ECE 253

Digital Image Processing

Introduction to Convolutional Neural Networks

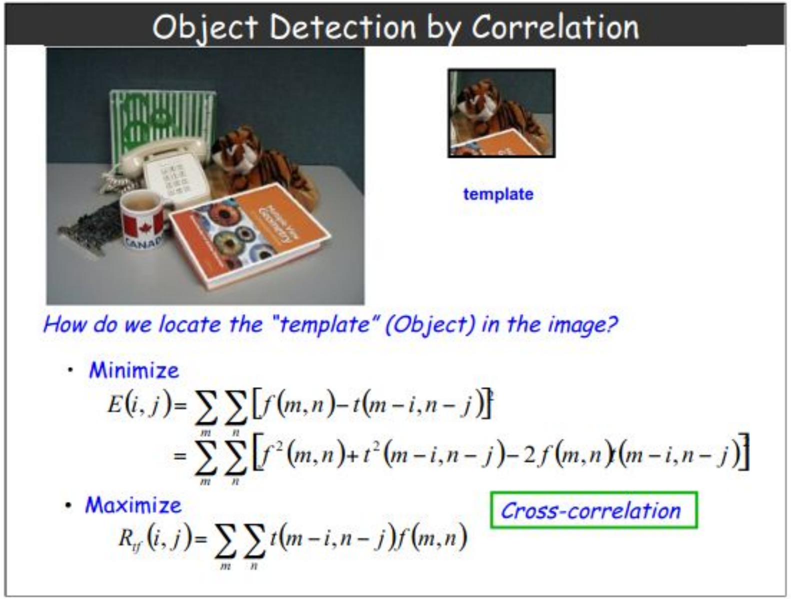
HW 3/4 Problem:

Given an image, can we determine the object?

Object: Stop Sign

HW 3/4 Problem:

Given an image, can we determine the object?



Object: Stop Sign

HW 3/4 Problem:

Given an image, can we determine the object?

Why can’t we just use correlation?

Object: Stop Sign

HW 3/4 Problem:

Given an image, can we determine the object?

Object: Stop Sign

Why can’t we just use correlation?

HW 3/4 Problem:

Given an image, can we determine the object?

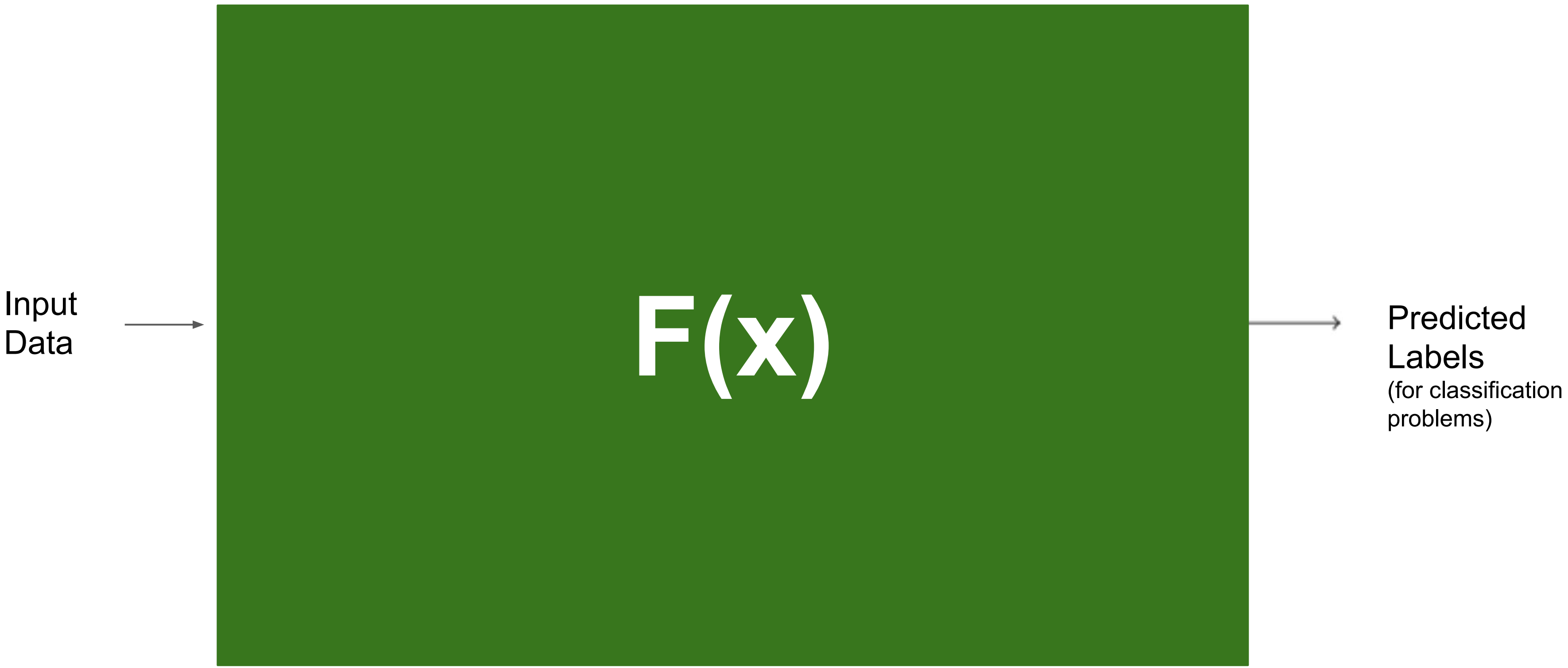
Why can’t we just use correlation?

`Object: Stop Sign`

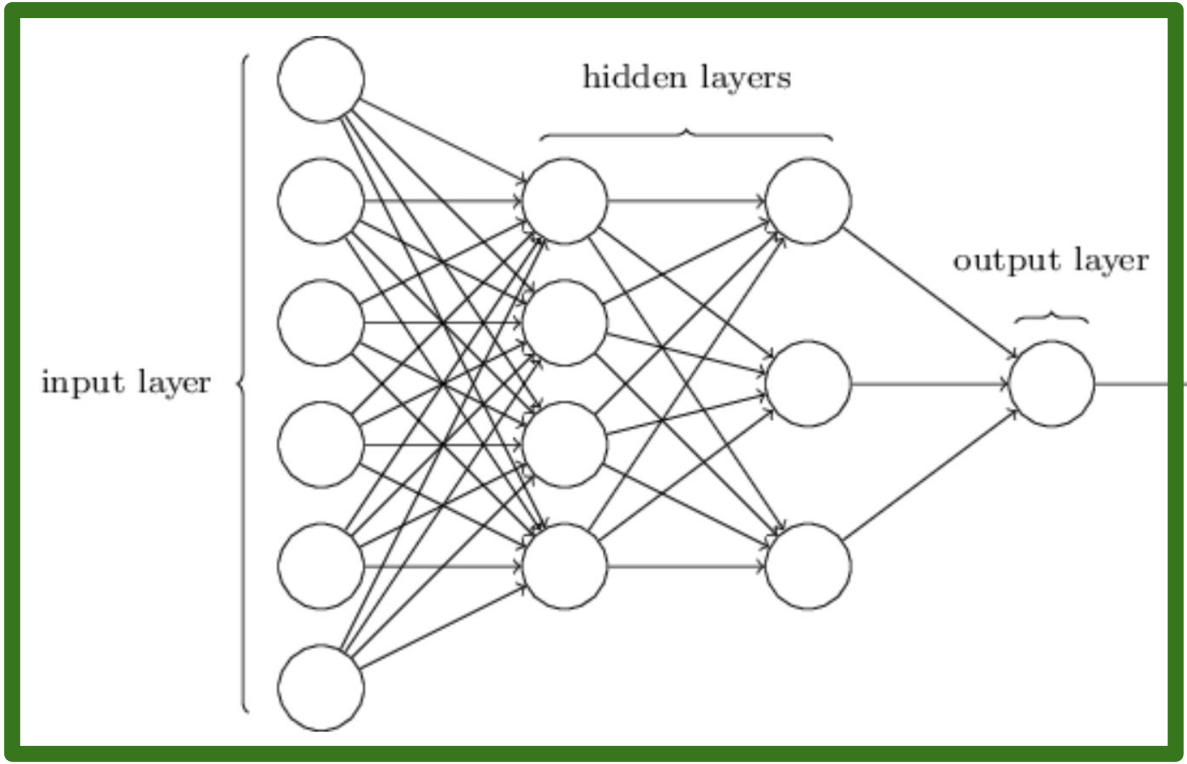
Many configurations (orientation, lighting, context) = too much information to be accurately and robustly contained by a single template filter, or even a group of filters.

Plus, many other possible objects!

Neural networks can be used to solve problems like **classification, clustering, regression, and generation** by **learning weights** as a way of **approximating a function** F which maps input data to the desired output.


Input
Data
F(x)
Predicted
Labels
(for classification
problems)

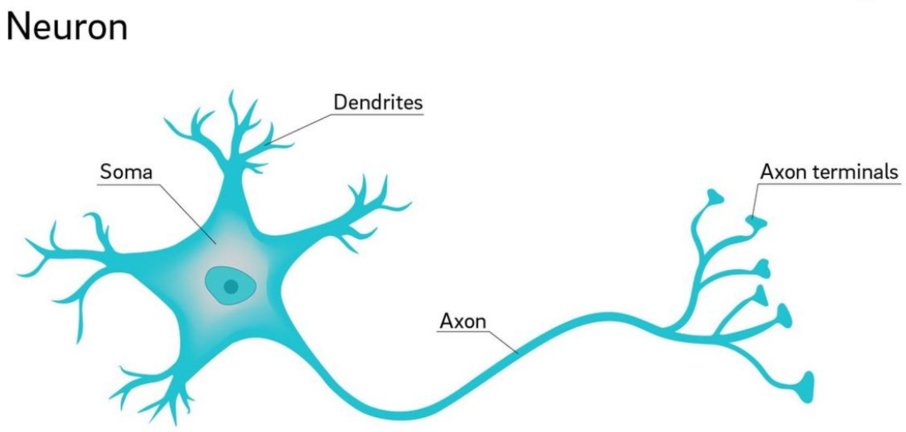
Input Data

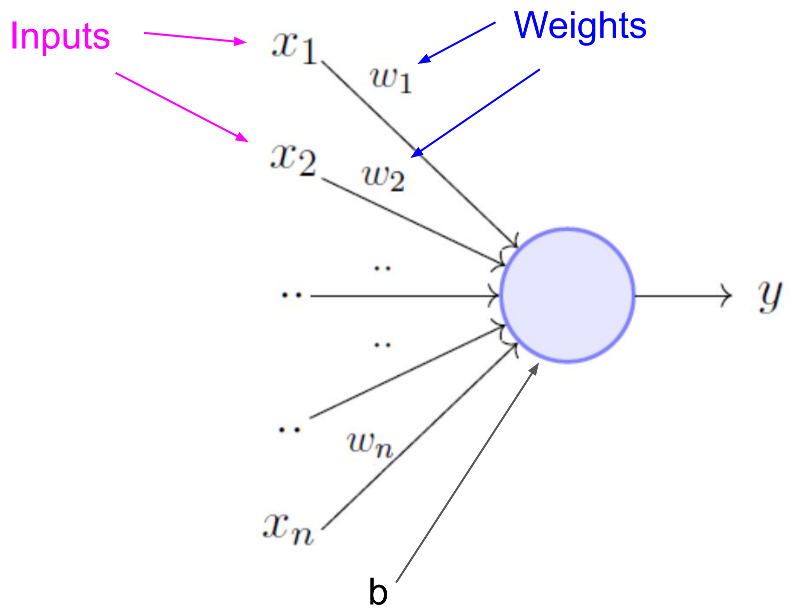


Predicted Labels
(for classification problems)

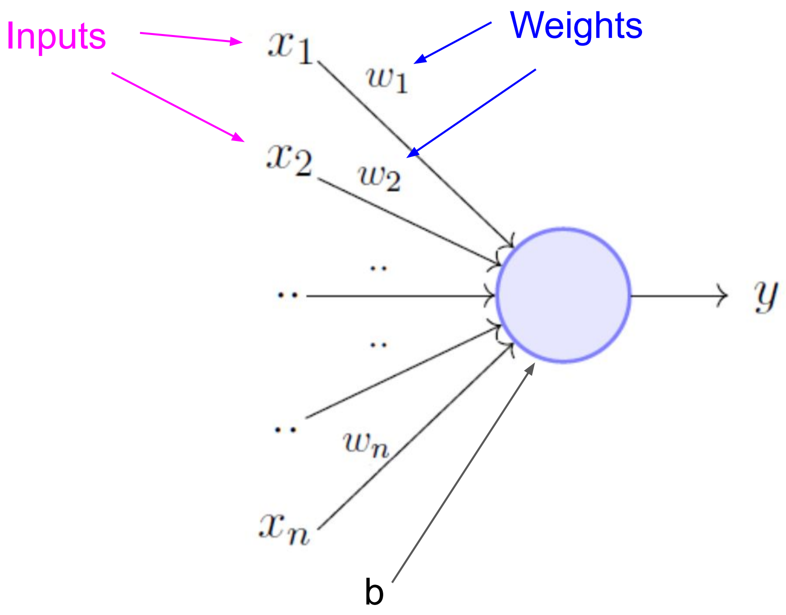
Modeled in Biology

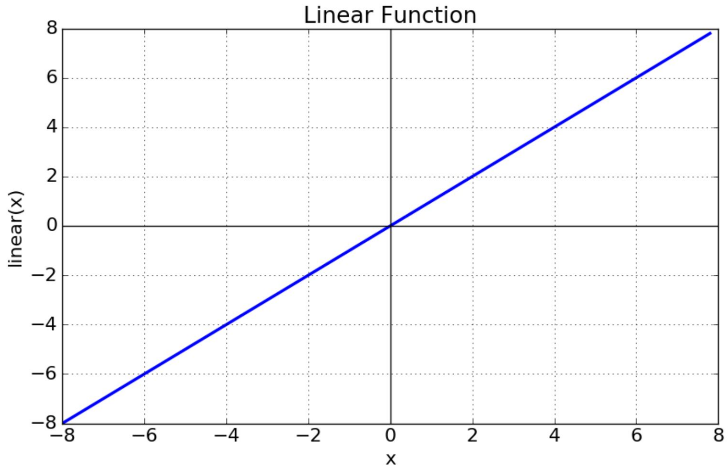
At the junction between two neurons (synapse), an action potential causes a neuron to release a chemical neurotransmitter (after voltage passes action potential threshold).





