COMP 5630/6630:Machine Learning

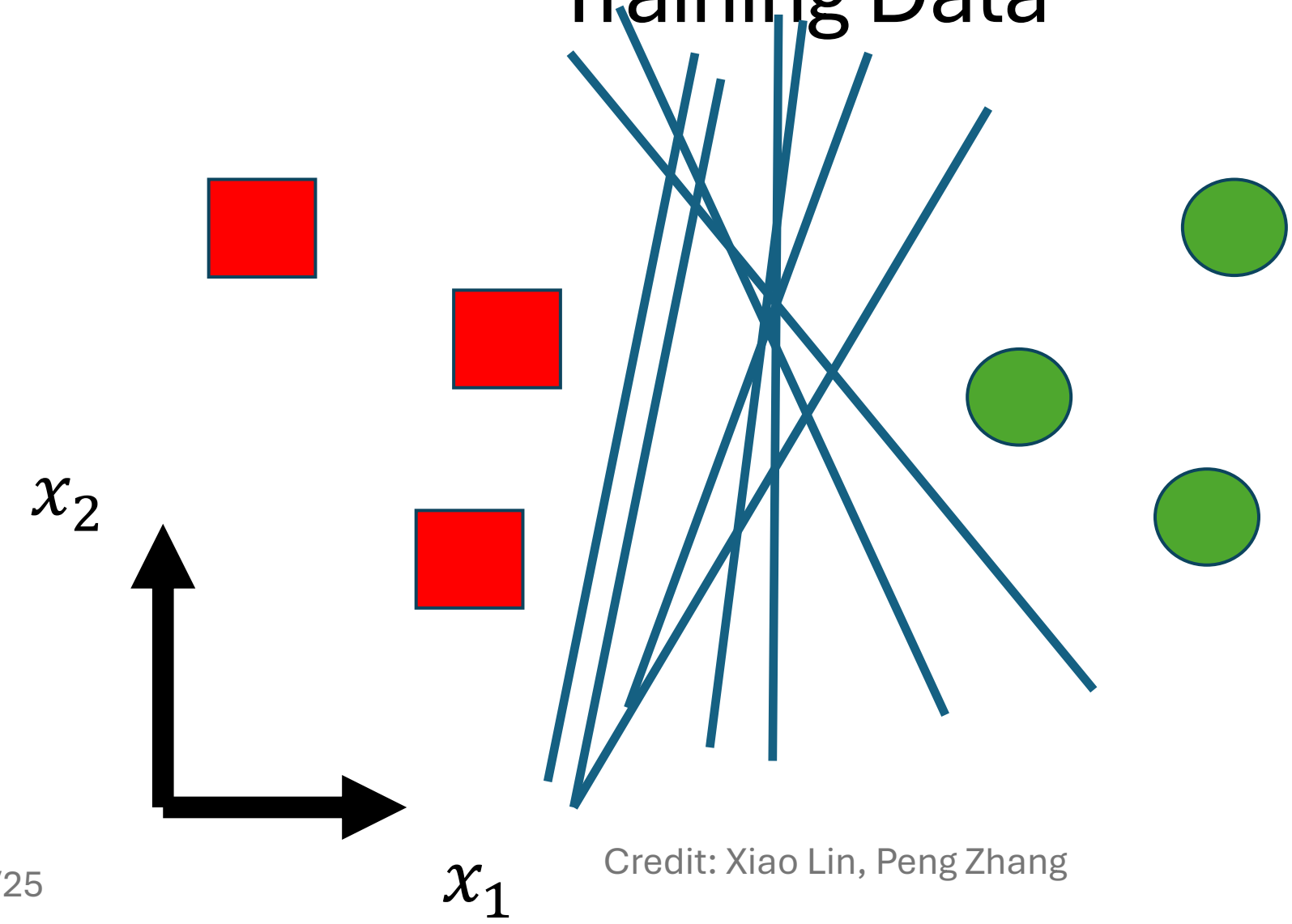
Lecture 9: Deep Learning, CNN

Adapted from: Drs. Aakur, : Marc'Aurelio Ranzato, Dhruv Batra

Regularization in Practice: Neural Network

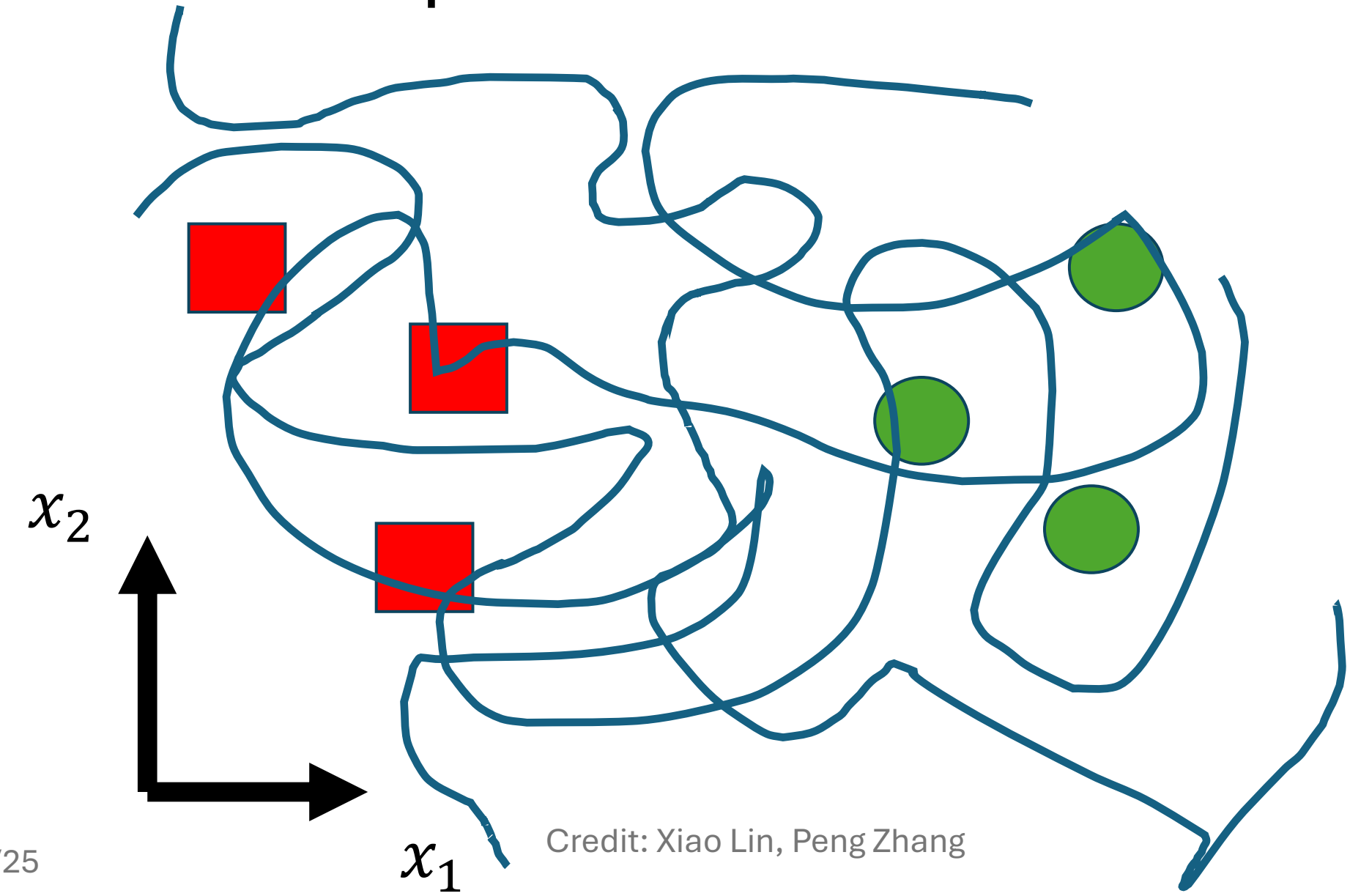
Finding the Optimal Decision Boundary

Training Data



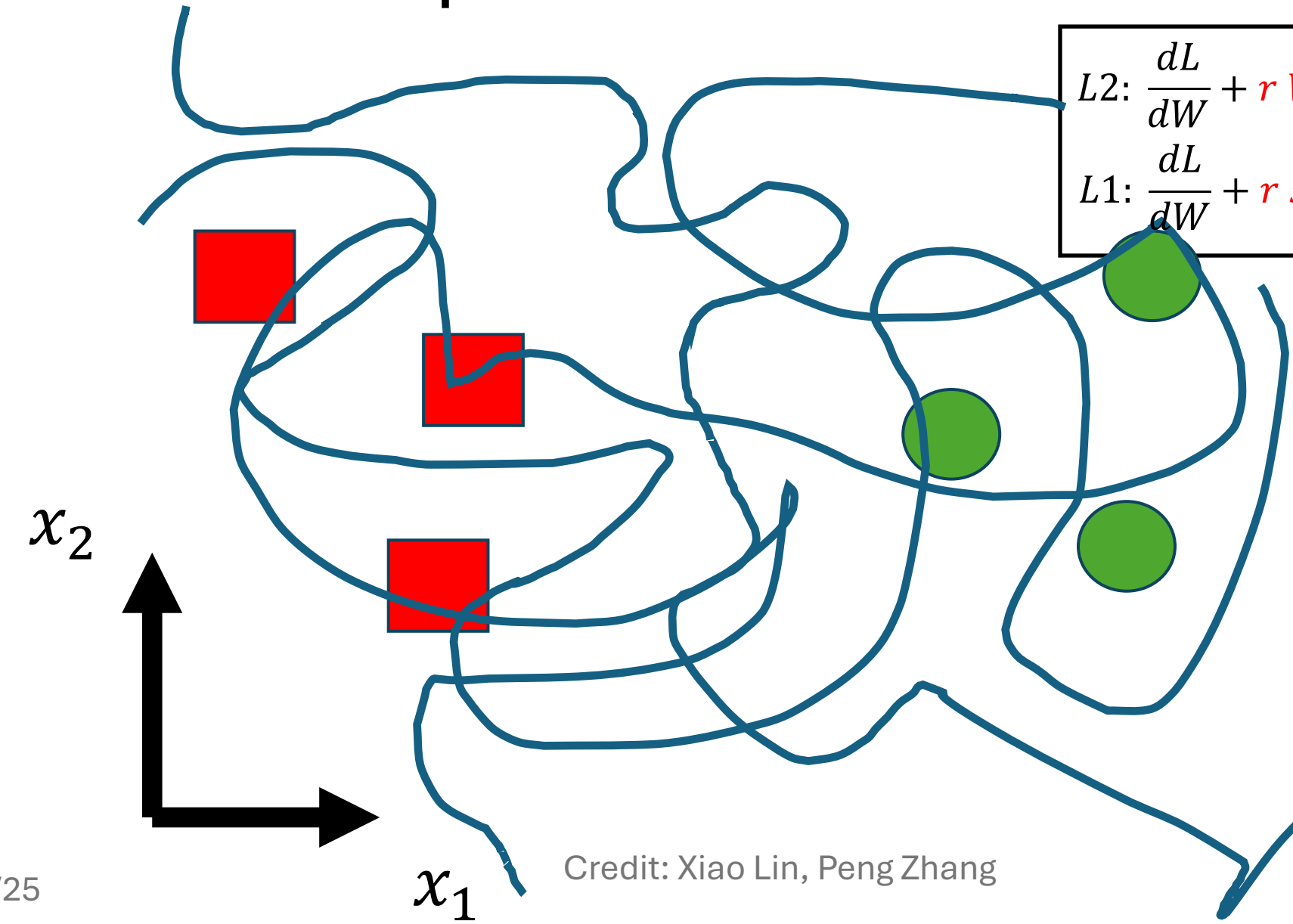
Credit: Xiao Lin, Peng Zhang

Optimal Neural Net?



Credit: Xiao Lin, Peng Zhang

Optimal Neural Net?



With Back propagation, we can compute and add the gradient of L1 or L2 norm. But they're not enough to prevent overfitting

Credit: Xiao Lin, Peng Zhang

Why regularization?

Generalization

- Prevent over-fitting

Occam's razor

Bayesian point of view

- Regularization corresponds to prior distributions on model parameters

Regularization in NN

1. Number of neurons
    - Fixing number of neurons in hidden units using validation set

2. Dropout

3. Early stopping
    - Validation data

4. Weight Decay

Regularization: Limiting Number of Neurons

No. of input/output units determined by dimensions

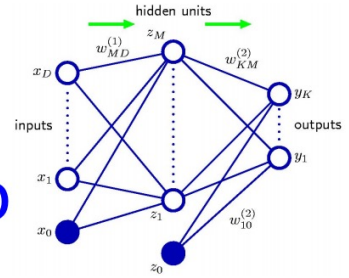
Number of hidden units M is a free parameter

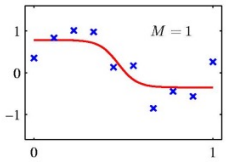
- Adjusted to get best predictive performance

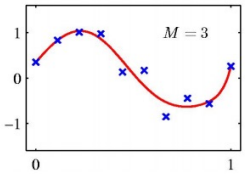
Possible approach is to get maximum likelihood estimate of M for balance between under-fitting and over-fitting

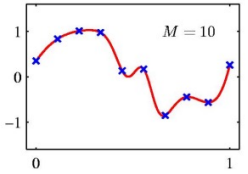
Regularization: Limiting Number of Neurons

Sinusoidal Regression Prob









Two layer network trained on 10 data points

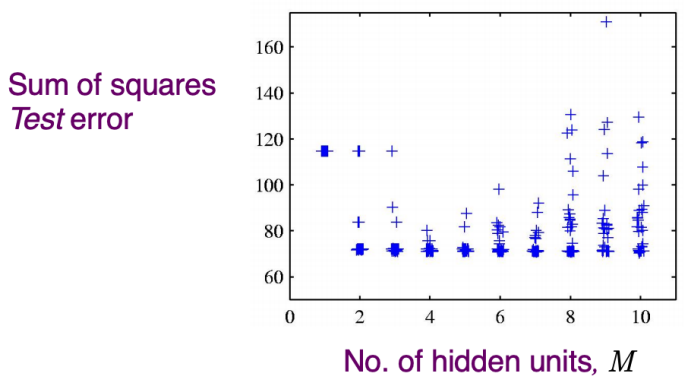
$M = 1, 3$ and 10 hidden units

Minimizing sum-of-squared error function
Using conjugate gradient descent

Generalization error is not a simple function of M
due to presence of local minima in error function

Regularization: Finding Number of Neurons Using Validation Set

Plot a graph choosing random starts and different numbers of hidden units M and choose the specific solution having smallest generalization error

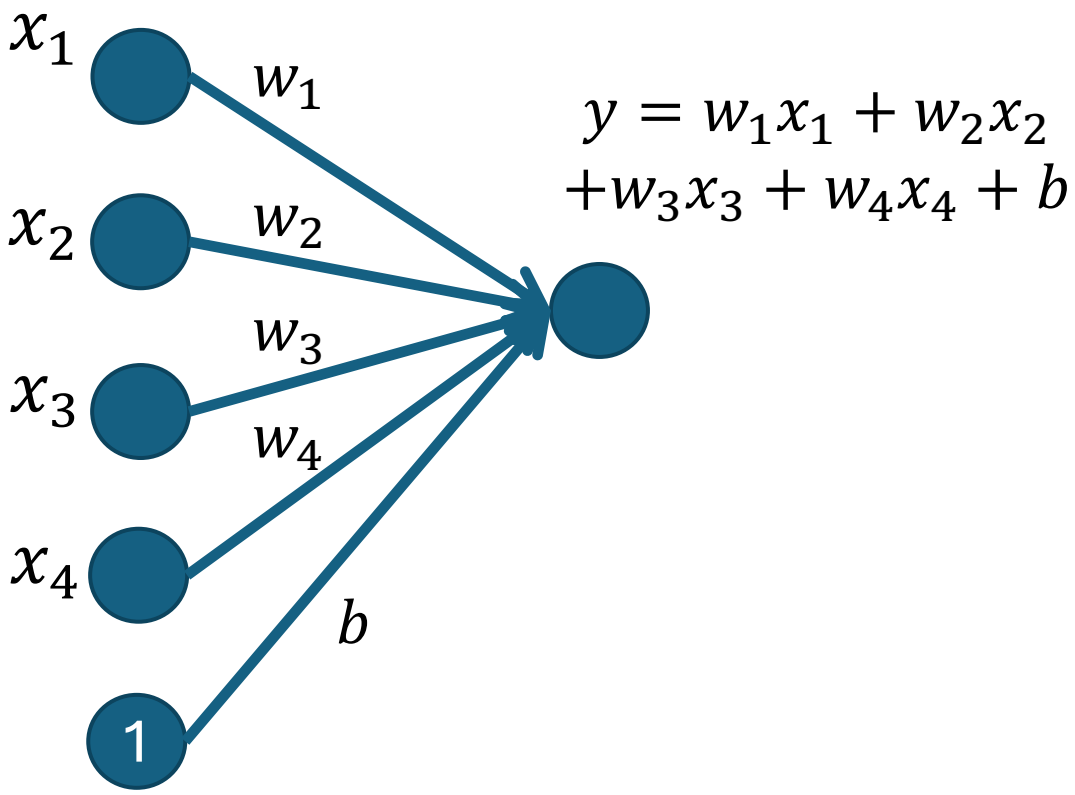


- 30 random starts for each M
  30 points in each column of graph
- Overall best *validation* set performance happened at $M=8$

