UNIVERSITY OF SURREY

Deep Learning Tutorial

UDRC Summer School

Muhammad Awais

UNIVERSITY OF SURREY

Outline

- Conventional Pattern recognition
- Learning Feature Representations
- Supervised Learning with Neural Network
- Loss Function
- Optimization
- Backpropagation in practice
- Backpropagation in deep learning libraries
- Introduction to CNN
- Latest development in CNN
- Application of CNN

Pattern Recognition

➢Pattern recognition architecture (first decade of 2000s)

Low-level Feature
Fixed
MFCC,
Gabor, SIFT,
SURF, GLOH,
LBP, etc.

Mid-Level Features
Unsupervised
k-mean, Visual
Phrases, GMM,
Sparse Coding,
Fisher Vector etc

pooling

Trainable Classifier
Supervised
Linear classifier, linear
regression, sofmax
SVM, FDA, decision
tree etc

Learning representation

Trainable Feature Transform

Trainable Classifier
Supervised
Linear classifier, linear regression, softmax, SVM, FDA, etc

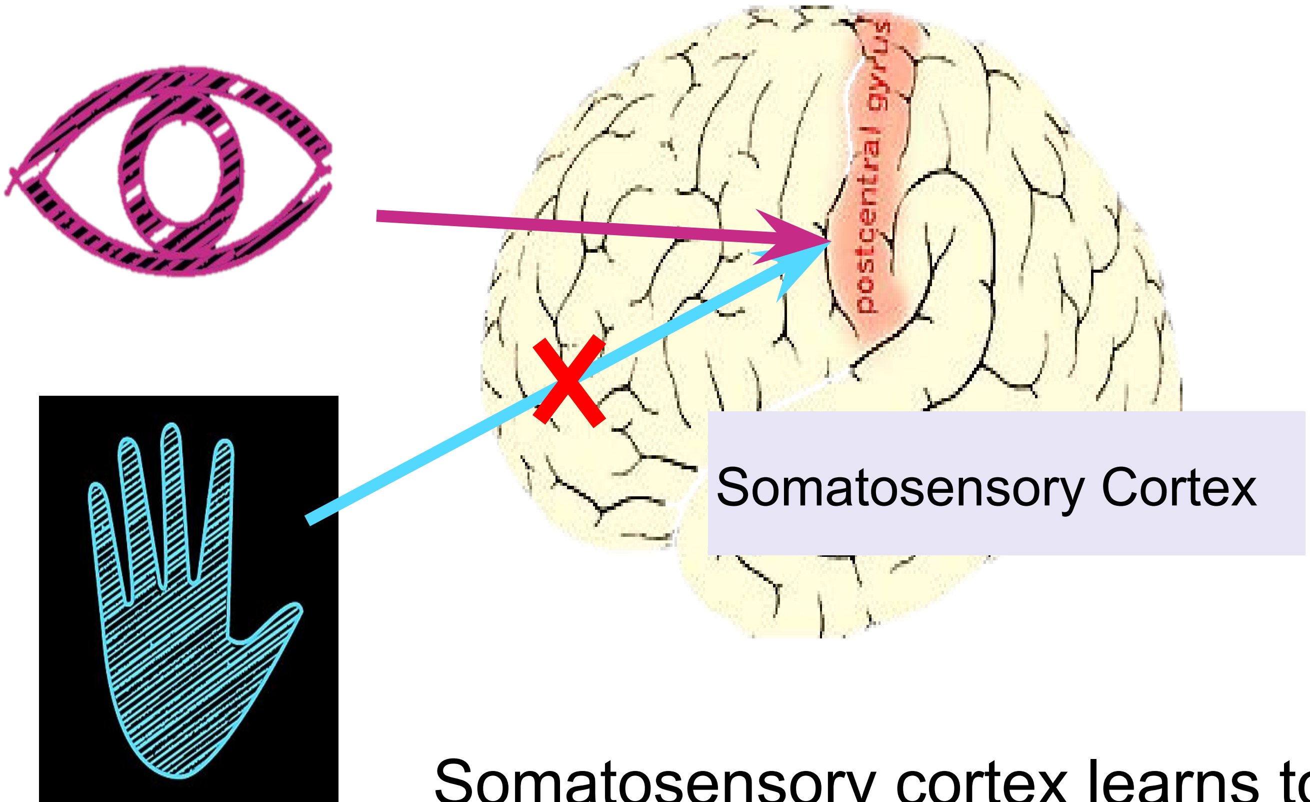
UNIVERSITY OF SURREY

Learning representation

→ **Trainable Feature Transform** → **Trainable Classifier**
Supervised
Linear classifier, linear regression, softmax, SVM, FDA, etc

The "One Learning Algorithm" hypothesis

The "one learning algorithm" hypothesis

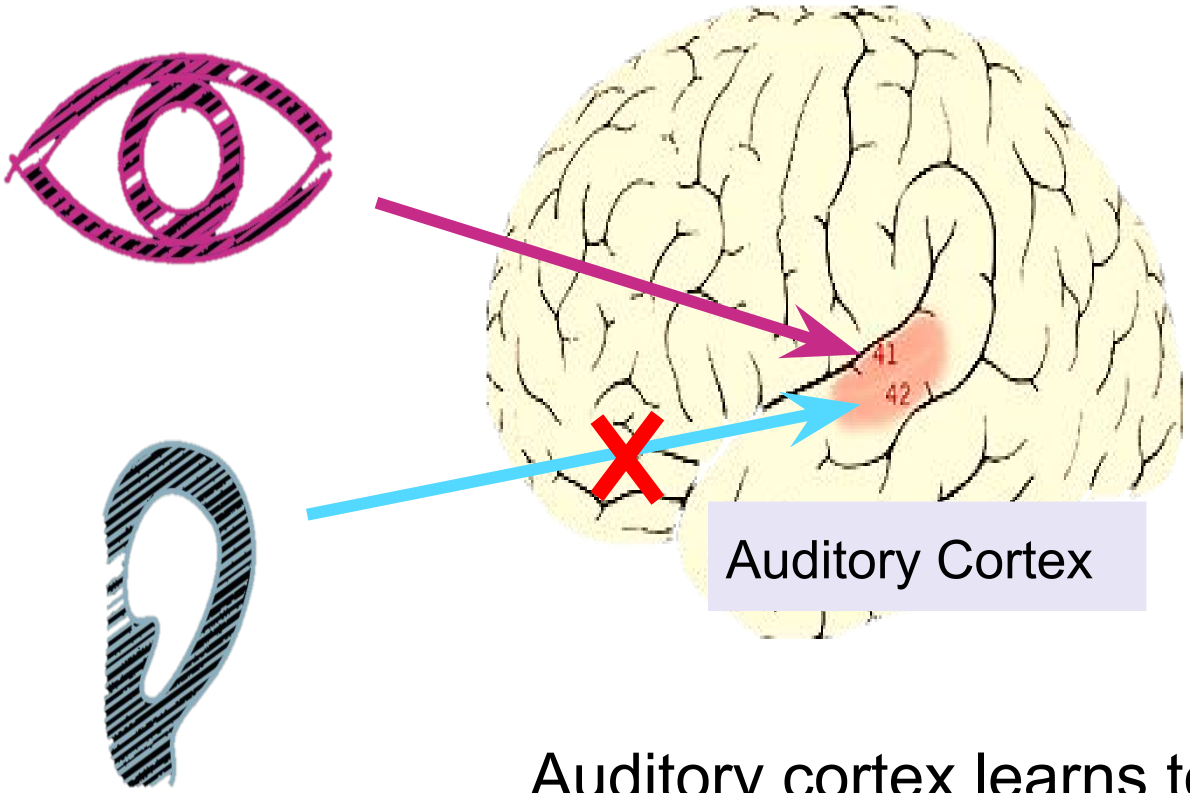


Somatosensory cortex learns to see

[Metin & Frost, 1989]