Inputs
Weights
x_1
x_2
w_1
w_2
w_n
x_n
y
b


Inputs
Weights
x_1
x_2
x_n
w_1
w_2
w_n
y
b


Linear Function
linear(x)
x
8
6
4
2
0
−2
−4
−6
−8

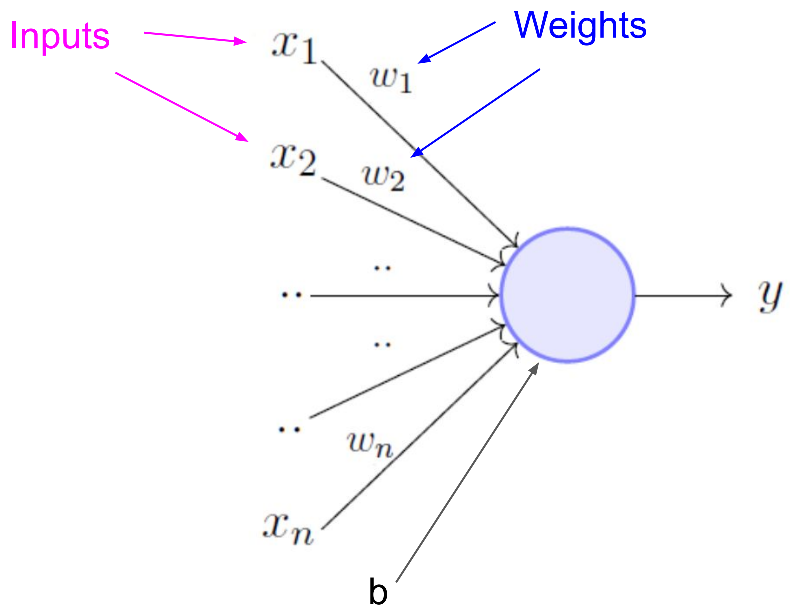
Neural Networks in Code

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x


net = Net()
```

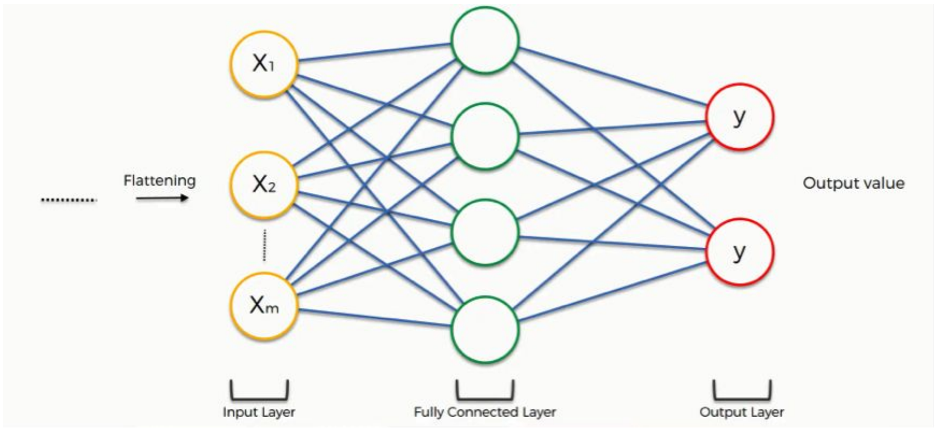
Neural Networks in Code

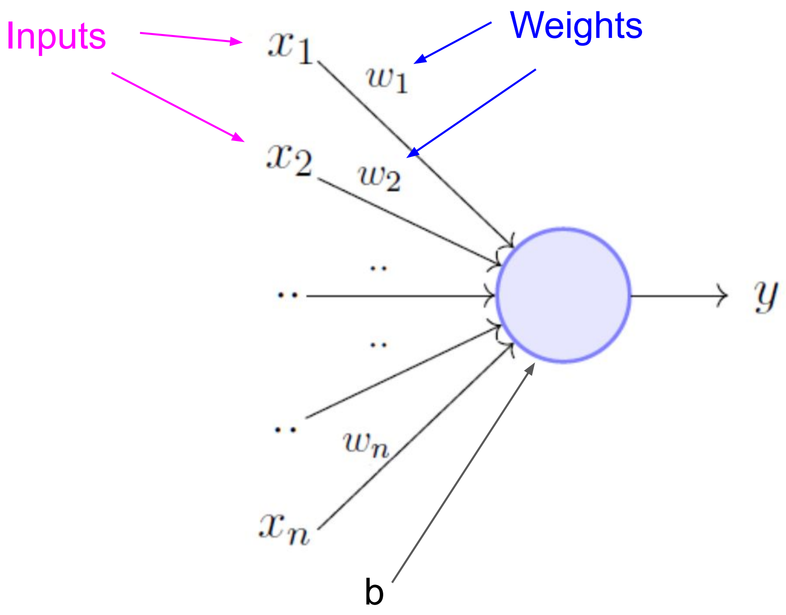
```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x


net = Net()
```

What is a linear layer?

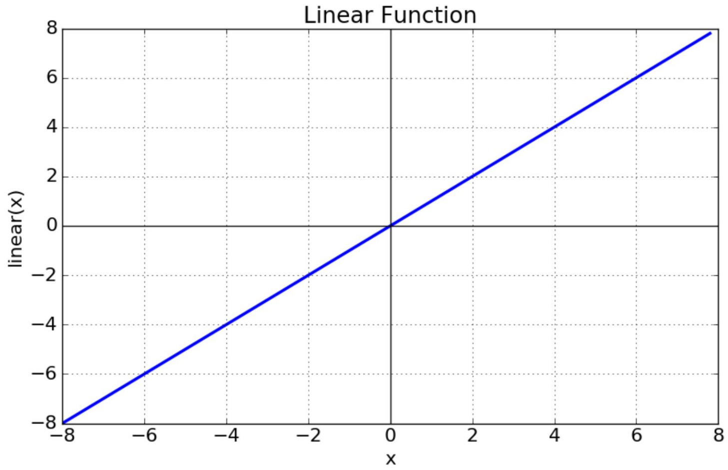


“Linear”, “Fully-Connected”, “Dense” Layer



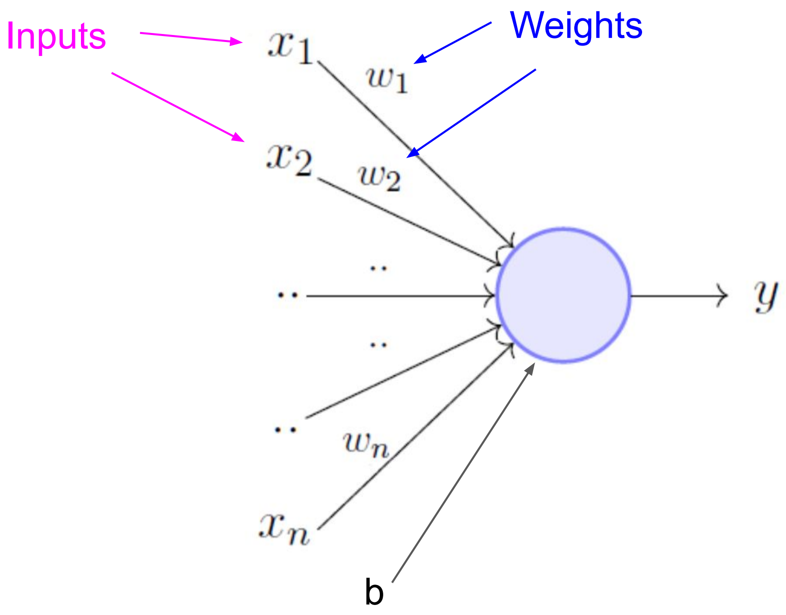


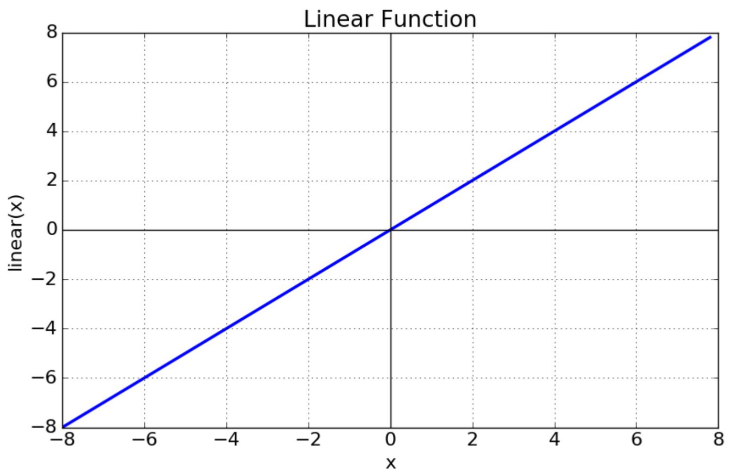
Inputs
Weights
x_1
x_2
x_n
w_1
w_2
w_n
y
b



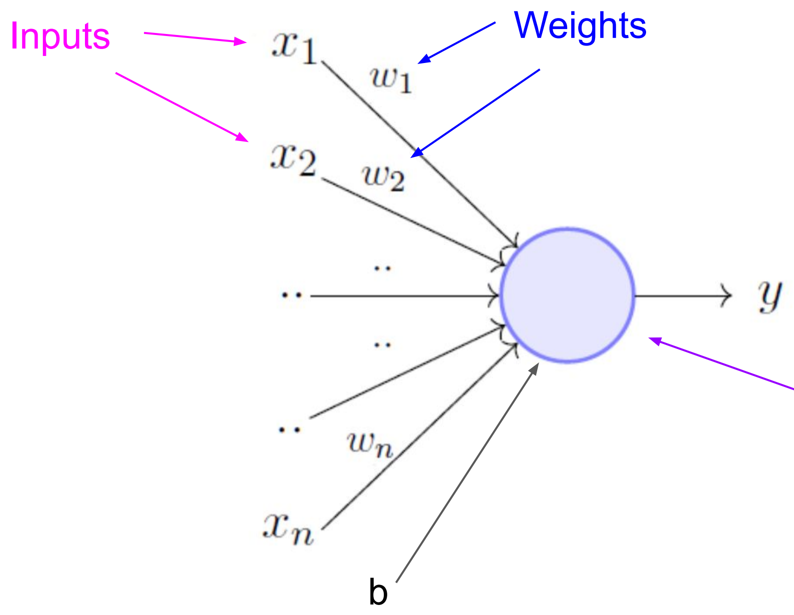
Linear Function
linear(x)
x
8
6
4
2
0
−2
−4
−6
−8

Where is the action potential response??





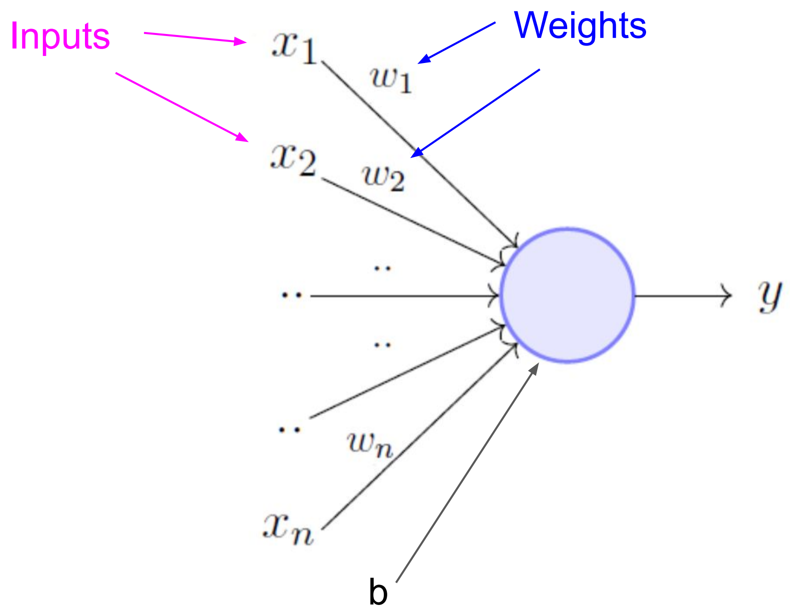
Perceptron

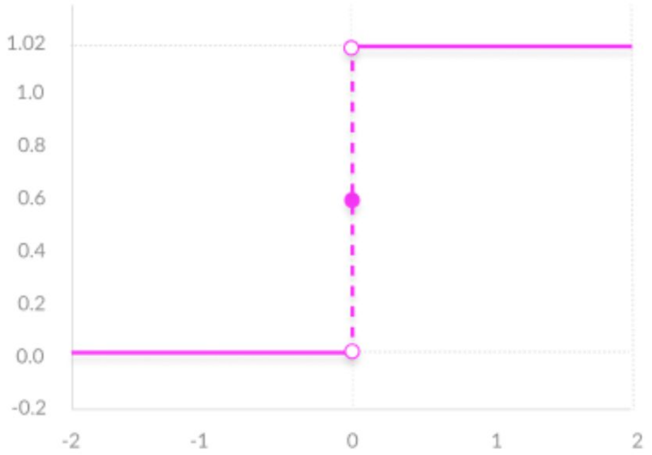


Threshold

$$
\begin{aligned}
y &= 1 \quad if \sum_{i=1}^{n} w_i * x_i \geq \theta \\
&= 0 \quad if \sum_{i=1}^{n} w_i * x_i < \theta
\end{aligned}
$$

Activation Function


Inputs
Weights
x_1
x_2
w_1
w_2
..
..
..
..
w_n
x_n
y
b


1.02
1.0
0.8
0.6
0.4
0.2
0.0
-0.2
-2
-1
0
1
2

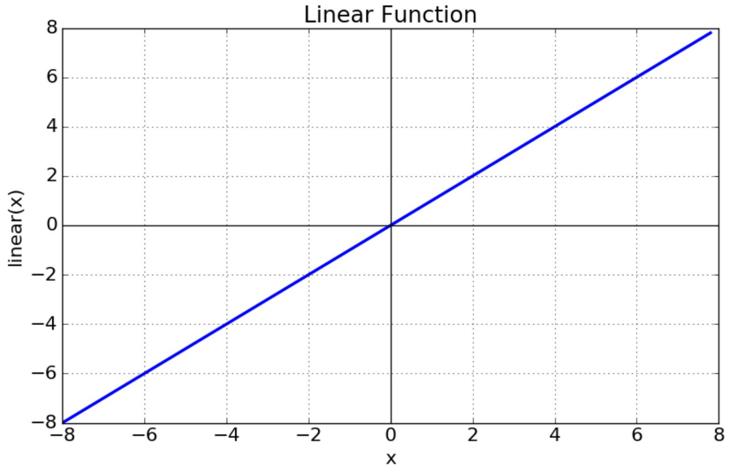
```python
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x


net = Net()
```

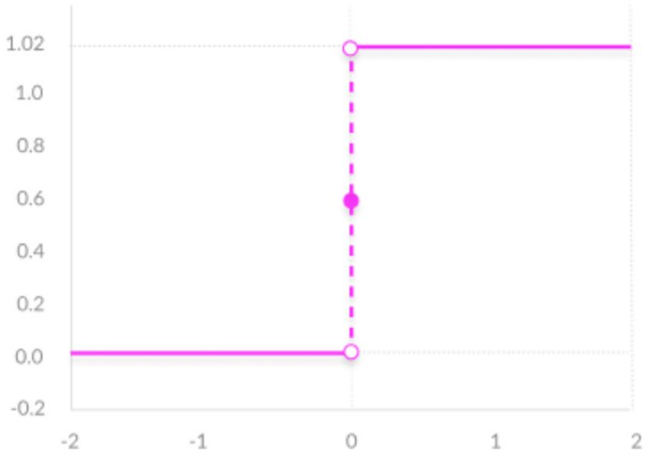
What is a linear layer?

What is a “relu”? This is a type of **Activation Layer.**

Activation Functions

- Activation function allows for non-linearity in the network approximation of function F.
- This is important since most complex data we collect is not linear!



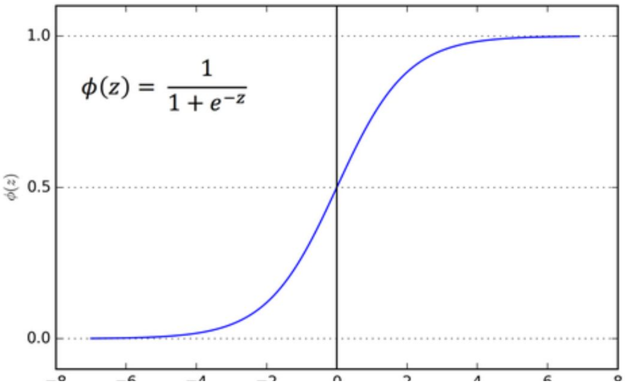


- Pitfall: by reducing input to binary output, we lose information. (Recall Quantization exercises)

Activation Functions

Continuous output (instead of binary) helps provide more information from each neuron.

Sigmoid

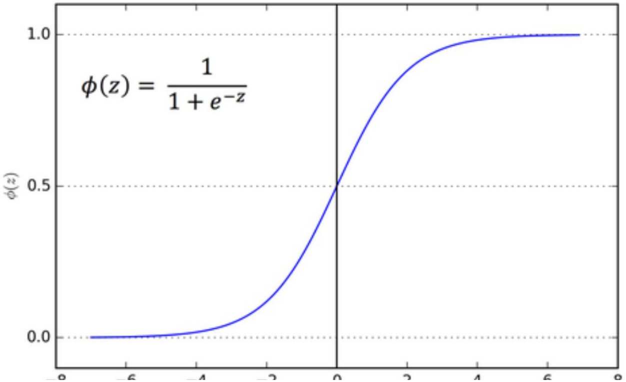


$$\frac{1}{1 + \exp(-\sum_j w_j x_j - b)}$$

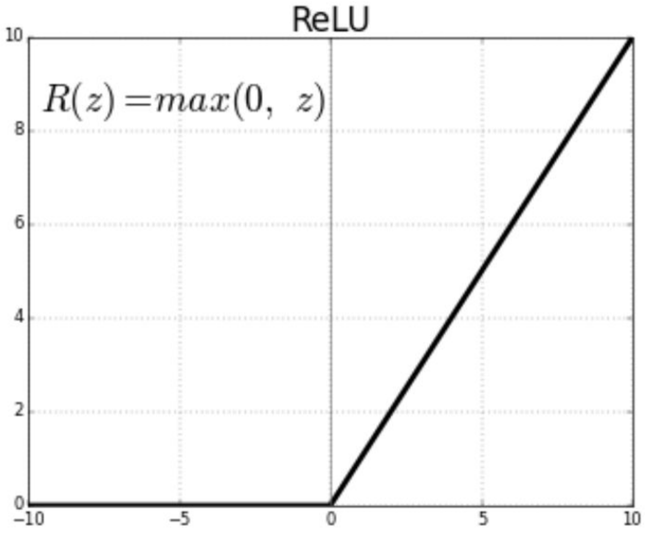
Activation Functions

Continuous output (instead of binary) helps provide more information from each neuron.

Sigmoid



$$\frac{1}{1+\exp(-\sum_j w_j x_j - b)}$$