There are other ways to control the complexity of a neural network in order to avoid over-fitting

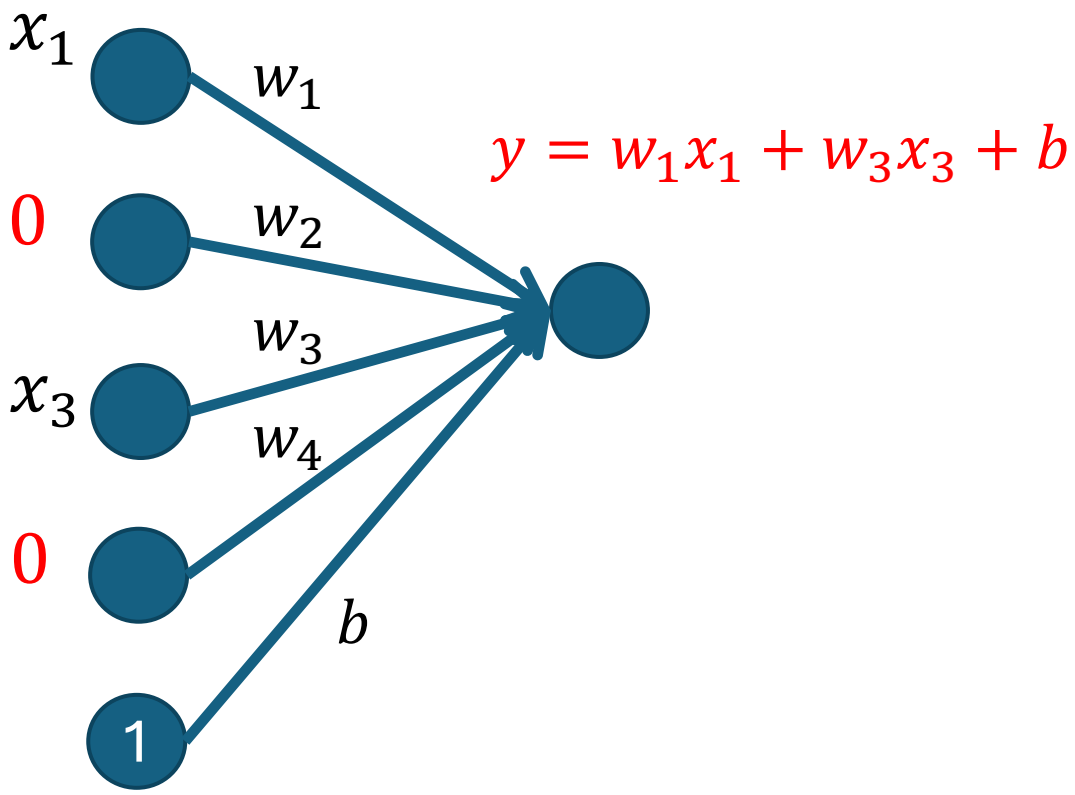
Alternative approach is to choose a relatively large value of M and then control complexity by adding a regularization term

Regularization: Dropout



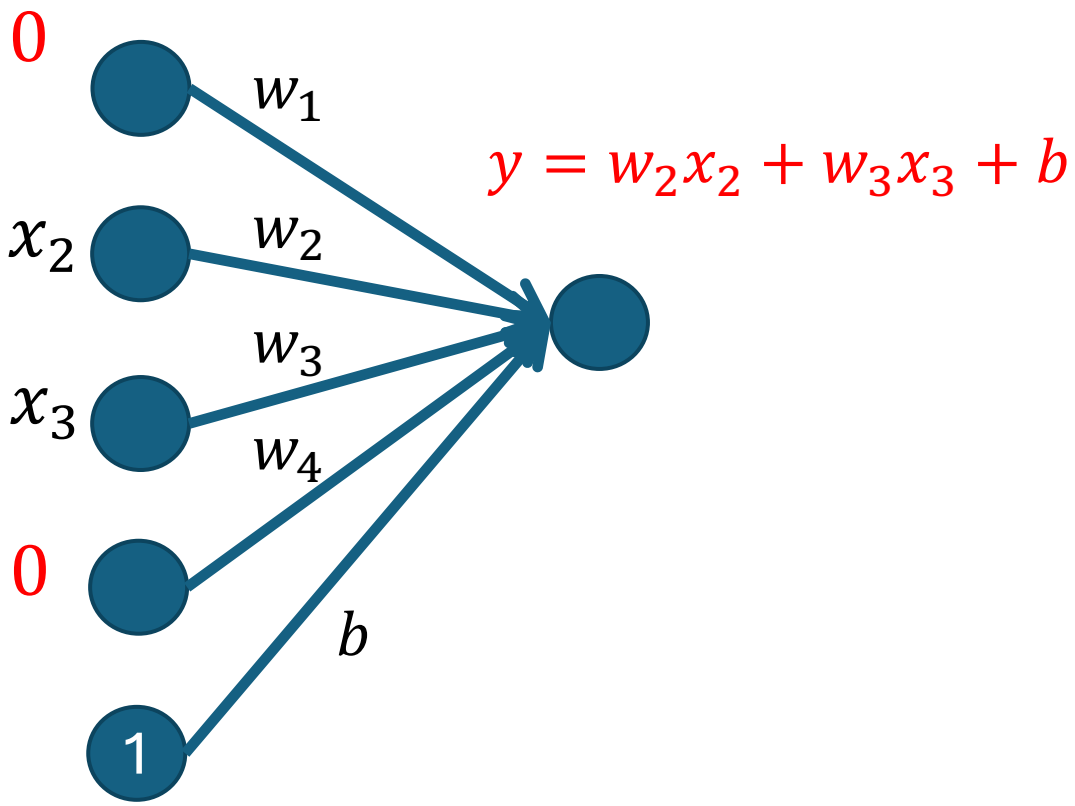
Regularization: Dropout

- During training
  - For each data point
  - Randomly set input to 0 with probability 0.5 “dropout ratio”.



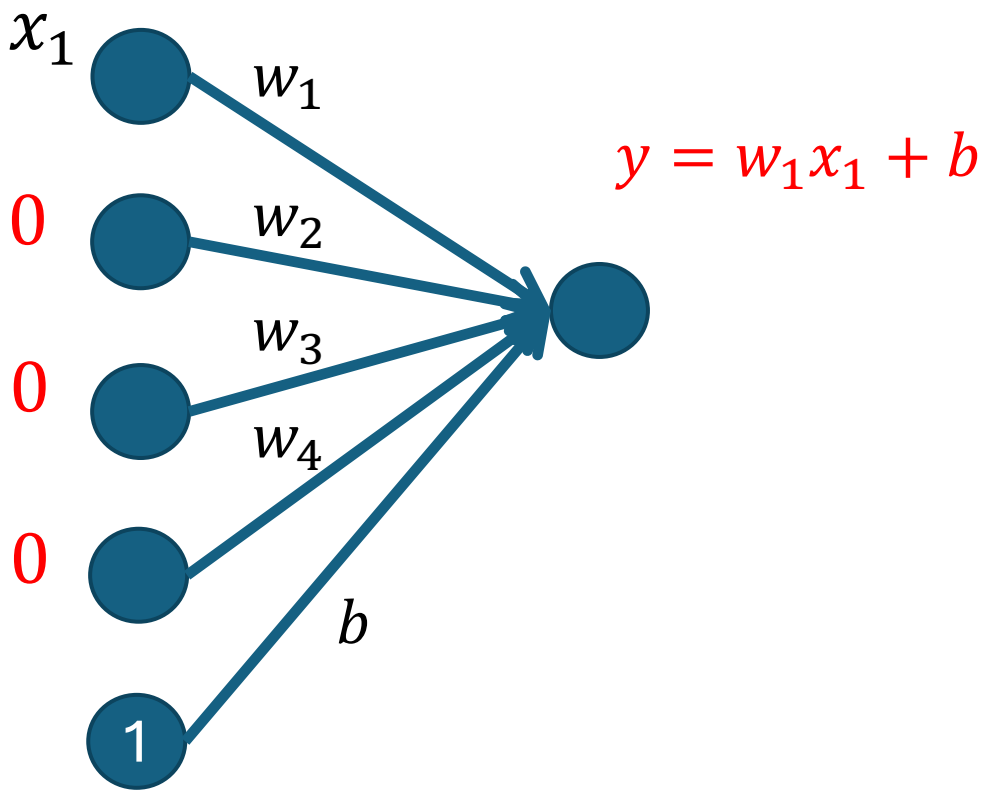
Regularization: Dropout

- During training
  - For each data point
  - Randomly set input to 0 with probability 0.5 “dropout ratio”.



Regularization: Dropout

- During training
  - For each data point
  - Randomly set input to 0 with probability 0.5 “dropout ratio”.



Regularization: Early Stopping

- Stopping training process of a neural network when the model's performance on a validation set starts to decline.

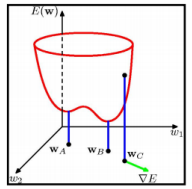
- Reduces overfitting without compromising on model accuracy.
- Steps

1. Split the data: Split the training data: training set and a validation set.
2. Train and evaluate: Use the training set to train model and evaluate its performance on the validation set after each epoch.
3. Patience setting: The number of epochs without improvement on the validation set before training stops
4. Identify the inflection point: Look for the point where the validation curve starts to increase, indicating overfitting.
5. Stop training: Stop training before the model overfits

Parameter Norm Penalty

Generalization error not a simple function of M

- Due to presence of local minima
- Need to control capacity to avoid over-fitting
  - Alternatively choose large M and control complexity by addition of regularization term



Simplest regularizer is *weight decay*

$$\tilde{E}(\boldsymbol{w}) = E(\boldsymbol{w}) + \frac{\lambda}{2}\boldsymbol{w}^{\mathrm{T}}\boldsymbol{w}$$

- Effective model complexity determined by choice of λ
- Regularizer is equivalent to a Gaussian prior over $\boldsymbol{w}$

In a 2-layer neural network

$$J_{regularized} = \underbrace{-\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}\log\left(a^{[L](i)}\right) + (1-y^{(i)})\log\left(1-a^{[L](i)}\right)\right)}_{\text{cross-entropy cost}} + \underbrace{\frac{1}{m}\frac{\lambda}{2}\sum_{l}\sum_{k}\sum_{j}W_{k,j}^{[l]2}}_{\text{L2 regularization cost}}$$

Regularization: Weight Decay

The name weight decay is due to the following

$$w_{t+1} = w_t - \alpha \nabla_w J - \lambda w_t$$

To prevent overfitting, every time we update a weight $\boldsymbol{w}$ with the gradient ∇J in respect to $\boldsymbol{w}$, we also subtract from it $\lambda \boldsymbol{w}$.

This gives the weights a tendency to decay towards zero, hence the name.

Deep Neural Networks

What is Deep Learning?

1. Computational models composed of multiple processing layers
   - To learn representations of data with multiple levels of abstraction
2. Dramatically improved state-of-the-art in:
   - Speech recognition, Visual object recognition, Object detection
   - Other domains: Drug discovery, Genomics
3. Discovers intricate structure in large data sets
   - Using backpropagation to change parameters
   - Compute representation in each layer from previous layer
4. Deep convolutional nets: image proc, video, speech
5. Recurrent nets: sequential data, e,g., text, speech

Limitations of Conventional ML

- Limited in ability to process natural data in raw form
- Pattern Recognition and Machine Learning systems require careful engineering and domain expertise to transform raw data, e.g., pixel values, into a feature vector for a classifier

Automatic Representation Learning

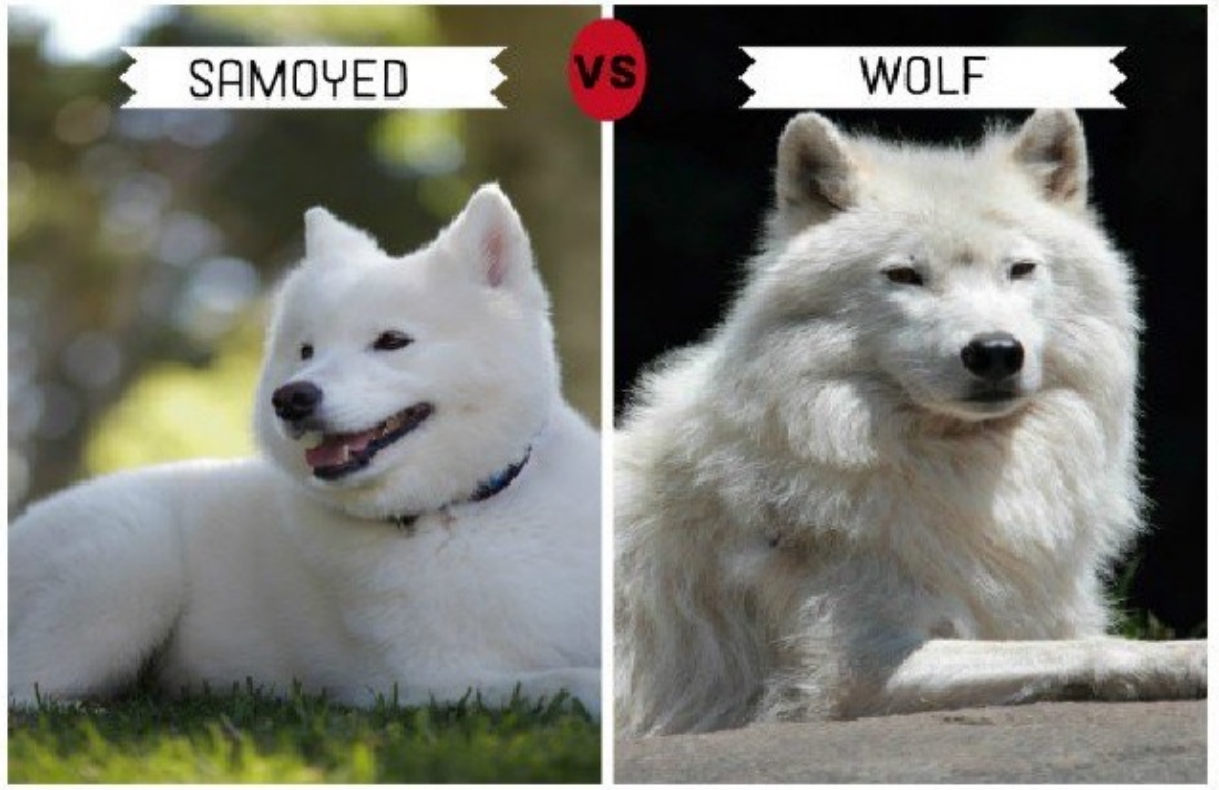
- Methods that allow a machine to be fed with raw data to automatically discover representations needed for detection or classification
- Deep Learning methods are Representation Learning Methods
- Use multiple levels of representation
  - Composing simple but non-linear modules that transform representation at one level (starting with raw input) into a representation at a higher slightly more abstract level
  - Complex functions can be learned
  - Higher layers of representation amplify aspects of input important for discrimination and suppress irrelevant variations