[Curtesy of Andrew Ng]

The “one learning algorithm” hypothesis



Auditory cortex learns to see

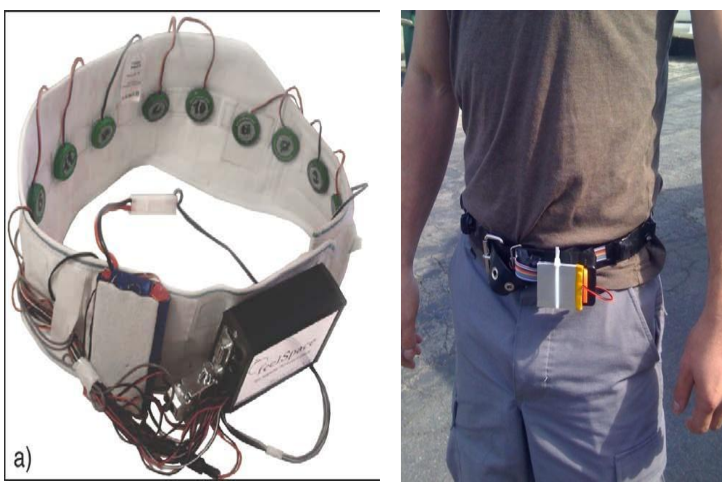
[Roe et al., 1992]

[Curtesy of Andrew Ng]

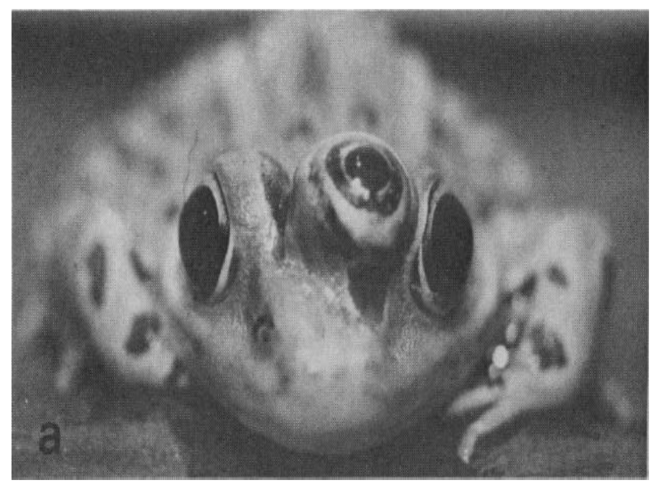
The “one learning algorithm” hypothesis

Seeing with your tongue

Human echolocation (sonar)



Haptic belt: Direction sense



Implanting a 3rd eye

[BrainPort; Welsh & Blasch, 1997; Nagel et al., 2005; Constantine-Paton & Law, 2009]

[Curtesy of Andrew Ng]

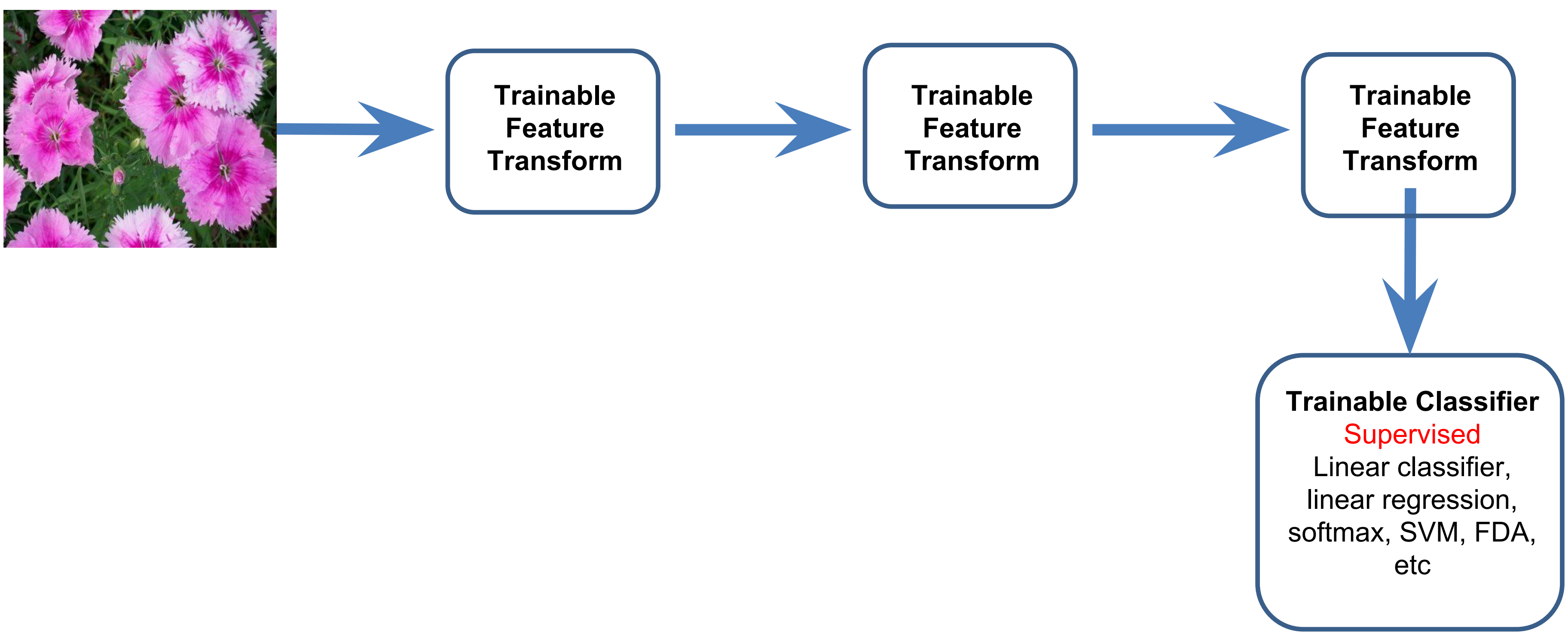
Learning representation

Trainable Feature Transform

Trainable Classifier
Supervised
Linear classifier, linear regression, softmax, SVM, FDA, etc

