CS60050: Machine Learning

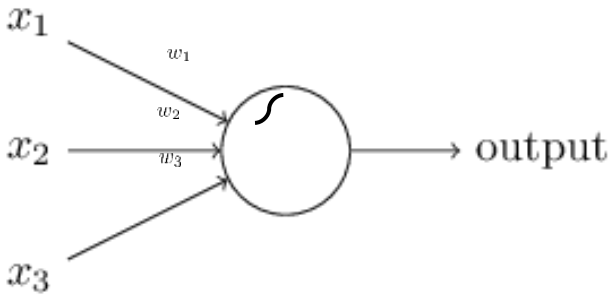
Neural Networks

Sourangshu Bhattacharya

Neural Networks

What is a Neuron?

- A neuron is a basic processing unit of a neural network.



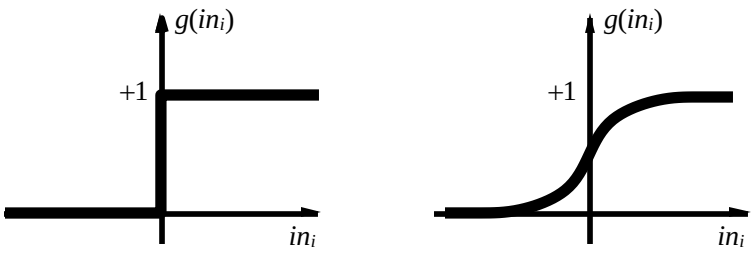
- Given several **inputs**: $x_1, x_2, x_3, \ldots \in \mathbb{R}$ and several **weights**: $w_1, w_2, w_3, \ldots \in \mathbb{R}$ and a **bias** value: $b \in \mathbb{R}$

- A neuron produces a single output:

$$o_i = s\left(\sum_i w_i x_i + b\right)$$

- This sum is called the **activation** of the neuron: $\sum_i w_i x_i + b$
- The function s is called the **activation function** for the neuron
- The weights and bias values are typically initialized randomly and learned during training

Activation functions



(a)is a step function or threshold function

(b)is a sigmoid function $1/(1 + e^{-x})$

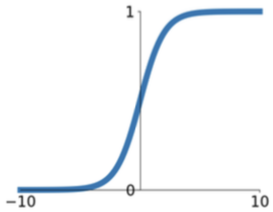
Changing the bias weight $W_{0,i}$ moves the threshold location

The purpose of the activation function is to add a non-linear transformation and in some cases squash the output to a specified range.

Commonly used Activation functions

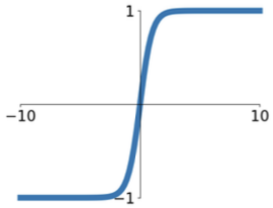
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



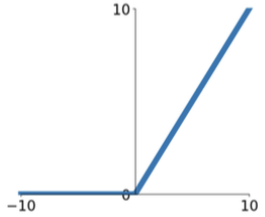
tanh

$$\tanh(x)$$



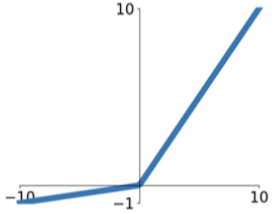
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

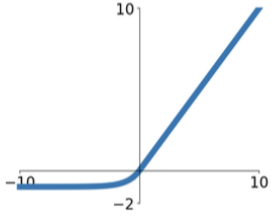


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

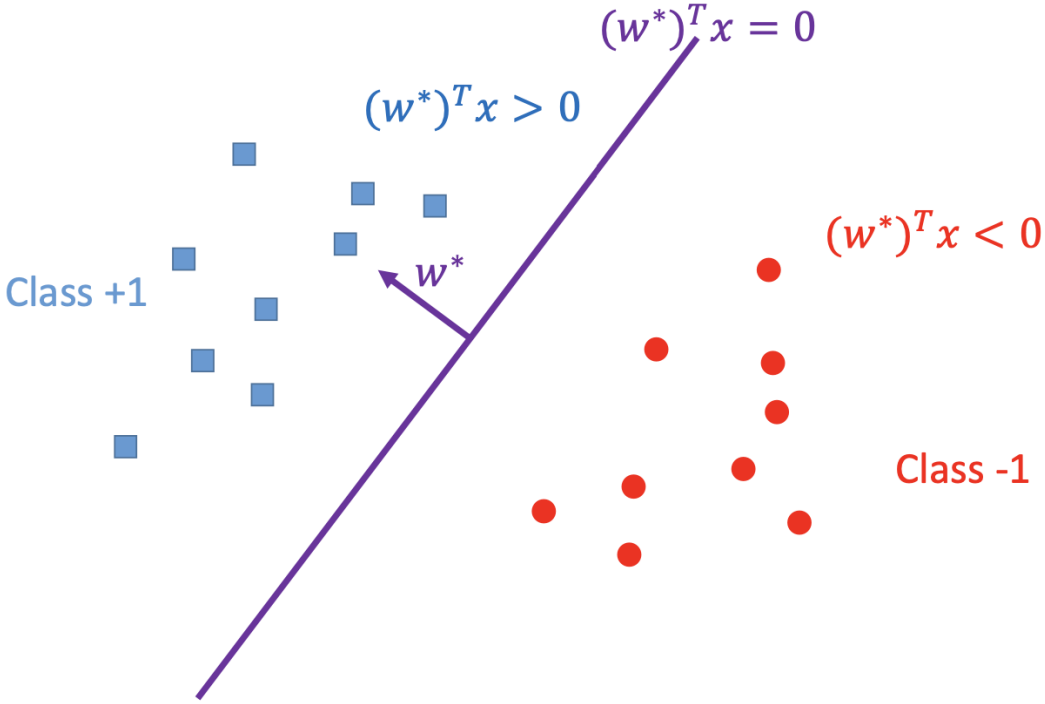
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Perceptron

Perceptron: a 1-layer NN

Task: Classification



Perceptron: a 1-layer NN

- Given training data $D = \{(x_i, y_i): 1 \leq i \leq n\}$ i.i.d. from distribution D

- Hypothesis $f_w(x) = w^T x$
  - $y = +1$ if $w^T x > 0$
  - $y = -1$ if $w^T x \leq 0$

- Prediction: $y = sign(f_w(x)) = sign(w^T x)$
  - Activation function: step function

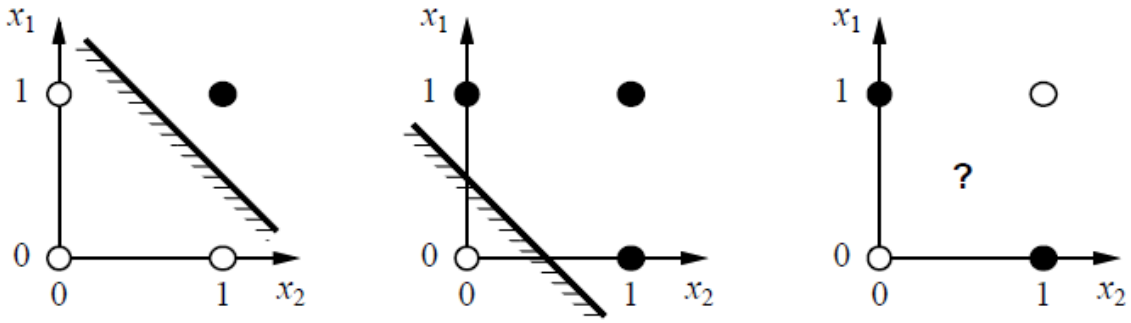
Expressiveness of Perceptrons

Consider a perceptron with g = step function (Rosenblatt, 1957, 1960)

Can represent AND, OR, NOT, majority, etc., but not XOR

Represents a linear separator in input space:

$\sum_j W_j x_j > 0$ or $\mathbf{W} \cdot \mathbf{x} > 0$



(a) x_1 **and** x_2 (b) x_1 **or** x_2 (c) x_1 **xor** x_2

Minsky & Papert (1969) pricked the neural network balloon

Multi-layer Perceptrons

Feed Forward Neural Networks

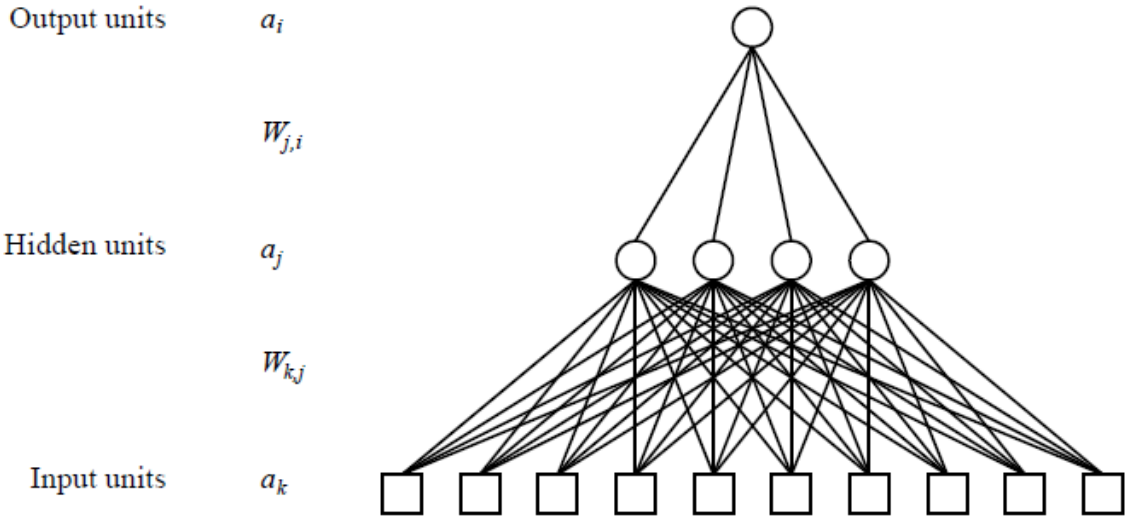
Many neurons are organized into layers.

Layer zero is input data.

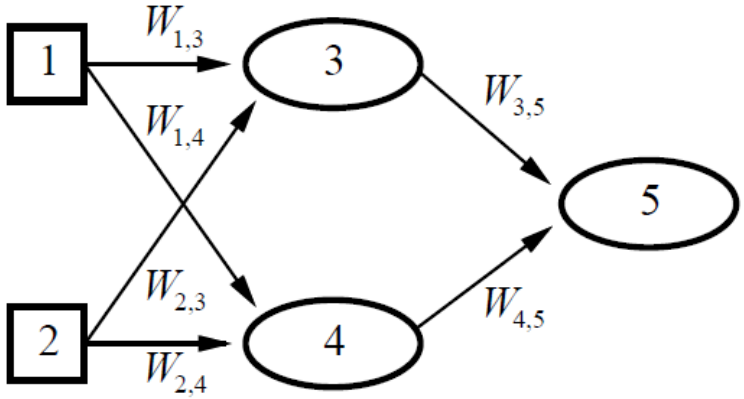
Neurons taking input from the input data are called layer one neurons.

Neurons taking input from the output of layer 1 neurons, or input data are called layer 2 neurons

Layer n neurons: Take input from Layer 0 … Layer $n-1$ neurons.



Feed forward Neural Network Computation



Feed-forward network = a parameterized family of nonlinear functions:

$$\begin{aligned} a_5 &= g(W_{3,5} \cdot a_3 + W_{4,5} \cdot a_4) \\ &= g(W_{3,5} \cdot g(W_{1,3} \cdot a_1 + W_{2,3} \cdot a_2) + W_{4,5} \cdot g(W_{1,4} \cdot a_1 + W_{2,4} \cdot a_2)) \end{aligned}$$

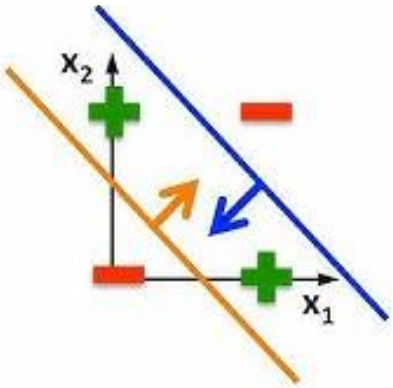
Adjusting weights changes the function: do learning this way!

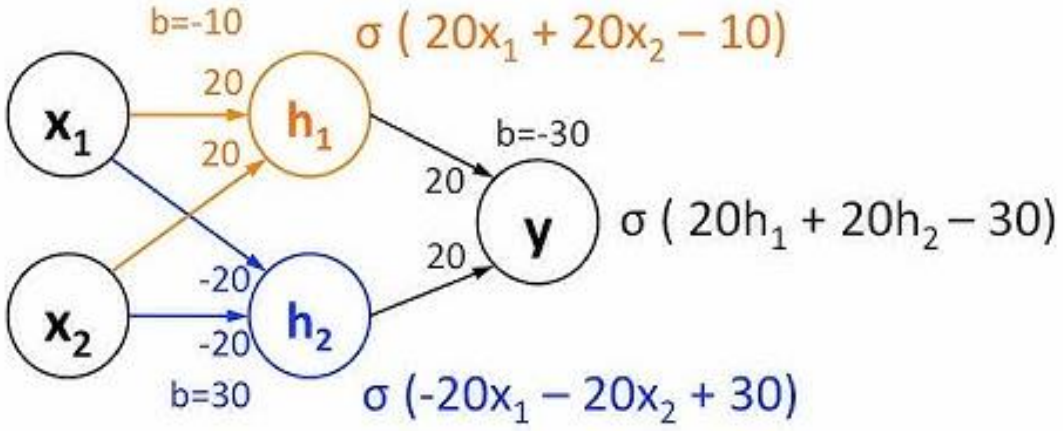
Hidden-Layer

- The hidden layer represent learned non-linear combination of input data

- For solving the XOR problem, we need a hidden layer
  - some neurons in the hidden layer will activate only for some combination of input features
  - the output layer can represent combination of the activations of the hidden neurons

Solution to the Xor problem





σ(20*0 + 20*0 – 10) ≈ 0
σ(20*1 + 20*1 – 10) ≈ 1
σ(20*0 + 20*1 – 10) ≈ 1
σ(20*1 + 20*0 – 10) ≈ 1

σ (-20*0 – 20*0 + 30) ≈ 1
σ (-20*1 – 20*1 + 30) ≈ 0
σ (-20*0 – 20*1 + 30) ≈ 1
σ (-20*1 – 20*0 + 30) ≈ 1

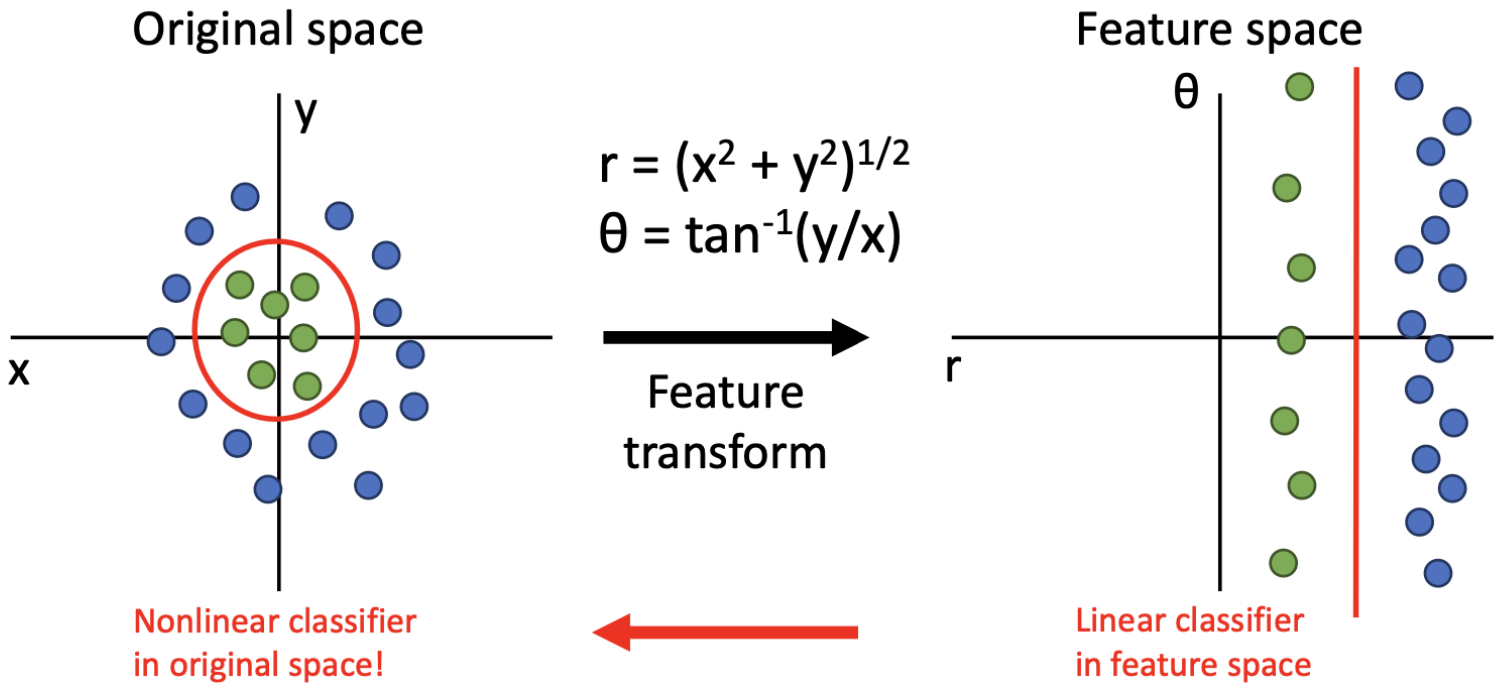
σ (20*0 + 20*1 – 30) ≈ 0
σ (20*1 + 20*0 – 30) ≈ 0
σ (20*1 + 20*1 – 30) ≈ 1
σ (20*1 + 20*1 – 30) ≈ 1

Composition of Transformations

Distributed feature representation

- Each neuron in neural network can be thought of as computing a useful representation of its inputs.
- A layer – composed of many neuron can be thought as a full representation of the input datapoint.
    - This representation is independent of the other datapoints – hence distributed representations.

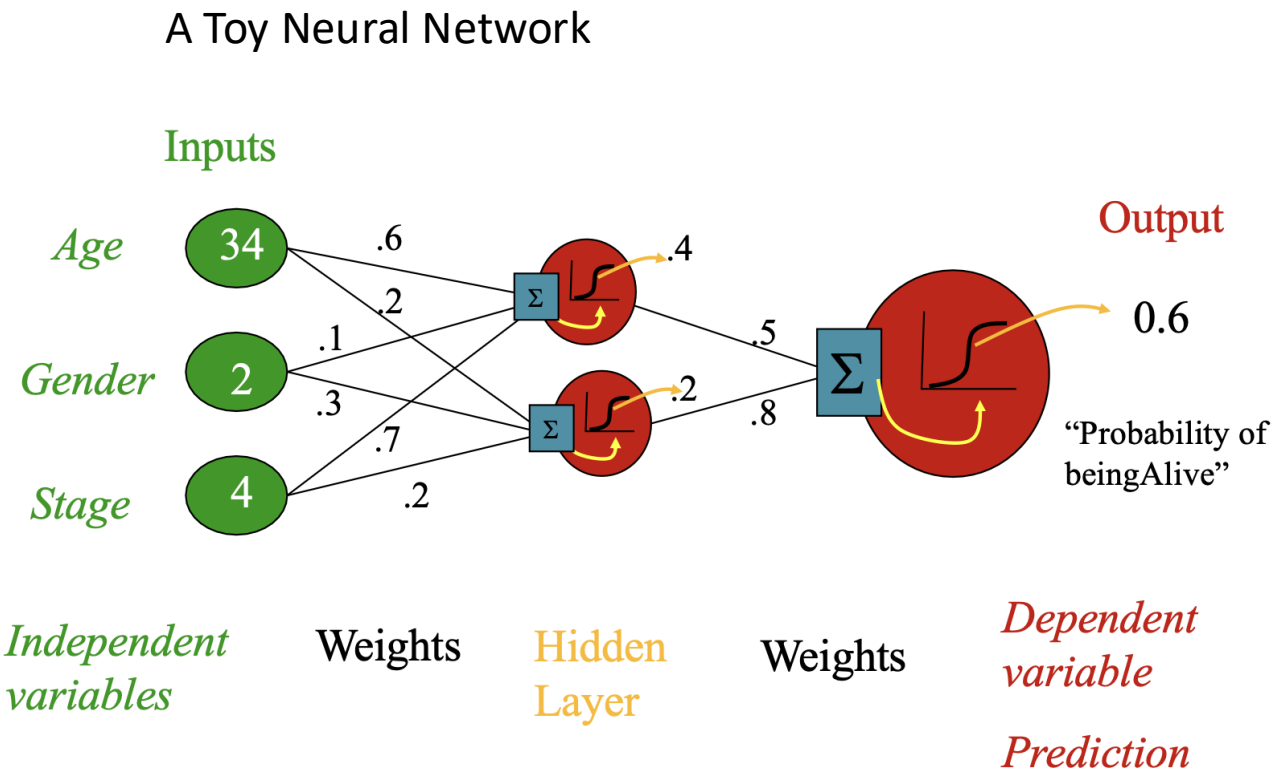
Example: A useful feature transformation



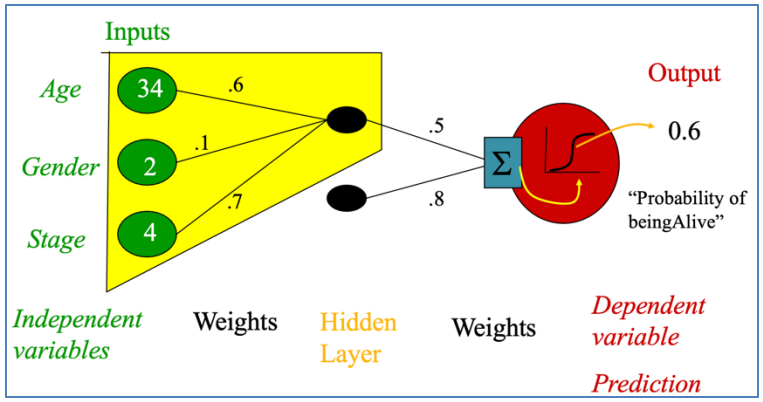
Distributed feature representation

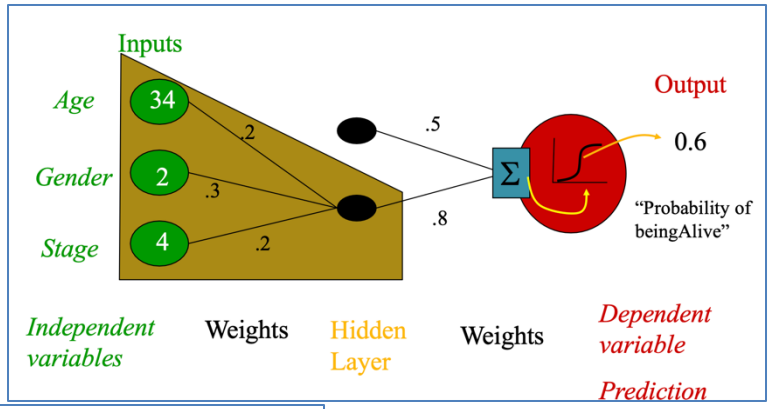
- Each neuron in neural network can be thought of as computing a useful representation of its inputs.
- A layer – composed of many neuron can be thought as a full representation of the input datapoint.
    - This representation is independent of the other datapoints – hence distributed representations.
- The representations are formed hierarchically in each layer - progressively becoming more useful for the end task.
- The neural network can be thought as a composition of such "feature encoders".