Example: Images

- Input is an array of pixel values
  - First stage is presence or absence of edges at particular locations and orientations of image
  - Second layer detects motifs by spotting particular arrangements of edges, regardless of small variations in edge positions
  - Third layer assembles motifs into larger combinations that corresponds to parts of familiar objects
  - Subsequent layers would detect objects as combinations of these parts
- Key aspect of deep learning:
  - These layers of features are not designed by human engineers
  - They are learned from data using a general purpose learning procedure

Deep versus Shallow Classifiers

- Linear classifiers can only carve the input space into very simple regions
- Image and speech recognition require input-output function to be insensitive to irrelevant variations of the input,
  - e.g., position, orientation and illumination of an object
  - Variations in pitch or accent of speech
  - While being sensitive to minute variations, e.g., white wolf and breed of wolf-like white dog called Samoyed
  - At pixel level two Samoyeds in different positions may be very different, whereas a Samoyed and a wolf in the same position and background may be very similar

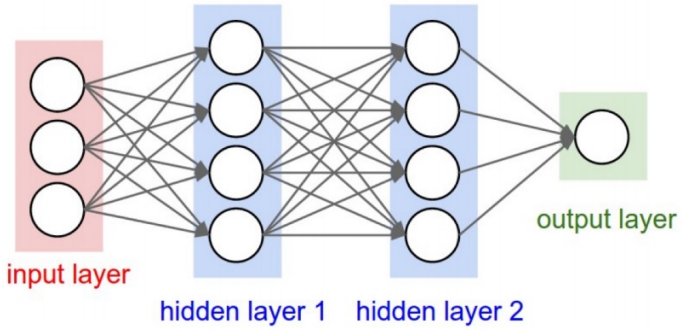


Selectivity-Invariance dilemma

- Shallow classifiers need a good feature extractor
- One that produces representations that are:
  - selective to aspects of image important for discrimination
  - but invariant to irrelevant aspects such as pose of the animal
- Generic features (e.g., Gaussian kernel) do not generalize well far from training examples
- Hand-designing good feature extractors requires engineering skill and domain expertise
- Deep learning learns features automatically

DL Architectures

- Multilayer stack of simple modules
- All modules (or most) subject to:
  - Learning
  - Non-linear input-output mappings
- Each module transforms input to improve both selectivity and invariance of the representation
- With depth of 5 to 20 layers can implement extremely intricate functions of input
  - Sensitive to minute details
    - Distinguish Samoyeds from white wolves
- Insensitive to irrelevant variations
  - Background, pose, lighting, surrounding objects



Convolutional Neural Networks

Convolutional Nets: Intuition

- Basic Idea
  - On board
  - Assumptions:
    - Local Receptive Fields
    - Weight Sharing / Translational Invariance / Stationarity
  - Each layer is just a convolution!

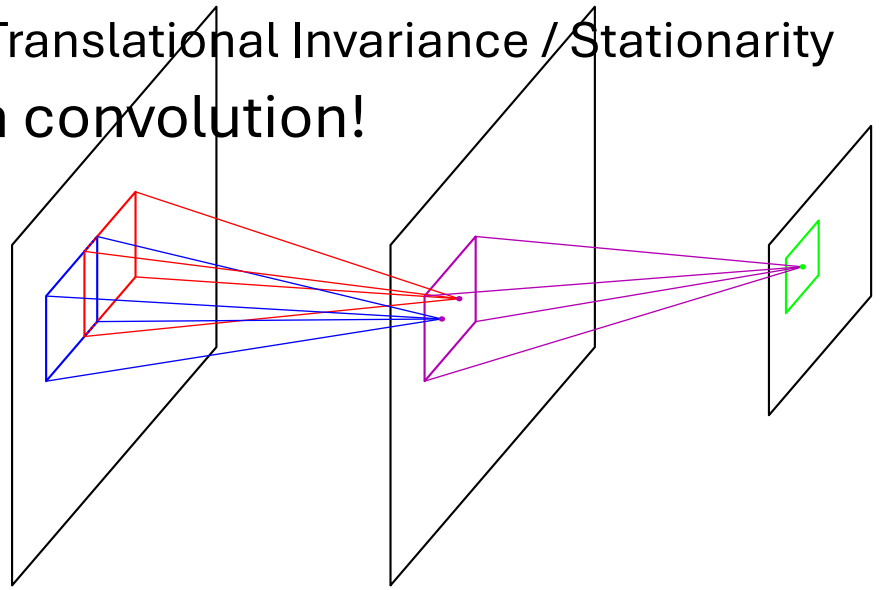
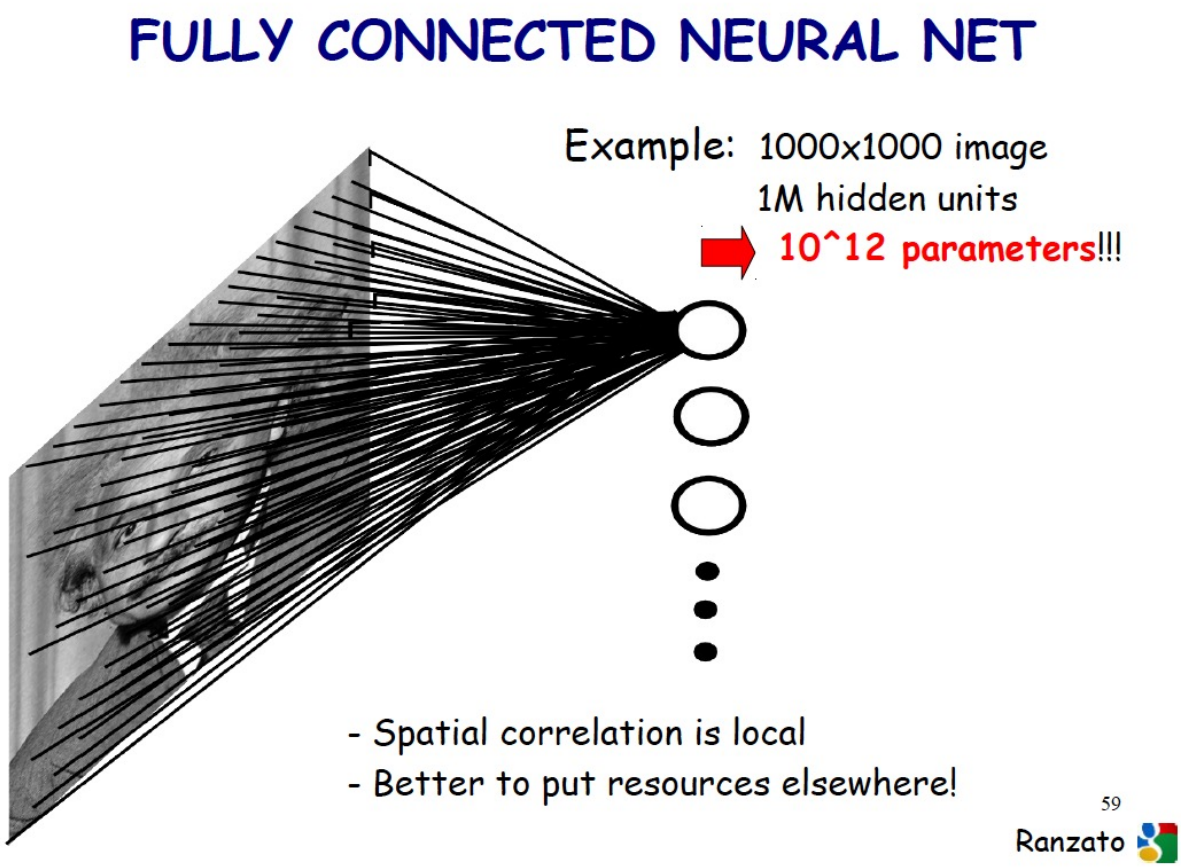


Image Credit: Chris Bishop

Convolutional Nets: Intuition



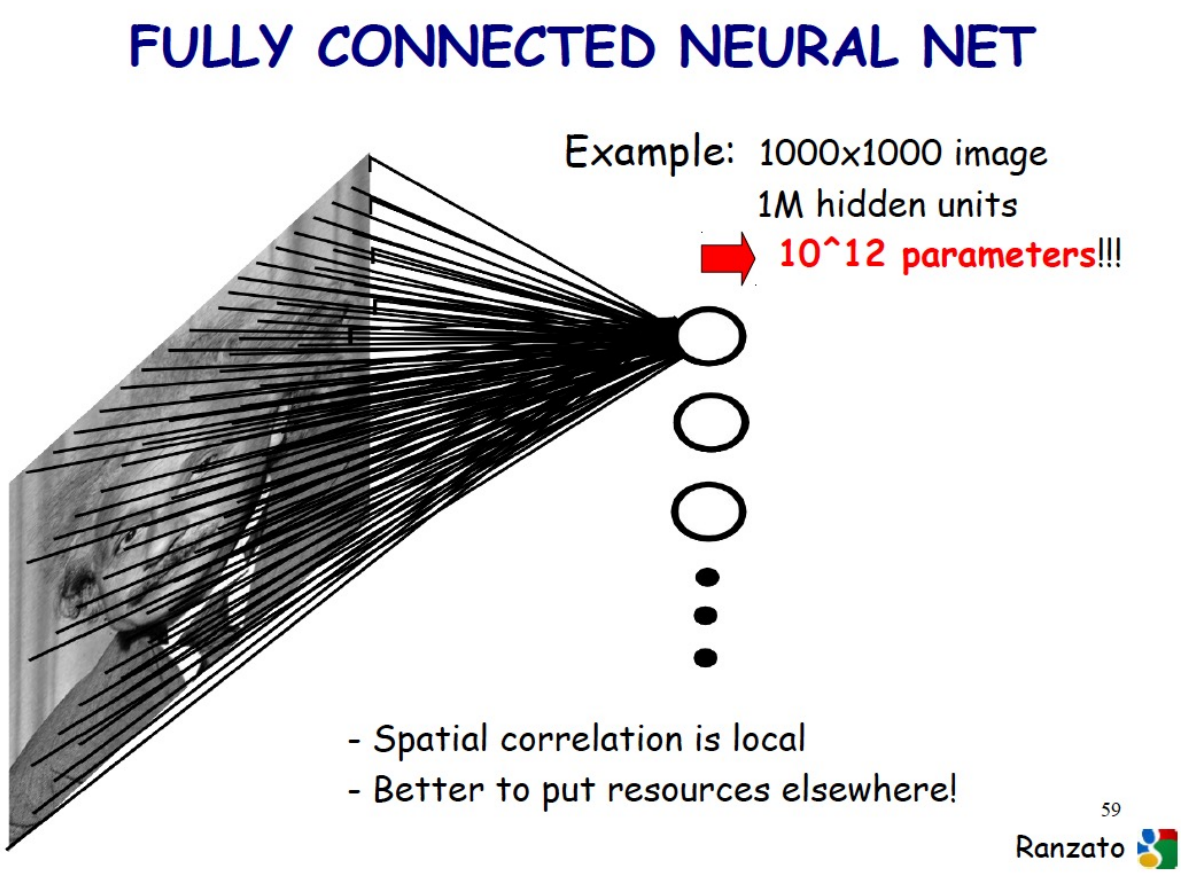
- Fully connected layer refers to a neural network in which each input node is connected to each output node.
  - We have seen MLP

Fully connected NN parameter count

Flatten Image (X)= 1000x 1000 =10^6
Hidden unit/neuron (W) = 10^6

Total number of parameters = ?

Convolutional Nets: Intuition



Fully connected layer refers to a neural network in which each input node is connected to each output node.

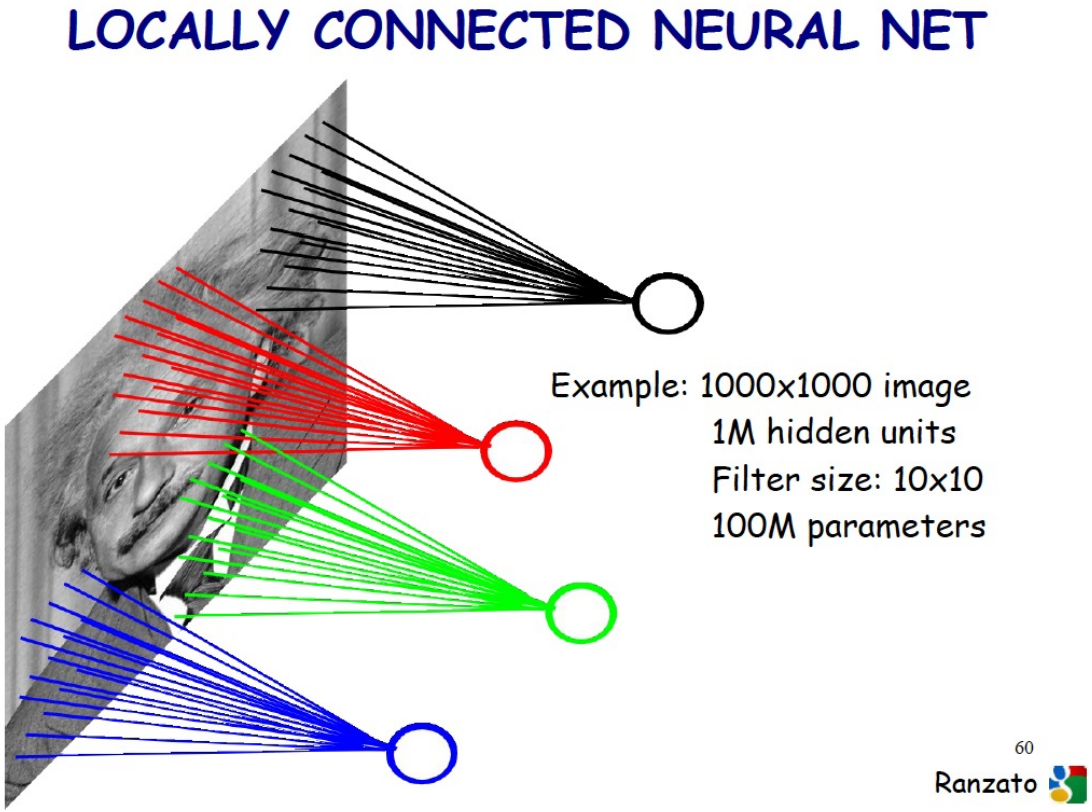
Fully connected NN parameter count

Flatten Image (X)= 1000x 1000 =10^6
Hidden unit/neuron (W) = 10^6

Total number of parameters = 10^6 x 10^6 = 10^12

Requires huge computational resource for large network.