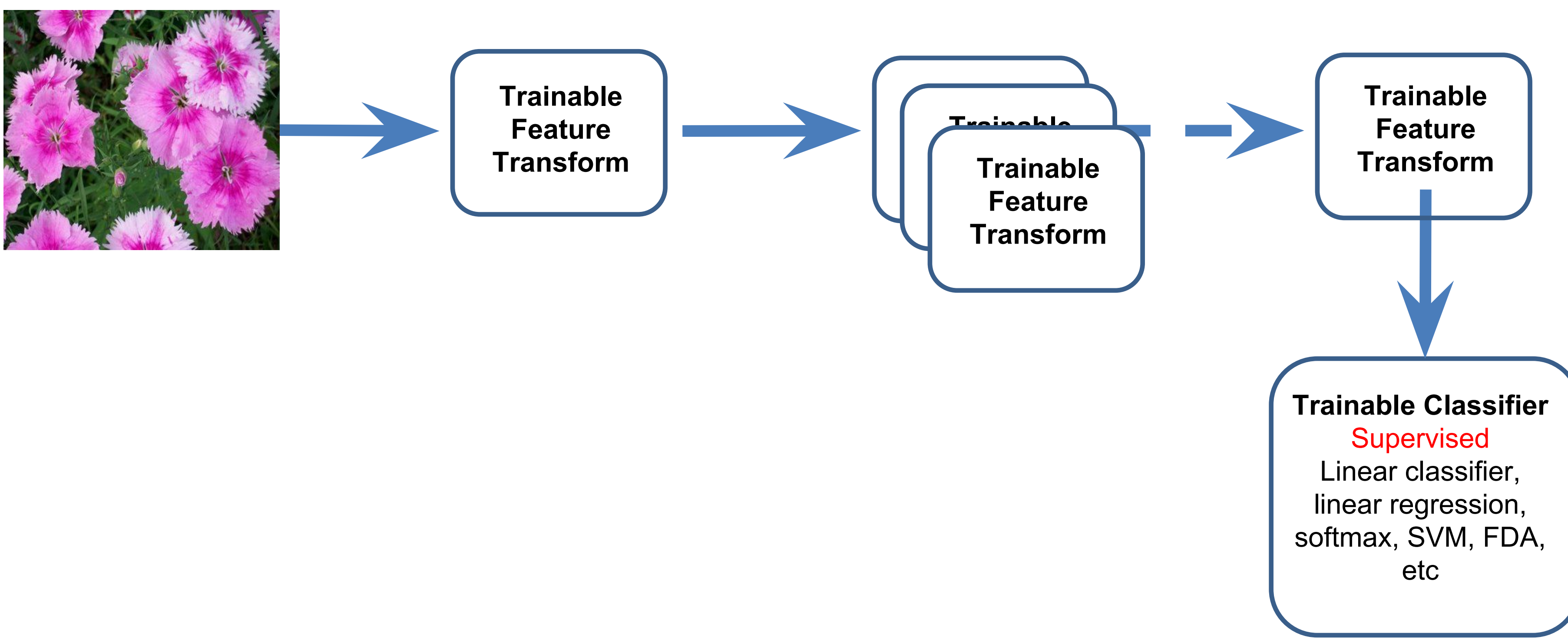
Learning Feature Hierarchy

➢Deep learning is all about learning feature hierarchies



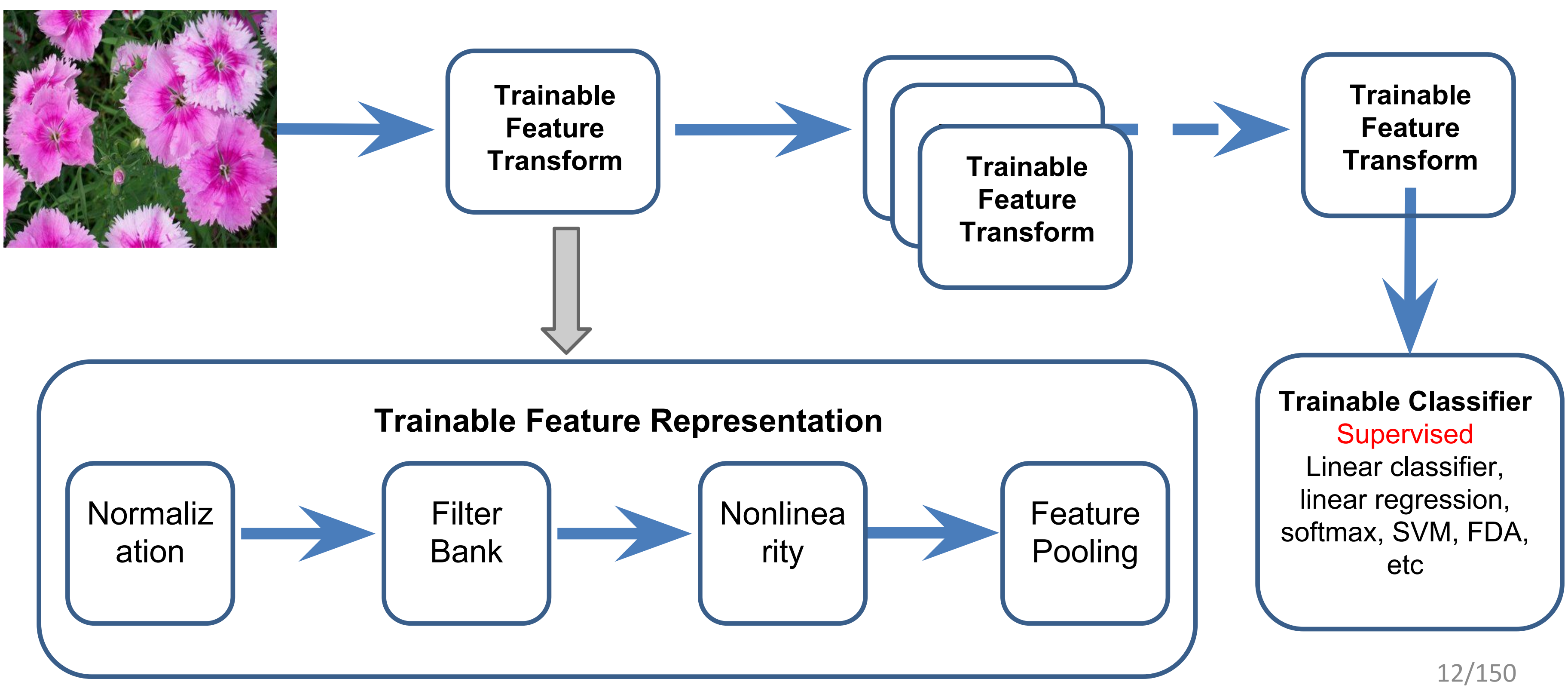
Going Deeper

➢Deep learning architecture



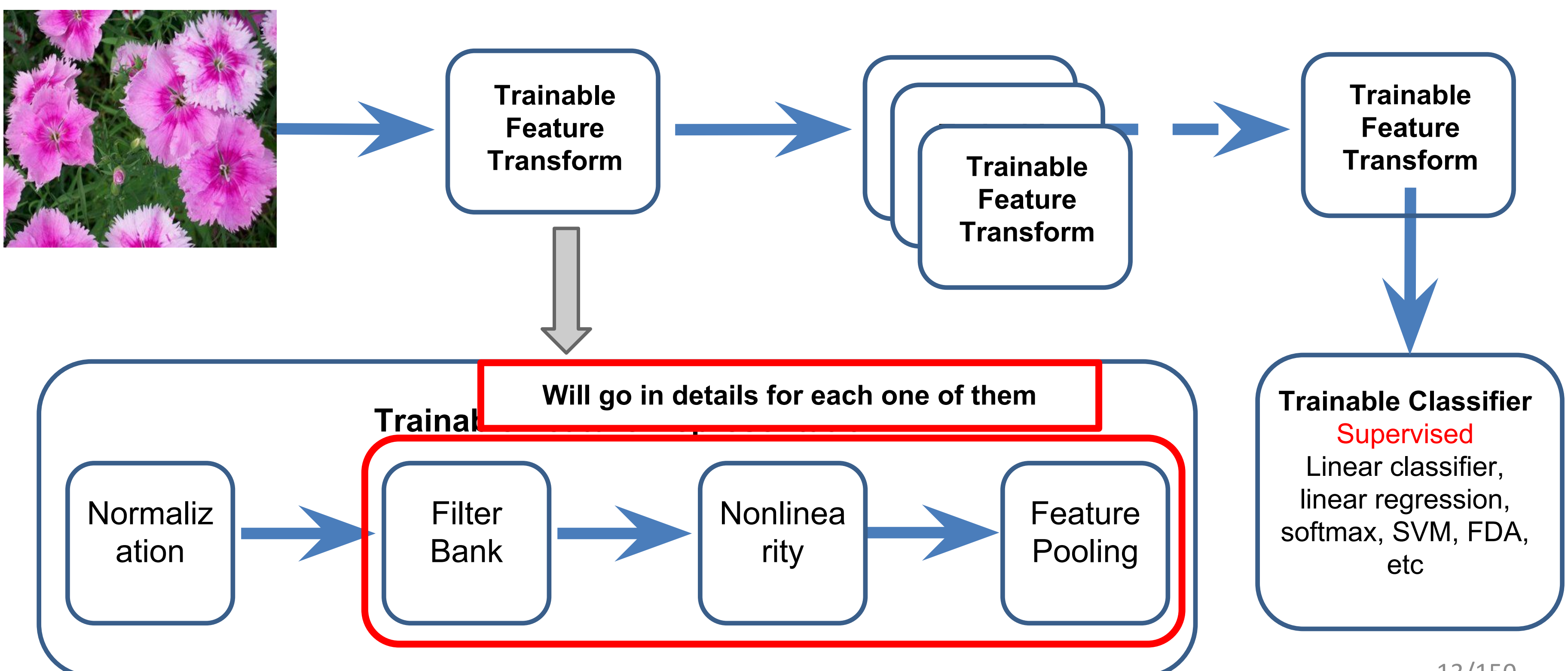
Deep Neural Network

➢Deep learning architecture



Deep Neural Network

➢Deep learning architecture



Supervised Learning with Neural Networks

- Neural Network training supervised learning
  - Dataset is given in term of input out pairs (x, y)
  - Define a loss/cost function for each example $J(W,b;x,y) = \frac{1}{2}\left\| h_{W,b}(x) - y \right\|^2.$
    - Cost function depends upon the type of problem
  - Compute an overall cost function J(W, b)
    - average over the training set
    - Add regularization term with trade off

$$
\begin{aligned}
J(W,b) &= \left[\frac{1}{m} \sum_{i=1}^{m} J(W,b;x^{(i)},y^{(i)}) \right] + \frac{\lambda}{2} \sum_{l=1}^{n_l-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} \left(W_{ji}^{(l)} \right)^2 \\
&= \left[\frac{1}{m} \sum_{i=1}^{m} \left(\frac{1}{2} \left\| h_{W,b}(x^{(i)}) - y^{(i)} \right\|^2 \right) \right] + \frac{\lambda}{2} \sum_{l=1}^{n_l-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} \left(W_{ji}^{(l)} \right)^2
\end{aligned}
$$

  - Use Stochastic Gradient Descent to update the weights of network
    - Use backpropagation to compute the gradients (just application of chain rule)

$$
\begin{aligned}
W_{ij}^{(l)} &= W_{ij}^{(l)} - \alpha \frac{\partial}{\partial W_{ij}^{(l)}} J(W,b) \\
b_i^{(l)} &= b_i^{(l)} - \alpha \frac{\partial}{\partial b_i^{(l)}} J(W,b)
\end{aligned}
$$

UNIVERSITY OF SURREY

Supervised Learning with Neural Networks

- Neural Network training supervised learning
  - Dataset is given in term of input out pairs (x, y)
  - Define a loss/cost function for each example $J(W,b;x,y) = \frac{1}{2}\left\| h_{W,b}(x) - y \right\|^2.$
