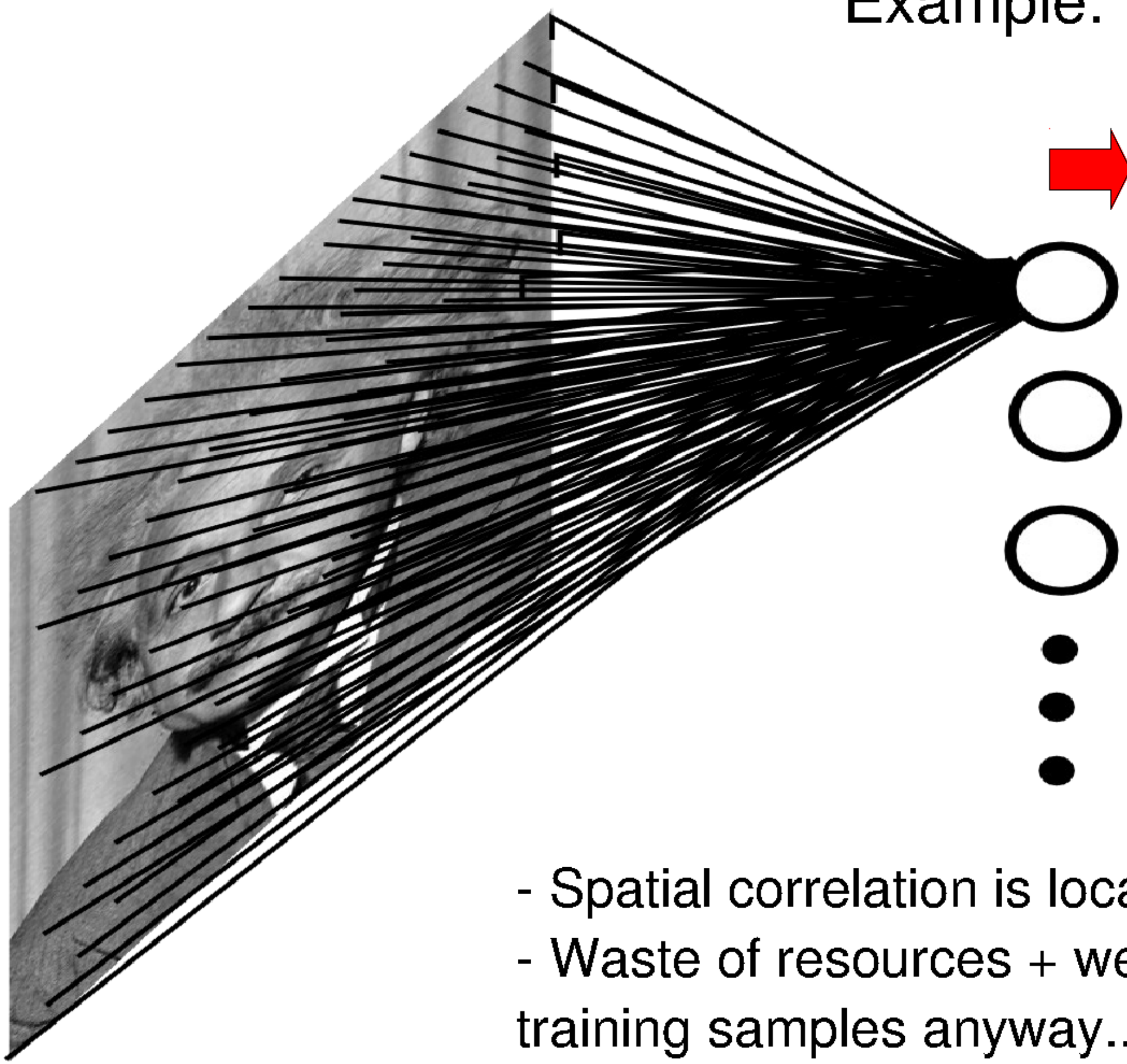
Images as input to neural networks



Example: 200x200 image
40K hidden units
~2B parameters!!!

Ranzato

Images as input to neural networks



Example: 200x200 image
40K hidden units
~2B parameters!!!

- Spatial correlation is local
- Waste of resources + we have not enough training samples anyway..

Ranzato

Convolutional Neural Networks

- CNN = a multi-layer neural network with
  - **Local** connectivity:
    - Neurons in a layer are only connected to a small region of the layer before it
  - **Share** weight parameters across spatial positions:
    - Learning shift-invariant filter kernels

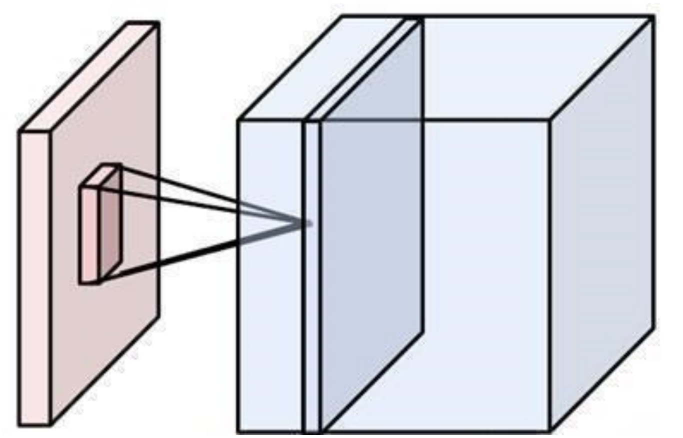
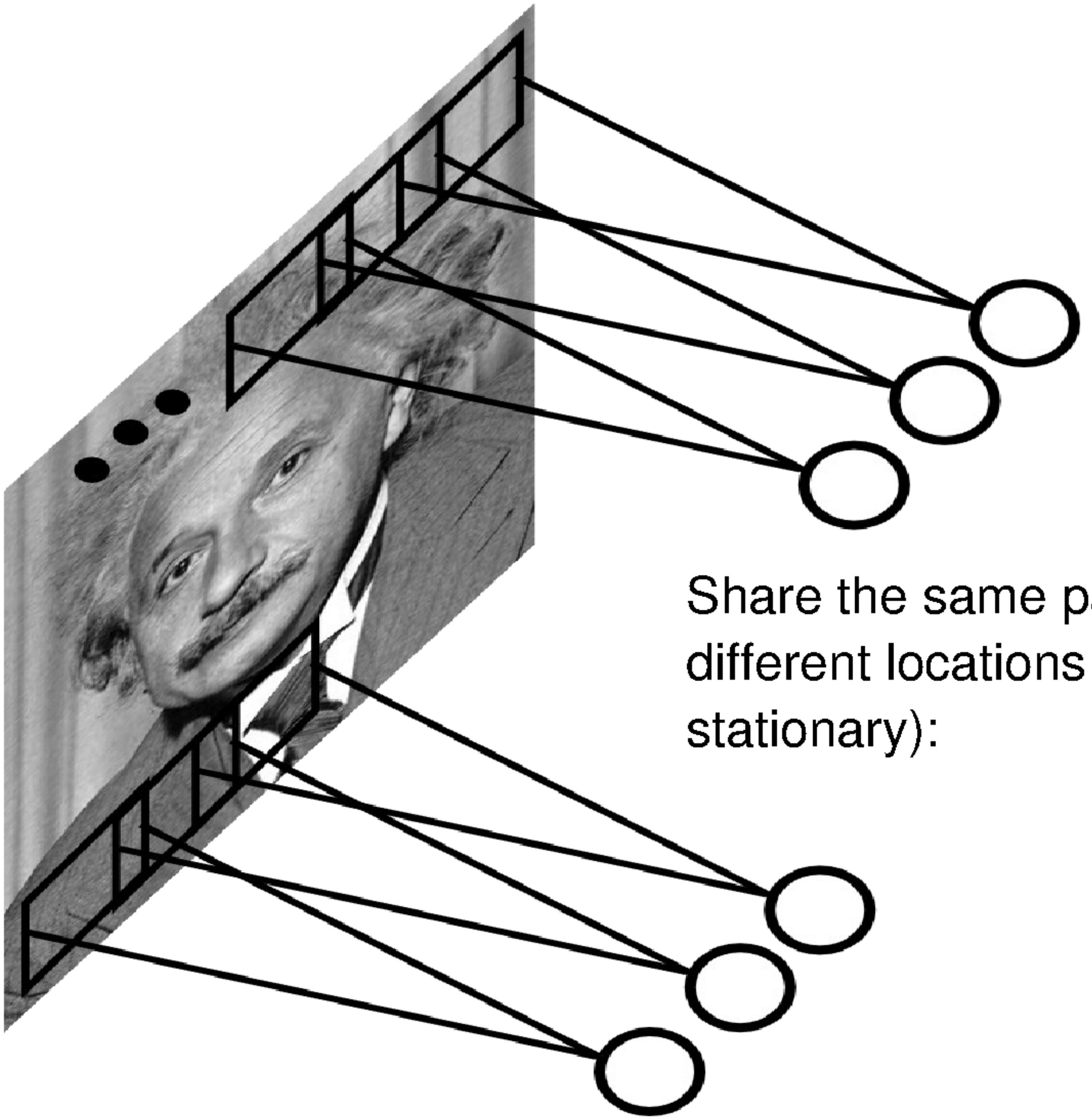
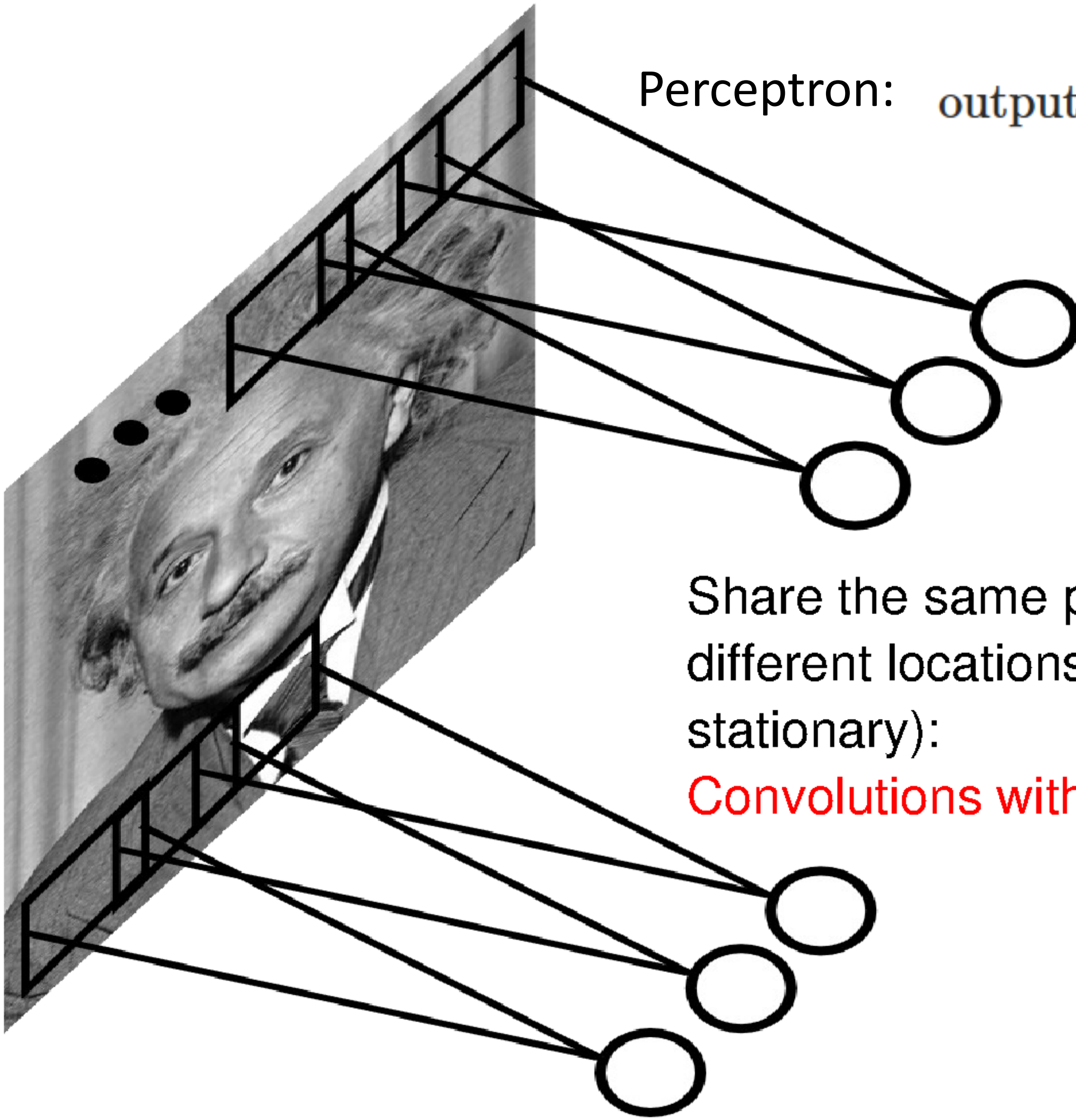


Image credit: A. Karpathy



Share the same parameters across different locations (assuming input is stationary):

Convolutional Layer



Perceptron: $\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$

$w \cdot x \equiv \sum_j w_j x_j$,

This is convolution!

Share the same parameters across different locations (assuming input is stationary):
Convolutions with learned kernels

Recap: Image filtering

$f[\cdot\,,\cdot\,] \quad \frac{1}{9}\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}$

$I[.,.]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

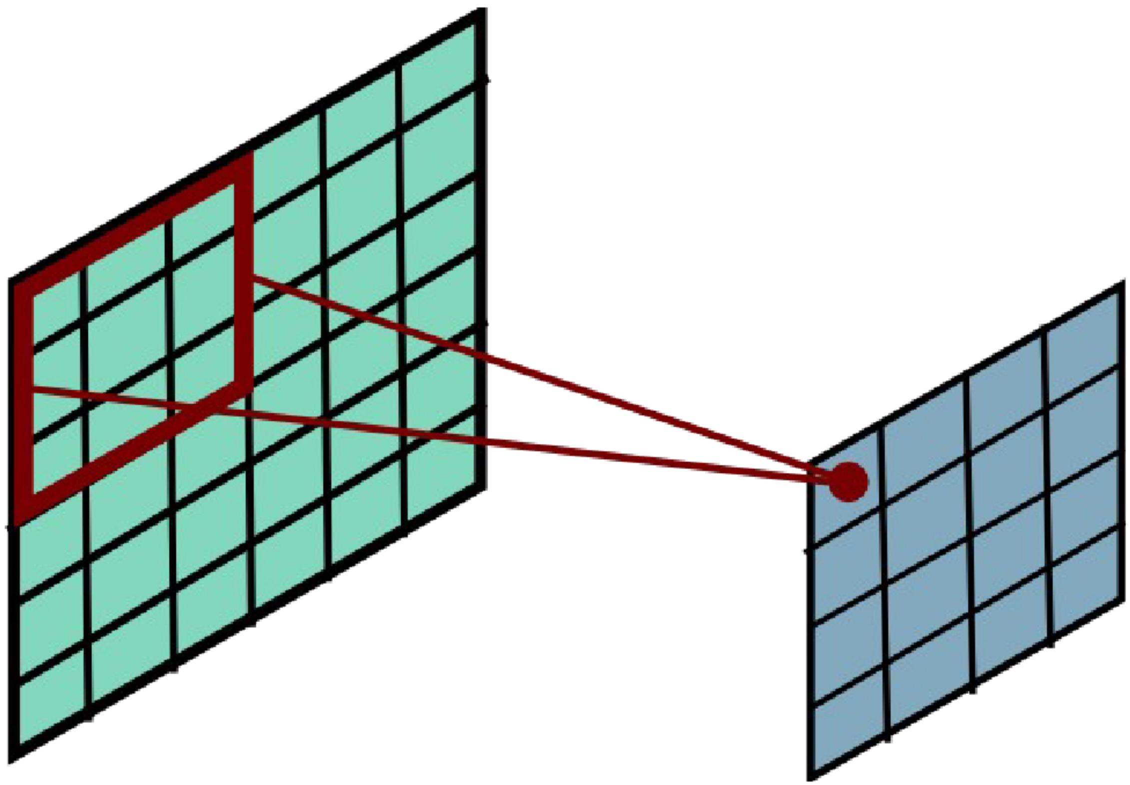
$h[.,.]$

	0	10	20	30	30	30	20	10	
	0	20	40	60	60	60	40	20	
	0	30	60	90	90	90	60	30	
	0	30	50	80	80	90	60	30	
	0	30	50	80	80	90	60	30	
	0	20	30	50	50	60	40	20	
	10	20	30	30	30	30	20	10	
	10	10	10	0	0	0	0	0	