Convolutional Nets: Assumption1

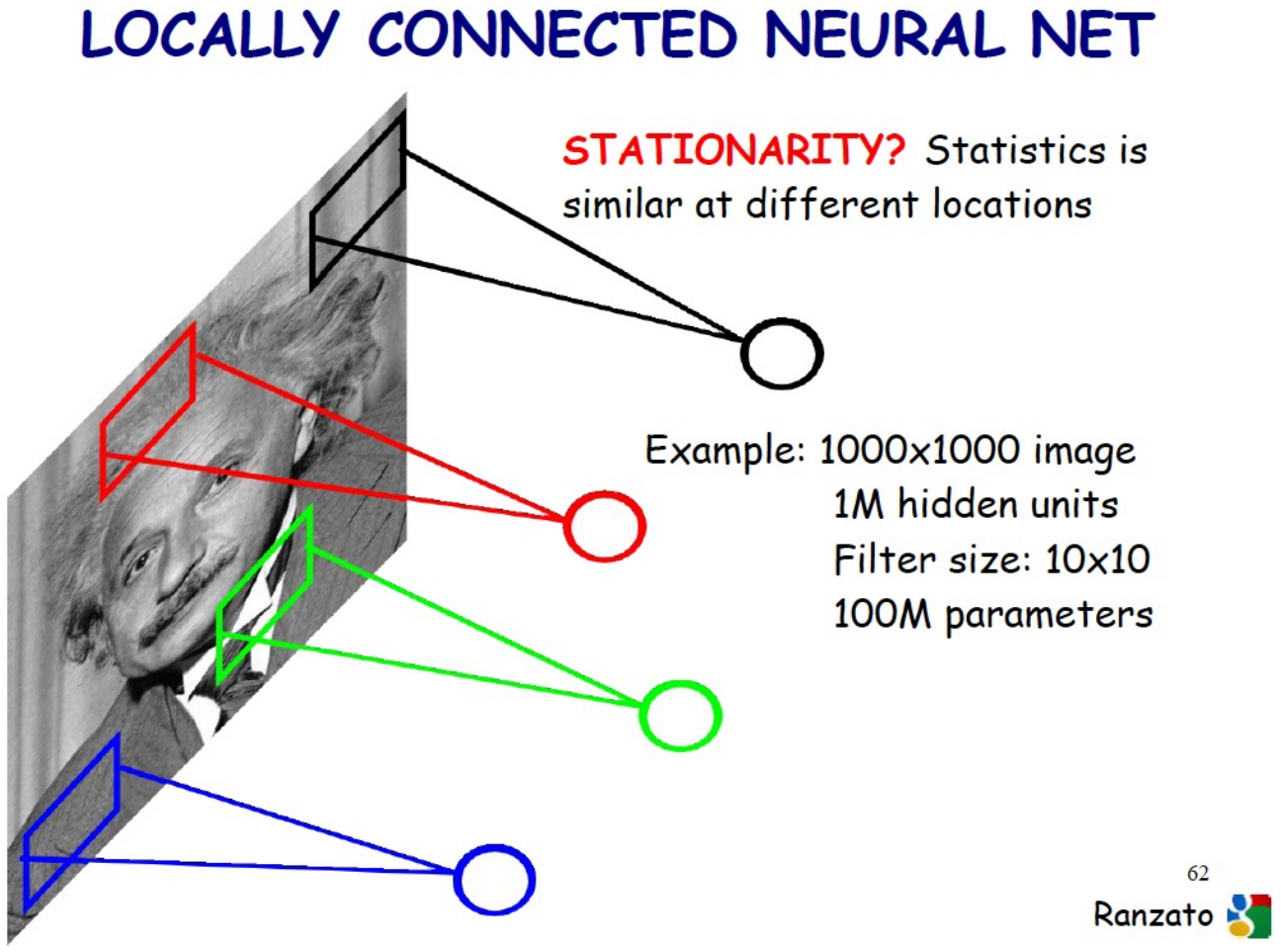


Flatten Image (X) 1000x 1000 =10^6
Hidden unit/neuron (W) = 10^6

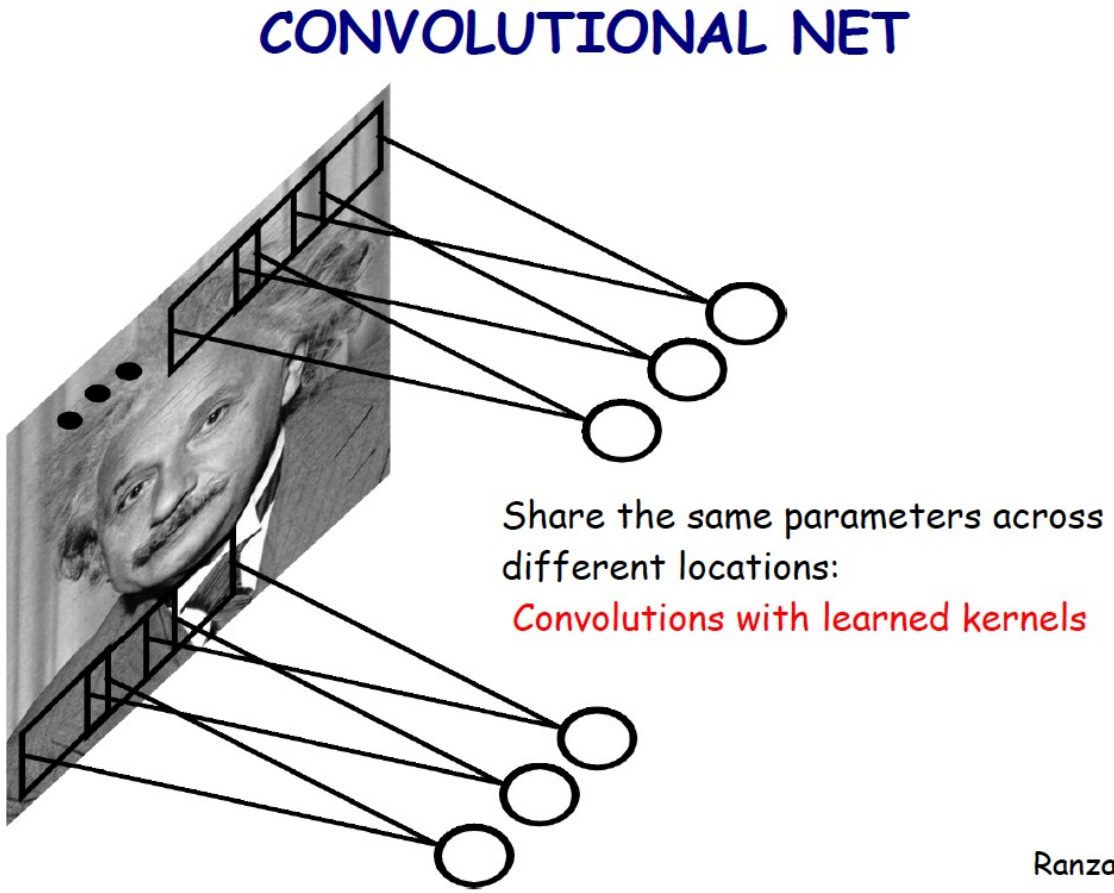
Each neuron is connected to only 10x10 units (Filter size)

Total number of parameters = 10^6 x 100 = 100M

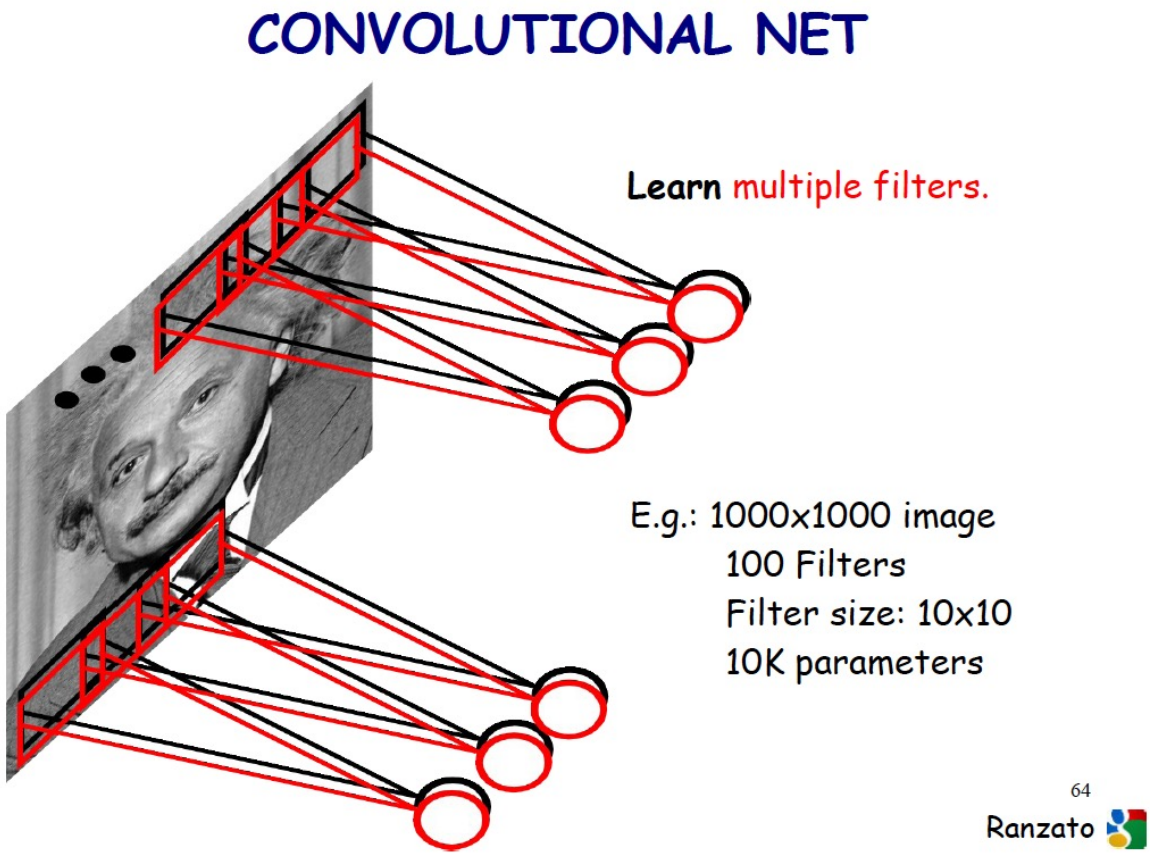
Convolutional Nets: Assumption 1



Convolutional Nets: Assumption 2



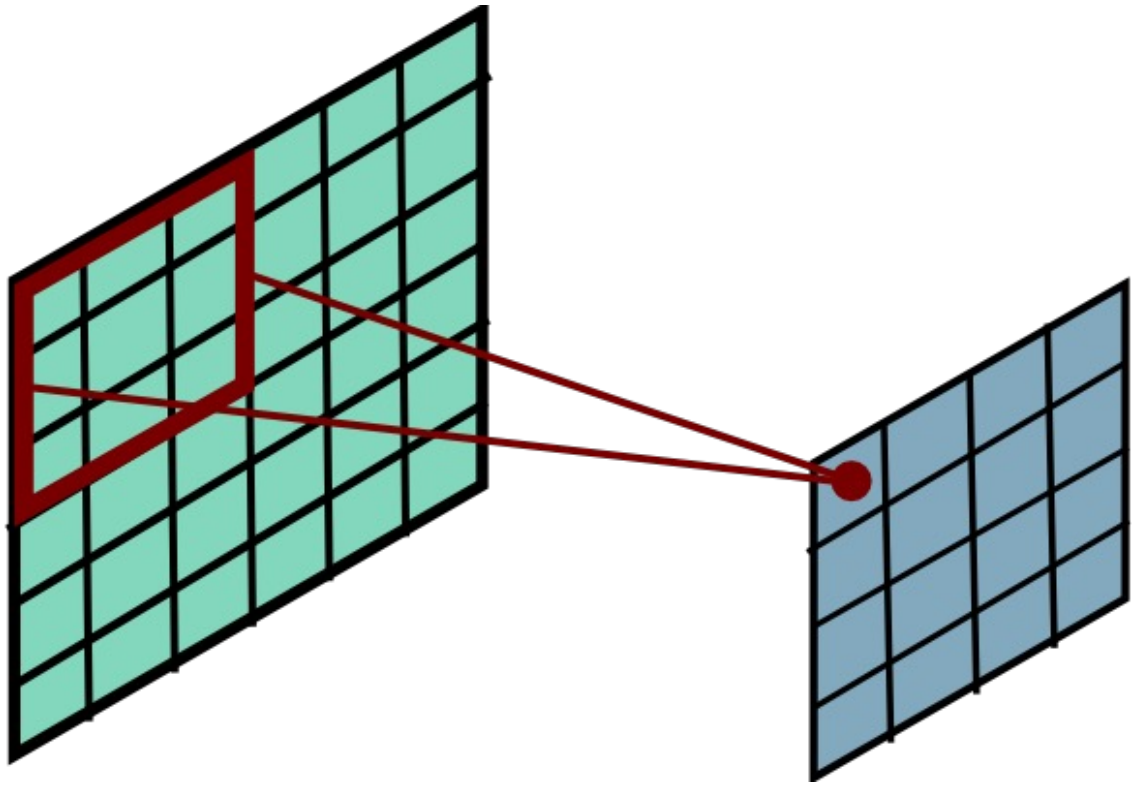
Convolutional Nets: Assumption 2



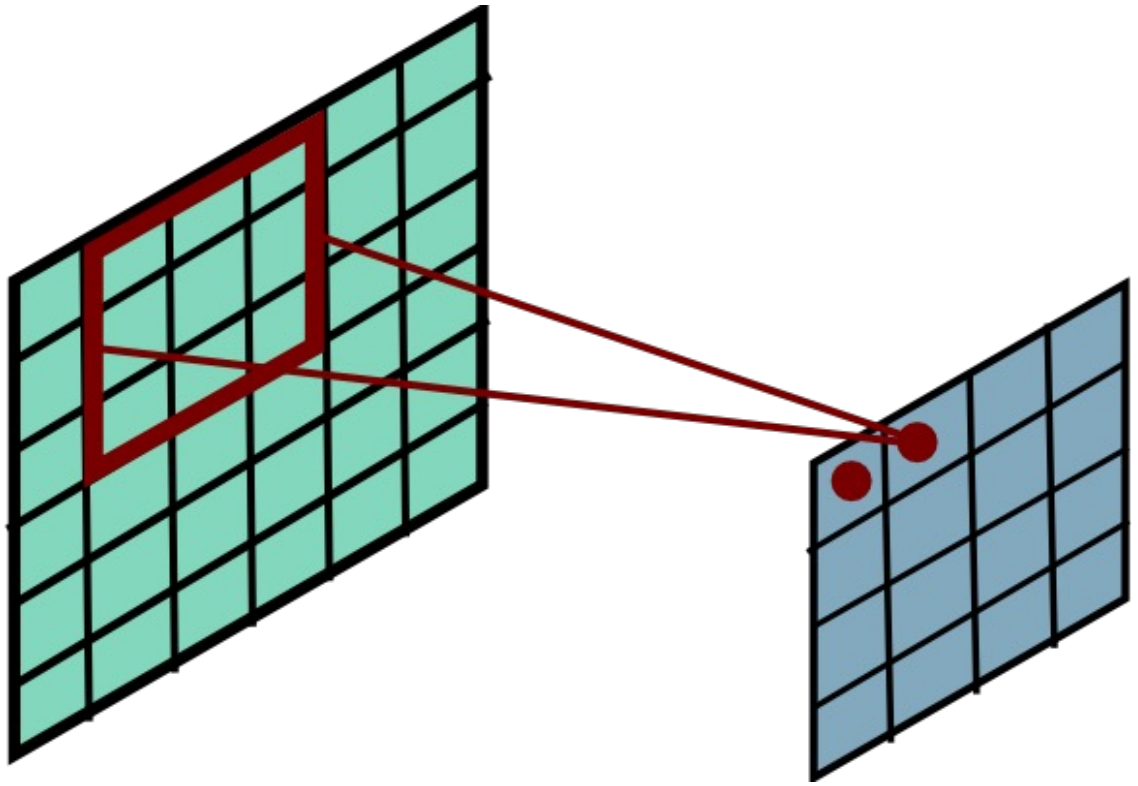
Share the same parameters across different locations (assuming input is stationary):