```python
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x


net = Net()
```

What is a **convolutional layer**?

What is a linear layer?

What is a “relu”? This is a type of **Activation Layer.**

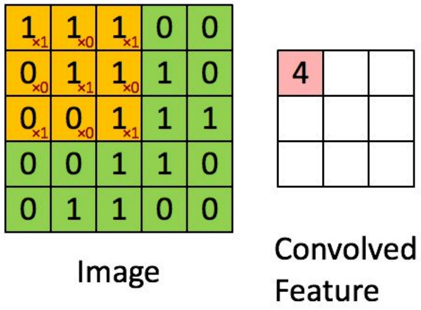
Modeling for 2D Spatial Input (i.e. images)

- Assume there is a spatial relationship between data points (the neighborhood of a point contains useful information).

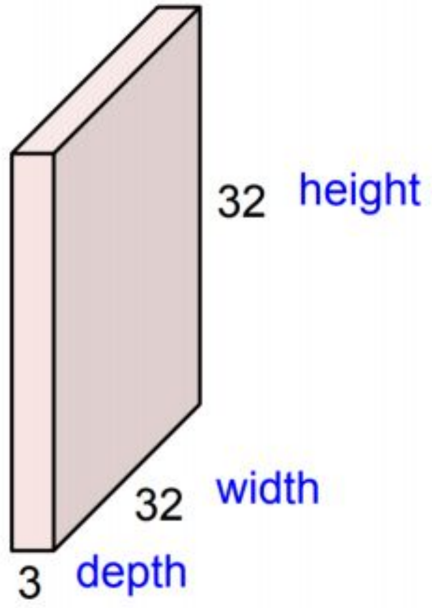
- How can we extract that information in an efficient way?

Terminology

Convolutional Layer

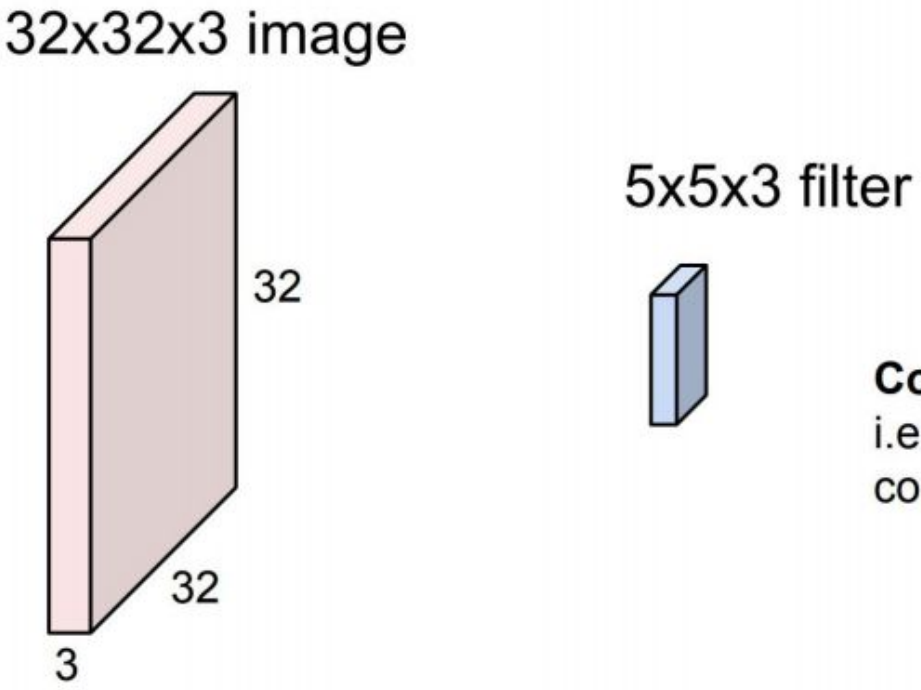


Convolutional Layer



Special Thanks to Nachiket Deo for images, also borrowed from CS231n: Convolutional Neural Networks for Visual Recognition, Fei-Fei Li & Justin Johnson & Serena Yeung

Convolutional Layer



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

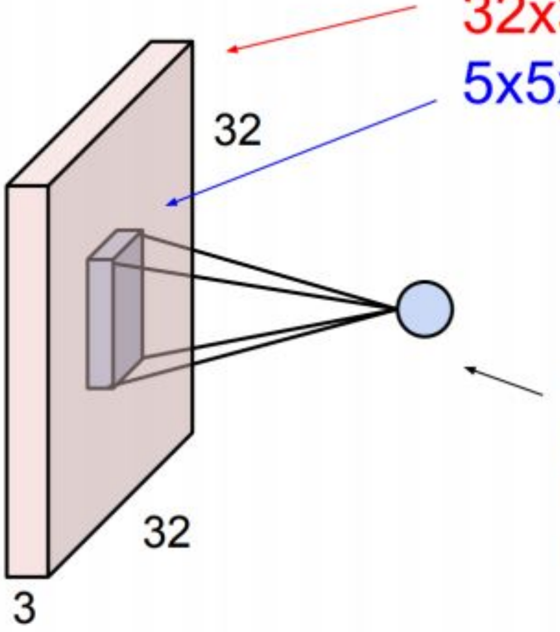
* Ok, technically this is correlation, but “Correlational Neural Network” doesn’t sound as cool.

Special Thanks to Nachiket Deo for images, also borrowed from CS231n: Convolutional Neural Networks for Visual Recognition, Fei-Fei Li & Justin Johnson & Serena Yeung

Convolutional Layer

32x32x3 image

5x5x3 filter w

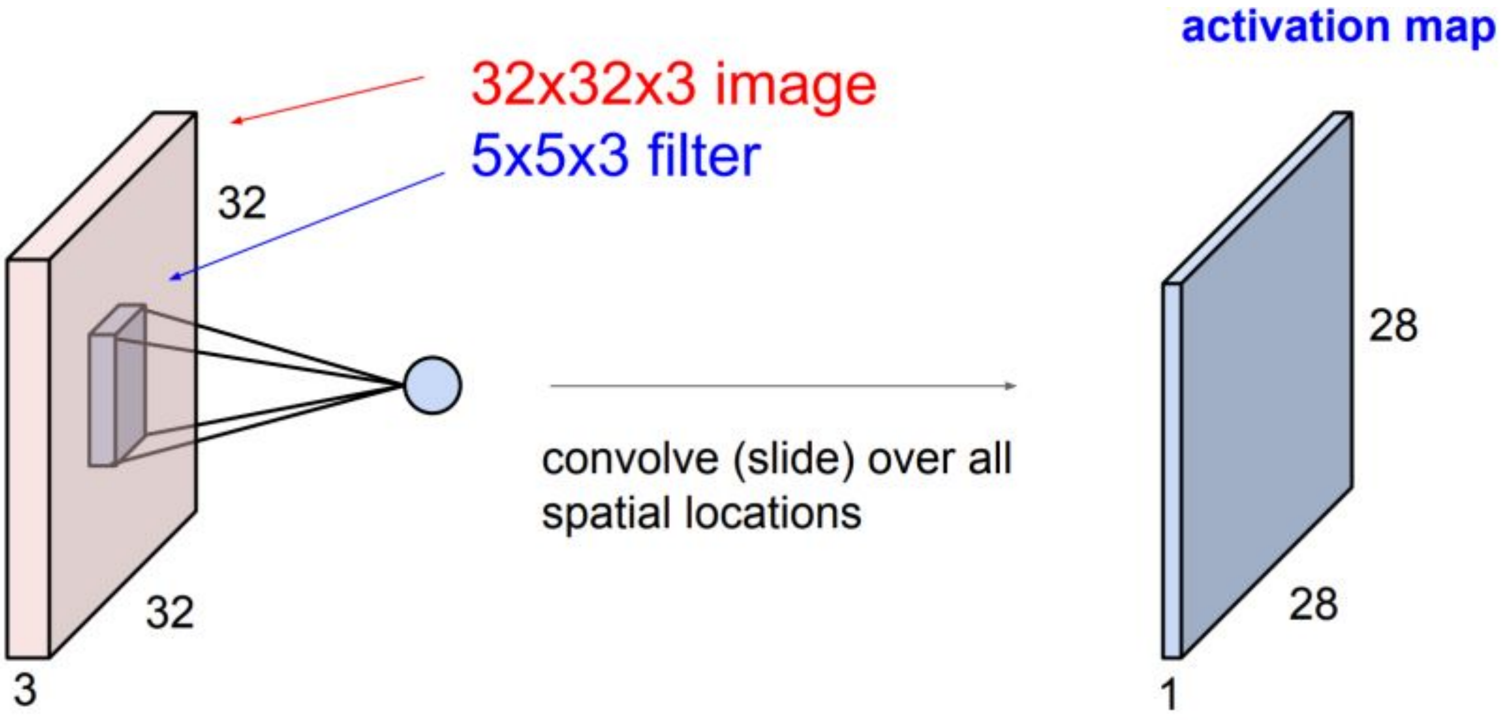


1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

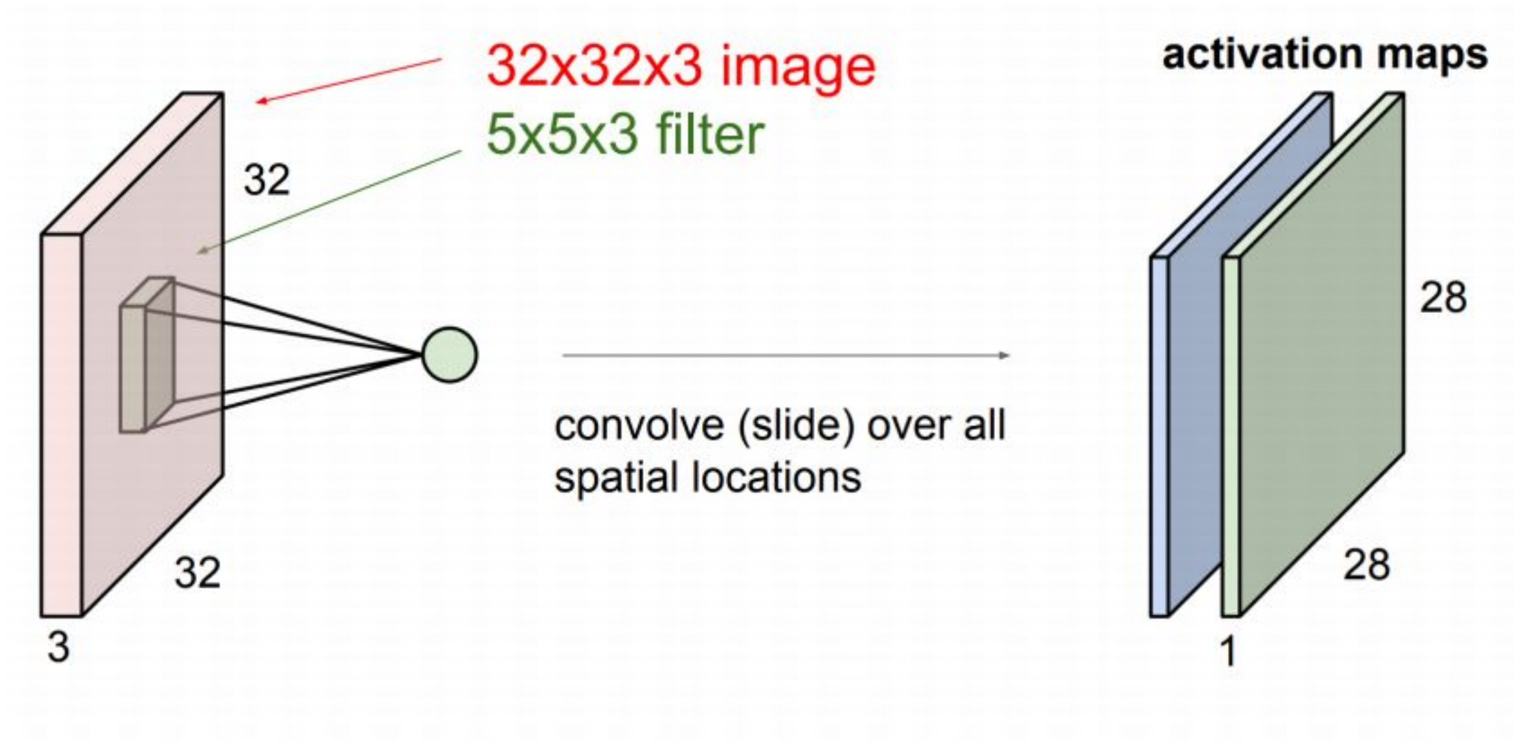
Special Thanks to Nachiket Deo for images, also borrowed from CS231n: Convolutional Neural Networks for Visual Recognition, Fei-Fei Li & Justin Johnson & Serena Yeung

Convolutional Layer



Special Thanks to Nachiket Deo for images, also borrowed from CS231n: Convolutional Neural Networks for Visual Recognition, Fei-Fei Li & Justin Johnson & Serena Yeung

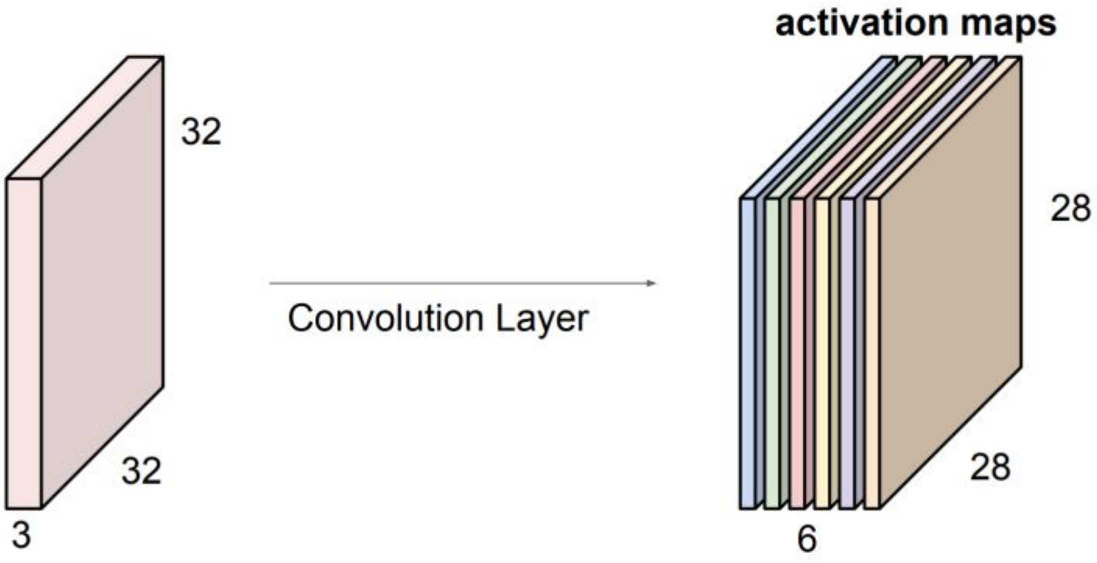
Convolutional Layer



Special Thanks to Nachiket Deo for images, also borrowed from CS231n: Convolutional Neural Networks for Visual Recognition, Fei-Fei Li & Justin Johnson & Serena Yeung

Convolutional 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!

Special Thanks to Nachiket Deo for images, also borrowed from CS231n: Convolutional Neural Networks for Visual Recognition, Fei-Fei Li & Justin Johnson & Serena Yeung