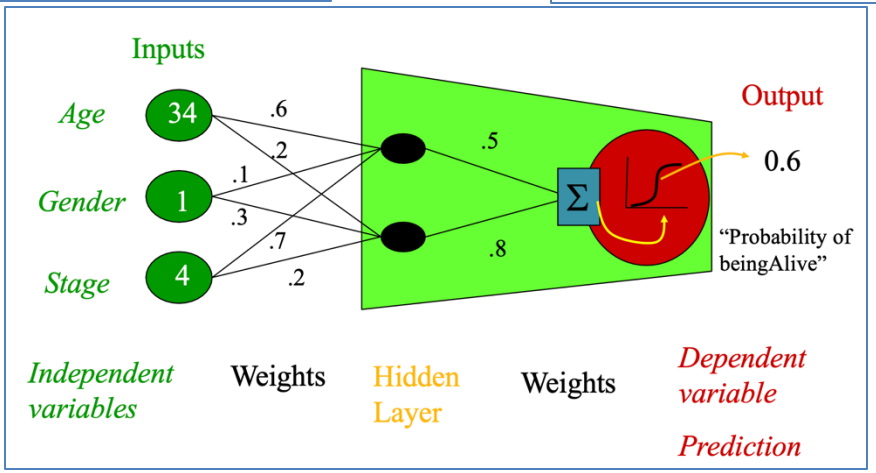
Feed forward Neural Network Example



Feed forward Neural Network Example







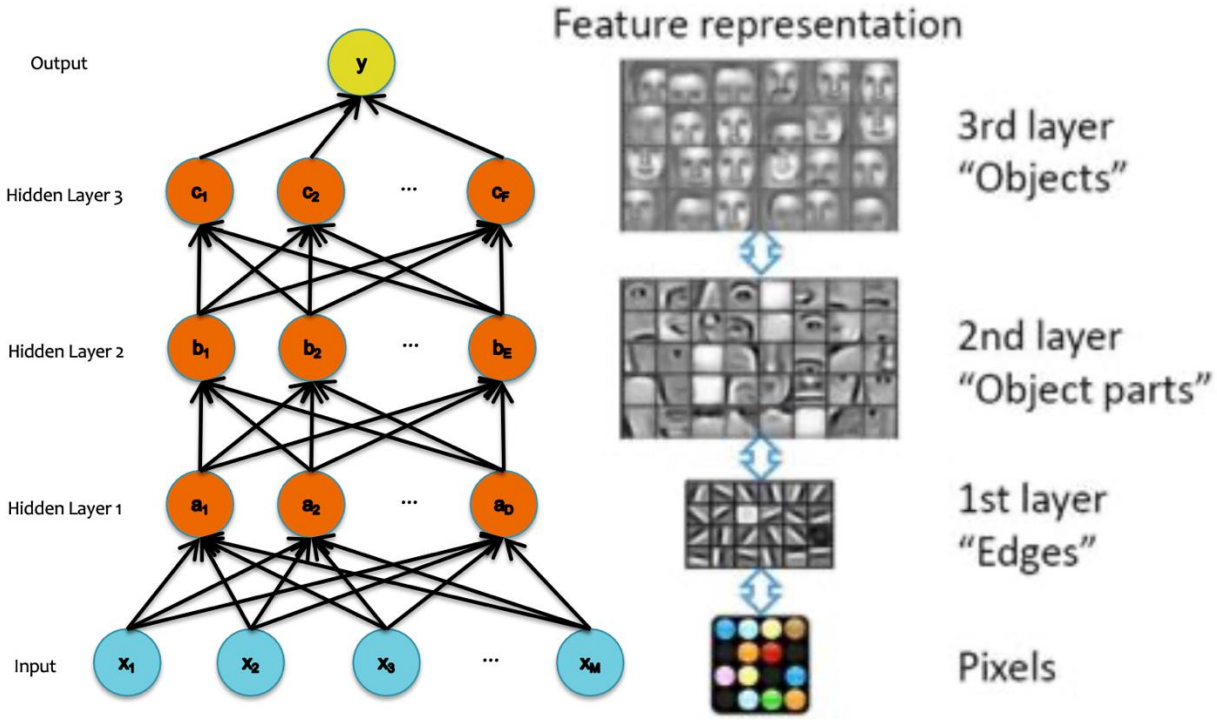
Hidden-Layer

- The hidden layer represent learned non-linear combination of input data
- For solving the XOR problem, we need a hidden layer
  - some neurons in the hidden layer will activate only for some combination of input features
  - the output layer can represent combination of the activations of the hidden neurons
- Neural network with one hidden layer **is a universal approximator**
  - Every function can be modeled as a shallow feed forward network
  - Not all functions can be represented *efficiently* with a single hidden layer
    $\Rightarrow$ we still need deep neural networks

Deep Neural Networks

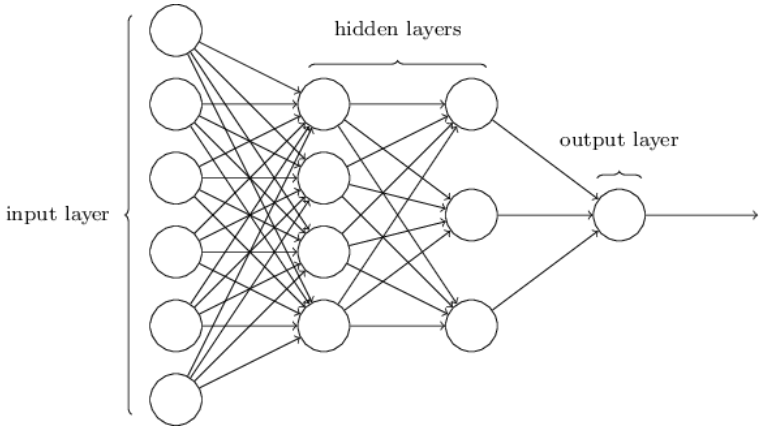
- Neural networks with more than 2 / 3 / 4 layers are considered “Deep”.
  - There is no agreement on the threshold.
- Require systematic approach to:
  - Train – Backpropagation.
  - Design – Modular approach.
  - Debug – Issues like vanishing gradient.
- Deeper neural networks have been shown to perform better than shallow ones.
  - Hierarchical buildup of useful features.

Deep Neural Networks – hierarchical features



Shallow to Deep Neural Networks

- Neural Networks can have several hidden layers
- Initializing the weights randomly and training all layers at once does hardly work
- Instead we train layerwise on unannotated data (a.k.a. pre-training):
  - Train the first hidden layer
  - Fix the parameters for the first layer and train the second layer.
  - Fix the parameters for the first & second layer, train the third layer



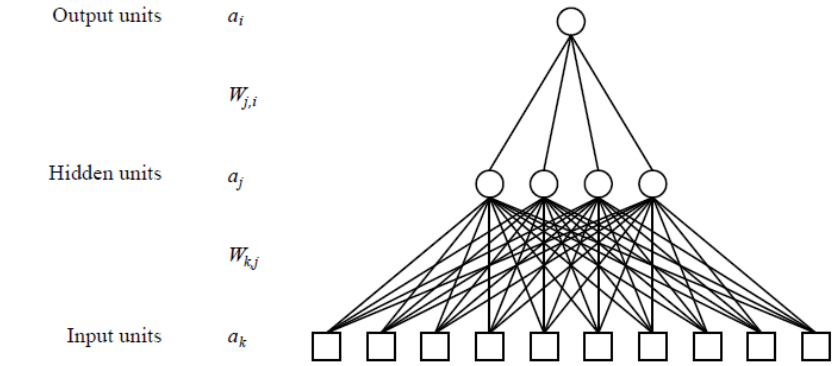
- After the pre-training, train all layers using your annotated data
- The pre-training on your unannotated data creates a high-level abstractions of the input data
- The final training with annotated data fine tunes all parameters in the network

TRAINING A NEURAL NETWORK

How to learn the weights ?

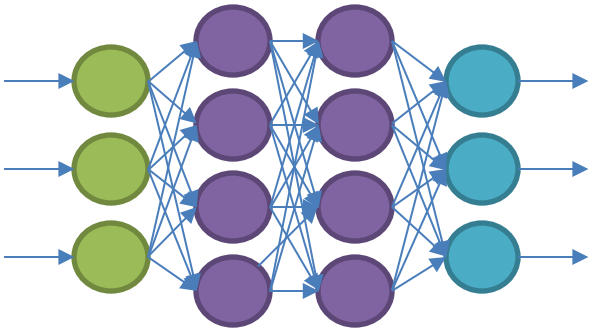
- Initialise the weights i.e. $W_{k,j}$ $W_{j,i}$ with random values
- With input entries we calculate the predicted output
- We compare the prediction with the true output
- The error is calculated
- The error needs to be sent as feedback for updating the weights

Layers are usually fully connected;
numbers of hidden units typically chosen by hand



How to Train a Neural Net?

Input
(Feature Vector)



Output
(Label)

- Put in Training inputs, get the output
- Compare output to correct answers: Look at loss function J
- Adjust and repeat!
- Backpropagation tells us how to make a single adjustment using calculus.

How to Train a Neural Net?

- Gradient Descent!

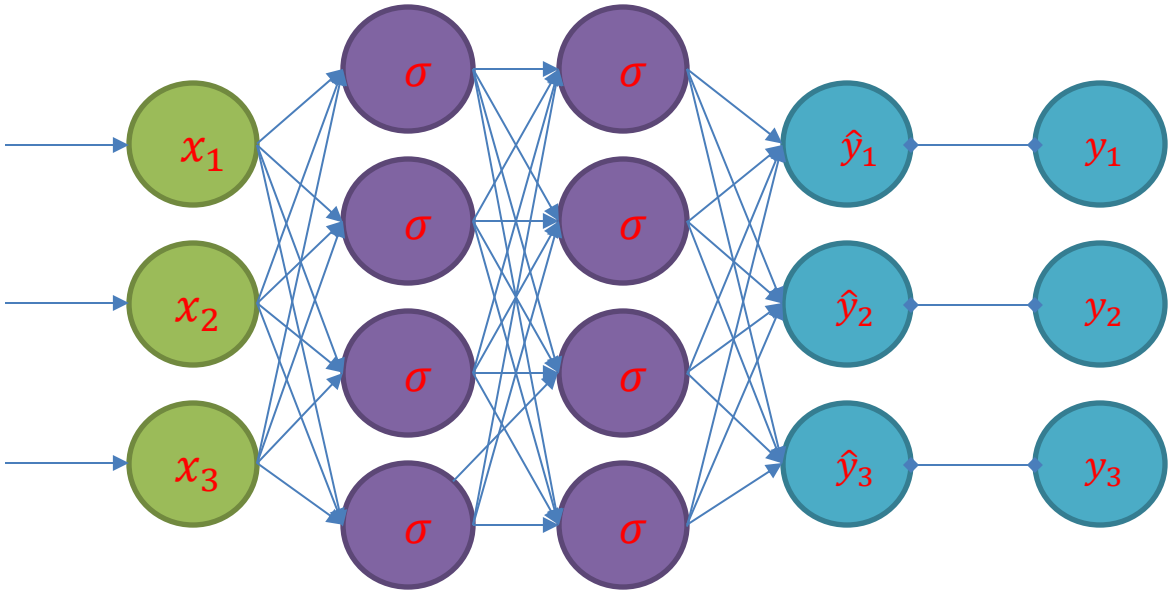
1. Make prediction
2. Calculate Loss
3. Calculate gradient of the loss function w.r.t. parameters
4. Update parameters by taking a step in the opposite direction
5. Iterate

How to Train a Neural Net?

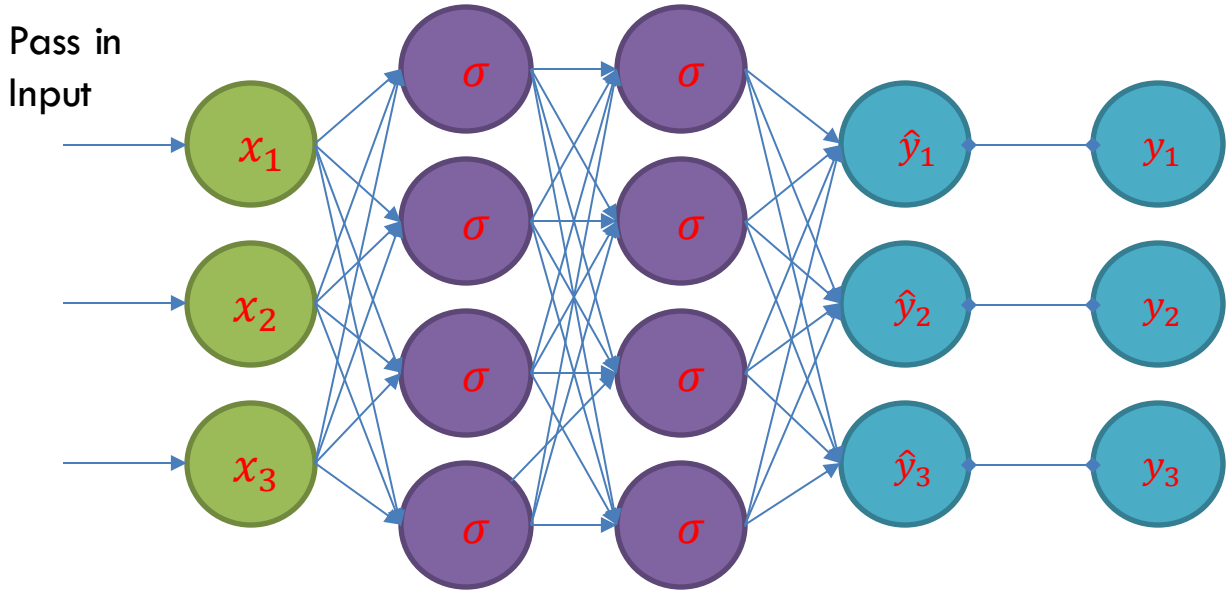
- Gradient Descent!

1. Make prediction
2. Calculate Loss
3. Calculate gradient of the loss function w.r.t. parameters
4. Update parameters by taking a step in the opposite direction
5. Iterate

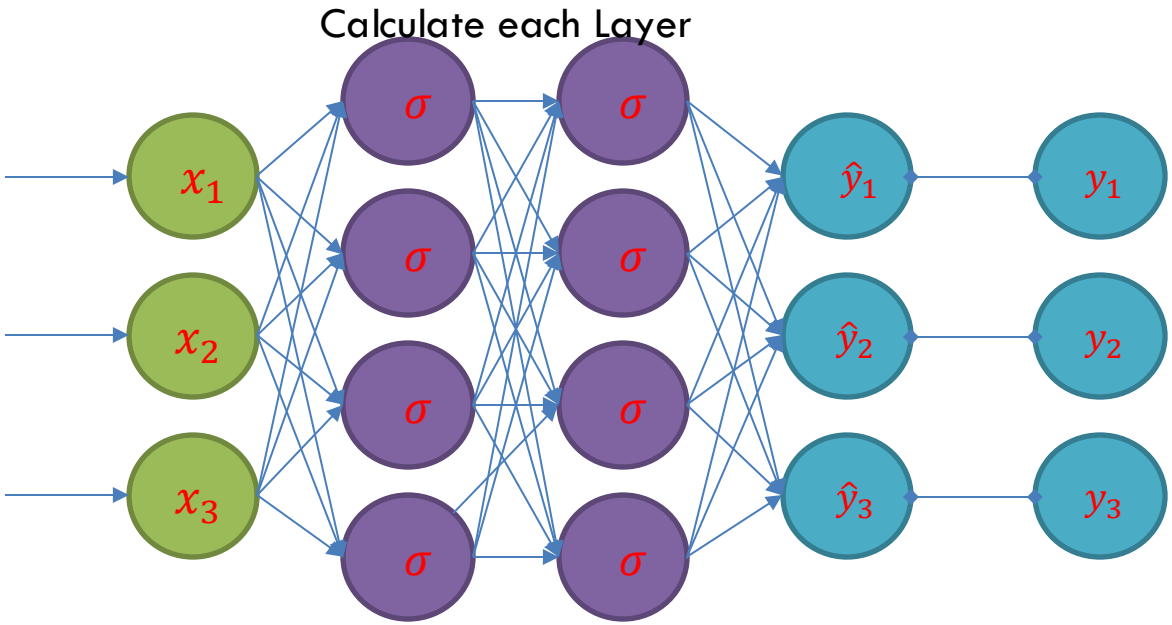
Forward Propagation



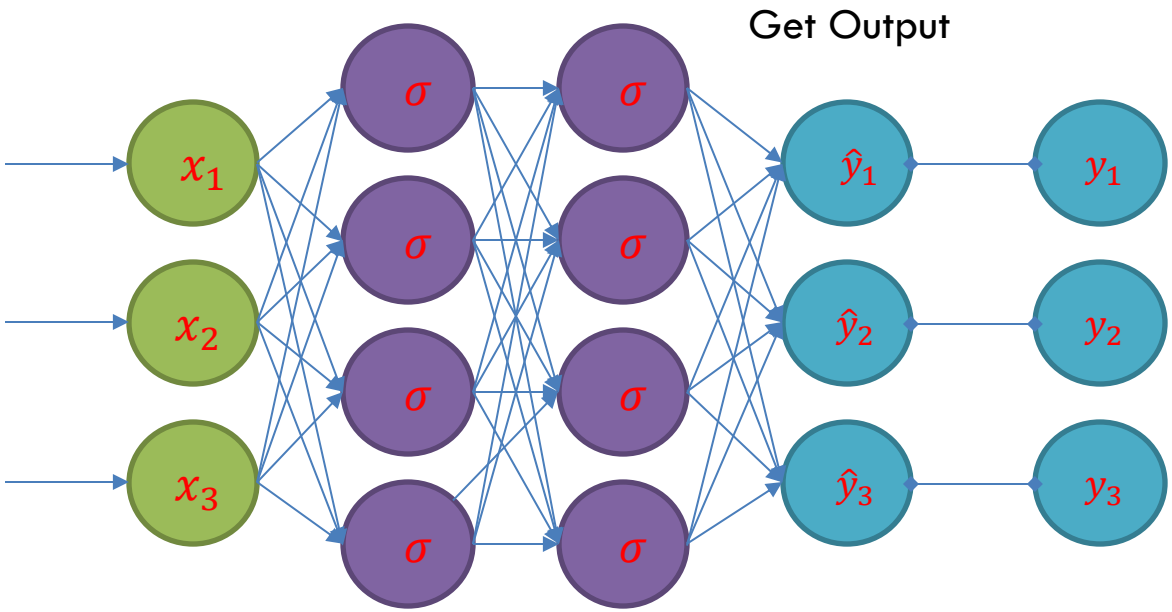
Forward Propagation



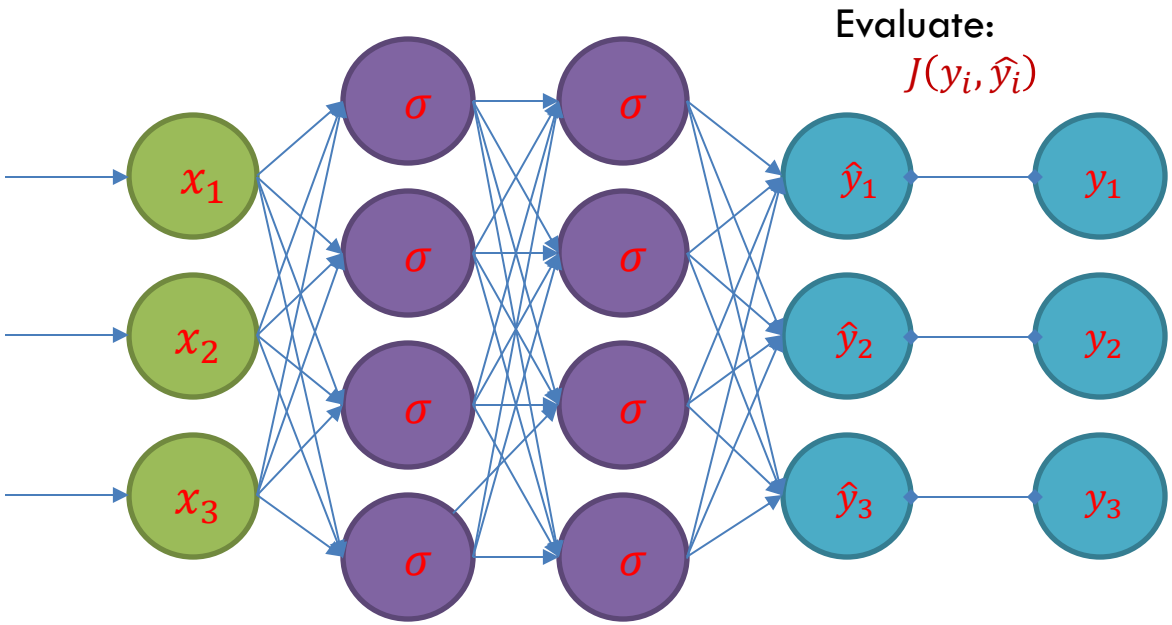
Forward Propagation



Forward Propagation



Forward Propagation



How to Train a Neural Net?

- Gradient Descent!

1. Make prediction
2. Calculate Loss
3. Calculate gradient of the loss function w.r.t. parameters
4. Update parameters by taking a step in the opposite direction
5. Iterate

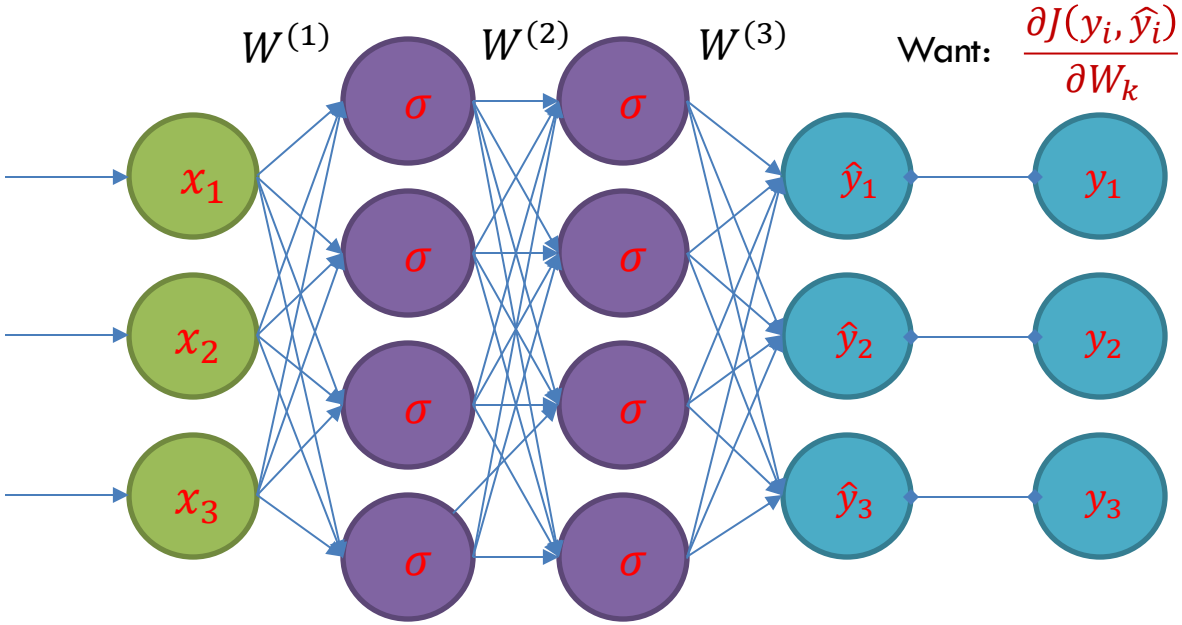
How to Train a Neural Net?

- How could we change the weights to make our Loss Function lower?
- Think of neural net as a function $F\colon\ X\ -> \ Y$
- F is a complex computation involving many weights W_k
- Given the structure, the weights "define" the function F (and therefore define our model)
- Loss Function is $J(y, F(x))$

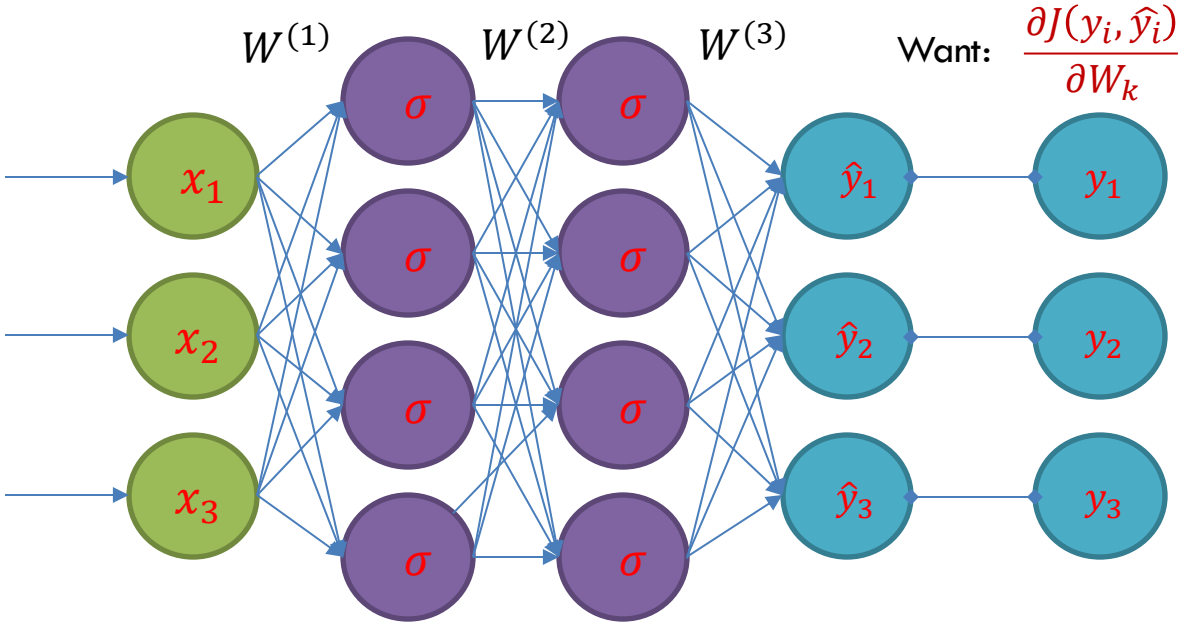
How to Train a Neural Net?

