SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

Convolutional Neural Networks

PROF LIM KWAN HUI

50.021 Artificial Intelligence

Outline & Objectives

- Be able to use neural networks for generating word representations, e.g., Word2Vec
- Have a general understanding of how sequence models work, including RNNs and LSTMs

Last week

- Understand how convolution neural networks work
- Be able to apply the various convolution-related operations in a simple example

This week

Computer vision tasks

Image Classification

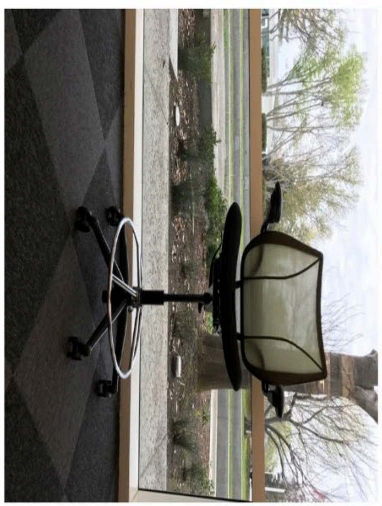
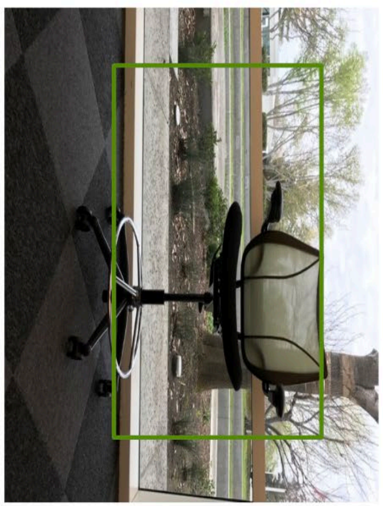
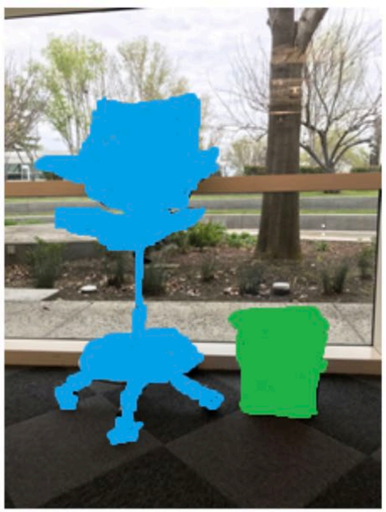


Image Classification + Localization



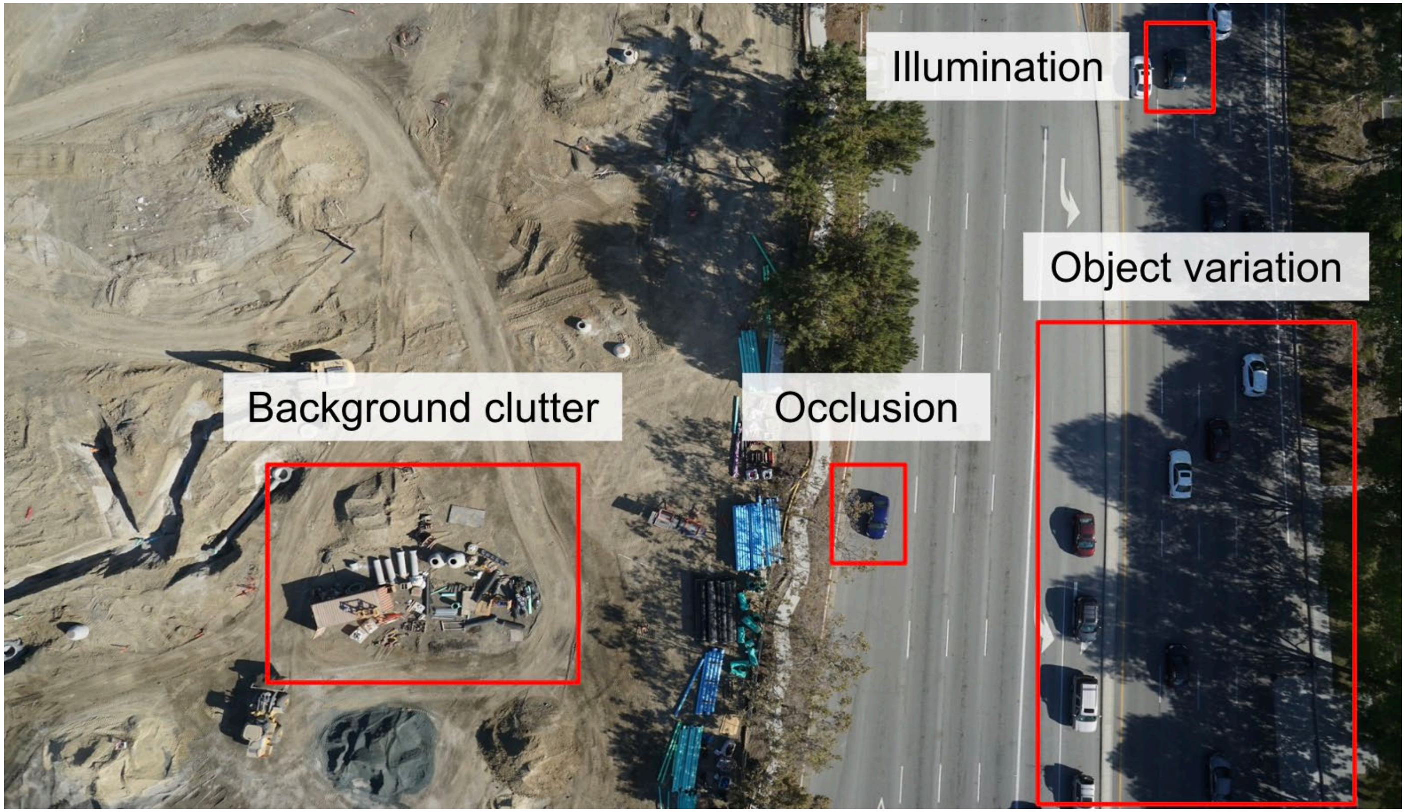
Object Detection

Image Segmentation

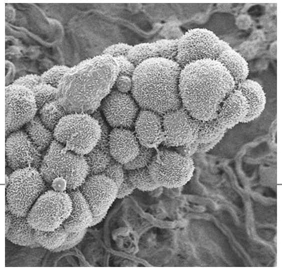
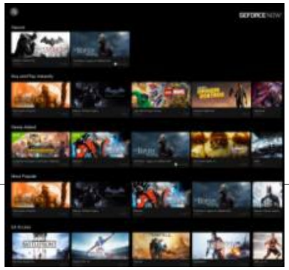


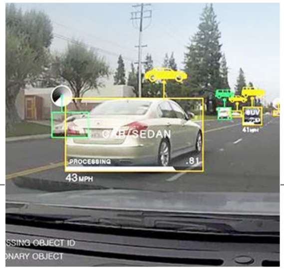
(inspired by a slide found in cs231n lecture from Stanford University)

Challenges in images



Deep learning with CNNs everywhere



INTERNET & CLOUD

Image Classification
Speech Recognition
Language Translation
Language Processing
Sentiment Analysis
Recommendation

MEDICINE & BIOLOGY

Cancer Cell Detection
Diabetic Grading
Drug Discovery

MEDIA & ENTERTAINMENT

Video Captioning
Video Search
Real Time Translation

SECURITY & DEFENSE

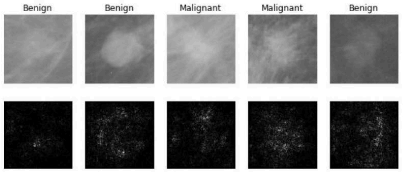
Face Detection
Video Surveillance
Satellite Imagery

AUTONOMOUS MACHINES

Pedestrian Detection
Lane Tracking
Recognize Traffic Sign

[Toshev, Szegedy 2014]