```python
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x


net = Net()
```

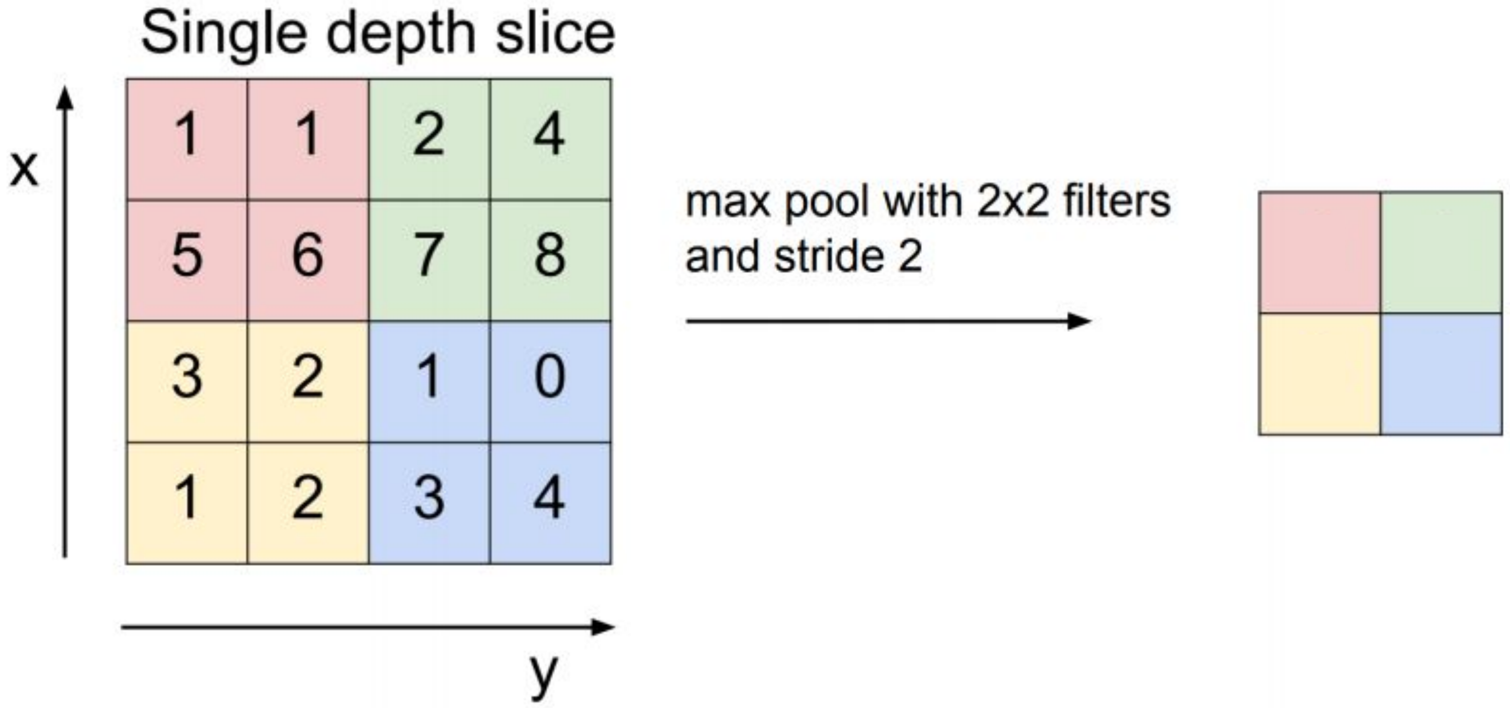
What is a **convolutional layer**?

What is max pool? This is a type of **Pooling Layer**.

What is a linear layer?

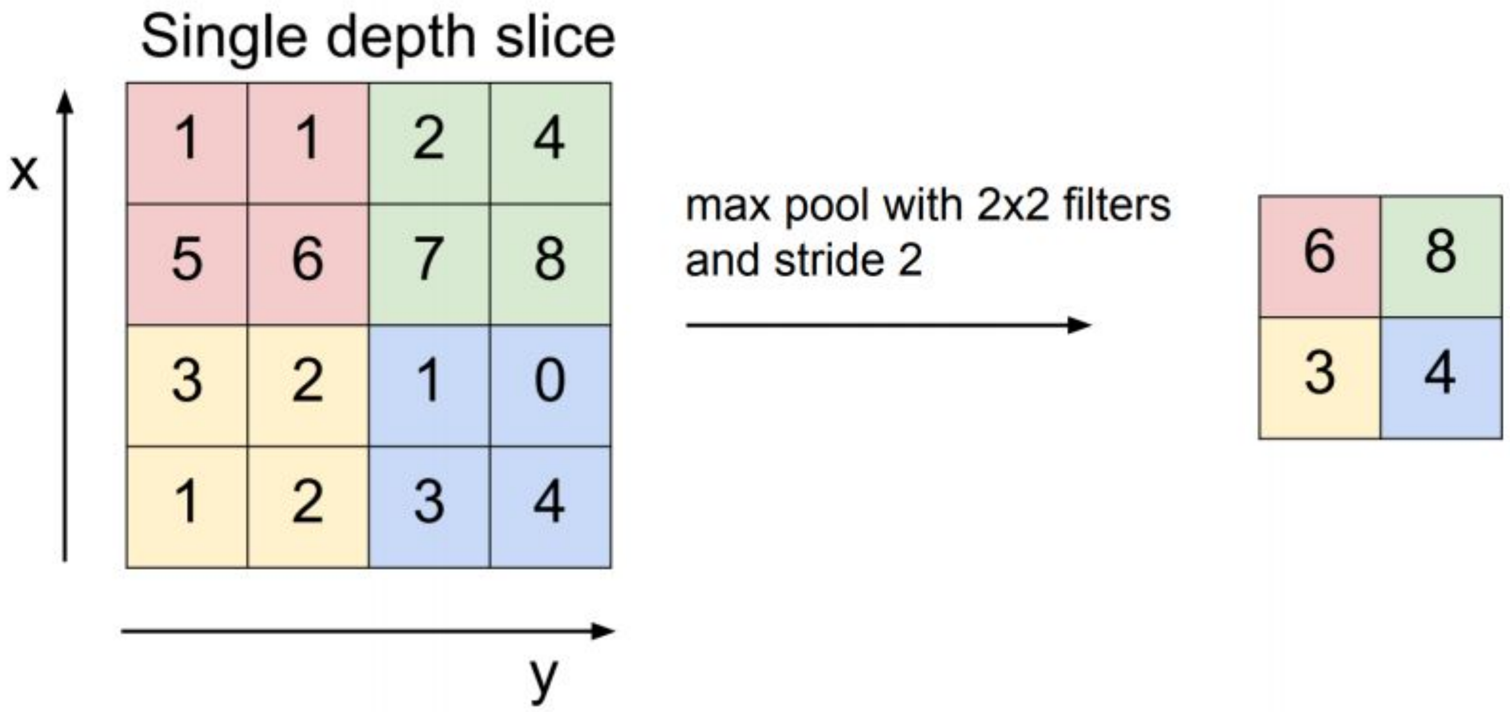
What is a “relu”? This is a type of **Activation Layer.**

Max Pooling



Special Thanks to Nachiket Deo for images, also borrowed from CS231n: Convolutional Neural Networks for Visual Recognition, Fei-Fei Li & Justin Johnson & Serena Yeung

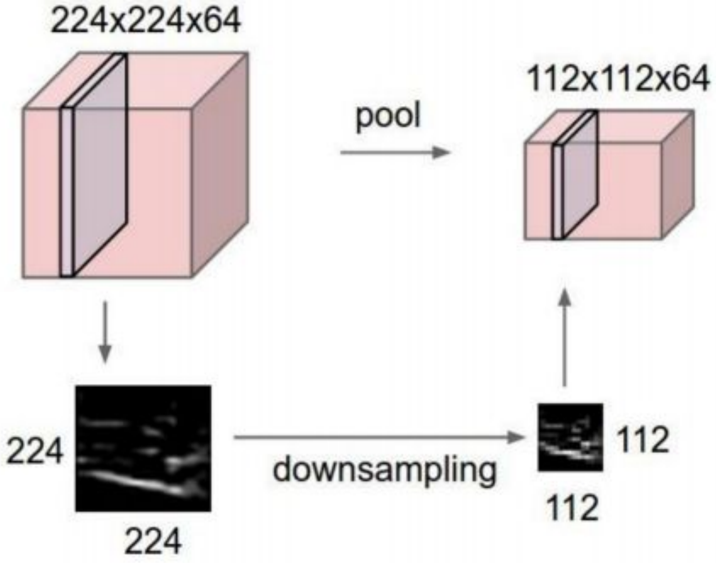
Max Pooling



Special Thanks to Nachiket Deo for images, also borrowed from CS231n: Convolutional Neural Networks for Visual Recognition, Fei-Fei Li & Justin Johnson & Serena Yeung

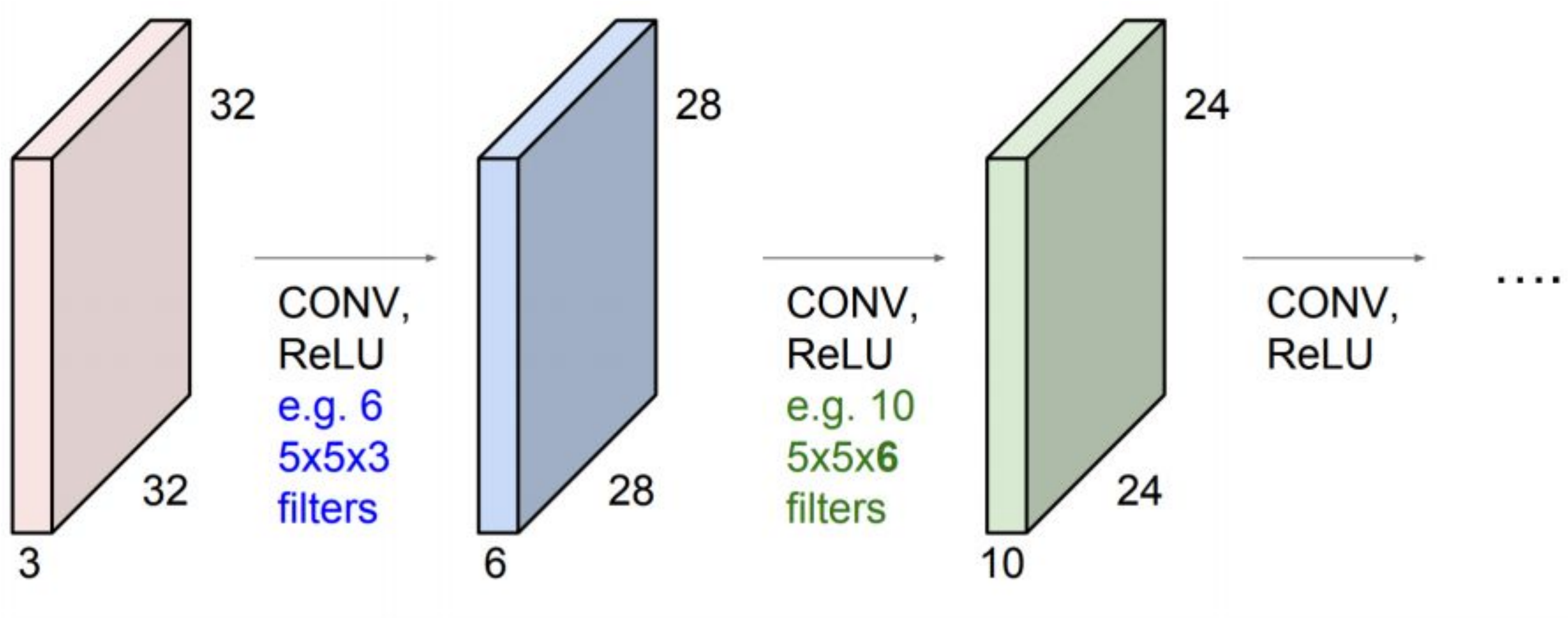
Max Pooling

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



Special Thanks to Nachiket Deo for images, also borrowed from CS231n: Convolutional Neural Networks for Visual Recognition, Fei-Fei Li & Justin Johnson & Serena Yeung

Putting it all together

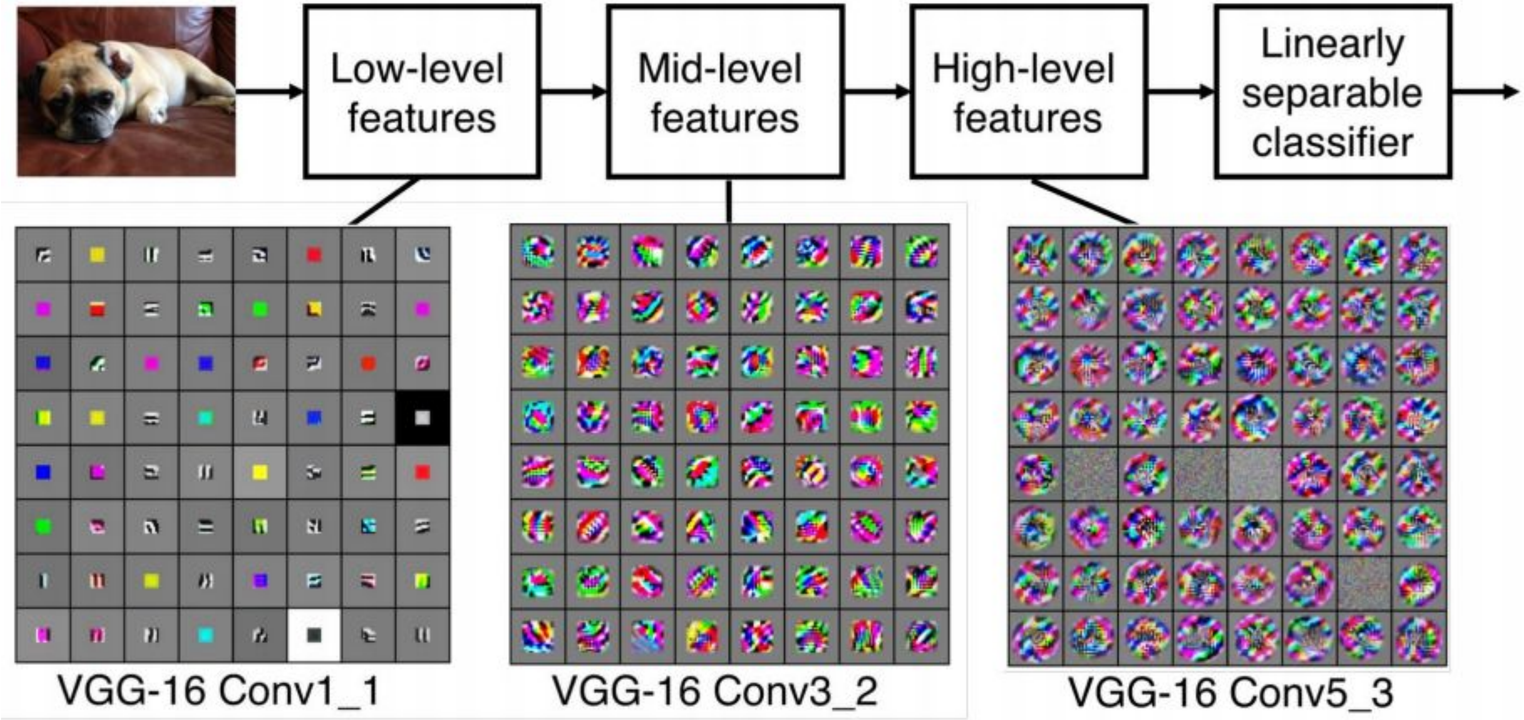


Special Thanks to Nachiket Deo for images, also borrowed from CS231n: Convolutional Neural Networks for Visual Recognition, Fei-Fei Li & Justin Johnson & Serena Yeung

[Zeiler and Fergus 2013]

Visualization of VGG-16 by Lane McIntosh. VGG-16 architecture from [Simonyan and Zisserman 2014].

Putting it all together



Special Thanks to Nachiket Deo for images, also borrowed from CS231n: Convolutional Neural Networks for Visual Recognition, Fei-Fei Li & Justin Johnson & Serena Yeung