Convolutions with learned kernels/filters

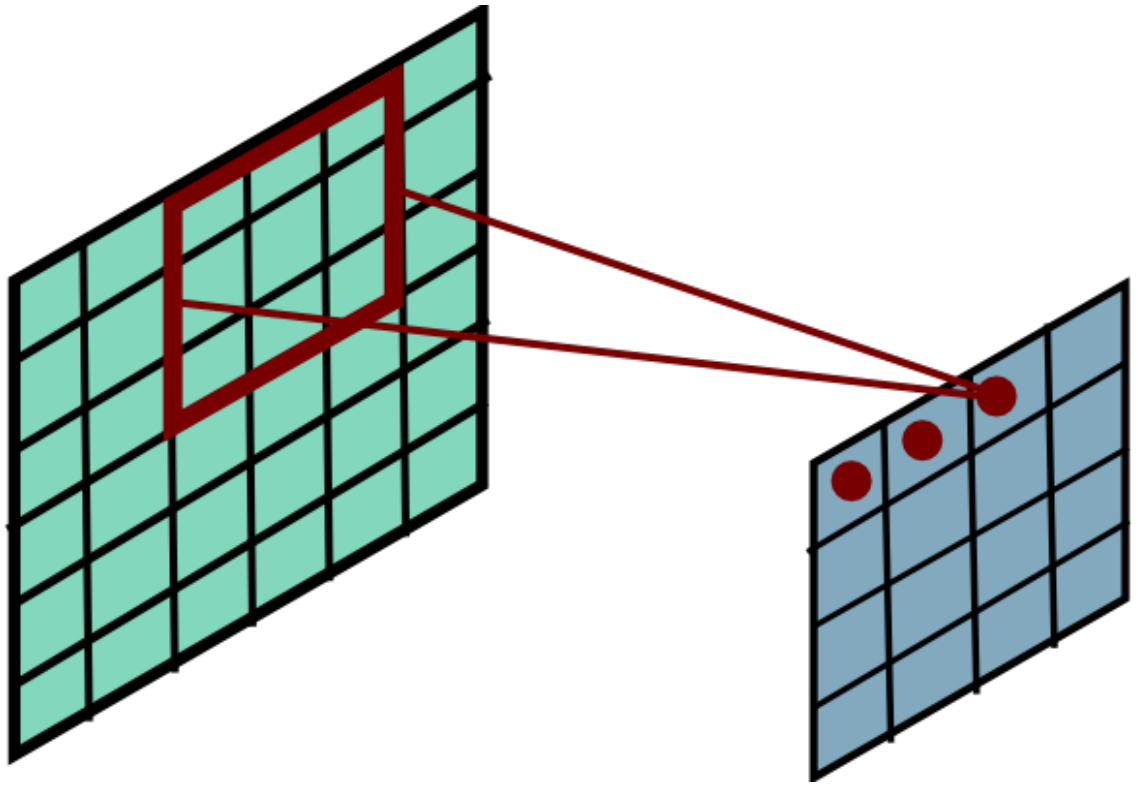
Convolutional Layer



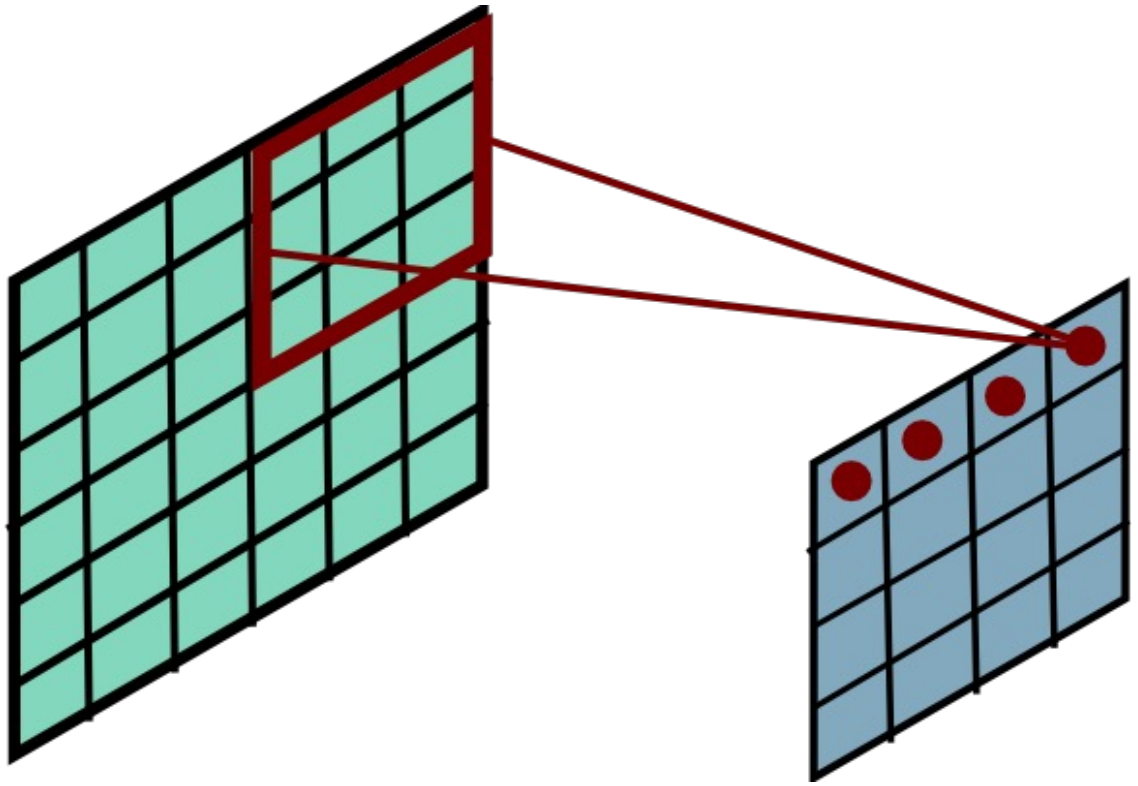
Convolutional Layer



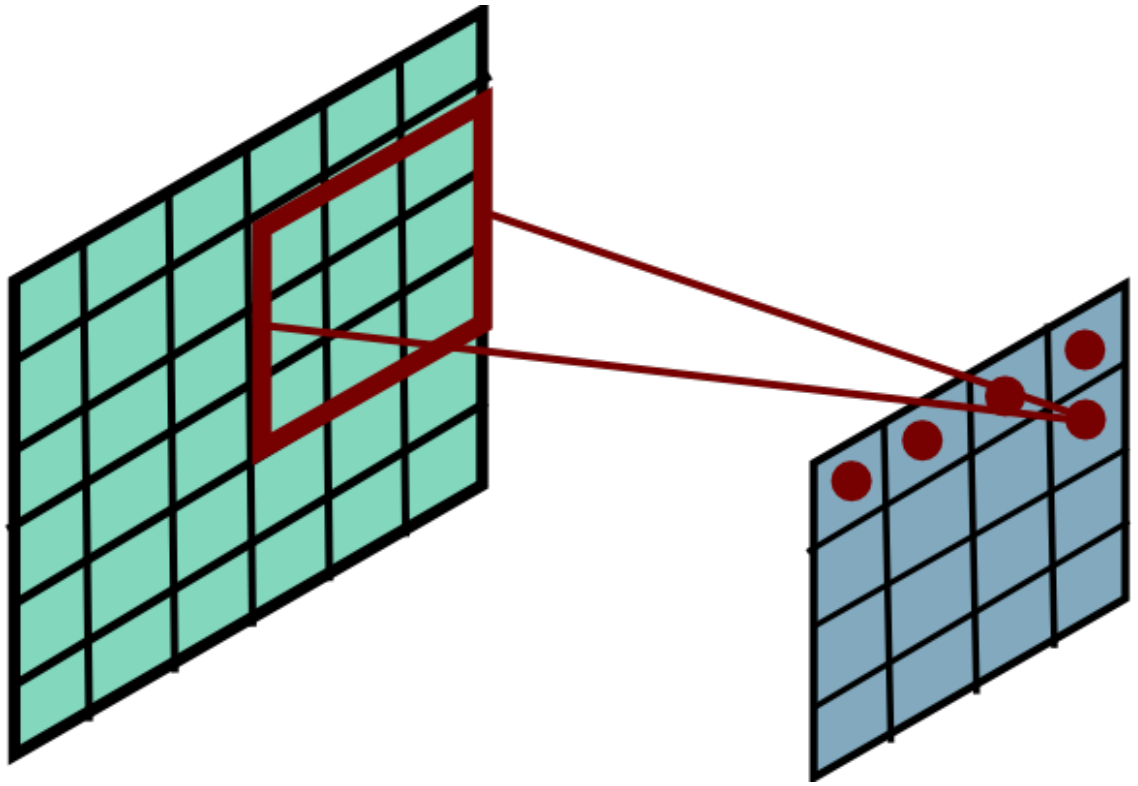
Convolutional Layer



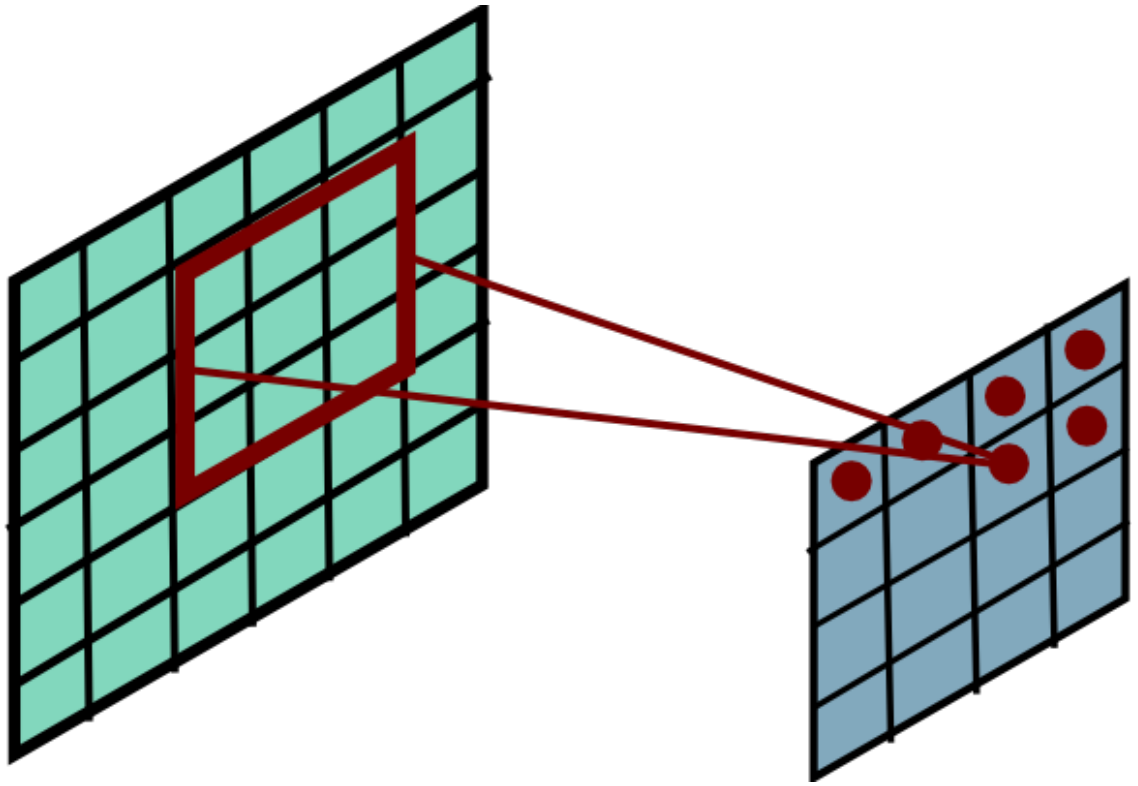
Convolutional Layer



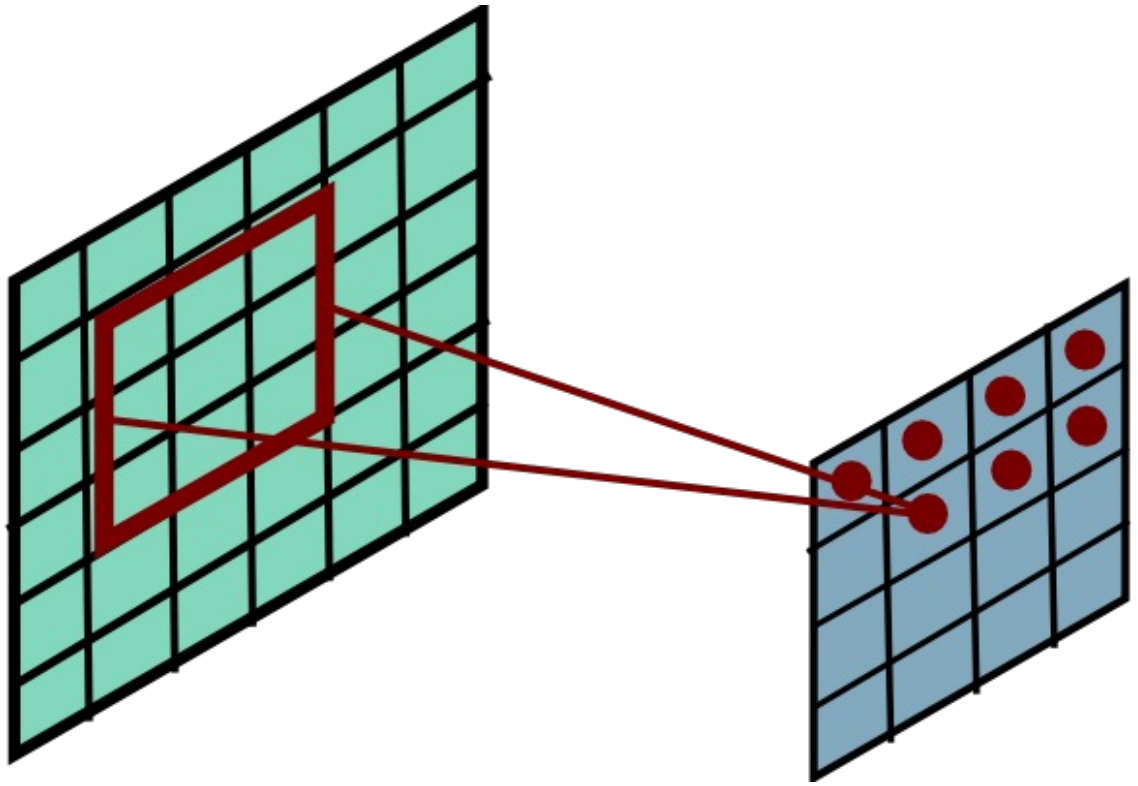
Convolutional Layer



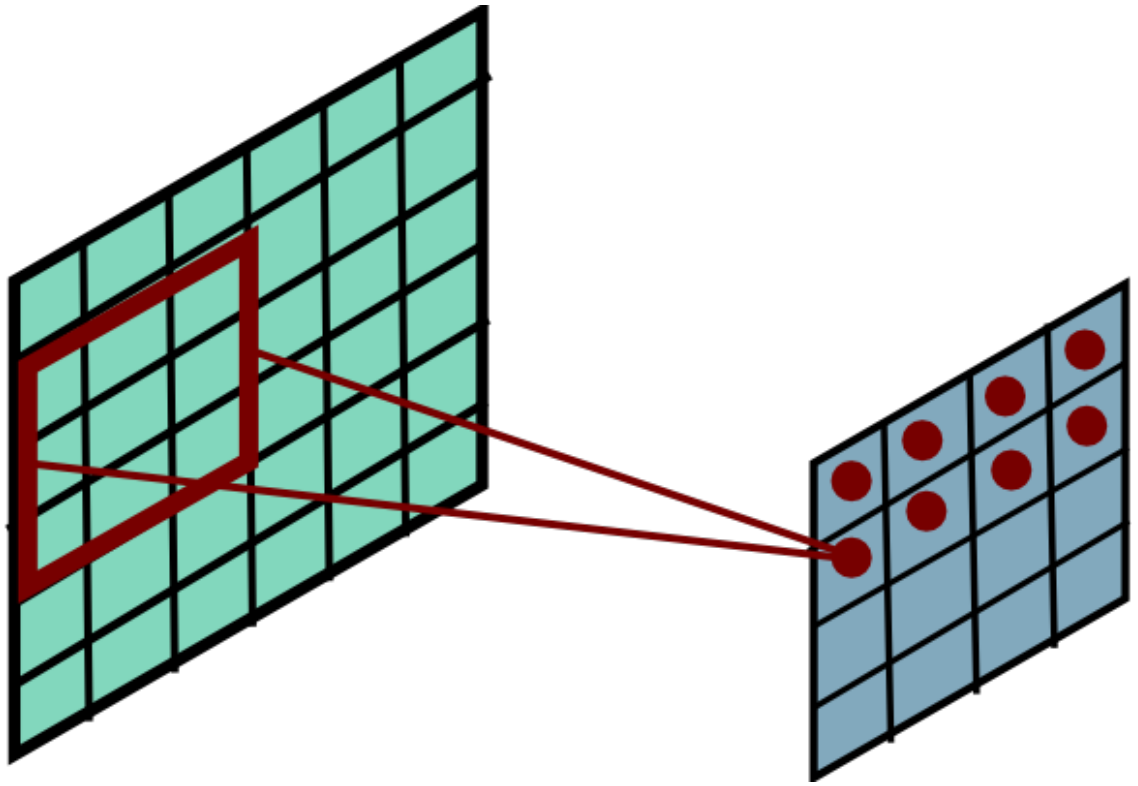
Convolutional Layer



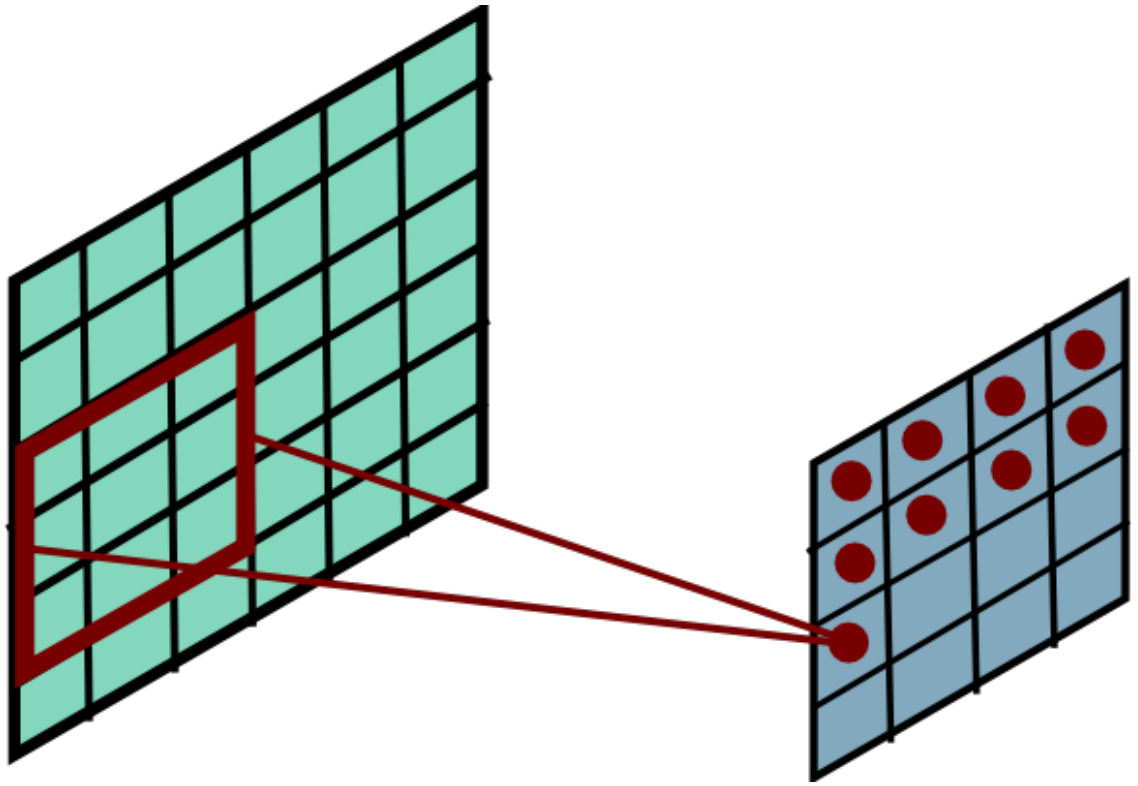
Convolutional Layer



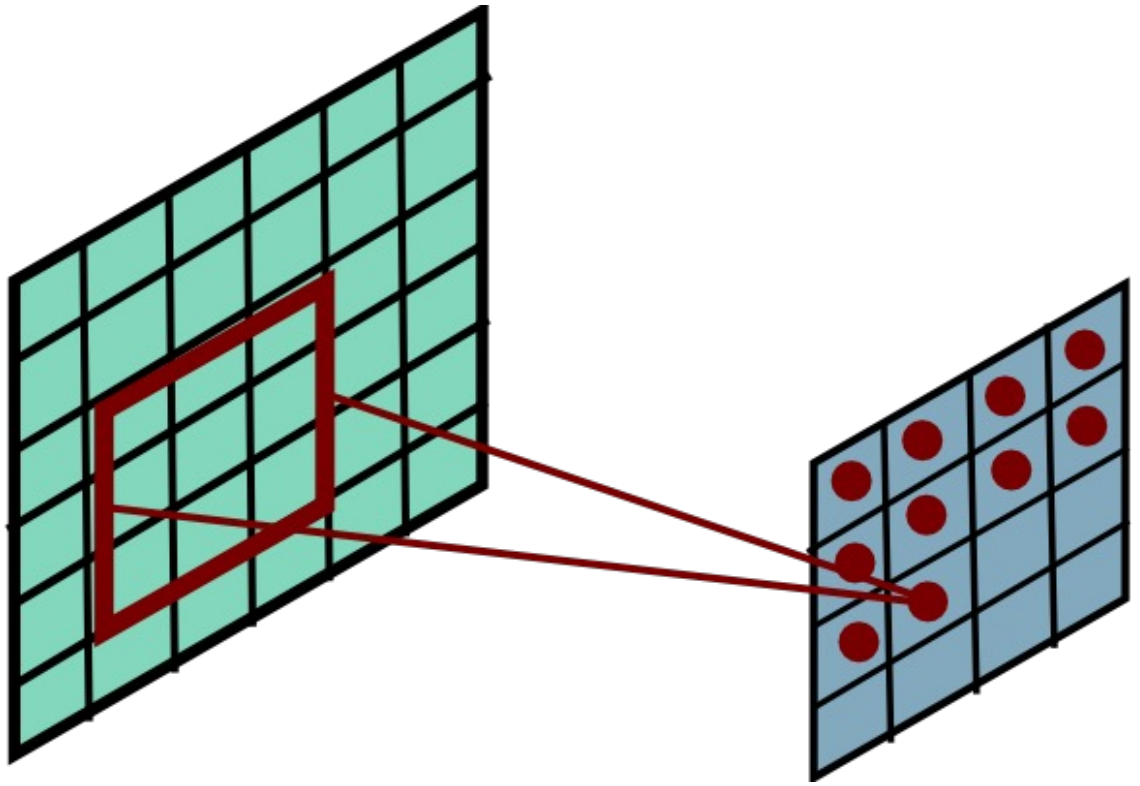
Convolutional Layer



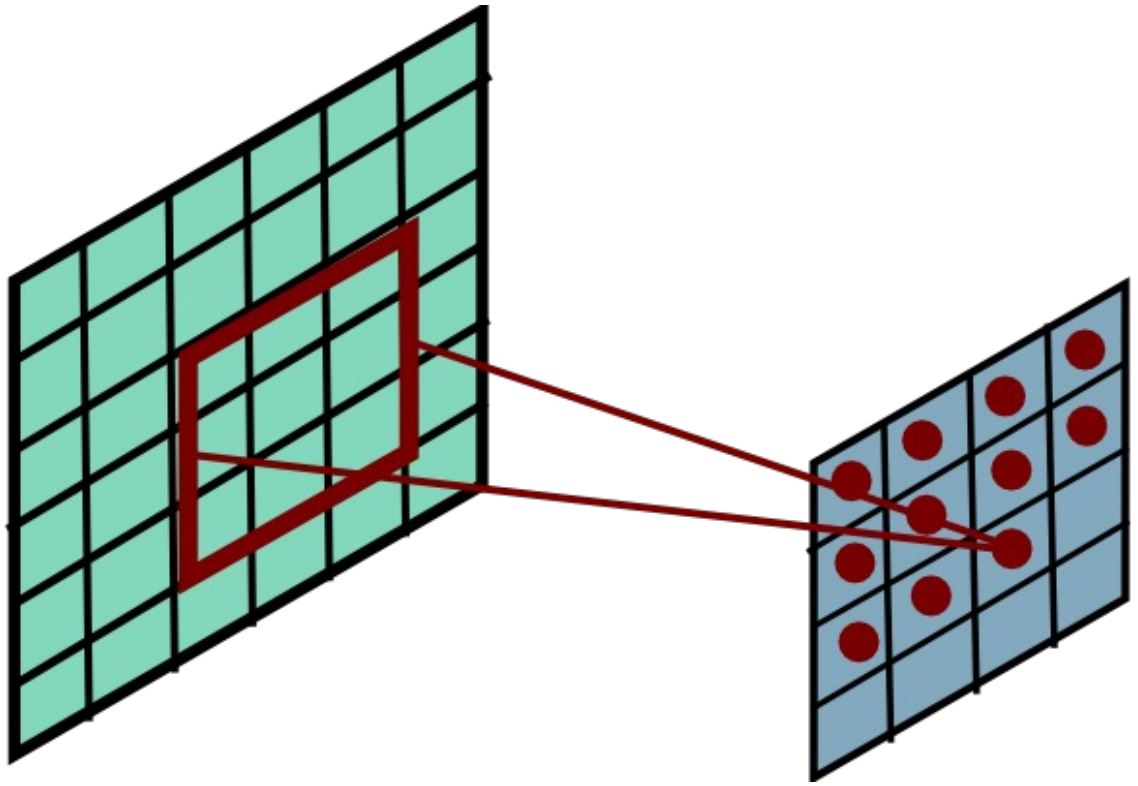
Convolutional Layer



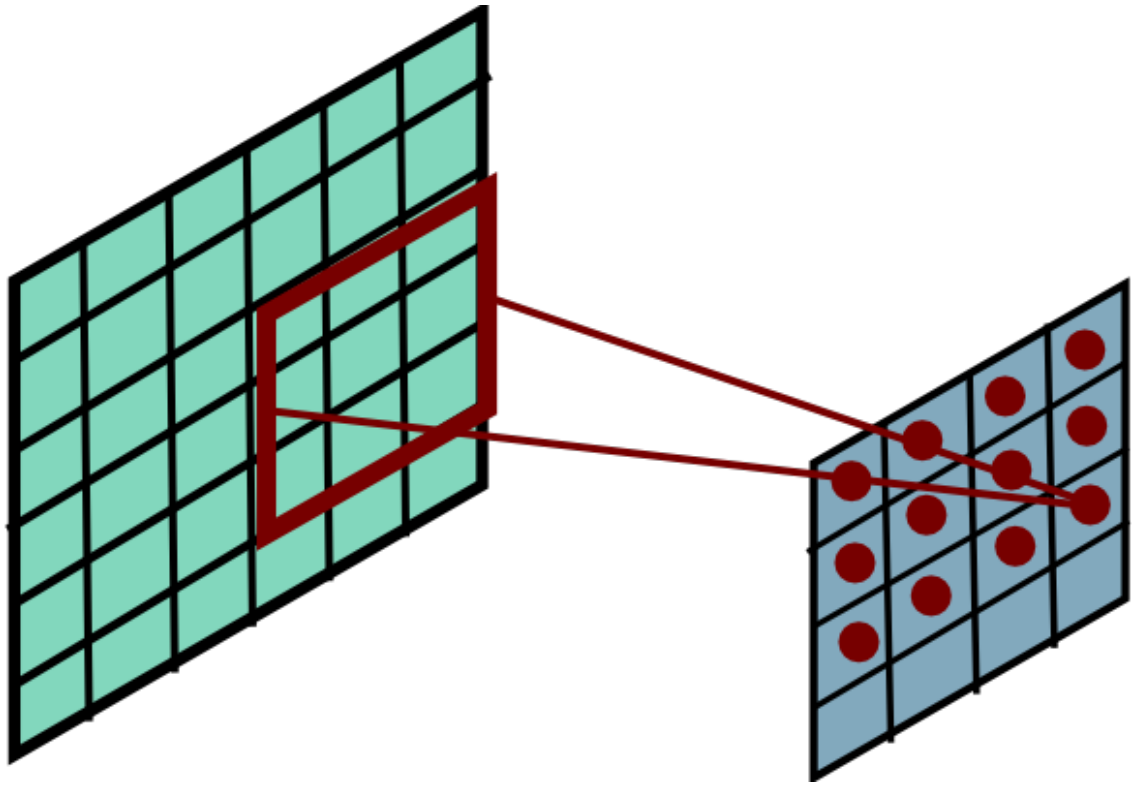
Convolutional Layer



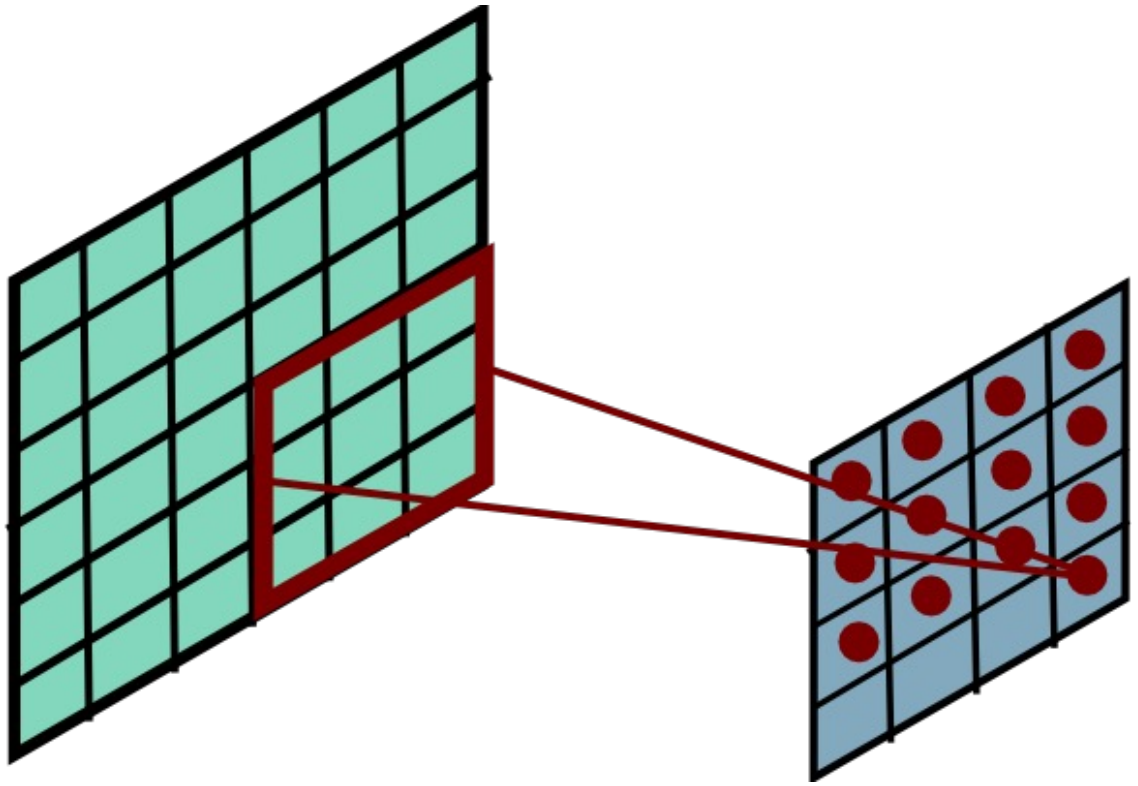
Convolutional Layer



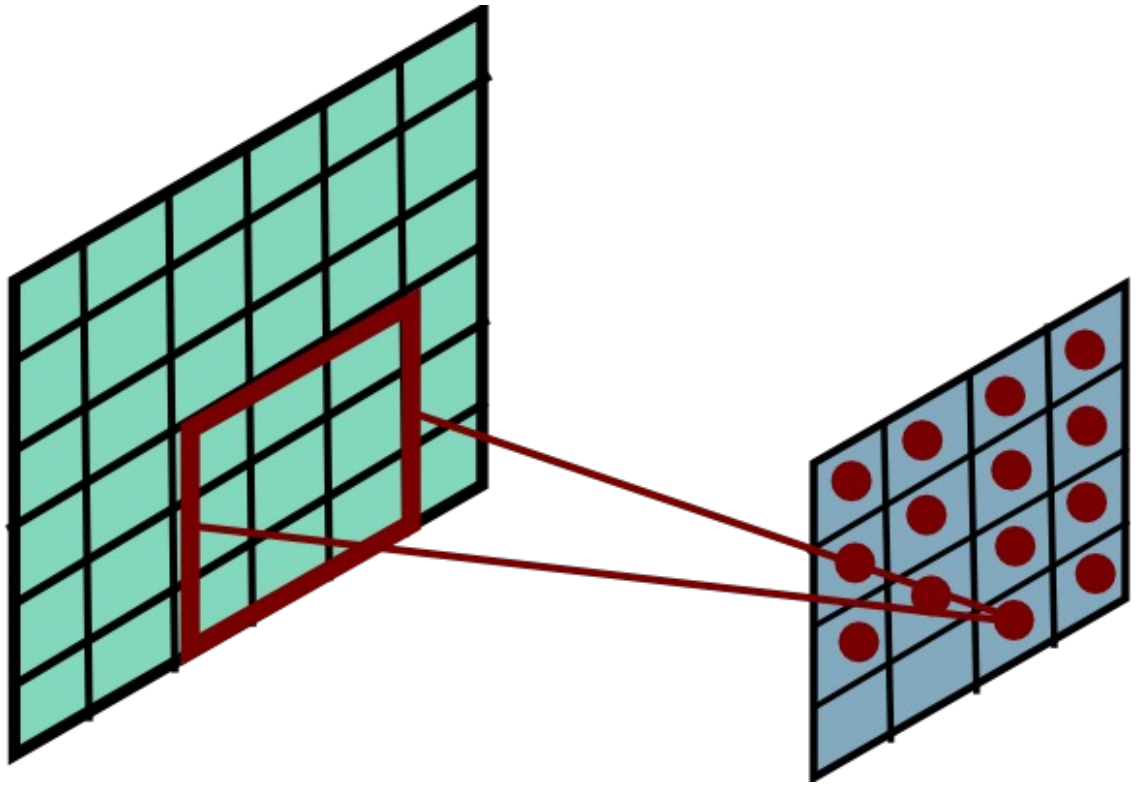
Convolutional Layer



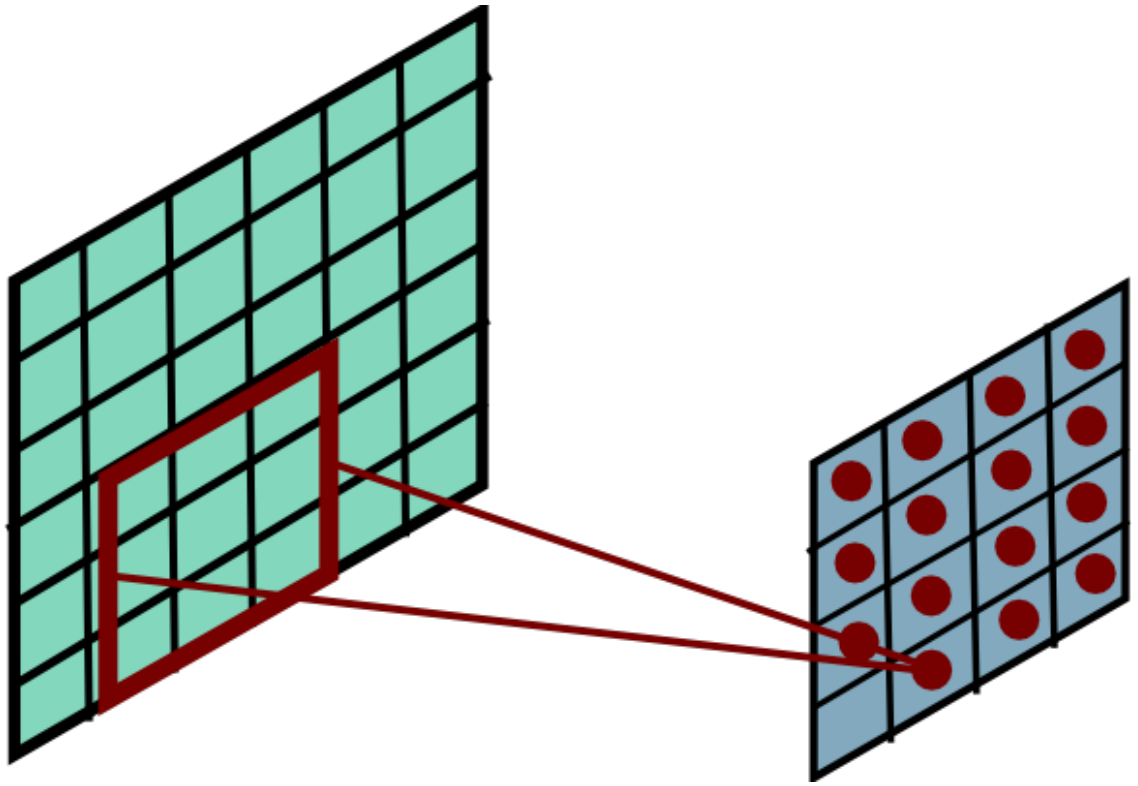
Convolutional Layer



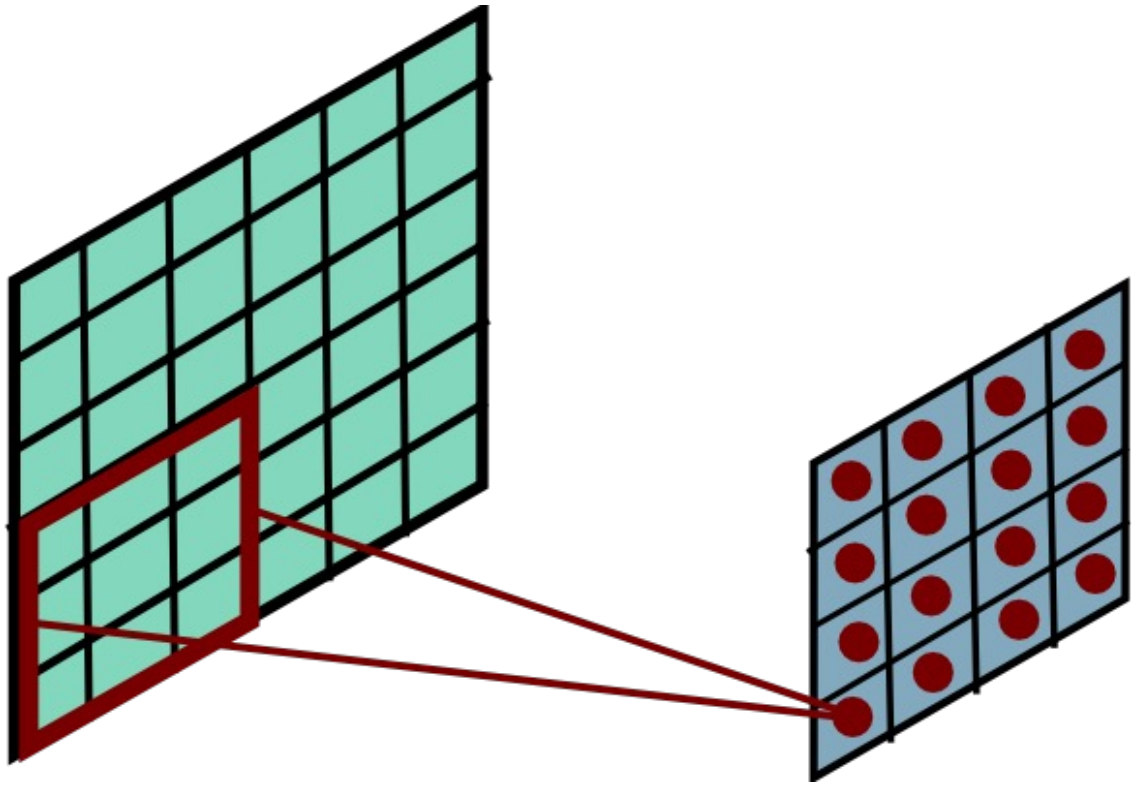
Convolutional Layer



Convolutional Layer



Convolutional Layer



Convolution Explained

- http://setosa.io/ev/image-kernels/
- https://github.com/bruckner/deepViz