- Get $\frac{\partial J}{\partial W_k}$ for every weight in the network.
- This tells us what direction to adjust each W_k if we want to lower our loss function.
- Make an adjustment and repeat!

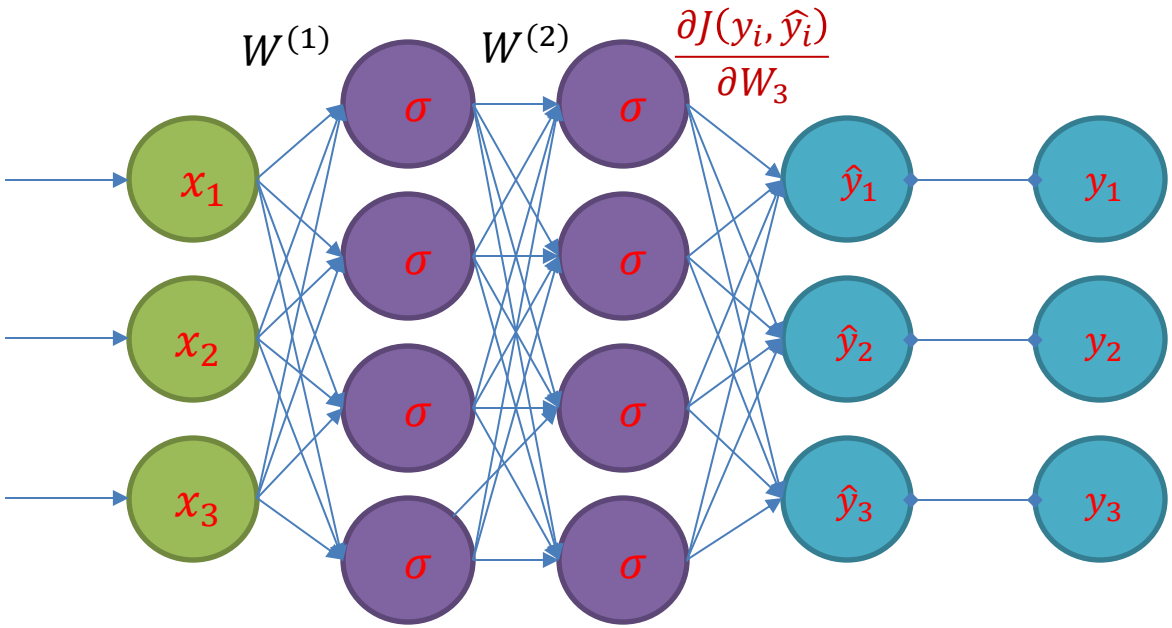
Feedforward Neural Network



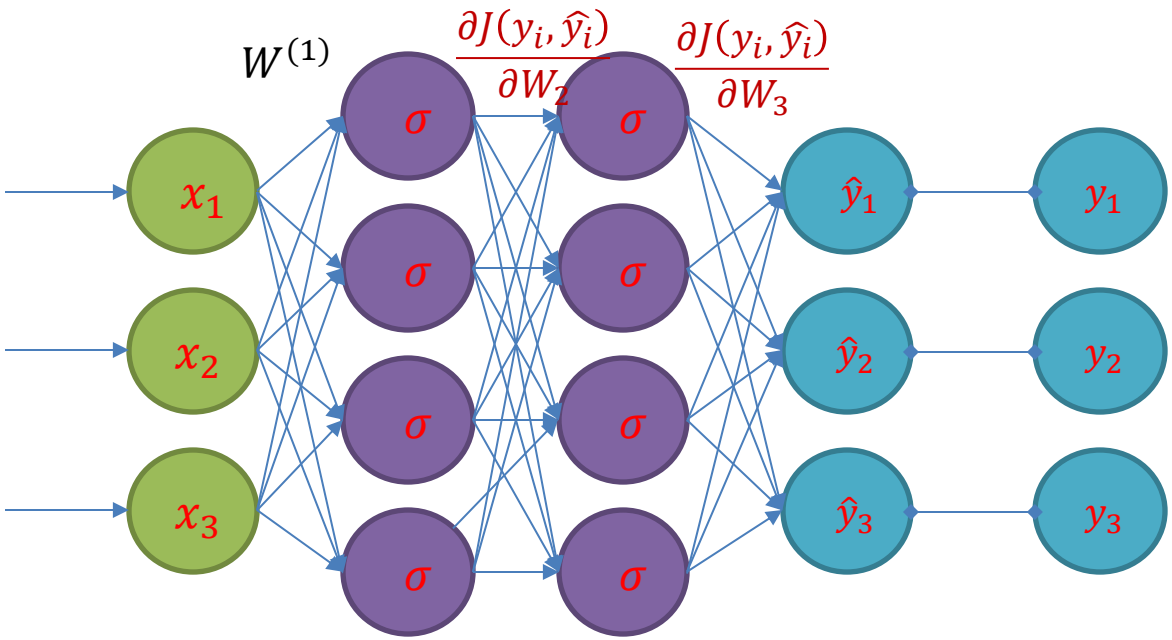
Backward Pass



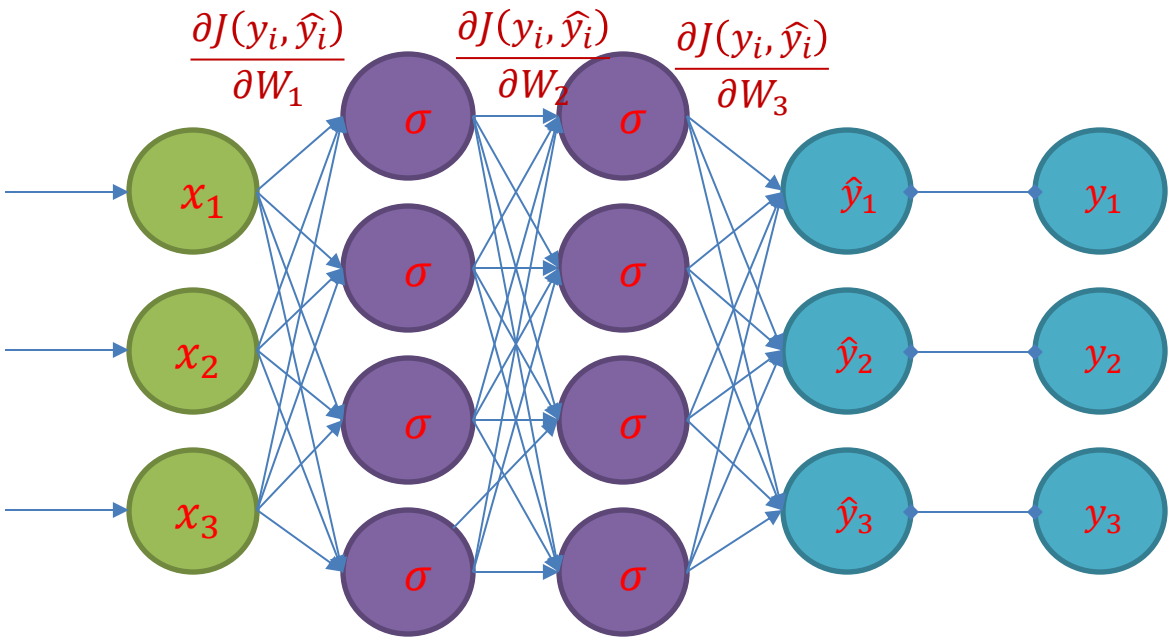
Backward Pass



Backward Pass



Backward Pass



Backpropagation Formula

$$\frac{\partial J}{\partial W^{(3)}} = (\hat{y} - y) \cdot a^{(3)}$$

$$\frac{\partial J}{\partial W^{(2)}} = (\hat{y} - y) \cdot W^{(3)} \cdot \sigma'\left(z^{(3)}\right) \cdot a^{(2)}$$

$$\frac{\partial J}{\partial W^{(1)}} = (\hat{y} - y) \cdot W^{(3)} \cdot \sigma'\left(z^{(3)}\right) \cdot W^{(2)} \cdot \sigma'\left(z^{(2)}\right) \cdot X$$

- Recall that: $\sigma'(z) = \sigma(z)(1 - \sigma(z))$
- Though they appear complex, above are easy to compute!

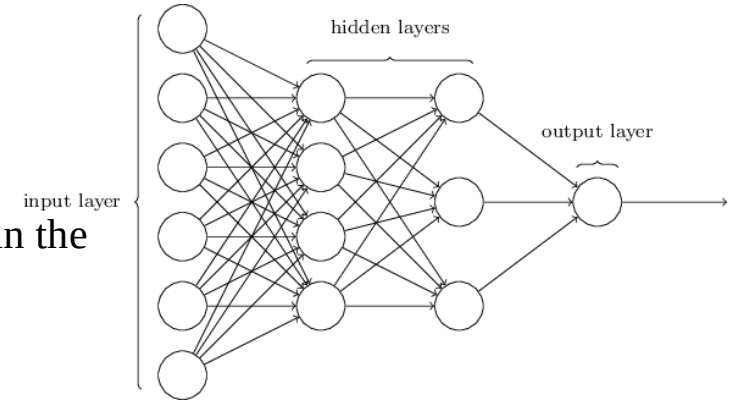
How to Train a Neural Net?

- Gradient Descent!

1. Make prediction
2. Calculate Loss
3. Calculate gradient of the loss function w.r.t. parameters
4. Update parameters by taking a step in the opposite direction
   a. $W^{t+1} = W^t - \eta_t \, {\partial J(F_w(x))}/{\partial W}$
5. Iterate

Going from Shallow to Deep Neural Networks

- Neural Networks can have several hidden layers
- Initializing the weights randomly and training all layers at once does hardly work
- Instead we train layerwise on unannotated data (a.k.a. pre-training):
  - Train the first hidden layer
  - Fix the parameters for the first layer and train the second layer.
  - Fix the parameters for the first & second layer, train the third layer



- After the pre-training, train all layers using your annotated data
- The pre-training on your unannotated data creates a high-level abstractions of the input data
- The final training with annotated data fine tunes all parameters in the network

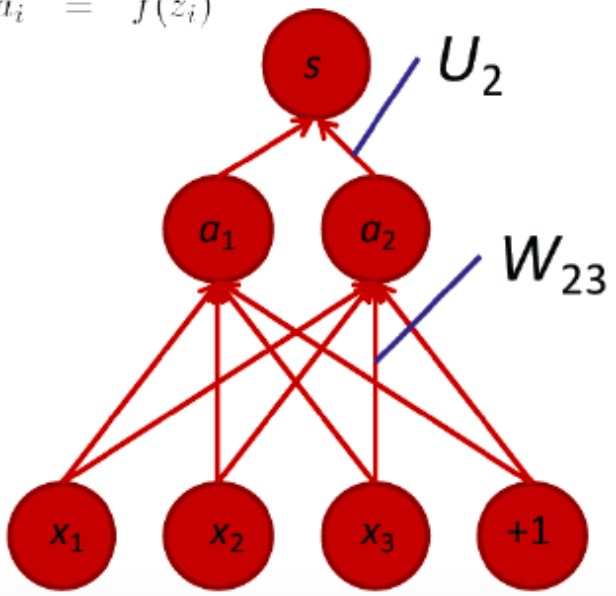
Training with backpropagation

Derivative of weight W_{ij}:

$$\frac{\partial}{\partial W_{ij}} U^T a \quad \rightarrow \quad \frac{\partial}{\partial W_{ij}} U_i a_i$$

$$\begin{aligned} U_i \frac{\partial}{\partial W_{ij}} a_i &= U_i \frac{\partial a_i}{\partial z_i} \frac{\partial z_i}{\partial W_{ij}} \\ &= U_i \frac{\partial f(z_i)}{\partial z_i} \frac{\partial z_i}{\partial W_{ij}} \\ &= U_i f'(z_i) \frac{\partial z_i}{\partial W_{ij}} \\ &= U_i f'(z_i) \frac{\partial W_i . x + b_i}{\partial W_{ij}} \end{aligned}$$

$$\begin{aligned} z_i &= W_i . x + b_i = \sum_{j=1}^{3} W_{ij} x_j + b_i \\ a_i &= f(z_i) \end{aligned}$$



Derivative continued ...

$$
\begin{aligned}
U_i \frac{\partial}{\partial W_{ij}} a_i &= U_i f'(z_i) \frac{\partial W_i.x + b_i}{\partial W_{ij}} \\
&= U_i f'(z_i) \frac{\partial}{\partial W_{ij}} \sum_k W_{ik} x_k \\
&= \underbrace{U_i f'(z_i)}\, x_j \\
&= \delta_i \quad x_j
\end{aligned}
$$

Local error signal

Local input signal

where $f'(z) = f(z)(1 - f(z))$ for logistic f

From single weight W_{ij} to full W:

$$\frac{\partial s}{\partial W_{ij}} = \delta_i x_j$$

- We want all combinations of $i = 1, 2, \ldots$ and $j = 1, 2, 3, \ldots$
- Solution: Outer product

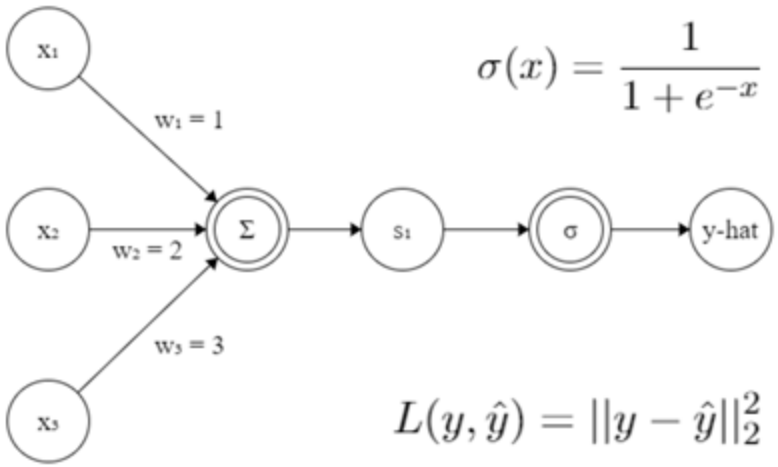
$$\frac{\partial J}{\partial W} = \delta x^T$$

Computational Graph

Definition: a data structure for storing gradients of variables used in computations.

- Node *v* represents variable
  - Stores value
  - Gradient
  - The function that created the node
- Directed edge *(u,v)* represents the partial derivative of *u* w.r.t. *v*
- To compute the gradient *dL/dv*, find the unique path from *L* to *v* and multiply the edge weights.

Backpropagation for neural nets



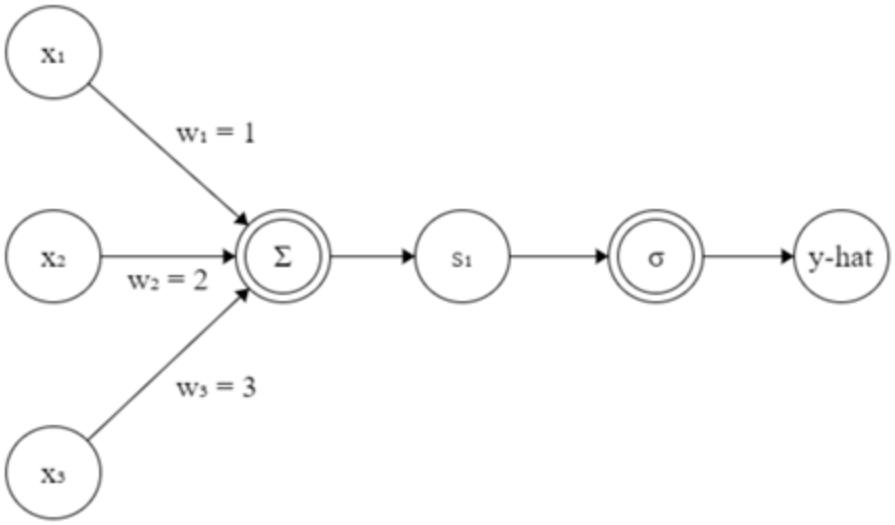
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

$$L(y, \hat{y}) = ||y - \hat{y}||_2^2$$

x	σ(x)	σ'(x)
-5	0.01	0.01
-4	0.02	0.02
-3	0.05	0.05
-2	0.12	0.10
-1	0.27	0.20
0	0.50	0.25
1	0.73	0.20
2	0.88	0.10
3	0.95	0.05
4	0.98	0.02
5	0.99	0.01

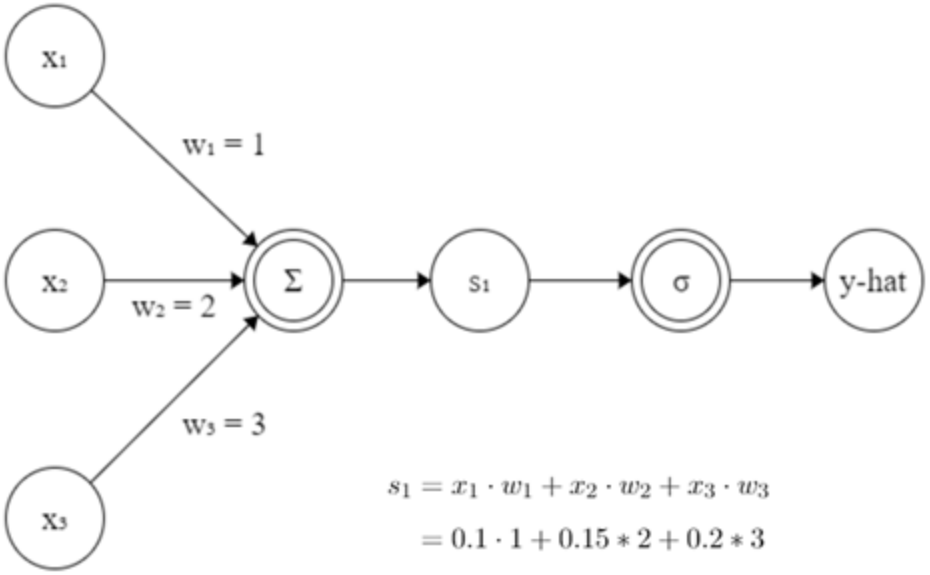
Given softmax activation, L2 loss, a point (x1, x2, x3, y) = (0. 1, 0.15, 0.2, 1), compute the gradient $\frac{\partial L}{\partial w_1}$

Backpropagation for neural nets: forward pass



$$
\begin{aligned}
s_1 &= x_1 \cdot w_1 + x_2 \cdot w_2 + x_3 \cdot w_3 \\
&= 0.1 \cdot 1 + 0.15 * 2 + 0.2 * 3 \\
&= 1 \\
\hat{y} &= \sigma(s_1) \\
&= \sigma(1) \\
&= \boxed{0.73}
\end{aligned}
$$

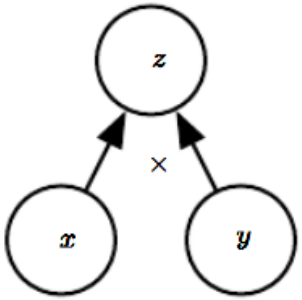
Backpropagation for neural nets: backward pass



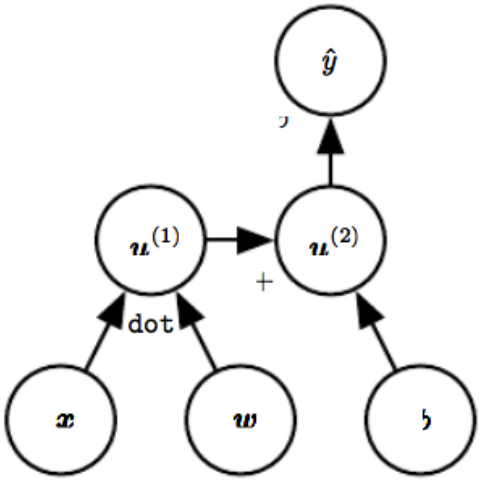
$$
\begin{aligned}
s_1 &= x_1 \cdot w_1 + x_2 \cdot w_2 + x_3 \cdot w_3 \\
&= 0.1 \cdot 1 + 0.15 * 2 + 0.2 * 3 \\
&= 1 \\
\hat{y} &= \sigma(s_1) \\
&= \sigma(1) \\
&= \boxed{0.73}
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial L}{\partial w_1} &= \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial s_1} \cdot \frac{\partial s_1}{\partial w_1} \\
&= 2||\hat{y} - y|| \times \sigma'(s_1) \times x_1 \\
&= 2 \cdot (\sigma(1) - 1) \times \sigma'(1) \times 0.1 \\
&= \boxed{-0.0106}
\end{aligned}
$$

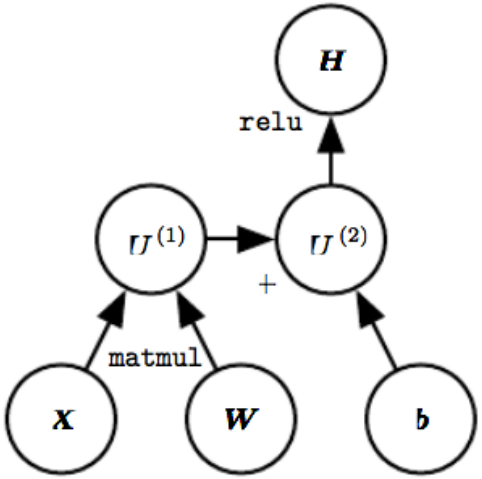
Computation Graphs

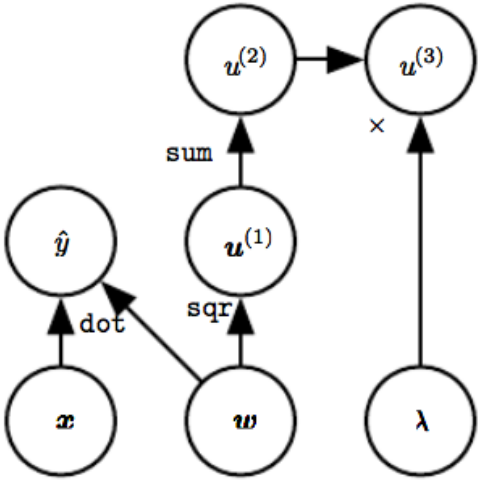


(a)



(b)





CONVOLUTIONAL NEURAL NETWORKS

Motivation – Image Data

- So far, the structure of our neural network treats all inputs interchangeably.
- No relationships between the individual inputs
- Just an ordered set of variables

- We want to incorporate domain knowledge into the architecture of a Neural Network.

Motivation

- Image data has important structures, such as;

- ”Topology” of pixels
- Translation invariance
- Issues of lighting and contrast
- Knowledge of human visual system
- Nearby pixels tend to have similar values
- Edges and shapes
- Scale Invariance – objects may appear at different sizes in the image.

Motivation – Image Data

- Fully connected would require a vast number of parameters
- MNIST images are small (32 x 32 pixels) and in grayscale
- Color images are more typically at least (200 x 200) pixels x 3 color channels (RGB) = 120,000 values.
- A single fully connected layer would require $(200x200x3)^2$ = 14,400,000,000 weights!
- Variance (in terms of bias-variance) would be too high
- So we introduce “bias” by structuring the network to look for certain kinds of patterns

Motivation

- Features need to be “built up”
- Edges -> shapes -> relations between shapes
- Textures

- Cat = two eyes in certain relation to one another + cat fur texture.
- Eyes = dark circle (pupil) inside another circle.
- Circle = particular combination of edge detectors.
- Fur = edges in certain pattern.