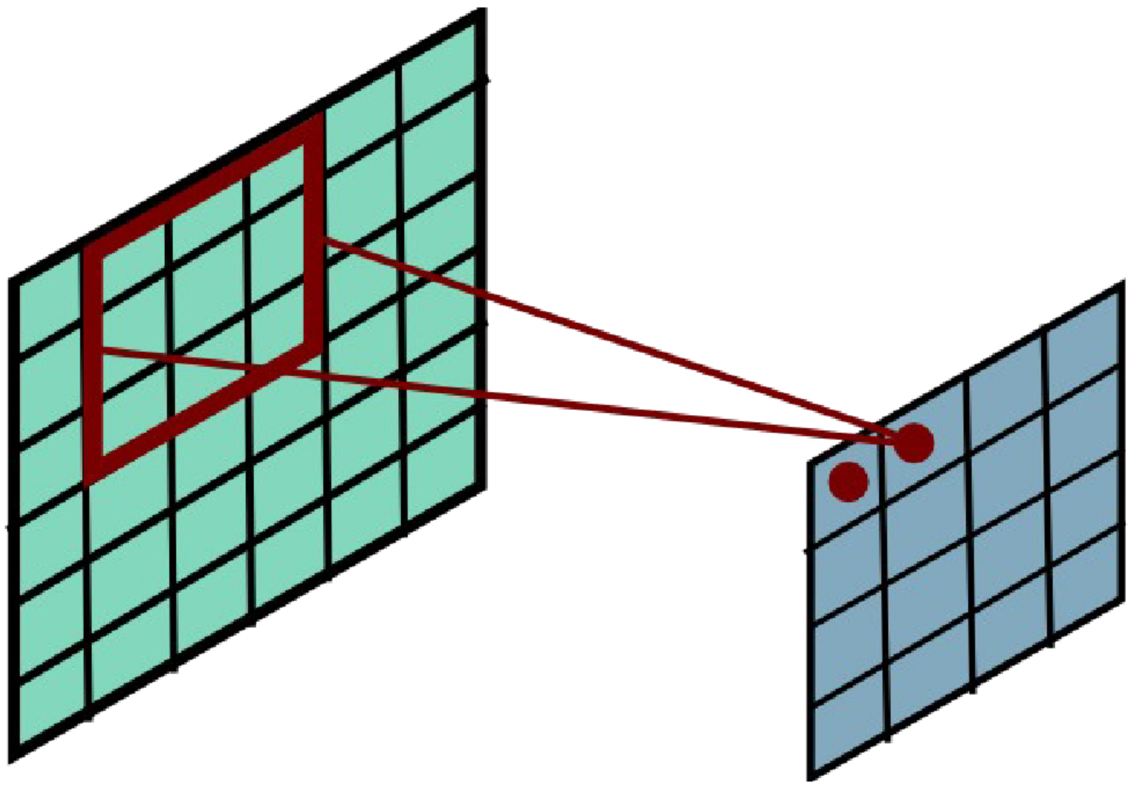
$$h[m,n] = \sum_{k,l} f[k,l]\, I[m+k, n+l]$$