```python
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x


net = Net()
```

What is a **convolutional layer**?

What is max pool? This is a type of **Pooling Layer**.

What is a linear layer?

What is a “relu”? This is a type of **Activation Layer.**

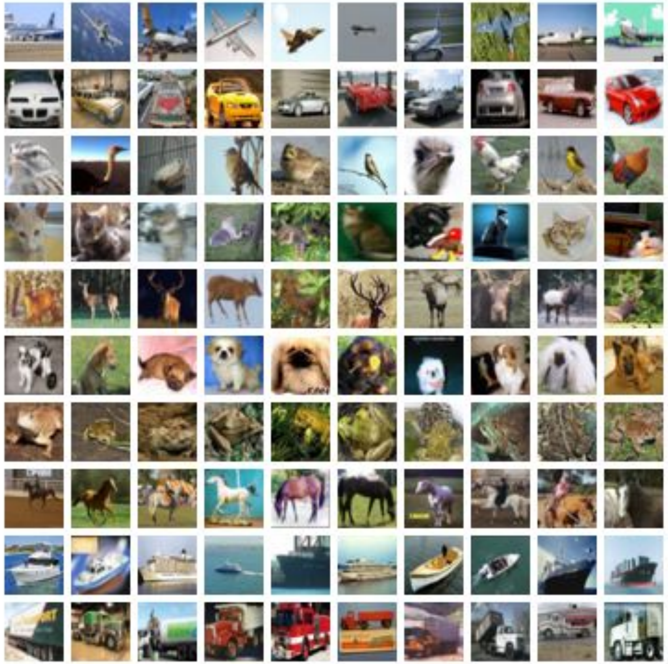
What does the output (x) look like?

We wanted something like “Stop Sign”...

How do we turn a name into a vector?

“One-Hot” Encoding

First, predefine:



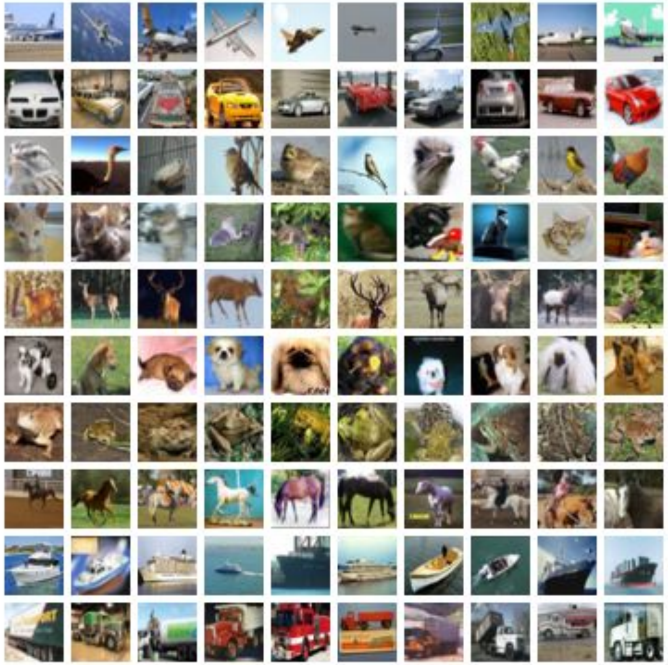
airplane	0
automobile	1
bird	2
cat	3
deer	4
dog	5
frog	6
horse	7
ship	8
truck	9

How do we turn a name into a vector?

“One-Hot” Encoding

First, predefine:

airplane	0
automobile	1
bird	2
cat	3
deer	4
dog	5
frog	6
horse	7
ship	8
truck	9



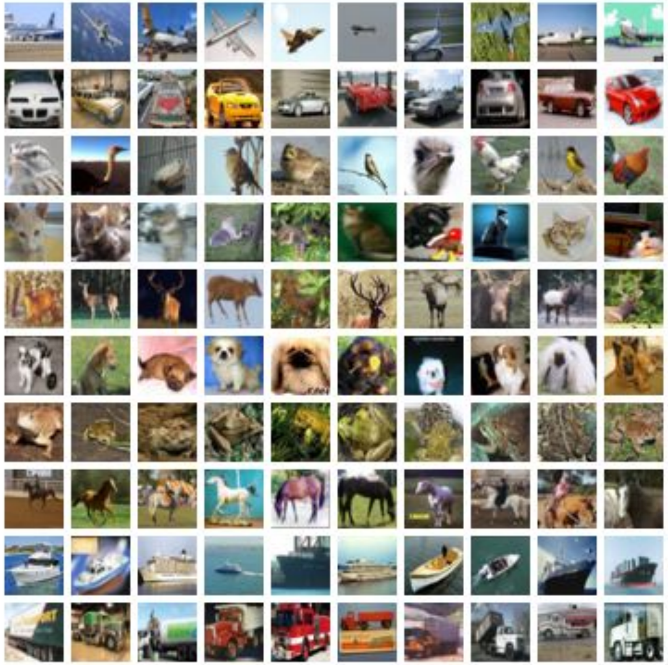
Dog → Class 5 →
$$\begin{pmatrix}0\\0\\0\\0\\0\\1\\0\\0\\0\\0\end{pmatrix}$$

Cat → Class 3 →
$$\begin{pmatrix}0\\0\\0\\1\\0\\0\\0\\0\\0\\0\end{pmatrix}$$

How do we turn a name into a vector?

“One-Hot” Encoding

First, predefine:



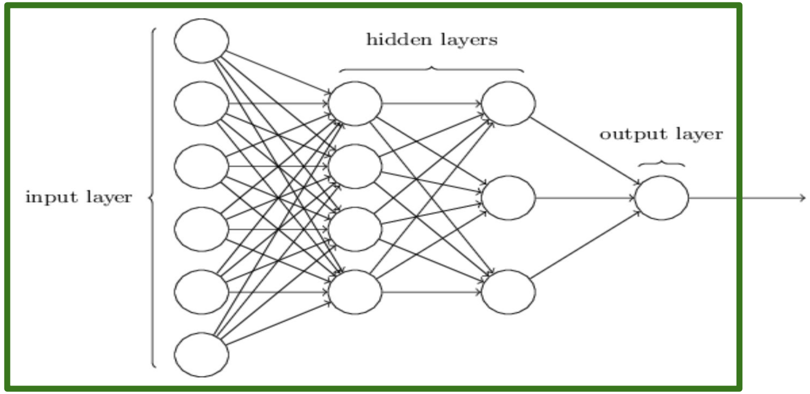
airplane	0
automobile	1
bird	2
cat	3
deer	4
dog	5
frog	6
horse	7
ship	8
truck	9

Dog → Class 5 → $\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$

Cat → Class 3 → $\begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$

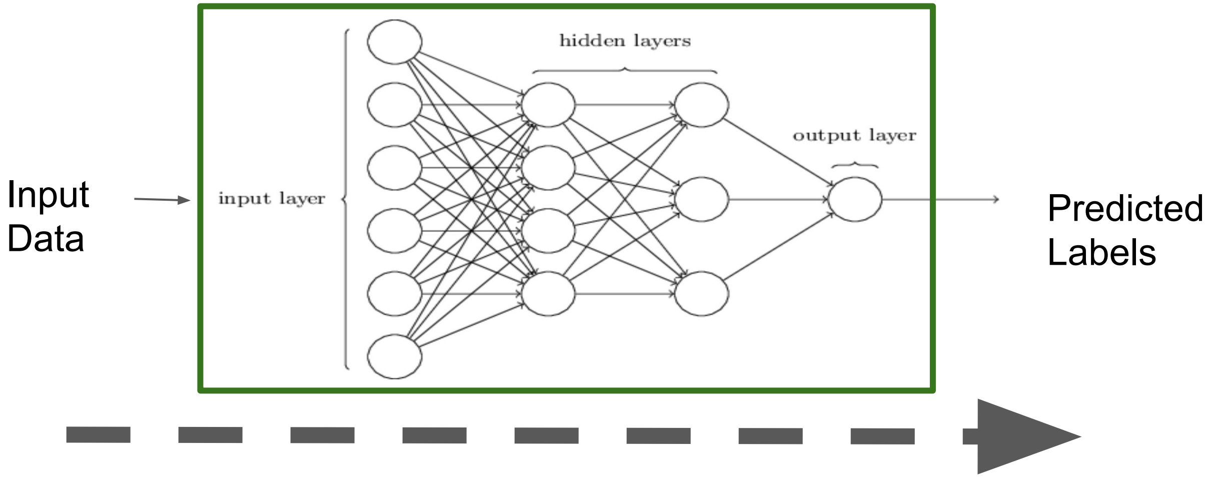
Forward Propagation

Input
Data



Predicted
Labels

Forward Propagation



- If we initialize all weights (w) randomly, there will be a difference between the predicted output (y_{pred}) and true, expected class (y).
- We quantify this difference via a **loss function** (L).

Loss Functions

Mean Squared-Error

$$\frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y_i})^2$$

$* \, n$ is the number of data points

$* \, Y_i$ represents observed values

$* \, \hat{Y}_i$ represents predicted values

Loss Functions

Mean Squared-Error

$$\frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

$*\, n$ is the number of data points

$*\, Y_i$ represents observed values

$*\, \hat{Y}_i$ represents predicted values

Cross-Entropy

$$-\sum_{c=1}^{M} y_{o,c} \log(p_{o,c})$$

- M - number of classes (dog, cat, fish)
- log - the natural log
- y - binary indicator (0 or 1) if class label c is the correct classification for observation o
- p - predicted probability observation o is of class c

Loss Functions

Cross-Entropy

$$-\sum_{c=1}^{M} y_{o,c} \log(p_{o,c})$$

- M - number of classes (dog, cat, fish)
- log - the natural log
- y - binary indicator (0 or 1) if class label c is the correct classification for observation o
- p - predicted probability observation o is of class c

Loss Functions

Dog: 0.00

Cross-Entropy

$$-\sum_{c=1}^{M} y_{o,c} \log(p_{o,c})$$

- M - number of classes (dog, cat, fish)
- log - the natural log
- y - binary indicator (0 or 1) if class label c is the correct classification for observation o
- p - predicted probability observation o is of class c

Loss Functions

Dog: 0.00
Cat: 0.02

Cross-Entropy

$$-\sum_{c=1}^{M} y_{o,c} \log(p_{o,c})$$

- M - number of classes (dog, cat, fish)
- log - the natural log
- y - binary indicator (0 or 1) if class label c is the correct classification for observation o
- p - predicted probability observation o is of class c

Loss Functions

Dog: 0.00
Cat: 0.02
Fish: 0.00

Cross-Entropy

$$-\sum_{c=1}^{M} y_{o,c} \log(p_{o,c})$$

- M - number of classes (dog, cat, fish)
- log - the natural log