Kernels

- A *kernel* is a grid of weights "overlaid" on image, centered on one pixel
- Each weight multiplied with pixel underneath it
- Output over the centered pixel is $\sum_{p=1}^{P} W_p \cdot pixel_p$
- Used for traditional image processing techniques:
  - Blur
  - Sharpen
  - Edge detection
  - Emboss

Kernel: 3x3 Example

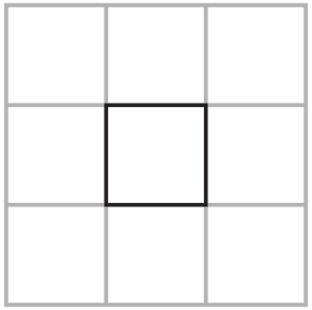
Input

3	2	1
1	2	3
1	1	1

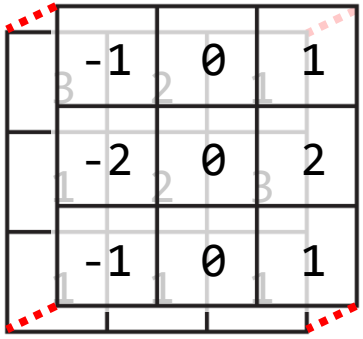
Kernel

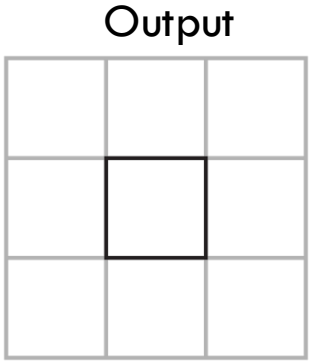
-1	0	1
-2	0	2
-1	0	1

Output



Kernel: 3x3 Example





Kernel: 3x3 Example

Input

3	2	1
1	2	3
1	1	1

Kernel

-1	0	1
-2	0	2
-1	0	1

Output

	2	

$$= (3 \cdot -1) + (2 \cdot 0) + (1 \cdot 1)$$
$$+(1 \cdot -2) + (2 \cdot 0) + (3 \cdot 2)$$
$$+(1 \cdot -1) + (1 \cdot 0) + (1 \cdot 1)$$

$$= -3 + 1 - 2 + 6 - 1 + 1 = 2$$

Kernel: Example

1	1	1
1	1	1
1	1	1

Unweighted 3x3 smoothing kernel

0	1	0
1	4	1
0	1	0

Weighted 3x3 smoothing kernel with Gaussian blur

0	-1	0
-1	5	-1
0	-1	0

Kernel to make image sharper

-1	-1	-1
-1	9	-1
-1	-1	-1

Intensified sharper image

Gaussian Blur

Sharpened image

Kernels as Feature Detectors

Can think of kernels as a ”local feature detectors”

Vertical Line Detector

-1	1	-1
-1	1	-1
-1	1	-1

Horizontal Line Detector

-1	-1	-1
1	1	1
-1	-1	-1

Corner Detector

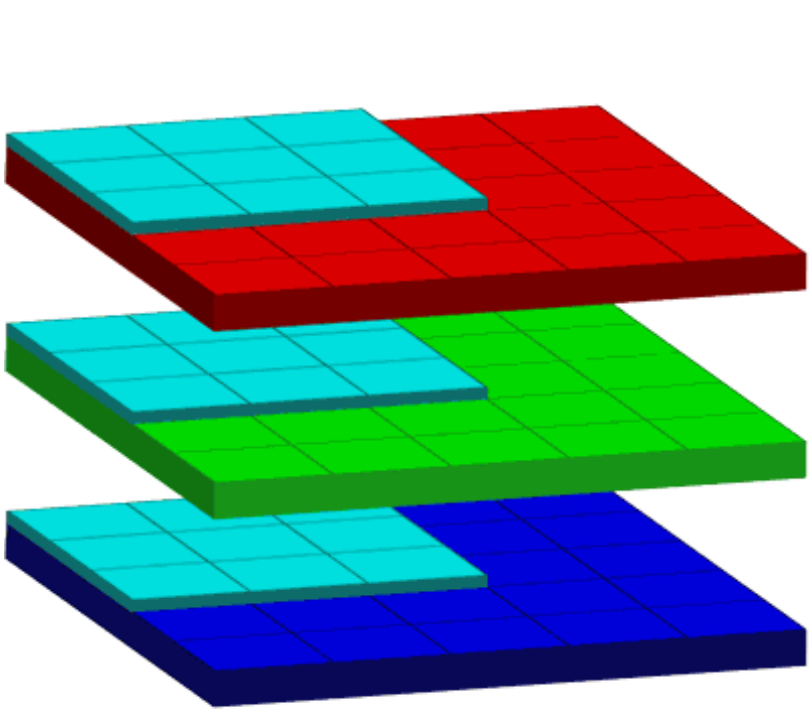
-1	-1	-1
-1	1	1
-1	1	1

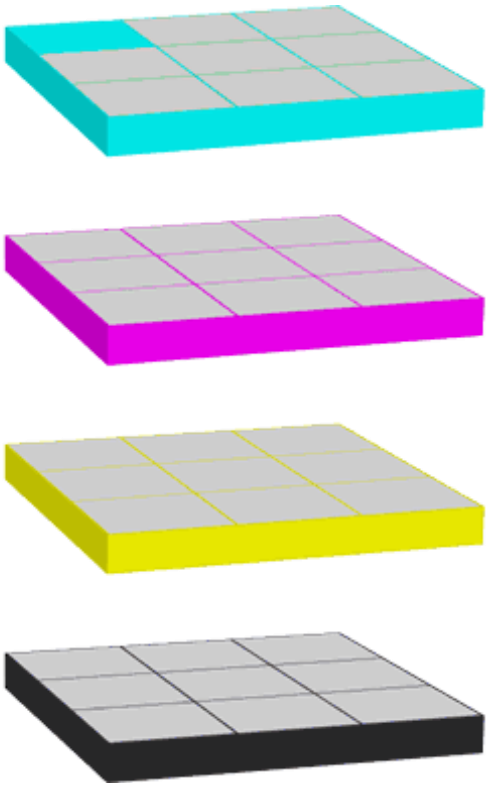
Convolutional Neural Nets

Primary Ideas behind Convolutional Neural Networks:

- Let the Neural Network learn which kernels are most useful
- Use same set of kernels across entire image (translation invariance)
- Reduces number of parameters and "variance" (from bias-variance point of view)

Convolutions



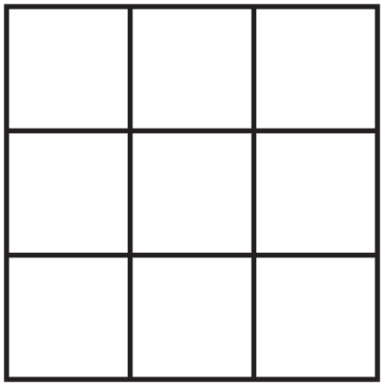


Convolution Settings – Grid Size

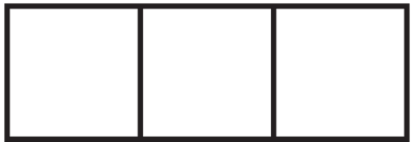
Grid Size (Height and Width):

- The number of pixels a kernel "sees" at once
- Typically use odd numbers so that there is a "center" pixel
- Kernel does not need to be square

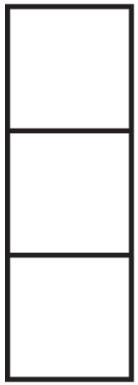
Height: 3, Width: 3



Height: 1, Width: 3



Height: 3, Width: 1

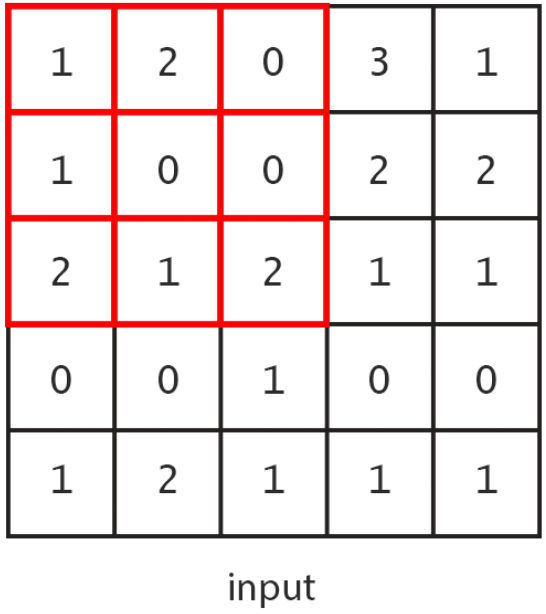


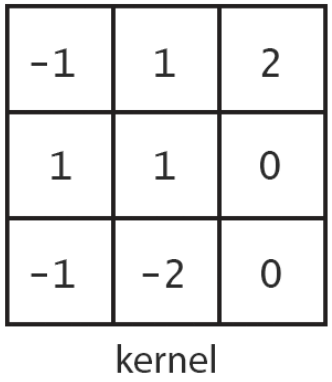
Convolution Settings - Padding

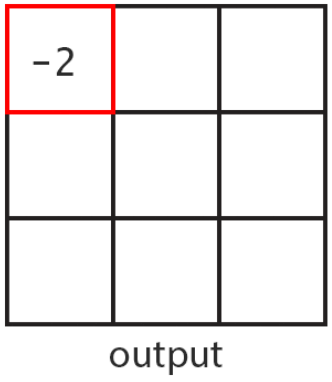
Padding

- Using Kernels directly, there will be an “edge effect”
- Pixels near the edge will not be used as “center pixels” since there are not enough surrounding pixels
- Padding adds extra pixels around the frame
- So every pixel of the original image will be a center pixel as the kernel moves across the image
- Added pixels are typically of value zero (zero-padding)

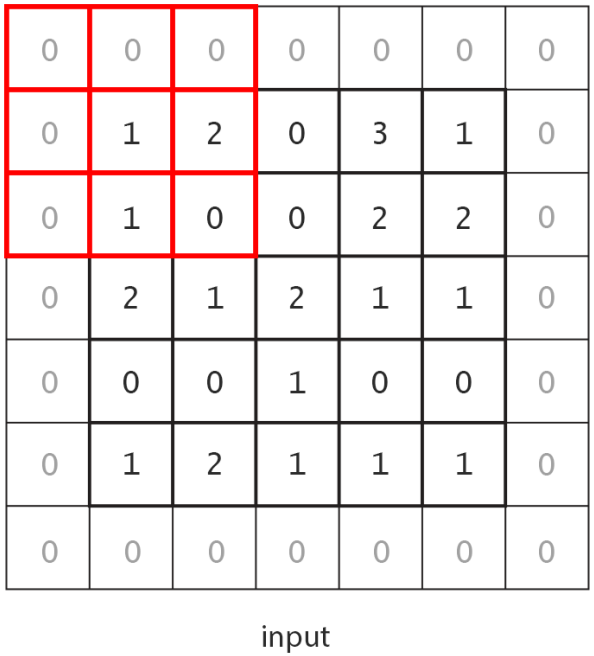
Without Padding

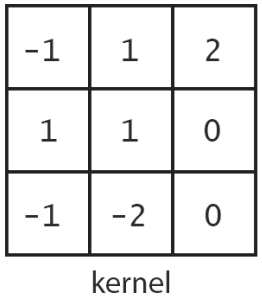


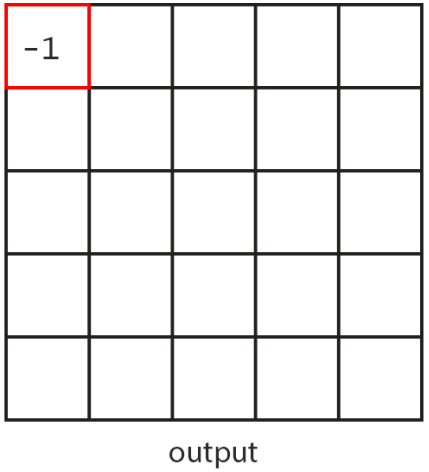




With Padding





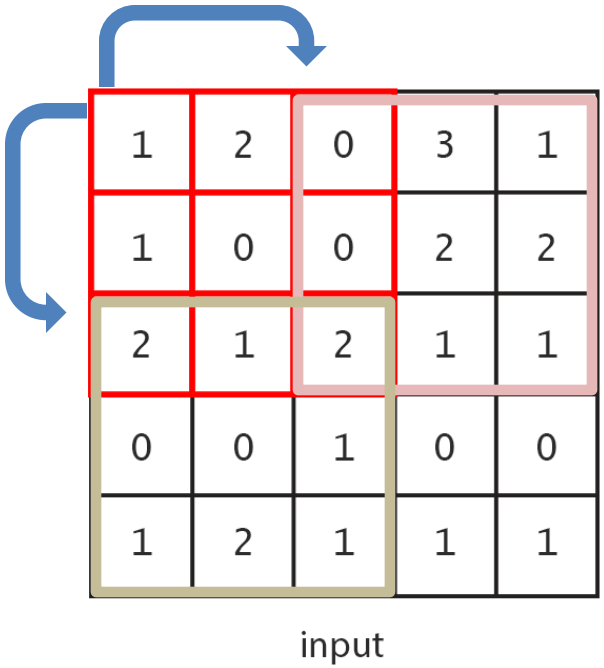


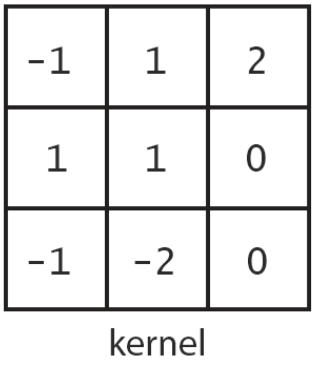
Convolution Settings

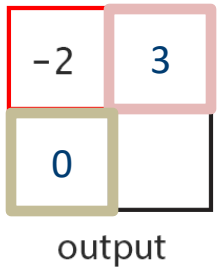
Stride

- The ”step size” as the kernel moves across the image
- Can be different for vertical and horizontal steps (but usually is the same value)
- When stride is greater than 1, it scales down the output dimension

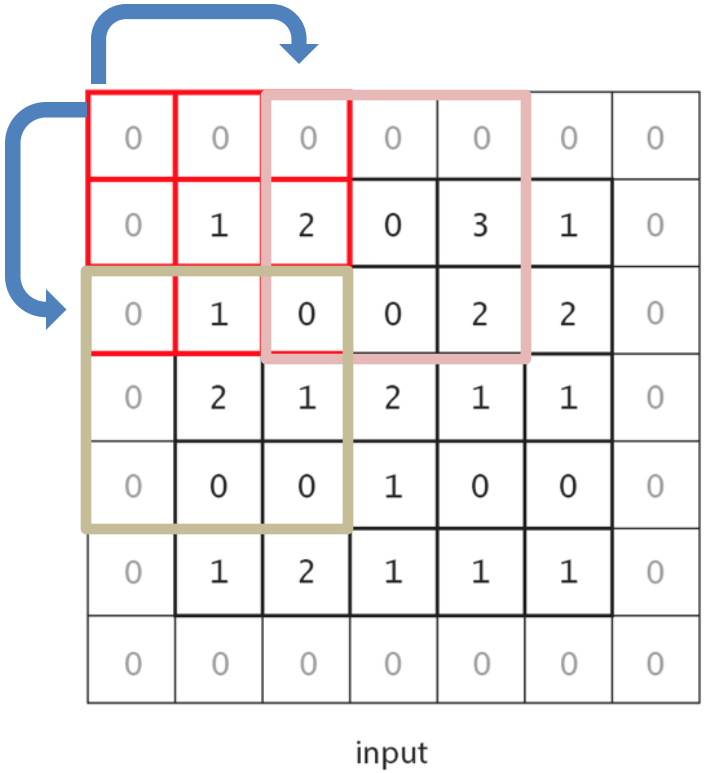
Stride 2 Example – No Padding

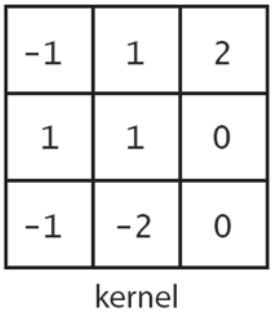


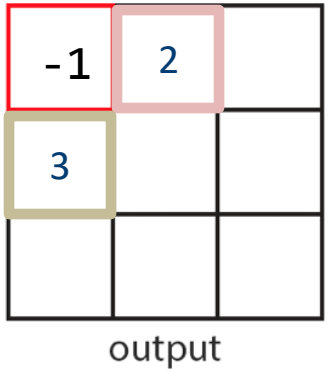




Stride 2 Example – With Padding







Convolutional Settings - Depth

- In images, we often have multiple numbers associated with each pixel location.
- These numbers are referred to as “channels”
  - RGB image – 3 channels
  - CMYK – 4 channels
- The number of channels is referred to as the “depth”
- So the kernel itself will have a “depth” the same size as the number of input channels
- Example: a 5x5 kernel on an RGB image
  - There will be 5x5x3 = 75 weights

Convolutional Settings - Depth

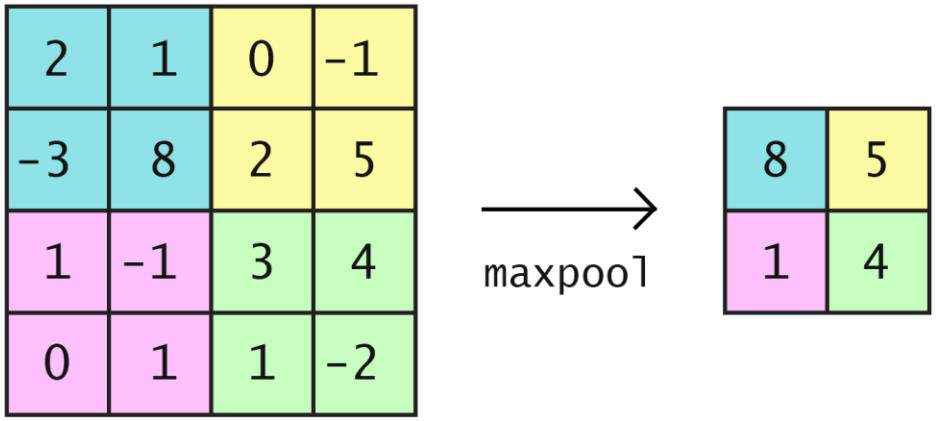
- The output from the layer will also have a depth
- The networks typically train many different kernels
- Each kernel outputs a single number at each pixel location
- So if there are 10 kernels in a layer, the output of that layer will have depth 10.

Pooling

- Idea: Reduce the image size by mapping a patch of pixels to a single value.
- Shrinks the dimensions of the image.
- Does not have parameters, though there are different types of pooling operations.

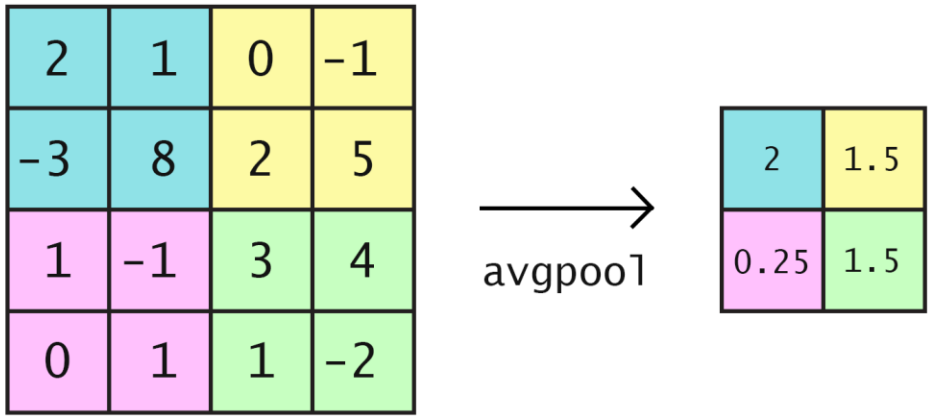
Pooling: Max-pool

- For each distinct patch, represent it by the maximum
- 2x2 maxpool shown below

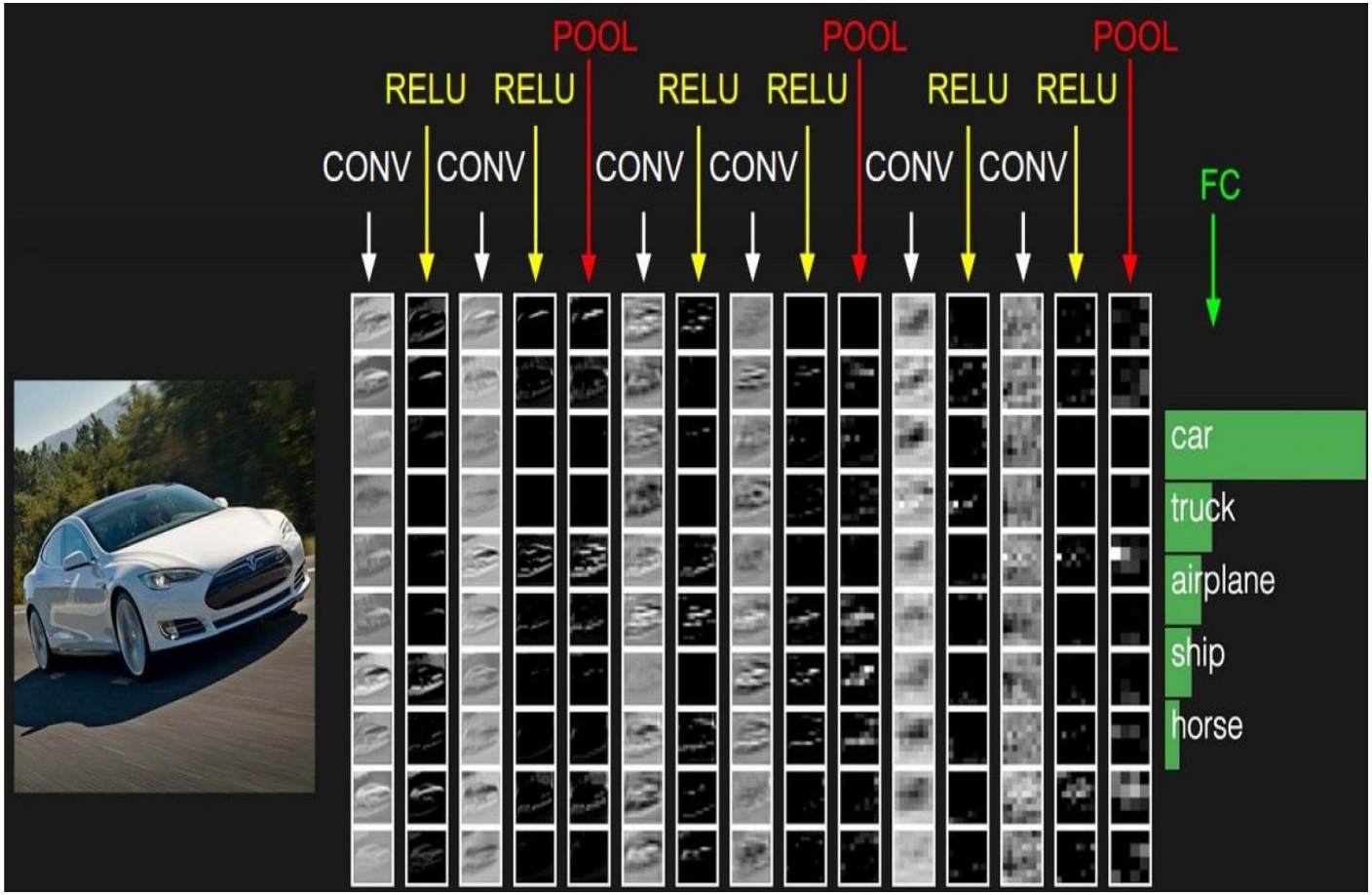


Pooling: Average-pool

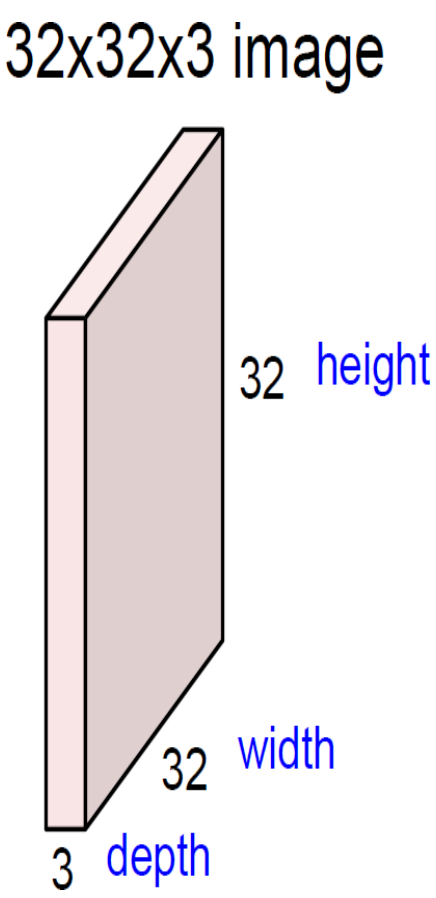
- For each distinct patch, represent it by the average
- 2x2 avgpool shown below.