Credit: S. Seitz

Convolutional Layer

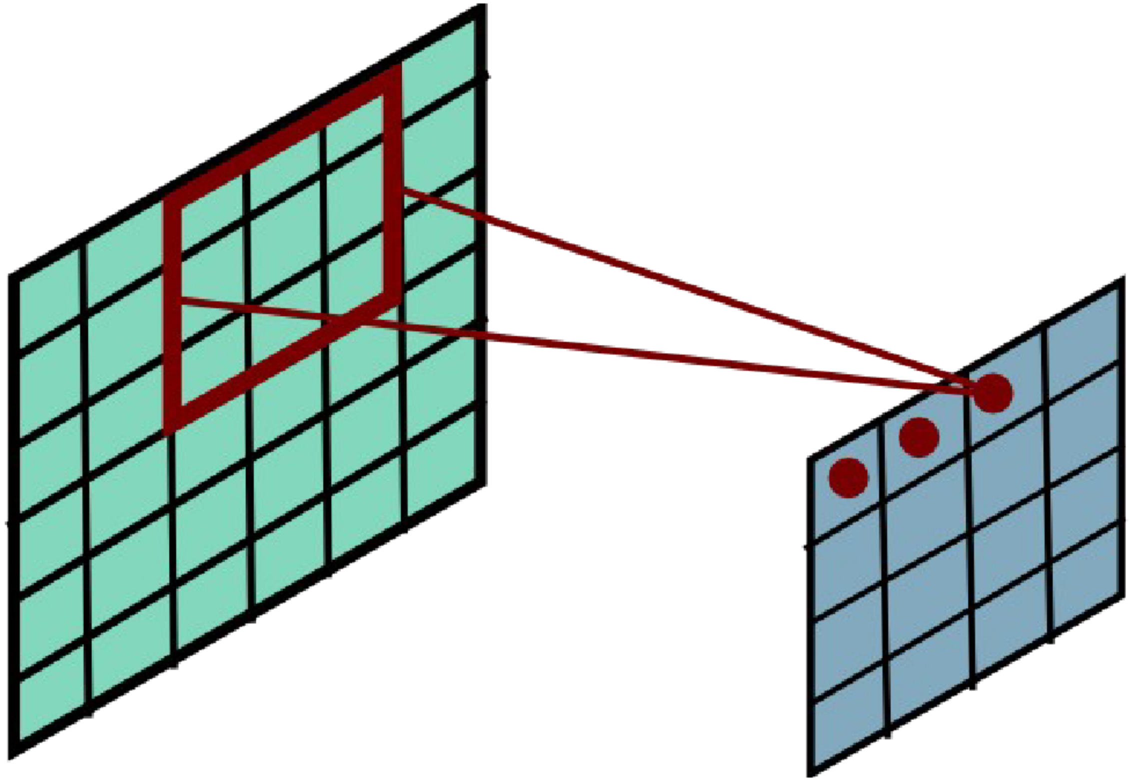


Convolutional Layer



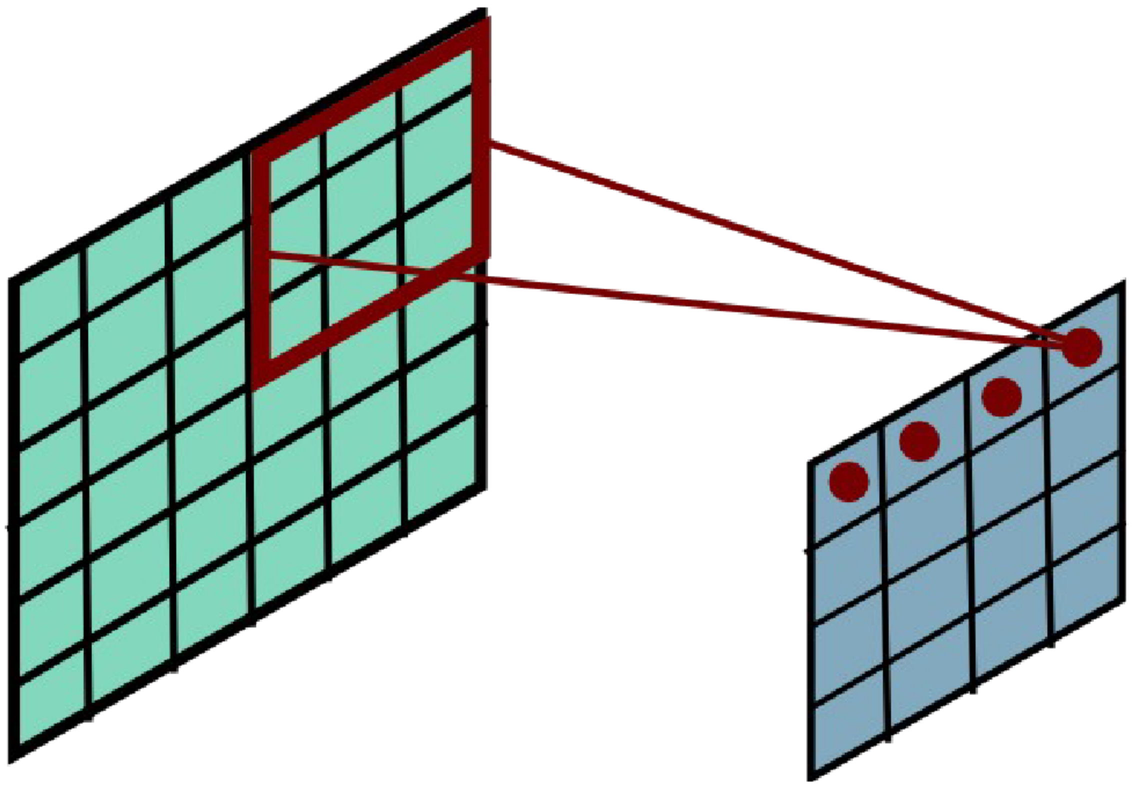
Ranzato

Convolutional Layer



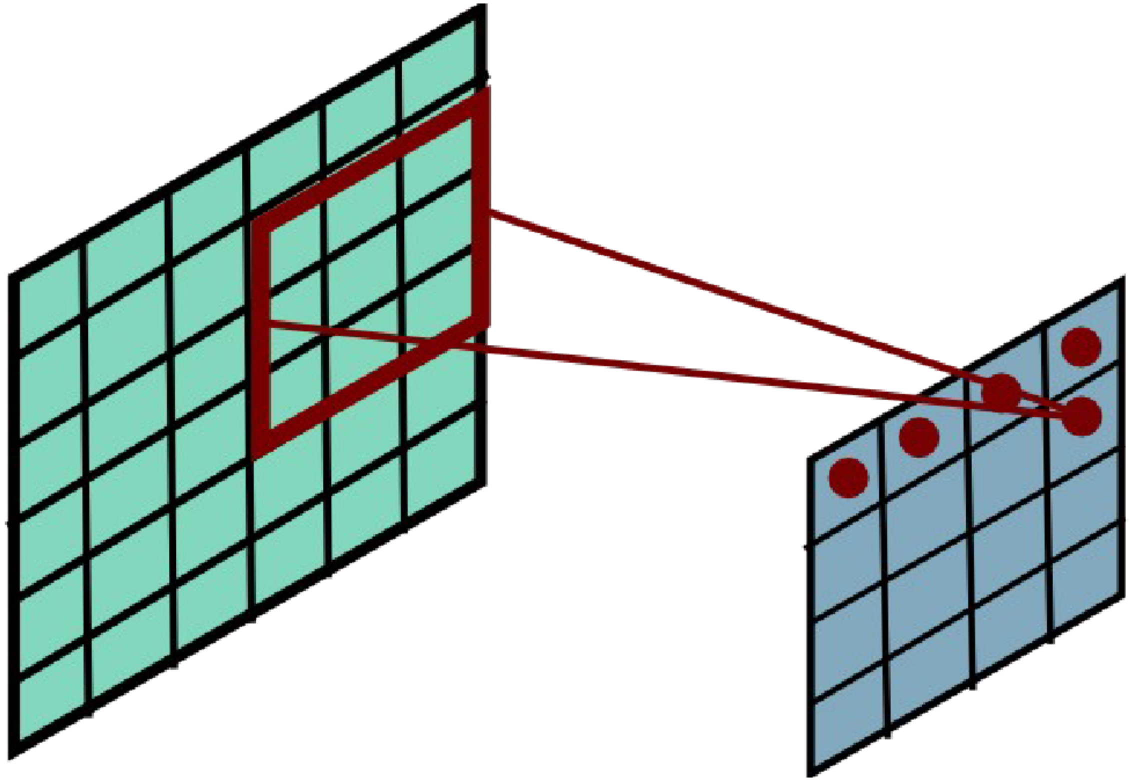
Ranzato

Convolutional Layer



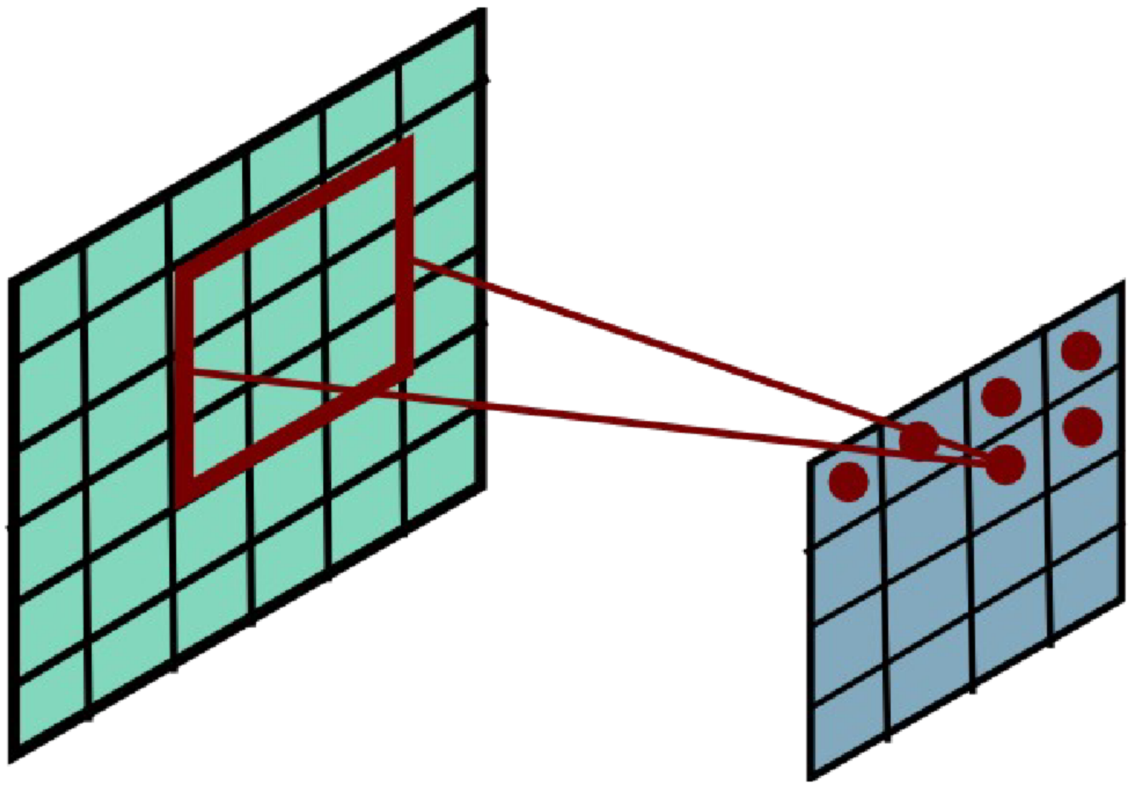
Ranzato

Convolutional Layer



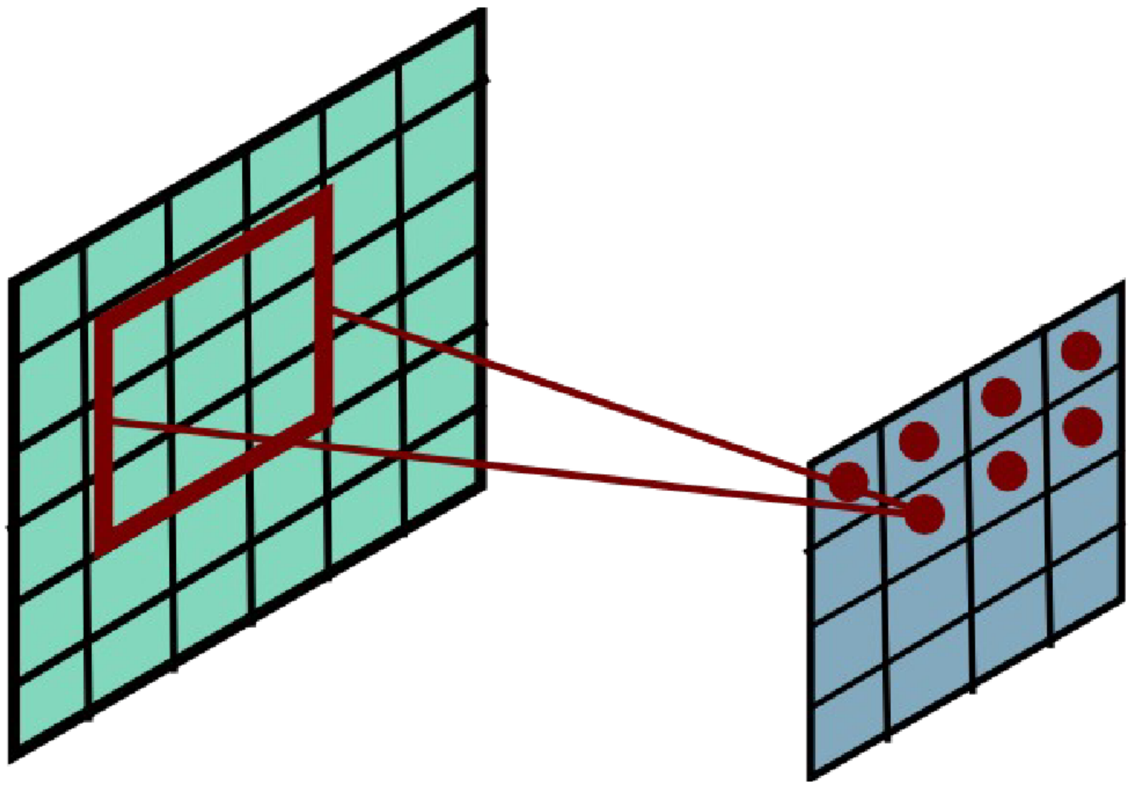
Ranzato

Convolutional Layer

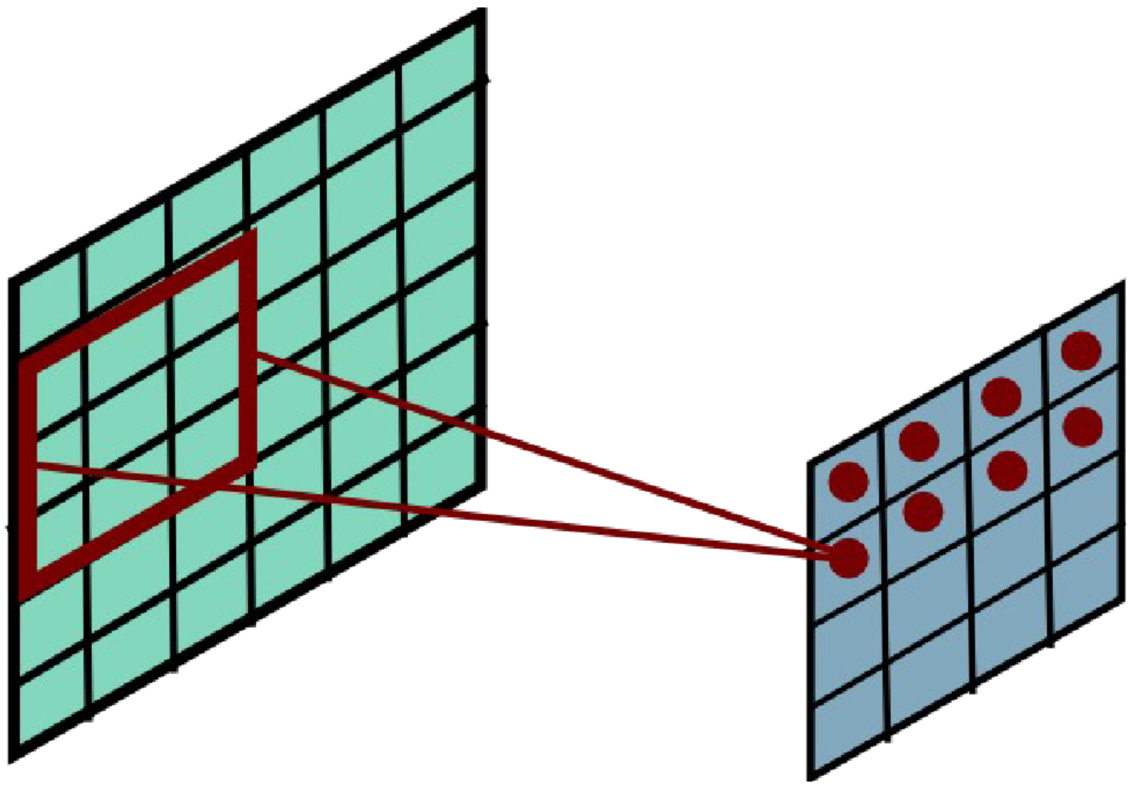


Ranzato

Convolutional Layer

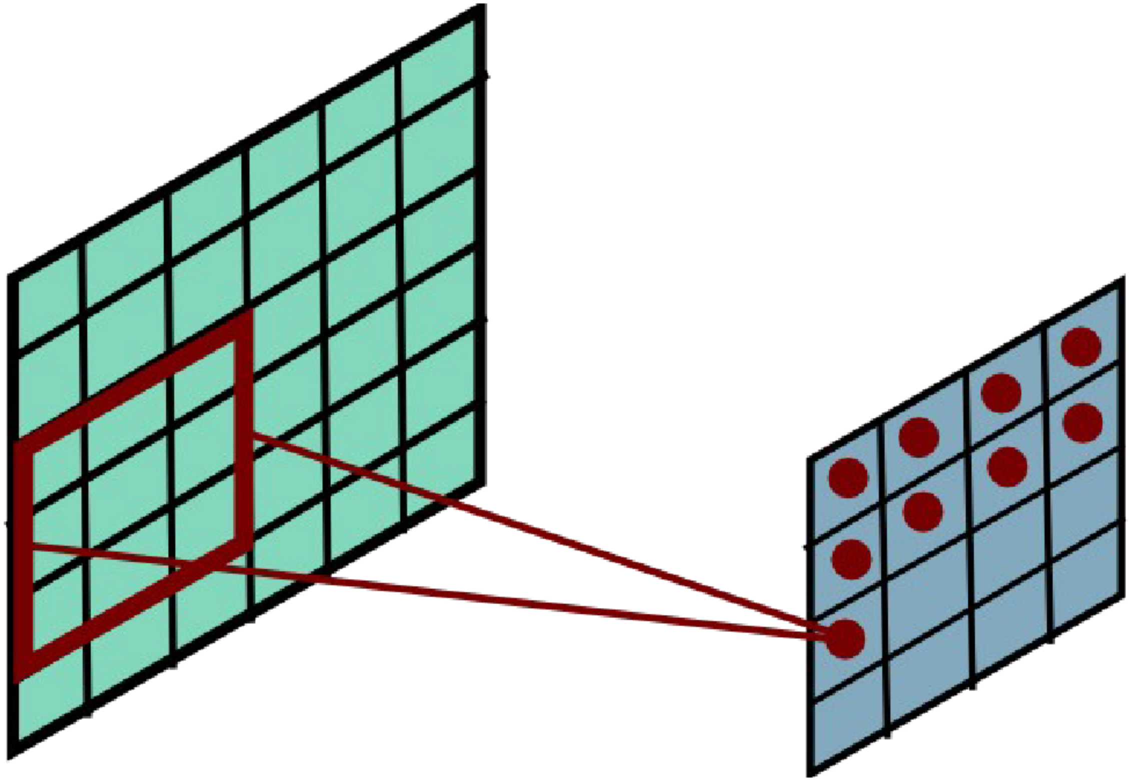


Convolutional Layer