- y - binary indicator (0 or 1) if class label c is the correct classification for observation o
- p - predicted probability observation o is of class c

Loss Functions

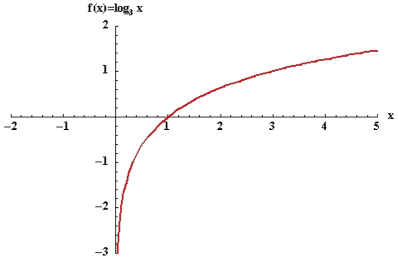
Dog: 0.00
Cat: 0.02
Fish: 0.00
Guacamole: 0.98

Cross-Entropy

$$-\sum_{c=1}^{M} y_{o,c} \log(p_{o,c})$$

- M - number of classes (dog, cat, fish)
- log - the natural log
- y - binary indicator (0 or 1) if class label c is the correct classification for observation o
- p - predicted probability observation o is of class c

Loss Functions



Dog: 0.00
Cat: 0.02
Fish: 0.00
Guacamole: 0.98

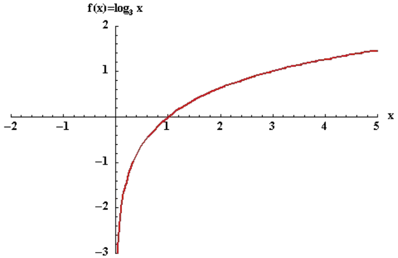
y	p	-ylog(p)
0	0	0
1	0.02	Large, positive
0	0	0
0	0.98	0

Cross-Entropy

$$-\sum_{c=1}^{M} y_{o,c} \log(p_{o,c})$$

- M - number of classes (dog, cat, fish)
- log - the natural log
- y - binary indicator (0 or 1) if class label c is the correct classification for observation o
- p - predicted probability observation o is of class c

Loss Functions



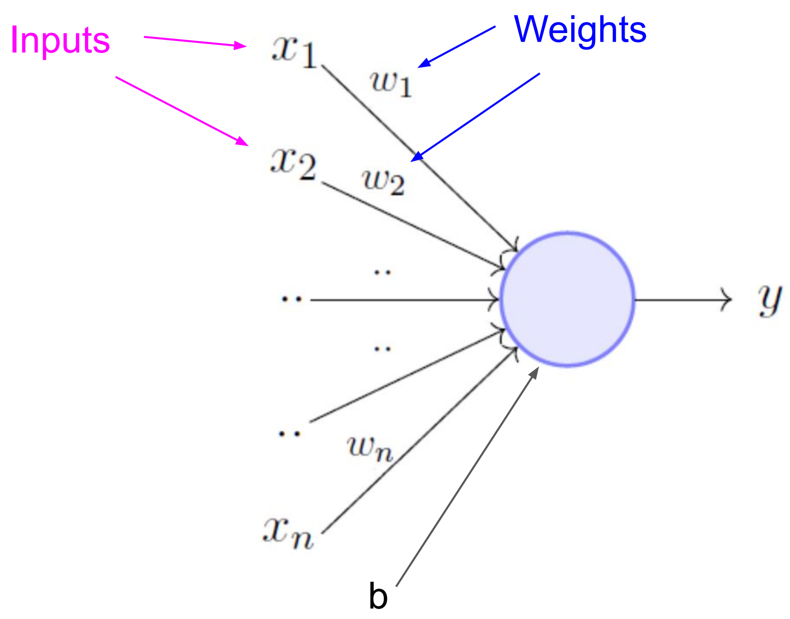
Dog: 0.00
Cat: 0.02
Fish: 0.00
Guacamole: 0.98

y	p	-ylog(p)
0	0.02	0
1	0.98	Near 0
0	0	0
0	0	0

Cross-Entropy

$$-\sum_{c=1}^{M} y_{o,c}\log(p_{o,c})$$

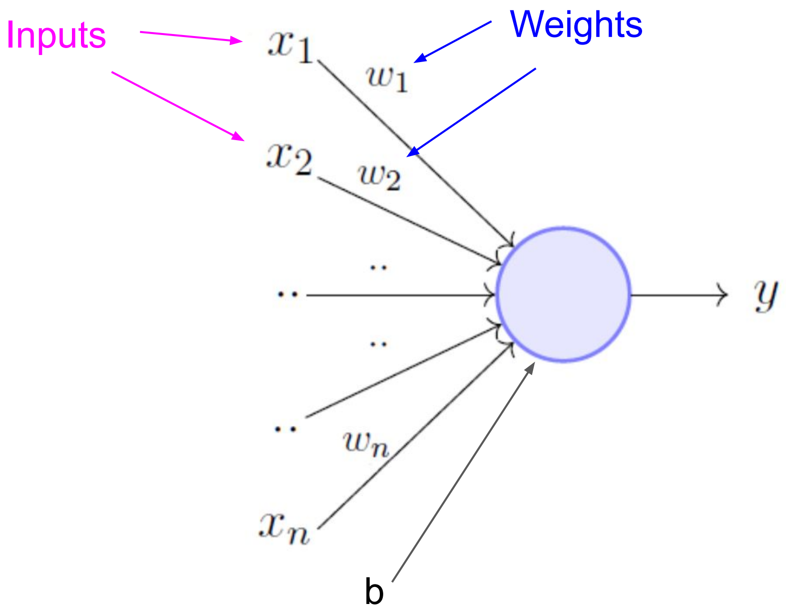
- M - number of classes (dog, cat, fish)
- log - the natural log
- y - binary indicator (0 or 1) if class label c is the correct classification for observation o
- p - predicted probability observation o is of class c



How do we choose these weights?

Back Propagation (Optimization)

- Ideally, L = 0 (predicted output matches correct output)



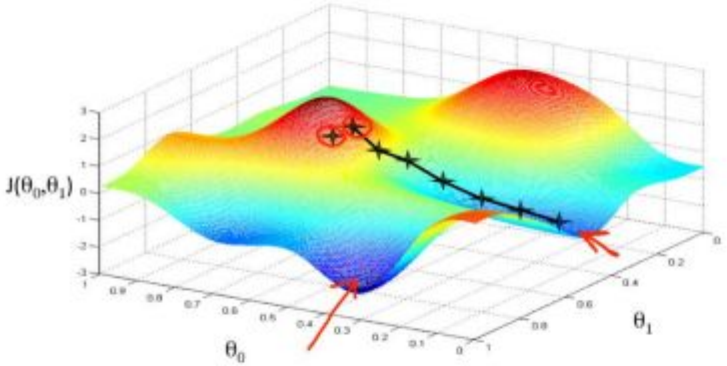
Idea: reduce loss by adjusting the weights!

Back Propagation (Optimization)

- Ideally, L = 0 (predicted output matches correct output)
- Is this always possible?

Training objective:

$$W^* = \text{argmin}_W \Sigma_i L(f_W(X^{(i)}), Y^{(i)})$$



Back Propagation (Optimization)

- We can reduce the loss by changing the values of the weights.
- How severely should we change the weight?

Learning Rate

$$\boldsymbol{W}^{+} = \boldsymbol{W} - \eta \nabla \boldsymbol{C}$$

$$\nabla \boldsymbol{C} = \begin{bmatrix} \frac{\partial C}{\partial w_1} \\ \frac{\partial C}{\partial w_2} \\ \vdots \\ \frac{\partial C}{\partial w_n} \end{bmatrix}$$

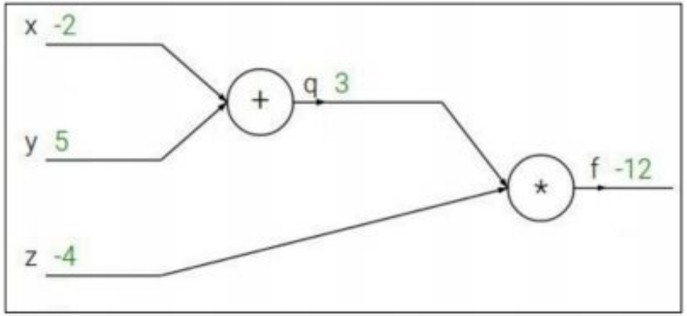
*Note, these equations refer to ‘L’ (loss) as ‘C’ (cost).

Back Propagation (Optimization)

Backpropagation: a simple example

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4



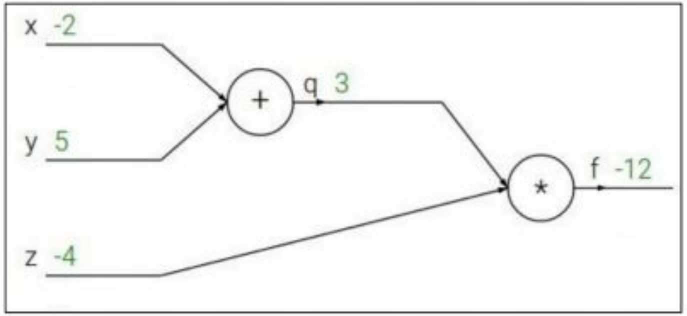
Back Propagation (Optimization)

Backpropagation: a simple example

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = \quad , \frac{\partial q}{\partial y} =$$



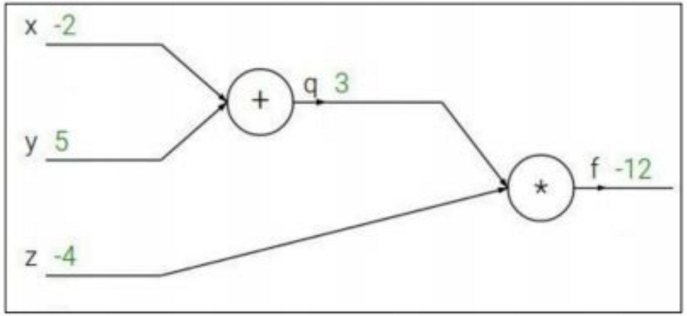
Back Propagation (Optimization)

Backpropagation: a simple example

$$f(x,y,z) = (x+y)z$$

e.g. x = -2, y = 5, z = -4

$$\boxed{q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1}$$



Back Propagation (Optimization)

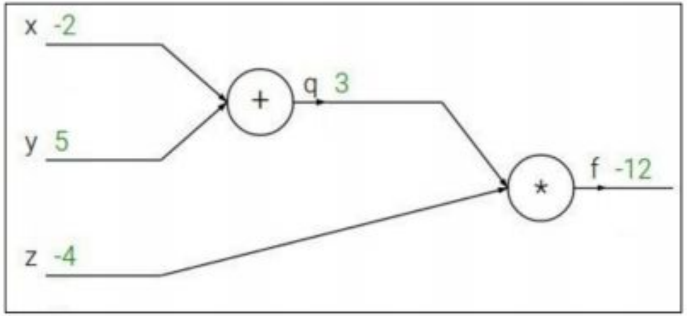
Backpropagation: a simple example

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = \quad , \frac{\partial f}{\partial z} =$$



Back Propagation (Optimization)

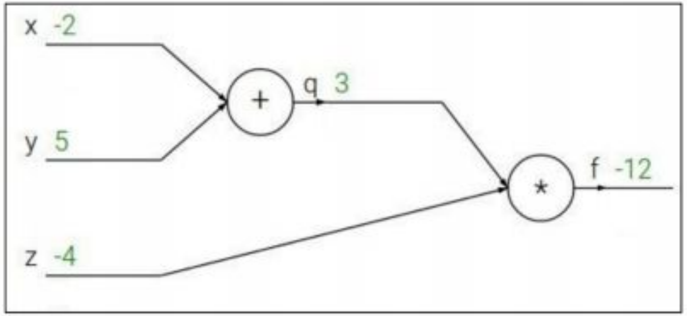
Backpropagation: a simple example

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$



Back Propagation (Optimization)

Backpropagation: a simple example

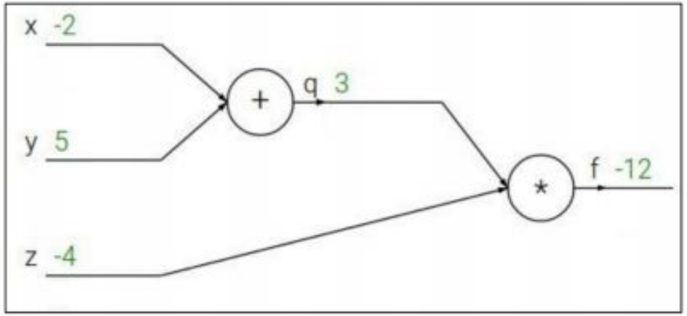