Key Ideas

- Take advantage of properties of natural signals
  - Local connections
  - Shared weights
  - Pooling
  - Use of many layers

Comparison with Regular NNs

- Regular, Feed-forward NNs:
  - Need substantial number of training samples
  - Slow learning (convergence times)
  - Inadequate parameter selection techniques that lead to poor minima
- **Solution?**

Comparison with Regular NNs

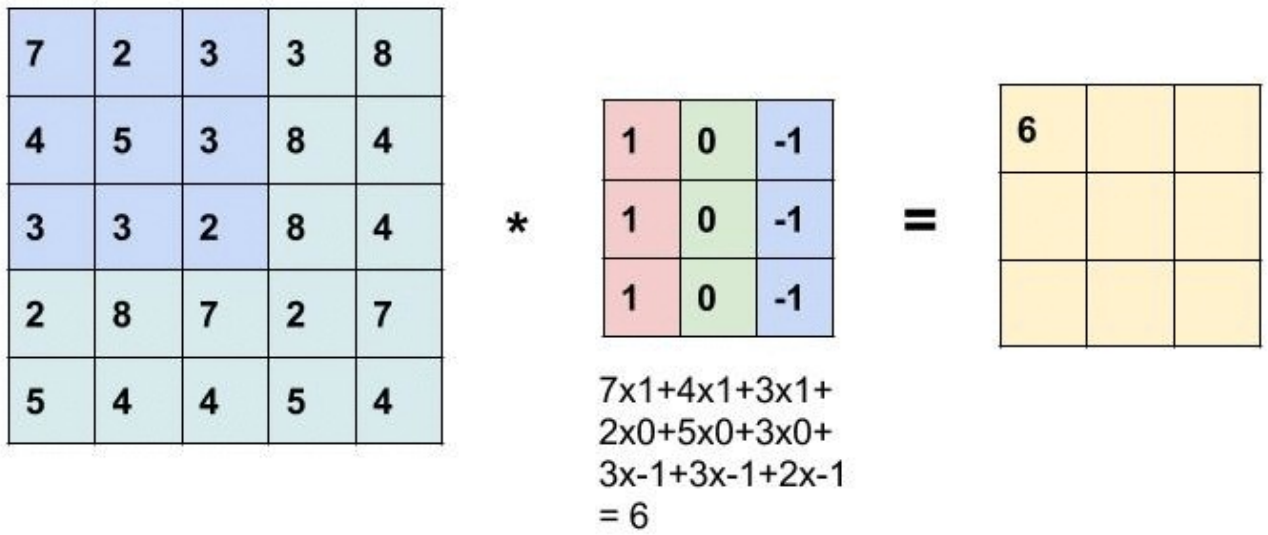
- Regular, Feed-forward NNs:
  - Need substantial number of training samples
  - Slow learning (convergence times)
  - Inadequate parameter selection techniques that lead to poor minima
- **Solution?**
- **Exploitation of Local Properties!**
- Network should exhibit invariance to translation, scaling and elastic deformations
  - A large training set can take care of this
- It ignores a key property of images
  - Nearby pixels are more strongly correlated than distant ones
  - Modern computer vision approaches exploit this property
- Information can be merged at later stages to get higher order features and about whole image