ConvNet: CONV, RELU, POOL and FC Layers

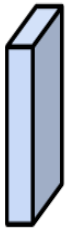


Convolution Layer



Filters always extend the full depth of the input volume

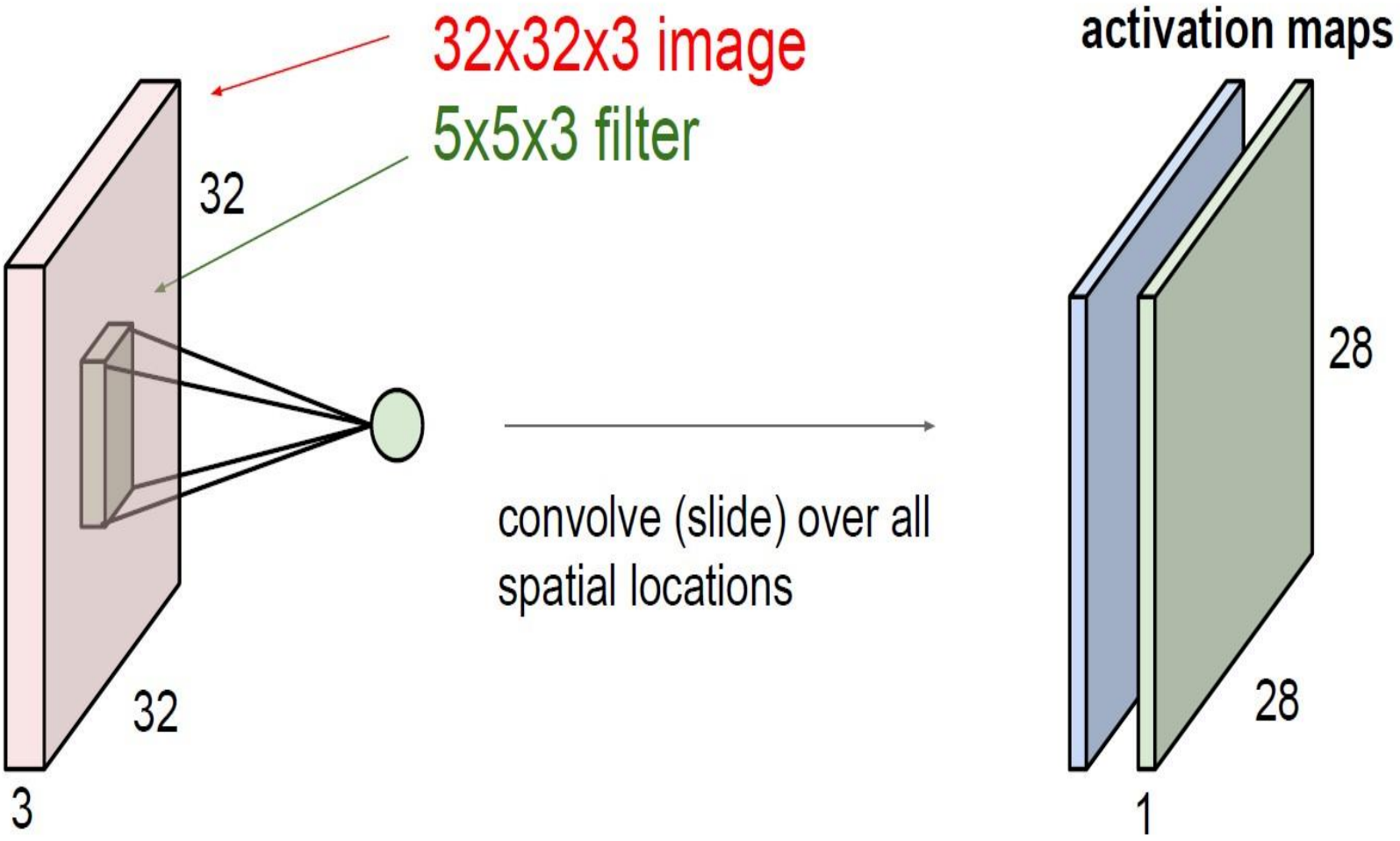
5x5x3 filter



Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”

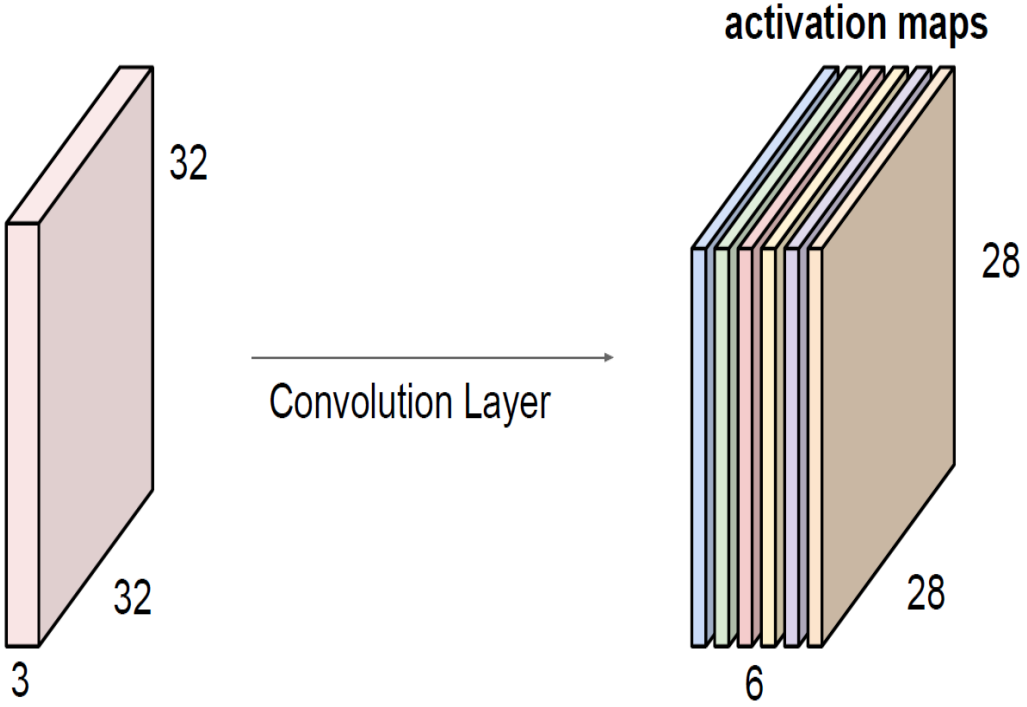
Convolution Layer

consider a second, green filter



Convolution Layer

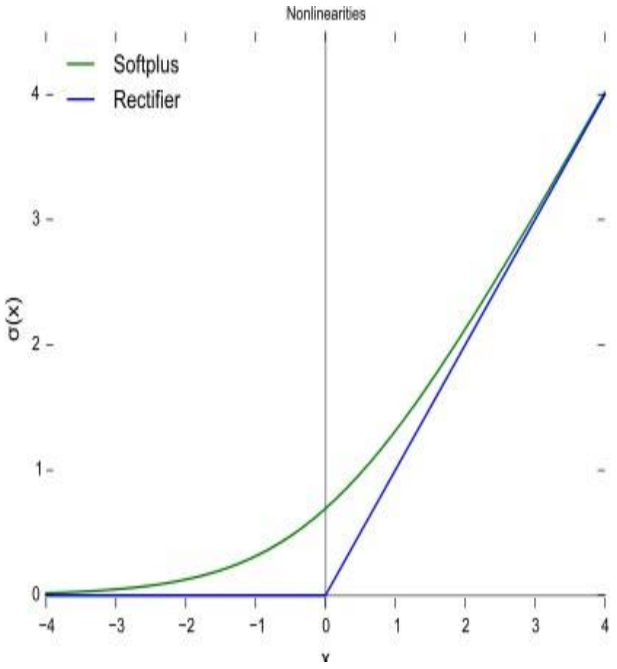
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



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

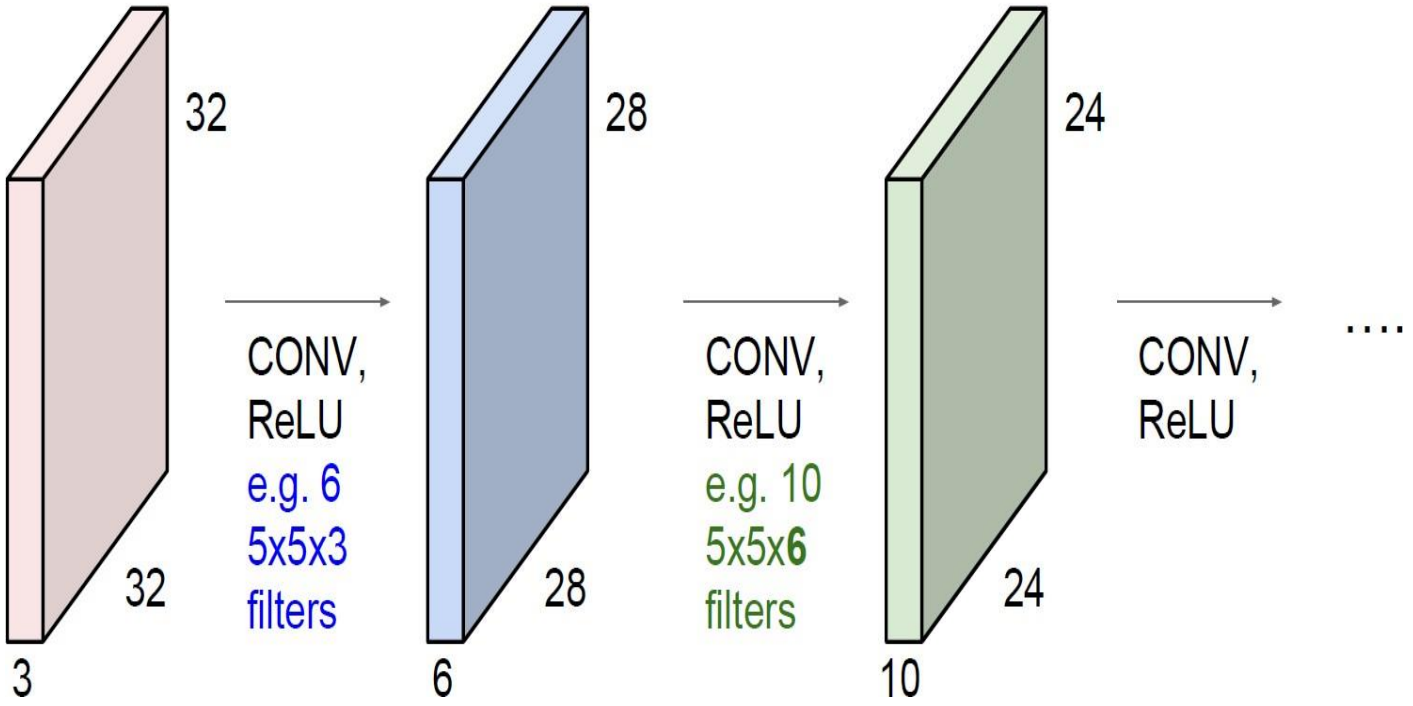
ReLU (Rectified Linear Units)Layer

- This is a layer of neurons that applies the activation function f(x)=max(0,x).
- It increases the nonlinear properties of the decision function and of the overall network without affecting the receptive fields of the convolutionlayer.
- Other functions are also used to increase nonlinearity, for example the hyperbolic tangent f(x)=tanh(x), and the sigmoid function.
- This is also known as a ramp function.

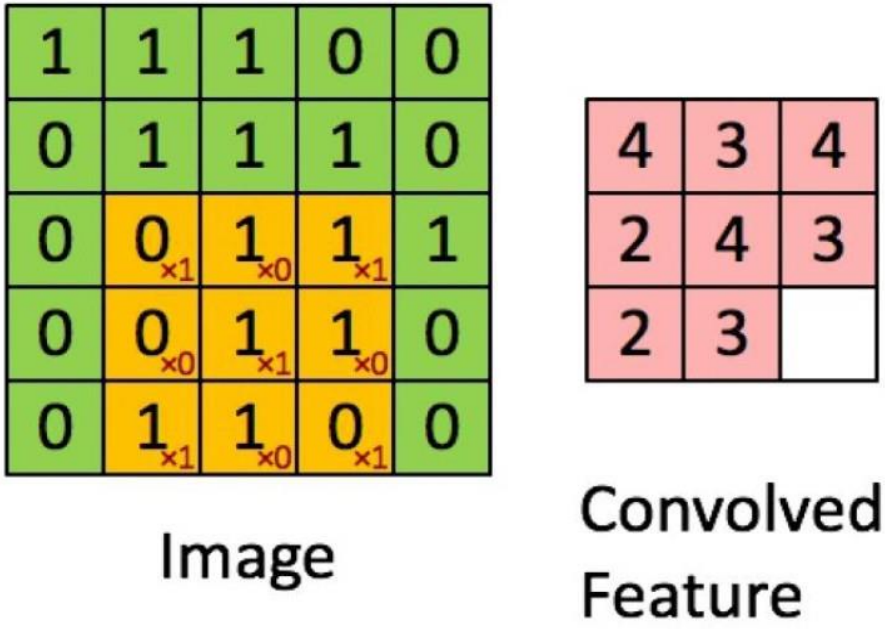


A Basic ConvNet

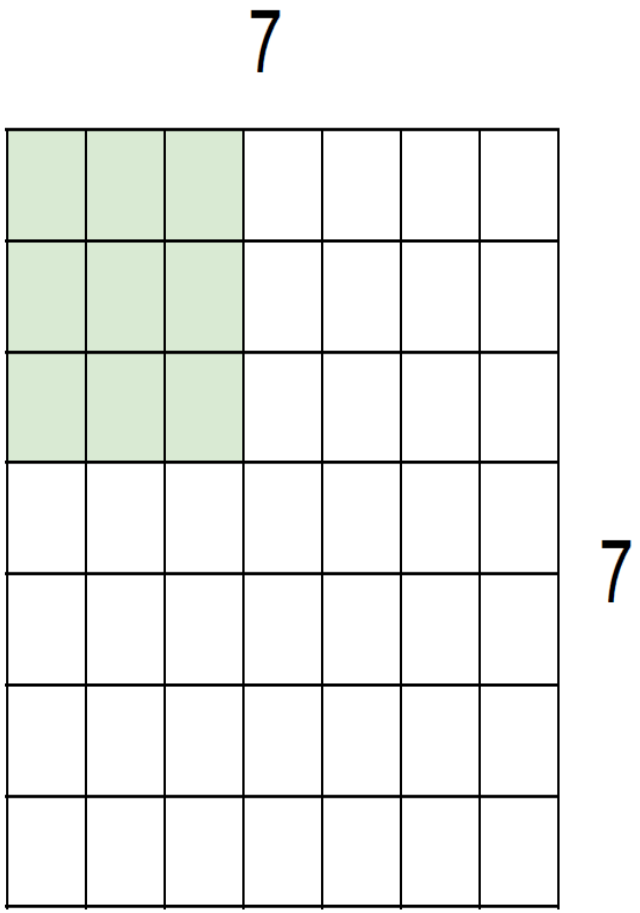
Preview: ConvNet is a sequence of Convolutional Layers, interspersed with activation functions



What is convolution of an image with a filter



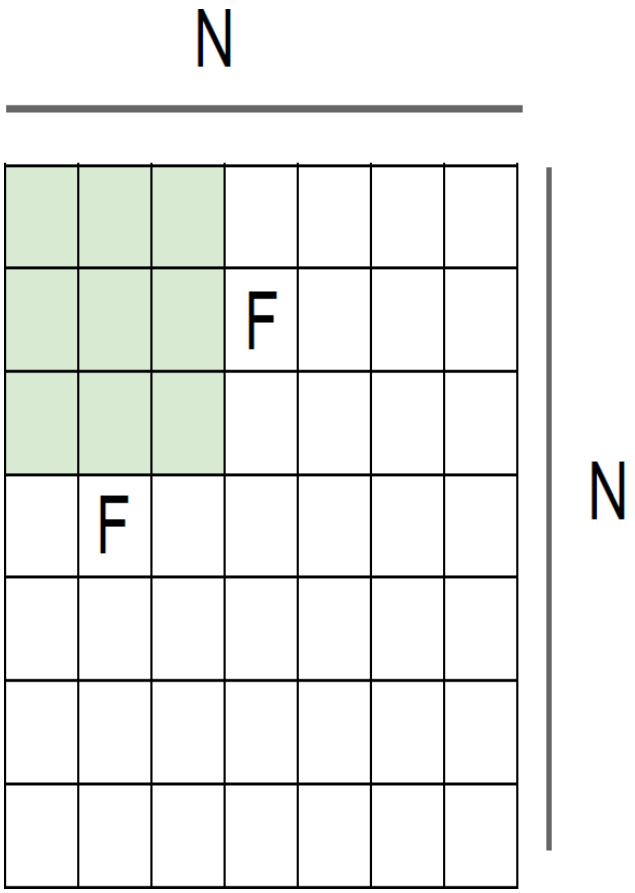
Details about the convolution layer



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

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

Details about the convolution layer

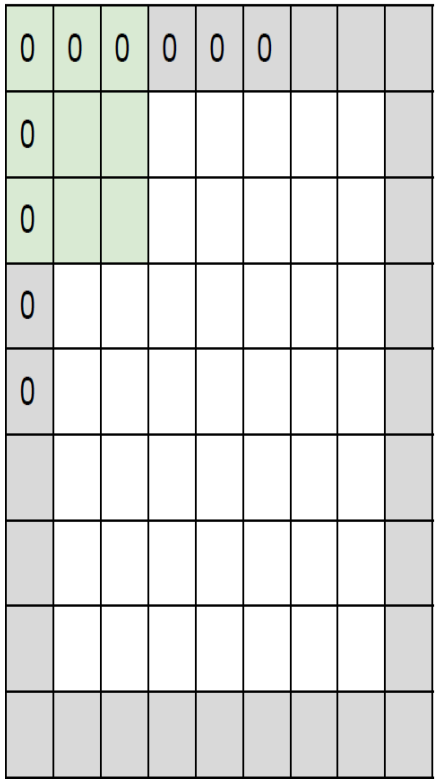


Output size:
(N - F) / stride + 1

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

Details about the convolution layer

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

Convolution layer examples

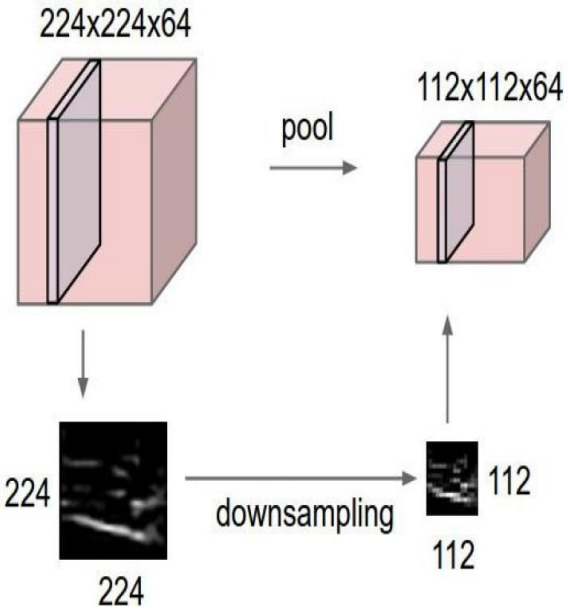
Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

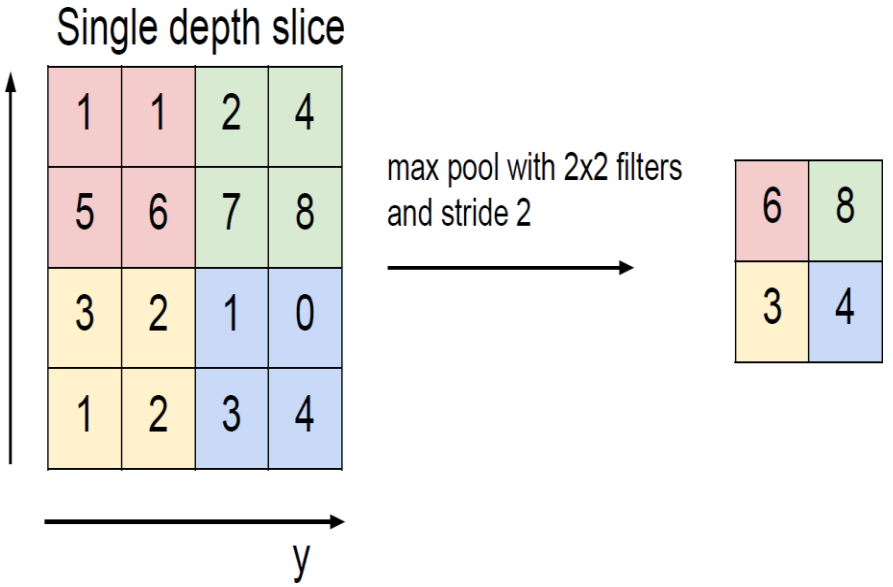
Output volume size: ?

(32+2*2-5)/1+1 = 32 spatially, so
32x32x10

Pooling Layer

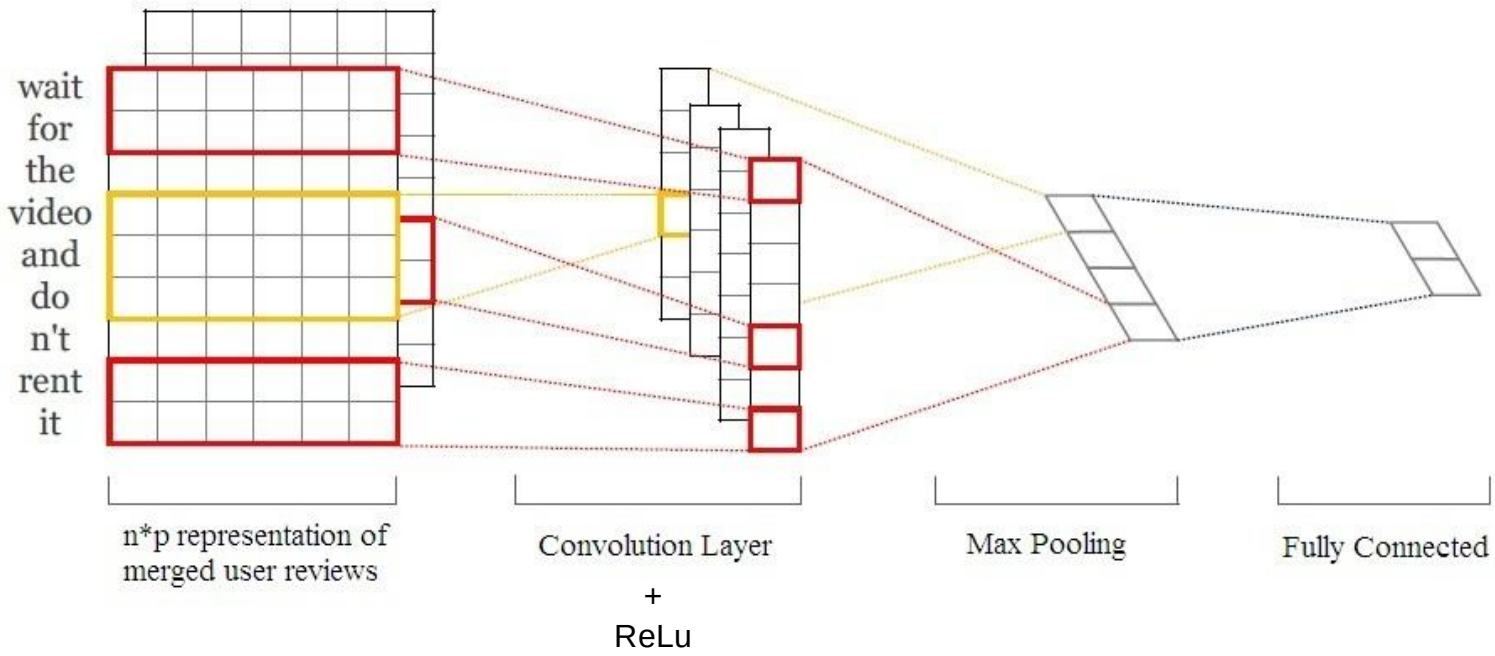
makes the representations smaller and more manageable
operates over each activation map independently:





- Invariance to image transformation and increases compactness to representation.
- Pooling types: Max, Average, L2 etc.

