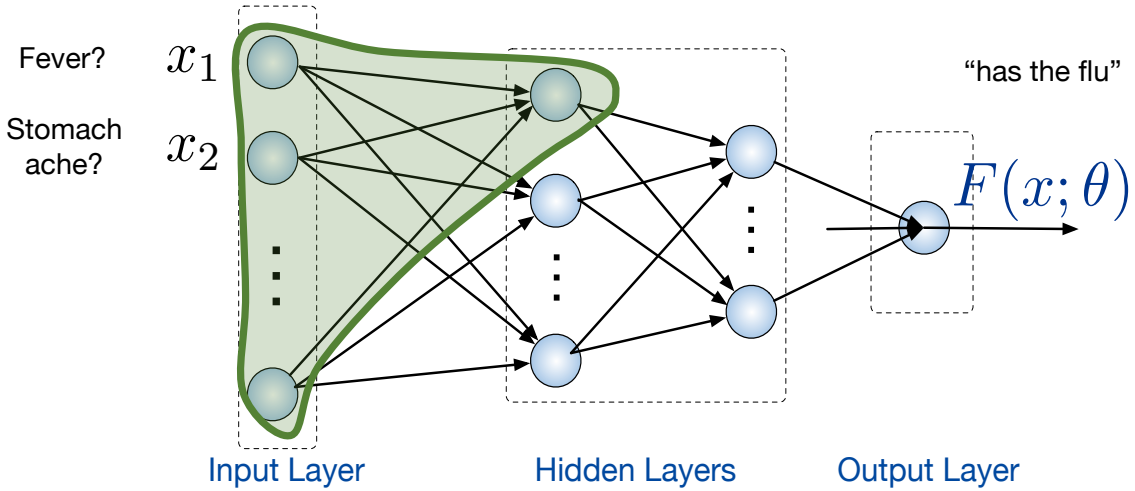
Massachusetts
Institute of
Technology

Deep Learning for Images:
Convolutional Neural Networks

Stefanie Jegelka
MIT

Monday: Fully connected NN



input data point

- Hierarchy of simple "detectors": increasingly complex decisions
- One unit = linear classifier:

$$\text{output} = \text{nonlinearity}\left(\sum_{j=1}^{k} x_j w_j + b\right)$$

e.g. ReLu, sigmoid

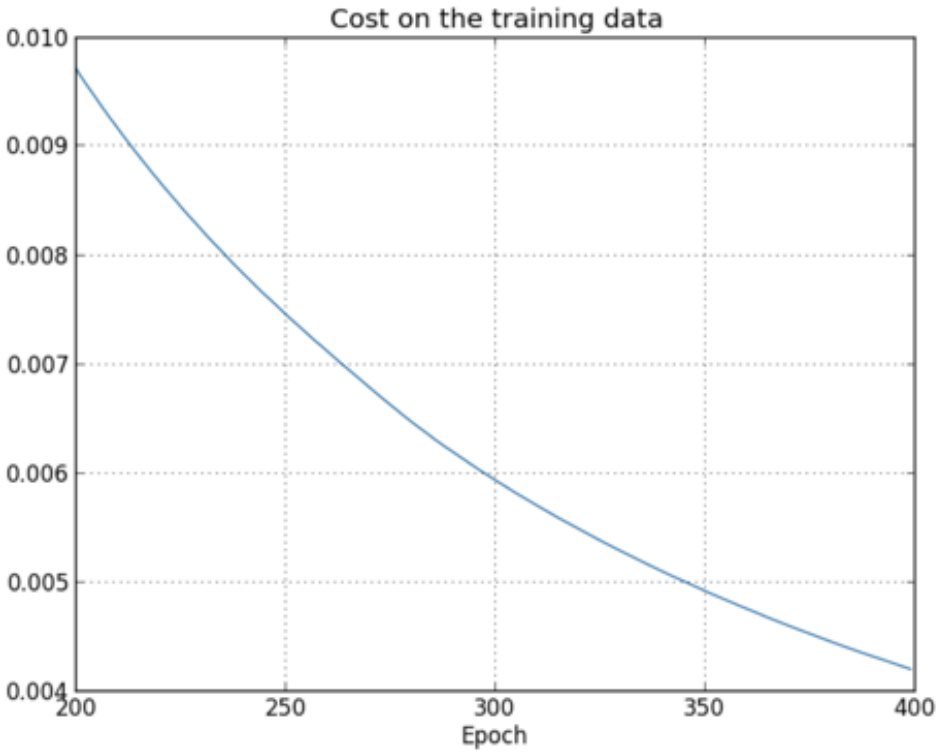
Input: data x or output of previous layer

Learned weights and bias

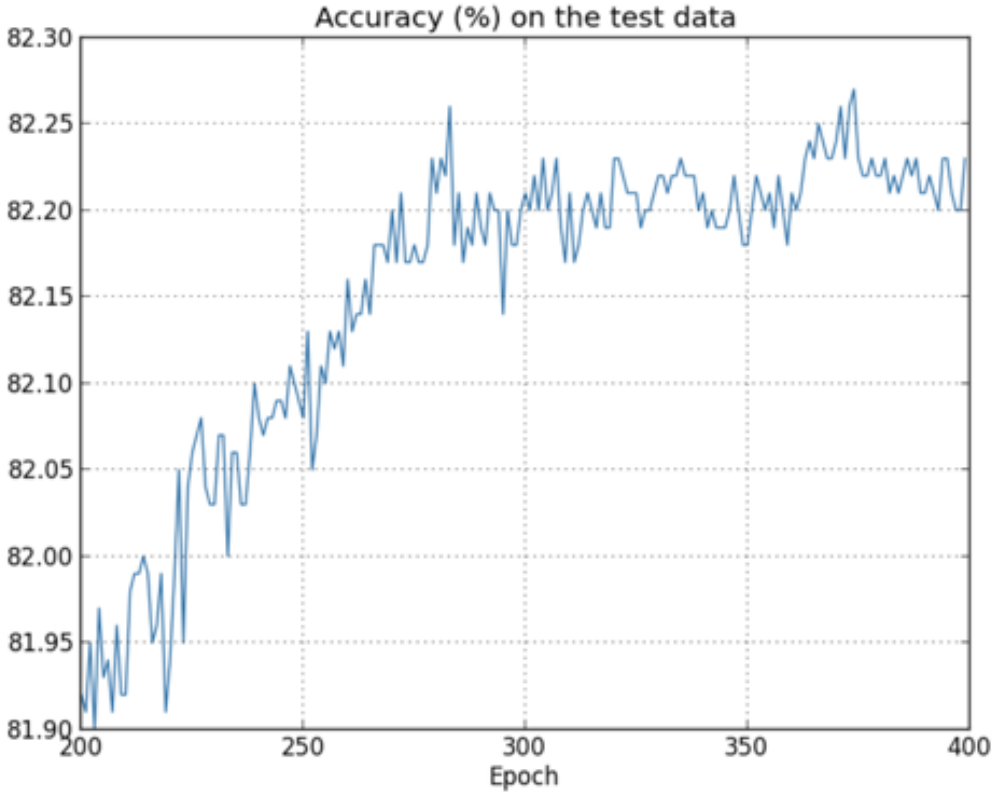
- Trained by stochastic gradient descent

Regularization

Loss on the training data



Accuracy on the test data



http://neuralnetworksanddeeplearning.com/chap3.html

Regularization

- **Squared norm / weight decay**
  add $+\frac{\lambda}{2}\|\theta\|^2$ to training loss

$$\sum_{j=1}^{d} \theta_j^2$$

Regularization

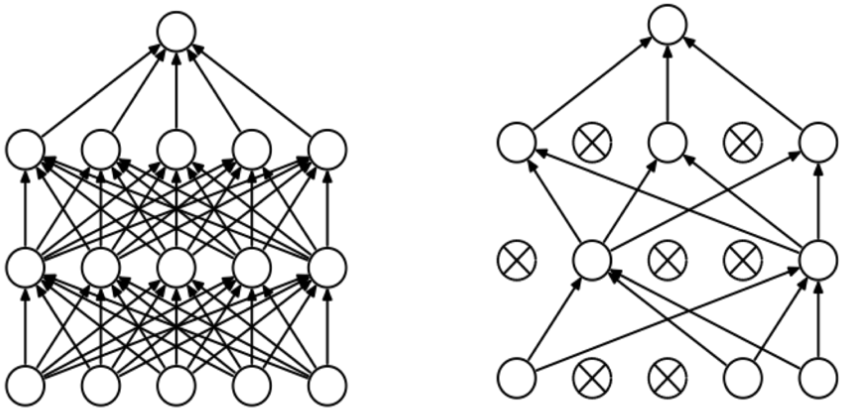
- **Squared norm / weight decay**
  add $+\frac{\lambda}{2}\|\theta\|^2$ to training loss
- **Early stopping**
  Check error on validation set after each epoch

Regularization

- **Squared norm / weight decay**
  add $+\frac{\lambda}{2}\|\theta\|^2$ to training loss
- **Early stopping**
  Check error on validation set after each epoch
- **Data augmentation**
  Perturbations (rotation, noise,…), averaging

Regularization

- **Squared norm / weight decay**
  add $+\frac{\lambda}{2}\|\theta\|^2$ to training loss
- **Early stopping**
  Check error on validation set after each epoch
- **Data augmentation**
  Perturbations (rotation, noise,…), averaging
- **Dropout**
  for each training data point, randomly switch off ½ of the units, don't update them either;
  at test time, scale weights by ½



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

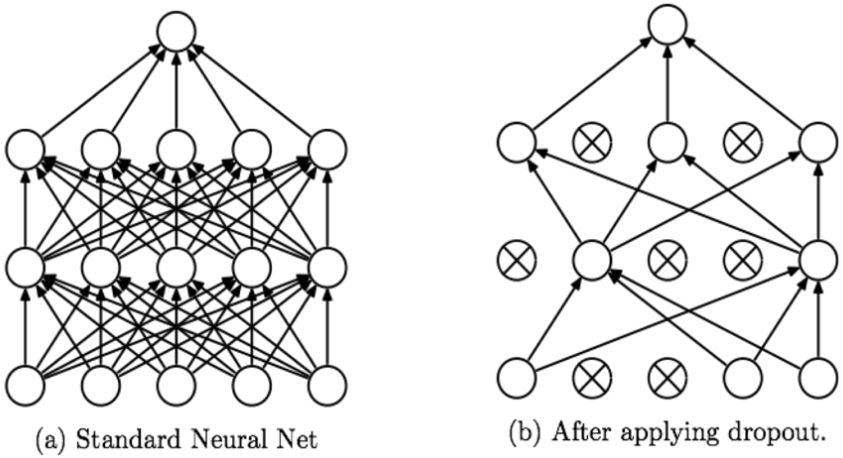
Figure: (Srivastava et al. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. JMLR, 2014)

Regularization

$\nabla(\text{loss} + \frac{\lambda}{2}\|\theta\|^2) =$

$\nabla \text{loss} - \lambda\theta$

- **Squared norm / weight decay**
  add $+\frac{\lambda}{2}\|\theta\|^2$ to training loss
- **Early stopping**
  Check error on validation set after each epoch
- **Data augmentation**
  Perturbations (rotation, noise,…), averaging
- **Dropout**
  for each training data point, randomly switch off ½ of the units, don't update them either;
  at test time, scale weights by ½
- **Batch normalization**
  normalize inputs of each layer



(a) Standard Neural Net

(b) After applying dropout.

Figure: (Srivastava et al. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. JMLR, 2014)

Specialized methods to encode

$\phi(\text{image})$

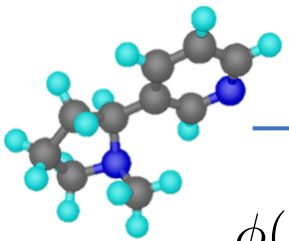
category

CNN (today)

RNN

$\phi(\text{text})$

sentiment



$\phi(\text{molecule})$

property

GNN (Fri)

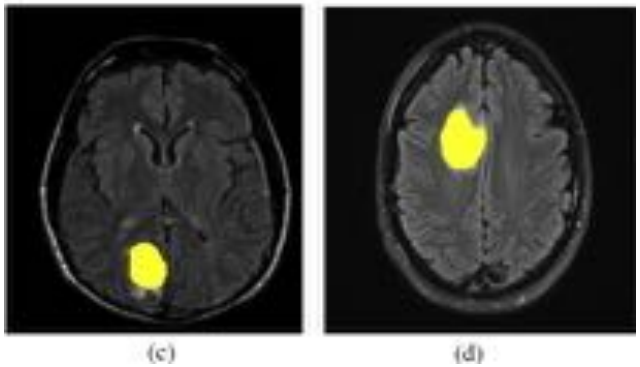
Outline

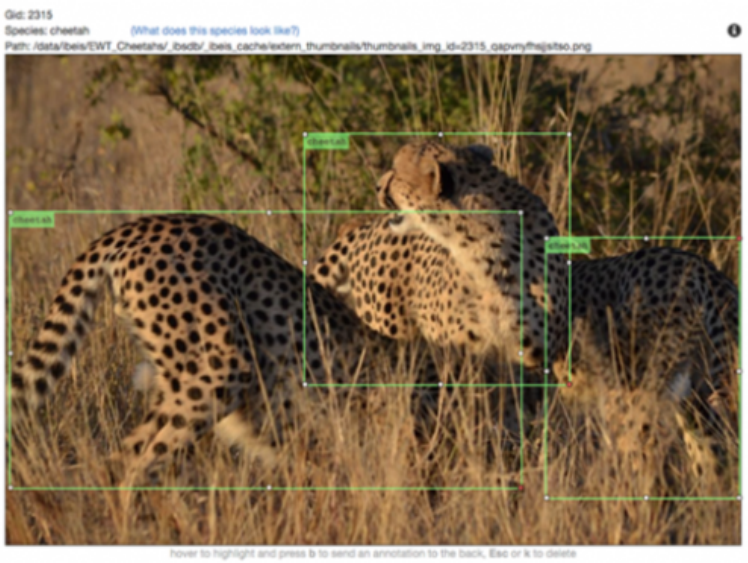
- **Mon: Feedforward neural networks**

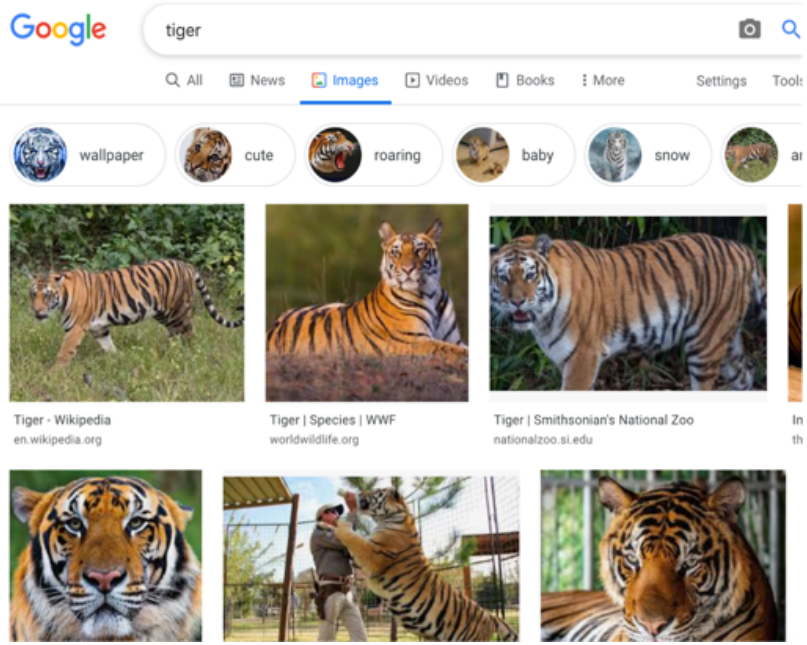
- **Today:** Neural networks for images: convolutional neural networks (CNNs)
  - Convolutions & Pooling
  - What do CNNs learn?
  - Improving training

- **Thu:** Neural networks for graphs

Image Inputs



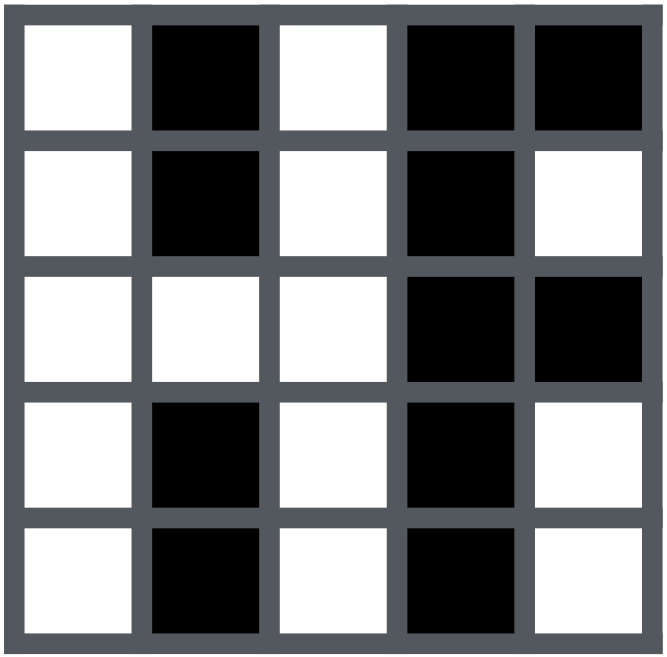


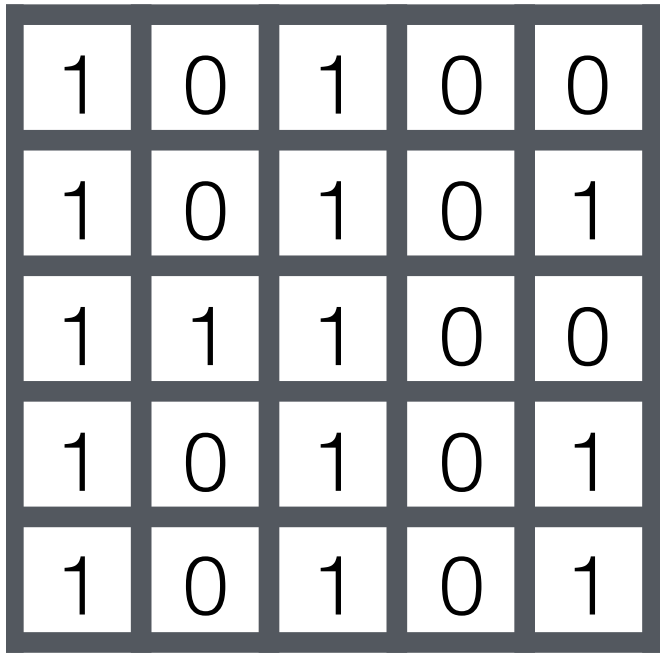


Outline: CNNs

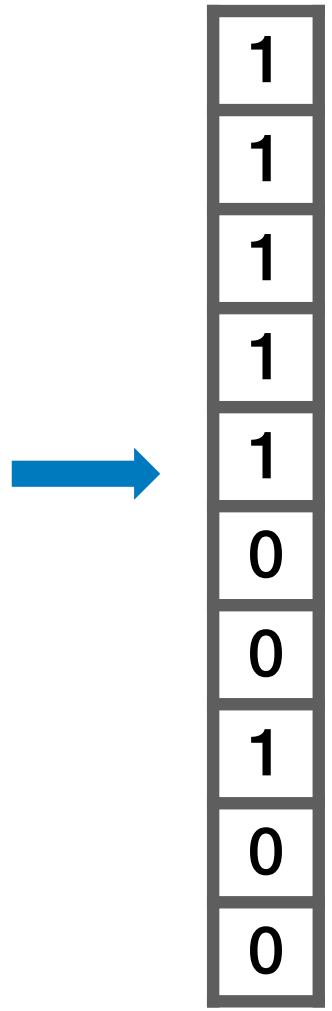
- Spatial locality & invariance
- Convolutional Neural Networks
  - Convolution and filters
  - Max Pooling
- CNN training
  - Illustrations: what do CNNs (not) learn?

Images are matrices (tensors)





1	0	1	0	0
1	0	1	0	1
1	1	1	0	0
1	0	1	0	1
1	0	1	0	1

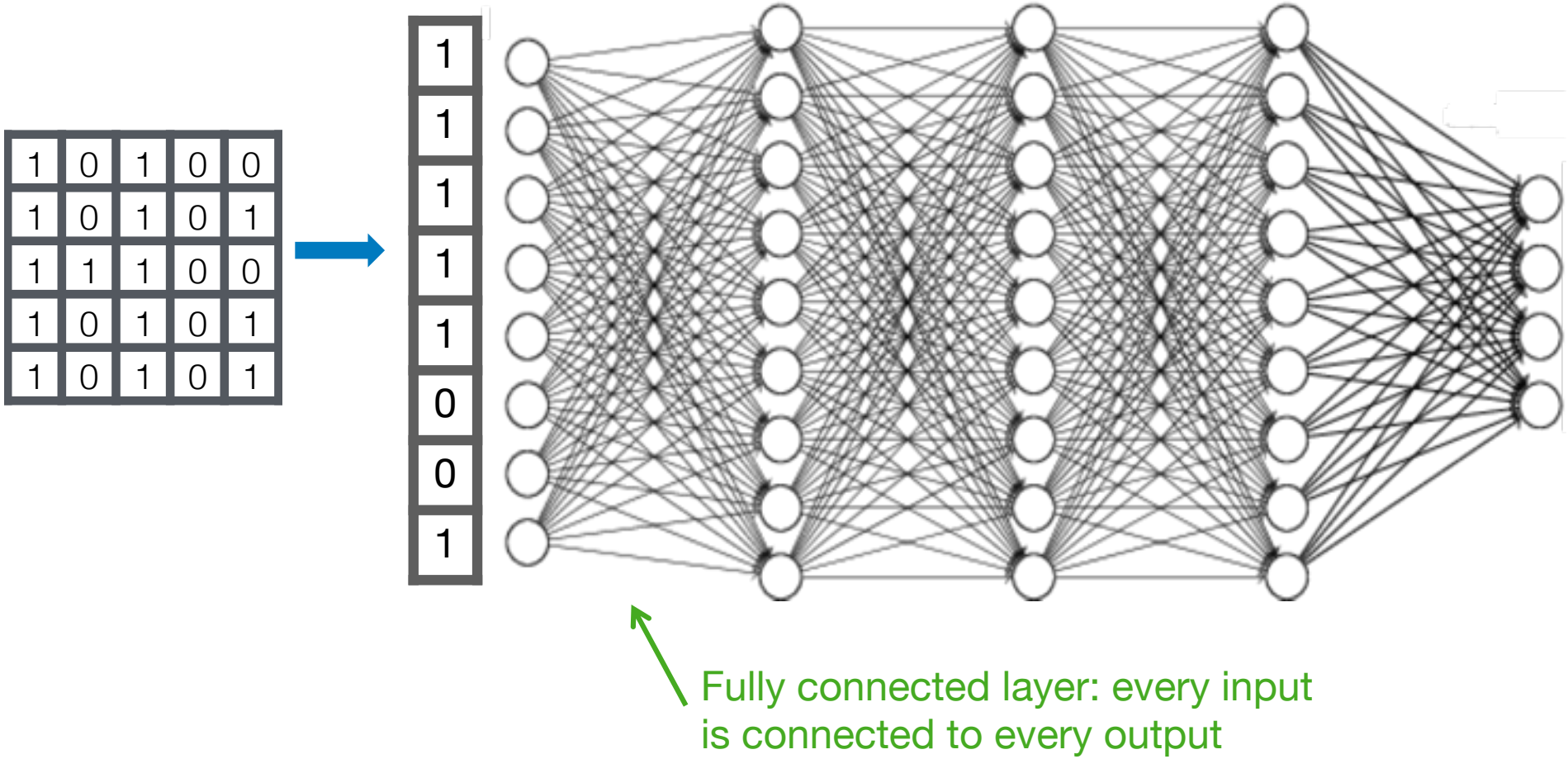


1
1
1
1
1
0
0
1
0
0
...