    - Cost function depends upon the type of problem
  - Compute an overall cost function J(W, b)
    - average over the training set
    - Add regularization term with trade off

Loss function

$$
\begin{aligned}
J(W,b) &= \left[\frac{1}{m}\sum_{i=1}^{m} J(W,b;x^{(i)},y^{(i)})\right] + \frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(W_{ji}^{(l)}\right)^2 \\
&= \left[\frac{1}{m}\sum_{i=1}^{m}\left(\frac{1}{2}\left\| h_{W,b}(x^{(i)}) - y^{(i)} \right\|^2\right)\right] + \frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(W_{ji}^{(l)}\right)^2
\end{aligned}
$$

  - Use Stochastic Gradient Descent to update the weights of network
    - Use backpropagation to compute the gradients (just application of chain rule)

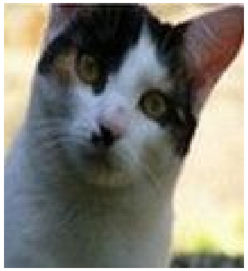
$$
W_{ij}^{(l)} = W_{ij}^{(l)} - \alpha \frac{\partial}{\partial W_{ij}^{(l)}} J(W,b)
$$

$$
b_{i}^{(l)} = b_{i}^{(l)} - \alpha \frac{\partial}{\partial b_{i}^{(l)}} J(W,b)
$$

Loss Functions

SVM Maximum Margin Hinge loss

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:

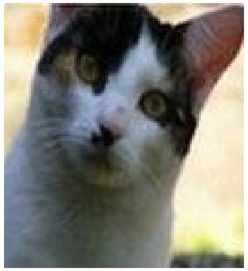
  

cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**

UNIVERSITY OF SURREY

SVM Maximum Margin Hinge loss

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:

cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**

Multiclass SVM loss:

Given an example (x_i, y_i)
where x_i is the image and
where y_i is the (integer) label,

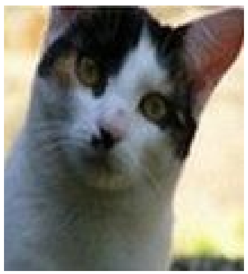
and using the shorthand for the scores vector: $s = f(x_i, W)$

the SVM loss has the form:

$$\boxed{L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)}$$

SVM Maximum Margin Hinge loss

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:

cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**
Losses:	2.9		

Multiclass SVM loss:

Given an example (x_i, y_i)
where x_i is the image and
where y_i is the (integer) label,

and using the shorthand for the scores vector: $s = f(x_i, W)$

the SVM loss has the form:

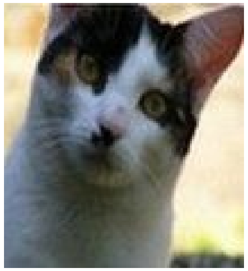
$$\boxed{L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)}$$

= max(0, 5.1 - 3.2 + 1)
+max(0, -1.7 - 3.2 + 1)
= max(0, 2.9) + max(0, -3.9)
= 2.9 + 0
= 2.9

UNIVERSITY OF SURREY

SVM Maximum Margin Hinge loss

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:

cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**
Losses:	2.9	0	

Multiclass SVM loss:

Given an example (x_i, y_i)
where x_i is the image and
where y_i is the (integer) label,

and using the shorthand for the scores vector: $s = f(x_i, W)$

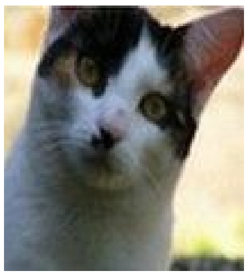
the SVM loss has the form:

$$\boxed{L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)}$$

$= \max(0, 1.3 - 4.9 + 1)$
$\quad + \max(0, 2.0 - 4.9 + 1)$
$= \max(0, -2.6) + \max(0, -1.9)$
$= 0 + 0$
$= 0$

SVM Maximum Margin Hinge loss

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:

cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**
Losses:	2.9	0	10.9

Multiclass SVM loss:

Given an example (x_i, y_i)
where x_i is the image and
where y_i is the (integer) label,

and using the shorthand for the scores vector: $s = f(x_i, W)$

the SVM loss has the form:

$$\boxed{L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)}$$

= max(0, 2.2 - (-3.1) + 1)
+max(0, 2.5 - (-3.1) + 1)
= max(0, 5.3) + max(0, 5.6)
= 5.3 + 5.6
= 10.9

SVM Maximum Margin Hinge loss

Suppose: 3 training examples, 3 classes.
With some W the scores $f(x, W) = Wx$ are:

cat	**3.2**	1.3	2.2
car	5.1	**4.9**	2.5
frog	-1.7	2.0	**-3.1**
Losses:	2.9	0	10.9

Multiclass SVM loss:

Given an example (x_i, y_i)
where x_i is the image and
where y_i is the (integer) label,

and using the shorthand for the scores vector: $s = f(x_i, W)$

the SVM loss has the form:

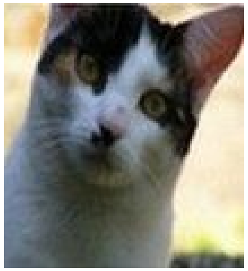
$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

and the full training loss is the mean over all examples in the training data:

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i$$

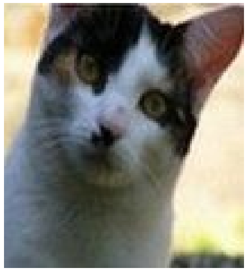
L = (2.9 + 0 + 10.9)/3
= **4.6**

Softmax Classifier (Multinomial Logistic Regression)



cat	**3.2**
car	5.1
frog	-1.7

Softmax Classifier (Multinomial Logistic Regression)

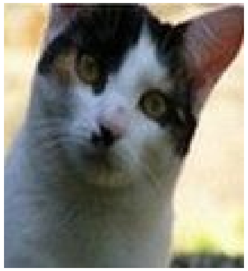


scores = unnormalized log probabilities of the classes.

$$\boxed{s = f(x_i; W)}$$

cat	**3.2**
car	5.1
frog	-1.7

Softmax Classifier (Multinomial Logistic Regression)

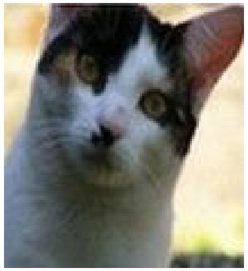


scores = unnormalized log probabilities of the classes.

$$P(Y=k|X=x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$ where $$s = f(x_i; W)$$

cat	**3.2**
car	5.1
frog	-1.7

Softmax Classifier (Multinomial Logistic Regression)



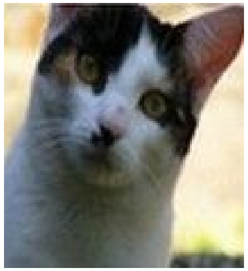
scores = unnormalized log probabilities of the classes.

$$P(Y=k|X=x_i) = \boxed{\frac{e^{s_k}}{\sum_j e^{s_j}}} \quad \text{where} \quad s = f(x_i; W)$$

Softmax function

cat	**3.2**
car	5.1
frog	-1.7

Softmax Classifier (Multinomial Logistic Regression)



scores = unnormalized log probabilities of the classes.

$$\boxed{P(Y=k|X=x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}} \quad \text{where} \quad \boxed{s = f(x_i; W)}$$

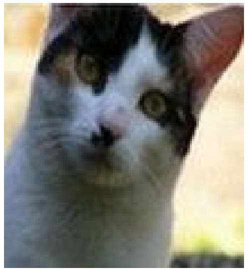
Want to maximize the log likelihood, or (for a loss function) to minimize the negative log likelihood of the correct class:

$$\boxed{L_i = -\log P(Y=y_i|X=x_i)}$$

cat	**3.2**
car	5.1
frog	-1.7

UNIVERSITY OF SURREY

Softmax Classifier (Multinomial Logistic Regression)



scores = unnormalized log probabilities of the classes.

$$\boxed{P(Y=k|X=x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}}$$ where $$\boxed{s = f(x_i; W)}$$

Want to maximize the log likelihood, or (for a loss function) to minimize the negative log likelihood of the correct class:

$$\boxed{L_i = -\log P(Y = y_i | X = x_i)}$$

cat	**3.2**
car	5.1
frog	-1.7

in summary: $$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

Loss function recap

- We have some dataset of (x,y)
- We have a **score function:** $s = f(x;W) \overset{\text{e.g.}}{=} Wx$
- We have a **loss function**:

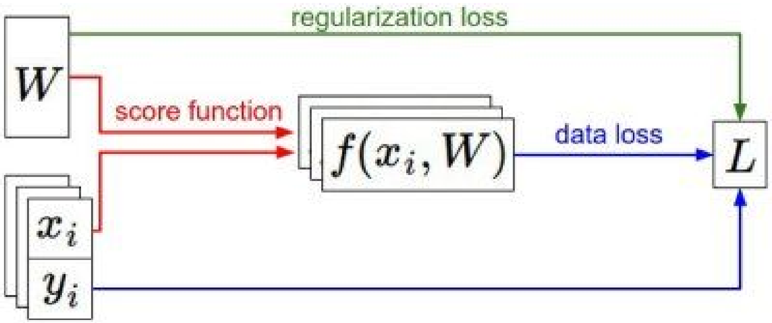
Softmax

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

SVM

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i + R(W)$$ Full loss



UNIVERSITY OF SURREY

Supervised Learning with Neural Networks

- Neural Network training supervised learning
  - Dataset is given in term of input out pairs (x, y)
  - Define a loss/cost function for each example

    $$J(W,b;x,y) = \frac{1}{2}\left\| h_{W,b}(x) - y \right\|^2.$$

    - Cost function depends upon the type of problem
  - Compute an overall cost function J(W, b)
    - average over the training set
    - Add regularization term with trade off

$$\begin{aligned} J(W,b) &= \left[\frac{1}{m}\sum_{i=1}^{m} J(W,b;x^{(i)},y^{(i)})\right] + \frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(W_{ji}^{(l)}\right)^2 \\ &= \left[\frac{1}{m}\sum_{i=1}^{m}\left(\frac{1}{2}\left\|h_{W,b}(x^{(i)}) - y^{(i)}\right\|^2\right)\right] + \frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(W_{ji}^{(l)}\right)^2 \end{aligned}$$

  - Use Stochastic Gradient Descent to update the weights of network
    - Use backpropagation to compute the gradients (just application of chain rule)

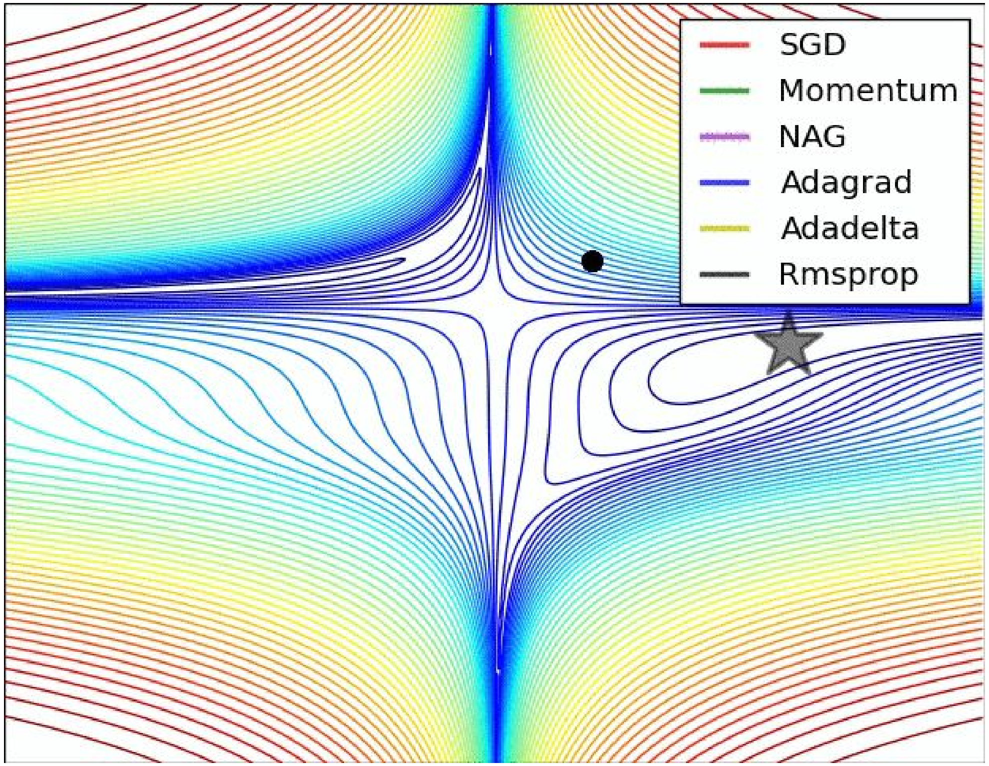
Optimization (SGD, Momentum,...)

$$W_{ij}^{(l)} = W_{ij}^{(l)} - \alpha\frac{\partial}{\partial W_{ij}^{(l)}}J(W,b)$$

$$b_{i}^{(l)} = b_{i}^{(l)} - \alpha\frac{\partial}{\partial b_{i}^{(l)}}J(W,b)$$

UNIVERSITY OF SURREY

Optimization

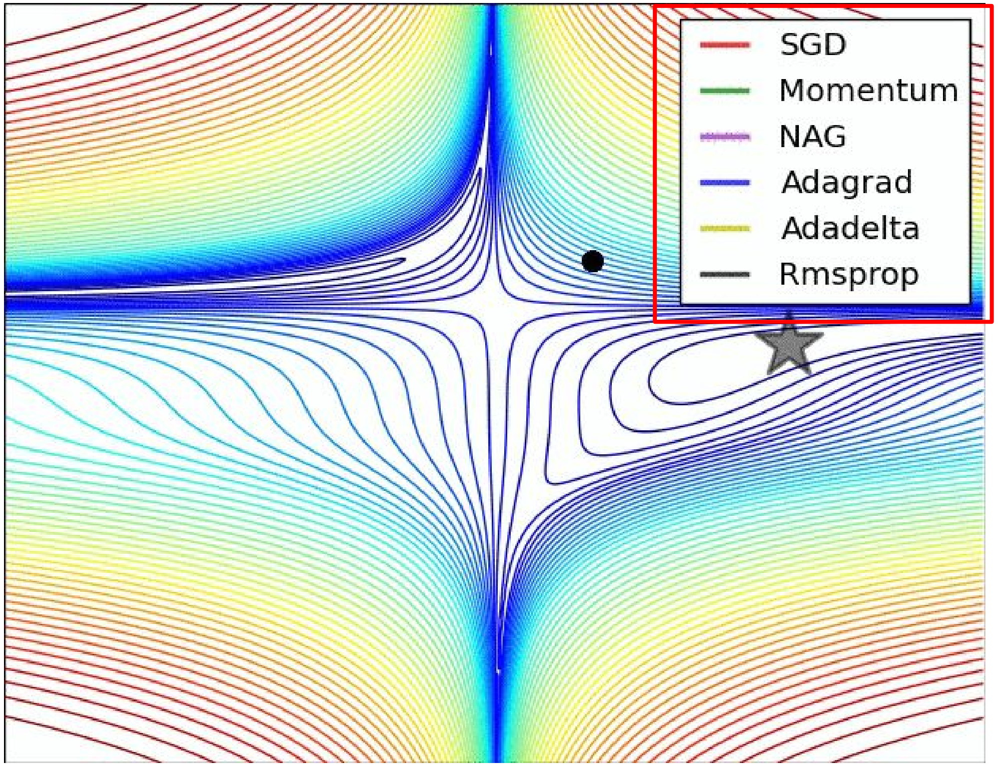


(image credits to Alec Radford)

```
# Vanilla Gradient Descent

while True:
  weights_grad = evaluate_gradient(loss_fun, data, weights)
  weights += - step_size * weights_grad # perform parameter update
```