[Guo et al. 2014]



[Levy et al. 2016]

[Dieleman et al. 2014]

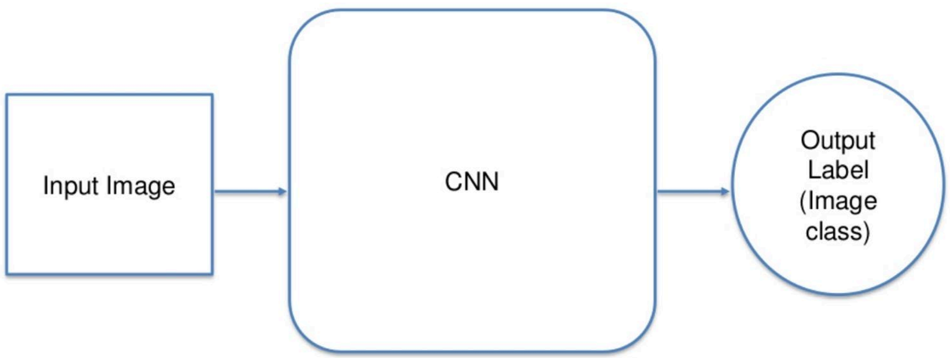
[Sermanet et al. 2011]
[Ciresan et al.]

Convolutional NN

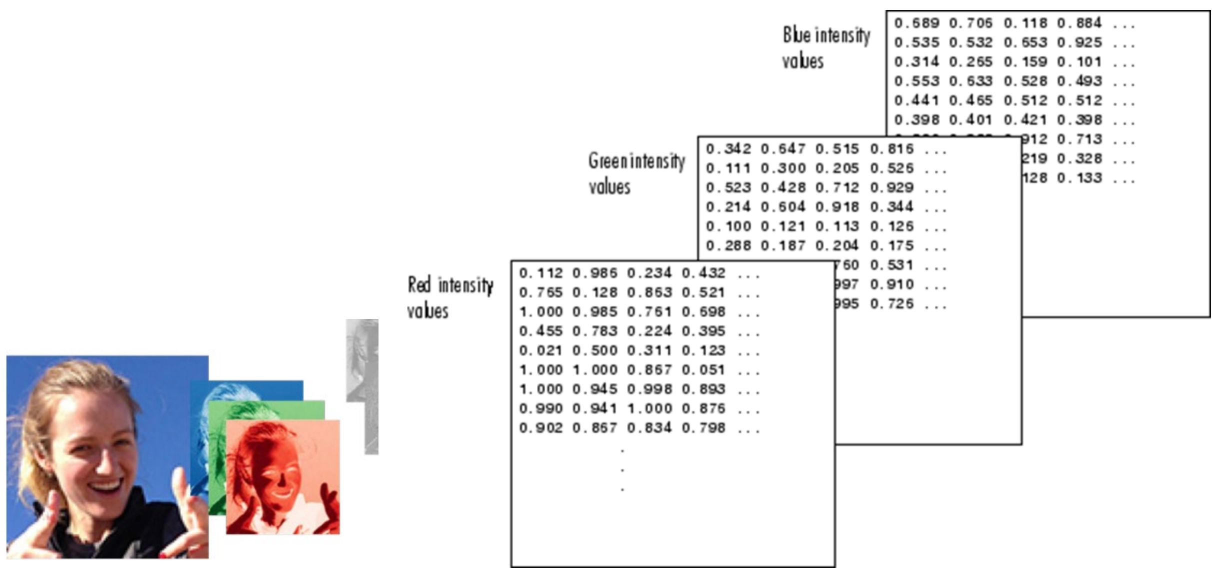
- LeCun, 1989 (Chief scientist FB)
- "…are a specialized kind of neural network for processing data that has a known **grid-like** topology. *Examples* include *time-series data*, which can be thought of as a 1-D grid taking samples at regular time intervals, and *image data*, which can be thought of as a 2-D grid of pixels. Convolutional networks have been tremendously successful in practical applications. The name "convolutional neural network" indicates that the network employs a **mathematical operation called convolution**. Convolution is a specialized kind of linear operation." (Goodfellow et al., 2016)

Convolutional neural networks

- Neural networks that use convolution in place of general matrix multiplication in at least one of their layers.



Images are numbers

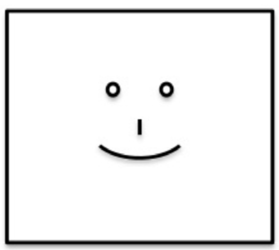


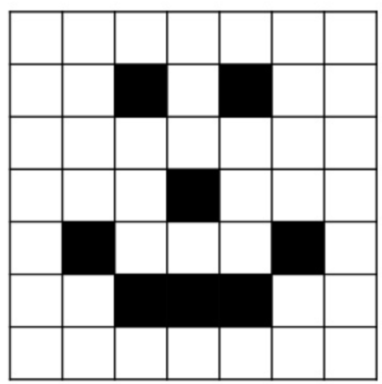
Original image RGB channels

https://blog.datawow.io/interns-explain-cnn-8a669d053f8b

Simple black and white image

- 2D matrix, no grayscale





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

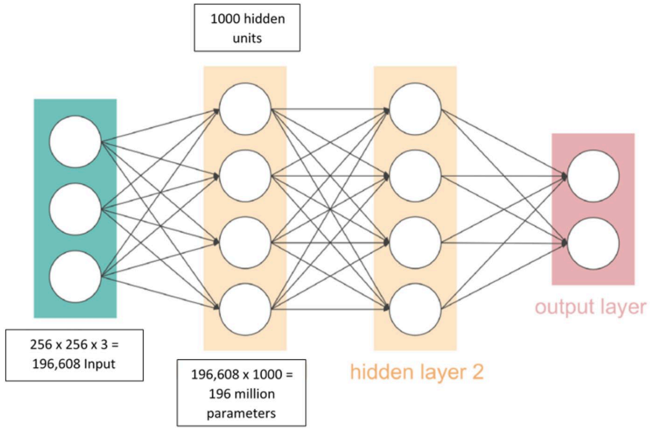
Before convolution

- Original values of a 24-bit color images (True Color):
  - 8-bit per color: 0 – 255.
  - Total: 256 * 256 * 256 = 16,777,216 colors
  - Value for red, green, and blue.
- Preprocessing color values: normalized between 0 and 1
  -> will increase performance

Simple FC Network

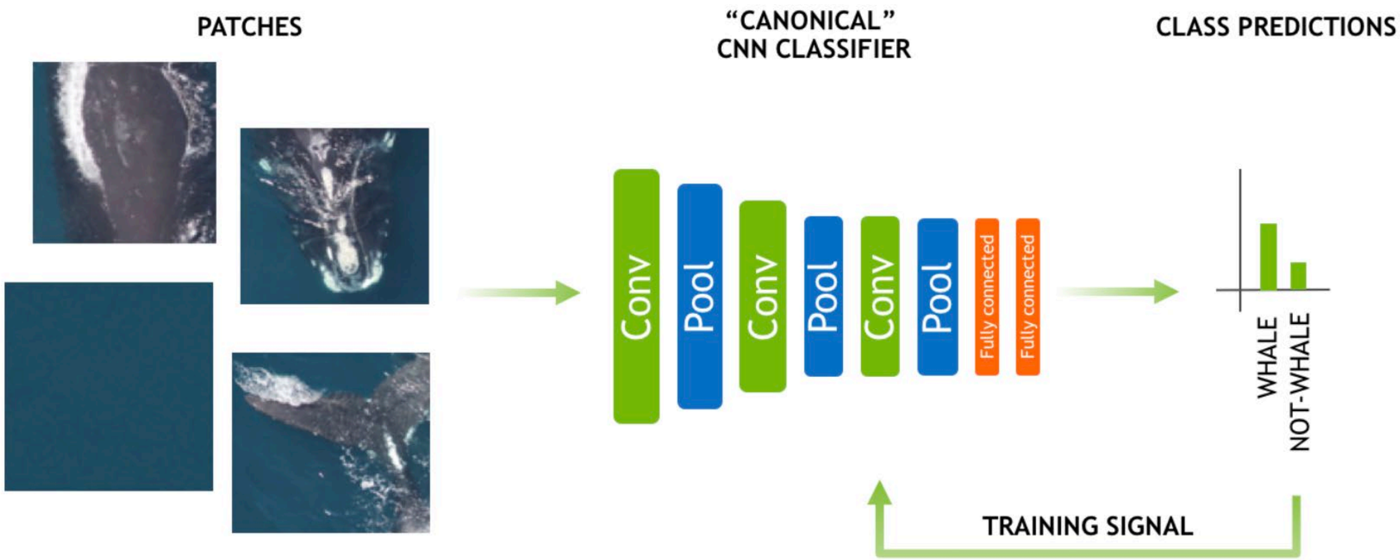
- A color image with size 300 x 300 would have 300 x 300 x 3 input values which is equal to 270,000 inputs. If, for example, we have 1,000 hidden units in our first hidden layer, there would be approximately 270 million parameters or weights for us to train which is infeasible.