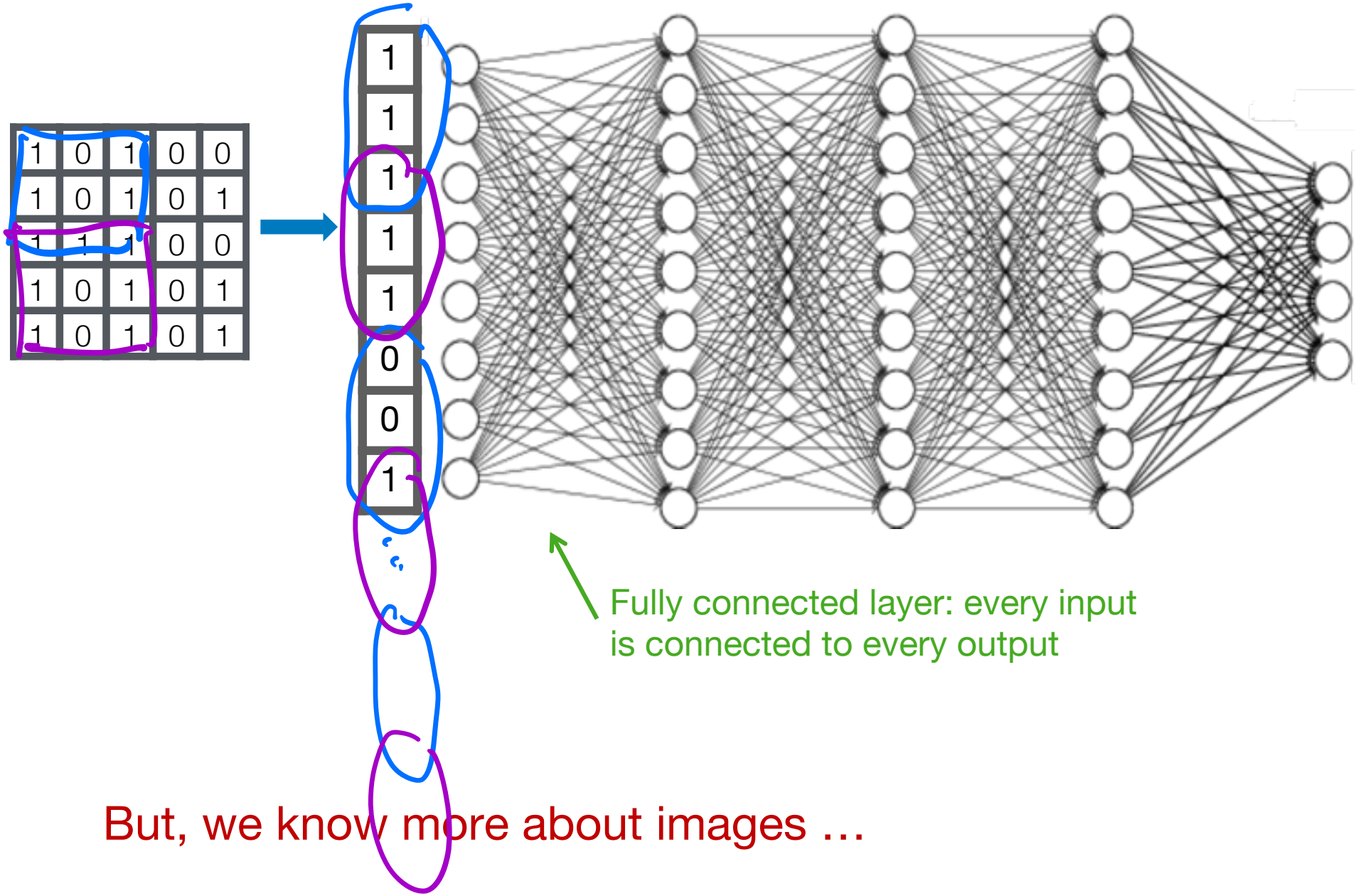
How do we input an image into a neural network?

So far: make it a vector and use fully connected network

Fully connected network?



Fully connected network?



Fully connected layer: every input is connected to every output

But, we know more about images …

Locality and translation invariance









Locality and translation invariance









- Spatial locality: objects occur in local patches
- Translation invariance: a shifted cup is still a cup

Locality and translation invariance









Locality and translation invariance

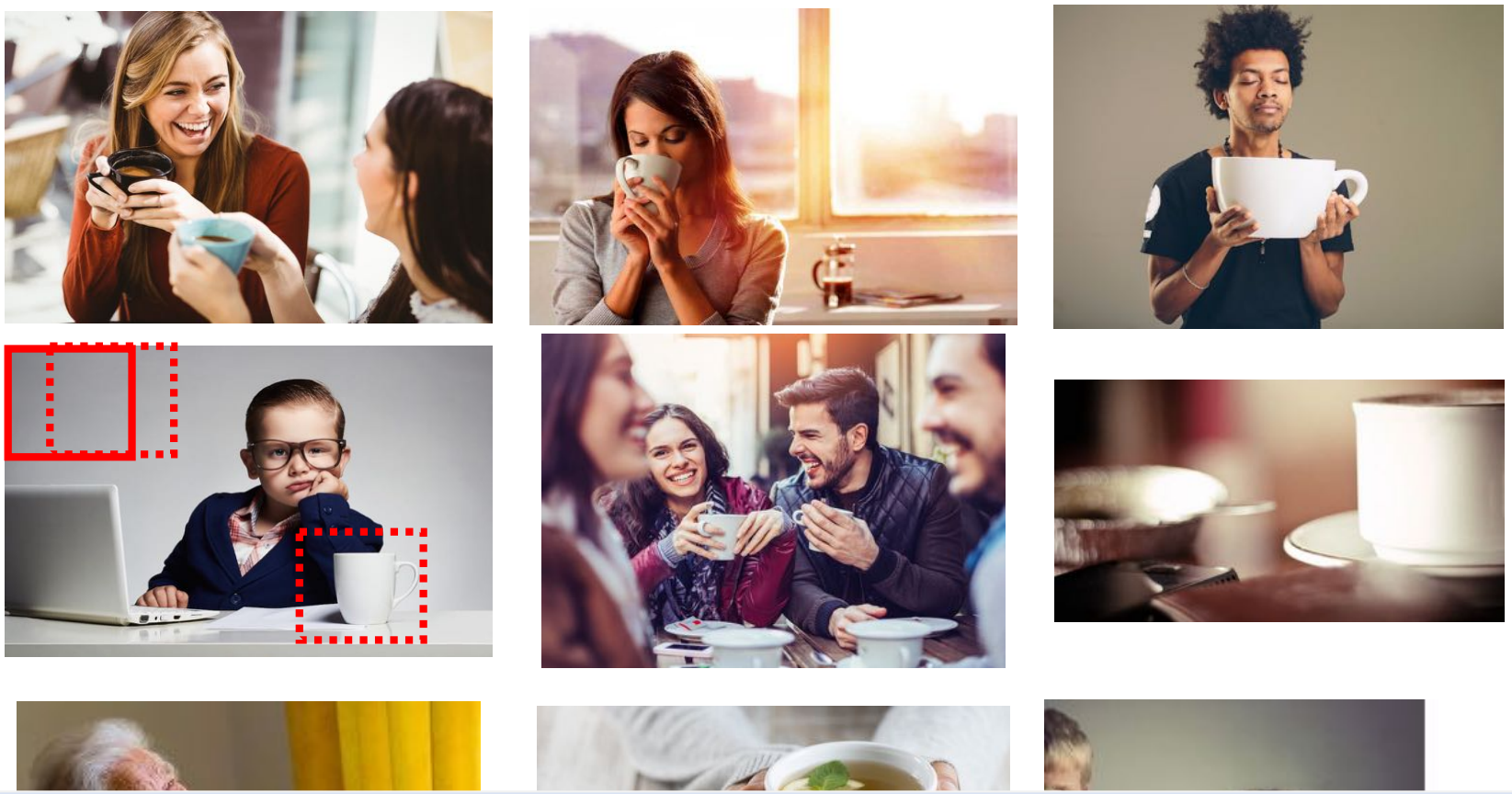








Locality and translation invariance

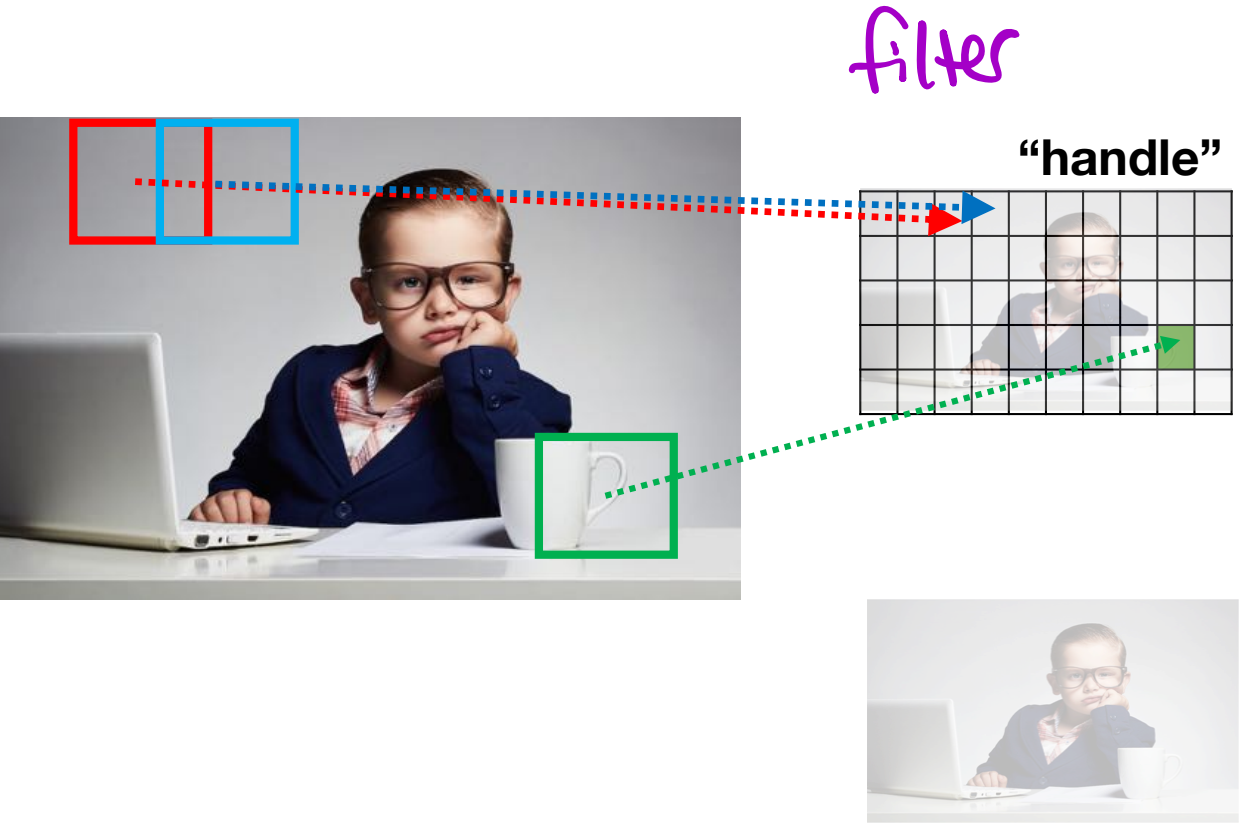


Idea:

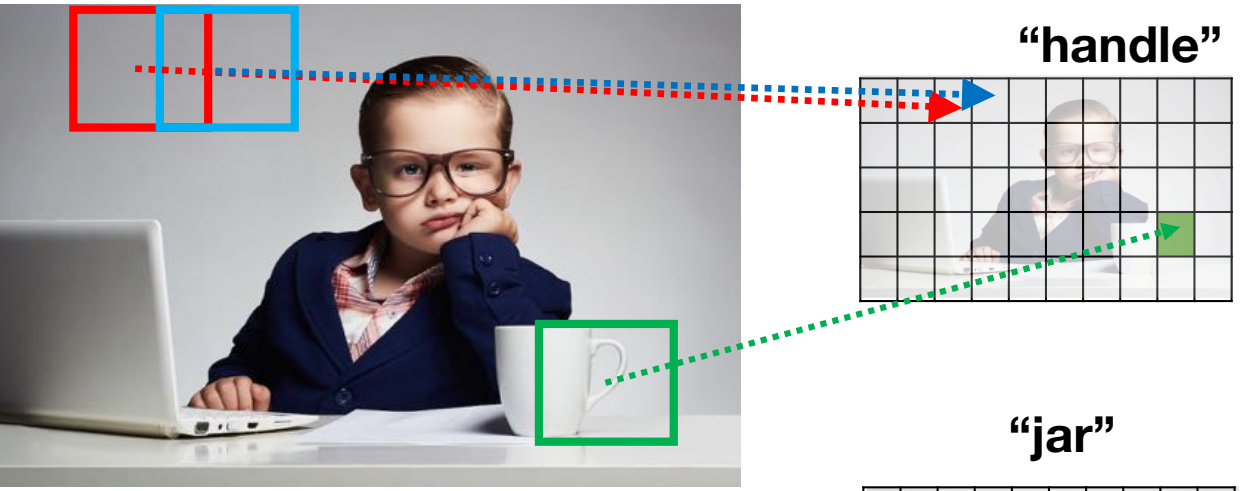
1. learn a patch-size cup detector and *slide it* across the image
2. say “yes” if it fires *anywhere*
3. Hierarchy: edges -> shapes / textures -> parts -> objects -> scenes

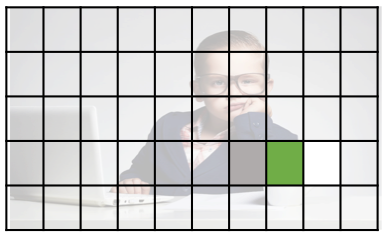
How do we implement this?

How do we implement this?

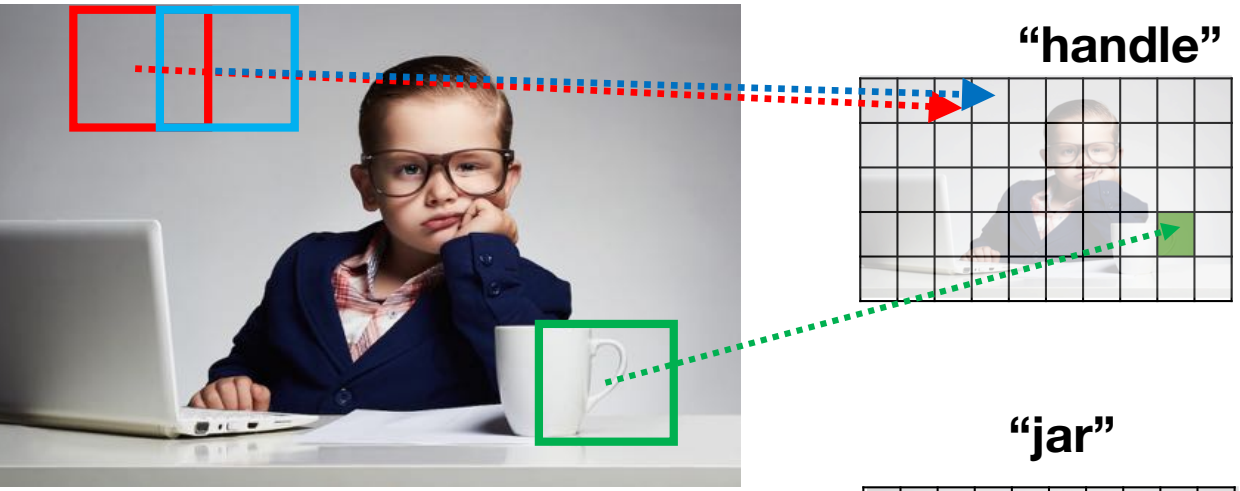


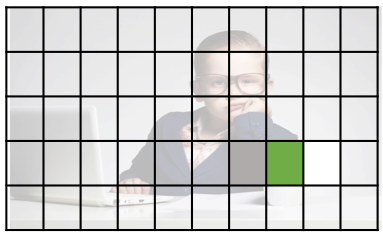
How do we implement this?





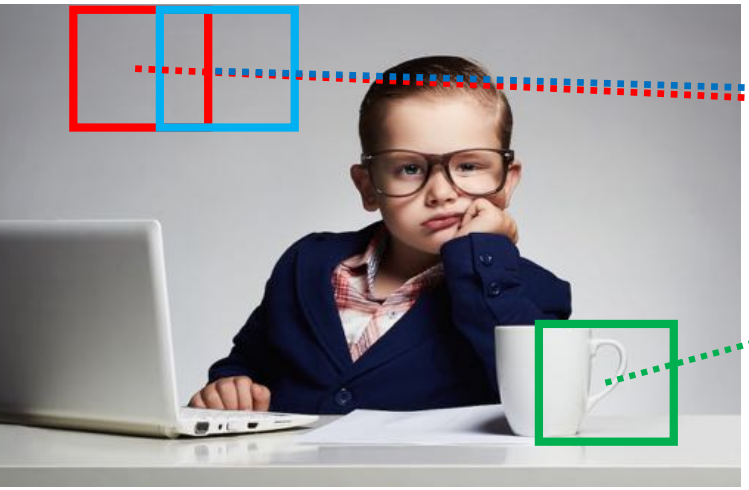
How do we implement this?

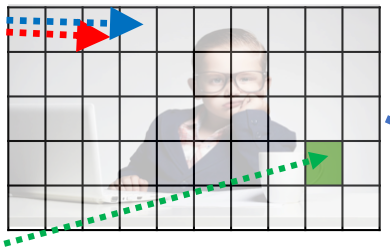




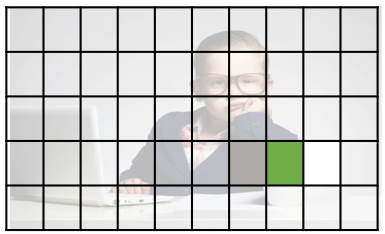
Convolution:
Slide a pattern detector

How do we implement this?



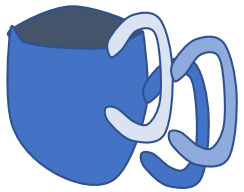


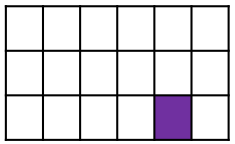
“handle”



“jar”

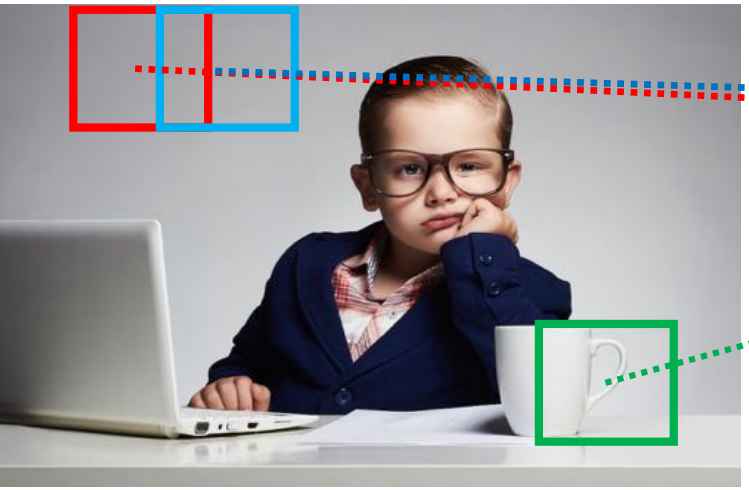
“jar + handle *somewhere in the vicinity*”

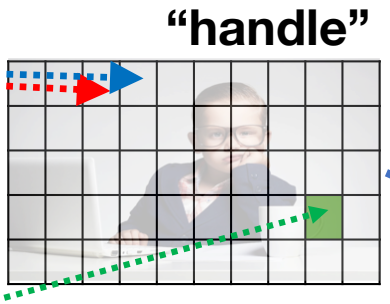


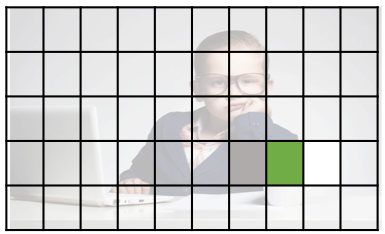


Convolution:
Slide a pattern detector

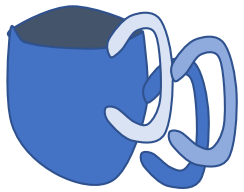
How do we implement this?

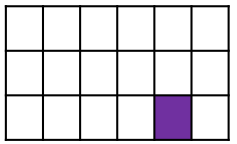






“jar + handle *somewhere in the vicinity*”

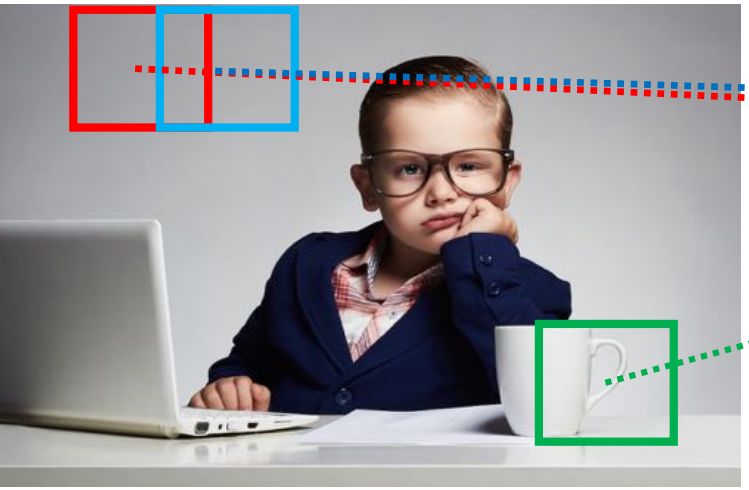


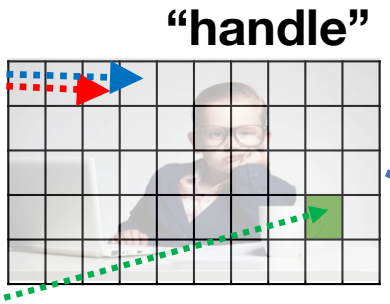


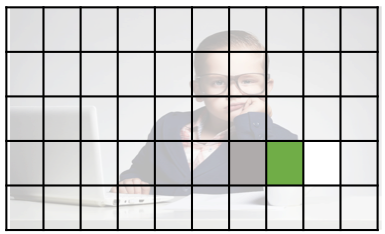
Convolution:
Slide a pattern detector

Pooling:
“forget” exact locations

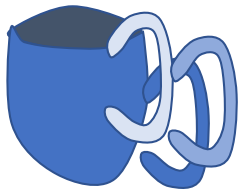
How do we implement this?