UNIVERSITY OF SURREY

Optimization



All of them need derivative of loss with respect of parameters

(image credits to Alec Radford)

```
# Vanilla Gradient Descent

while True:
  weights_grad = evaluate_gradient(loss_fun, data, weights)
  weights += - step_size * weights_grad # perform parameter update
```

Optimization

Two ways to compute gradient:

Numerical gradient

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Analytic gradient by using calculus

Numerical gradient: slow (unsuitable for large # of parameters), approximate but easy to code
Analytic gradient: fast (suitable for large # of parameters), exact but error-prone

In practice: Derive analytic gradient, check implementation for smaller problems with numerical gradient

Optimization

Two ways to compute gradient:

Numerical gradient

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Analytic gradient by using calculus

Numerical gradient: slow (unsuitable for large # of parameters), approximate but easy to code

Analytic gradient: fast (suitable for large # of parameters), exact but error-prone

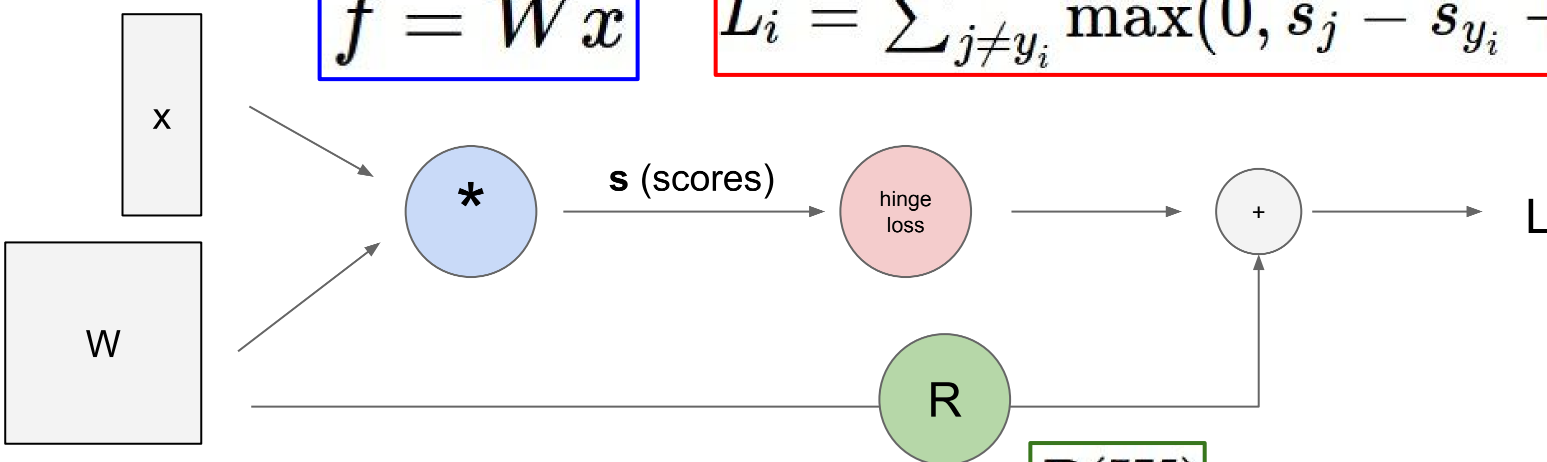
In practice: Derive analytic gradient, check implementation for smaller problems with numerical gradient

Optimization

Computational Graph

$$f = Wx$$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$



$$R(W)$$

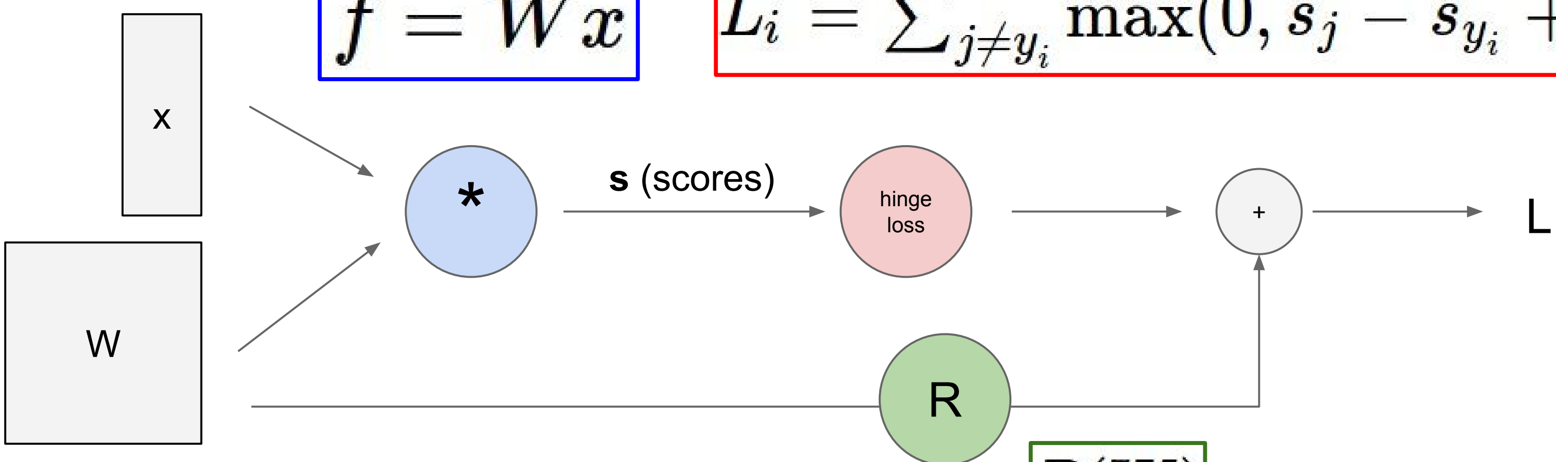
Optimization

An example of f(x) is DCNN

Computational Graph

$$\boxed{f = Wx}$$

$$\boxed{L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)}$$



$$\boxed{R(W)}$$

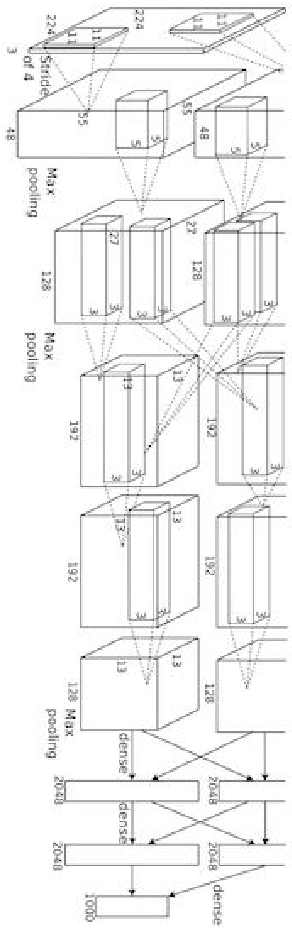
Optimization

Convolutional Network
(AlexNet)