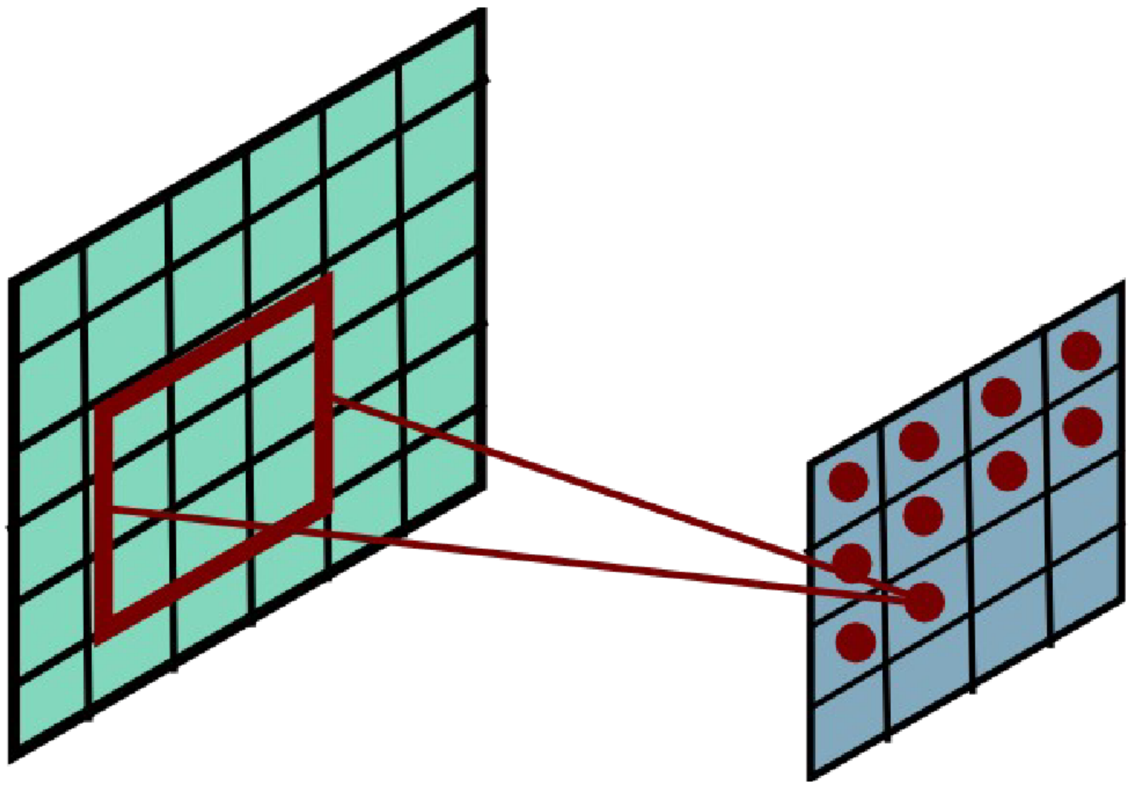
Ranzato

Convolutional Layer



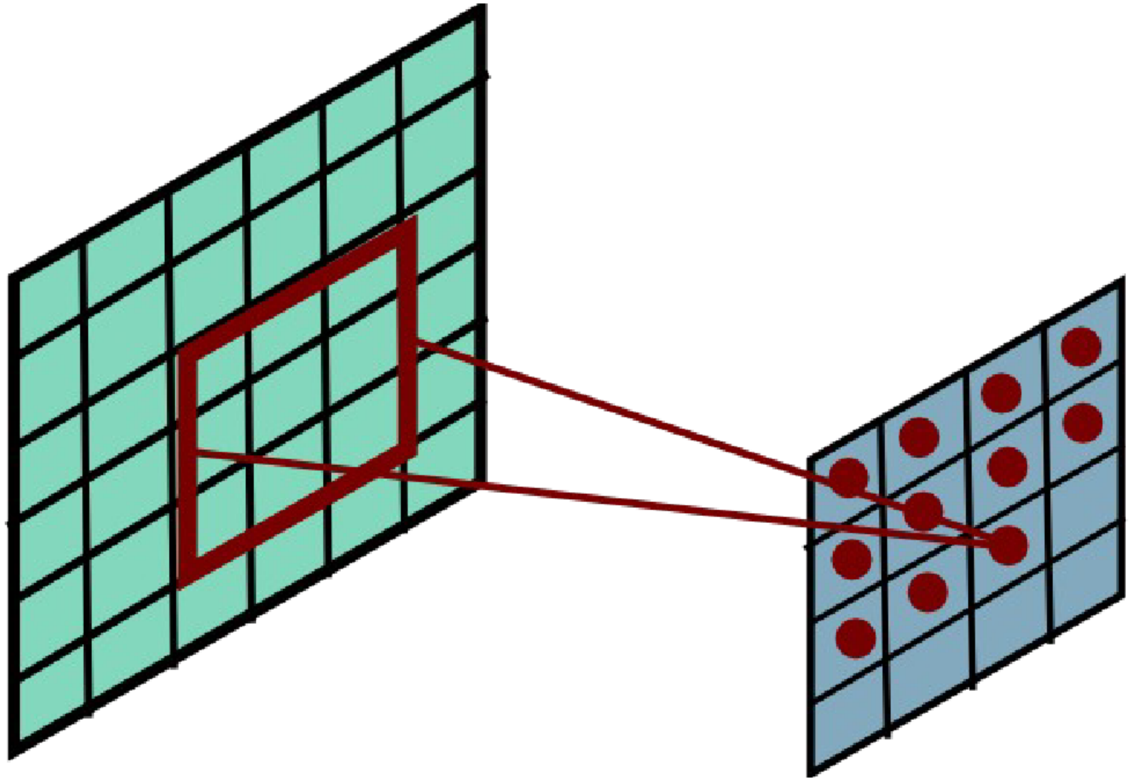
Ranzato

Convolutional Layer



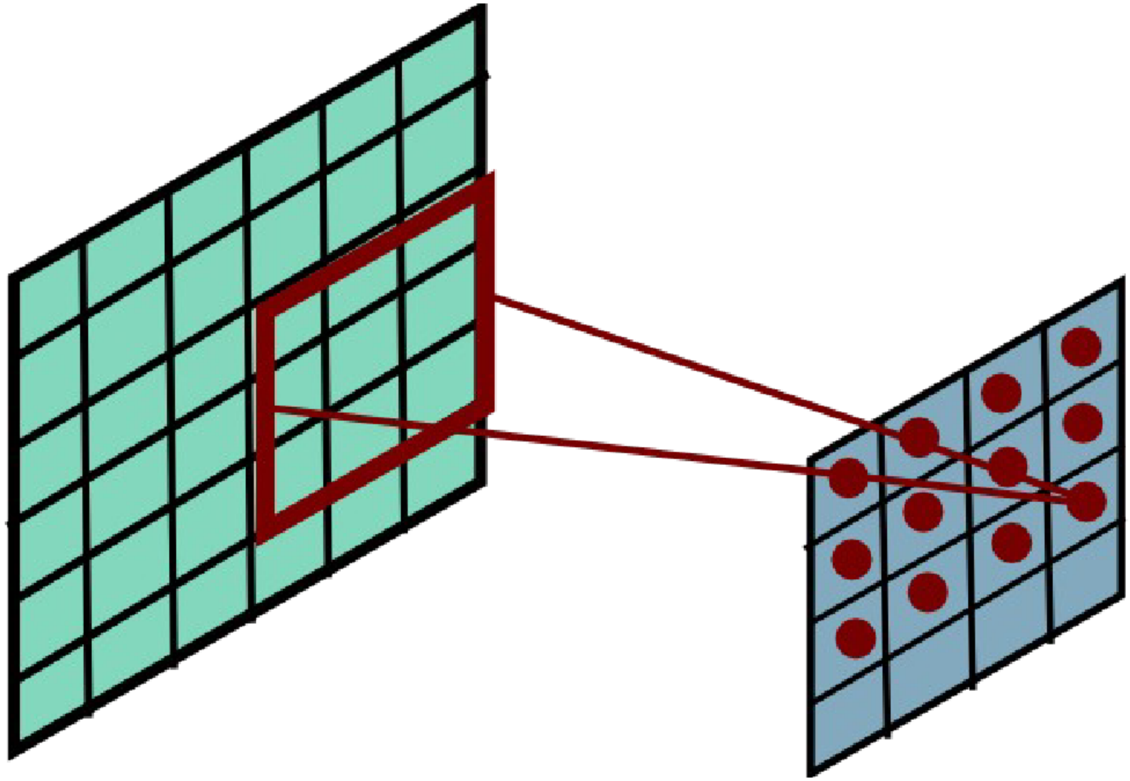
Ranzato

Convolutional Layer



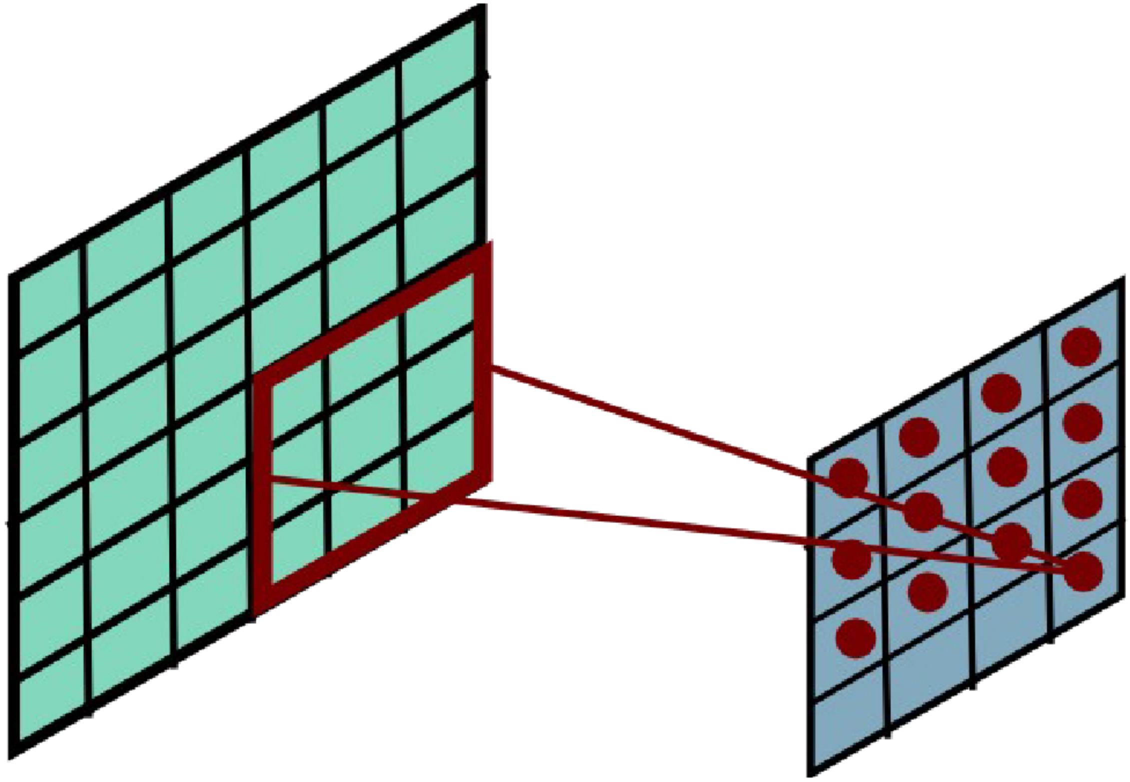
Ranzato

Convolutional Layer

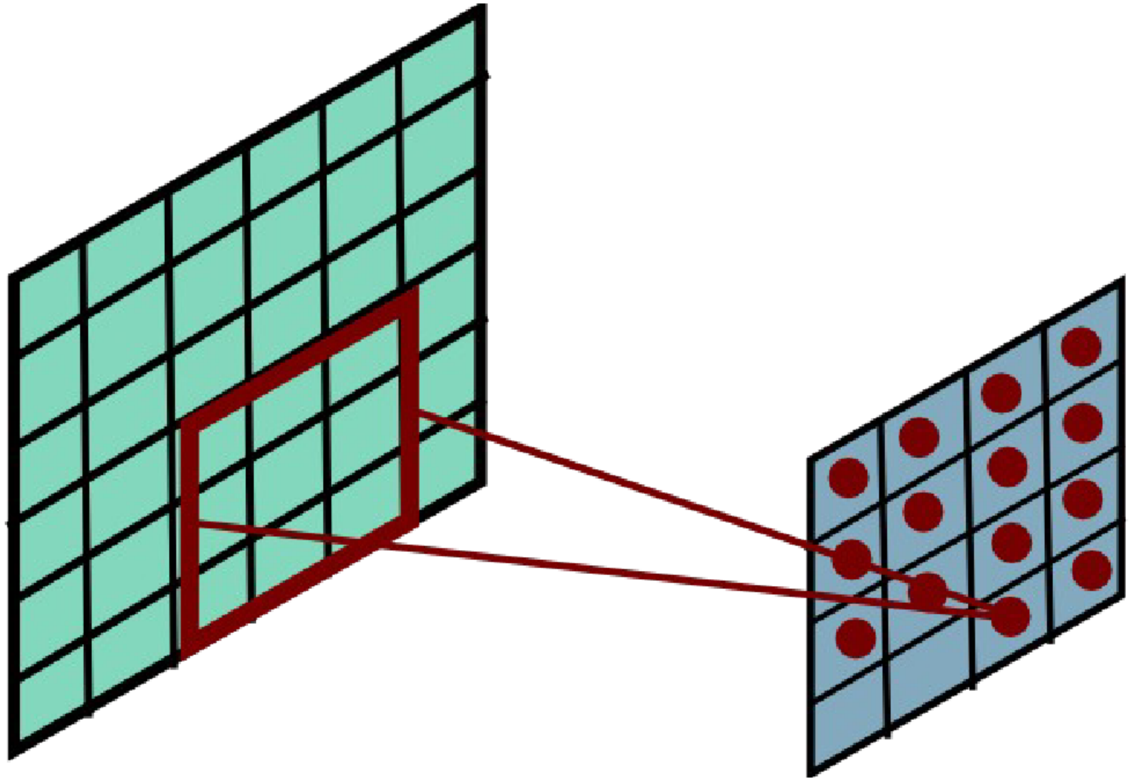


Ranzato

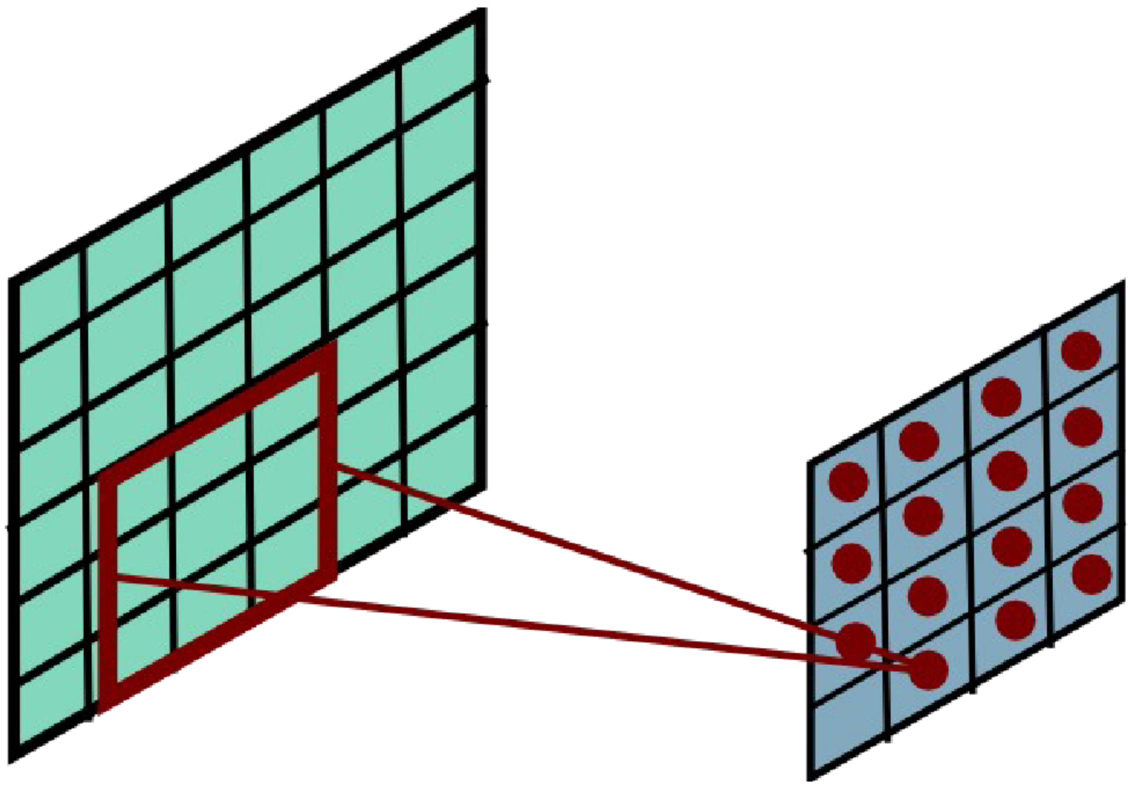
Convolutional Layer



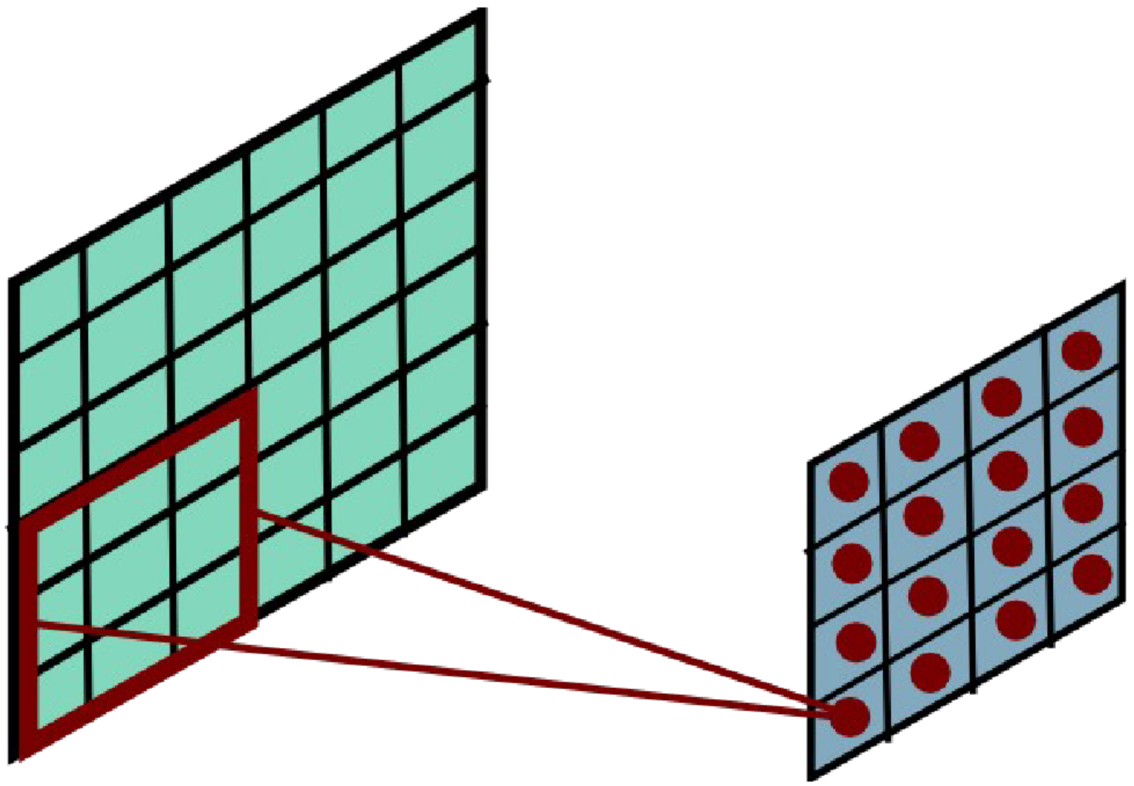
Convolutional Layer



Convolutional Layer

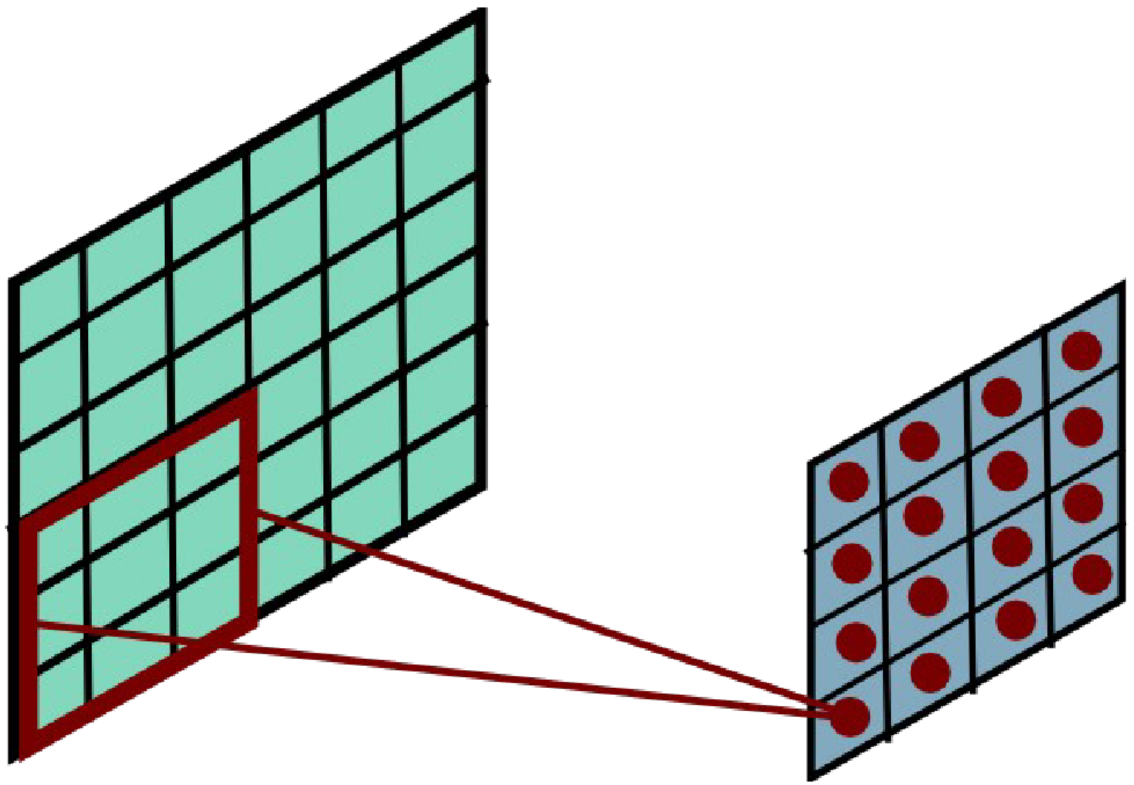


Convolutional Layer