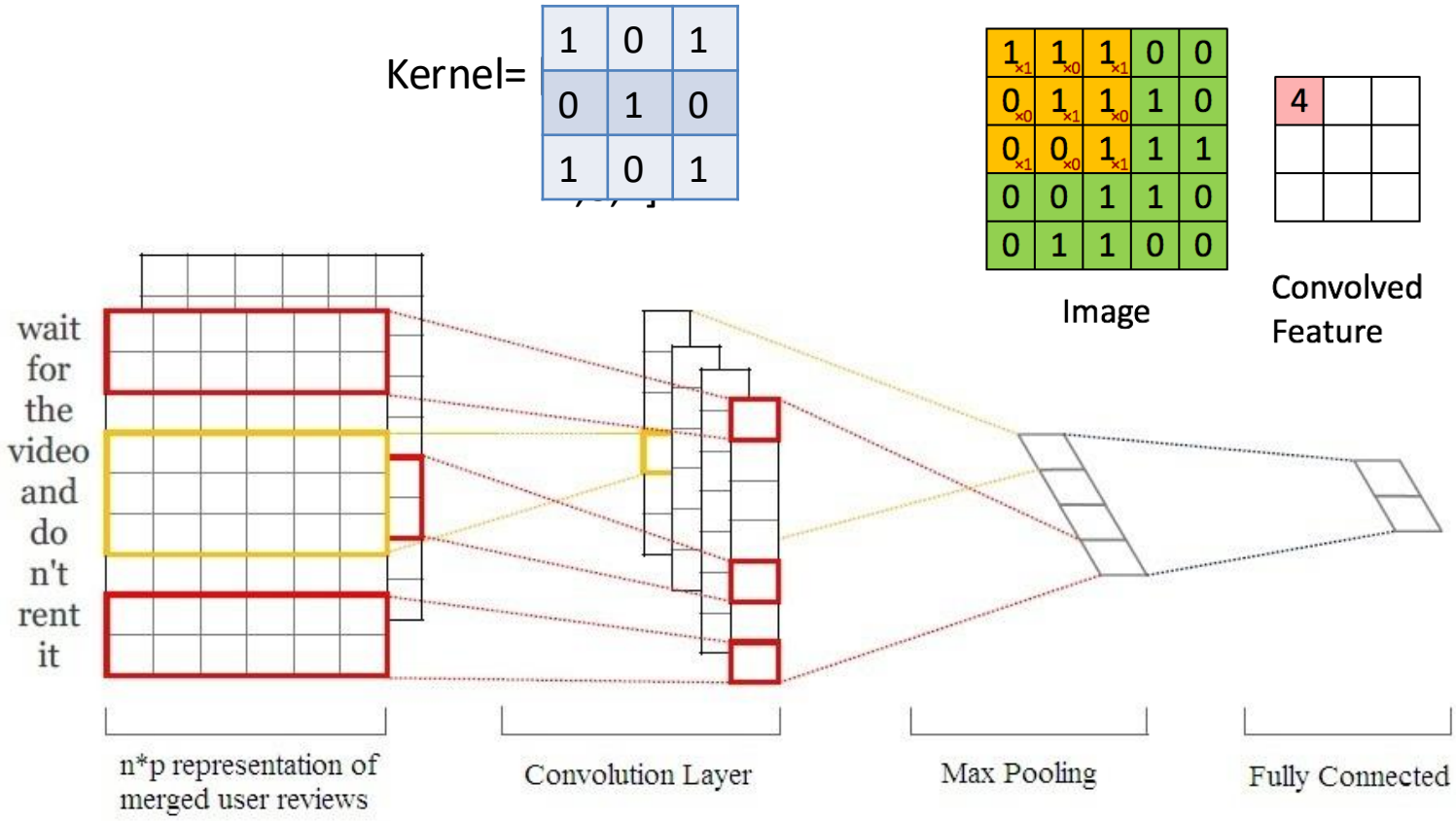
Convolutional Neural Networks



$$z_j = f(V^u_{1:n} * K_j + b_j)$$

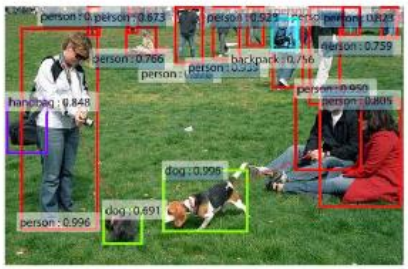
Where ReLu is used as f.

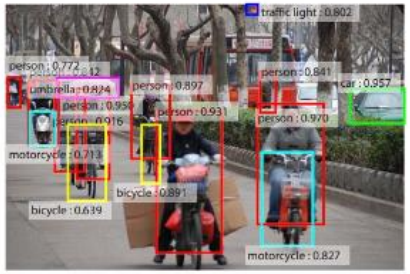
Convolutional Neural Networks



Applications

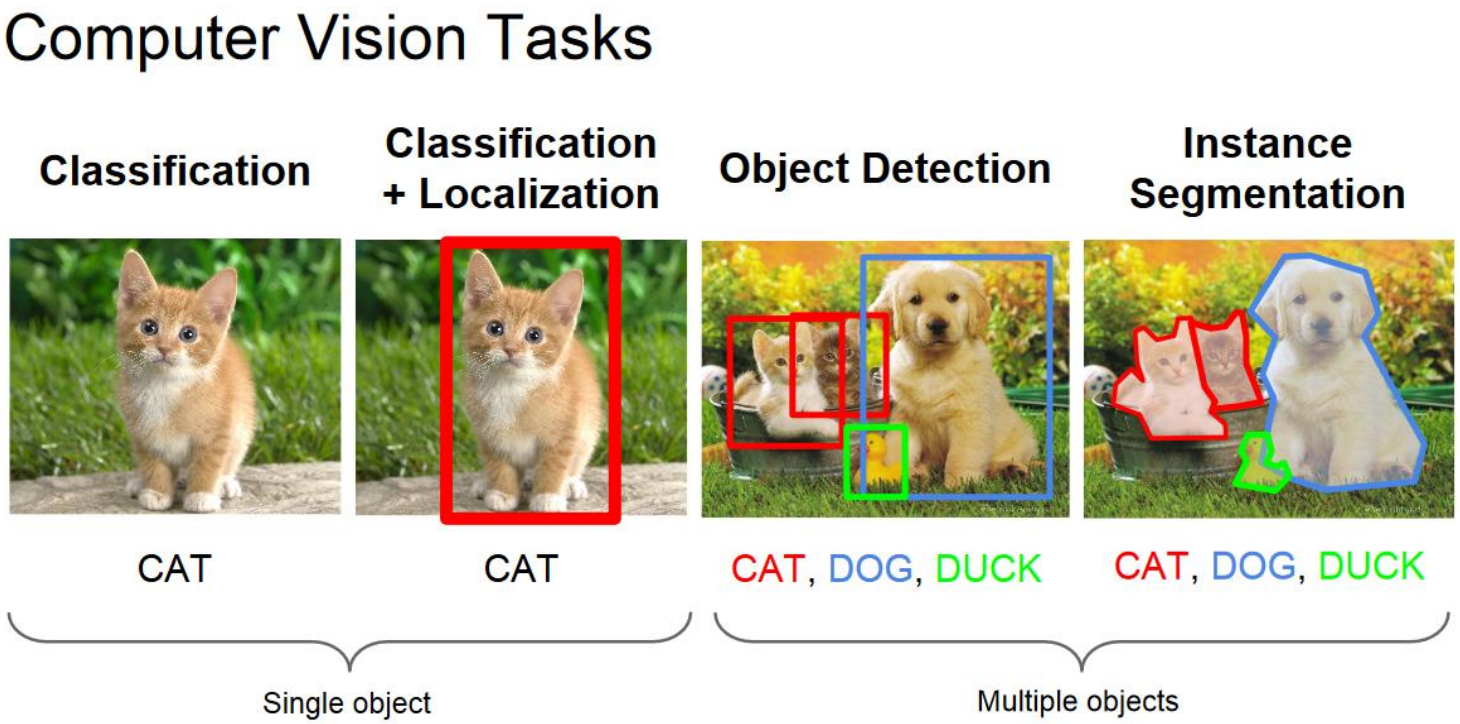
Localization and Detection





Results from Faster R-CNN, Ren et al 2015

Applications



ConvNet: CONV, RELU, POOL and FC Layers

Pytorch Implementation

```
import torch
import torch.nn as nn

class CCP(nn.Module):
    def __init__(self, in_channels, out_channels1, out_channels2, kernel_size=3, pool_kernel=2, stride=1, padding=1):
        super(CCP, self).__init__()

        # First Conv -> ReLU layer
        self.conv1 = nn.Conv2d(in_channels, out_channels1, kernel_size, stride, padding)
        self.relu1 = nn.ReLU()
        # Second Conv -> ReLU layer
        self.conv2 = nn.Conv2d(out_channels1, out_channels2, kernel_size, stride, padding)
        self.relu2 = nn.ReLU()
        # Pooling layer (default is max pooling)
        self.pool = nn.MaxPool2d(kernel_size=pool_kernel)

    def forward(self, x):
        # First conv -> ReLU
        x = self.relu1(self.conv1(x))
        # Second conv -> ReLU
        x = self.relu2(self.conv2(x))
        # Apply pooling
        x = self.pool(x)

        return x
```

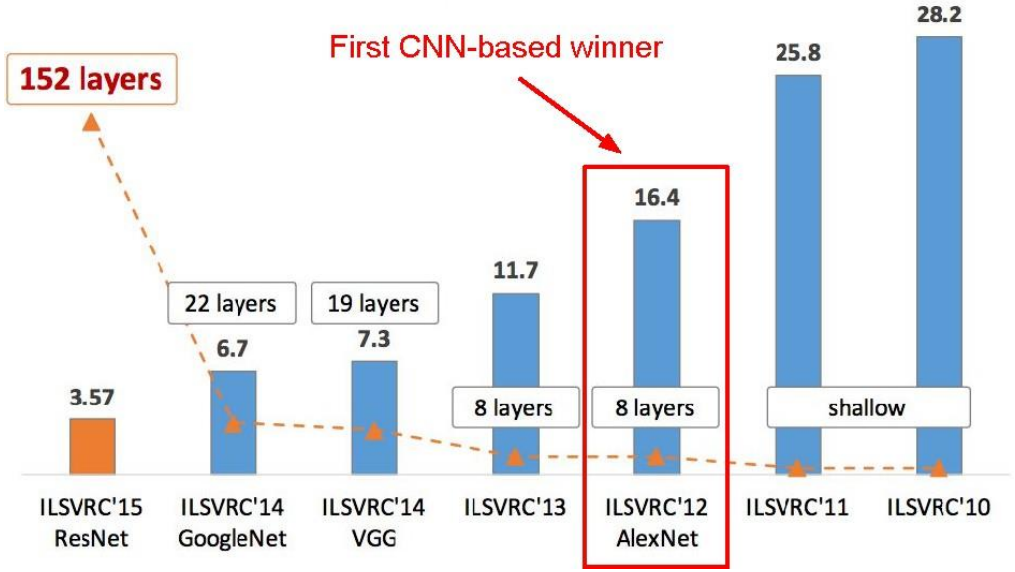
ConvNet: CONV, RELU, POOL and FC Layers

Pytorch Implementation

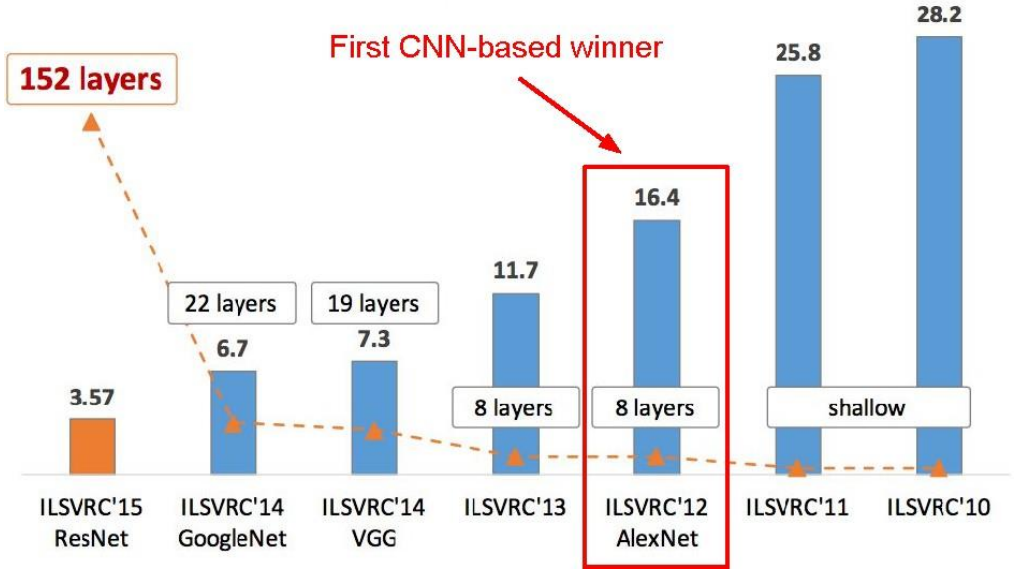
```python
class ClassificationModel(nn.Module):
    def __init__(self, num_classes):
        super(ClassificationModel, self).__init__()

        # Stack 3 layers of CCP
        self.layer1 = CCP(in_channels=3, out_channels1=64, out_channels2=64)  # For RGB image (3 channels)
        self.layer2 = CCP(in_channels=64, out_channels1=32, out_channels2=16)
        self.layer3 = CCP(in_channels=16, out_channels1=16, out_channels2=8)

        # Fully connected layer
        self.fc = nn.Linear(8 * 6 * 6, num_classes)  # Adjust based on final feature map size (example for 224x224 input)

    def forward(self, x):
        # Pass through stacked layers
        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)

        # Flatten the feature maps
        x = x.view(x.size(0), -1)  # Flatten the tensor

        # Fully connected layer
        x = self.fc(x)

        return x
```

EVOLUTION OF MODEL ARCHITECURES

ImageNet Large Scale Visual Recognition Challenge (ILSVRC)



ILSVRC


First CNN-based winner
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

AlexNet

Architecture
CONV1
MAX POOL1
NORM1
CONV2
MAX POOL2
NORM2
CONV3
CONV4
CONV5
Max POOL3
FC6
FC7
FC8

- Input: 227x227x3 images (224x224 before padding)
- First layer: 96 11x11 filters applied at stride 4
  - **Output volume size?**

    (N-F)/s+1 = (227-11)/4+1 = 55 -> [55x55x96]
  - **Number of parameters in this layer?**

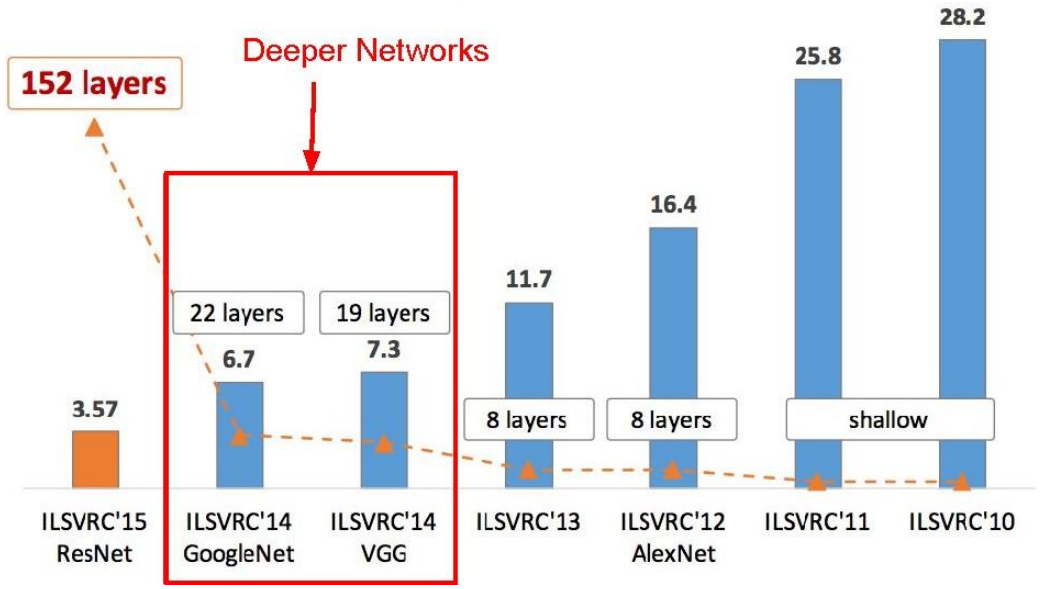
    (11*11*3)*96 = 35K

AlexNet

Details/Retrospectives:

- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- 7 CNN ensemble

ILSVRC winners



VGGNet

- **Smaller filters**
  Only 3x3 CONV filters, stride 1, pad 1 and 2x2 MAX POOL , stride 2

- **Deeper network**
  AlexNet: 8 layers
  VGGNet: 16 - 19 layers

- ZFNet: 11.7% top 5 error in ILSVRC'13
- VGGNet: 7.3% top 5 error in ILSVRC'14

VGGNet

- **Why use smaller filters? (3x3 conv)**

Stack of three 3x3 conv (stride 1) layers has the same effective receptive field as one 7x7 conv layer.

- **What is the effective receptive field of three 3x3 conv (stride 1) layers?**

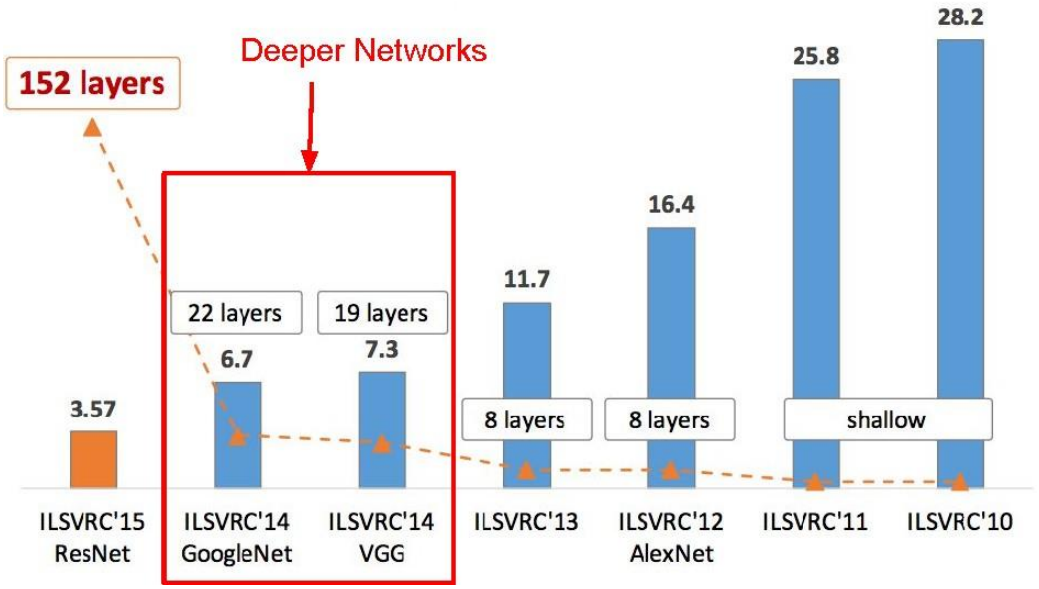
7x7

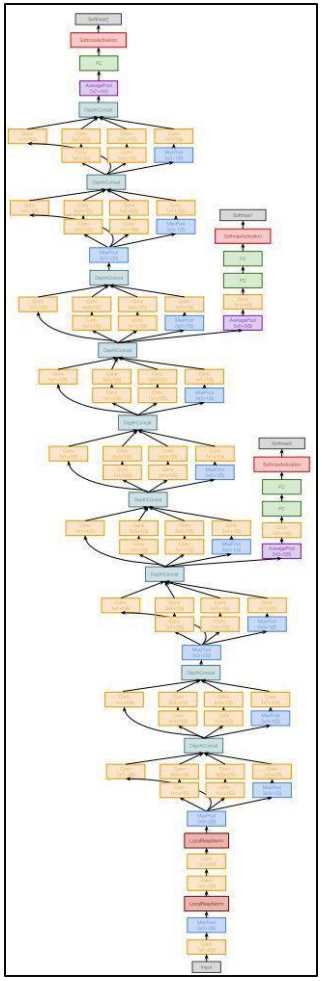
But deeper, more non-linearities

And fewer parameters: $3 * (3^2C^2)$ vs. 7^2C^2 for C channels per layer

[Simonyan and Zisserman, 2014]

ILSVRC winners



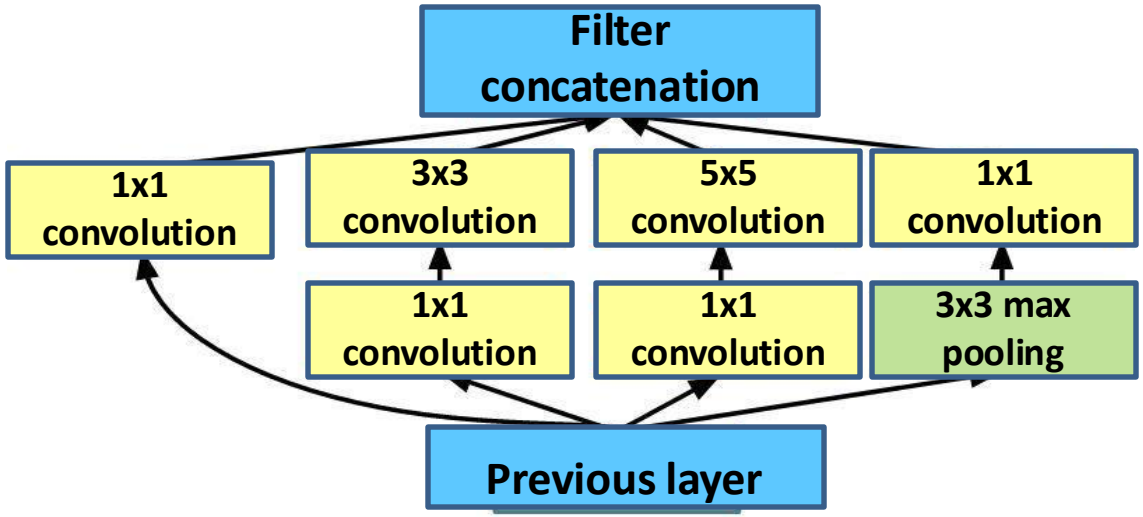


GoogleNet

- 22 layers
- Efficient **“Inception” module** - strayed from the general approach of simply stacking conv and pooling layers on top of each other in a sequential structure
- No FC layers
- Only 5 million parameters!
- ILSVRC’14 classification winner (6.7% top 5 error)

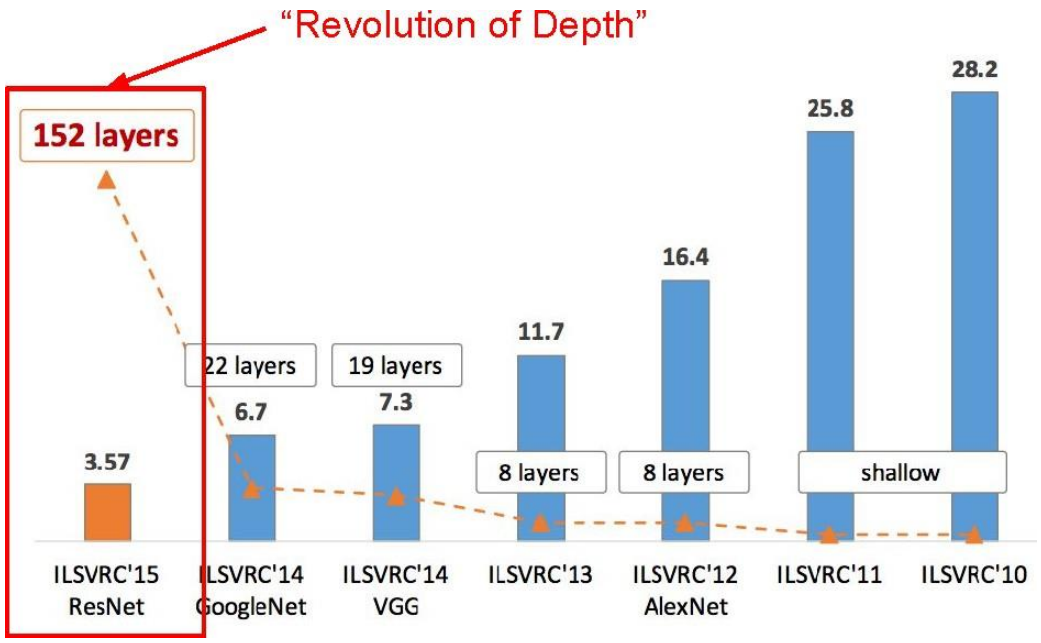
GoogleNet

"**Inception module":** design a good local network topology (network within a network) and then stack these modules on top of each other



[Szegedy et al., 2014]

ILSVRC winners



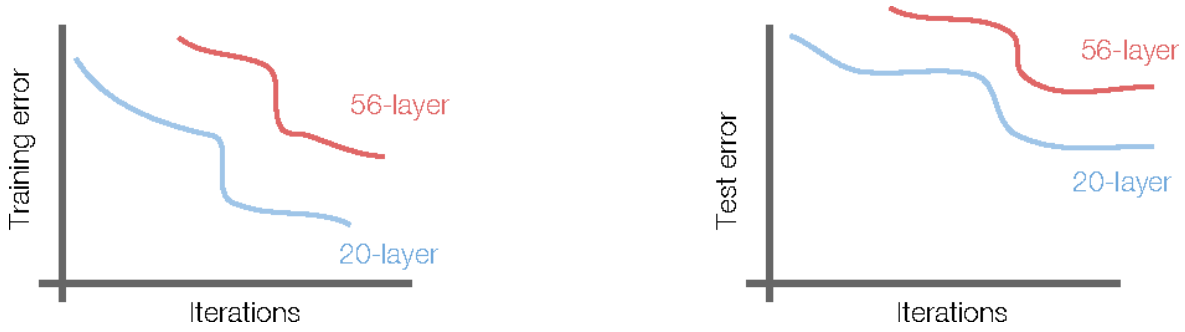
ResNet

- *Deep Residual Learning for Image Recognition - Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun; 2015*
- Extremely deep network – 152 layers
- Deeper neural networks are more difficult to train.
- Deep networks suffer from vanishing and exploding gradients.
- Present a residual learning framework to ease the training of networks that are substantially deeper than those used previously.