Basic Mechanisms in CNNs

- Three Mechanisms of Convolutional Neural Networks:
  - Convolution Operation
  - Local Receptive Fields
  - Subsampling
  - Weight (Parameter) Sharing

CNN Vocabulary

- Padding
- Stride
- Pooling



Visualization of convolution: https://github.com/effat/MLP-Demo/blob/main/Convolution_gif.gif

https://medium.com/@Tms43/understanding-padding-strides-in-convolutional-neural-networks-cnn-for-effective-image-feature-1b0756a52918

Stride

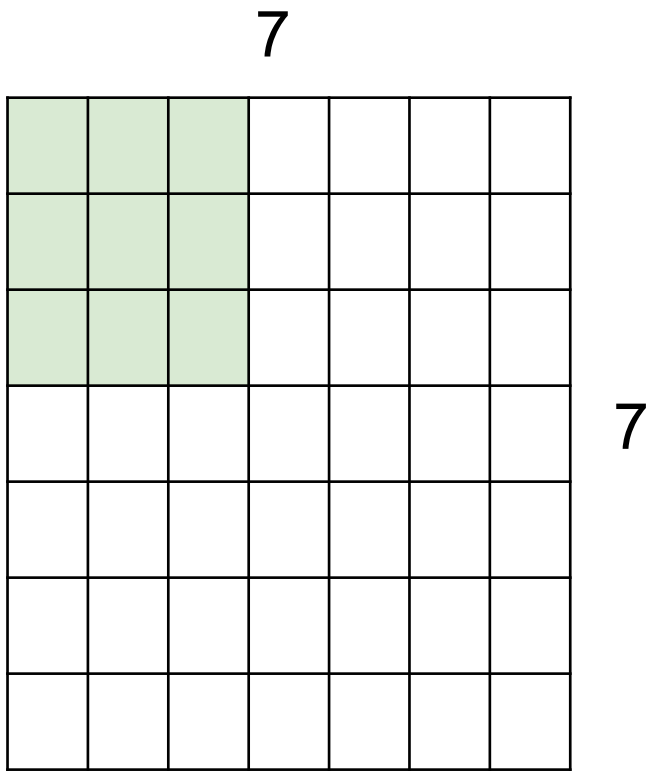
- Number of rows and columns traversed per slide as the *stride*
- Previous example
- Stride = 1 for both height and width

Stride Limitations

1. The output feature map is smaller (3x3) than the input image (5x5).
    1. Applying more convolutional layers → the spatial dimensions decreases→ loss of information.

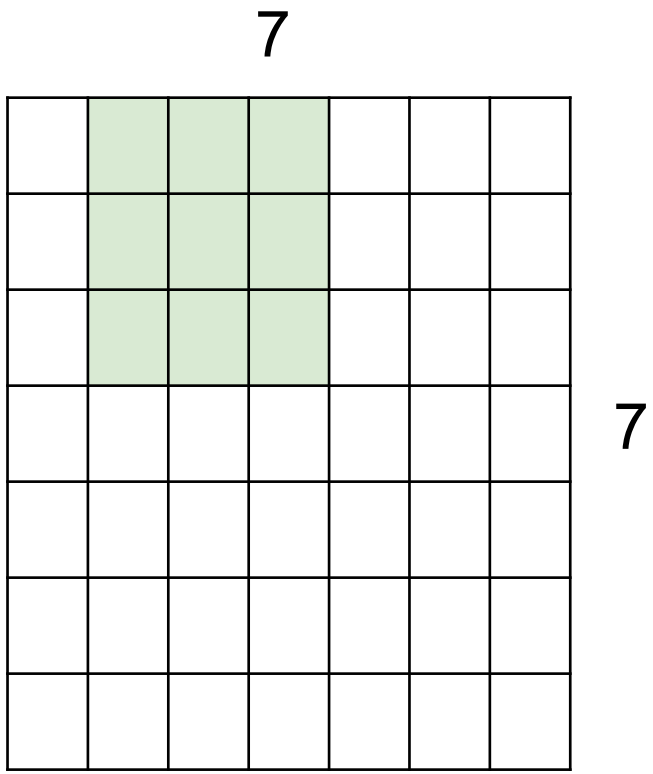
2. The pixels at the borders of receives fewer convolutional operations compared to the center pixels.
    1. Can impact model's performance if important features are present at the edges.