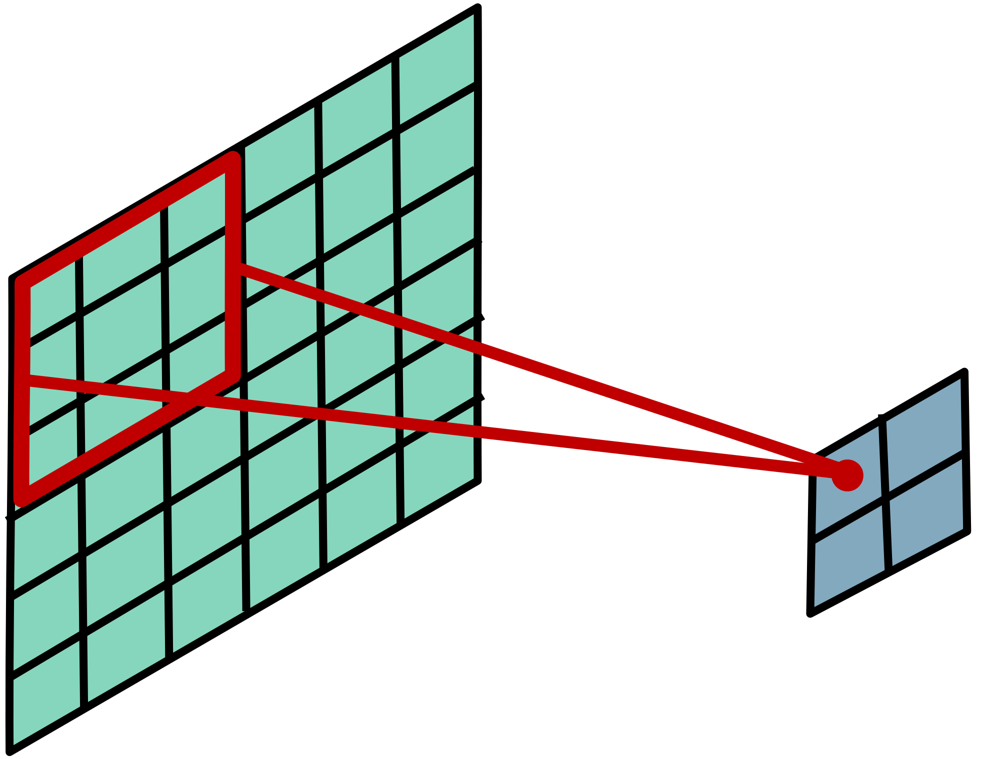


Ranzato

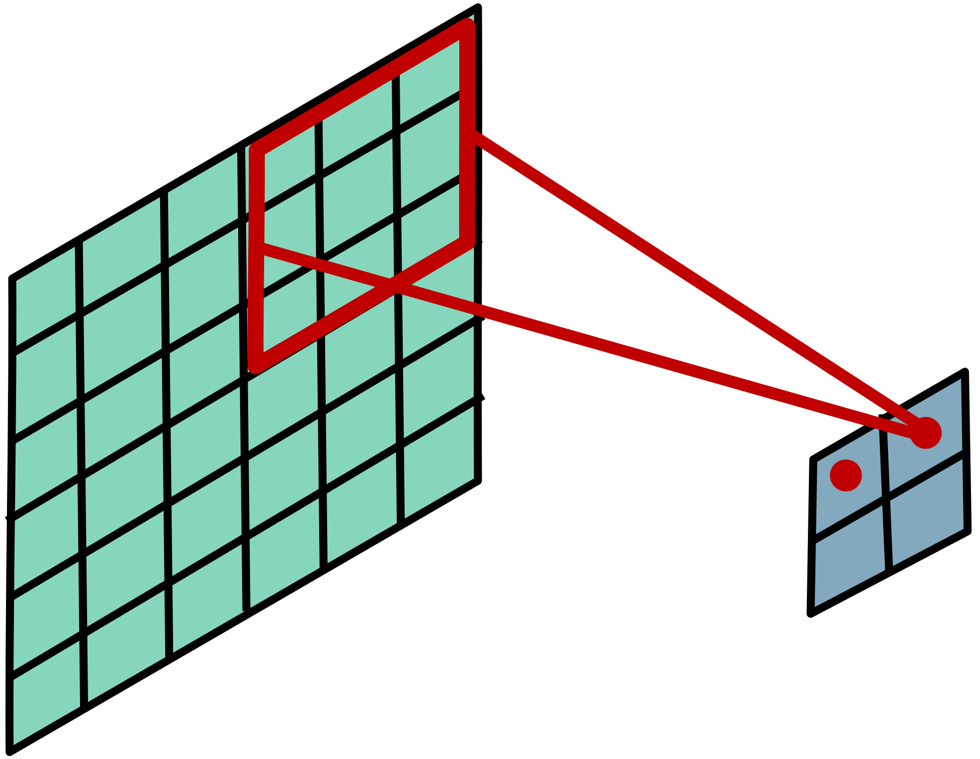
Stride = 3



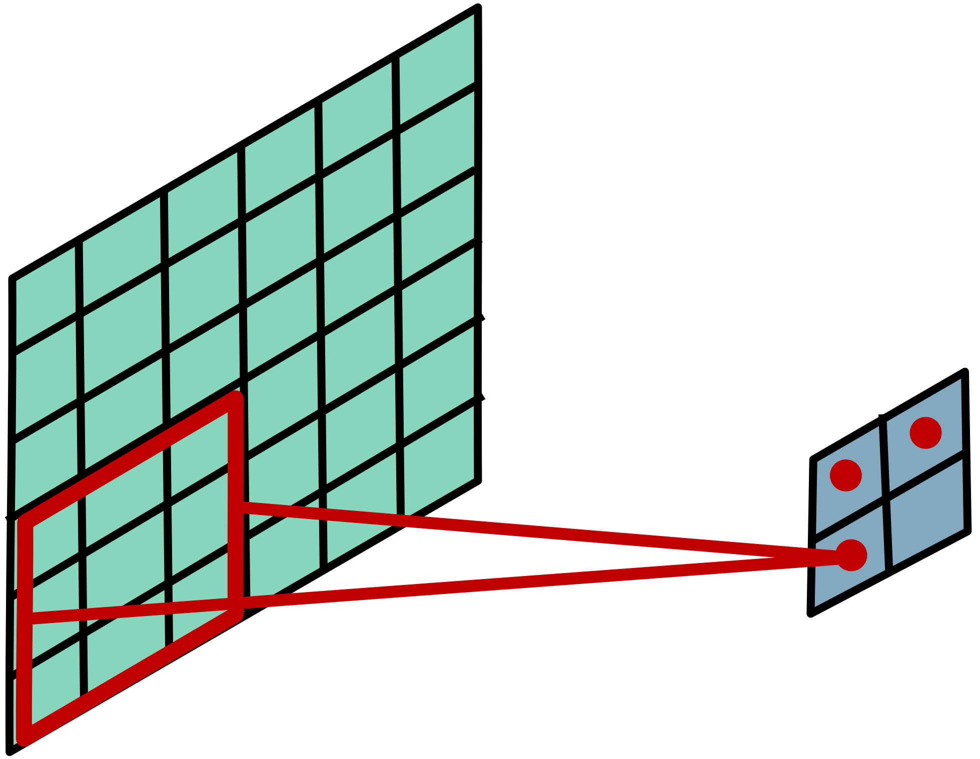
Stride = 3



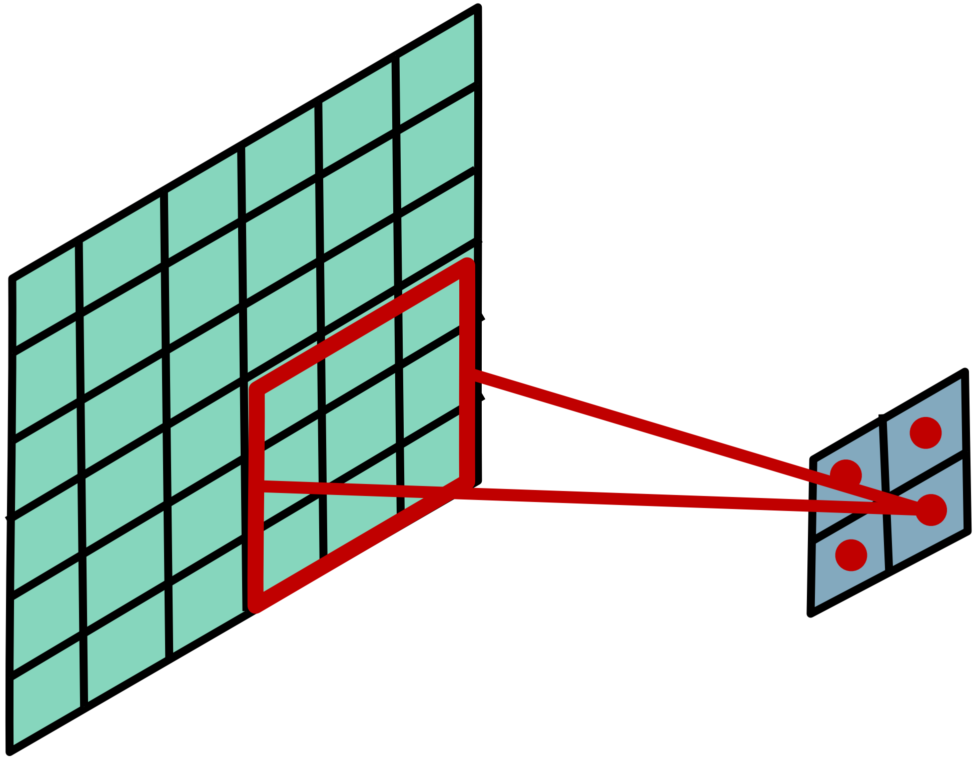
Stride = 3



Stride = 3

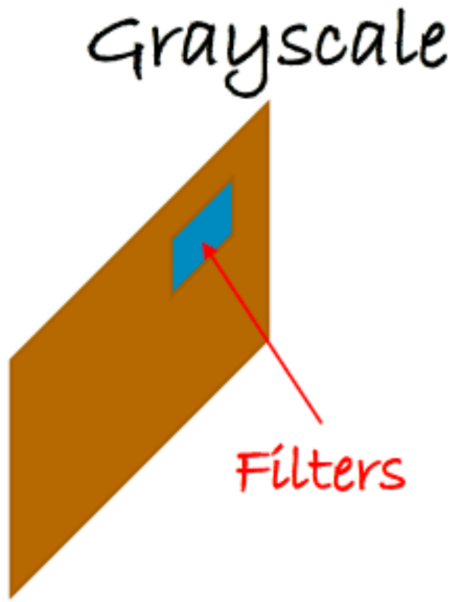


Stride = 3



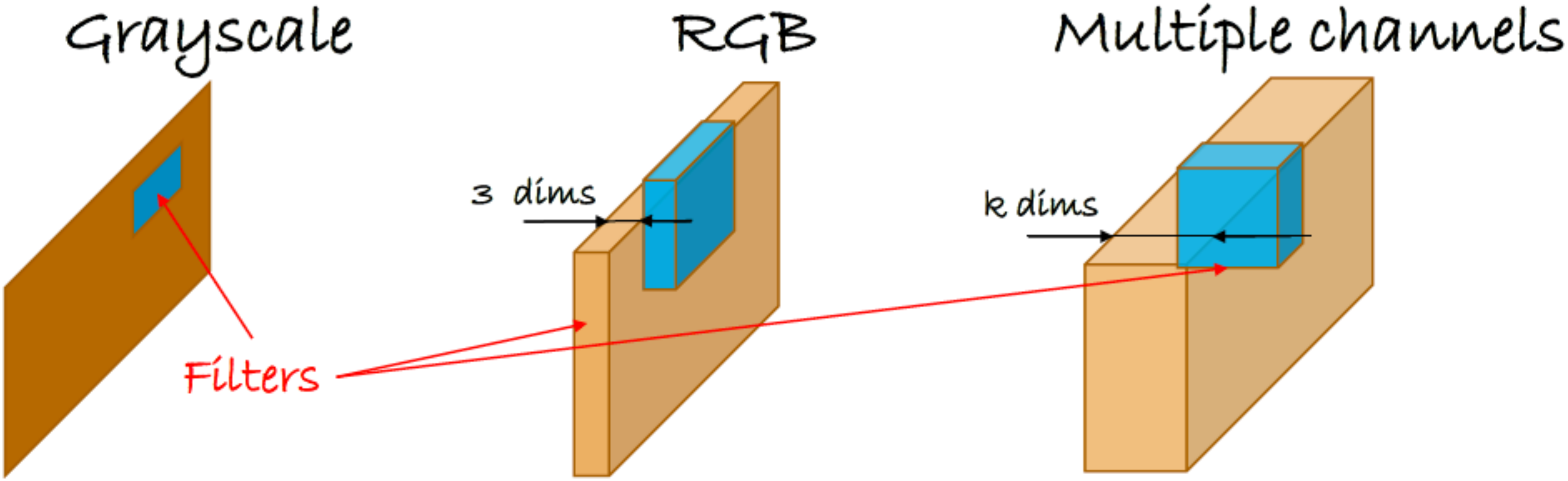
2D spatial filters

- If images are 2-D, parameters should also be organized in 2-D
  - That way they can learn the local correlations between input variables
  - That way they can "exploit" the spatial nature of images