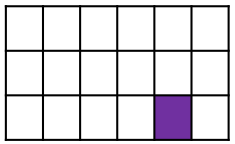






“jar + handle *somewhere in the vicinity*”





Convolution:
Slide a pattern detector

Pooling:
“forget” exact locations

Convolution:
Slide a pattern detector

Hierarchy from Depth

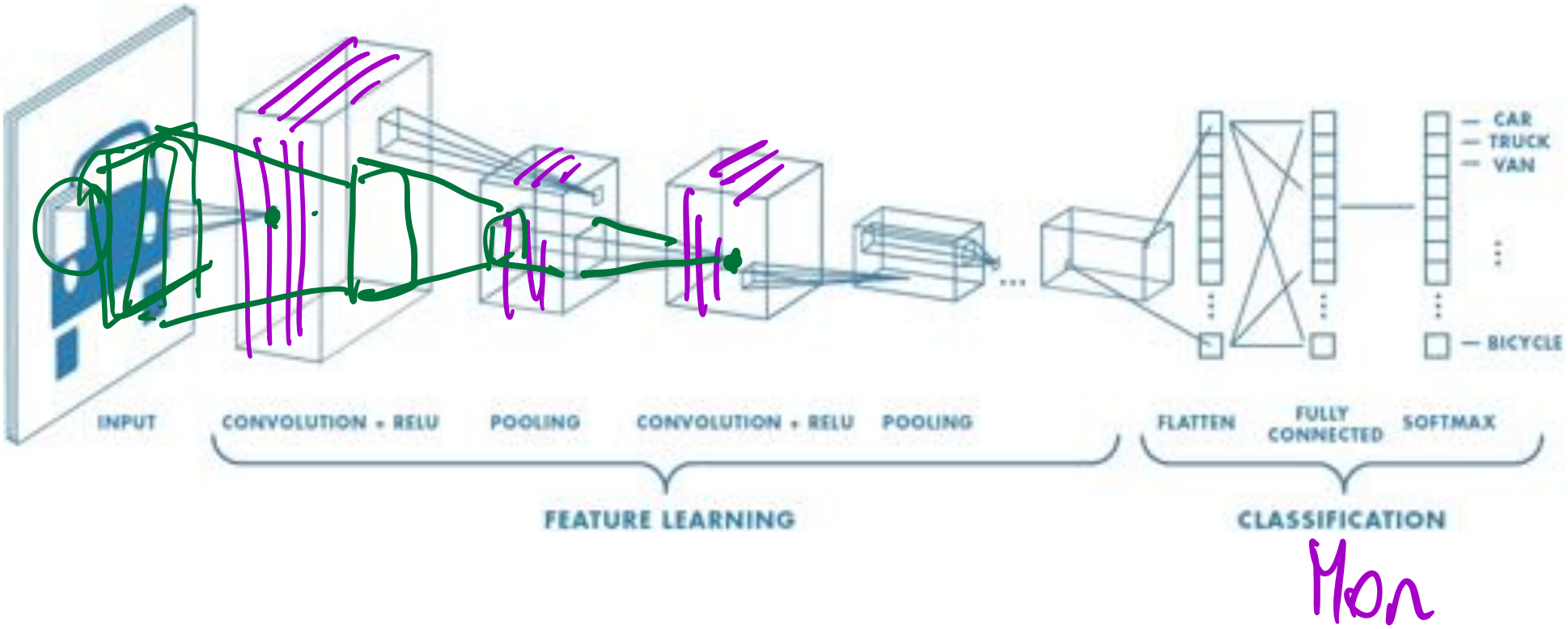


Image: https://www.mathworks.com/solutions/deep-learning/convolutional-neural-network.html

CNNs: main ideas

- **Convolution: “local detectors”**
  spatial locality

1. *learn a patch-size cup detector and slide it across the image*

CNNs: main ideas

- **Convolution: “local detectors”**
  spatial locality

1. *learn a patch-size cup detector and slide it across the image*

- **Weight sharing:**
  apply same detector to all image patches
  efficiency, translation invariance

CNNs: main ideas

- **Convolution: "local detectors"**
  spatial locality

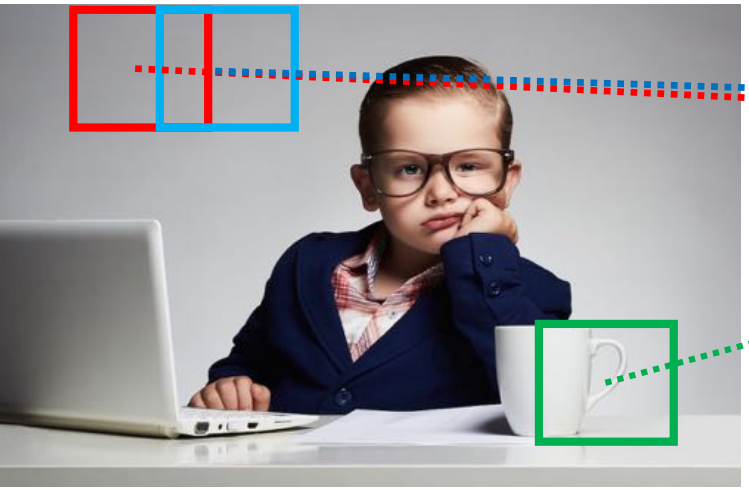
1. *learn a patch-size cup detector and slide it across the image*

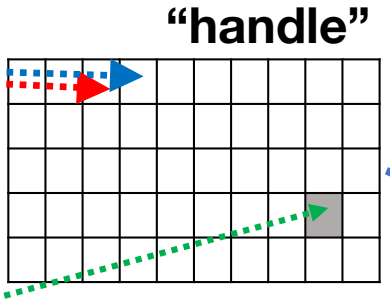
- **Weight sharing:**
  apply same detector to all image patches
  efficiency, translation invariance

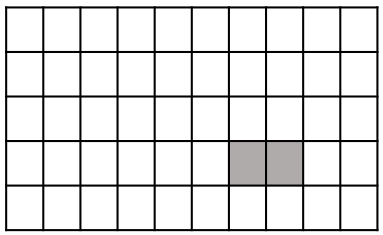
- **Pooling**
  abstract away locality

2. *say "yes" if it fires anywhere in a region*

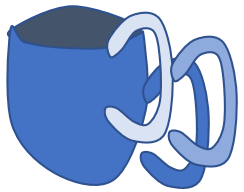
How do we implement this?

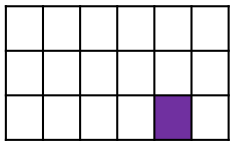






“jar + handle *somewhere in the vicinity*”





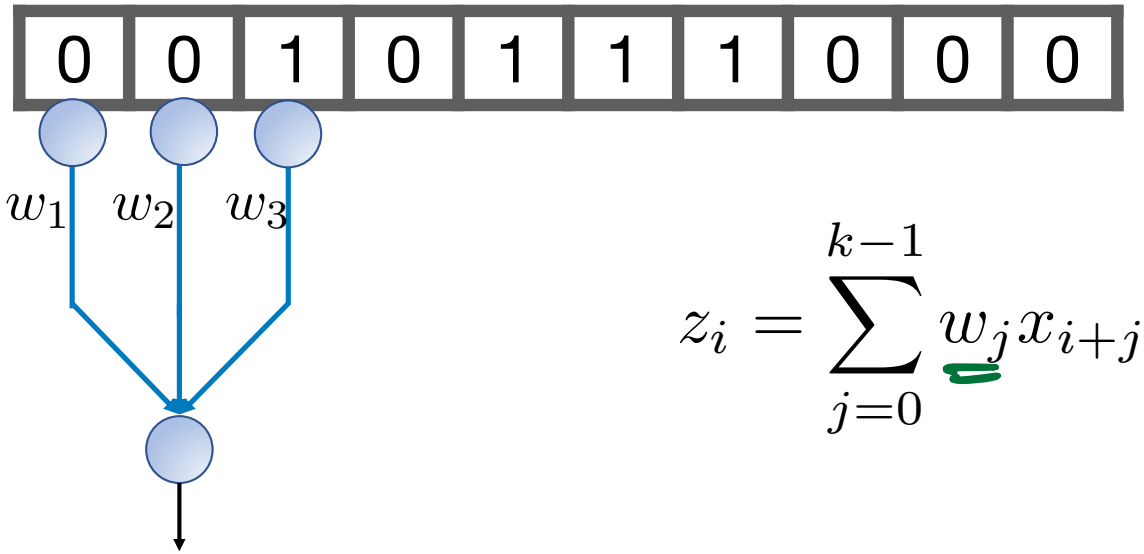
Convolution: **Slide a pattern detector**

Pooling: **“forget” exact locations**

Convolution: **Slide a pattern detector**

Convolutional Layer: 1D example

- 1D image
- Filter
  (weights, bias)
- After convolution



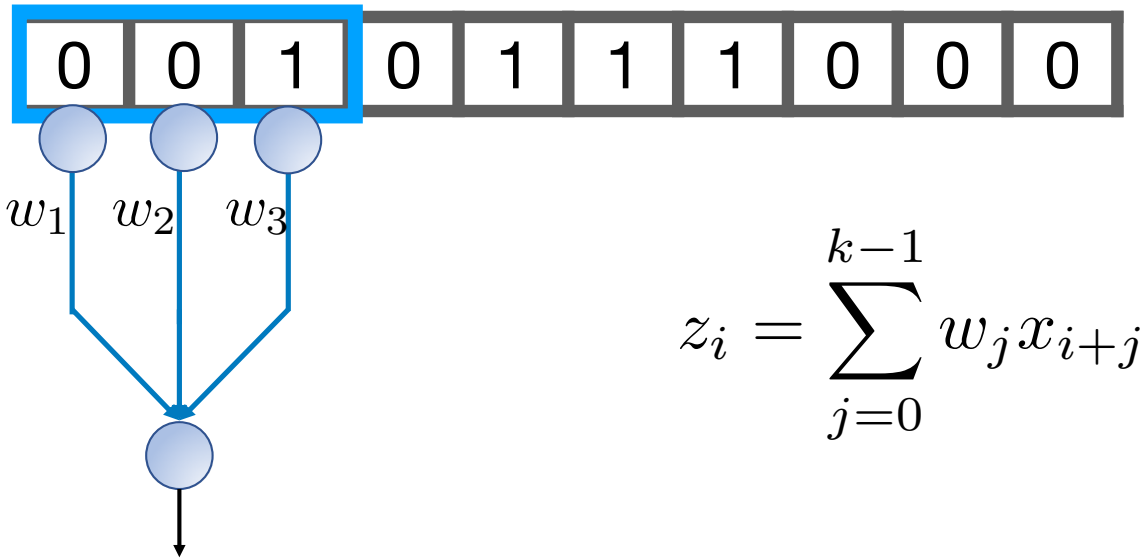
$$z_i = \sum_{j=0}^{k-1} w_j x_{i+j} + b_i$$

*A **filter** is like a*
***locally** connected neuron*

(here, will use bias=0)

Convolutional Layer: 1D example

- 1D image
- Filter
  (weights, bias)
- After convolution



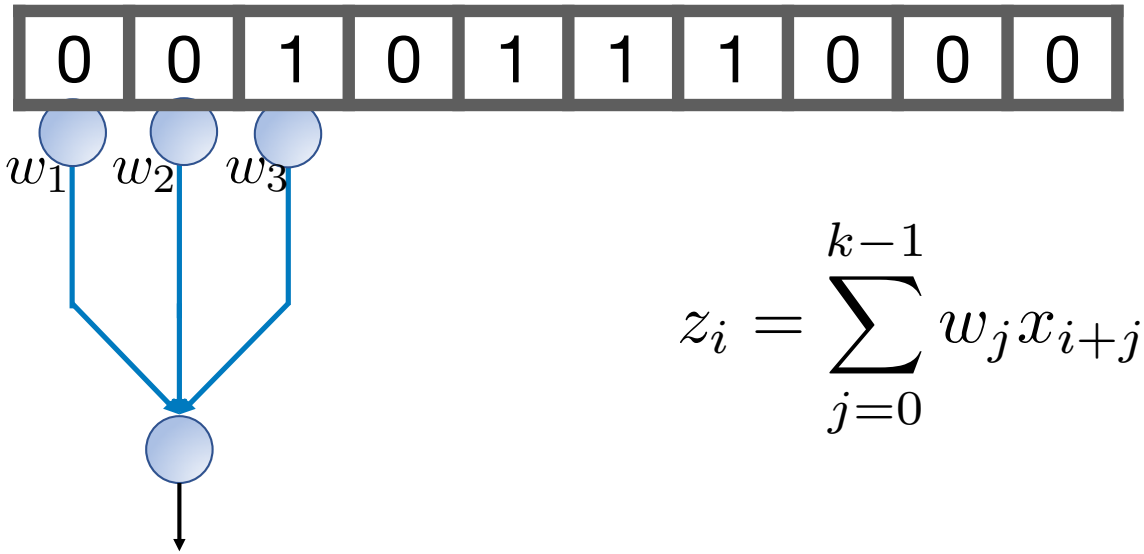
$$z_i = \sum_{j=0}^{k-1} w_j x_{i+j}$$

*A **filter** is like a*
***locally** connected neuron*

(here, will use bias=0)

Convolutional Layer: 1D example

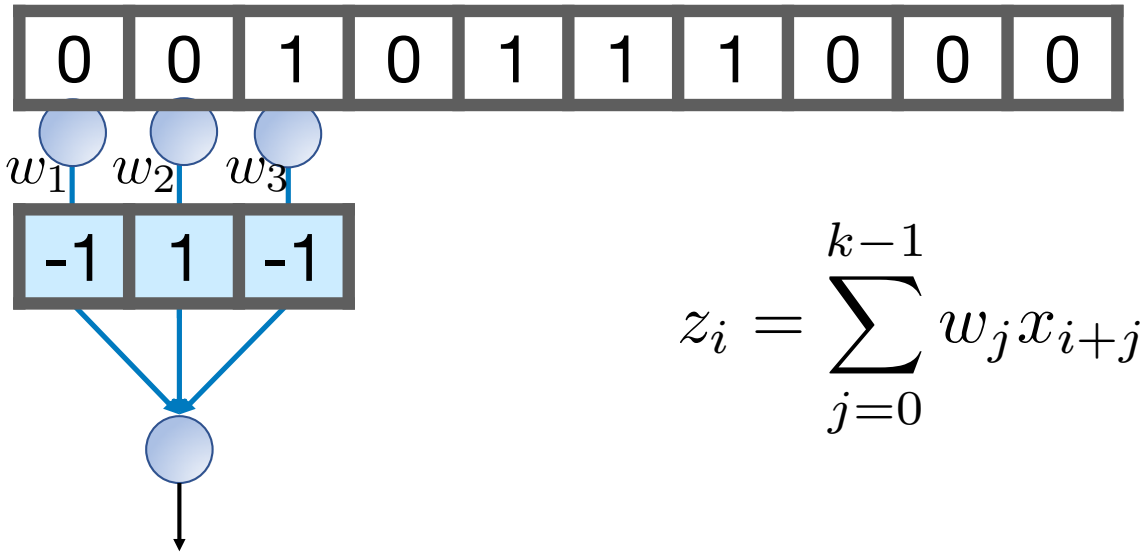
- 1D image
- Filter
  (weights, bias)
- After convolution



$$z_i = \sum_{j=0}^{k-1} w_j x_{i+j}$$

Convolutional Layer: 1D example

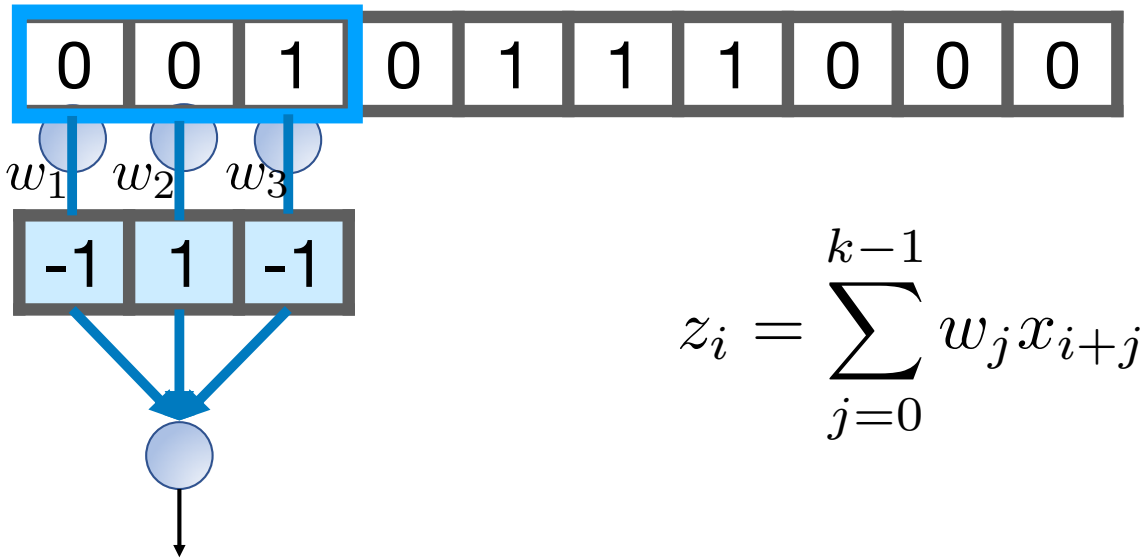
- 1D image
- Filter
  (weights, bias)
- After convolution



$$z_i = \sum_{j=0}^{k-1} w_j x_{i+j}$$

Convolutional Layer: 1D example

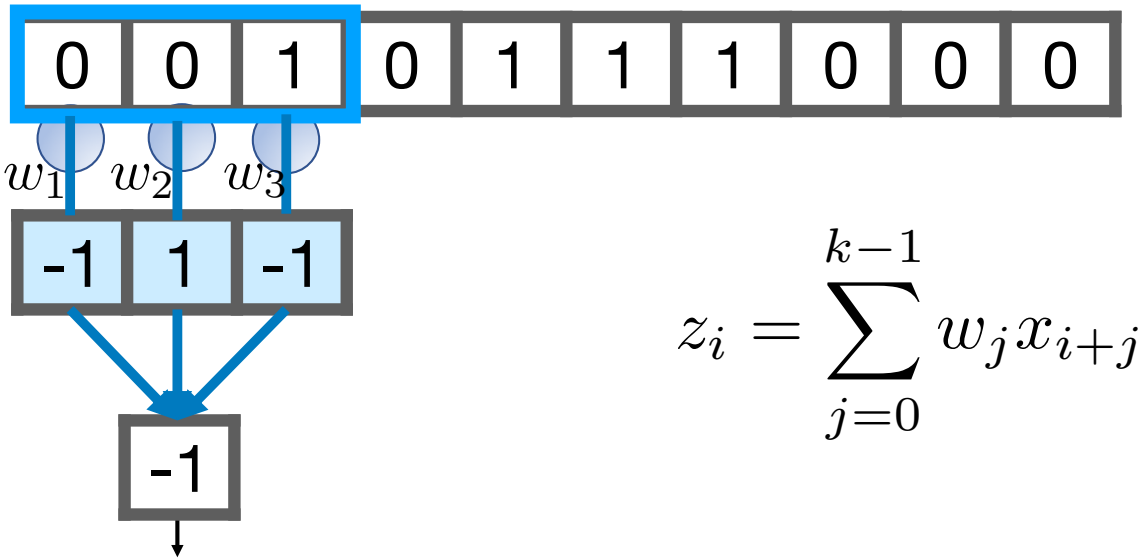
- 1D image
- Filter
  (weights, bias)
- After convolution



$$z_i = \sum_{j=0}^{k-1} w_j x_{i+j}$$

Convolutional Layer: 1D example

- 1D image
- Filter
  (weights, bias)
- After convolution

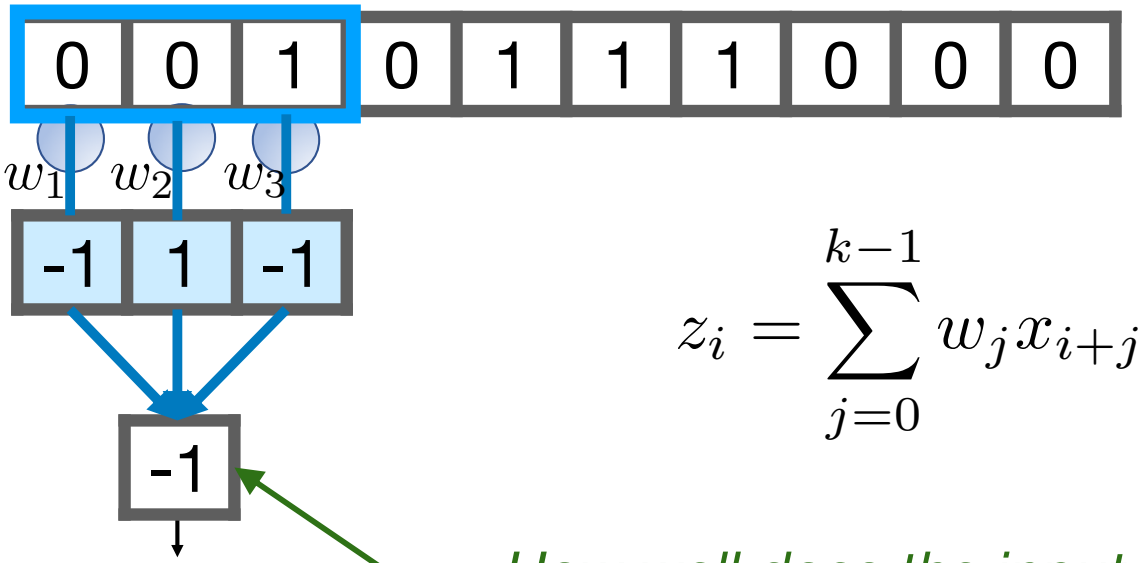


$$z_i = \sum_{j=0}^{k-1} w_j x_{i+j}$$

Example: $(-1) \cdot 0 + 1 \cdot 0 + (-1) \cdot 1 = -1$

Convolutional Layer: 1D example

- 1D image
- Filter
  (weights, bias)
- After convolution



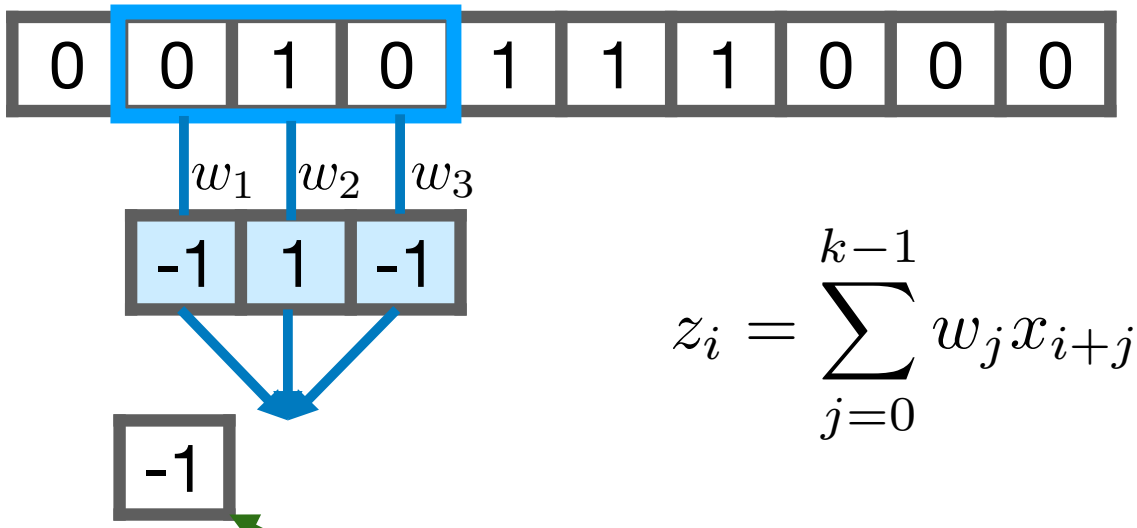
$$z_i = \sum_{j=0}^{k-1} w_j x_{i+j}$$

How well does the input match the filter pattern (low-high-low)? larger == better