- High chance of overfitting and highly complex network

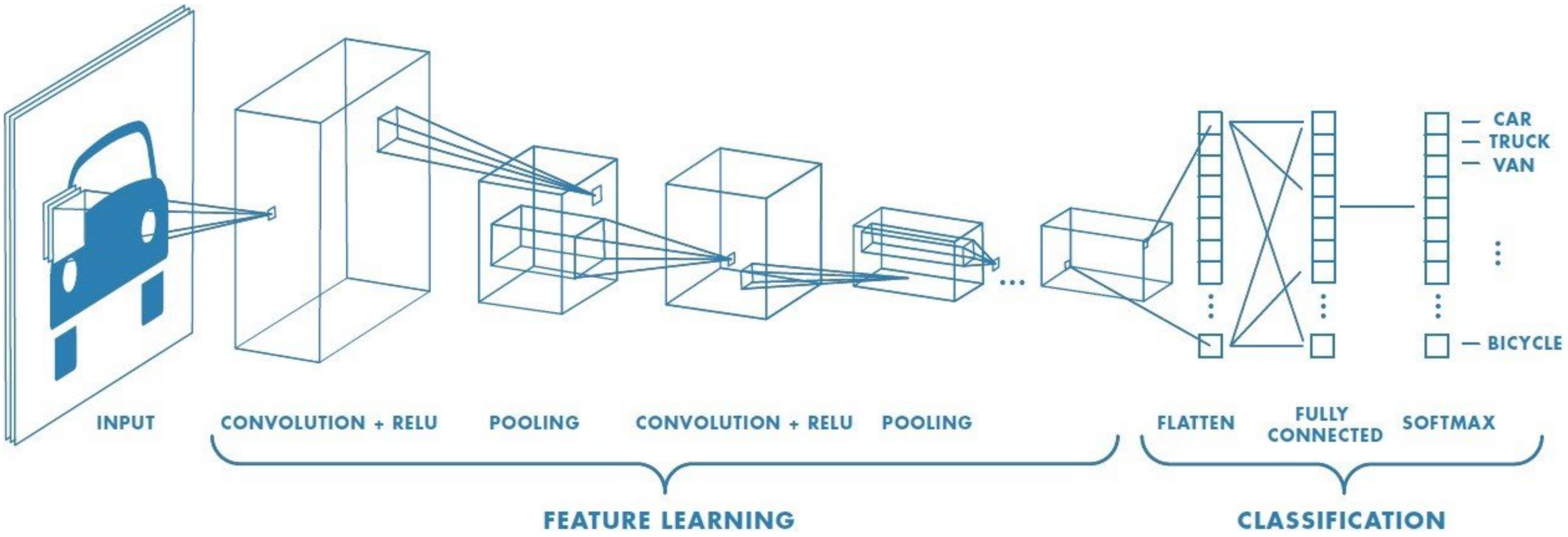


Solution: convolution

- Reduces the number of parameters we need to learn.
- Preserves locality. We don't have to flatten the image matrix into a vector, thus the relative positions of the image pixels are preserved.

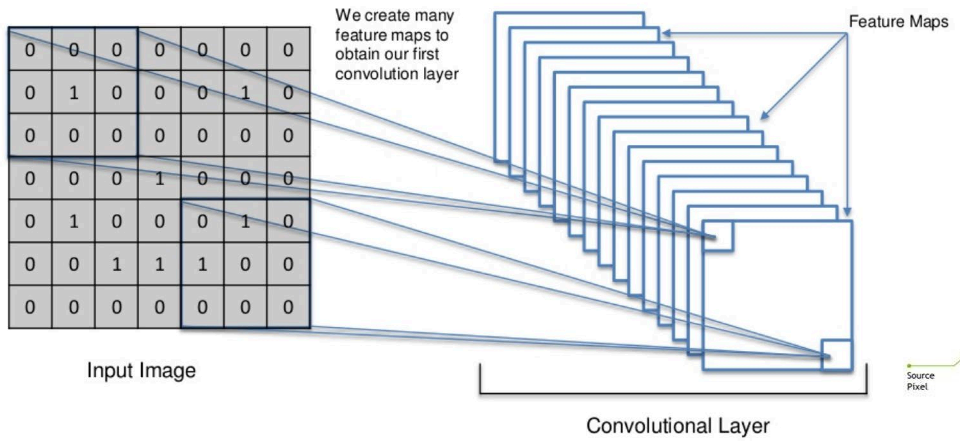


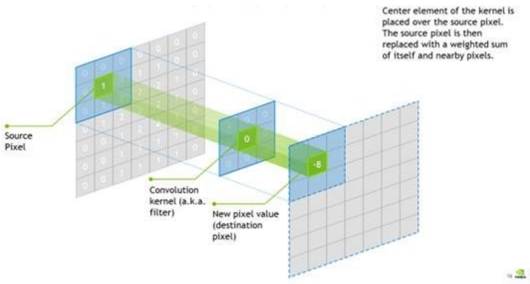
A typical CNN



Convolutional layer

- Many feature maps are created, using filters (also called kernels).
- Kernels (or filters) are learned to best fit the task at hand.





[Image source](#)

Convolution - example

- Edge detection filter/kernel

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

6 x 6

*

1	0	-1
1	0	-1
1	0	-1

3 x 3
filter

=

Convolution - example

- Edge detection filter/kernel

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

6 x 6

*

1	0	-1
1	0	-1
1	0	-1

3 x 3
filter

=

Convolution - example

- Edge detection filter/kernel

$= 3\text{x}1 + 1\text{x}1 + 2\text{x}1 + 0\text{x}0 + 5\text{x}0 + 7\text{x}0 + 1\text{x}(-1) + 8\text{x}(-1) + 2\text{x}(-1)$

$$\begin{array}{cccccc} \boxed{\begin{array}{ccc} 3 & 0 & 1 \\ 1 & 5 & 8 \\ 2 & 7 & 2 \end{array}} & \begin{array}{ccc} 2 & 7 & 4 \\ 9 & 3 & 1 \\ 5 & 1 & 3 \end{array} \\ \begin{array}{ccc} 0 & 1 & 3 \\ 4 & 2 & 1 \\ 2 & 4 & 5 \end{array} & \begin{array}{ccc} 1 & 7 & 8 \\ 6 & 2 & 8 \\ 2 & 3 & 9 \end{array} \end{array} * \begin{array}{ccc} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{array} = \boxed{-5}$$

6 x 6

3 x 3
filter

Convolution - example

- Edge detection filter/kernel, **stride size = 1**

$$\begin{matrix} 3 & 0 & 1 & 2 & 7 & 4 \\ 1 & 5 & 8 & 9 & 3 & 1 \\ 2 & 7 & 2 & 5 & 1 & 3 \\ 0 & 1 & 3 & 1 & 7 & 8 \\ 4 & 2 & 1 & 6 & 2 & 8 \\ 2 & 4 & 5 & 2 & 3 & 9 \end{matrix} \quad * \quad \begin{matrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{matrix} \quad = \quad \begin{matrix} -5 & \boxed{-4} \end{matrix}$$