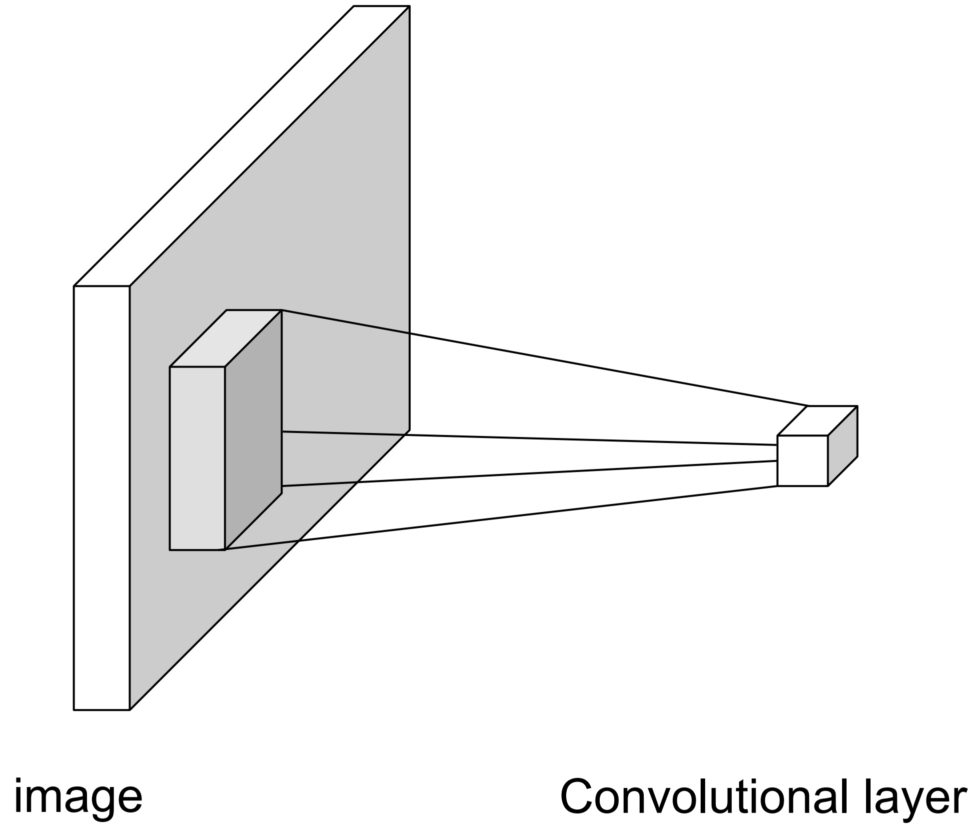


k-D spatial filters

- Similarly, if images are k-D, parameters should also be k-D

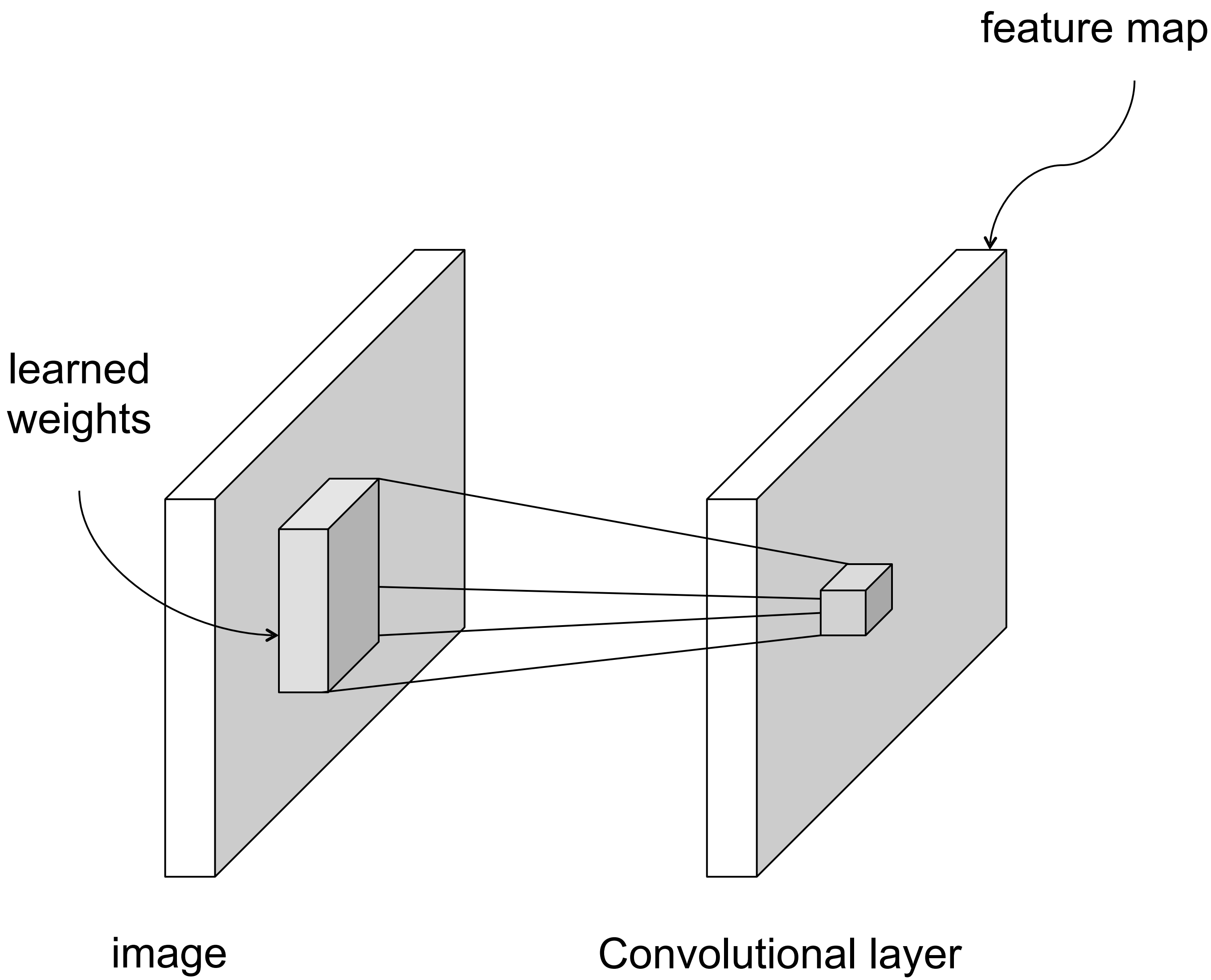


Dimensions of convolution



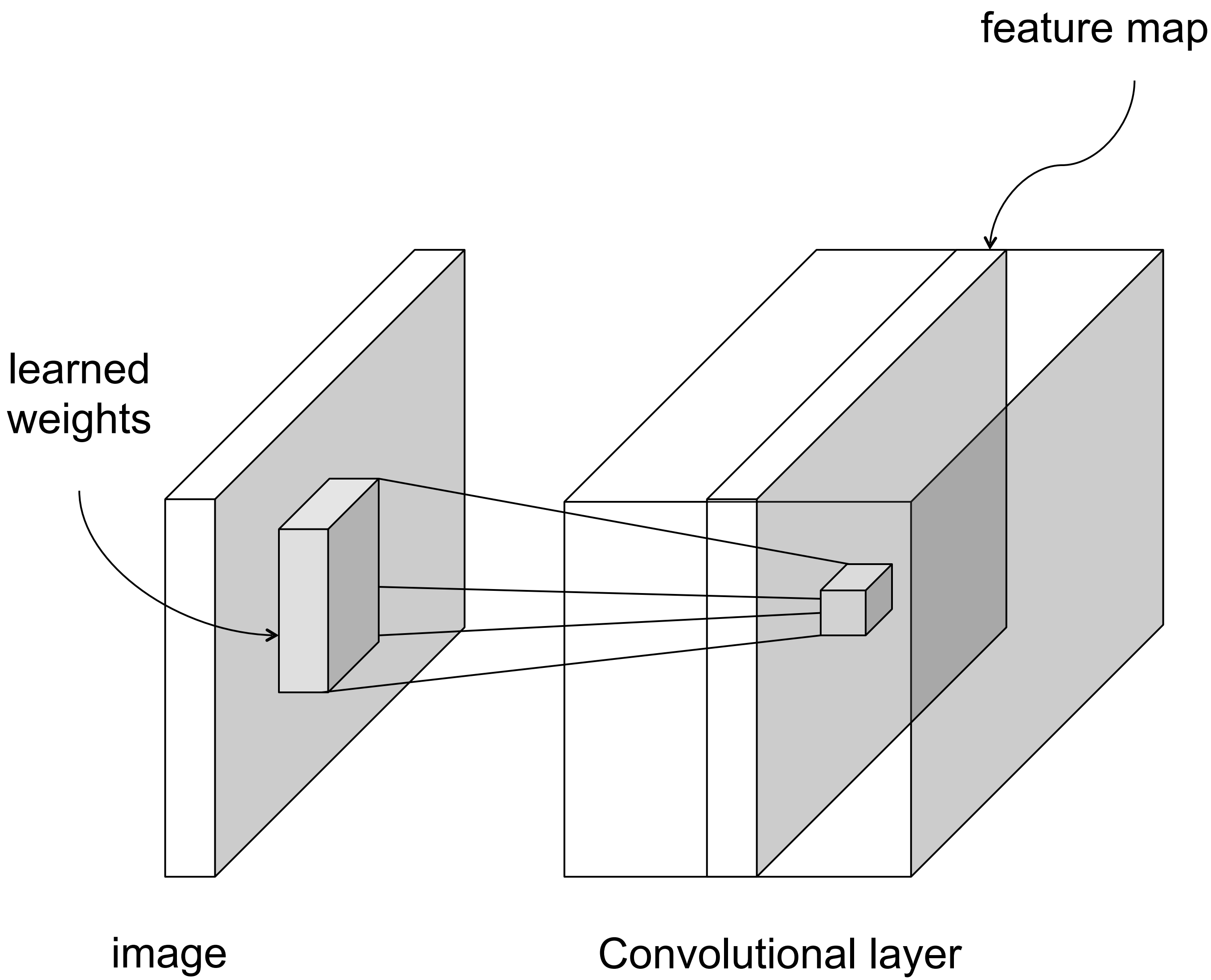
Slide: Lazebnik

Dimensions of convolution

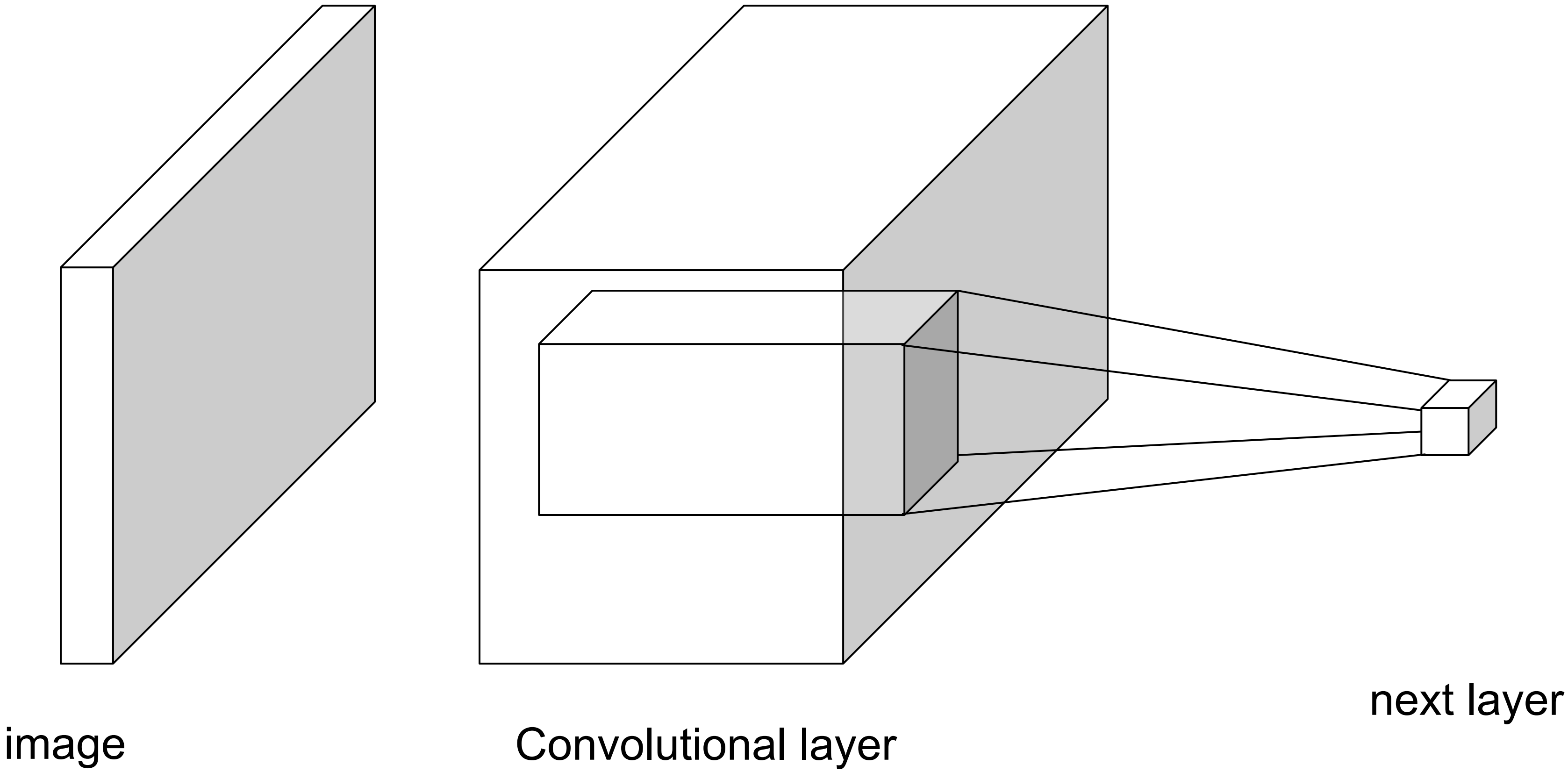


Slide: Lazebnik

Dimensions of convolution

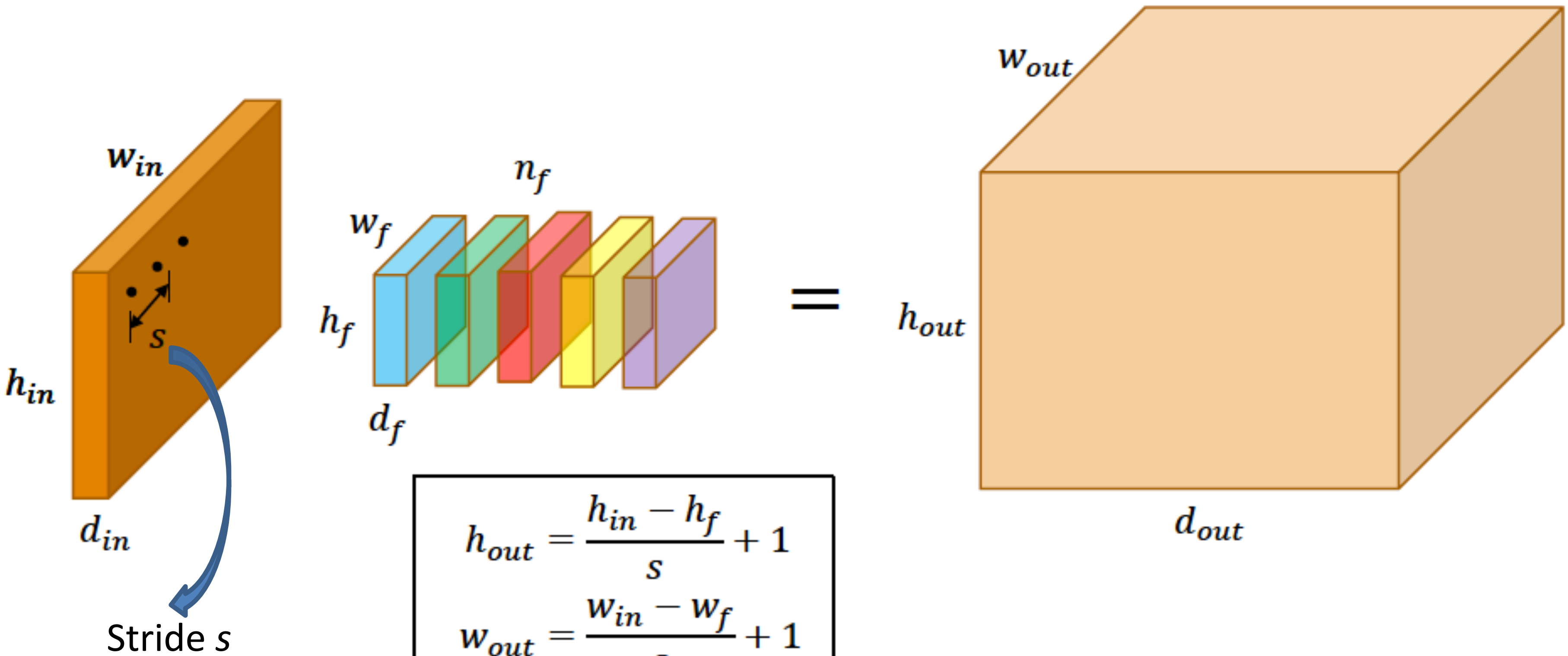


Dimensions of convolution



Slide: Lazebnik

Dimensions of convolution



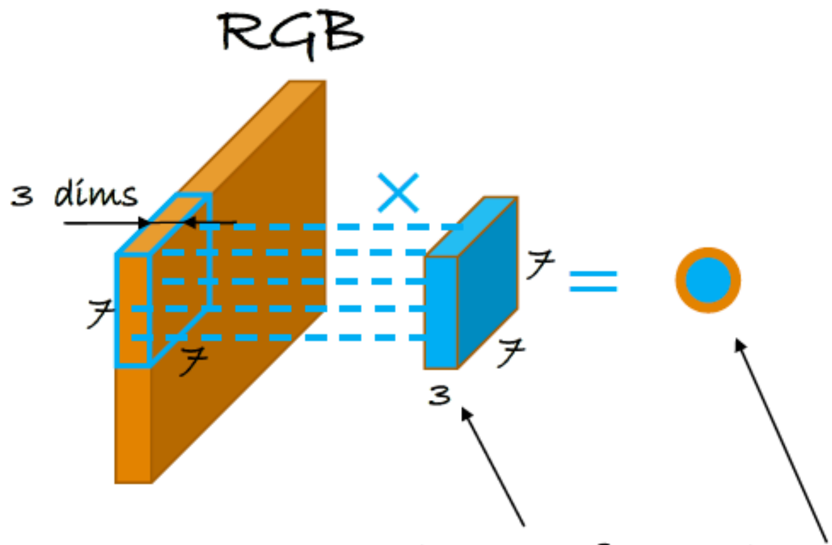
$$h_{out} = \frac{h_{in} - h_f}{s} + 1$$

$$w_{out} = \frac{w_{in} - w_f}{s} + 1$$

$$d_{out} = n_f$$

$$d_f = d_{in}$$

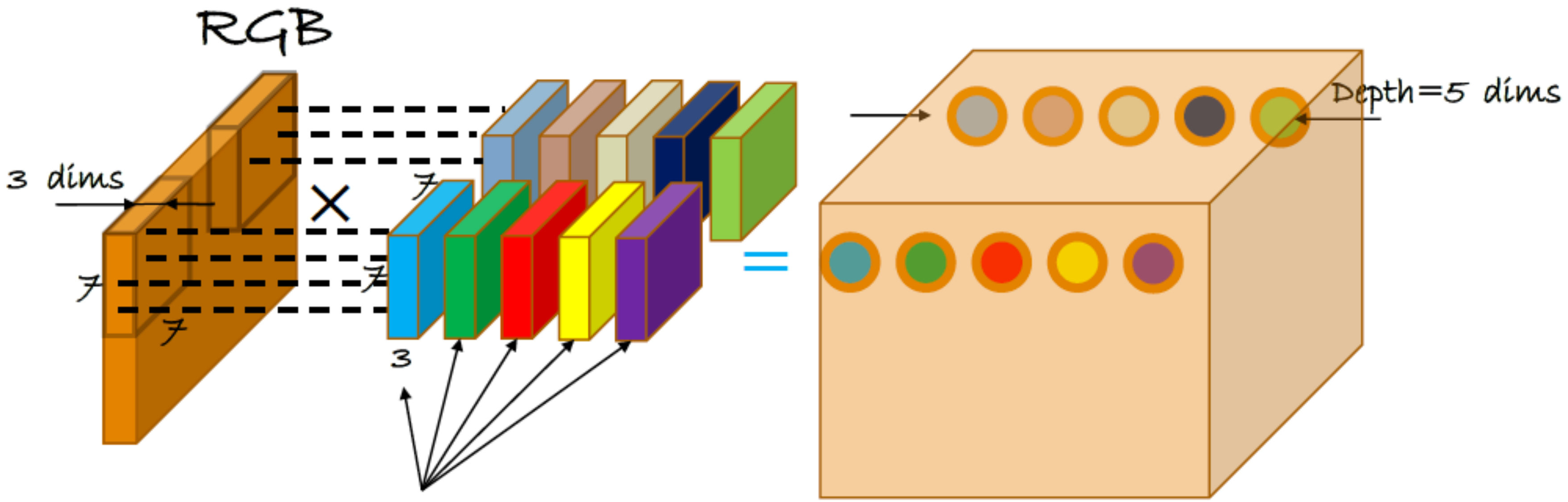
Number of weights



How many weights for this neuron?

$$7 \cdot 7 \cdot 3 = 147$$

Number of weights



How many weights for these 5 neurons?

$5 \cdot 7 \cdot 7 \cdot 3 = 735$

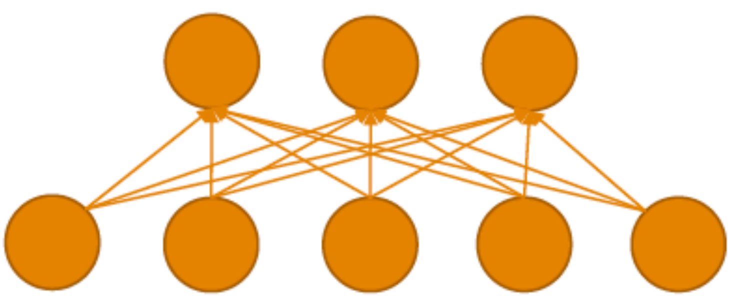
Convolutional Neural Networks

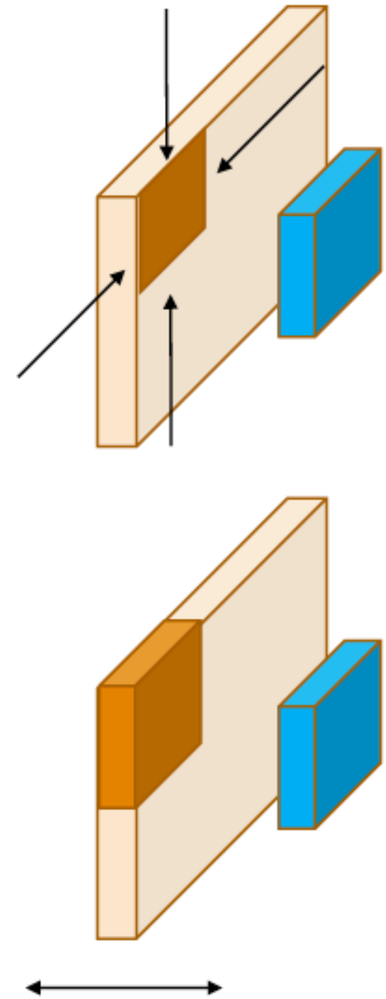
- Question: Spatial structure?
  - Answer: Convolutional filters
- Question: Huge input dimensionalities?
  - Answer: Parameters are shared between filters
- Question: Local variances?
  - Answer: Pooling

[Slides credit: Efstratios Gavves]

Local connectivity

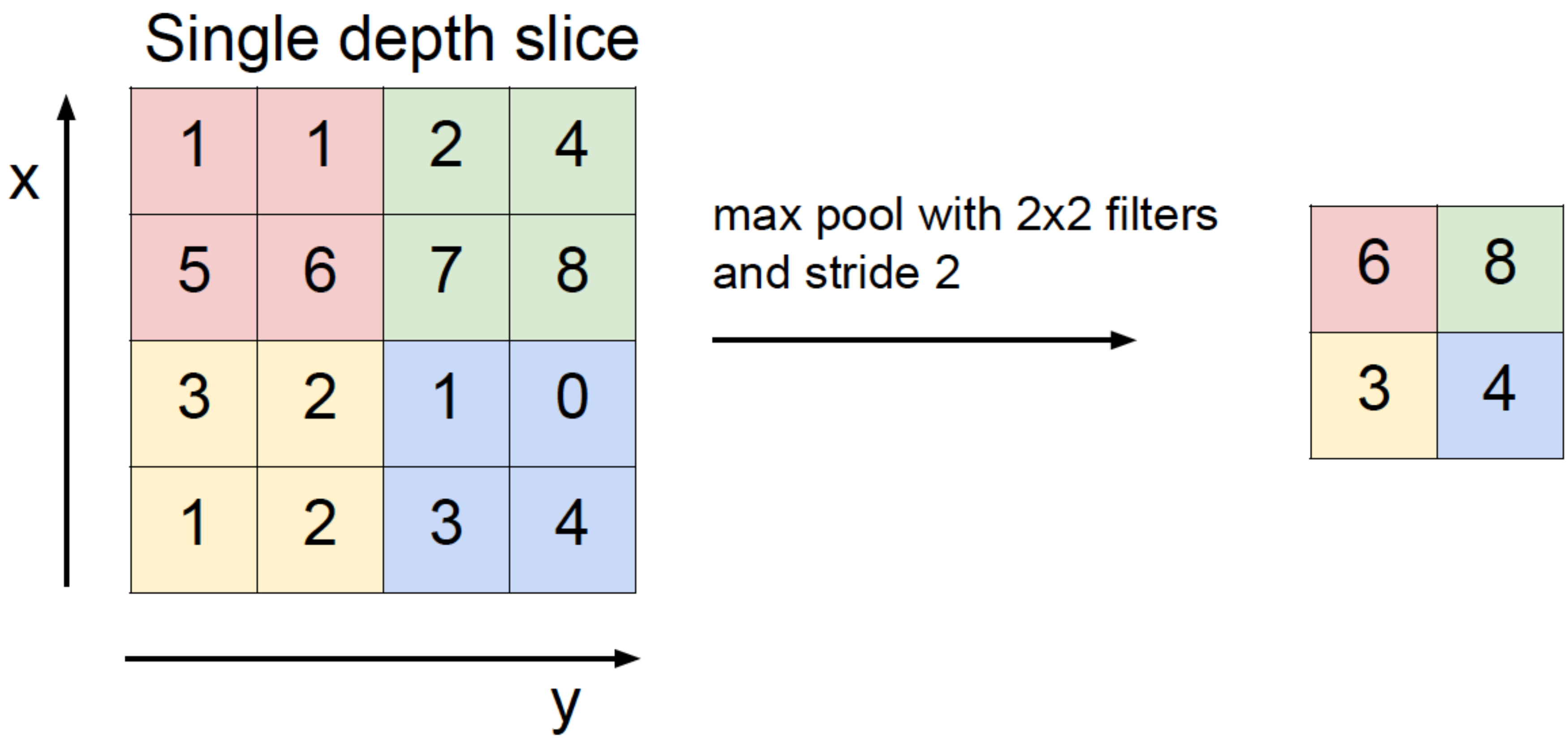
- The weight connections are surface-wise local!
  - Local connectivity
- The weights connections are depth-wise global
- For standard neurons no local connectivity
  - Everything is connected to everything