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Back Propagation (Optimization)

Backpropagation: a simple example

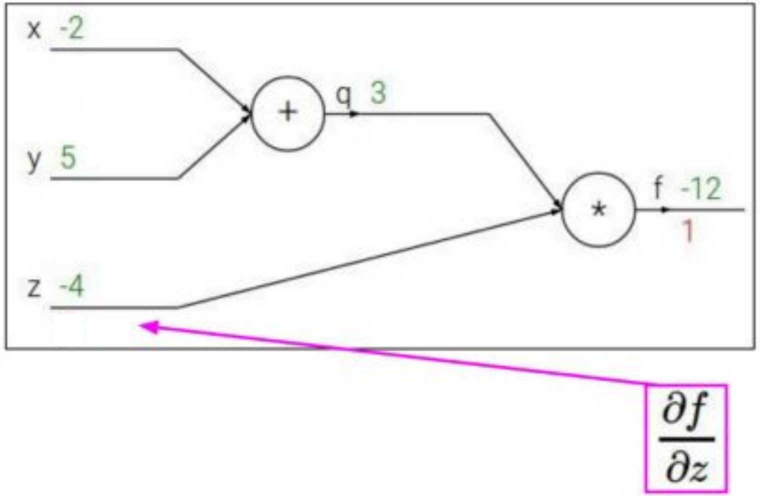
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Back Propagation (Optimization)

Backpropagation: a simple example

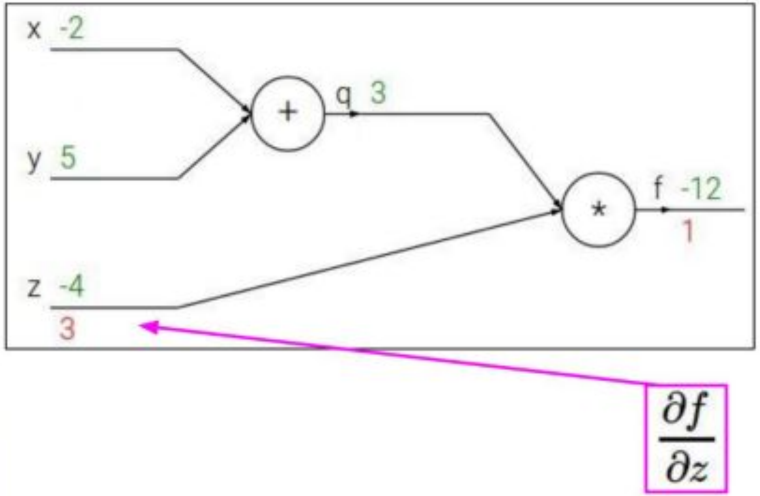
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Back Propagation (Optimization)

Backpropagation: a simple example

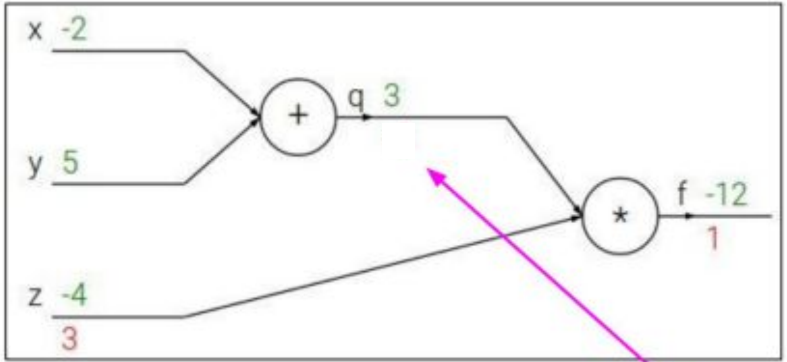
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial q}$$

Back Propagation (Optimization)

Backpropagation: a simple example

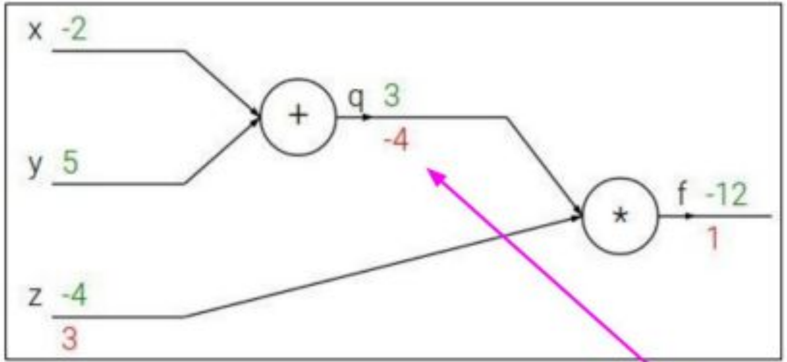
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial q}$$

Back Propagation (Optimization)

Backpropagation: a simple example

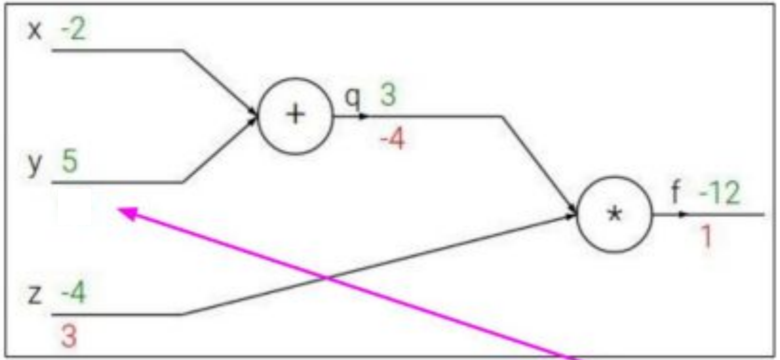
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial y}$$

Back Propagation (Optimization)

Backpropagation: a simple example

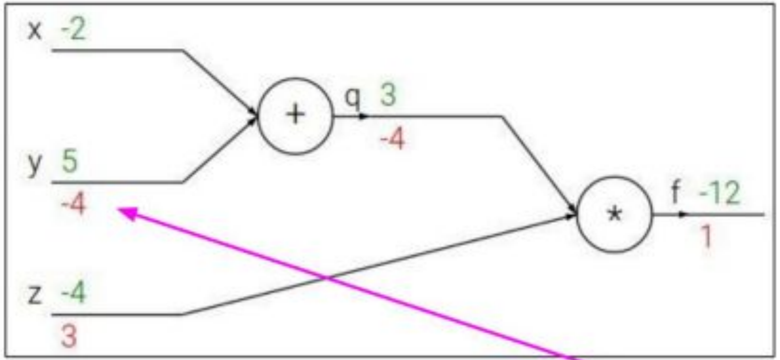
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial y}$$

Chain rule:

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial y}$$

Back Propagation (Optimization)

Backpropagation: a simple example

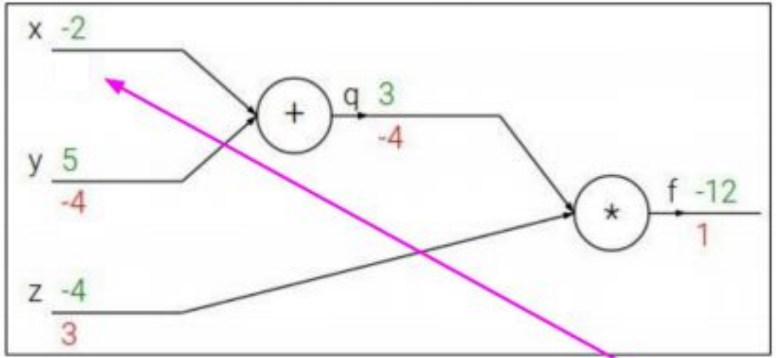
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial x}$$

Chain rule:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}$$

Back Propagation (Optimization)

Backpropagation: a simple example

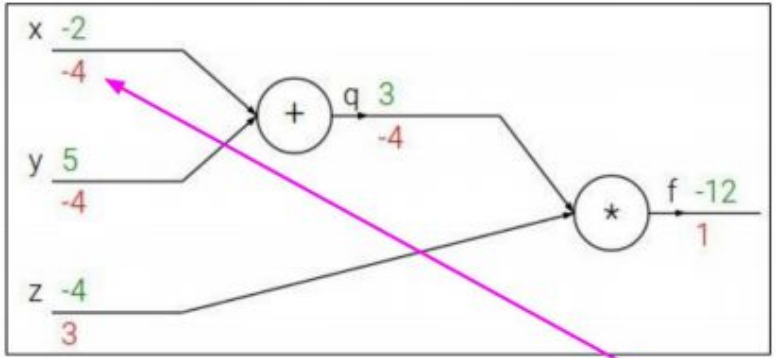
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$\boxed{q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1}$$

$$\boxed{f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q}$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\boxed{\frac{\partial f}{\partial x}}$$

Chain rule:

$$\boxed{\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}}$$

Back Propagation (Optimization)

- We can reduce the loss by changing the values of the weights.
- How severely should we change the weight?

Learning Rate

$$W^{+} = W - \eta \nabla C$$

$$\nabla C = \begin{bmatrix} \frac{\partial C}{\partial w_1} \\ \frac{\partial C}{\partial w_2} \\ \vdots \\ \frac{\partial C}{\partial w_n} \end{bmatrix}$$

*Note, these equations refer to 'L' (loss) as 'C' (cost).

Back Propagation (Optimization)

From Professor Trivedi:

UCSD interesting Trivia:

Back propagation was “invented” in 1986…

David E. Rumelhart, Geoffrey E. Hinton und Ronald J. Williams. *Learning representations by back-propagating errors*., Nature (London) 323, S. 533-536.

All three authors were working at UCSD in 1986!

Back Propagation (Optimization)

- We can reduce the loss by changing the values of the weights.
- How severely should we change the weight?

Learning Rate

$$\mathbf{W}^{+} = \mathbf{W} - \eta \nabla \mathbf{C}$$

$$\nabla \mathbf{C} = \begin{bmatrix} \frac{\partial C}{\partial w_1} \\ \frac{\partial C}{\partial w_2} \\ \vdots \\ \frac{\partial C}{\partial w_n} \end{bmatrix}$$

*Note, these equations refer to 'L' (loss) as 'C' (cost).

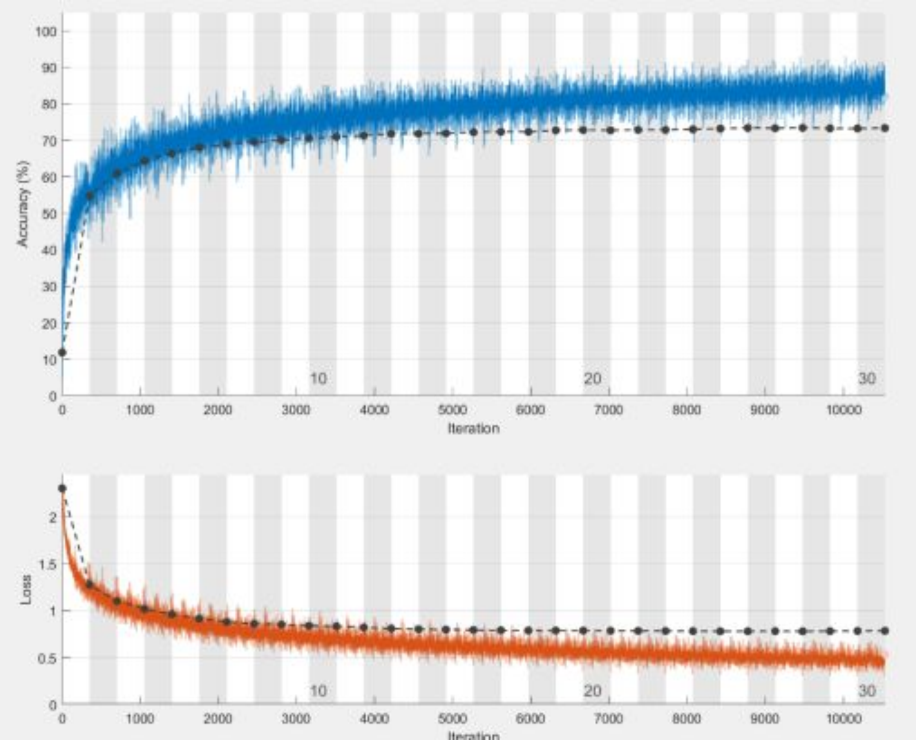
Can be extremely slow and memory inefficient ...
• Use smarter update rules that keep memory of past gradient values (eg. Momentum, adagrad, Adam)
• Use minibatches of data to update gradients, rather than the entire dataset

Training a Neural Network

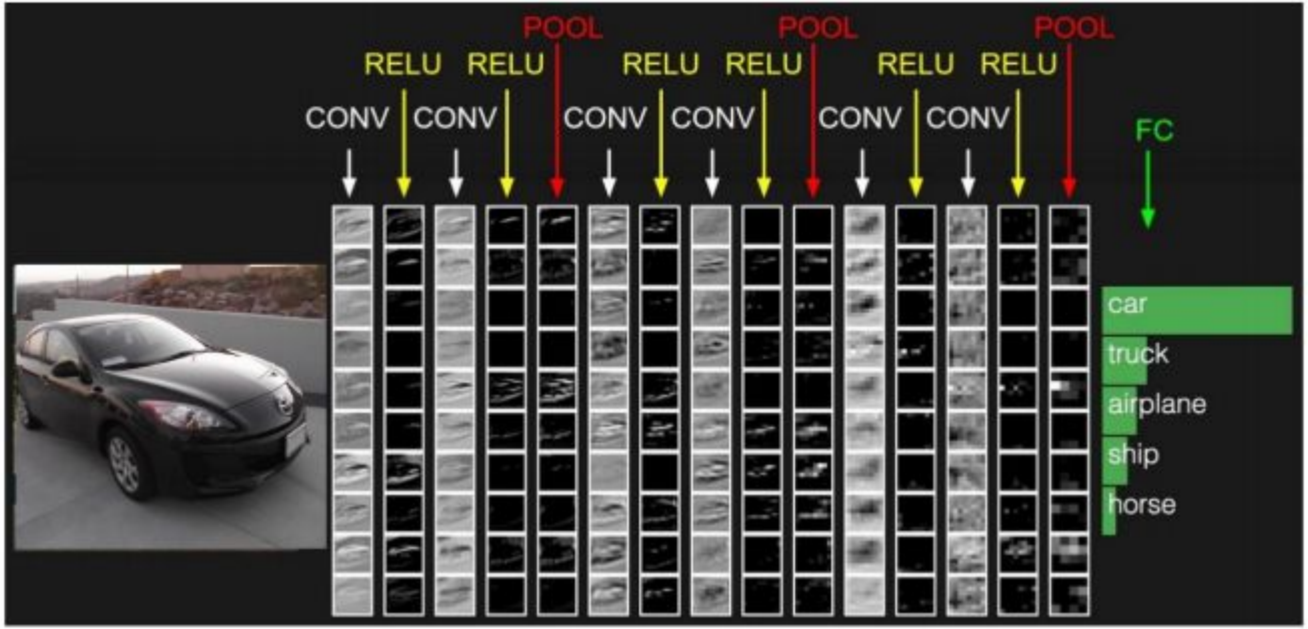
- Choose a network structure
  - Shape of input/output?