6 x 6

3 x 3
filter

Convolution - example

- Edge detection filter/kernel, **stride size = 1**

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

6 x 6

*

1	0	-1
1	0	-1
1	0	-1

3 x 3
filter

=

-5	-4	0

Convolution - example

- Edge detection filter/kernel, **stride size = 1**

$$
\underset{6 \times 6}{\begin{matrix} 3 & 0 & 1 & 2 & 7 & 4 \\ 1 & 5 & 8 & 9 & 3 & 1 \\ 2 & 7 & 2 & 5 & 1 & 3 \\ 0 & 1 & 3 & 1 & 7 & 8 \\ 4 & 2 & 1 & 6 & 2 & 8 \\ 2 & 4 & 5 & 2 & 3 & 9 \end{matrix}}
\;*\;
\underset{\substack{3 \times 3 \\ \text{filter}}}{\begin{matrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{matrix}}
\;=\;
\underset{4 \times 4}{\begin{matrix} -5 & -4 & 0 & 8 \\ -10 & -2 & 2 & 3 \\ 0 & -2 & -4 & -7 \\ -3 & -2 & -3 & -16 \end{matrix}}
$$

Convolution - exercise

- Edge detection filter/kernel, **stride size = 1**

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

6 x 6

*

1	0	-1
1	0	-1
1	0	-1

3 x 3
filter

=

?

? x ?