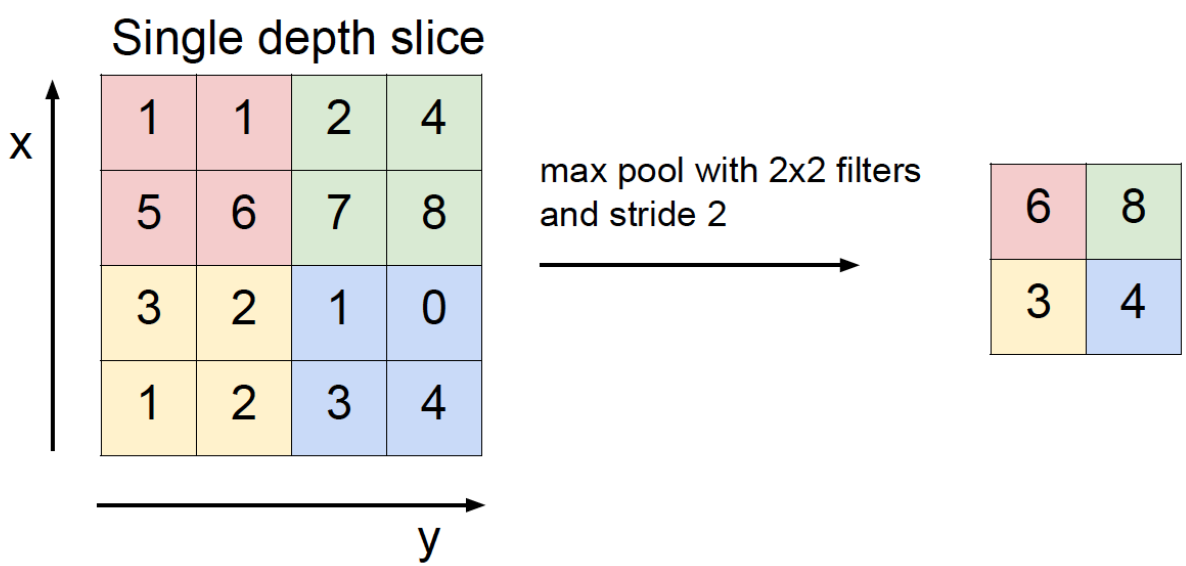
Pooling operations

- Aggregate multiple values into a single value

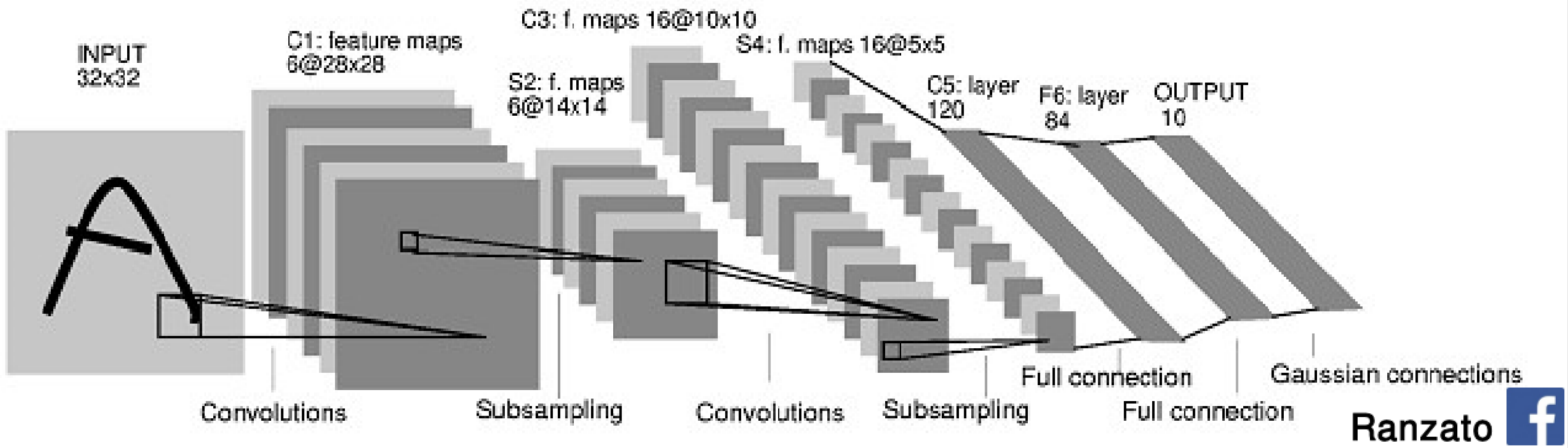


Pooling operations

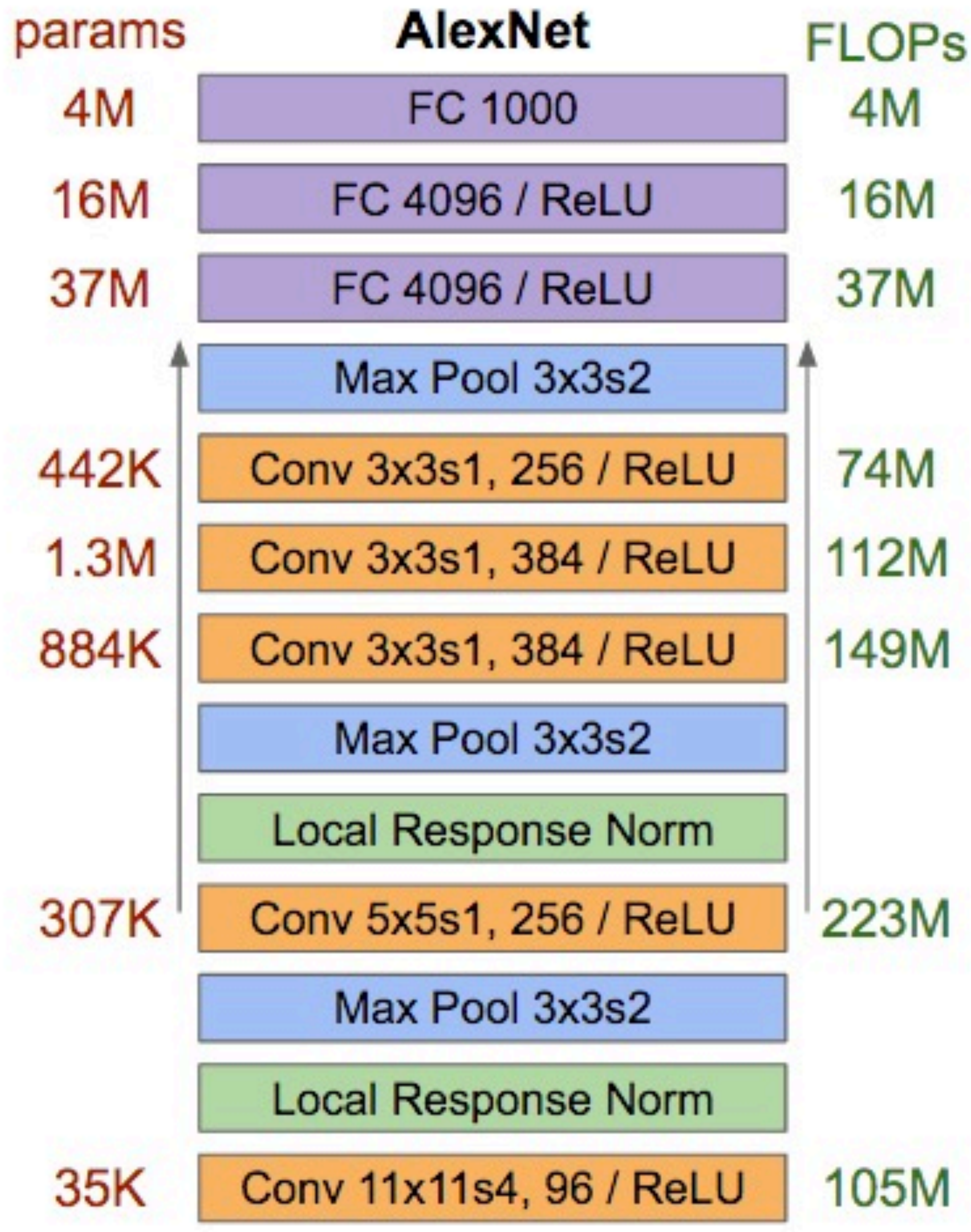
- Aggregate multiple values into a single value
- Invariance to small transformations
  - Keep only most important information for next layer
- Reduces the size of the next layer
  - Fewer parameters, faster computations
- Observe larger receptive field in next layer
  - Hierarchically extract more abstract features



Yann LeCun's MNIST CNN architecture



AlexNet for ImageNet



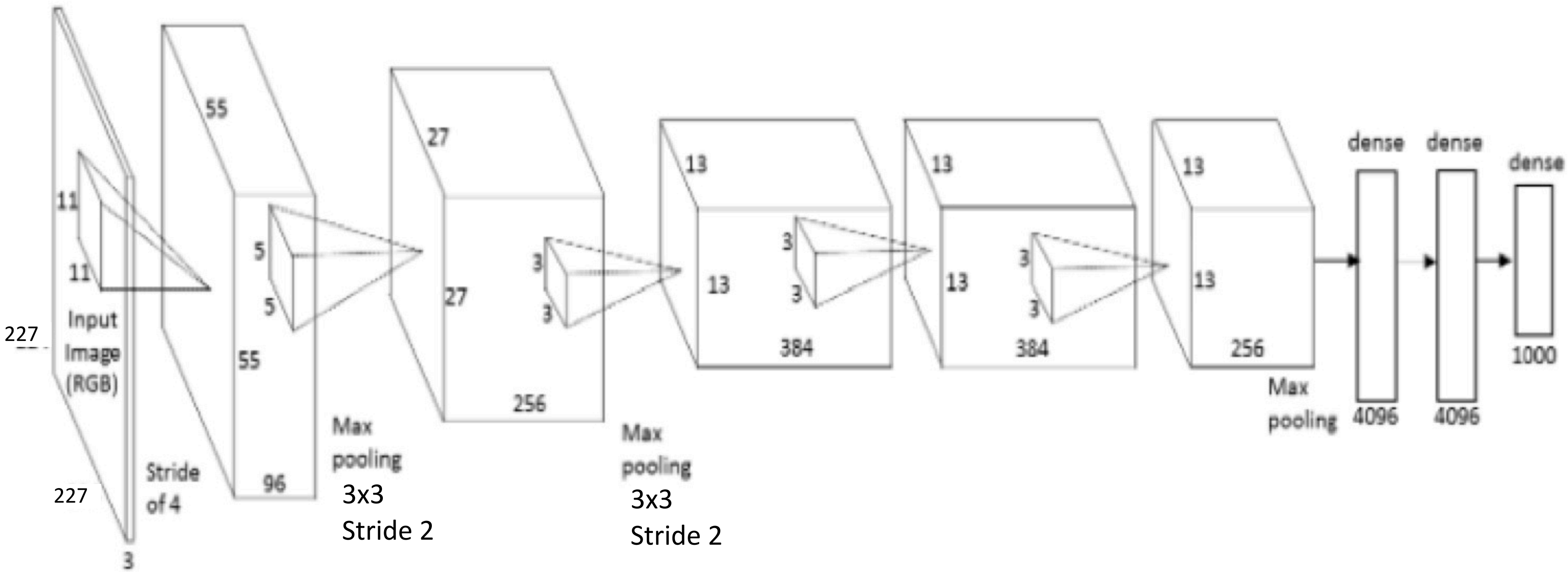
Layers

- Kernel sizes
- Strides
- # channels
- # kernels
- Max pooling

[Krizhevsky et al. 2012]

AlexNet diagram (simplified)

Input size
227 x 227 x 3



Conv 1
11 x 11 x 3
Stride 4
96 filters

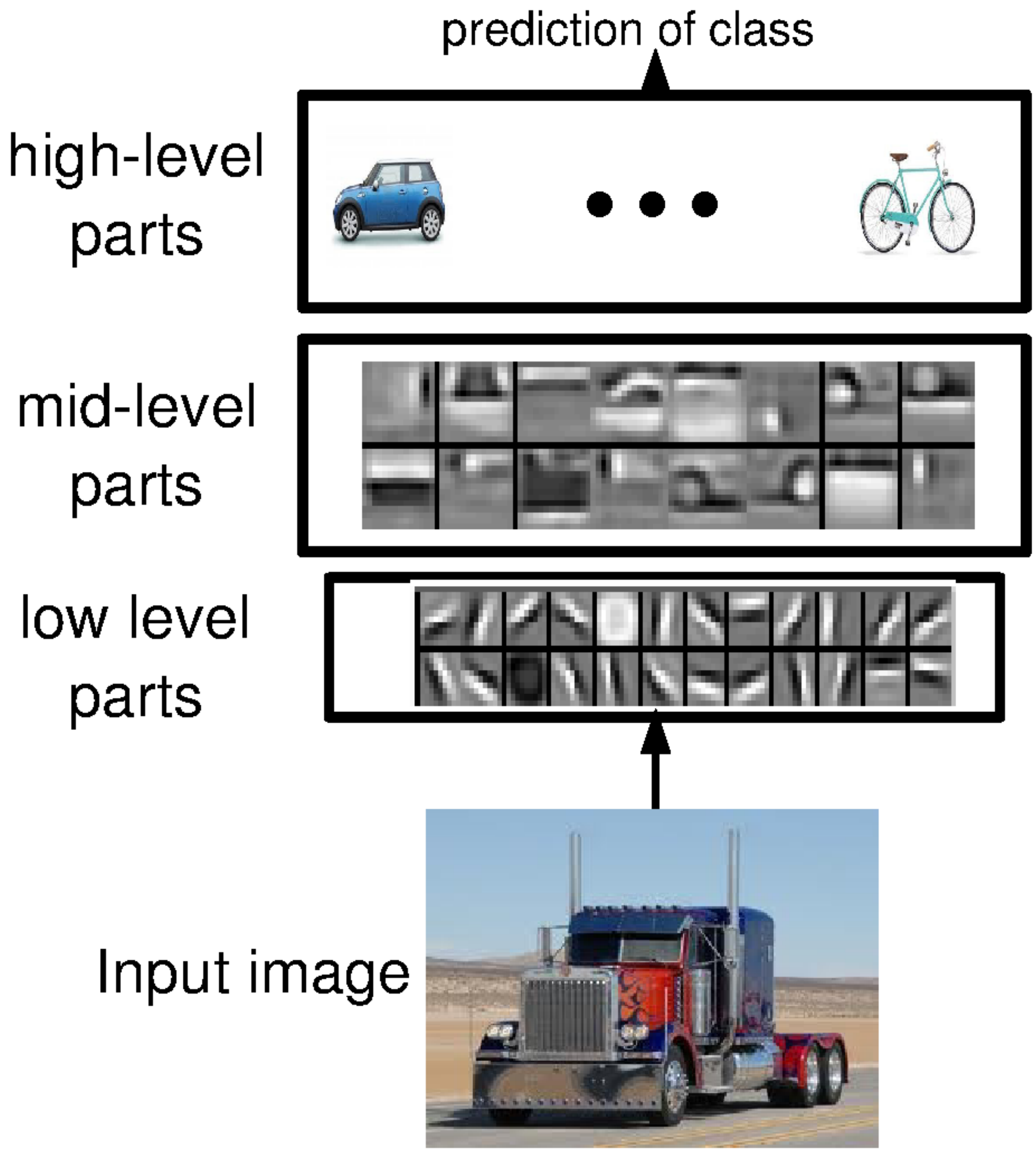
Conv 2
5 x 5 x 48
Stride 1
256 filters

Conv 3
3 x 3 x 256
Stride 1
384 filters

Conv 4
3 x 3 x 192
Stride 1
384 filters

Conv 4
3 x 3 x 192
Stride 1
256 filters

Interpretation



- distributed representations
- feature sharing
- compositionality

Lee et al. “Convolutional DBN's ...” ICML 2009

Learning Neural Networks

Practice II: Setting Hyperparameters

Practice I: Setting Hyperparameters

Idea #1: Choose hyperparameters that work best on the data

Your Dataset

Practice I: Setting Hyperparameters

Idea #1: Choose hyperparameters that work best on the data

BAD: big network always works perfectly on training data

Your Dataset

Practice I: Setting Hyperparameters

Idea #1: Choose hyperparameters that work best on the data

BAD: big network always works perfectly on training data

Your Dataset

Idea #2: Split data into **train** and **test**, choose hyperparameters that work best on test data

train

test

Practice I: Setting Hyperparameters

Idea #1: Choose hyperparameters that work best on the data

BAD: big network always works perfectly on training data

Your Dataset

Idea #2: Split data into **train** and **test**, choose hyperparameters that work best on test data

BAD: No idea how algorithm will perform on new data

train

test

Practice I: Setting Hyperparameters

Idea #1: Choose hyperparameters that work best on the data

BAD: big network always works perfectly on training data

Your Dataset

Idea #2: Split data into **train** and **test**, choose hyperparameters that work best on test data

BAD: No idea how algorithm will perform on new data

train | test

Idea #3: Split data into **train**, **val**, and **test**; choose hyperparameters on val and evaluate on test

Better!

train | validation | test

Practice II: Select Optimizer

Stochastic gradient descent

Gradient from entire training set:

$$\nabla C = \frac{1}{n}\sum_x \nabla C_x$$

- For large training data, gradient computation takes a long time
  - Leads to “slow learning”

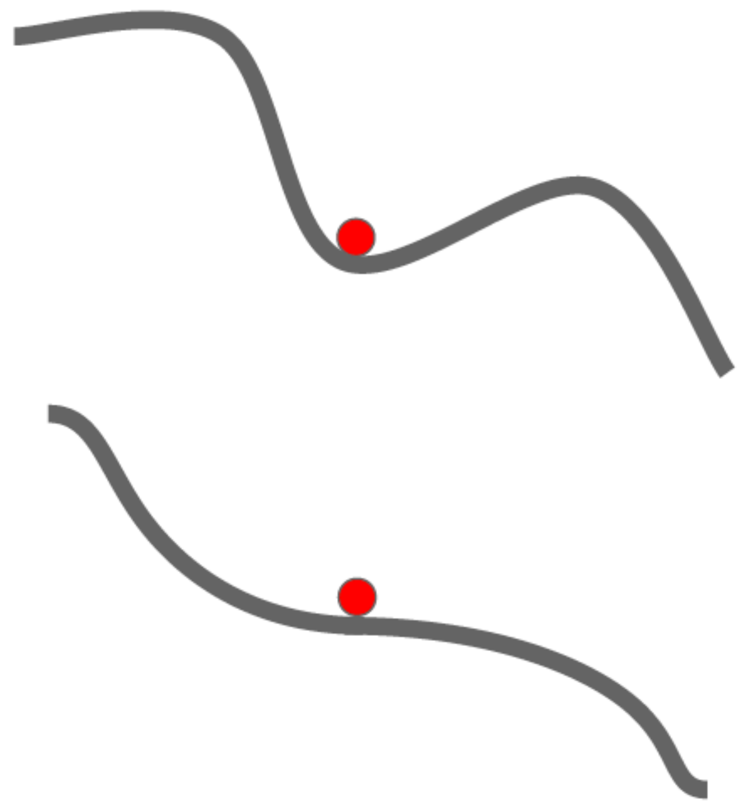
- Instead, consider a mini-batch with *m* samples
- If sample size is large enough, properties approximate the dataset

$$\frac{\sum_{j=1}^{m} \nabla C_{X_j}}{m} \approx \frac{\sum_x \nabla C_x}{n} = \nabla C$$