  - How many hidden layers, and how should the layers be connected?
- Initialize the network weights
- For training data (x, y):
  - Forward pass data x through the network to make a prediction, y_{pred}
  - Calculate loss $L(y, y_{pred})$
  - Change weights according to learning rate & gradient of loss with respect to weights
- Repeat … and repeat … and repeat
- For new data points (x) without a known output (y), we can pass x through the trained network to form a prediction!

When should we stop training?

- Each time we loop through the dataset is called an 'epoch'.
- Divide data into a *training set* and *validation set*.
  - Do not train on validation set.
  - Stop training when the validation error begins to increase.
  - Training beyond this is referred to as 'overfitting,' since the model becomes overfit to the training data (which is only a subset of the possible real data!).



Quick Recap:



POOL
POOL
POOL
RELU
RELU
RELU
RELU
RELU
RELU
CONV
CONV
CONV
CONV
CONV
CONV
FC
car
truck
airplane
ship
horse

HW 4 Problem:

Given an image, can we determine an object label for all pixels?
(“Segmentation”)

Segmentation Challenges

Based on what we have seen for image classification, how would you go about creating a network for pixel classification (i.e. image segmentation)?

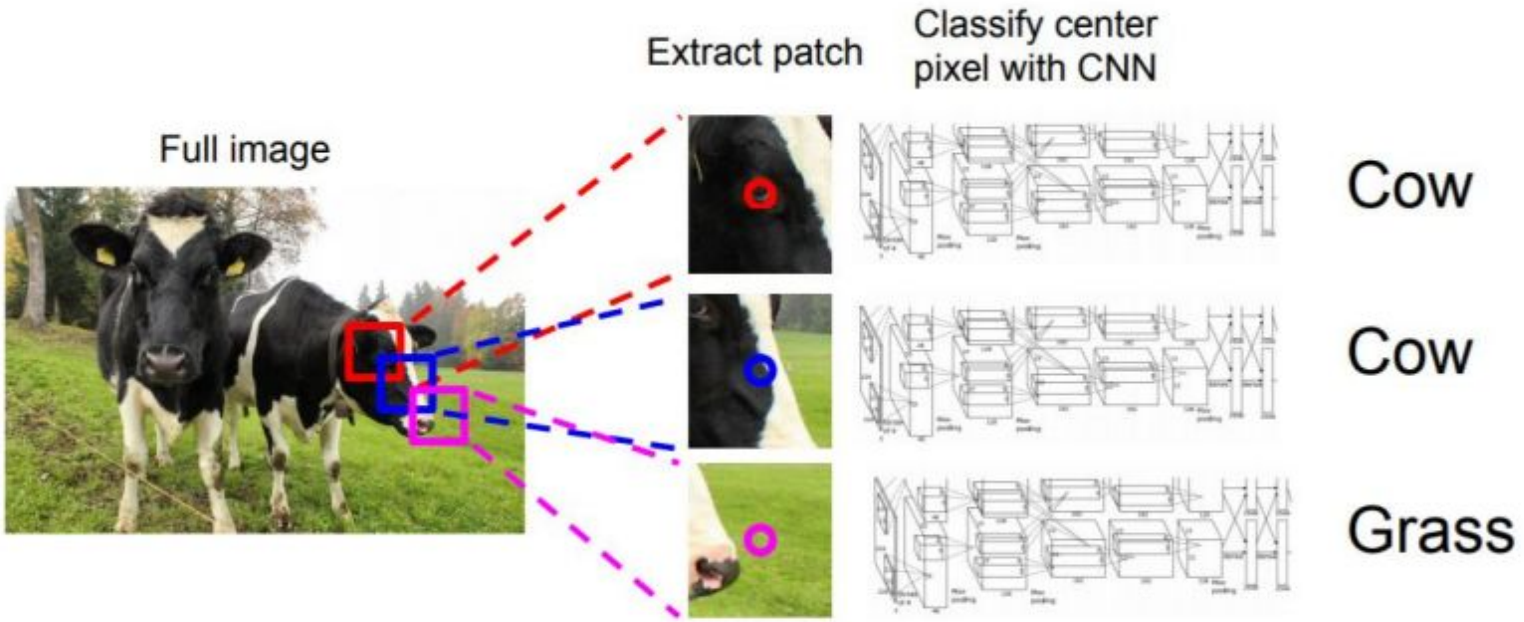
Segmentation Challenges

Based on what we have seen for image classification, how would you go about creating a network for pixel classification (i.e. image segmentation)?

Preliminary idea (Ciresan et al.):

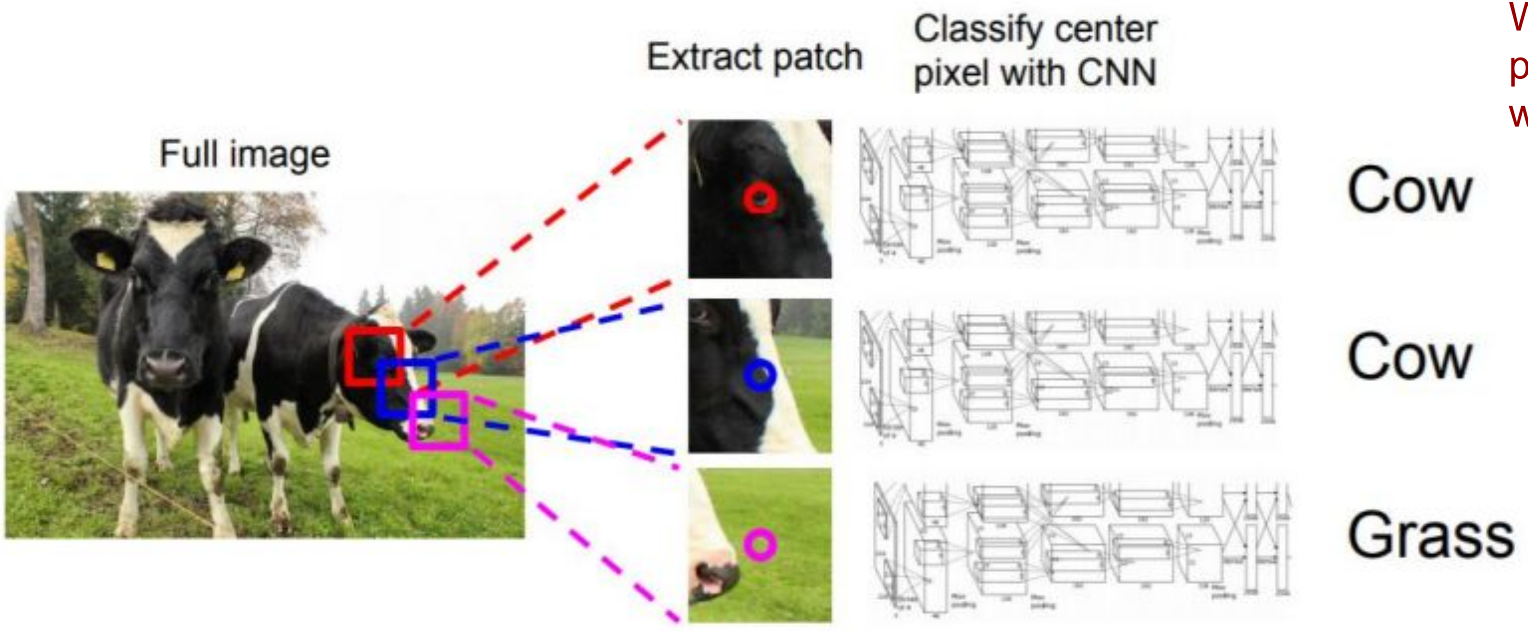
> Move a sliding window centered over each pixel. For each window location, predict the class of the pixel.

Semantic Segmentation Idea: Sliding Window



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

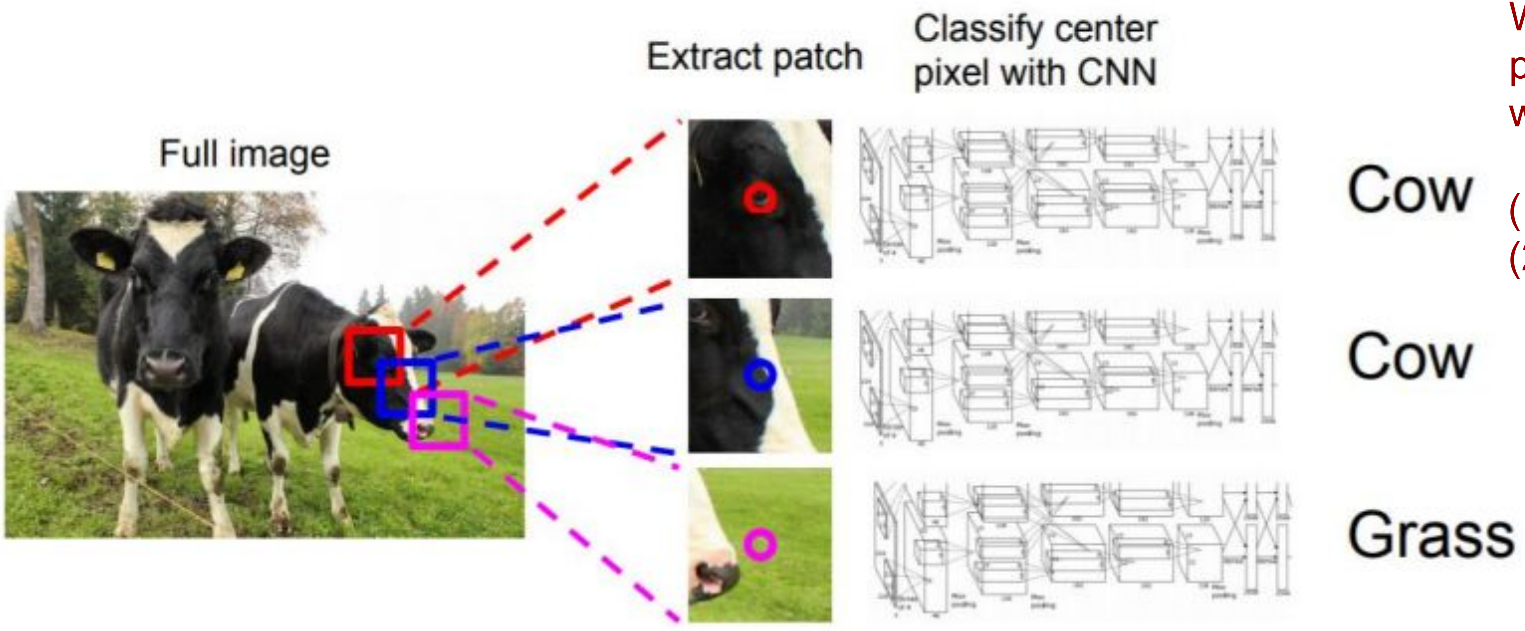
Semantic Segmentation Idea: Sliding Window



What are some possible problems with this method?

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: Sliding Window

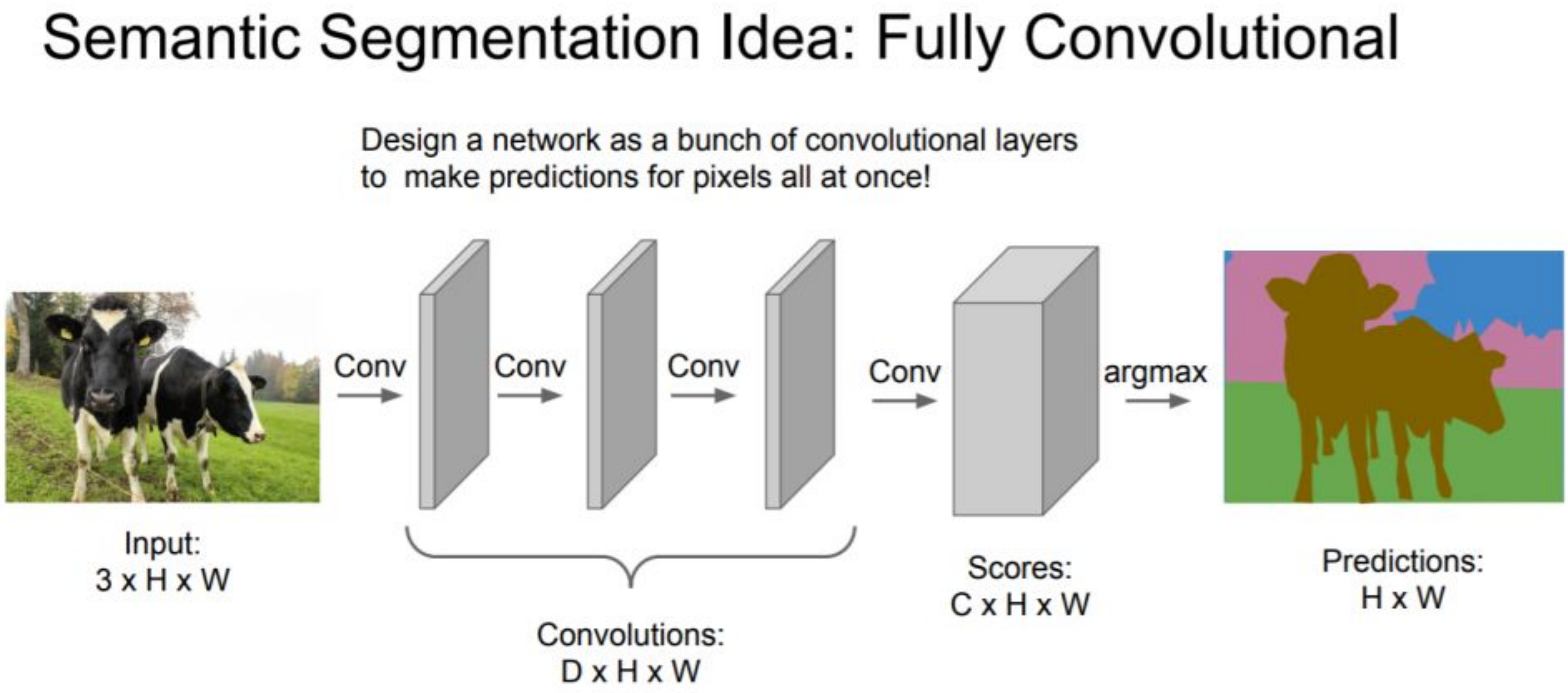


What are some possible problems with this method?

(1) Inefficient
(2) Distinguishing features occur at different scales

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

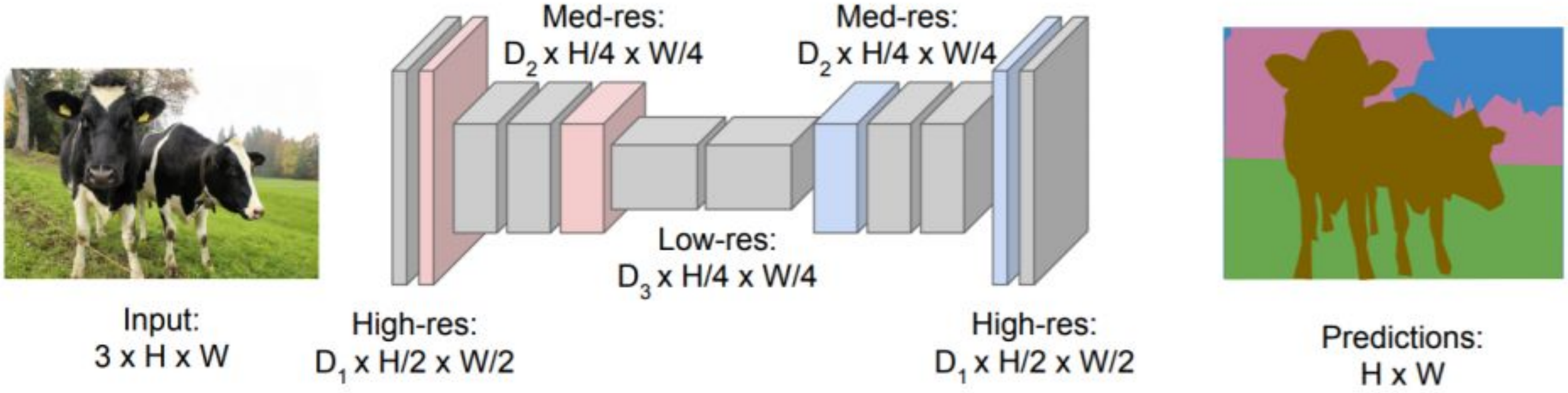
Addressing Inefficiency



Addressing Feature Scales

Semantic Segmentation Idea: Fully Convolutional

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

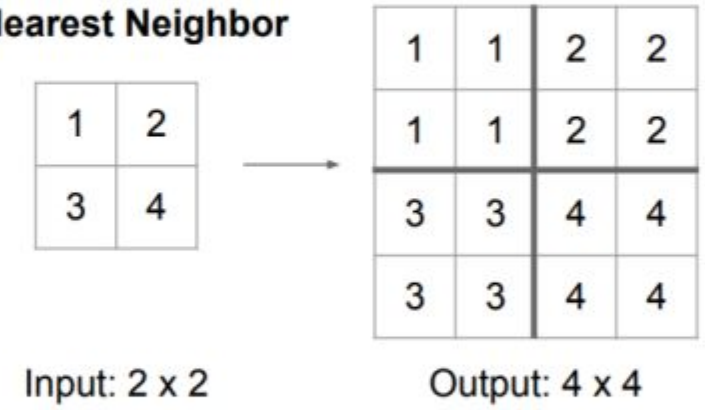


Long, Shelhamer, and Darrell, "Fully Convolutional Networks for Semantic Segmentation", CVPR 2015

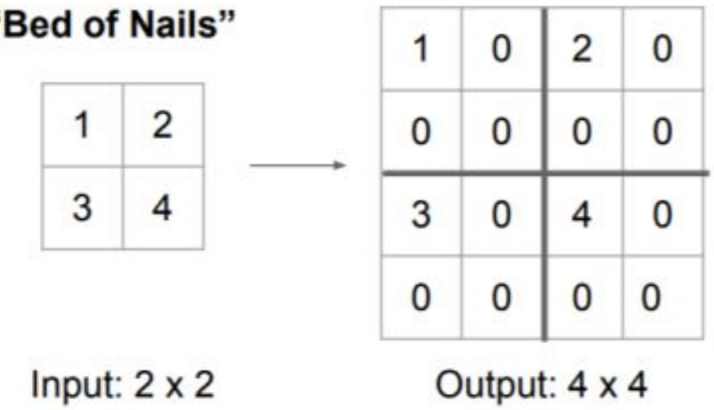
Noh et al, "Learning Deconvolution Network for Semantic Segmentation", ICCV 2015

Upsampling

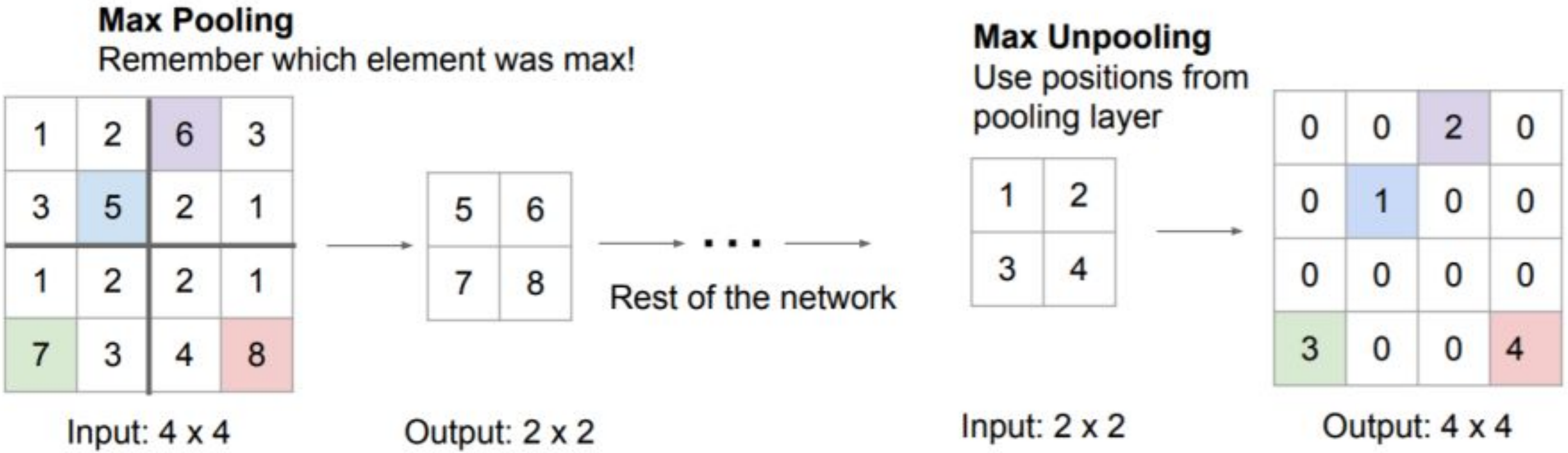
Nearest Neighbor



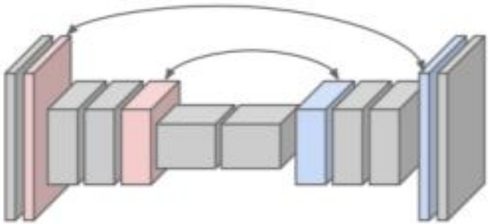
"Bed of Nails"



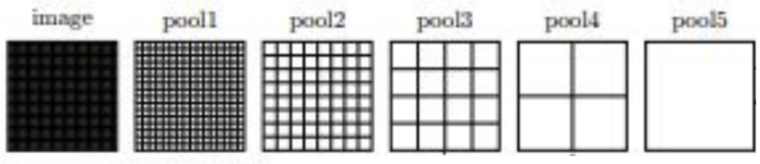
Upsampling



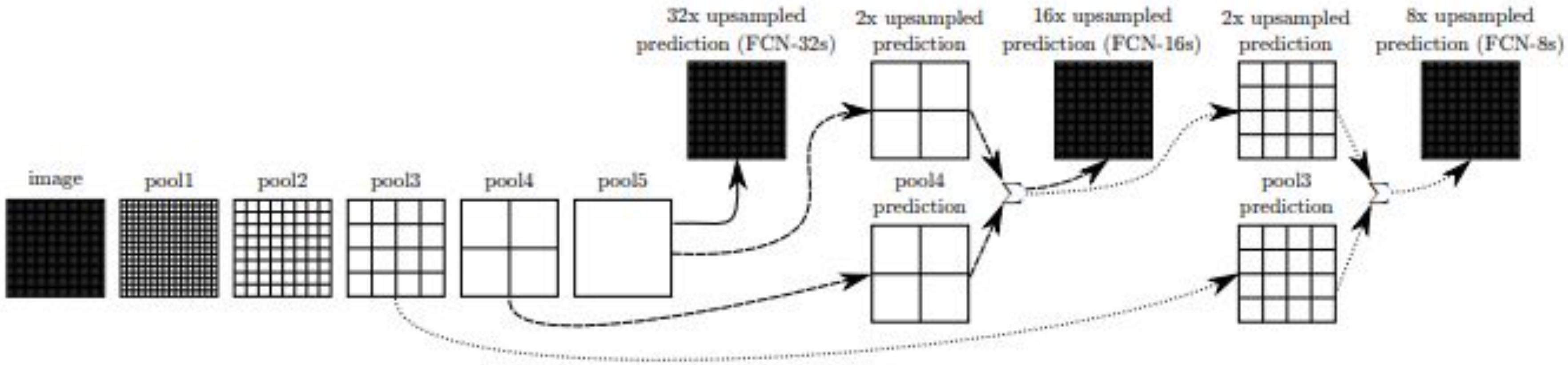
Corresponding pairs of downsampling and upsampling layers



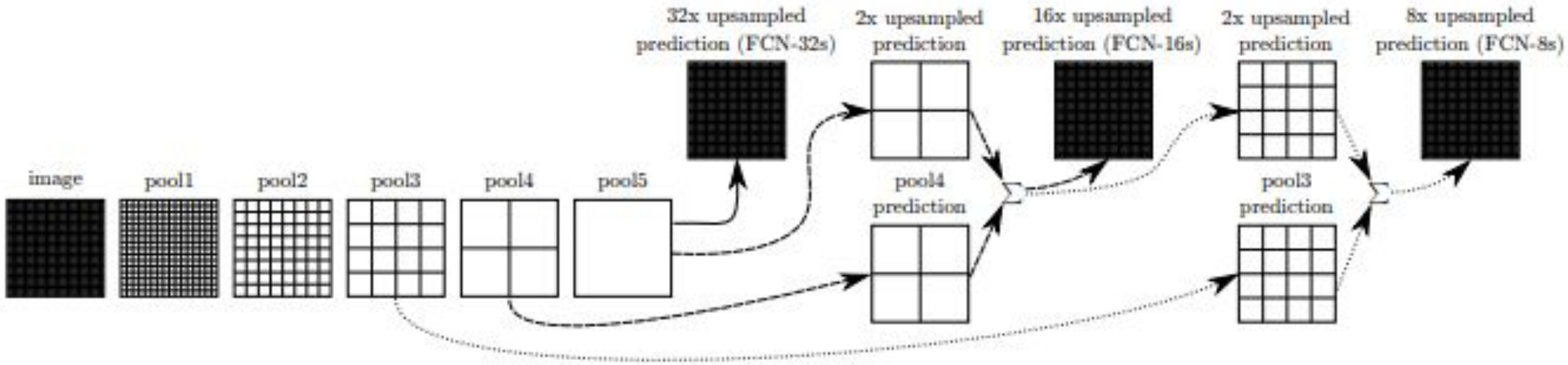
Fully Convolutional Networks



Fully Convolutional Networks



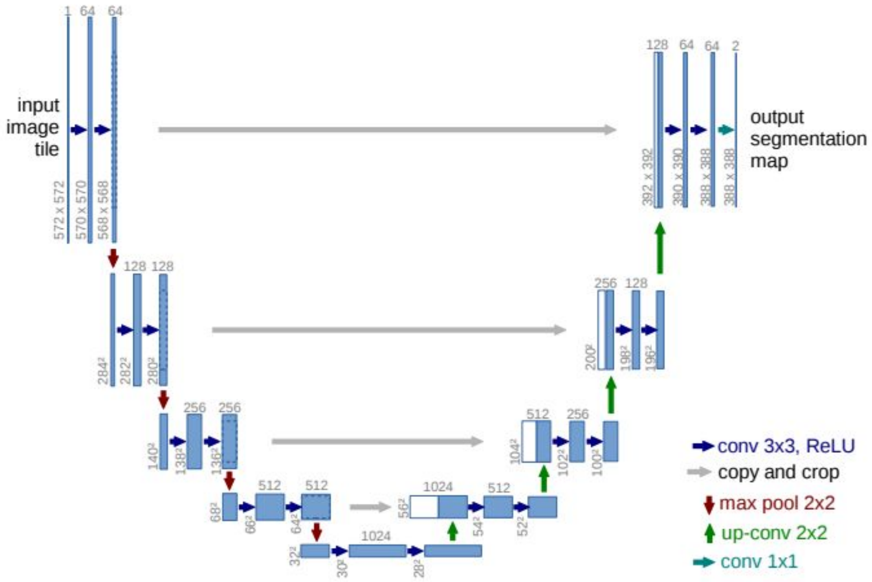
Fully Convolutional Networks



Output size matches input!

Refining the Approach: U-Net

“The architecture consists of a contracting path to capture context and a symmetric expanding path that enables precise localization.” (Ronneberger et al.)



Refining the Approach: U-Net