Convolution - exercise

- Edge detection filter/kernel, **stride size = 1**

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

6 x 6

*

1	0	-1
1	0	-1
1	0	-1

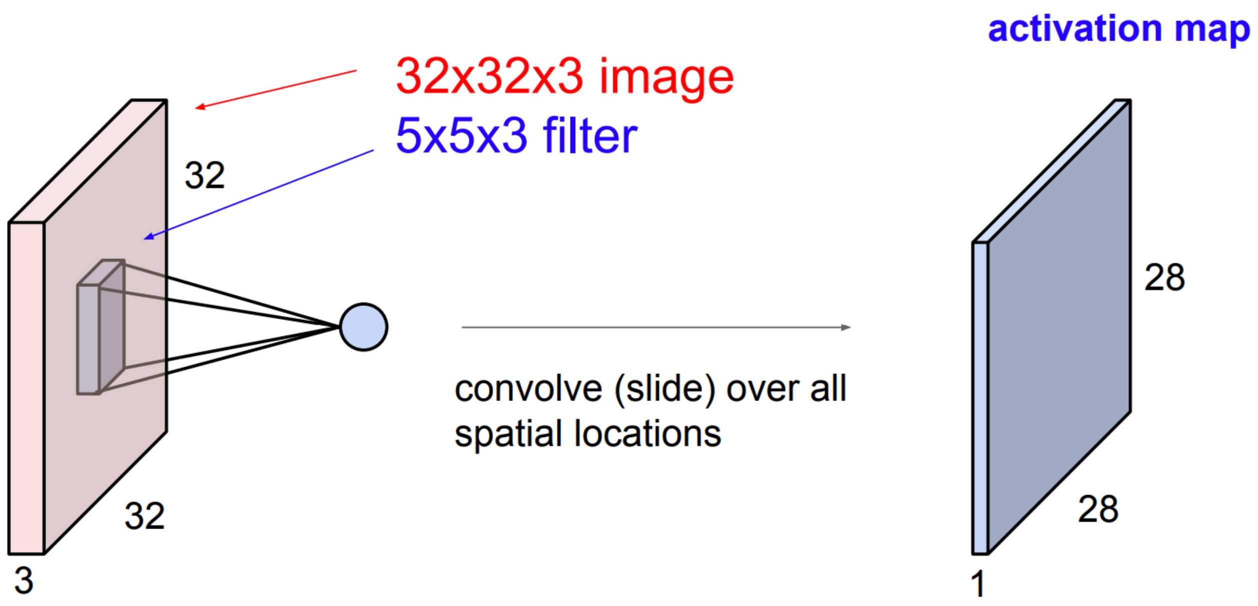
3 x 3 filter

=

0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0

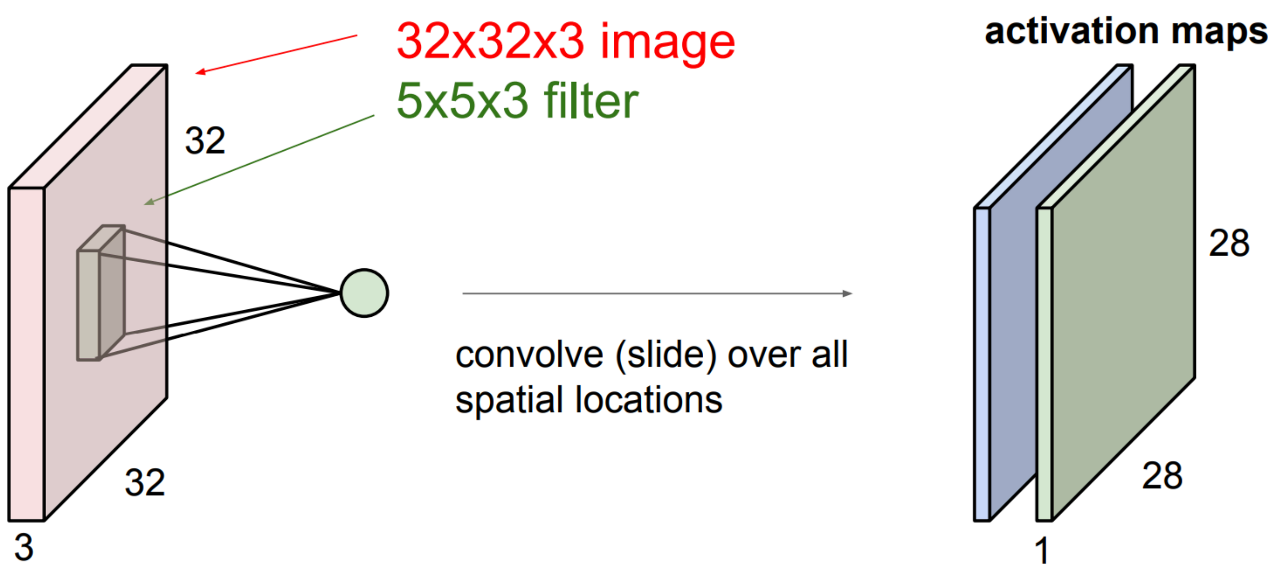
4 x 4

Activation maps



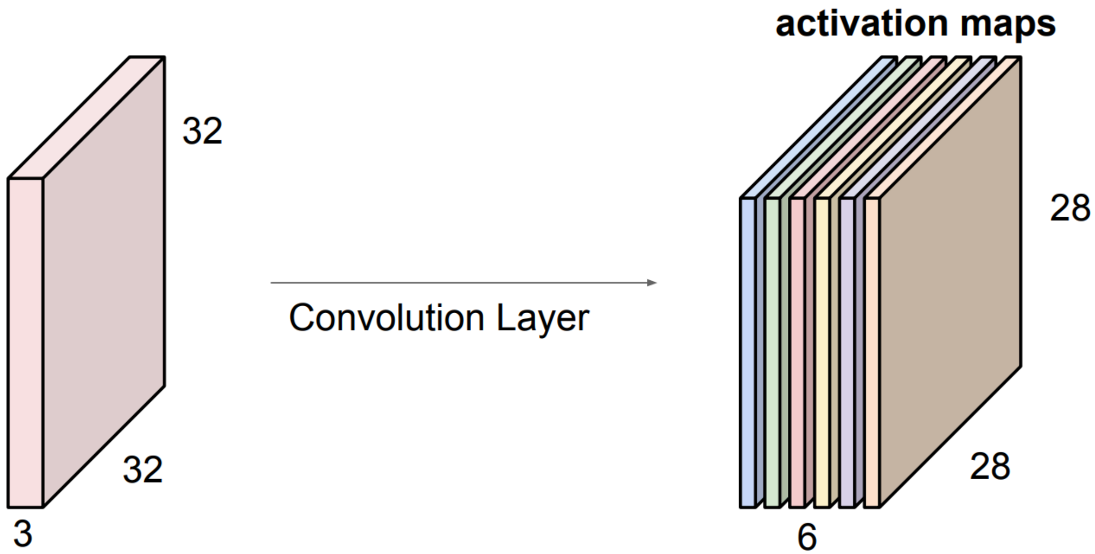
Activation maps

- Each filter creates an activation map



Activation maps

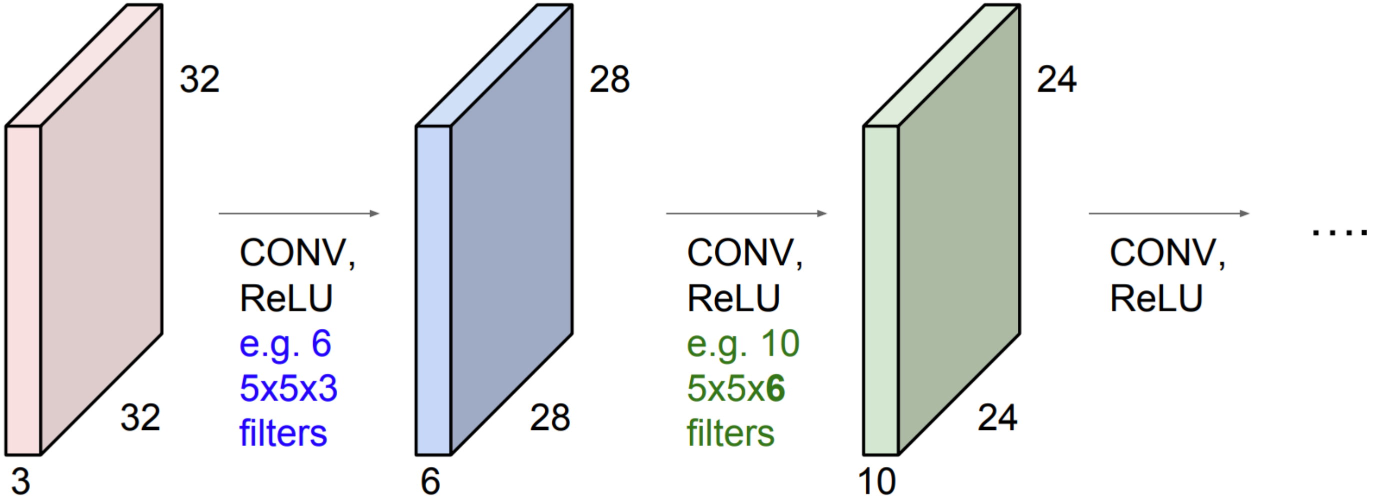
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



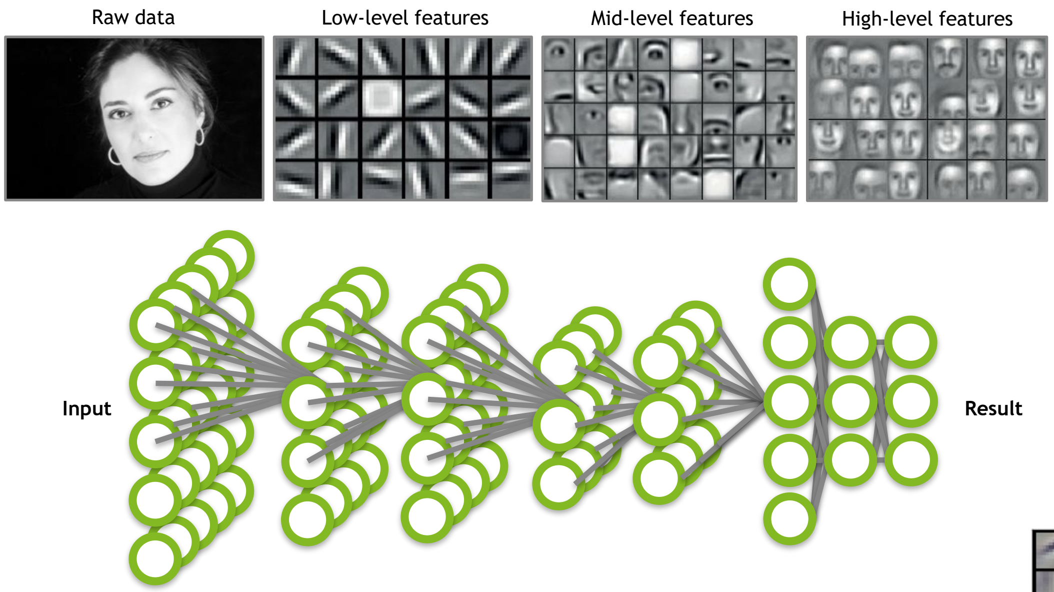
We stack these up to get a "new image" of size 28x28x6!

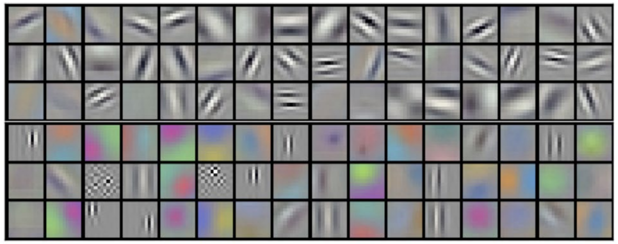
ConvNet

Preview: ConvNet is a sequence of Convolutional Layers, interspersed with activation functions



Different level of filters

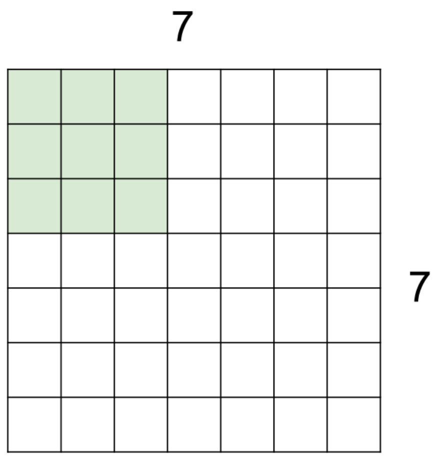




http://matlabtricks.com/post-5/3x3-convolution-kernels-with-online-demo

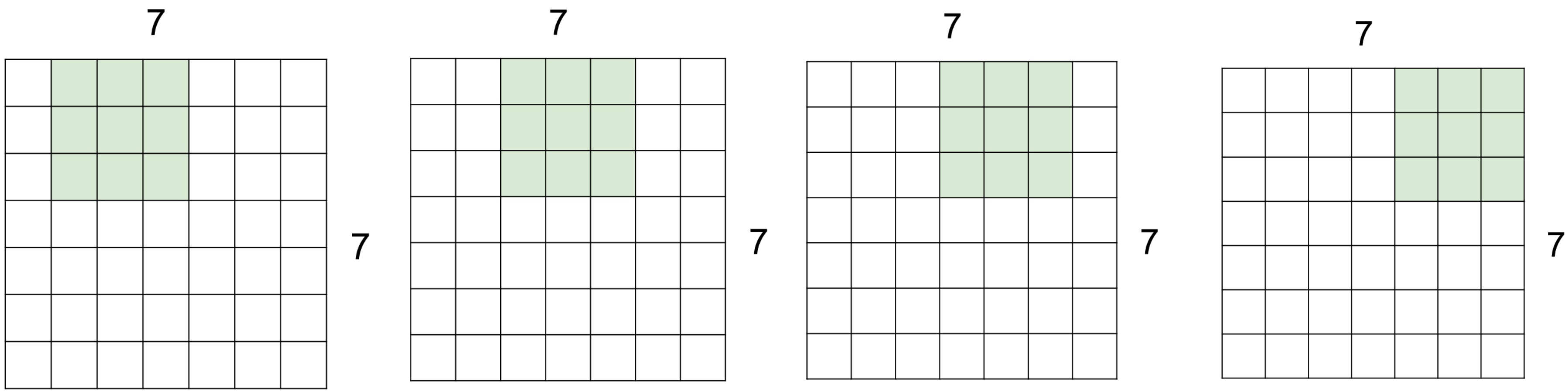
A closer look at dimensions

- A 7x7 input with a 3x3 filter:



A closer look at dimensions

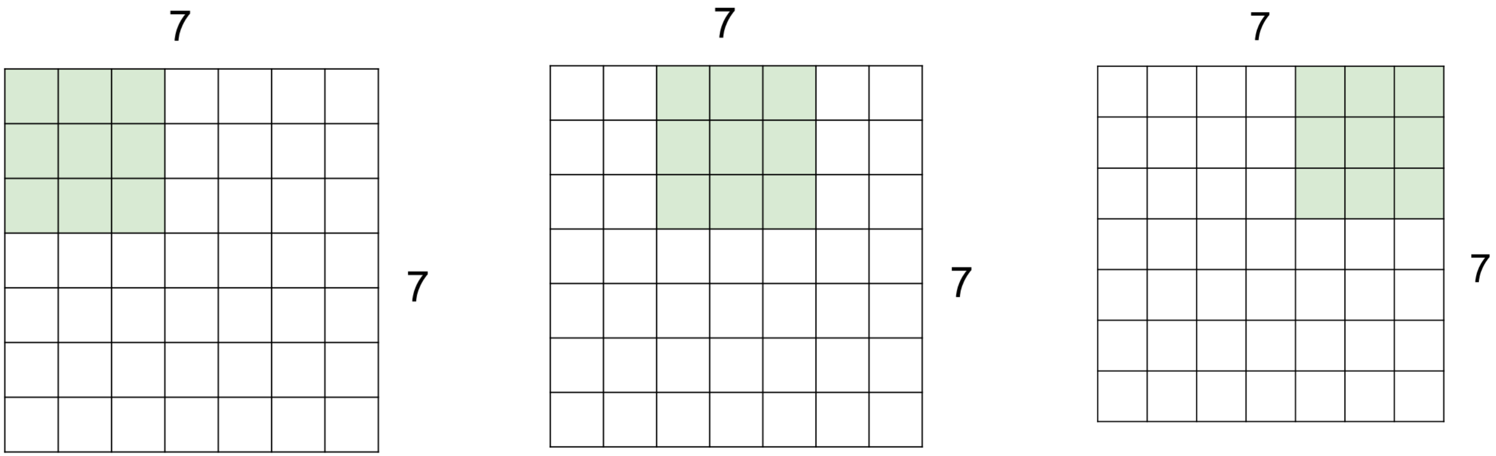
- A 7x7 input with a 3x3 filter, stride = 1



Result: 5x5 output!

A closer look at dimensions

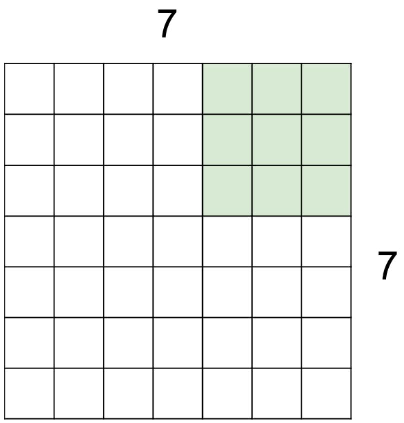
- A 7x7 input with **stride 2** and a 3x3 filter:



- 3 x 3 output!

Ex: A closer look at dimensions

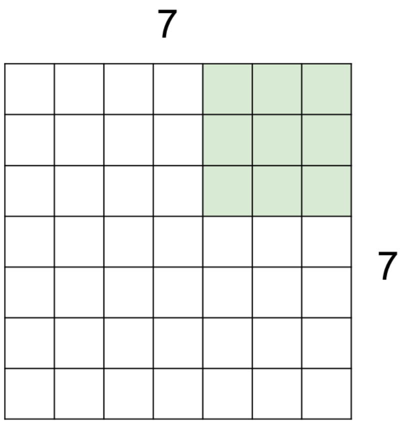
- A 7x7 input with stride 3? What is the output size?
- [(N - F) / stride] + 1



- Exercise: N = 7, F = 3:
  - stride 1 => ?
  - stride 2 => ?
  - stride 3 => ?

Ex: A closer look at dimensions

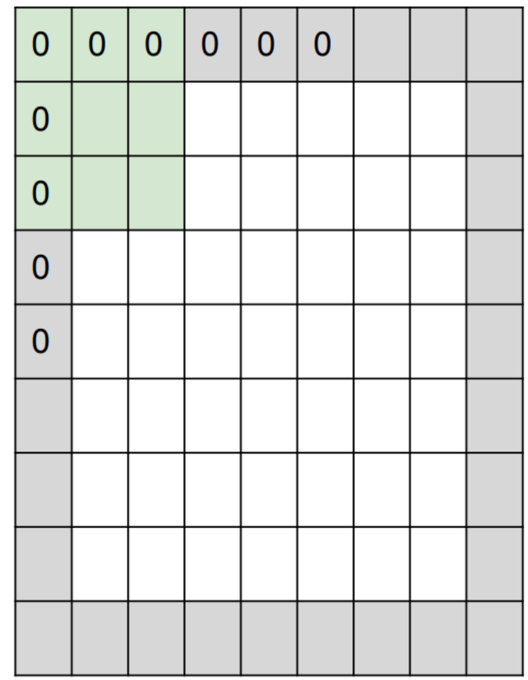
- A 7x7 input with stride 3? What is the output size?
- [(N - F) / stride] + 1



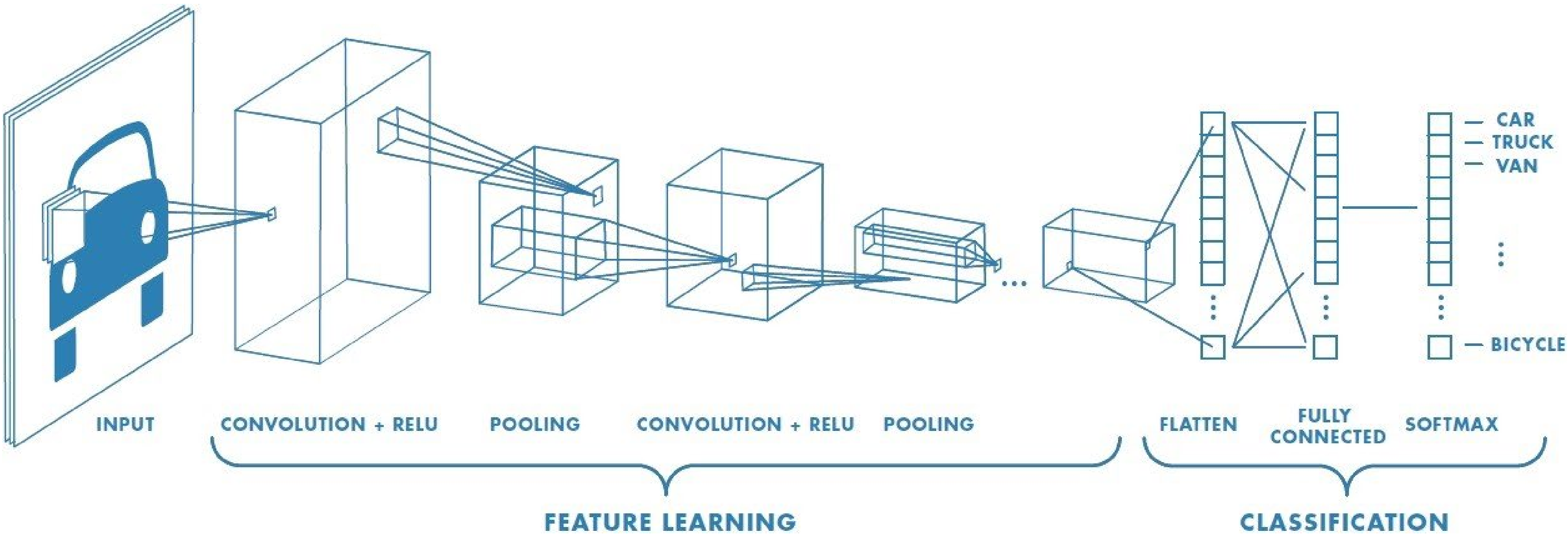
- Exercise: N = 7, F = 3:
  - stride 1 => (7 - 3)/1 + 1 = 5
  - stride 2 => (7 - 3)/2 + 1 = 3
  - stride 3 => (7 - 3)/3 + 1 = 2.33