Stochastic gradient descent

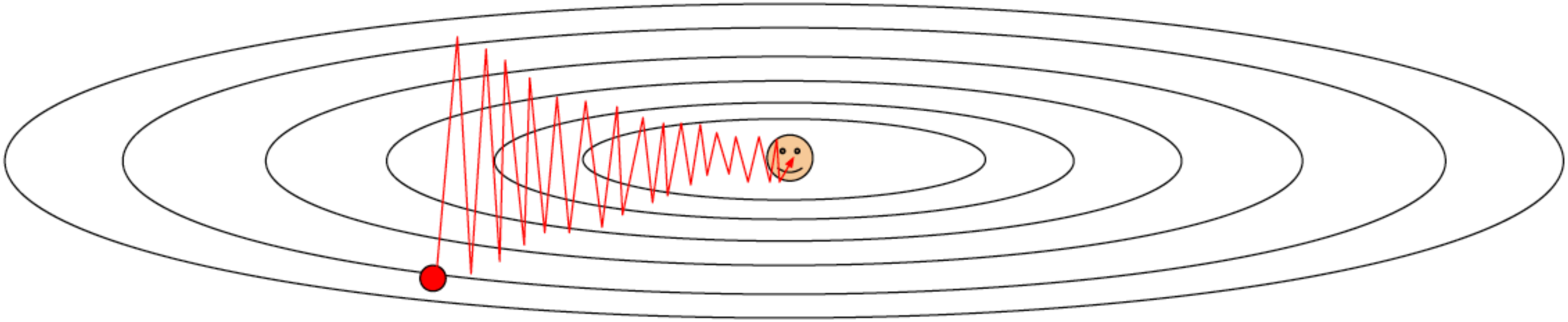
What if the loss function has a **local minima** or **saddle point**?

Zero gradient, gradient descent gets stuck



Stochastic gradient descent

What if loss changes quickly in one direction and slowly in another?
What does gradient descent do?
Very slow progress along shallow dimension, jitter along steep direction



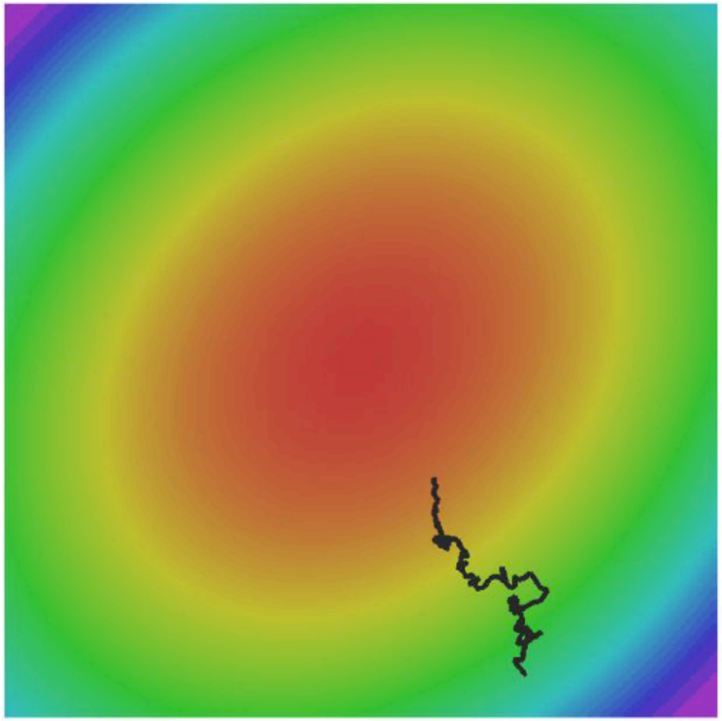
Loss function has high **condition number**: ratio of largest to smallest singular value of the Hessian matrix is large

Stochastic gradient descent

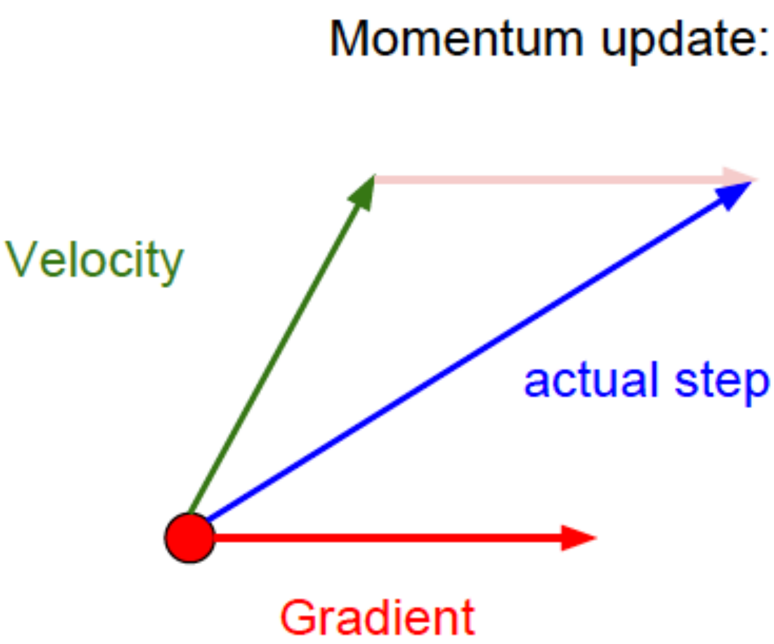
Our gradients come from minibatches so they can be noisy!

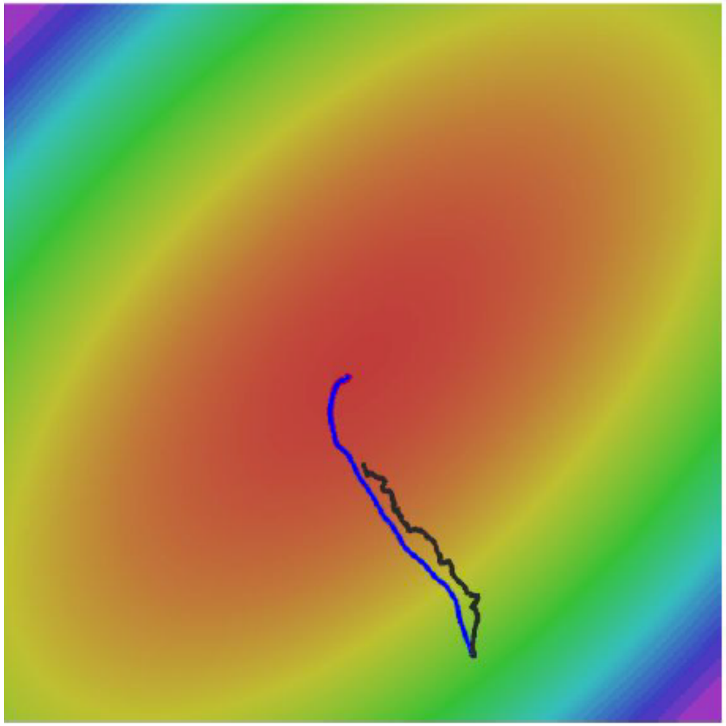
$$L(W) = \frac{1}{N} \sum_{i=1}^{N} L_i(x_i, y_i, W)$$

$$\nabla_W L(W) = \frac{1}{N} \sum_{i=1}^{N} \nabla_W L_i(x_i, y_i, W)$$



Stochastic gradient descent





SGD

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

SGD+Momentum

$$v_{t+1} = \rho v_t + \nabla f(x_t)$$
$$x_{t+1} = x_t - \alpha v_{t+1}$$

Build up velocity as a running mean of gradients.

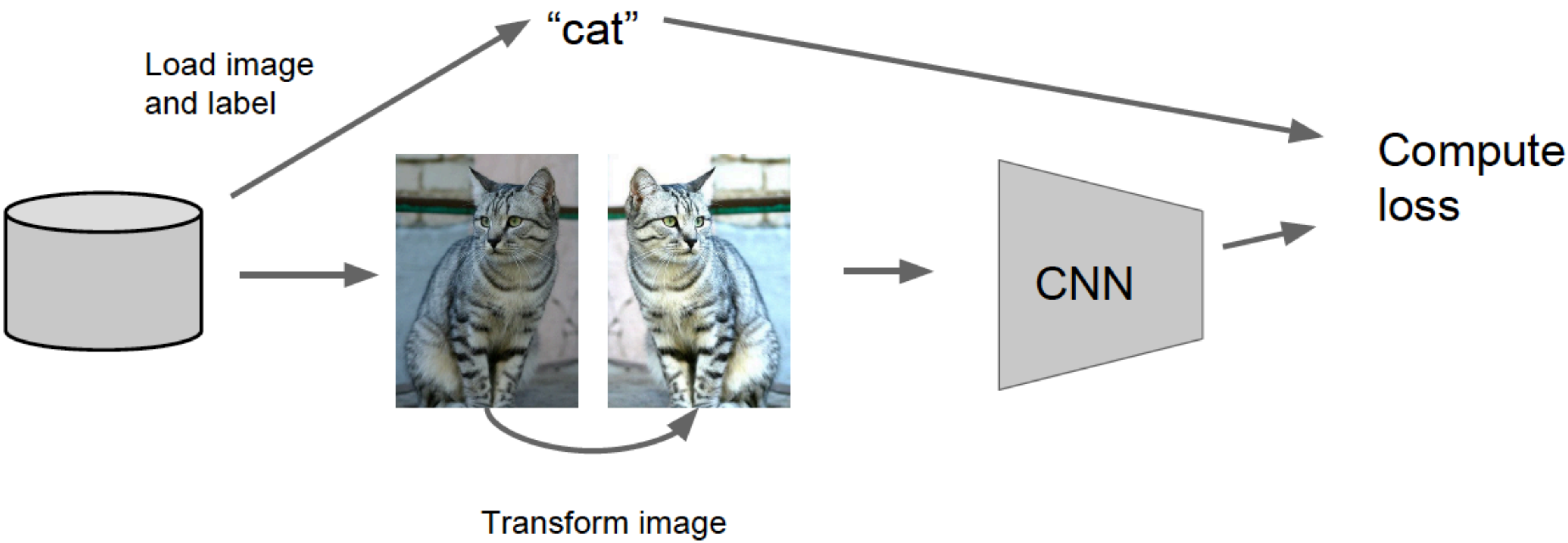
Many variations of using momentum

- In PyTorch, you can manually specify the momentum of SGD

- Or, you can use other optimization algorithms with “adaptive” momentum, e.g., ADAM
  - ADAM: Adaptive Moment Estimation

- Empirically, ADAM usually converges faster, but SGD gives local minima with better generalizability

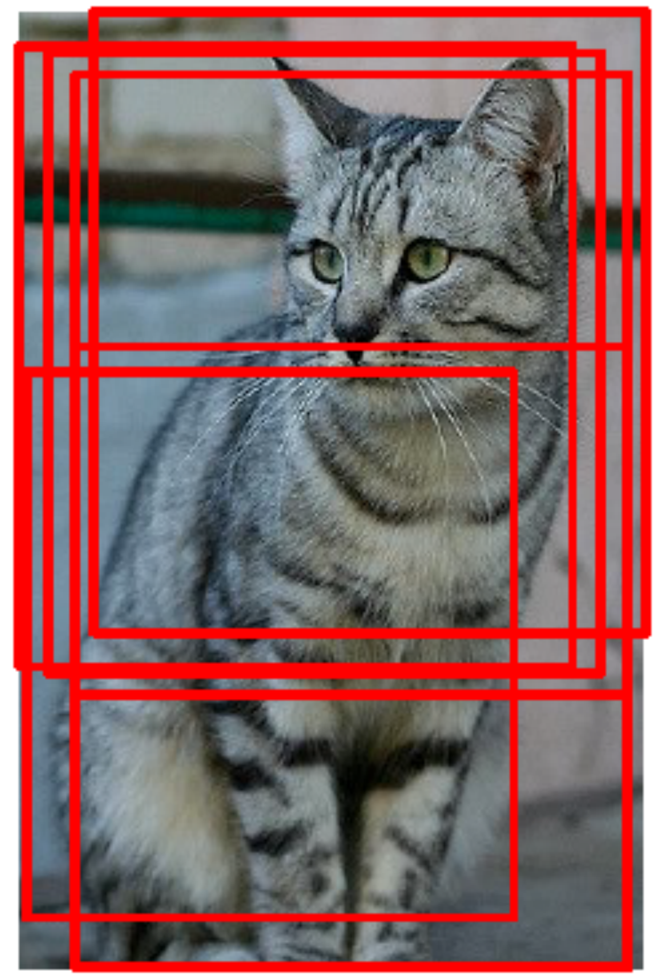
Practice III: Data Augmentation


"cat"
Load image
and label
Compute
loss
CNN
Transform image

Horizontal flips

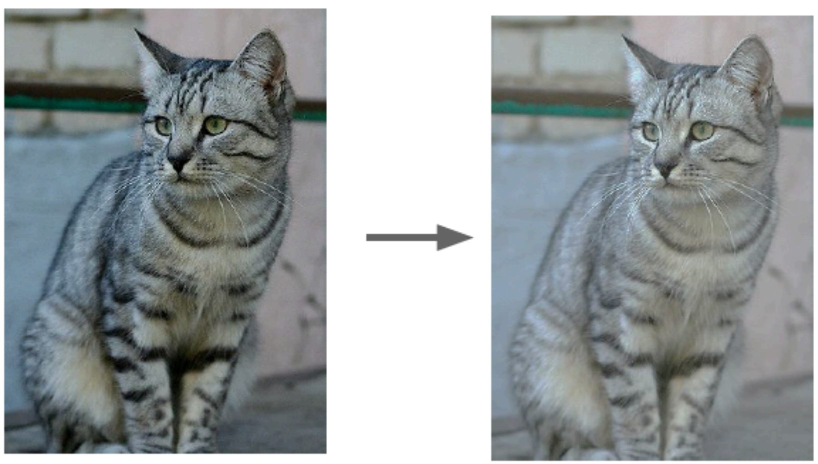
Random crops and scales

1. Pick random L in range [256, 480]
2. Resize training image, short side = L
3. Sample random 224 x 224 patch



Color jitter

Simple: Randomize contrast and brightness

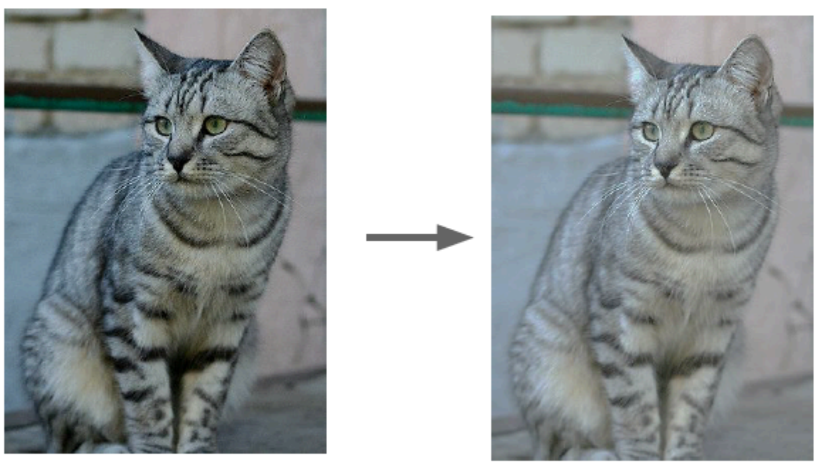


More Complex:

1. Apply PCA to all [R, G, B] pixels in training set
2. Sample a "color offset" along principal component directions
3. Add offset to all pixels of a training image

Color jitter

Simple: Randomize
contrast and brightness



More Complex:

1. Apply PCA to all [R, G, B] pixels in training set
2. Sample a “color offset” along principal component directions
3. Add offset to all pixels of a training image

Can do a lot more: rotation, shear, non-rigid, motion blur, lens distortions, ….

Exam

- Linear algebra, such as
  - rank, null space, range, invertible, eigen decomposition, SVD, pseudo inverse, basic matrix calculus
- Optimization:
  - Least square, low-rank approximation, statistical interpretation of PCA
- Image formation
  - diffuse/specular reflection, Lambertian lighting equation
- Filtering
  - Linear filter, filter vs convolution, properties of filters, filterbank, usage of filters, median filter
- Statistics:
  - Bias, variance, bias-variance tradeoff, overfitting, underfitting
- Neural network
  - Linear classifier, softmax, why linear classifier is insufficient, activation function, feed-forward pass, universality theorem, what does back-propagation do, stochastic gradient descent, concepts in neural networks, why CNN, concepts in CNN, how to set hyperparameter, moment in SGD, data augmentation