[He et al., 2015]

ResNet

- What happens when we continue stacking deeper layers on a convolutional neural network?



- 56-layer model performs worse on both training and test error

-> The deeper model performs worse (not caused by overfitting)!

ResNet

- **Hypothesis**: The problem is an optimization problem. Very deep networks are harder to optimize.
- **Solution**: Use network layers to fit residual mapping instead of directly trying to fit a desired underlying mapping.

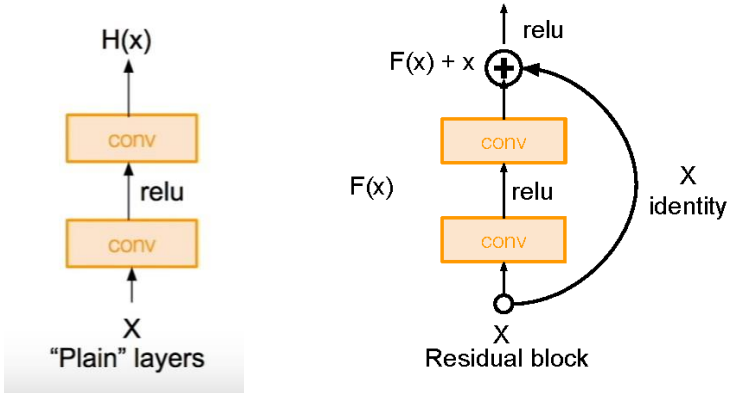
- We will use **skip connections** allowing us to take the activation from one layer and feed it into another layer, much deeper into the network.
- Use layers to fit residual $F(x) = H(x) – x$ instead of $H(x)$ directly

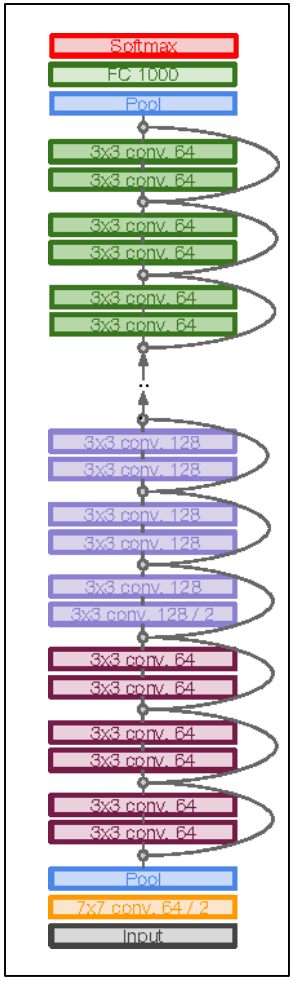
ResNet

Residual Block

Input x goes through conv-relu-conv series and gives us F(x). That result is then added to the original input x. Let's call that H(x) = F(x) + x.

In traditional CNNs, H(x) would just be equal to F(x). So, instead of just computing that transformation (straight from x to F(x)), we're computing the term that we have to *add*, F(x), to the input, x.



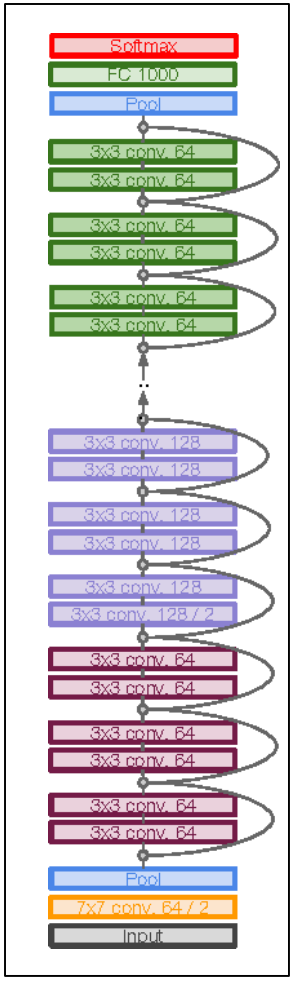


ResNet

Full ResNet architecture:

- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride 2 (in each dimension)
- Additional conv layer at the beginning
- No FC layers at the end (only FC 1000 to output classes)

[He et al., 2015]



ResNet

- Total depths of 34, 50, 101, or 152 layers for ImageNet
- For deeper networks (ResNet-50+), use "bottleneck" layer to improve efficiency (similar to GoogLeNet)

[He et al., 2015]

ResNet

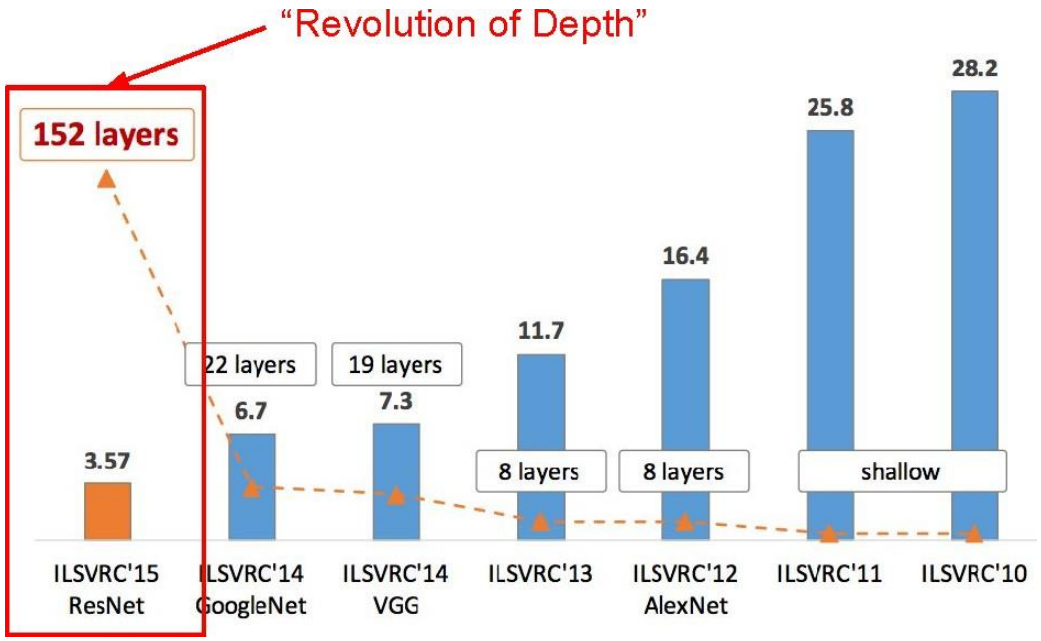
Experimental Results:

- Able to train very deep networks without degrading
- Deeper networks now achieve lower training errors as expected

The **best** CNN architecture that we currently have and is a great innovation for the idea of residual learning.

Even better than human performance!

ILSVRC winners



ARCHITECTURES FOR ADVANCED APPLICATIONS

Computer Vision Tasks

Classification

CAT

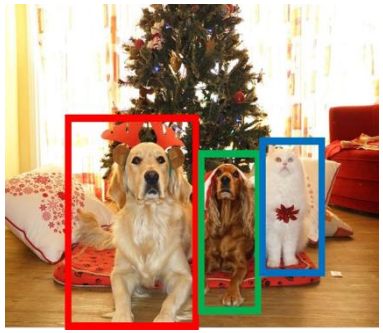
No spatial extent

Semantic Segmentation

GRASS, CAT, TREE, SKY

No objects, just pixels

Object Detection



DOG, DOG, CAT

Instance Segmentation

DOG, DOG, CAT

Multiple Object

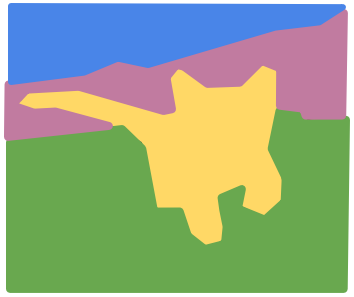
Semantic Segmentation

Classification

CAT

No spatial extent

Semantic Segmentation



GRASS, CAT, TREE, SKY

No objects, just pixels

Object Detection

DOG, DOG, CAT

Instance Segmentation

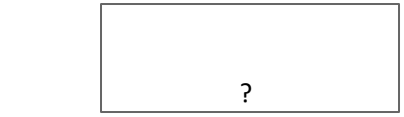
DOG, DOG, CAT

Multiple Object

Semantic Segmentation: The Problem

GRASS, CAT, TREE, SKY, …

Paired training data: for each training image, each pixel is labeled with a semantic category.



At test time, classify each pixel of a new image.

Semantic Segmentation Idea: Sliding Window

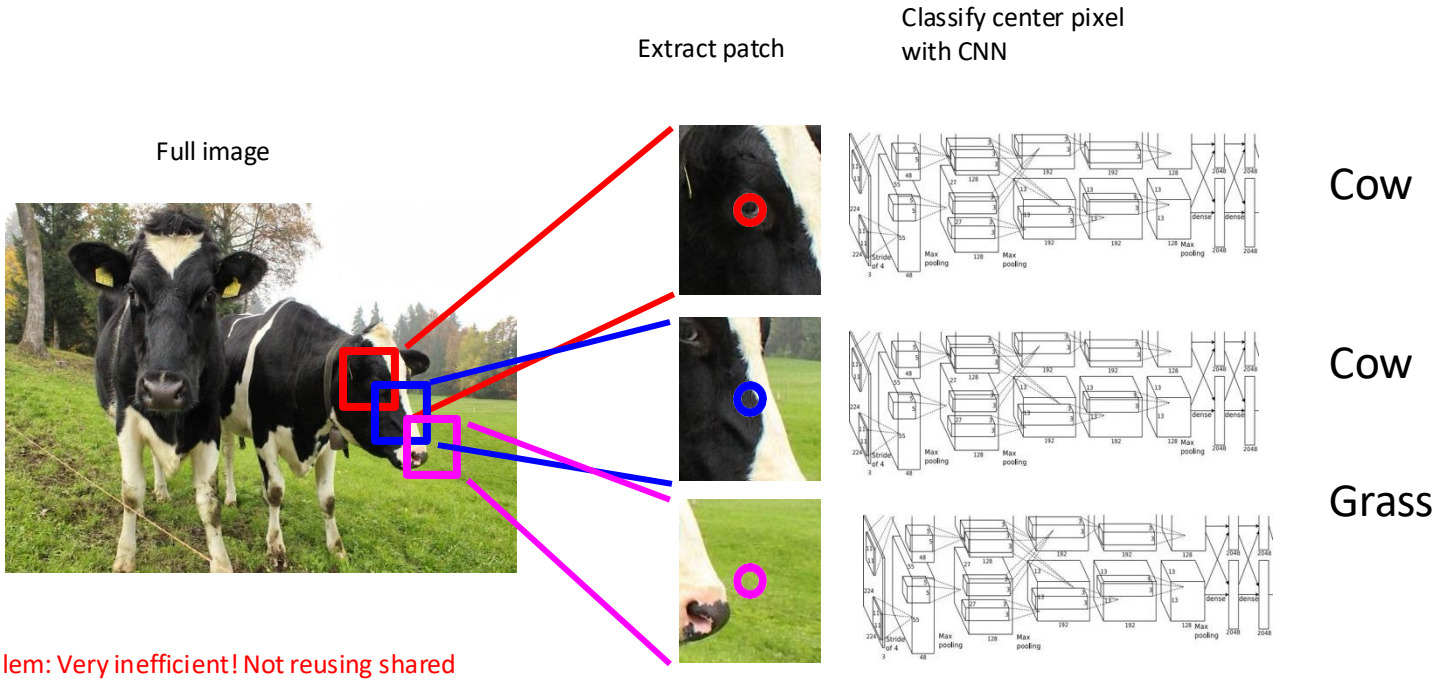
Full image

?

Impossible to classify without context

Q: how do we include context?

Semantic Segmentation Idea: Sliding Window

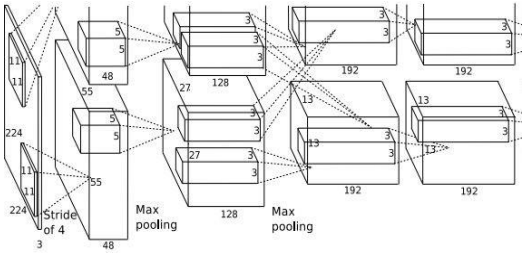


Problem: Very inefficient! Not reusing shared features between overlapping patches

Farabet et al, "Learning Hierarchical Features for Scene Labeling," TPAMI 2013
Pinheiro and Collobert, "Recurrent Convolutional Neural Networks for Scene Labeling", ICML 2014

Semantic Segmentation Idea: Convolution

Full image

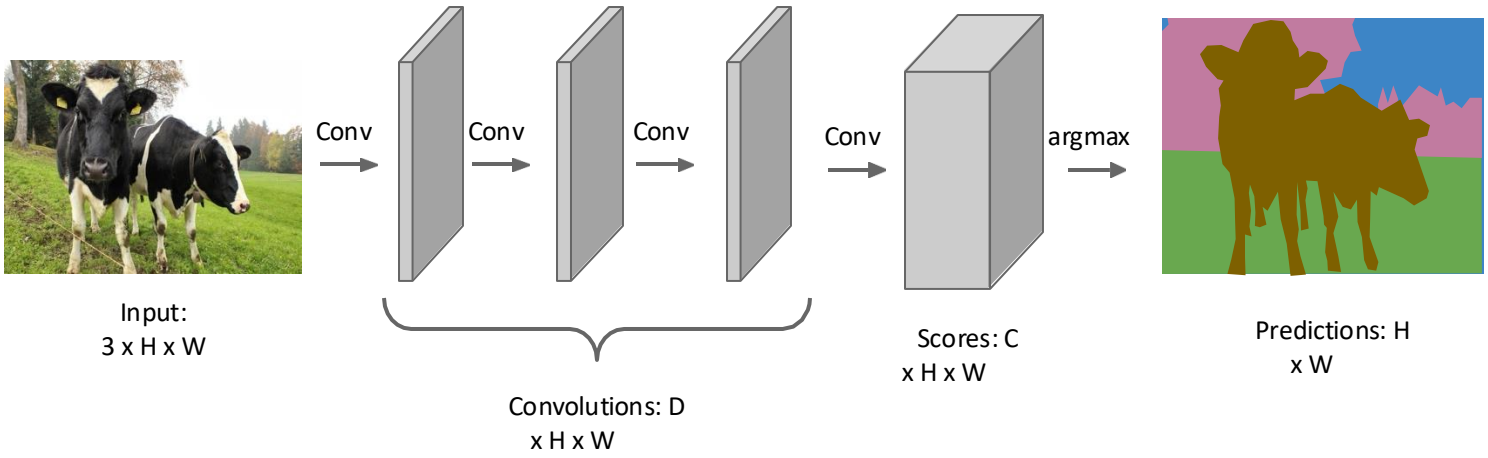


An intuitive idea: encode the entire image with conv net, and do semantic segmentation on top.

Problem: classification architectures often reduce feature spatial sizes to go deeper, but semantic segmentation requires the output size to be the same as input size.

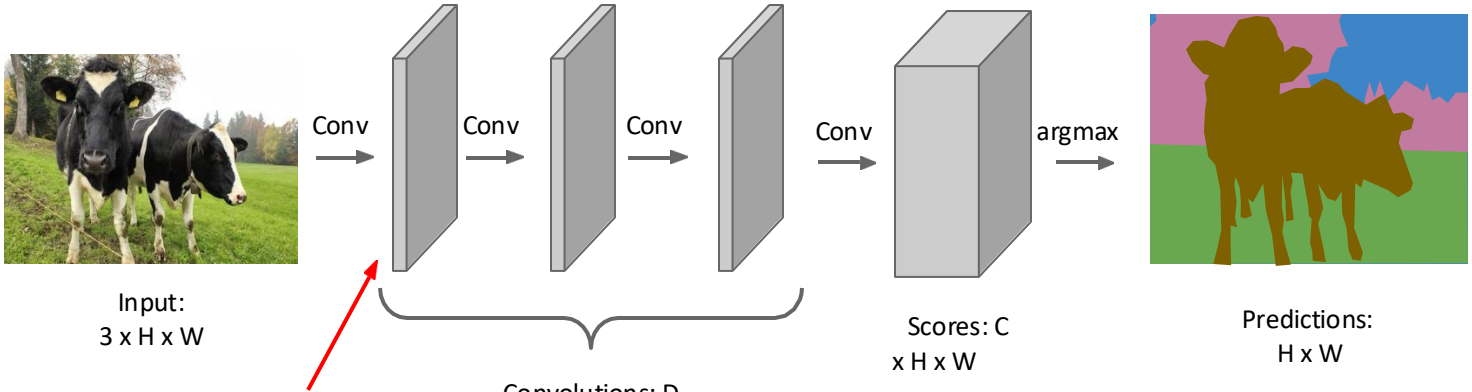
Semantic Segmentation Idea: Fully Convolutional

Design a network with only convolutional layers without downsampling operators to make predictions for pixels all at once!



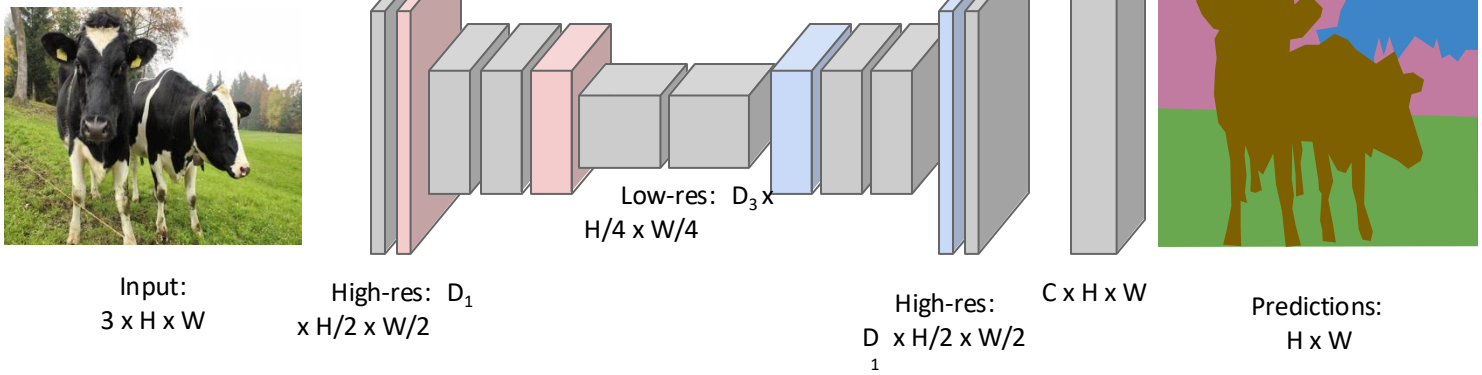
Semantic Segmentation Idea: Fully Convolutional

Design a network with only convolutional layers without downsampling operators to make predictions for pixels all at once!



Semantic Segmentation Idea: Fully Convolutional

Design network as a bunch of convolutional layers, with **downsampling** and **upsampling** inside the network!



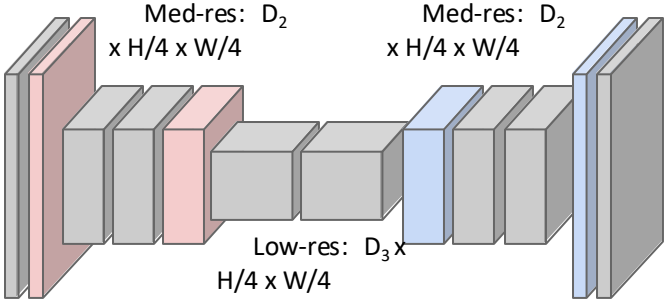
Semantic Segmentation Idea: Fully Convolutional

Downsampling:
Pooling, strided convolution

Design network as a bunch of convolutional layers, with **downsampling** and **upsampling** inside the network!

Upsampling:
???