Padding

- e.g. input 7x7 image, 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
  - zero-padding with (F-1)/2
- This will preserve size spatially



After convolution



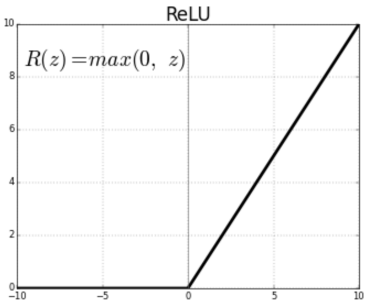
Non-linear activation

$$f(x) = \max(0, x)$$

- Tanh, sigmoid, or ReLu activation function
- Most popular (often performs best): ReLu

  => to introduce non-linearity in our ConvNet, since most of the real-world data we would want our ConvNet to learn would be non-linear.

- After each conv layer, it is convention to apply a non-linear layer (or activation layer) immediately afterwards.

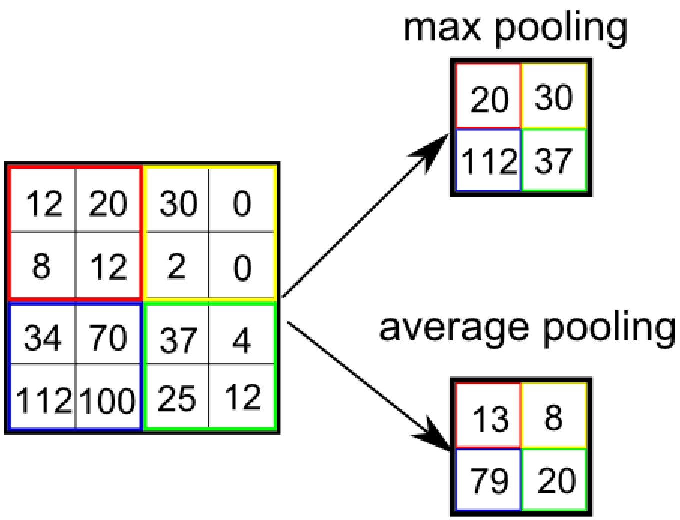


Pooling

- Non-linear down-sampling to simplify the output of the convolutional layer.
- ConvNets often use pooling layers to reduce the size of the representation, speed up the computation, as well as make some of the detected features a bit more robust.
- Types of pooling:
  - Max pooling (popular)
  - Average pooling
- Typical shape: 2x2 or sometimes 4x4
- Too large window: dramatic loss of information
- Non-overlapping windows perform the best

Pooling

- Hyperparameters:
  - Stride size
  - Pooling window size



- Pooling layers don't learn themselves, they just reduce the size of the problem

Image: https://medium.com/data-science-group-iitr/building-a-convolutional-neural-network-in-python-with-tensorflow-d251c3ca8117

Exercise: Pooling

- Hyperparameters:
  - Stride size: 2
  - Pooling window size: 2 x 2

10	3	0	18
0	3	6	4
10	12	0	8
0	2	30	2

Max Pooling

?

? x ?

Average Pooling

?

? x ?

Exercise: Pooling

- Hyperparameters:
  - Stride size: 2
  - Pooling window size: 2 x 2

10	3	0	18
0	3	6	4
10	12	0	8
0	2	30	2

Max Pooling

10	18
12	30

2 x 2

Average Pooling

?

? x ?