A closer look at spatial dimensions:



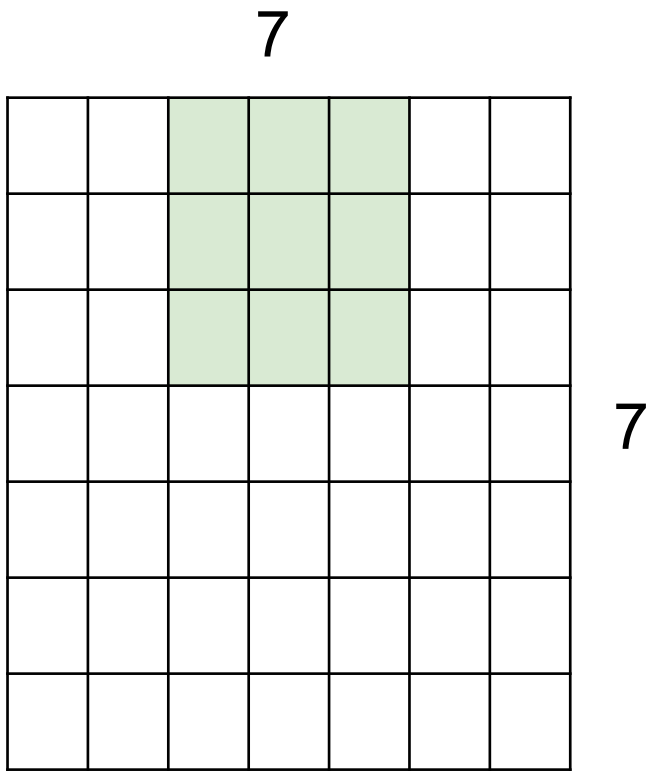
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



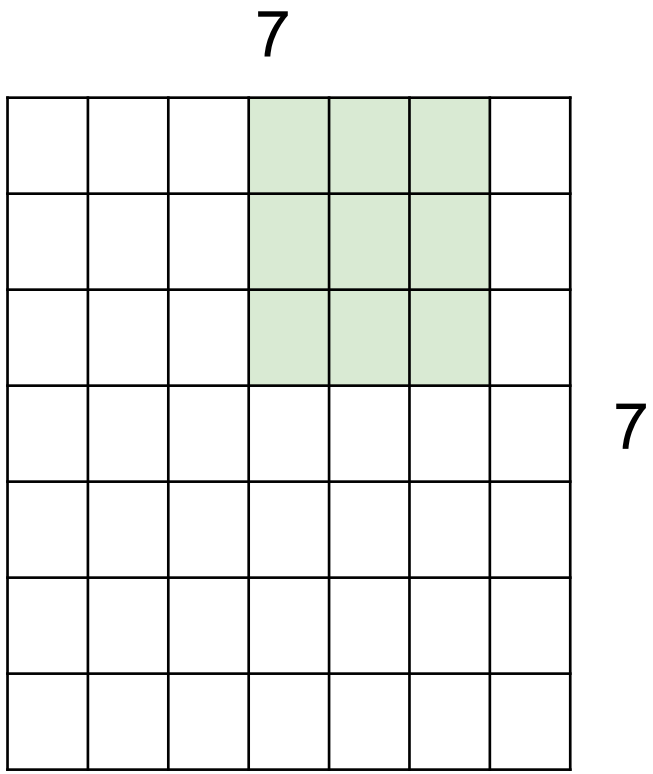
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



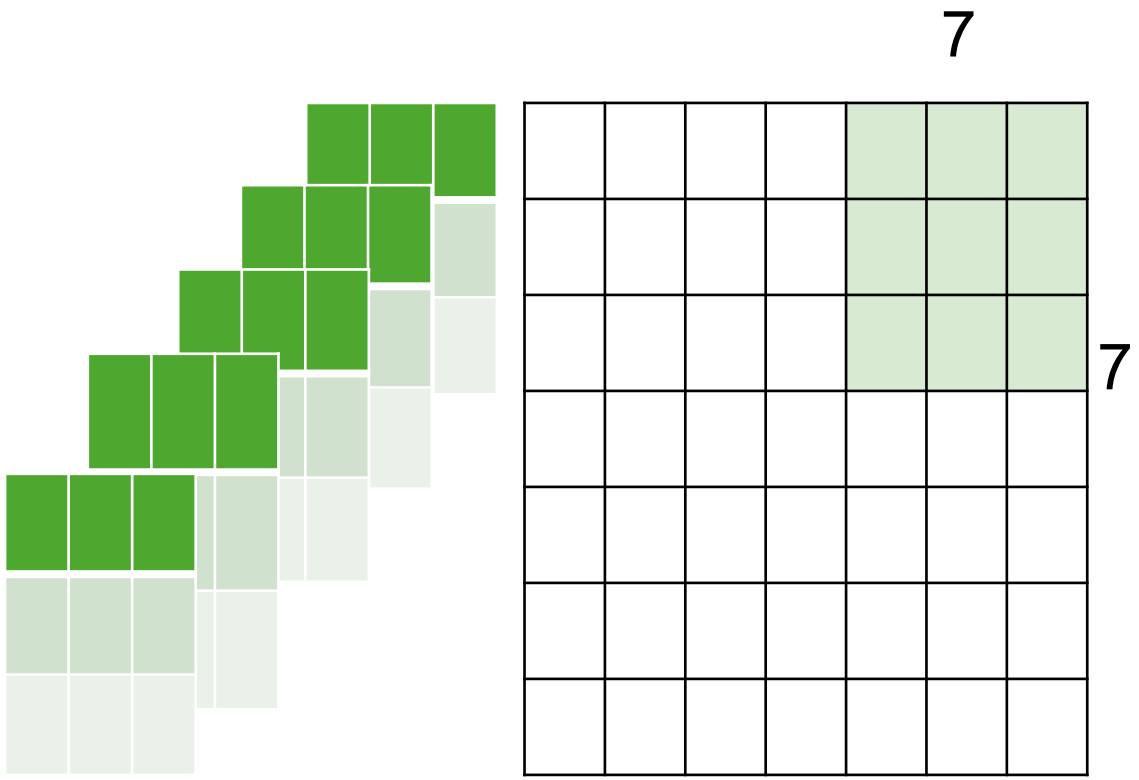
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



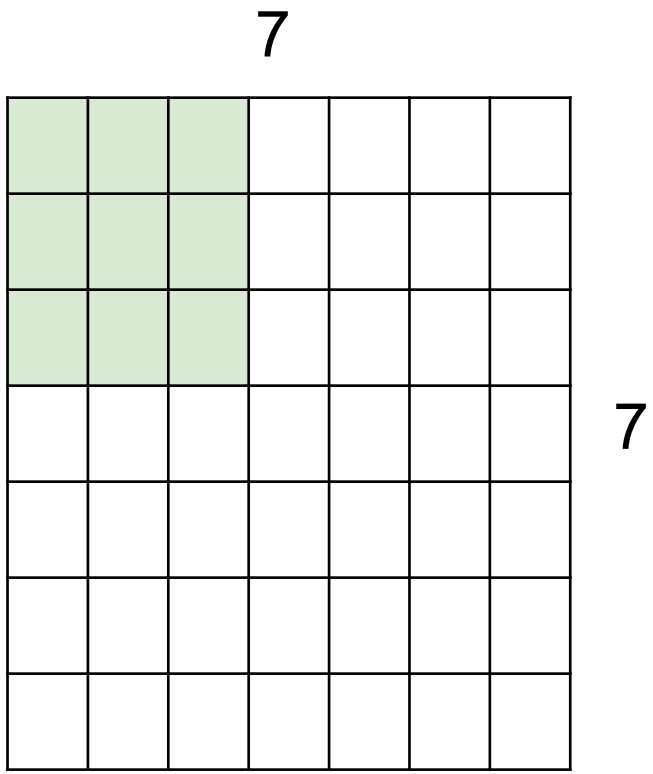
7x7 input (spatially)
assume 3x3 filter

⇒ **5x5 output**

⇒ **With stride =1, takes 5 times sliding to traverse the width**

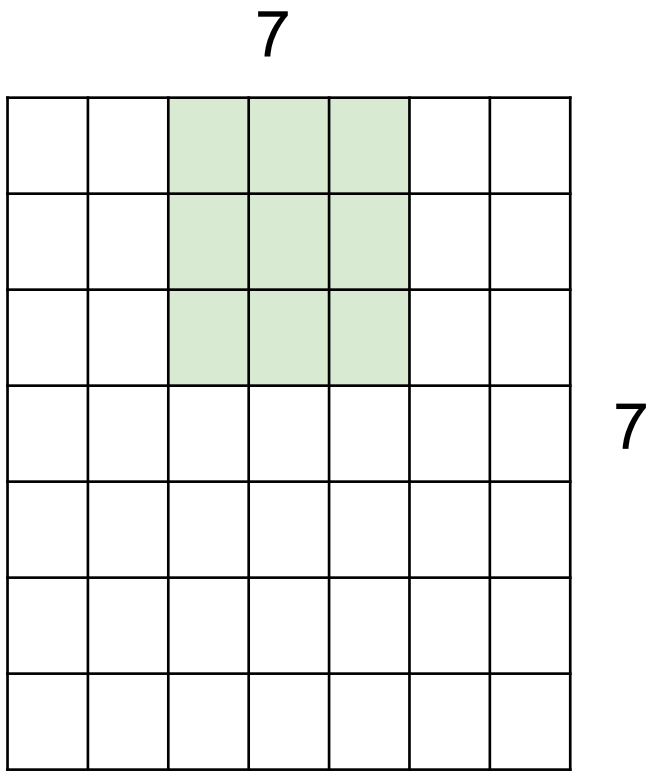
⇒ **3 x 3 filter will need 5 times sliding to cover the height with stride = 1**

A closer look at spatial dimensions:



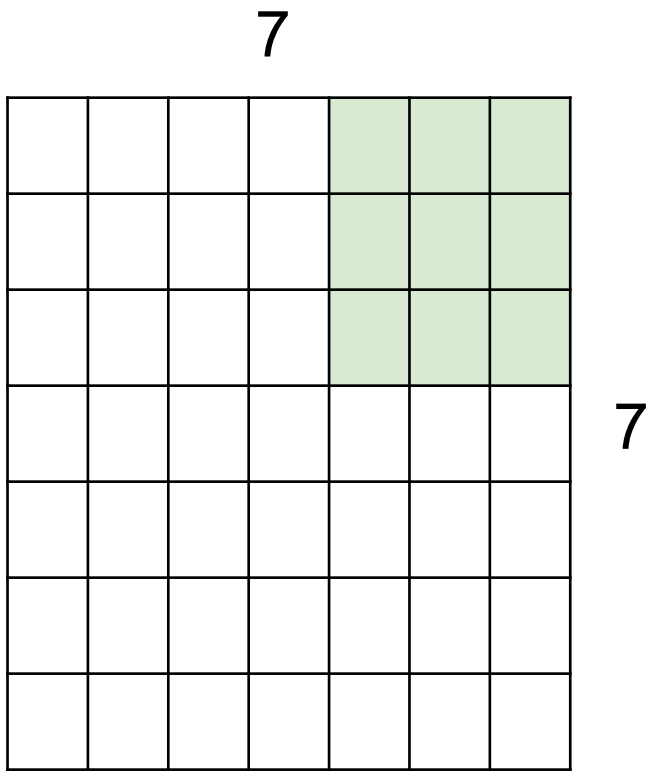
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



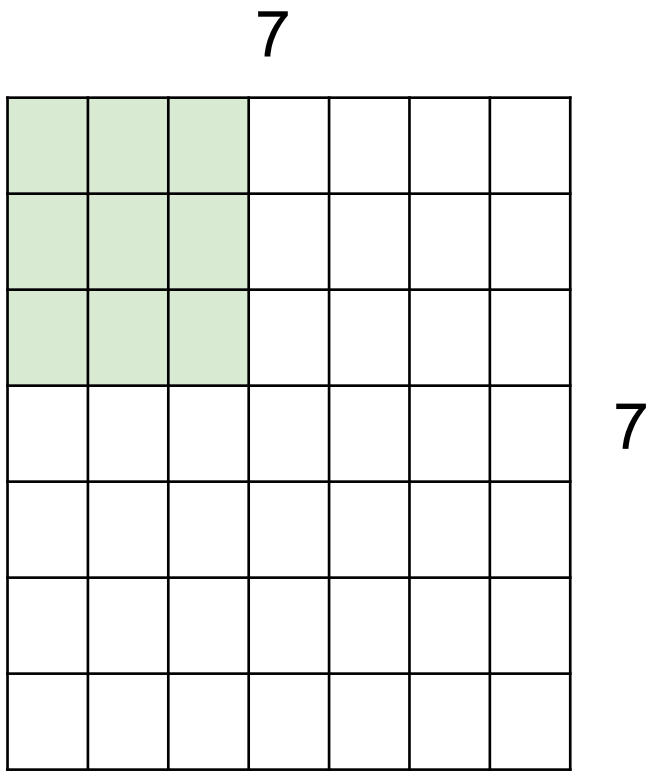
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



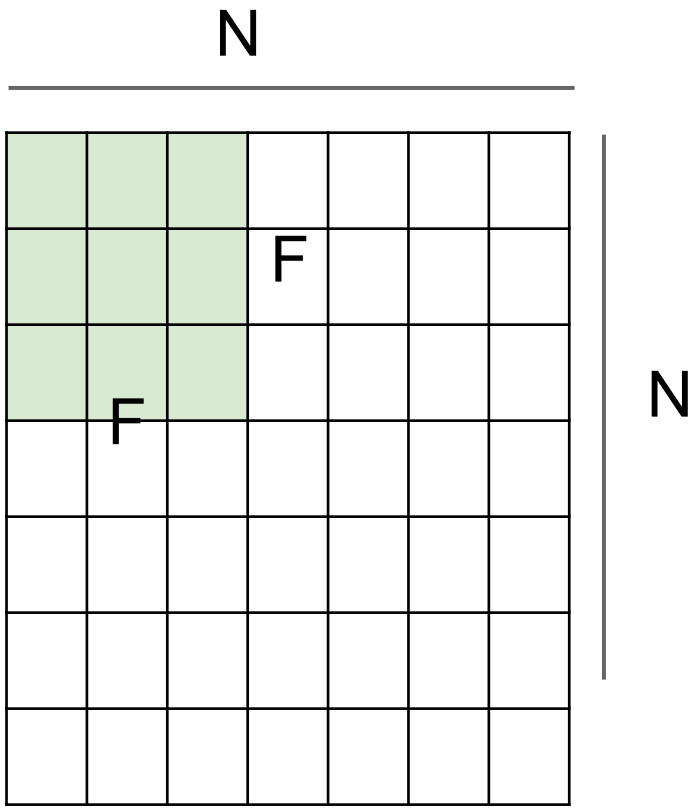
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

doesn't fit!
cannot apply 3x3 filter on
7x7 input with stride 3.



Assume input image height (== width) = N
Applied stride's height(== width) = F

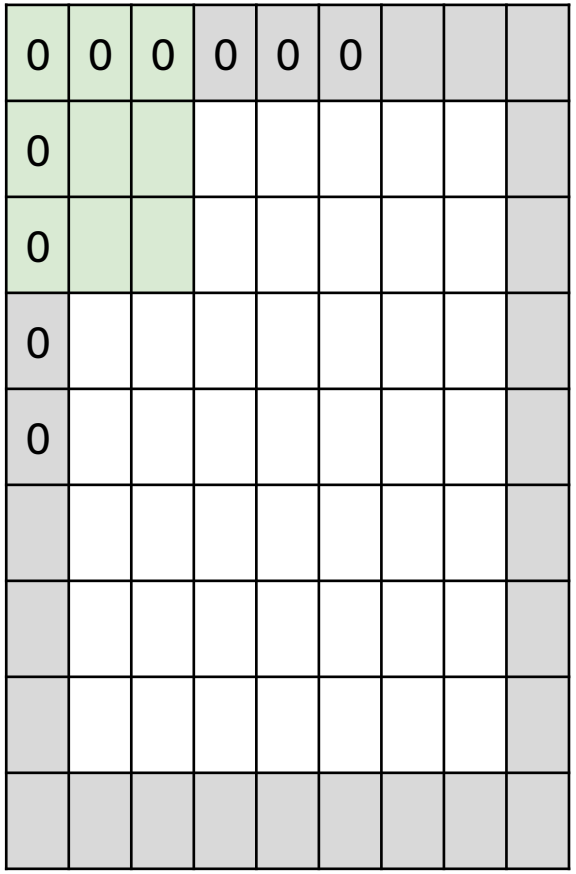
Output width (==height)
(N - F) / stride + 1

e.g. N = 7, F = 3:
stride 1 => (7 - 3)/1 + 1 = 5
stride 2 => (7 - 3)/2 + 1 = 3
stride 3 => (7 - 3)/3 + 1 = 2.33 :\

Padding the Input Before Convolution

- Solution
  - Add extra pixels of filler around the boundary of our input image, thus increasing the effective size of the image.
  - Set values of extra pixels with zeros

In practice: Common to zero pad the border



e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!
in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)
e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

Stride, Padding, and the Output Volume

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.

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Stride, Padding, and the Output Volume

Common settings:

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.

K = (powers of 2, e.g. 32, 64, 128, 512)

- F = 3, S = 1, P = 1
- F = 5, S = 1, P = 2
- F = 5, S = 2, P = ? (whatever fits)
- F = 1, S = 1, P = 0

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n