Example: $(-1) \cdot 0 + 1 \cdot 0 + (-1) \cdot 1 = -1$

Convolutional Layer: 1D example

- 1D image
- Filter
- After convolution

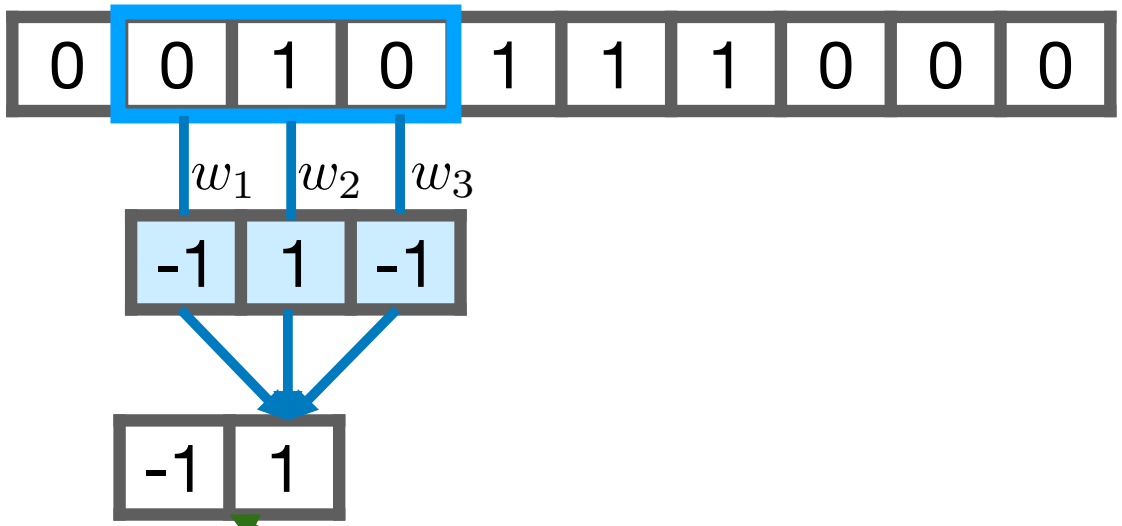


$$z_i = \sum_{j=0}^{k-1} w_j x_{i+j}$$

How well does the input match the filter pattern (low-high-low)? larger == better

Convolutional Layer: 1D example

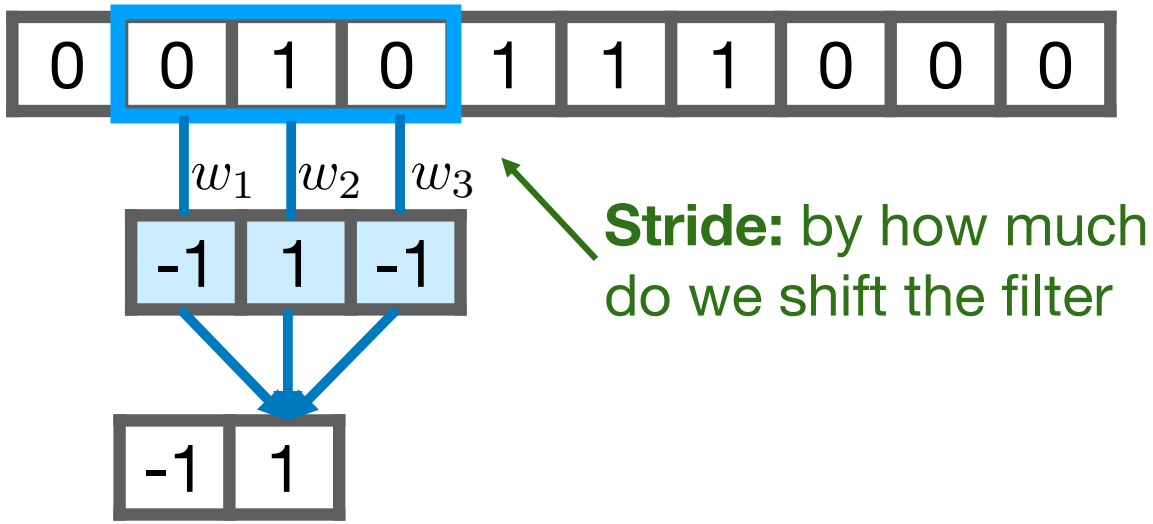
- 1D image
- Filter
- After convolution



Example: $(-1) \cdot 0 + 1 \cdot 1 + (-1) \cdot 0 = 1$

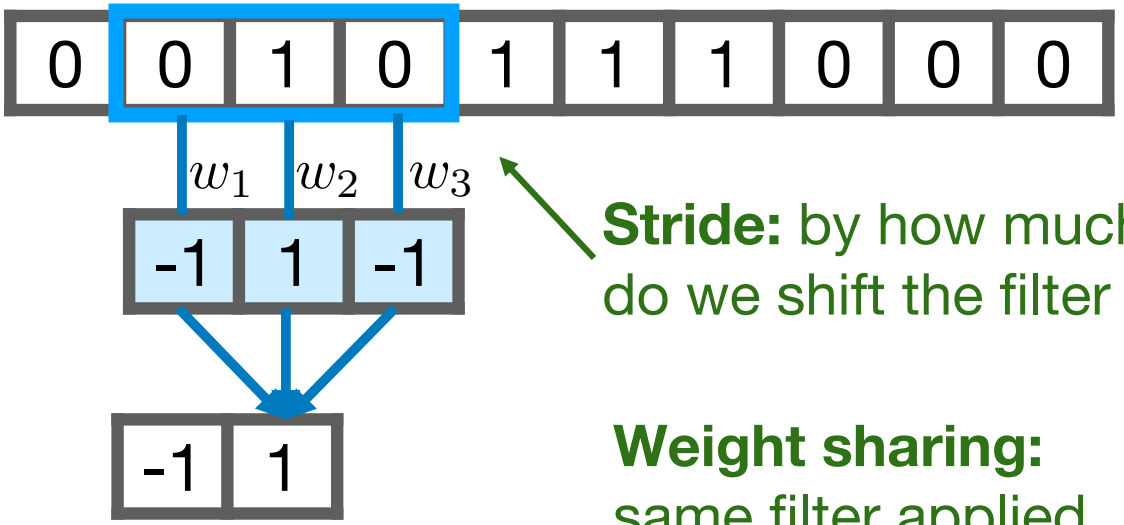
Convolutional Layer: 1D example

- 1D image
- Filter
- After convolution



Convolutional Layer: 1D example

- 1D image
- Filter
- After convolution

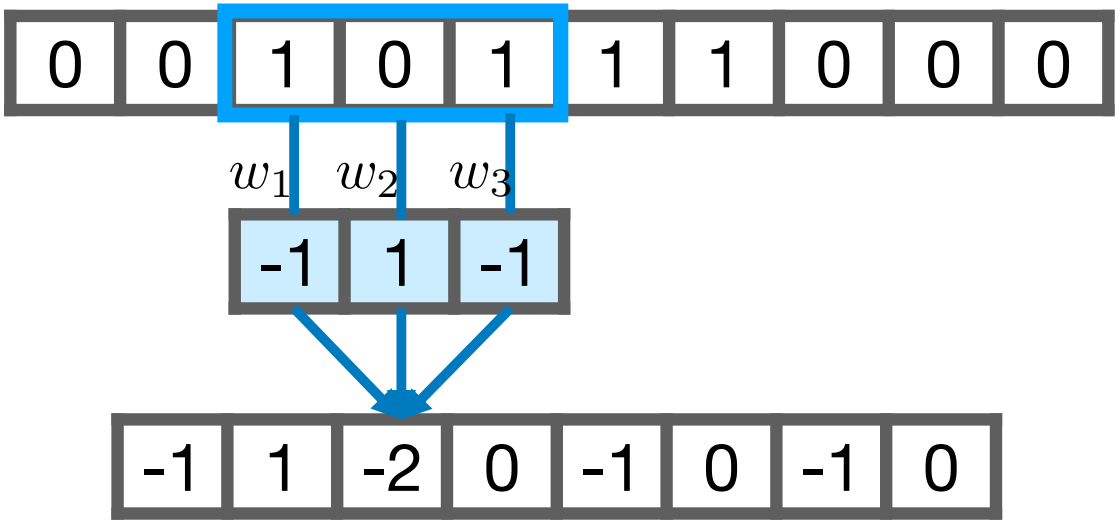


Stride: by how much do we shift the filter

Weight sharing: same filter applied to both (all) patches

Convolutional Layer: 1D example

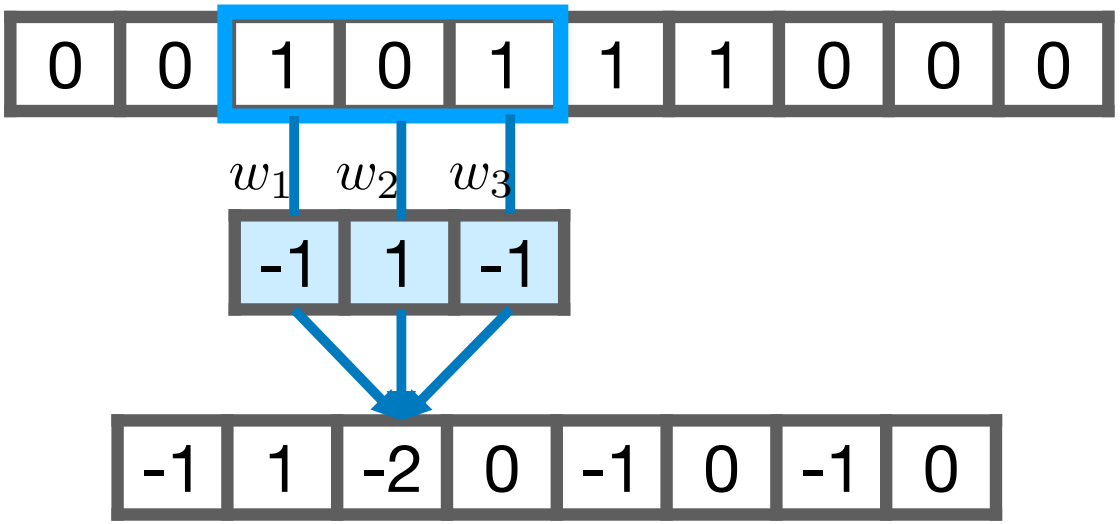
- 1D image
- Filter
- After convolution



Big advantage: due to **weight sharing**, needs much **fewer weights than a fully connected network**

Convolutional Layer: 1D example

- 1D image
- Filter
- After convolution



Big advantage: due to **weight sharing**, needs much **fewer weights than a fully connected network**

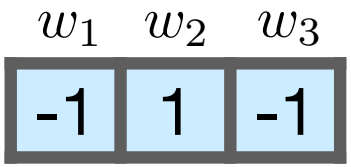
Example:

1. CNN: 5x5filters -> 26 parameters per filter (with bias).
   For 20 filters: **520** parameters per layer – independent of input size!
2. Fully connected, 28x28 inputs, 30 hidden units:
   28*28*30+30 = **23,550** parameters for one layer

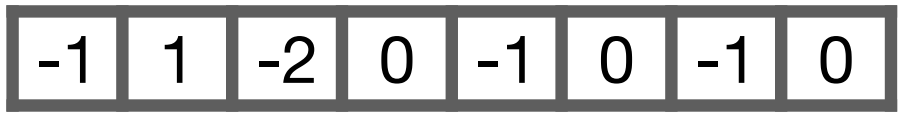
Convolutional Layer: 1D example

- 1D image

- Filter



- After convolution



- After ReLu

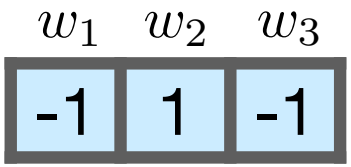
$$ReLU(x) = \max\{0, x\}$$

$$= \begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{otherwise} \end{cases}$$

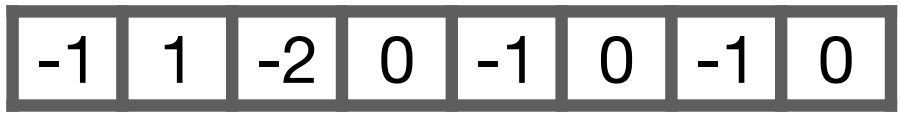
Convolutional Layer: 1D example

- 1D image

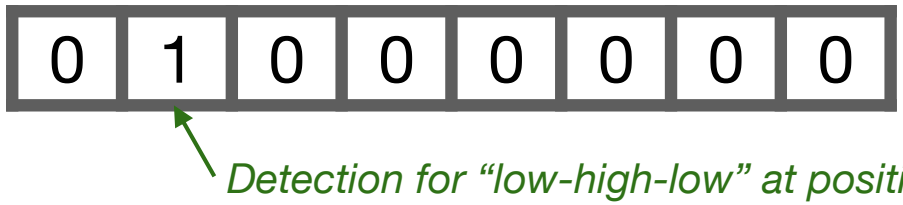
- Filter



- After convolution

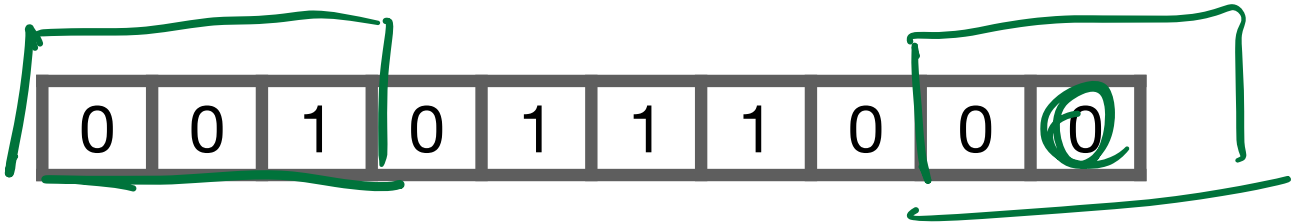


- After ReLu



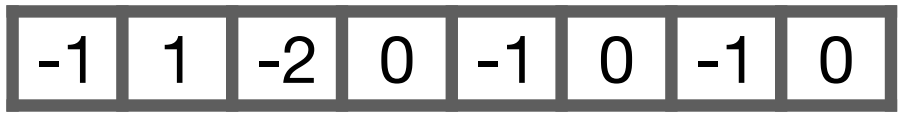
Padding

- 1D image

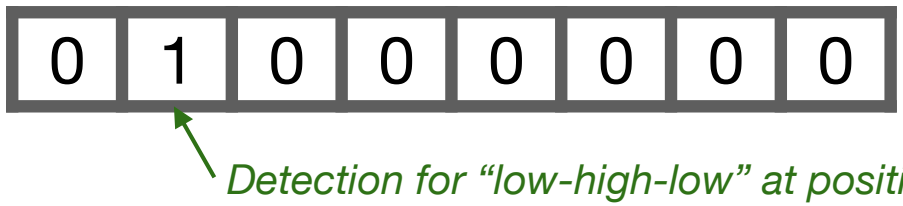


- Filter

- After convolution



- After ReLu



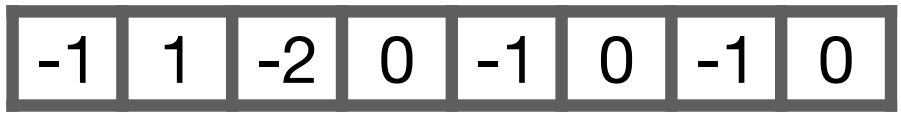
Output is smaller! (why?)

Padding

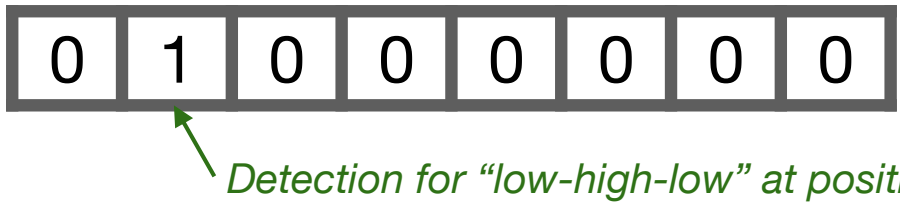
- 1D image

- Filter

- After convolution



- After ReLu



Output is smaller! (why?)

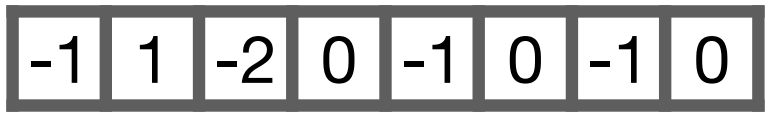
Remedy: pad input with zeros

Padding

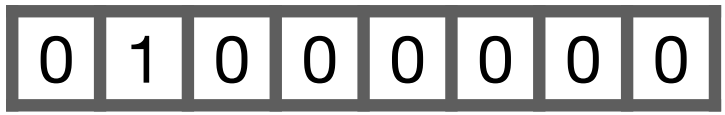
- 1D image

- Filter

- After convolution



- After ReLu



Output is smaller! (why?)

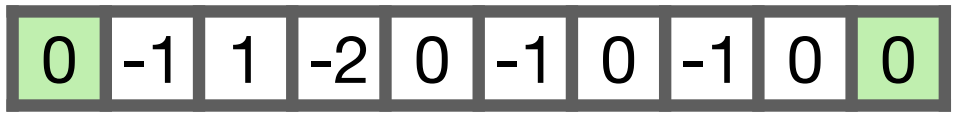
Remedy: pad input with zeros

Padding

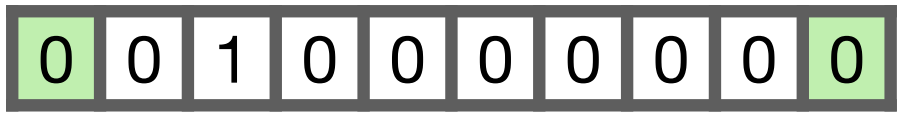
- 1D image

- Filter

- After convolution



- After ReLu

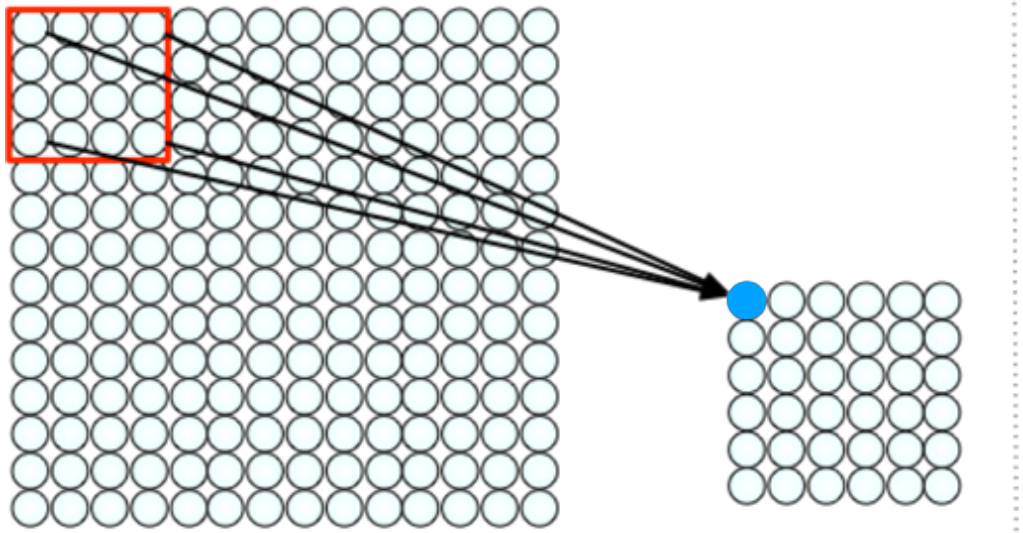


Output is smaller! (why?)

Remedy: pad input with zeros

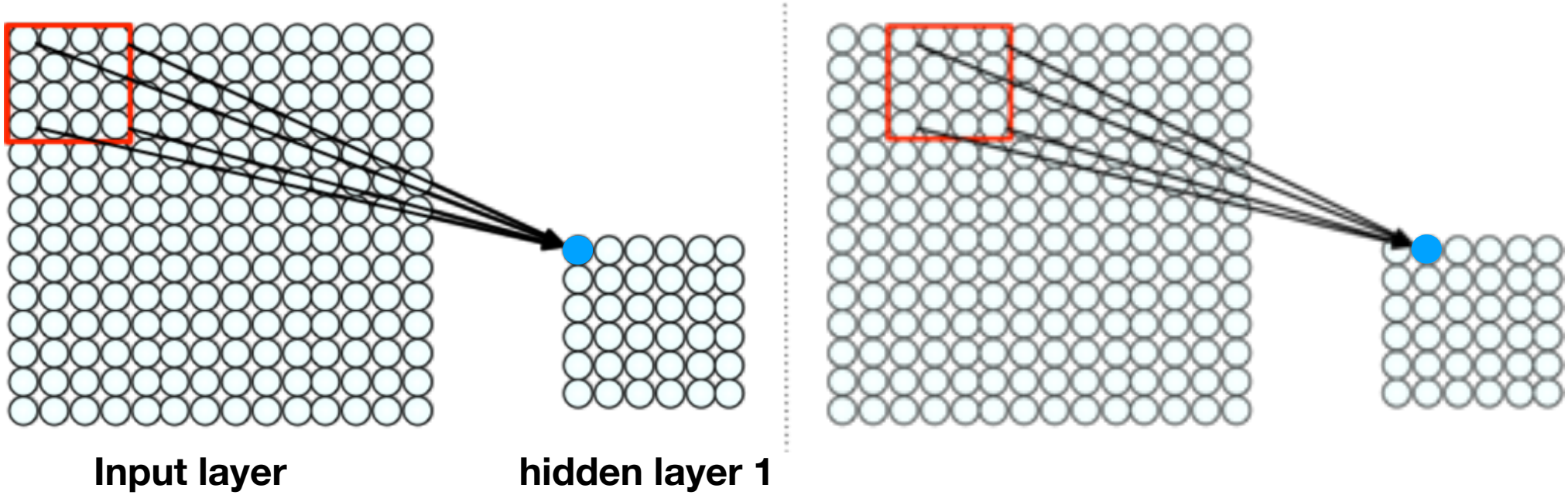
2D Convolution

- In 2D: shift filter in 2 directions (right, down)
- Output is an “image” too
- Example of a 2D convolution, 4x4 filter, stride 2



2D Convolution

- In 2D: shift filter in 2 directions (right, down)
- Output is an “image” too
- Example of a 2D convolution, 4x4 filter, stride 2