Exercise: Pooling

- Hyperparameters:
  - Stride size: 2
  - Pooling window size: 2 x 2

10	3	0	18
0	3	6	4
10	12	0	8
0	2	30	2

Max Pooling

10	18
12	30

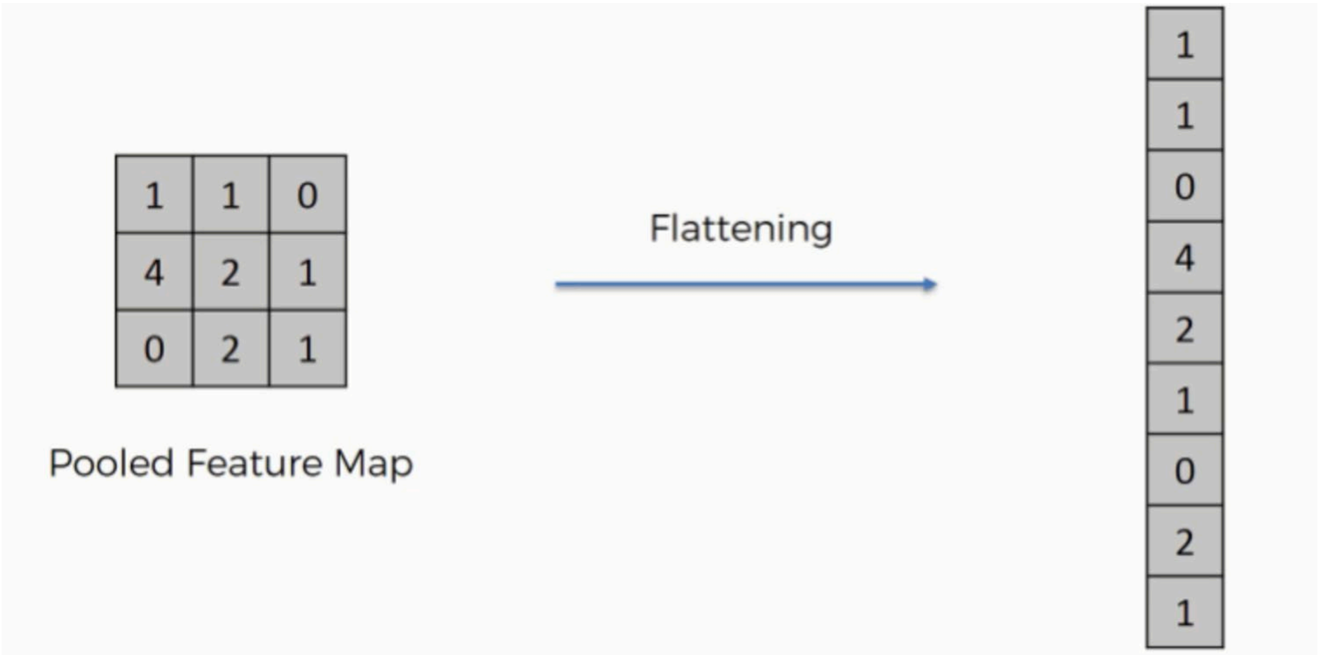
2 x 2

Average Pooling

4	7
6	10

2 x 2

Flatten

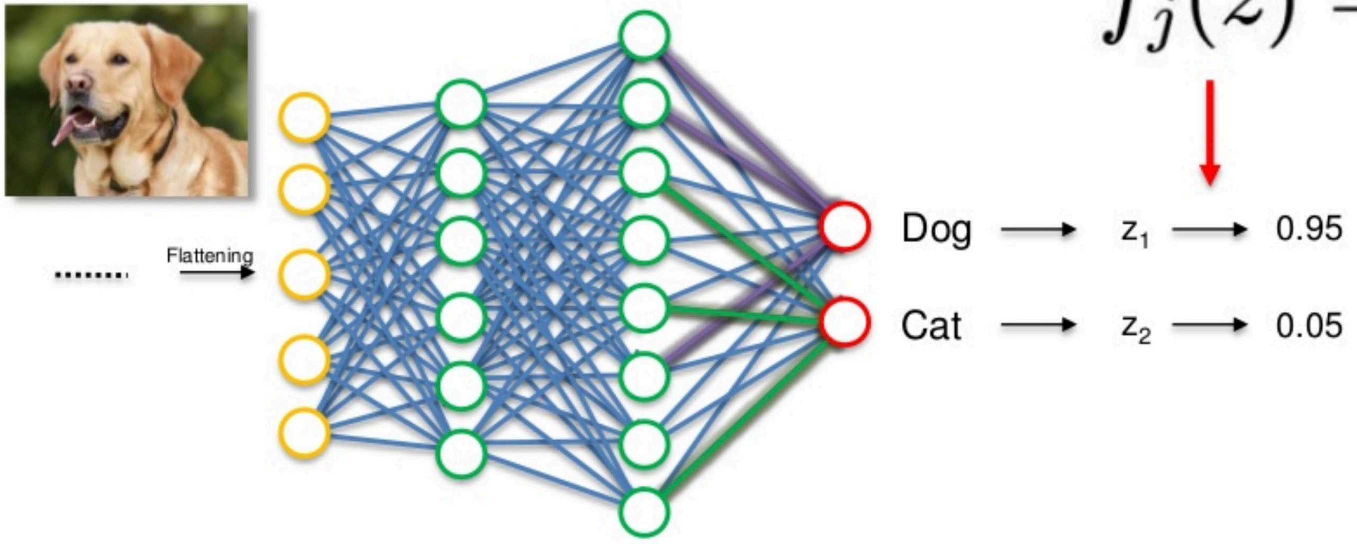


FC

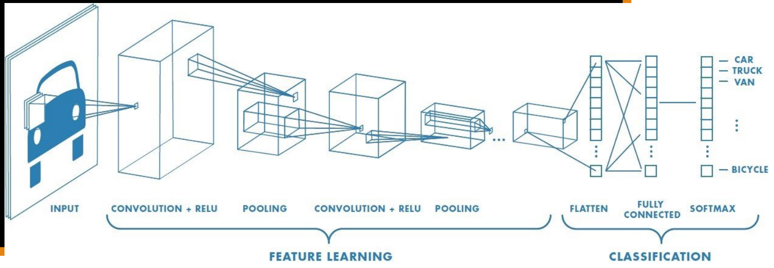
- The high-level reasoning in the neural network is done via fully connected layers. Neurons in a fully connected layer have connections to all activations in the previous layer, as seen in regular neural networks. Their activations can hence be computed with a matrix multiplication followed by a bias offset
- Training Loss: how training penalizes the deviation between the predicted and true labels and is normally the final layer. Various loss functions appropriate for different tasks may be used there:
  - Softmax loss (a Softmax activation plus a Cross-Entropy loss) is used for multiclass classification (it distributes the probability throughout each output node, meaning that the sum of all probabilities is 1)
  - Sigmoid cross-entropy loss (Sigmoid activation plus a Cross-Entropy loss) is used for binary classification
  - Euclidean loss is used for regressing to real-values. (could be mean squared error: mse)

Softmax

$$f_j(z) = \frac{e^{z_j}}{\sum_k e^{z_k}}$$



```
class myCNN(nn.Module):
    def __init__(self):
        super(myCNN, self).__init__()

        # Convolutional layers.
        self.conv1 = nn.Conv2d(1, 6, 5) #in_channels, out_channels, kernel_size
        self.conv2 = nn.Conv2d(6, 12, 5)

        # Linear layers.
        self.fc1 = nn.Linear(12*4*4, 120)

    def forward(self, x):
        # Conv1 + ReLU + MaxPooling.
        out = F.relu(self.conv1(x))
        out = F.max_pool2d(out, 2)

        # Conv2 + ReLU + MaPooling.
        out = F.relu(self.conv2(out))
        out = F.max_pool2d(out, 2)
        out = out.view(out.size(0), -1) #flatten

        # Linear layer + ReLU.
        out = F.relu(self.fc1(out))
        Return out

model = myCNN()
loss_fn = nn.CrossEntropyLoss() #includes softmax
```



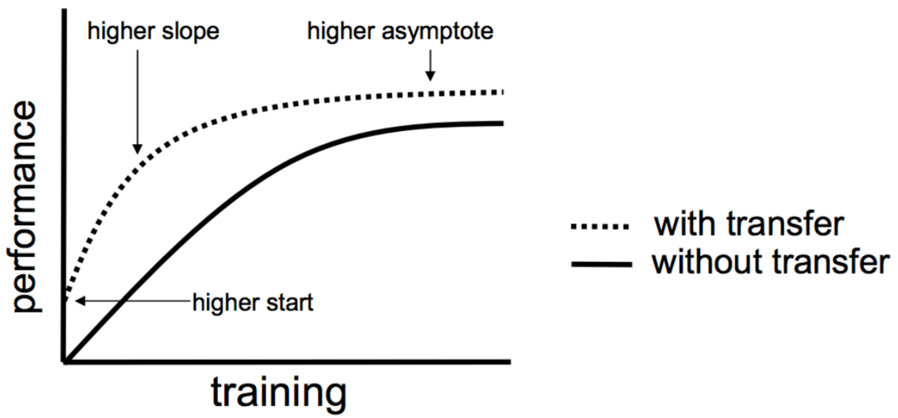
Popular CNNs

- LeNet-5: Developed 1998, Parameters 60k
- AlexNet: Developed 2012, Parameters 60M
- VGG-16: Developed 2014, Parameters 138M
- InceptionV3: Developed 2014, Parameters 23M
- ResNet50: Developed 2015, Parameters 25M
- ResNext50: Developed 2016, Parameters 25M
- DenseNet201: Developed 2017, Parameters 20M

Transfer Learning

- Transfer learning is an optimization, a shortcut to saving time or getting better performance.

- Don’t reinvent the wheel
  - Use pertained models from a larger dataset or related task and use those to represent your input
  - Then you can finetune these weights further



PyTorch finetune pertained networks

```
cnn_model = models.inception_v3(pretrained = True)

#or

cnn_model = models.resnet50(pretrained = True)

# print(cnn_model) # Print the model to see what you can modify.

# We are modifying the last layer which is stored in the fc property
# for this model as you can see by printing out the network.

cnn_model.fc = nn.Linear(2048, len(train_dataset.classes))

# print(cnn_model)  # Verify that the last linear layer was changed.
```

Similarly for text

- Remember we were not reinventing the wheel…
  - Google - word2vec
    - Google News: 3 million 300-dimension English word vectors)
  - Stanford - GLOVE
    - Wikipedia 2014 + Gigaword 5 (6B tokens, 400K vocab, uncased, 50d, 100d, 200d, & 300d vectors, 822 MB download)
  - Google – BERT
    - BooksCorpus (800M words) and English Wikipedia (2,500M words). Available on https://huggingface.co/

Credits

- Images thanks to:
  - MIT 6.S191
  - https://towardsdatascience.com/https-medium-com-piotr-skalski92-deep-dive-into-deep-networks-math-17660bc376ba
  - https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/
  - https://colah.github.io/posts/2015-08-Understanding-LSTMs/
  - https://colah.github.io/
  - https://www.deeplearningbook.org/contents/convnets.html
  - https://github.com/BlackBindy/MNIST-invert-color
  - https://www.slideshare.net/GauravMittal68/convolutional-neural-networks-cnn
  - http://slazebni.cs.illinois.edu/spring17/lec01_cnn_architectures.pdf
  - https://www.jeremyjordan.me/convnet-architectures/#resnext
  - https://www.superdatascience.com/ppt-the-ultimate-guide-to-convolutional-neural-networks-cnn/