Input:
3 x H x W



High-res: D_1 x H/2 x W/2

High-res: D x H/2 x W/2

C x H x W

Predictions:
H x W

Long, Shelhamer, and Darrell, "Fully Convolutional Networks for Semantic Segmentation", CVPR 2015
Noh et al, "Learning Deconvolution Network for Semantic Segmentation", ICCV 2015

OBJECT DETECTION

Object Detection

Classification

CAT

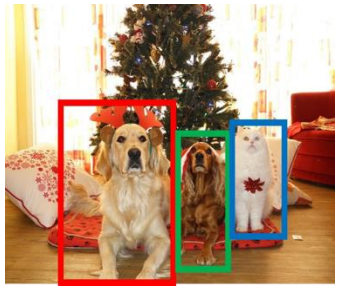
No spatial extent

Semantic Segmentation

GRASS, CAT, TREE, SKY

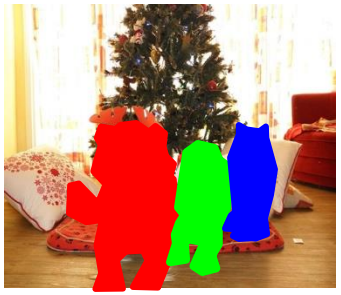
No objects, just pixels

Object Detection



DOG, DOG, CAT

Instance Segmentation

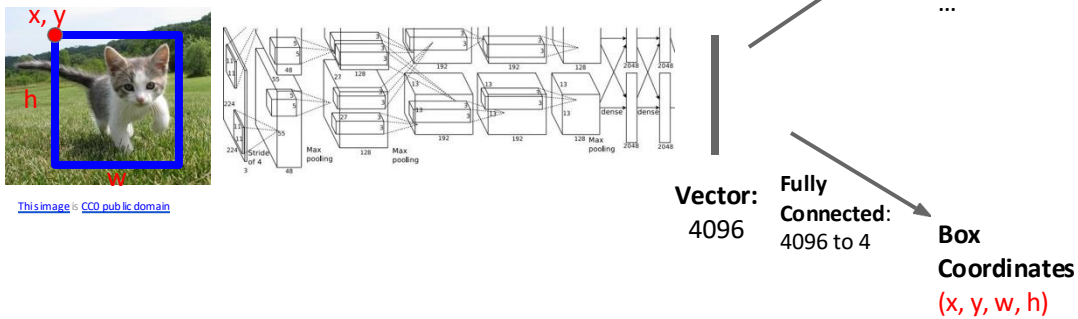


DOG, DOG, CAT

Multiple Object

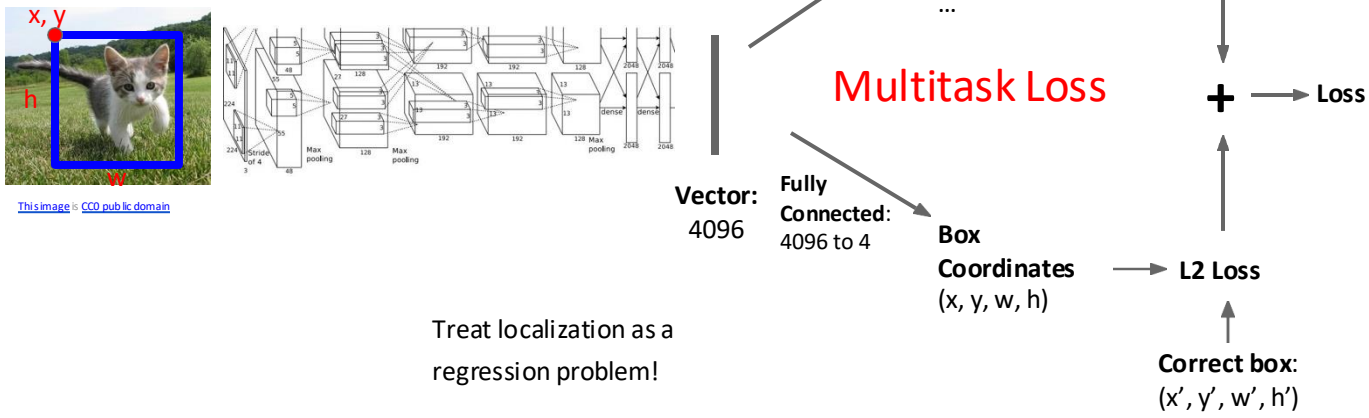
Object Detection: Single Object

(Classification + Localization)



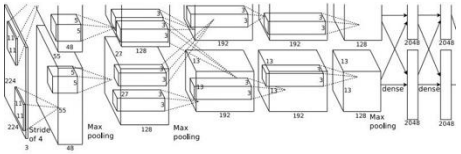
Object Detection: Single Object

(Classification + Localization)



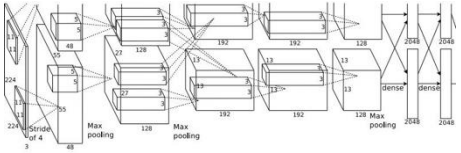
Object Detection: Multiple Objects

Each image needs a different number of outputs!



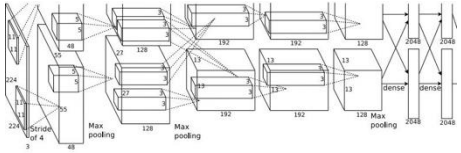
CAT: (x, y, w, h)

4 numbers



DOG: (x, y, w, h)
DOG: (x, y, w, h)
CAT: (x, y, w, h)

12 numbers

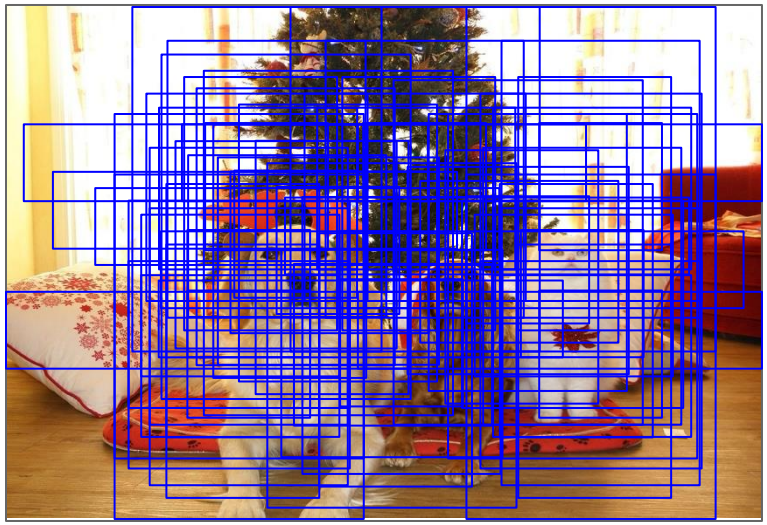


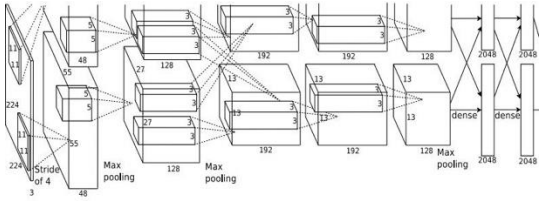
DUCK: (x, y, w, h)
DUCK: (x, y, w, h)
....

Many numbers!

Object Detection: Multiple Objects

Apply a CNN to many different crops of the image, CNN classifies each crop as object or background



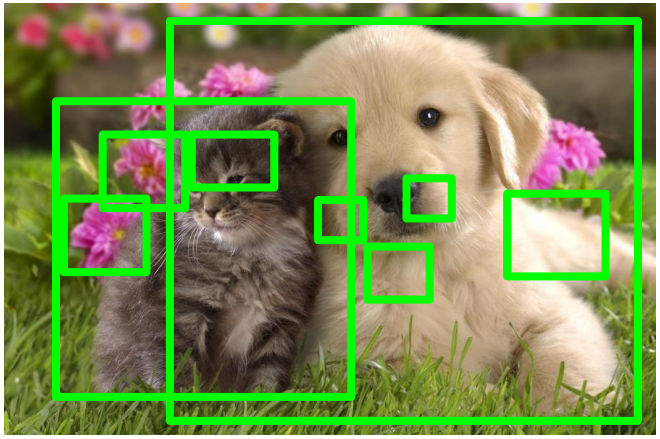


Dog? NO
Cat? YES
Background? NO

Problem: Need to apply CNN to huge number of locations, scales, and aspect ratios, very computationally expensive!

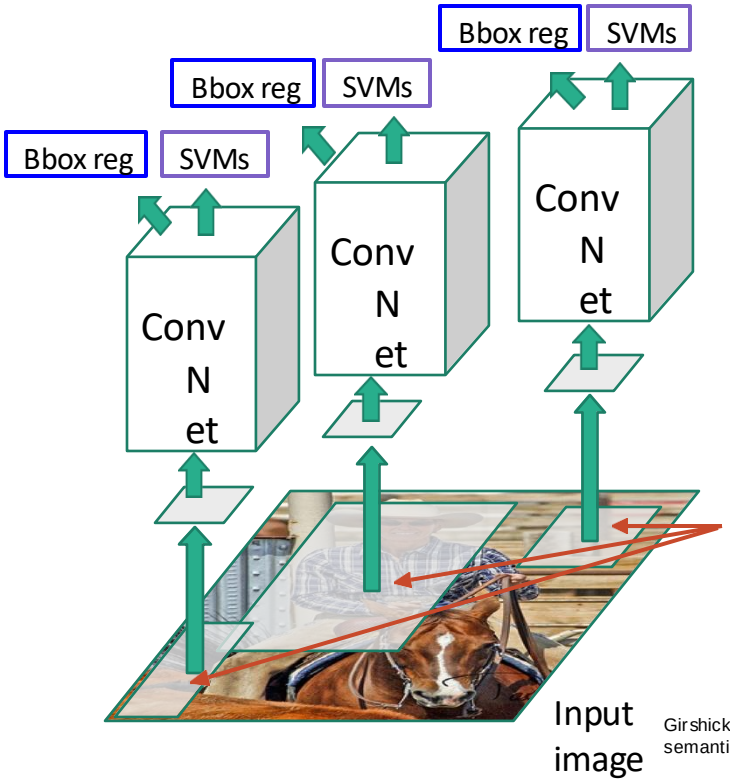
Region Proposals: Selective Search

- Find “blobby” image regions that are likely to contain objects
- Relatively fast to run; e.g. Selective Search gives 2000 region proposals in a few seconds on CPU



Alexe et al, “Measuring the objectness of image windows”, TPAMI 2012
Uijlings et al, “Selective Search for Object Recognition”, IJCV 2013
Cheng et al, “BING: Binarized normed gradients for objectness estimation at 300fps”, CVPR 2014
Zitnick and Dollar, “Edge boxes: Locating object proposals from edges”, ECCV 2014

“Slow” R-CNN



Predict “corrections” to the RoI: 4 numbers: (dx, dy, dw, dh)

Classify regions with SVMs

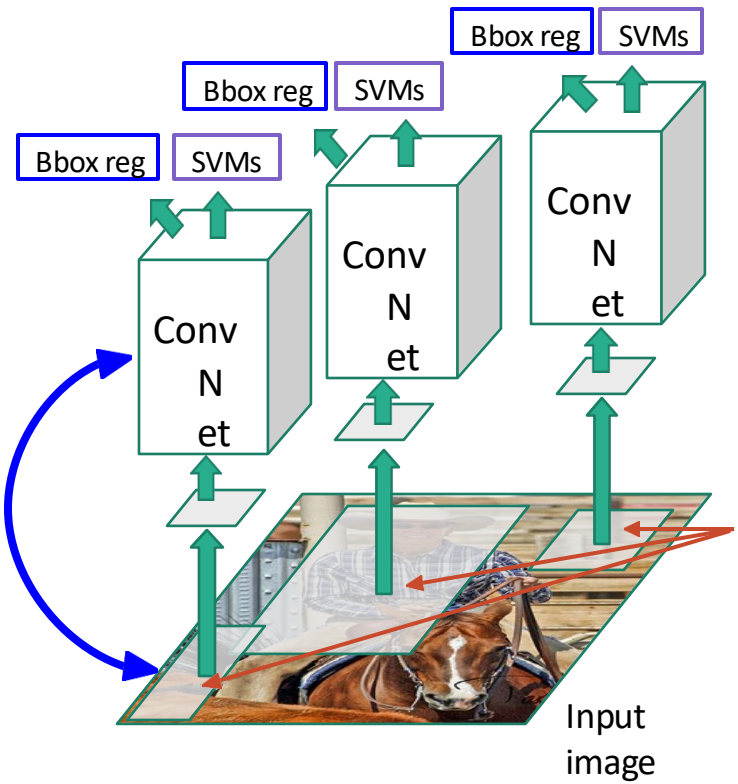
Forward each region through ConvNet

Warped image regions (224x224 pixels)

(RoI) from a proposal method (~2k)

Girshick et al, “Rich feature hierarchies for accurate object detection and semantic segmentation”, CVPR 2014.

"Slow" R-CNN



Predict "corrections" to the RoI: 4 numbers: (dx, dy, dw, dh)

Classify regions with SVMs

Forward each region through ConvNet

Warped image regions (224x224 pixels)

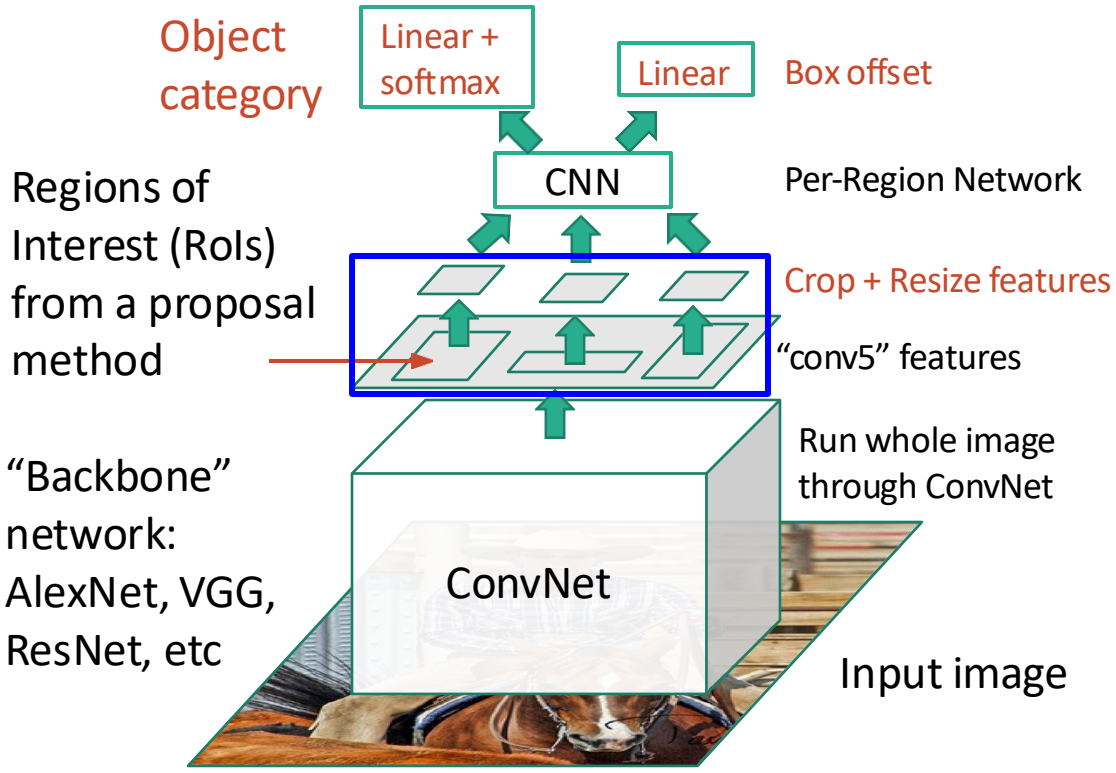
(RoI) from a proposal method (~2k)

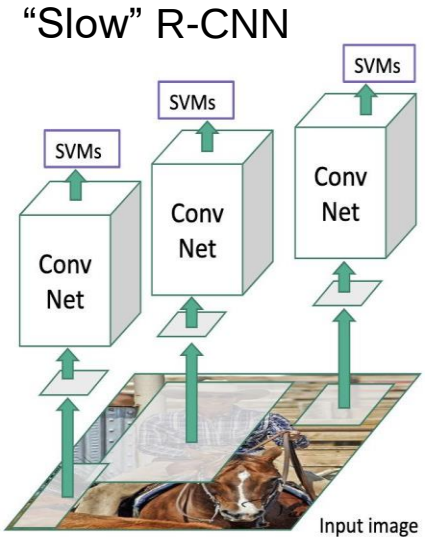
Problem: Very slow! Need to do ~2k independent forward passes for each image!

Idea: Pass the image through convnet before cropping! Crop the conv feature instead!

Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.

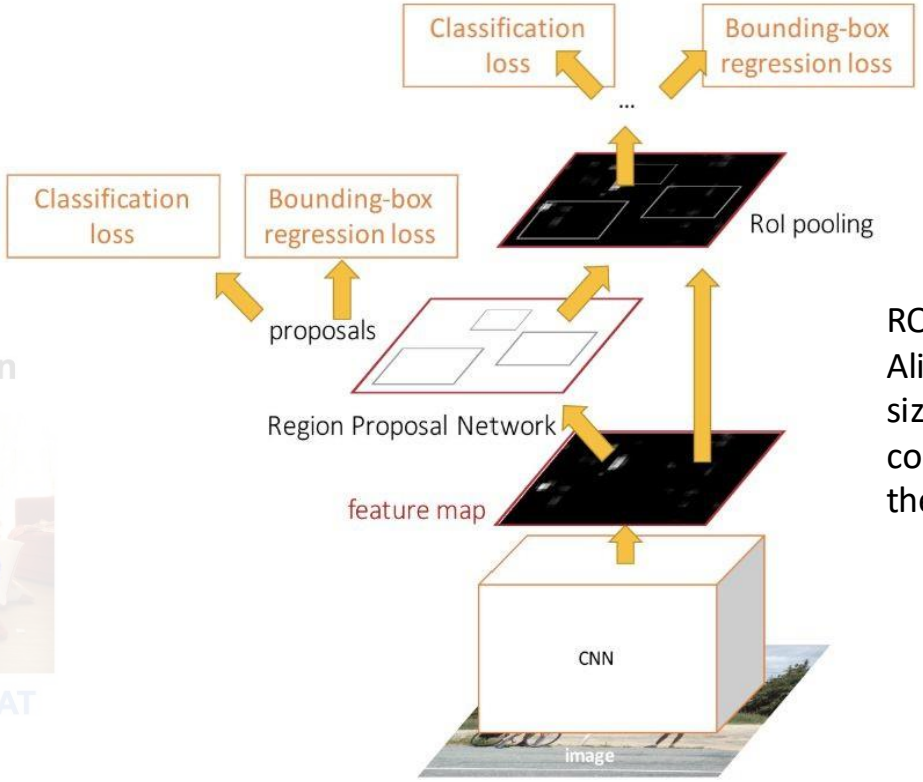
Fast R-CNN





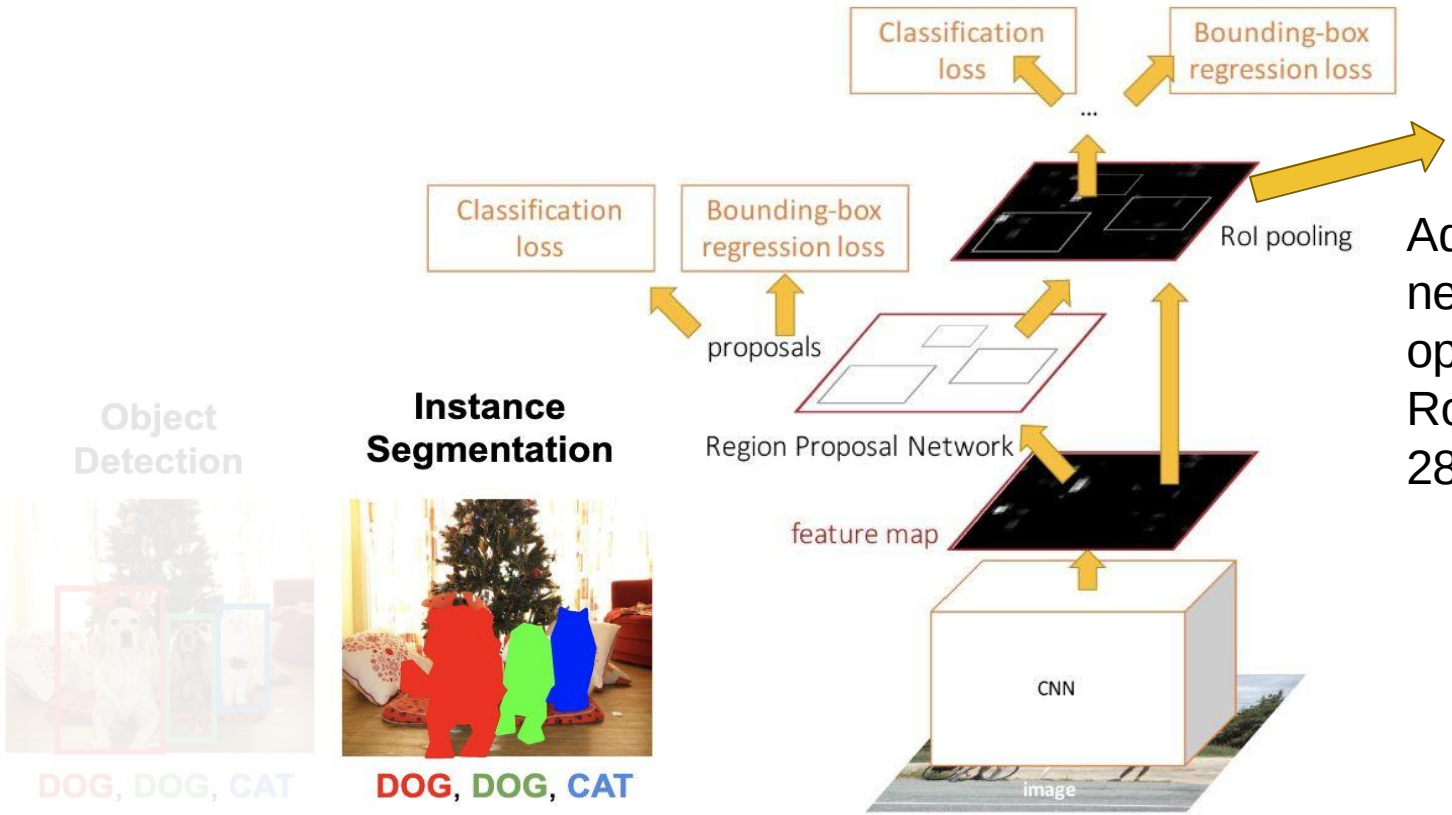
Girshick, "Fast R-CNN", ICCV 2015.

Object Detection: Faster R-CNN



ROI pooling / ROI Alignment creates fixed size feature maps from convolutional maps of the ROI.

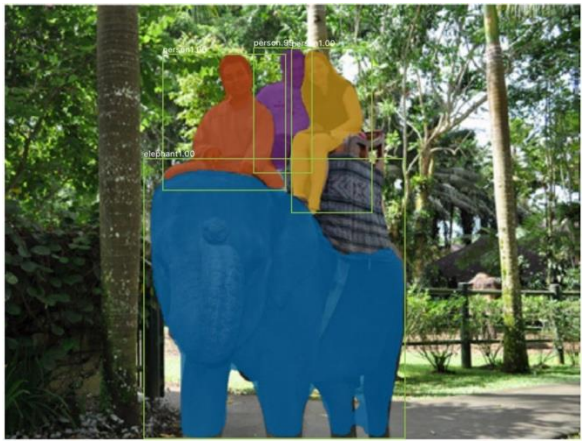
Instance Segmentation: Mask R-CNN

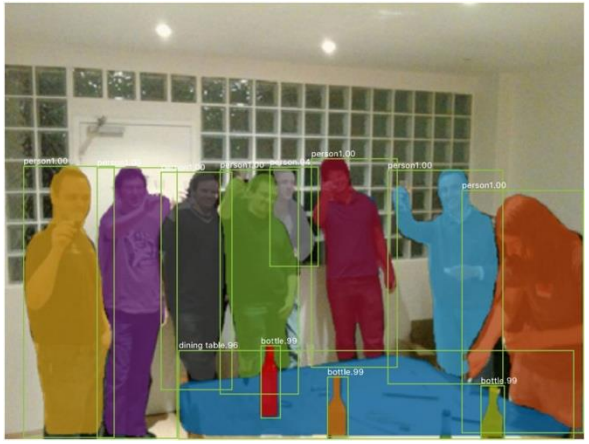


Add a small mask network that operates on each RoI and predicts a 28x28 binary mask

He et al, “Mask R-CNN”, ICCV 2017

Mask R-CNN: Very Good Results!





He et al, “Mask R-CNN”, ICCV 2017

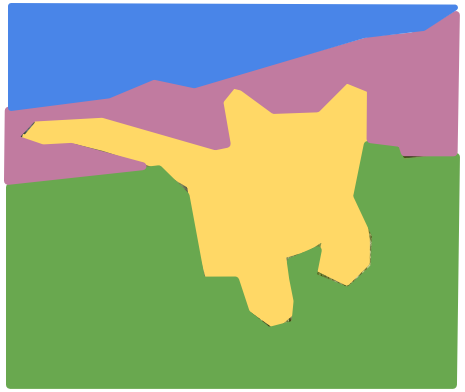
Summary: Lots of computer vision tasks!

Classification

CAT

No spatial extent

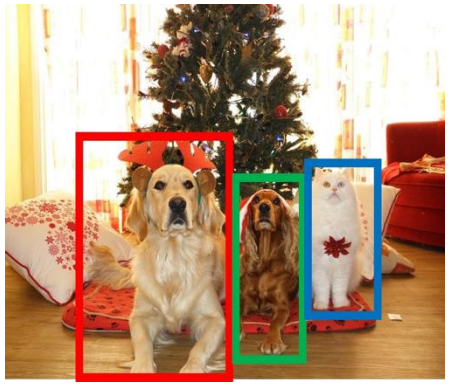
Semantic Segmentation



GRASS, **CAT**, **TREE**, **SKY**

No objects, just pixels

Object Detection



DOG, **DOG**, **CAT**

Instance Segmentation

DOG, **DOG**, **CAT**

Multiple Object

References

- CS231n: Deep Learning for Computer Vision.
  https://cs231n.stanford.edu/
  Lecture slides, and lecture videos

- NPTEL course: Deep Learning for Computer Vision
  Vineeth N Balasubramanian
  https://onlinecourses.nptel.ac.in/noc20_cs88/preview