Examples of Convolutions

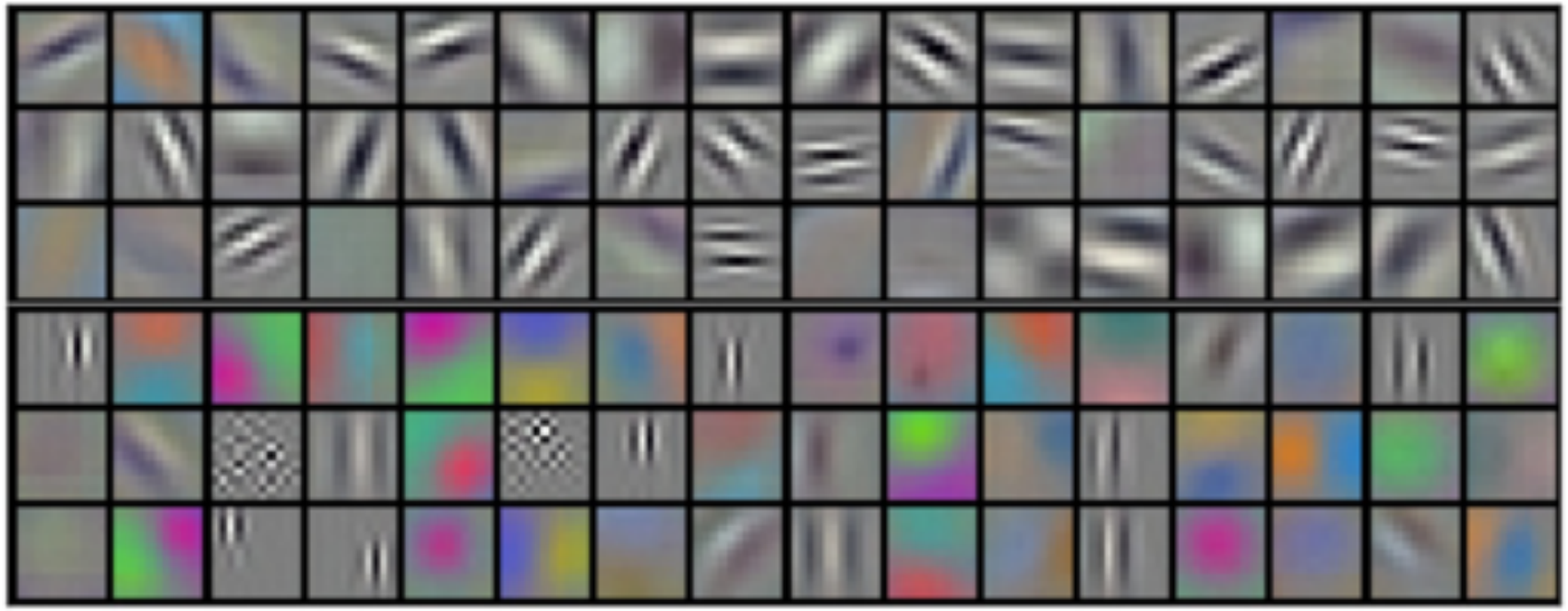
orig

sharpen

edge detect

strong edges

Learned filters (1st layer)



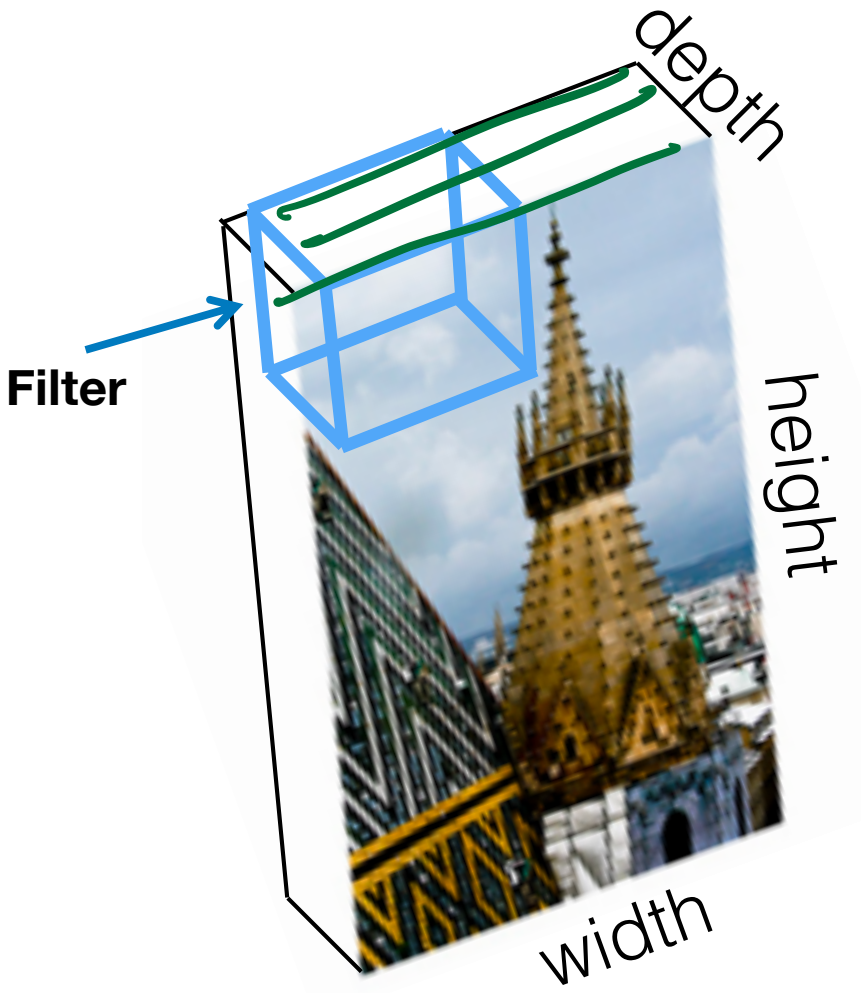
Size: 11x11x3

First layer learns edge detectors!

(Krizhevsky, Sutskever, Hinton: ImageNet Classification with Deep Convolutional Neural Networks, NIPS 2012)

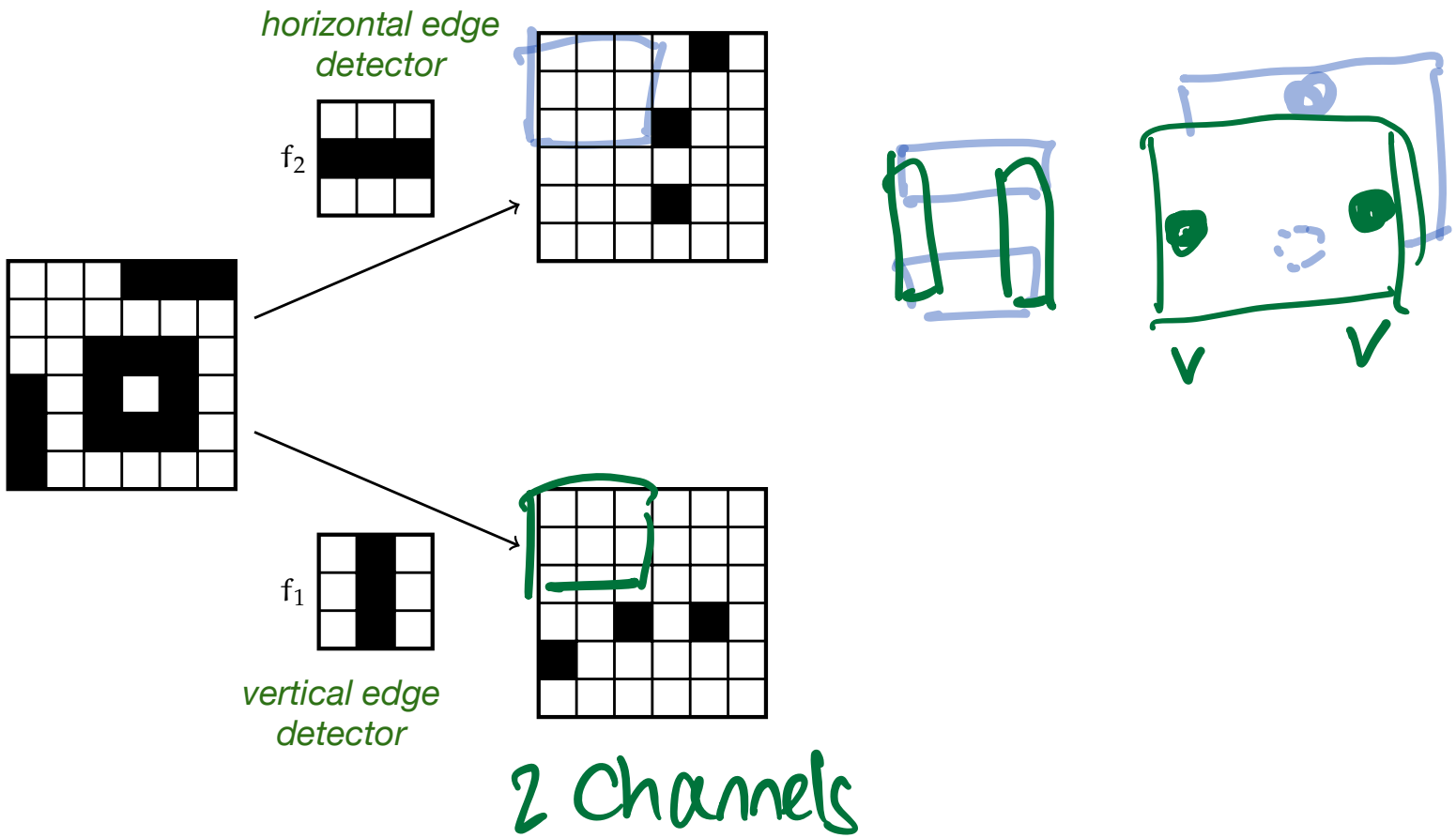
3D convolution

- Color image is actually a tensor (3 matrices): RGB
- Filter is a tensor (generalization of matrix) too



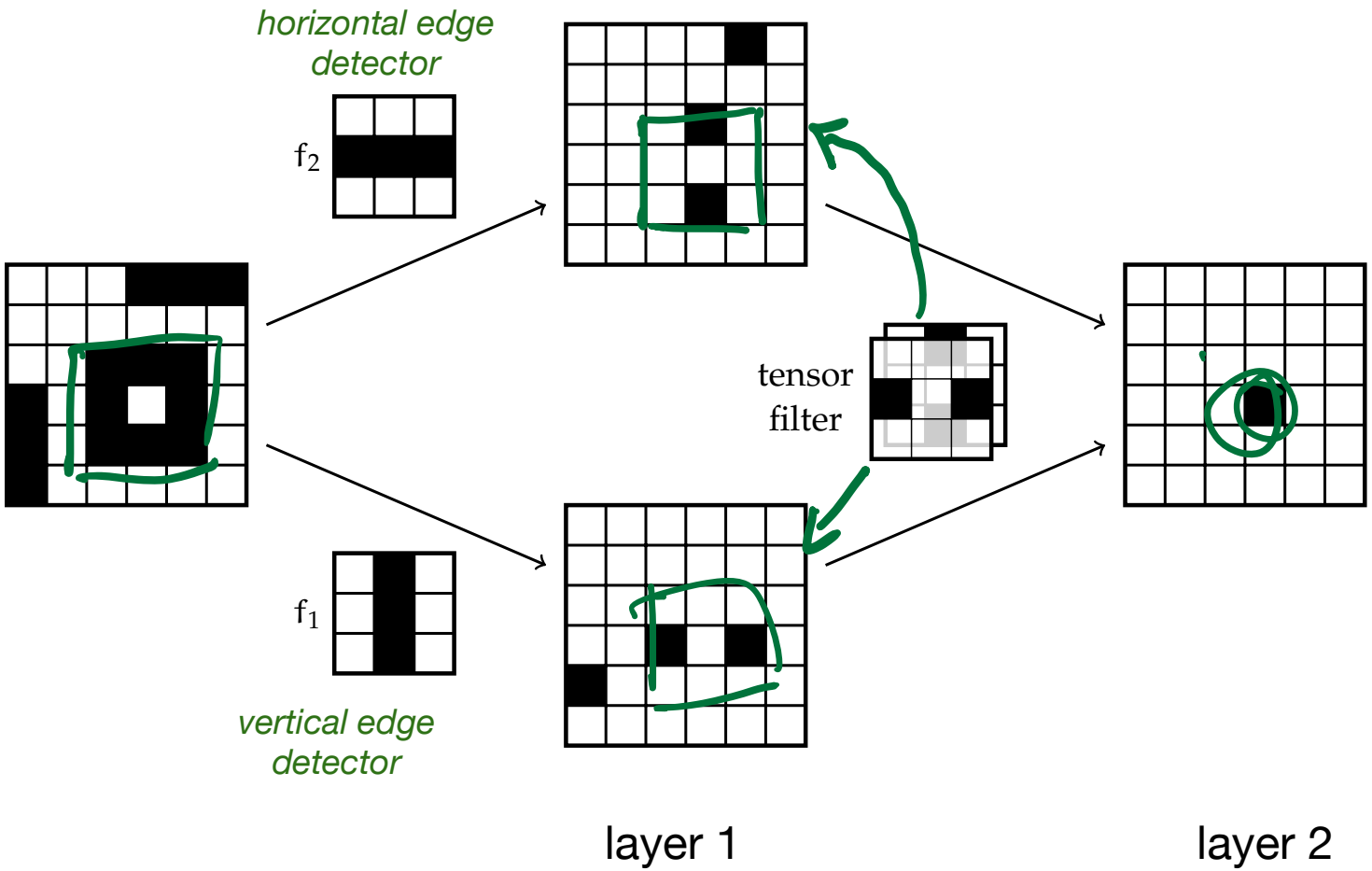
Multiple channels/filters

- Typically, we do not only want to learn one edge detector: shape = specific combination of multiple types of edges
- Use multiple filters *(filter bank)* → multiple output "images"

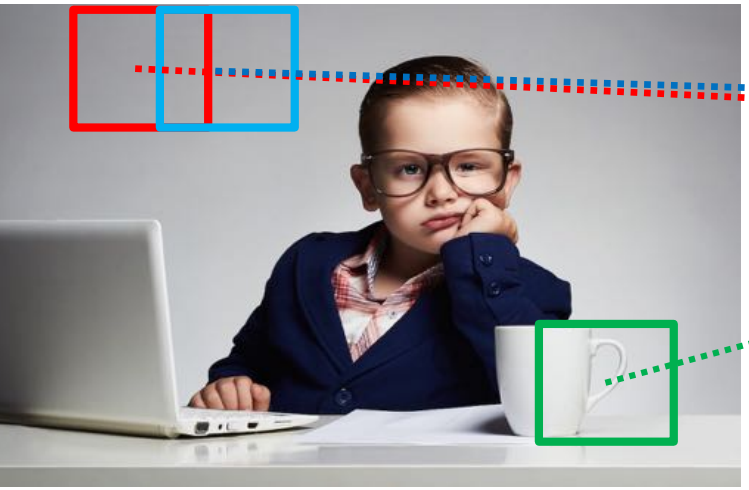


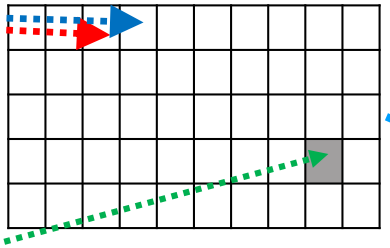
Multi-channel convolution

- Next layer can combine inputs from multiple channels:
  3D (tensor) convolution
  e.g. edges -> shapes



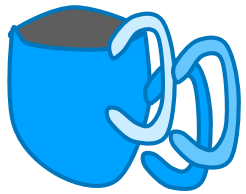
How do we implement this?

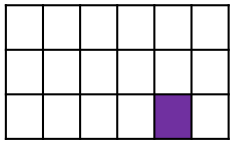


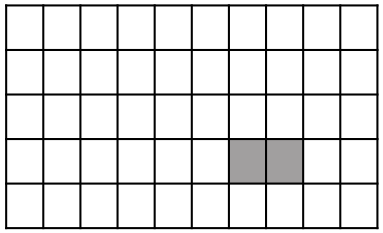


“handle”

“jar + handle *somewhere in the vicinity*”







“jar”

channels

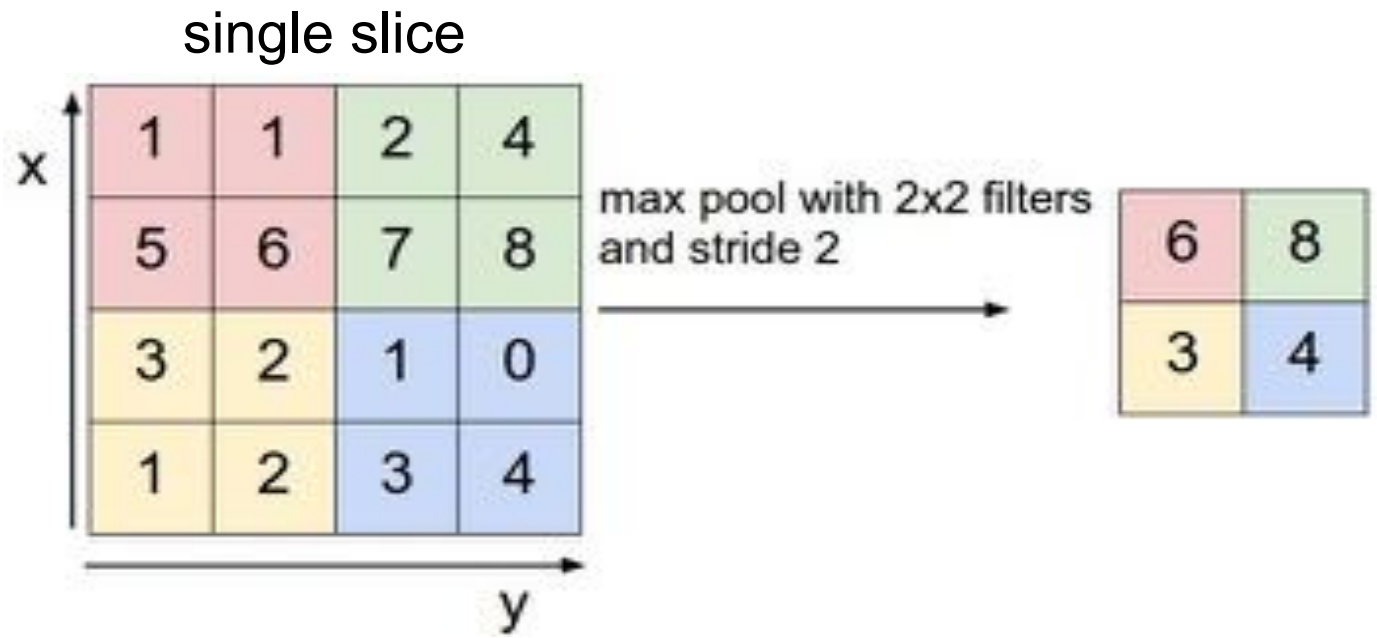
Convolution: **Slide a pattern detector**

Pooling: **“forget” exact locations**

Convolution: **Slide a pattern detector**

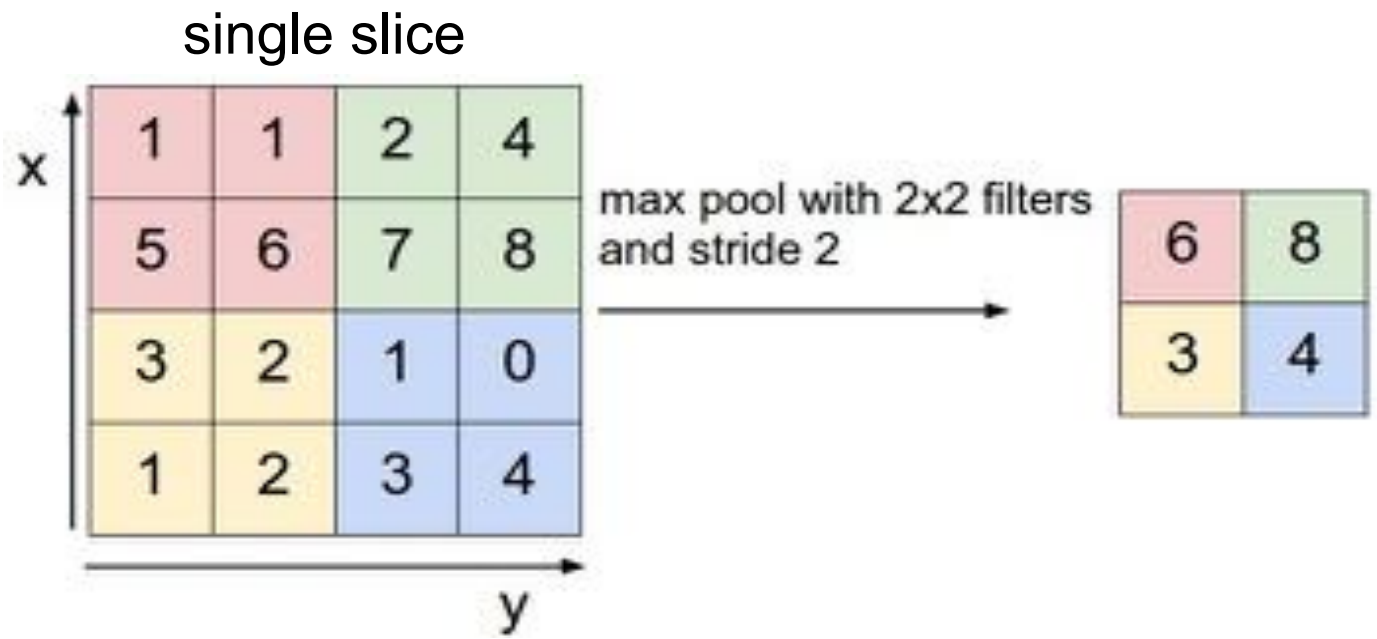
Max Pooling

- Similar to filtering, but output the maximum entry in the patch, instead of a weighted sum



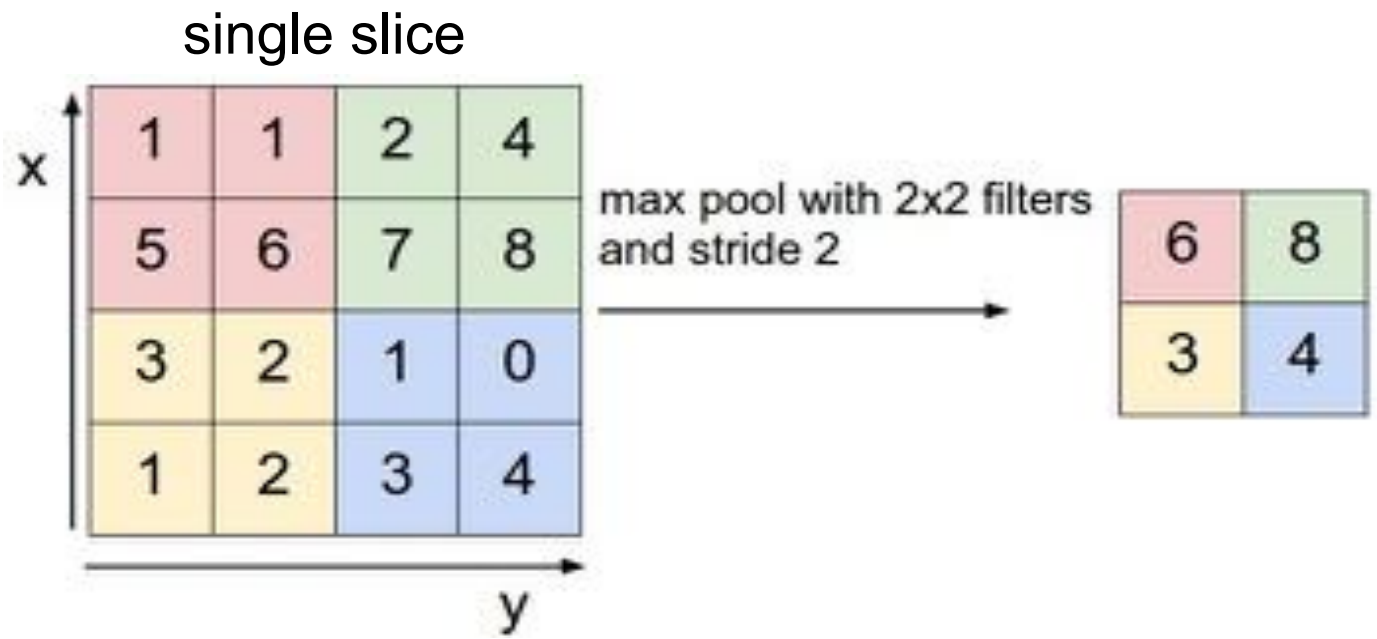
Max Pooling

- Similar to filtering, but output the maximum entry in the patch, instead of a weighted sum
- Typically output image is smaller than input → filters look at successively larger areas in the input image



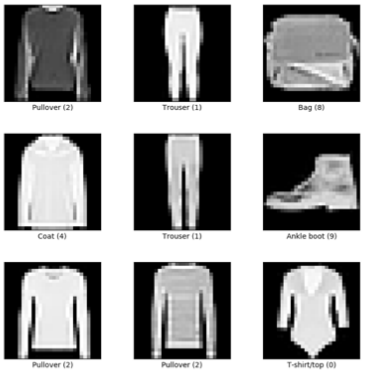
Max Pooling

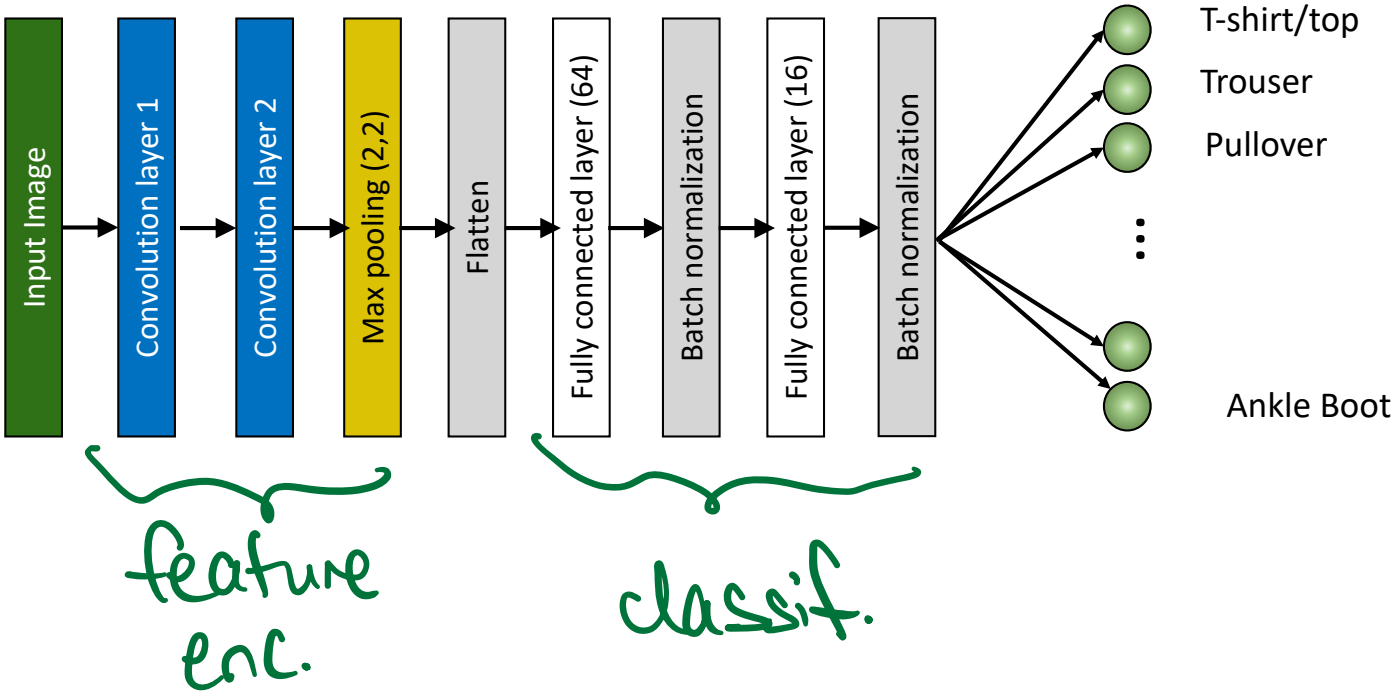
- Similar to filtering, but output the maximum entry in the patch, instead of a weighted sum
- Typically output image is smaller than input → filters look at successively larger areas in the input image
- Usually applied per channel, not across channel



Case study: CNN for Fashion MNIST

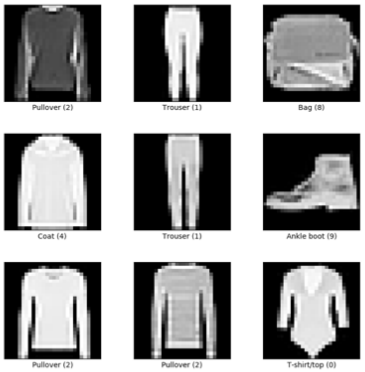
- 2 convolution layers with 16 filters of size (3,3)

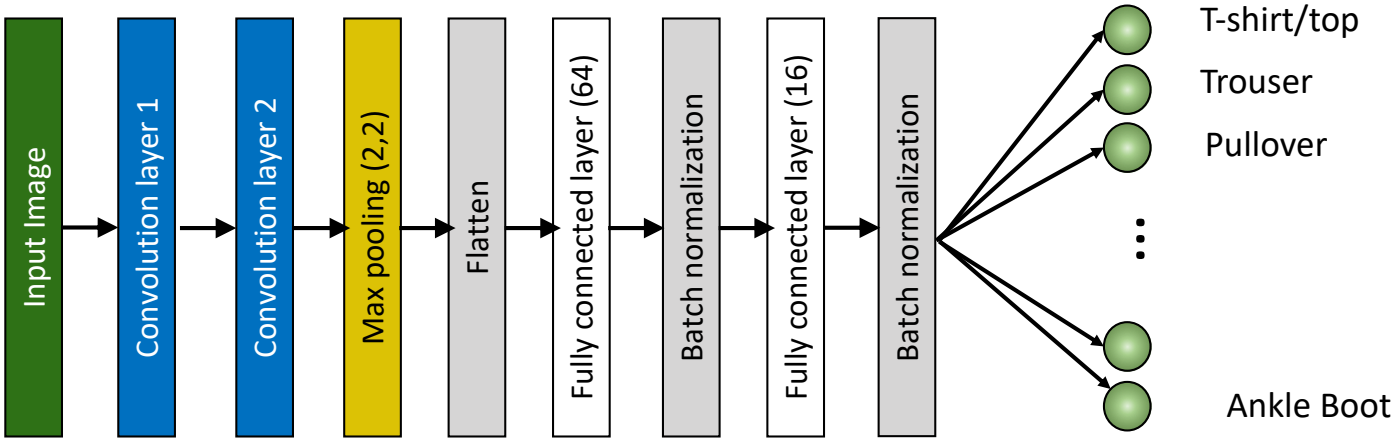




Case study: CNN for Fashion MNIST

- 2 convolution layers with 16 filters of size (3,3)

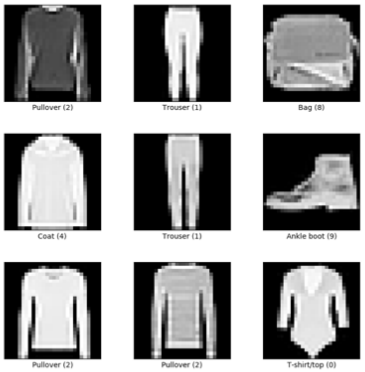


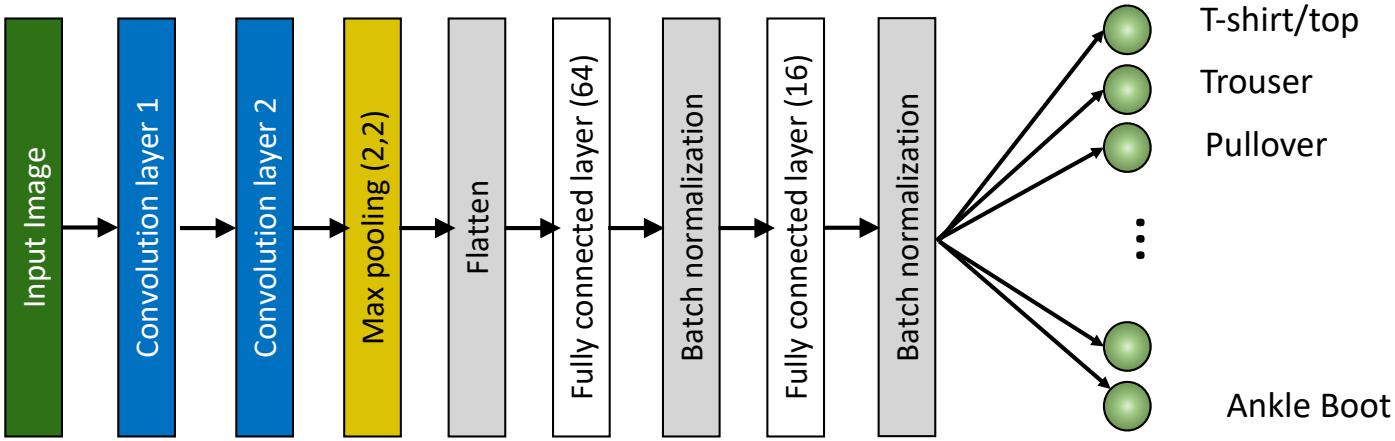


- Train with Adam for 10 epochs, mini-batch of 128

Case study: CNN for Fashion MNIST

- 2 convolution layers with 16 filters of size (3,3)



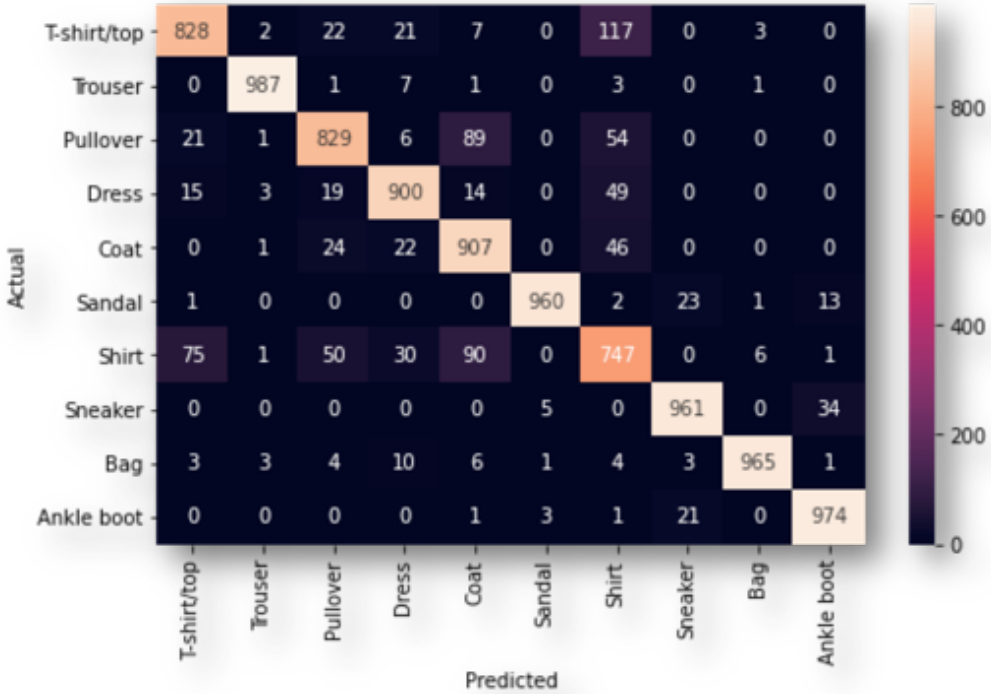


- Train with Adam for 10 epochs, mini-batch of 128

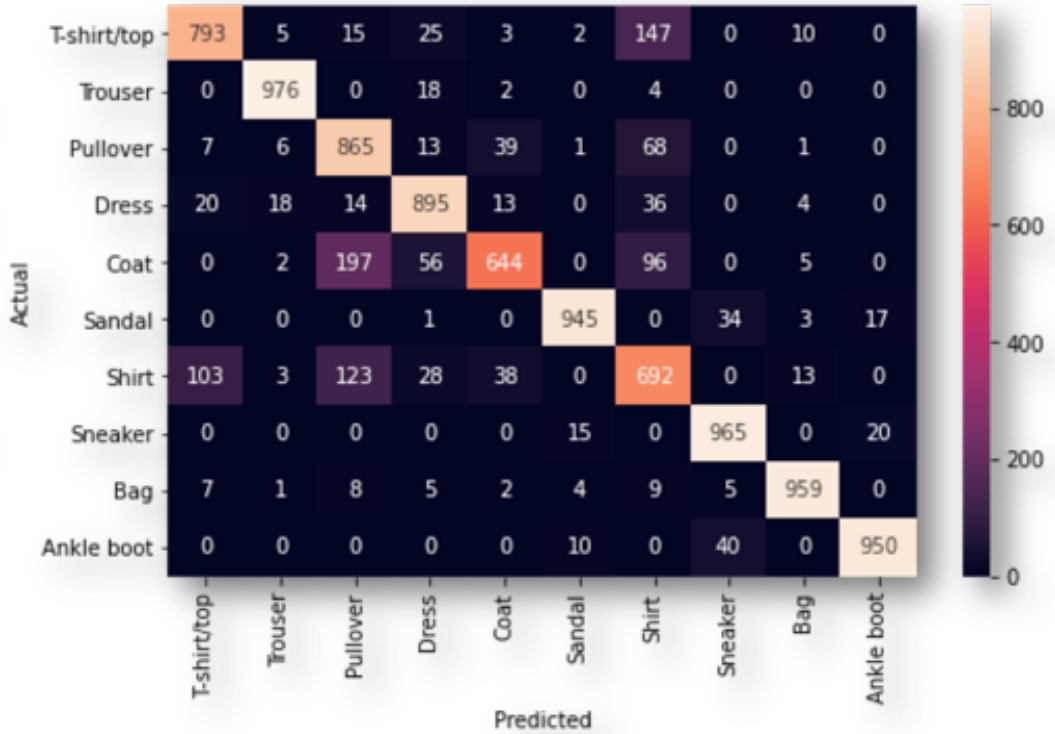
- Accuracy: 0.91 (vs 0.87 for fully connected)

Confusion matrices: CNN vs. fully connected

CNN

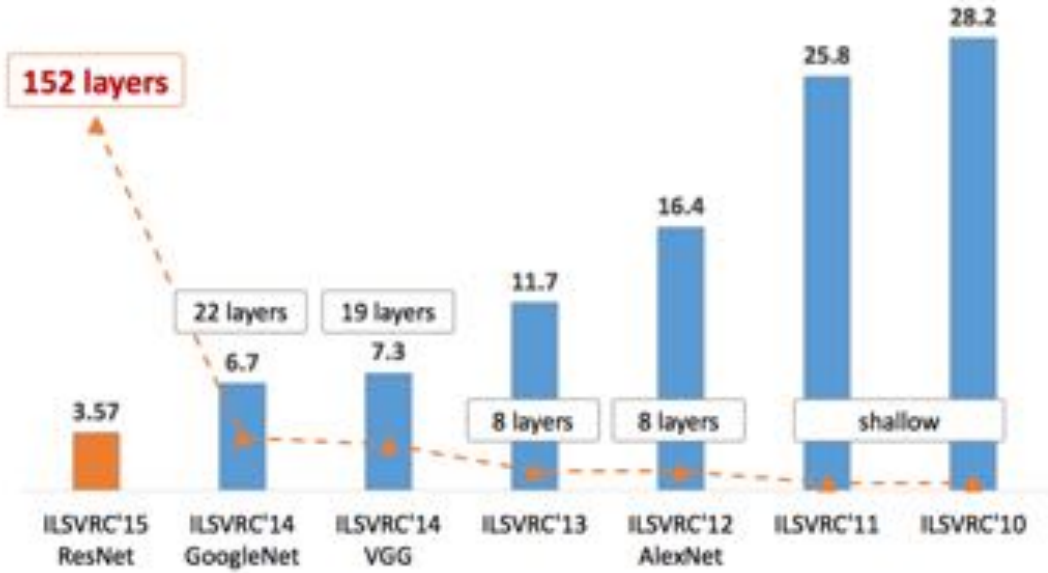


FNN



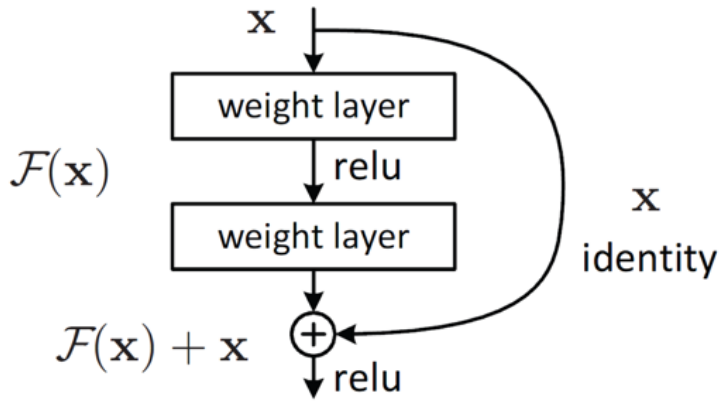
Deep CNNs

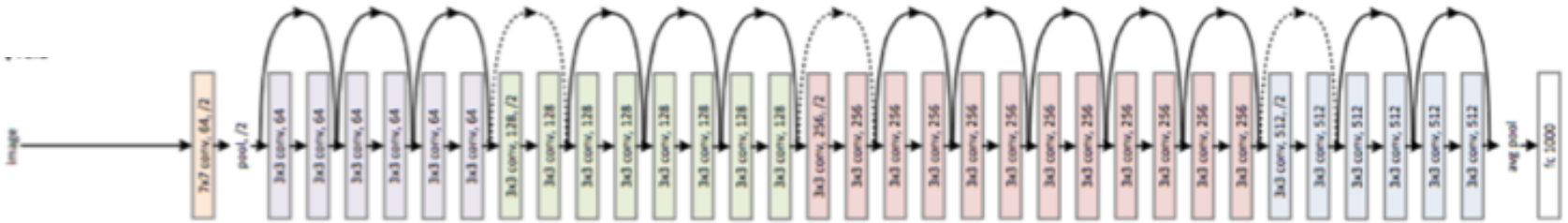

I WAS WINNING IMAGENET
UNTIL A DEEPER MODEL CAME ALONG


152 layers
22 layers
19 layers
8 layers
8 layers
shallow
3.57
6.7
7.3
11.7
16.4
25.8
28.2
ILSVRC'15 ResNet
ILSVRC'14 GoogleNet
ILSVRC'14 VGG
ILSVRC'13
ILSVRC'12 AlexNet
ILSVRC'11
ILSVRC'10

Optional: ResNet

- Skip connections improve training for very deep network
- Idea: add a layer's input to its output (shortcuts)





Summary: CNN model

- **Convolution: “local detectors”**
  spatial locality
- **Weight sharing:**
  apply same detector to all image patches
  efficiency: much fewer parameters!, translation invariance
- **Pooling: “did I see this pattern anywhere at all in this region?”**
  abstract away locality

- Trained with SGD and backpropagation

Outline: CNNs

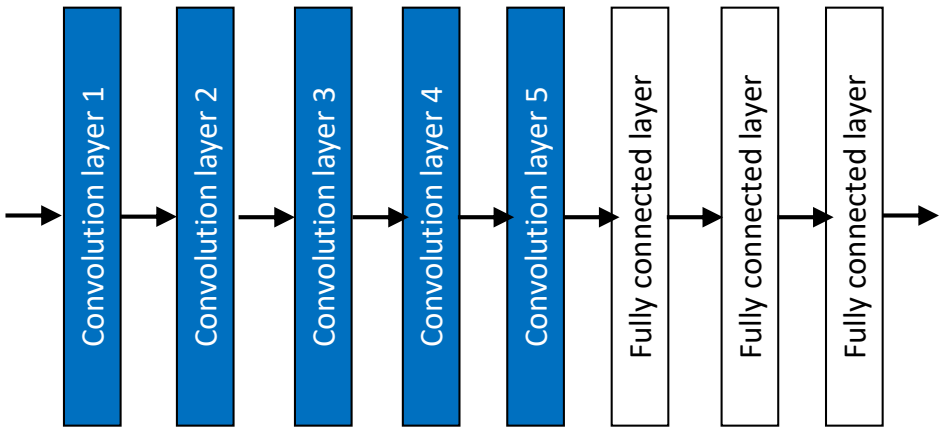
- Spatial locality & invariance

- Convolutional Neural Networks
  - Convolution and filters
  - Max Pooling
- Example: Fashion MNIST

- CNN training
  - Illustrations: what do CNNs (not) learn?

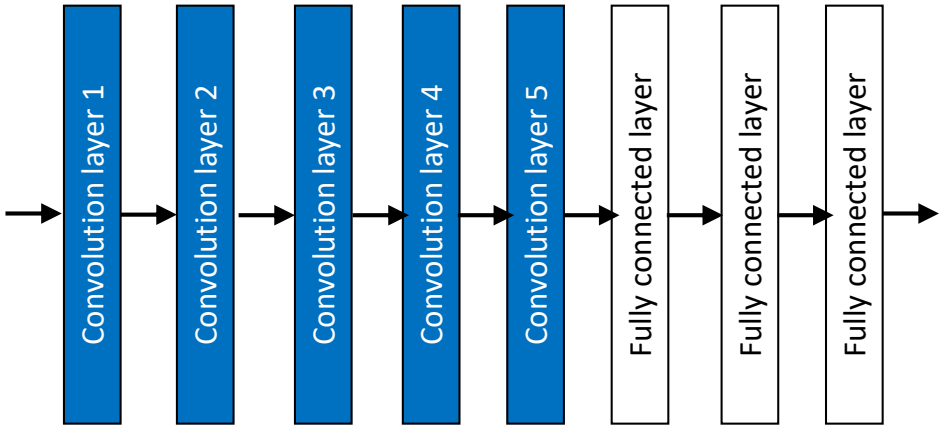
What do neurons in a CNNs learn?

- **Visualization:**
  View neurons as pattern detectors -- find patches that activate a neuron, and see whether they classify a “concept”



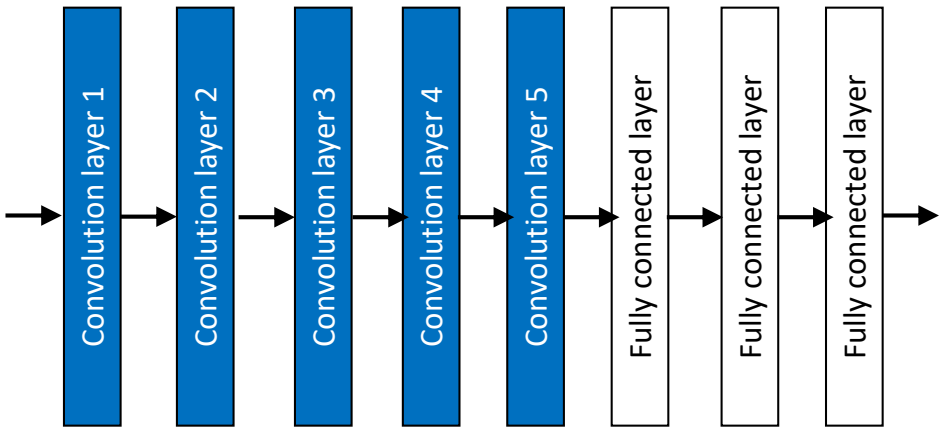
What do neurons in a CNNs learn?

- **Visualization:**
  View neurons as pattern detectors -- find patches that activate a neuron, and see whether they classify a “concept”
- What would you expect? Hierarchical representations
  edges -> shapes/textures -> parts -> objects

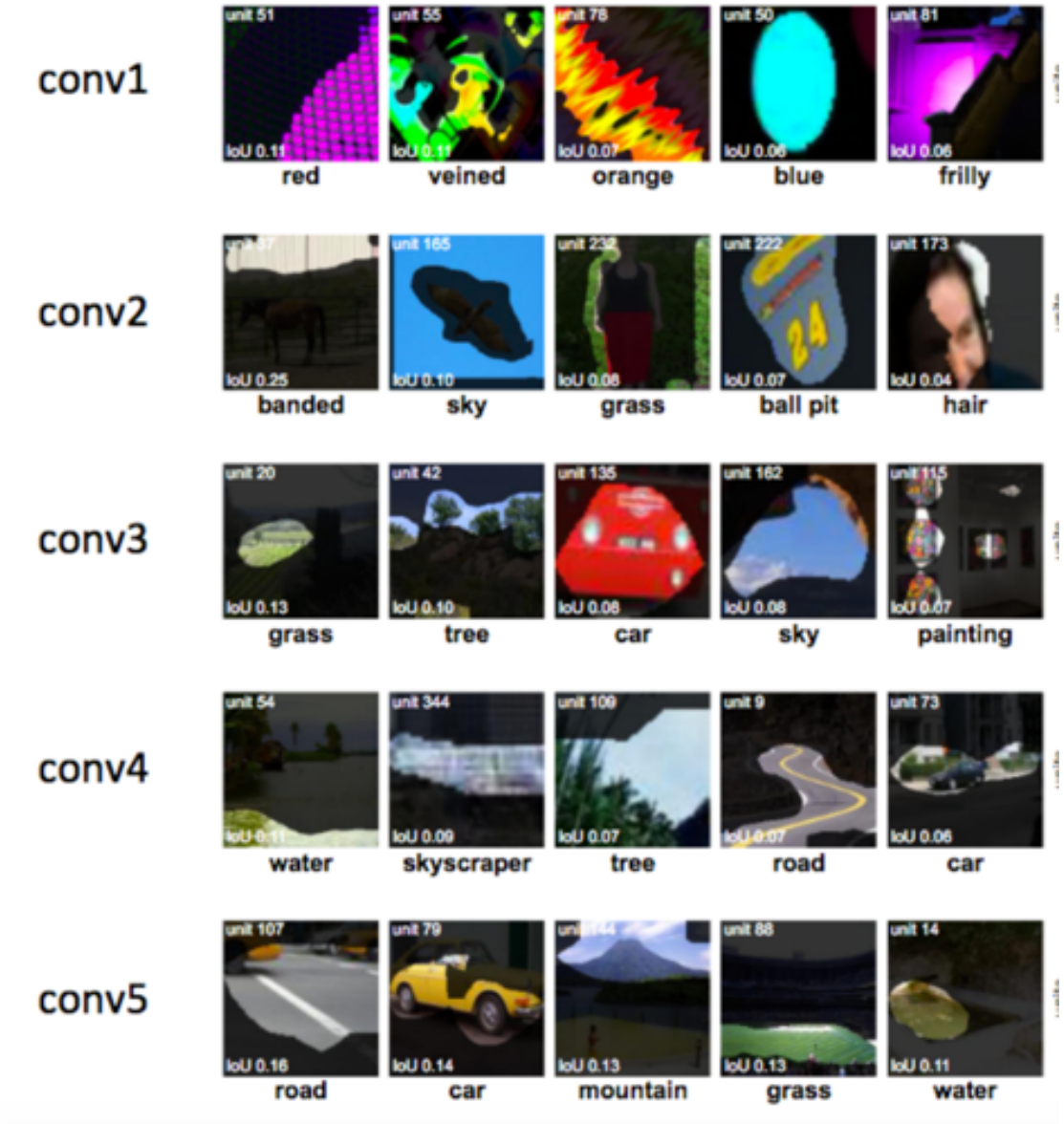


What do neurons in a CNNs learn?

- **Visualization:**
  View neurons as pattern detectors -- find patches that activate a neuron, and see whether they classify a “concept”
- What would you expect? Hierarchical representations
  edges -> shapes/textures -> parts -> objects
- Example: train AlexNet to classify scenes



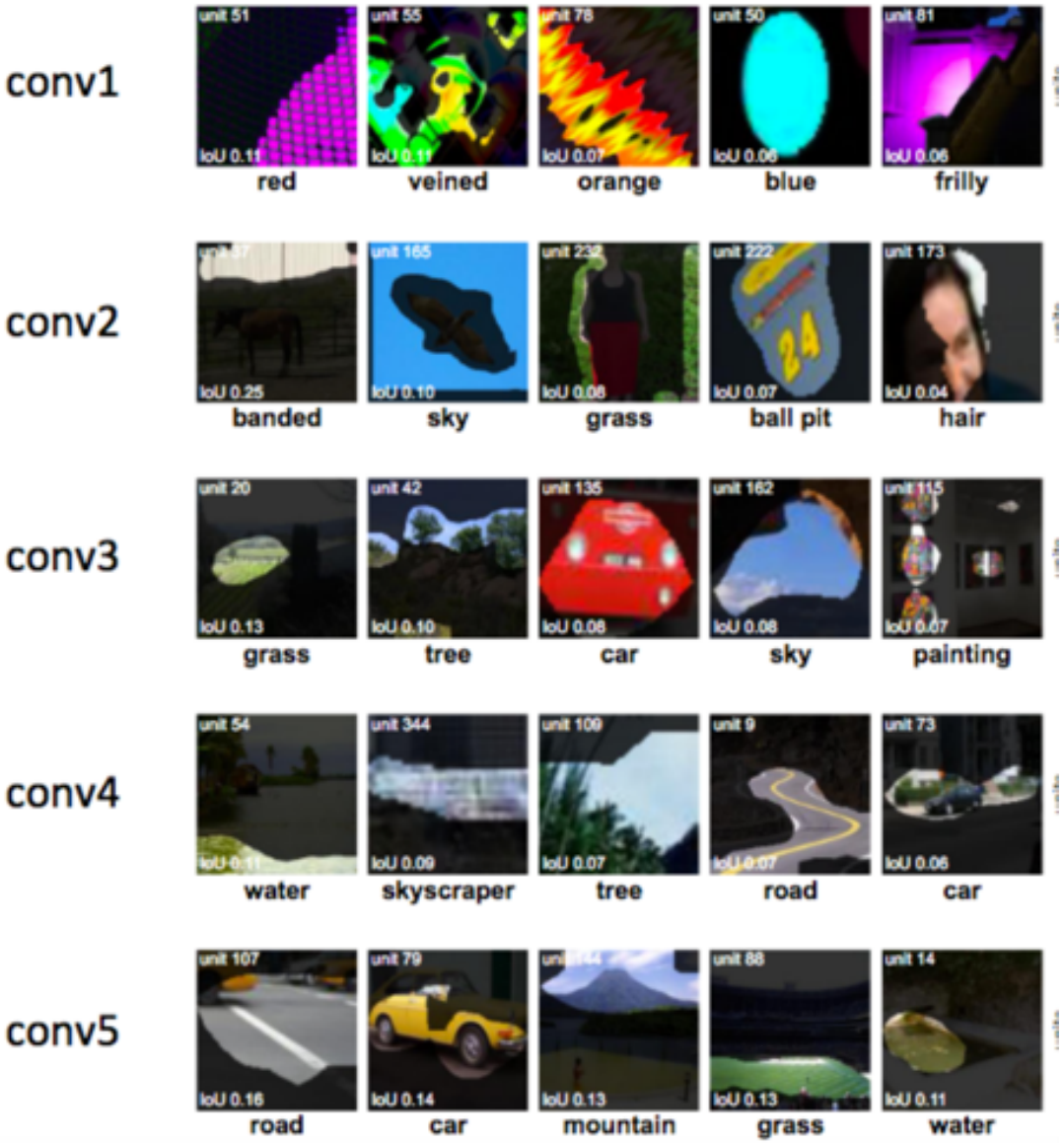
What do neurons in a CNNs learn?

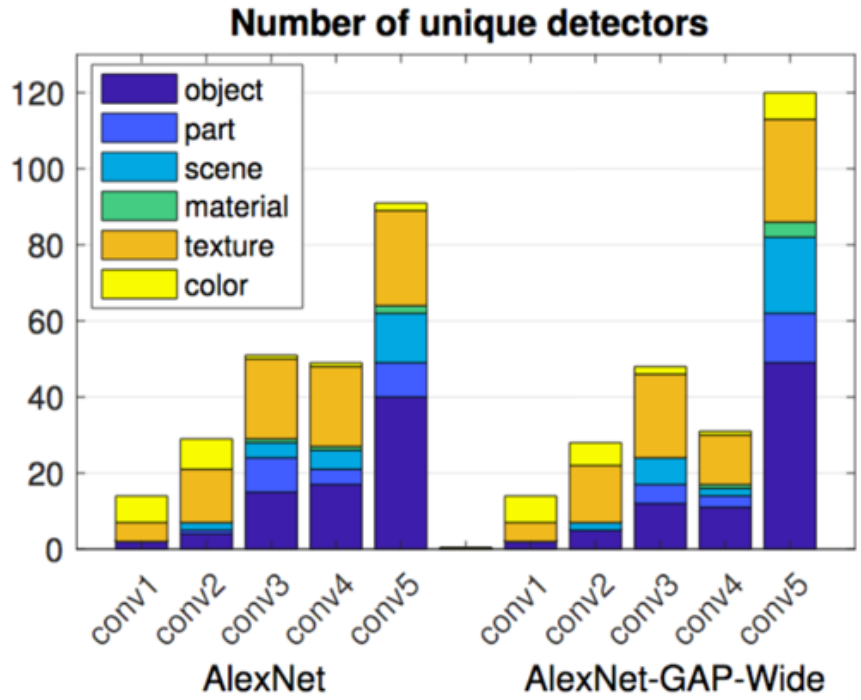


First layer(s): detectors for
colors, edges, textures
Later layers: detectors for
parts, objects, scenes

http://netdissect.csail.mit.edu/

What do neurons in a CNNs learn?





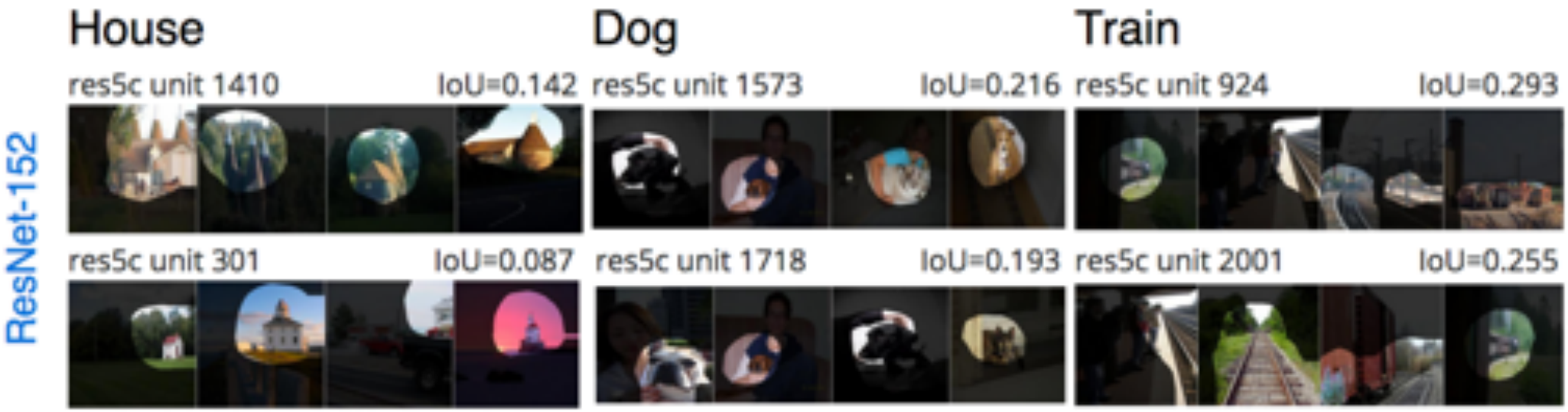
First layer(s): detectors for
colors, edges, textures
Later layers: detectors for
parts, objects, scenes

http://netdissect.csail.mit.edu/

Example: scene detection

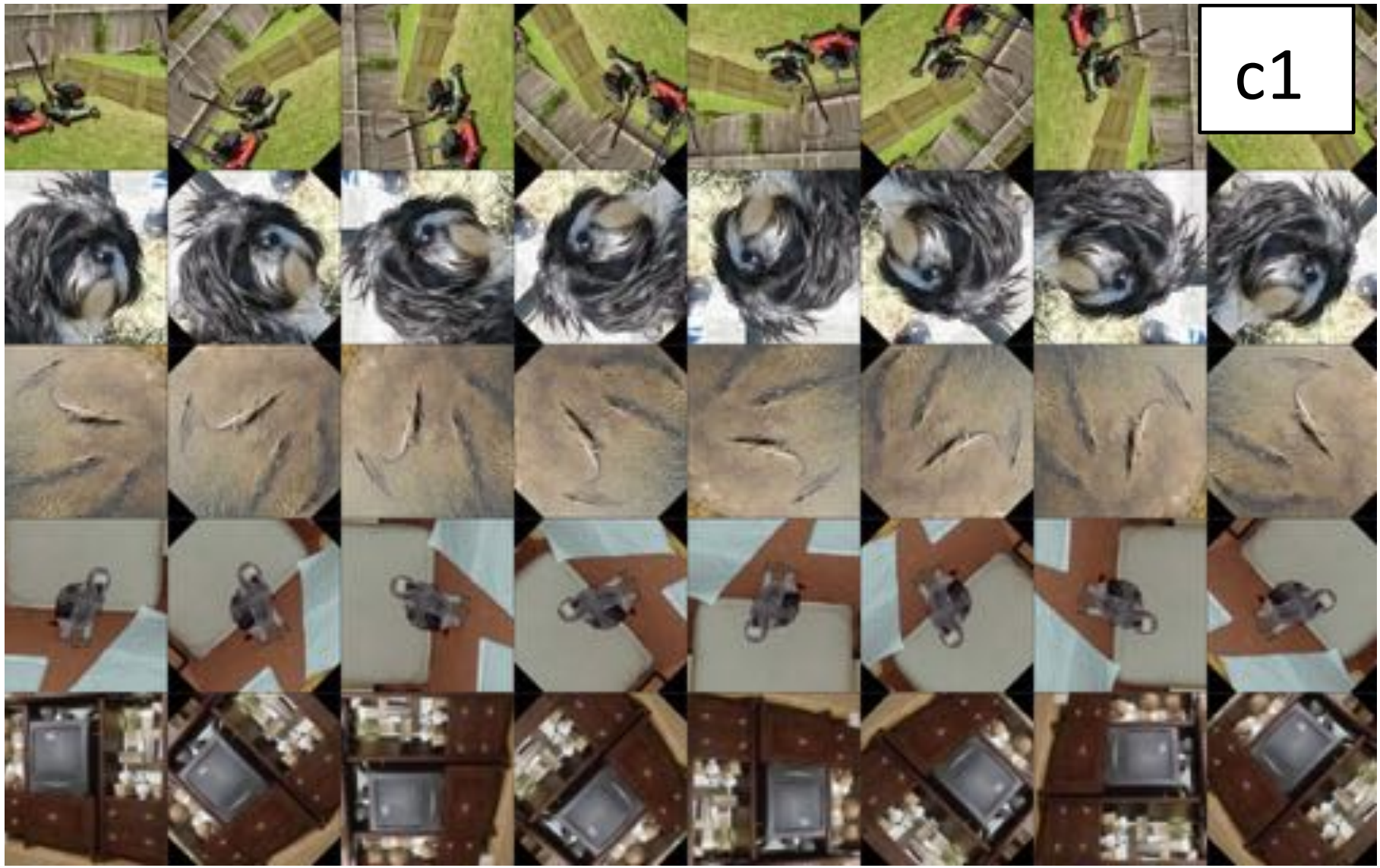
When you learn to classify scenes …

… the network learns object detectors!



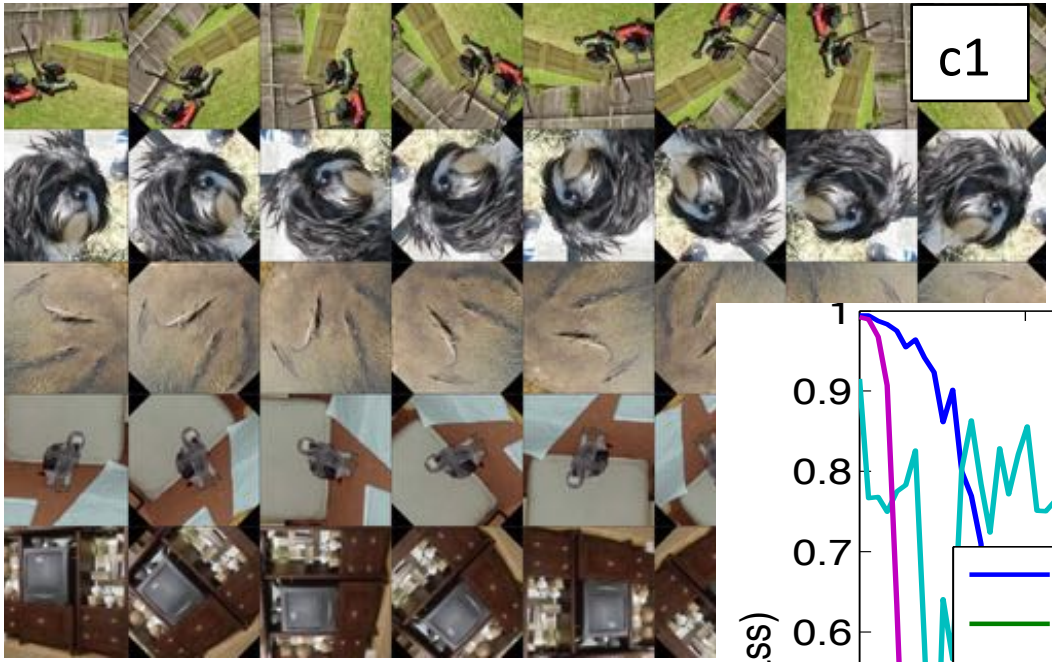
http://netdissect.csail.mit.edu/

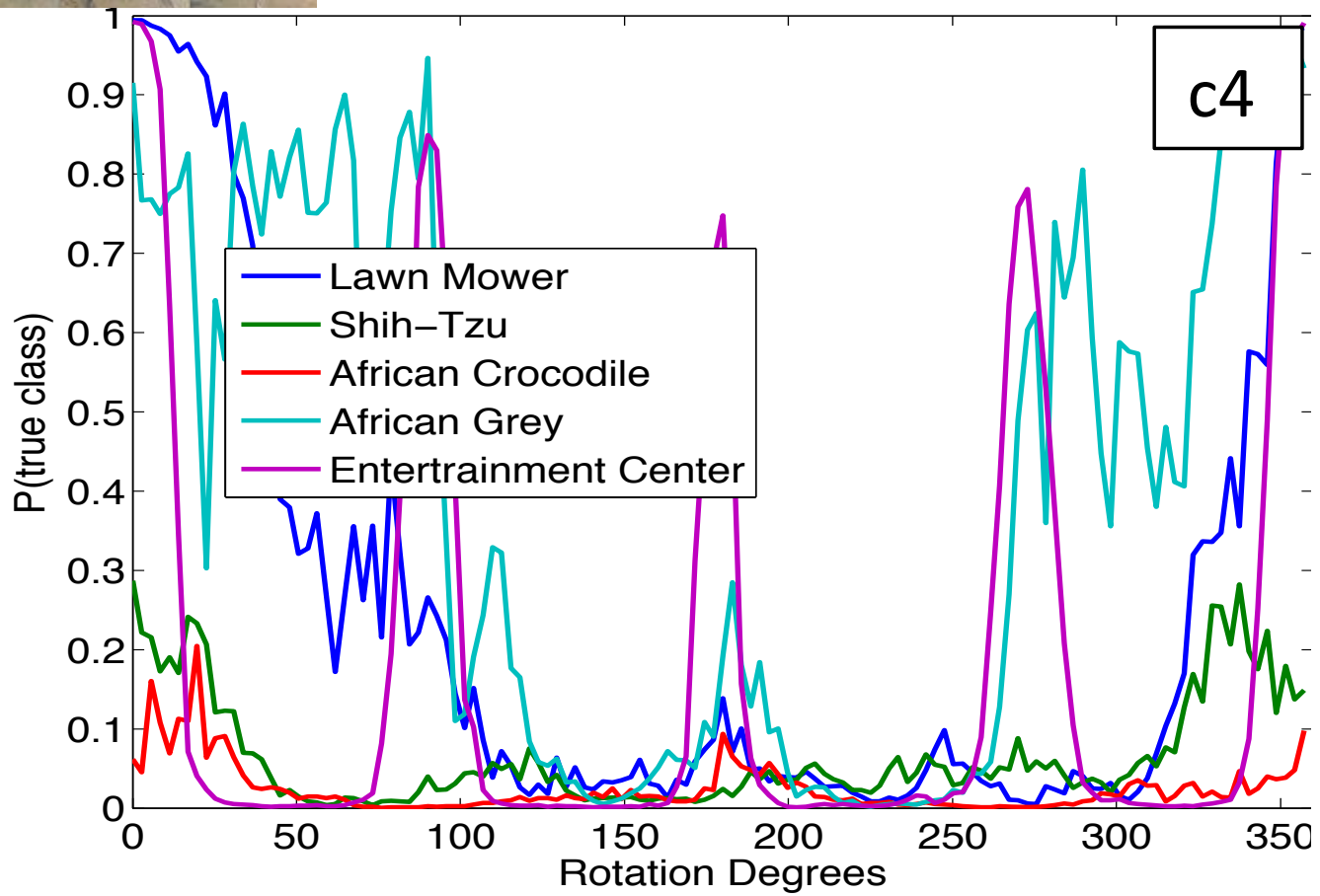
Input rotations



Zeiler & Fergus 2013

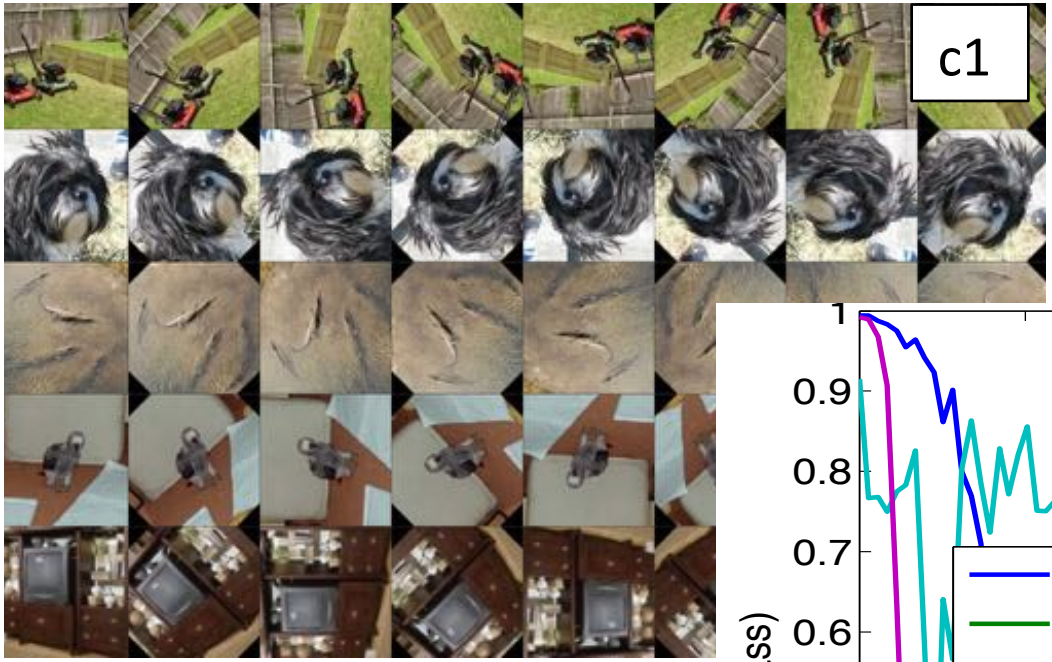
Input rotations

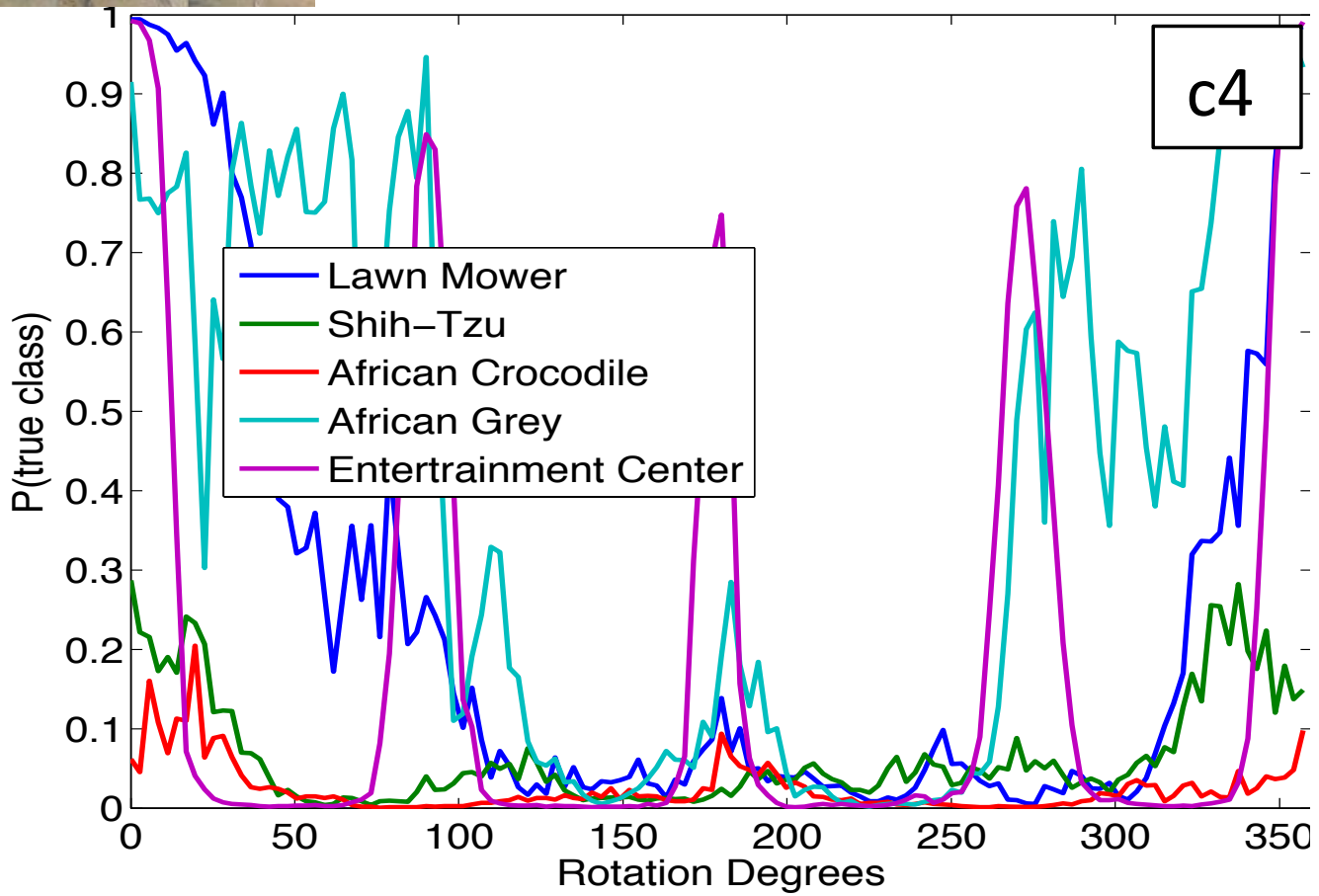




Zeiler & Fergus 2013

Input rotations



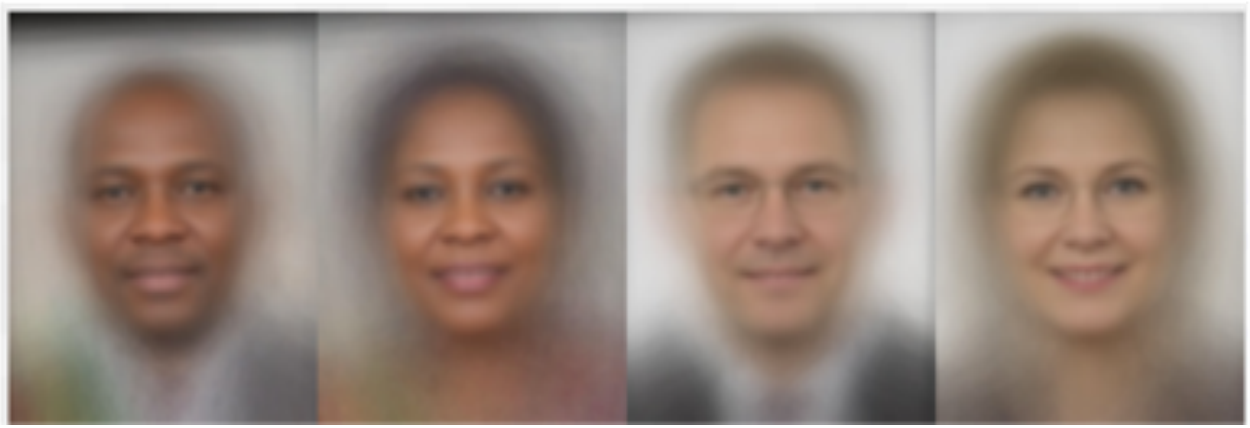


CNNs can fail when the input is rotated.

Zeiler & Fergus 2013

Bias

Gender Classifier	Darker Male	Darker Female	Lighter Male	Lighter Female	Largest Gap
Microsoft	94.0%	79.2%	100%	98.3%	20.8%
FACE++	99.3%	65.5%	99.2%	94.0%	33.8%
IBM	88.0%	65.3%	99.7%	92.9%	34.4%



Errors can be very different for different classes.

http://gendershades.org/overview.html

Are we using the right features?

"car"








Are we using the right features?

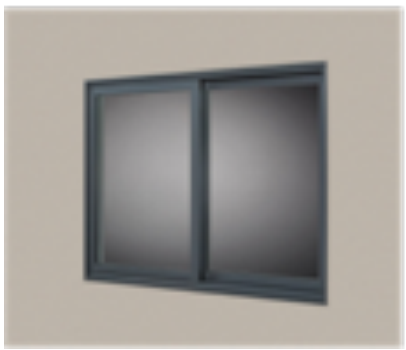
“car”








“not a car”




Are we using the right features?

“Shortcut solutions”

Good test performance within distribution, but bad under distribution shifts

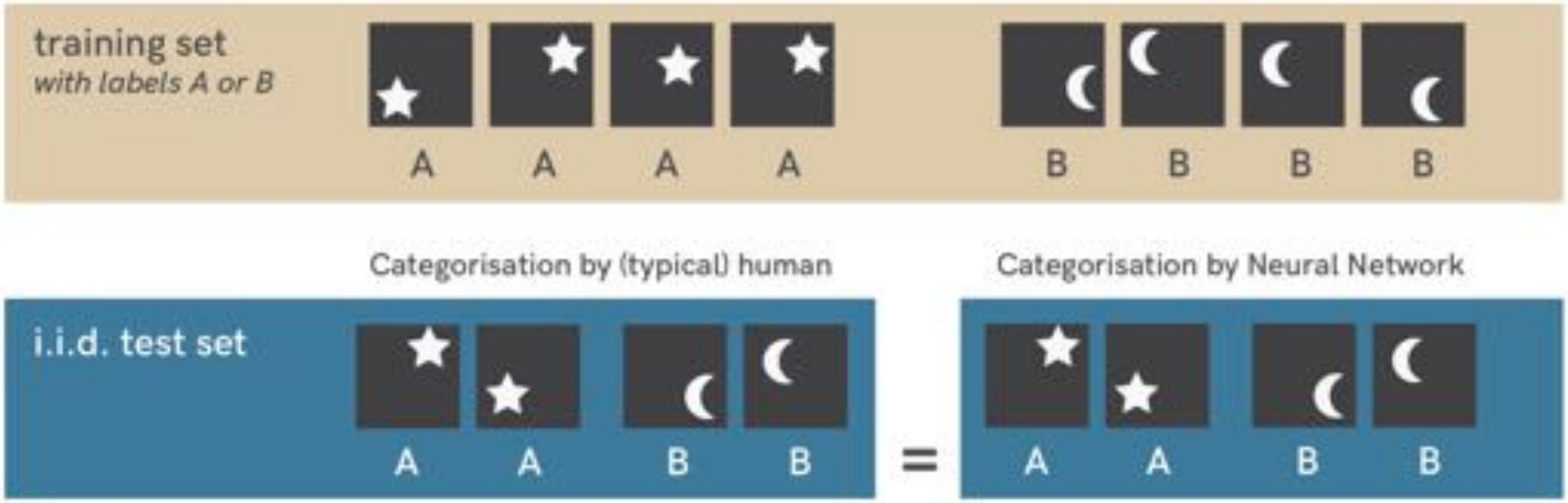


Illustration: Geirhos et al. Shortcut Learning in Deep Neural Networks. https://arxiv.org/pdf/2004.07780.pdf

“Shortcut solutions”

Good test performance within distribution, but bad under distribution shifts

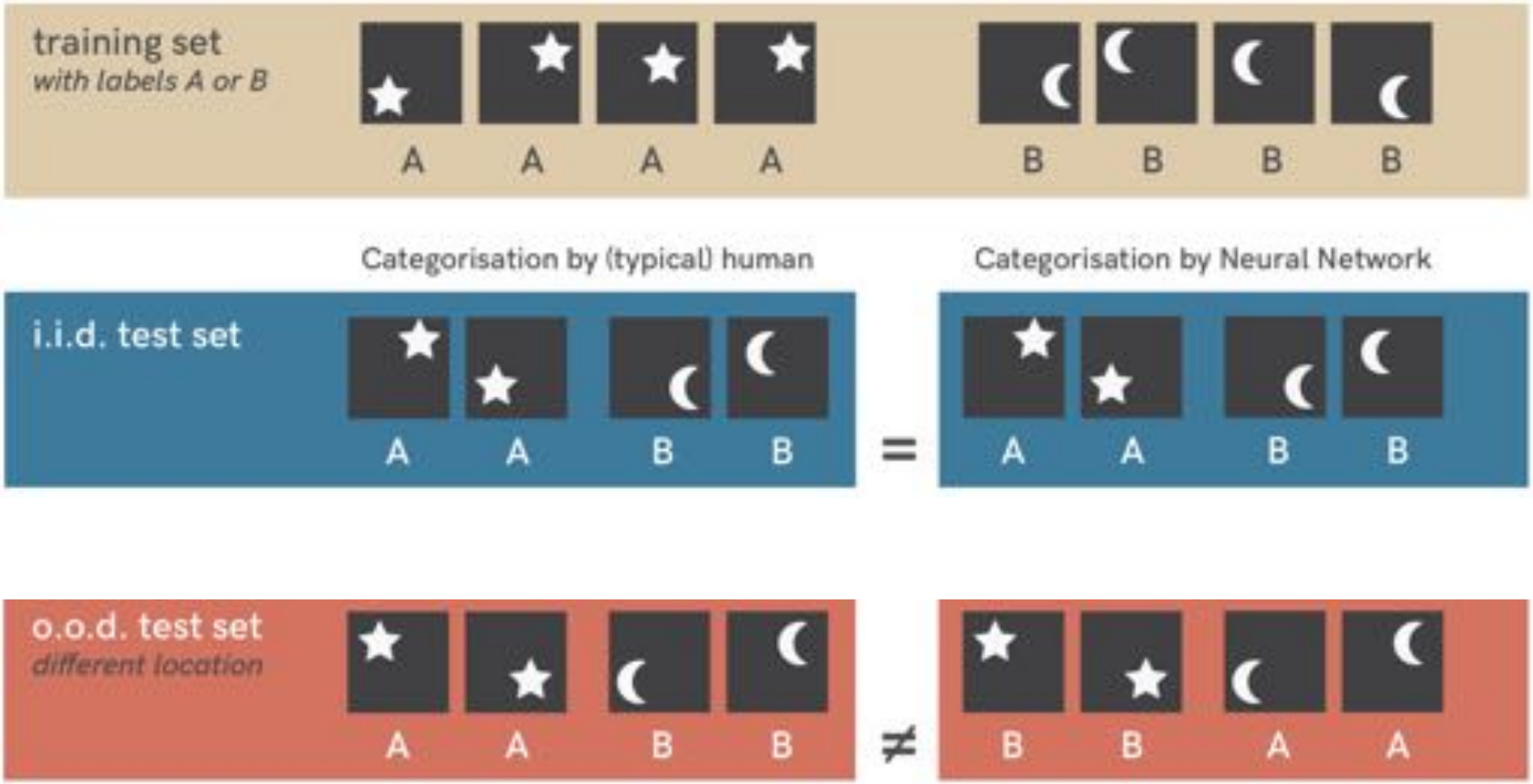
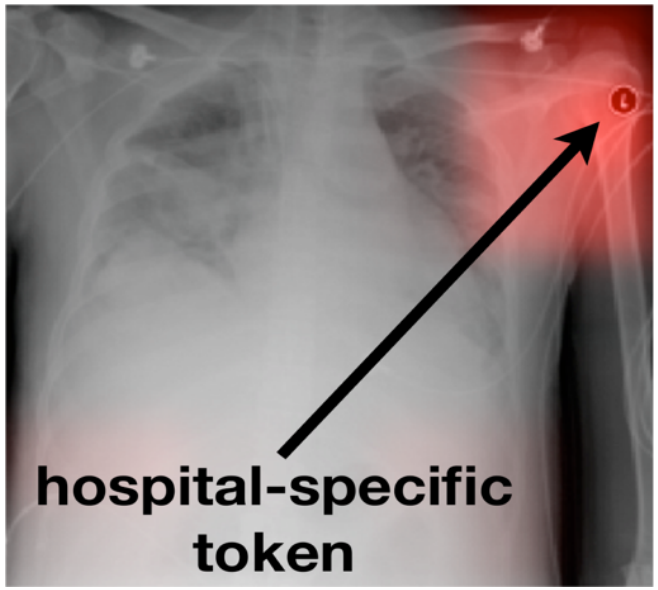


Illustration: Geirhos et al. Shortcut Learning in Deep Neural Networks. https://arxiv.org/pdf/2004.07780.pdf

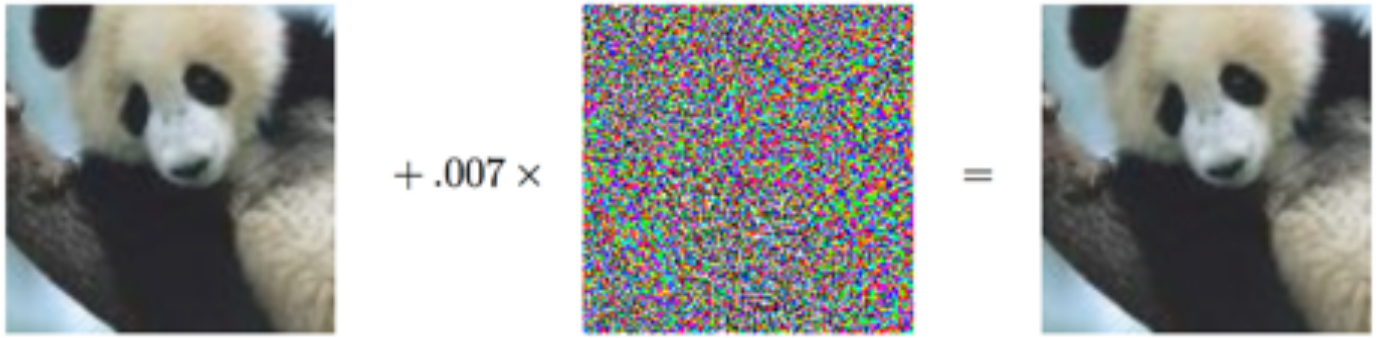
“Shortcuts” more generally

Detecting pneumonia from x-rays



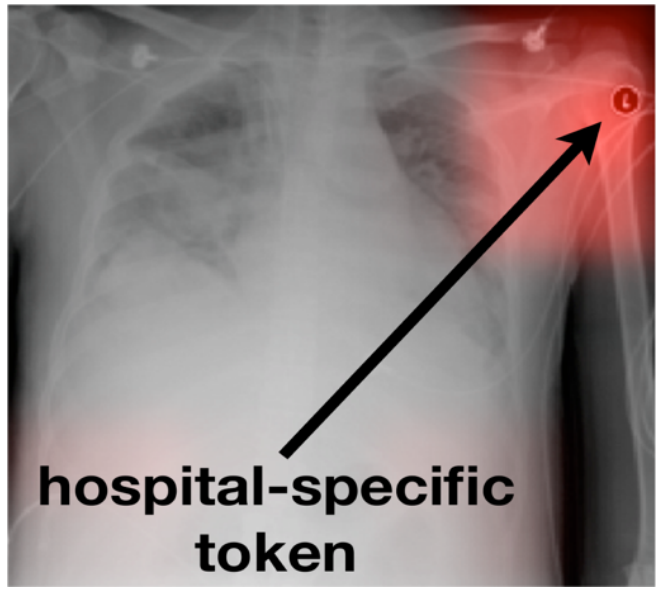
https://thegradient.pub/shortcuts-neural-networks-love-to-cheat/

“Shortcuts” more generally



(Goodfellow et al 2014,…)

Detecting pneumonia from x-rays

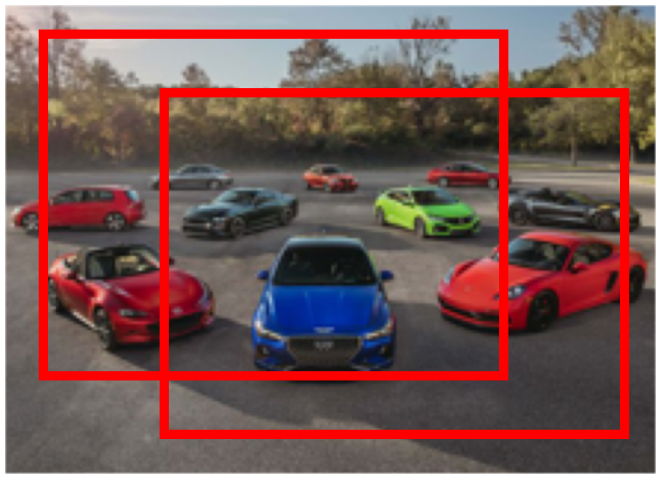


https://thegradient.pub/shortcuts-neural-networks-love-to-cheat/

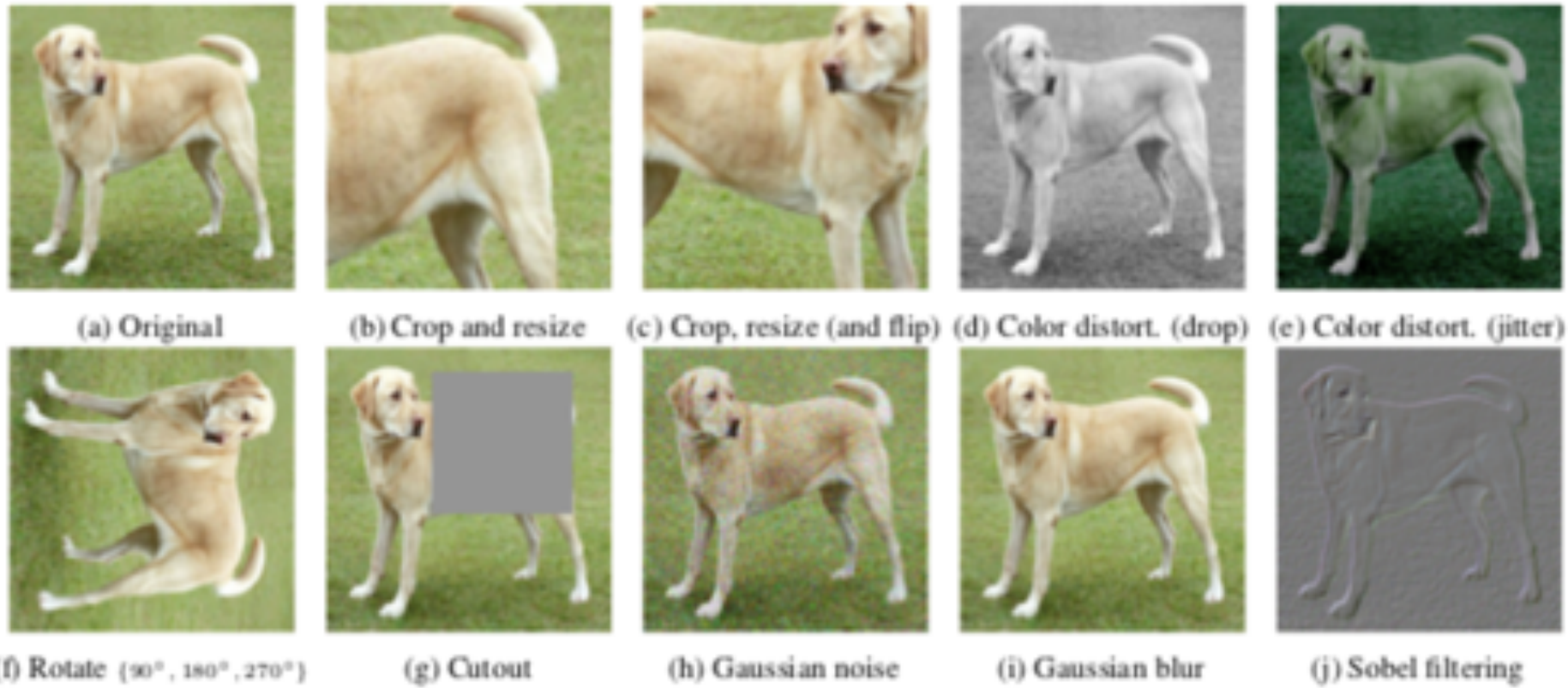
Are we using the right features?

What can we do?

- collect more data
- data augmentation (cropping, rotation, …)
- pre-training
- …



Examples of data transformations



(Chen, Kornblith, Norouzi, Hinton 2020)

Selection of tools: e.g. Tensorflow, Keras preprocessing layers

Summary: CNNs

- Neural Network for images
- **Convolution: "local detectors"**
  spatial locality, efficiency via weight sharing
- **Pooling**: abstract away locality
- Trained with SGD and backpropagation
- Modern CNNs are deep

- Data augmentation to improve performance

regul: $\|\theta\|_2^2 = \sum_{j=1}^{d} \theta_j^2$ $\|\theta\|_1 = \sum_{j=1}^{d} |\theta_j|$