input image

weights

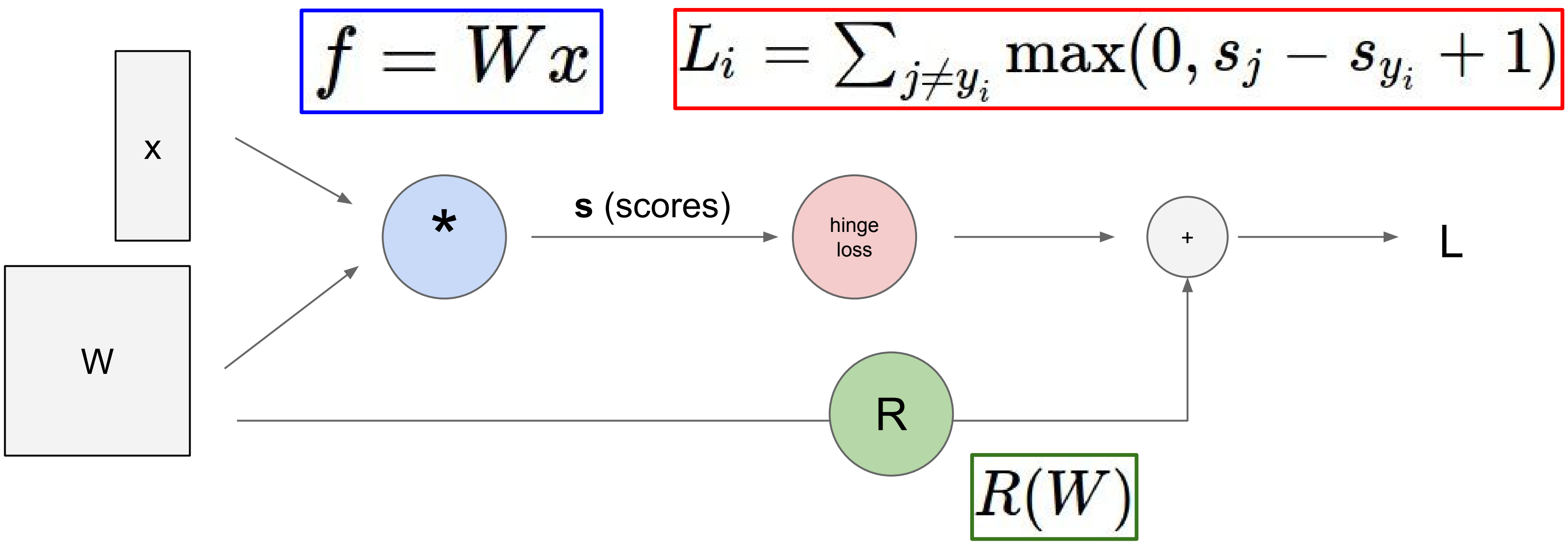
loss



Optimization

Need analytic gradient to learn W

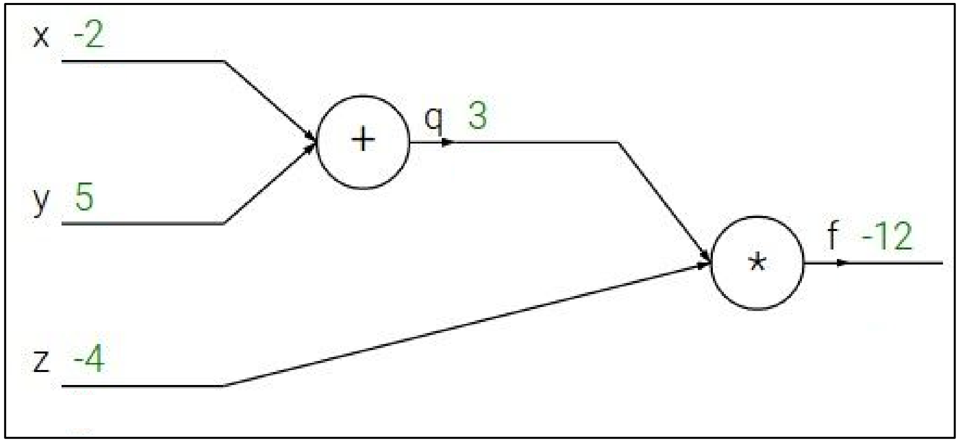
Computational Graph



Analytic gradient

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

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



Analytic gradient

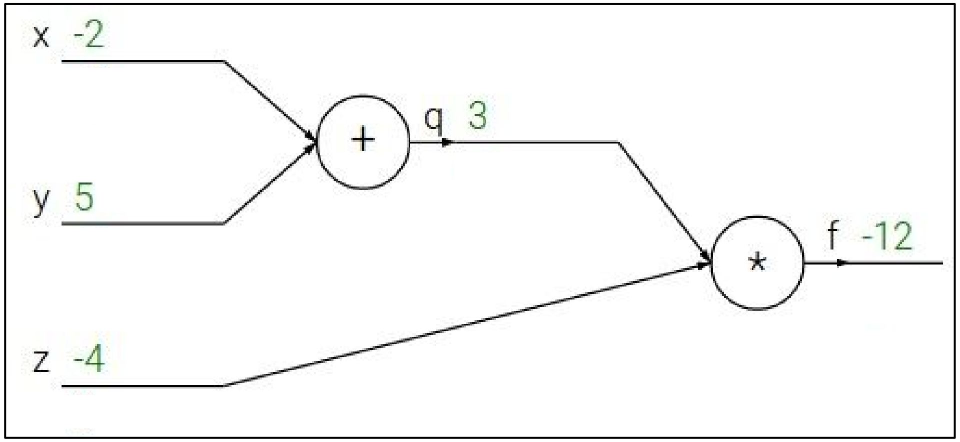
$$f(x, y, z) = (x + y)z$$

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



Analytic gradient

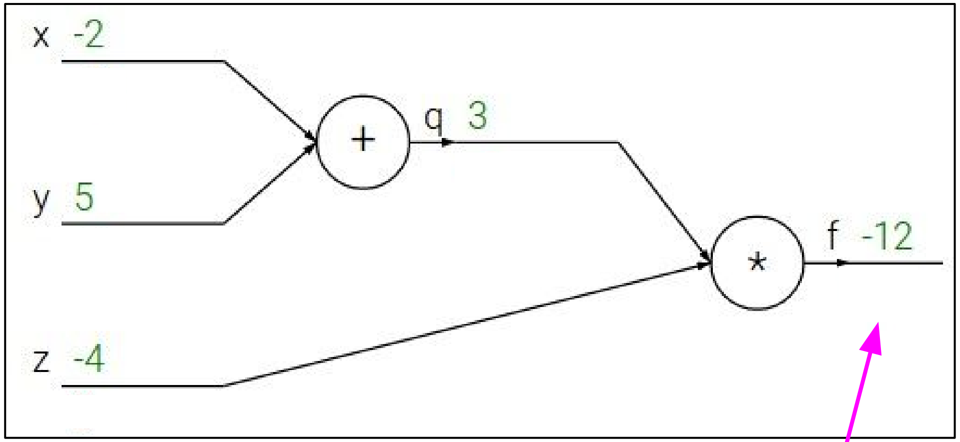
$$f(x,y,z) = (x+y)z$$

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



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

Analytic gradient

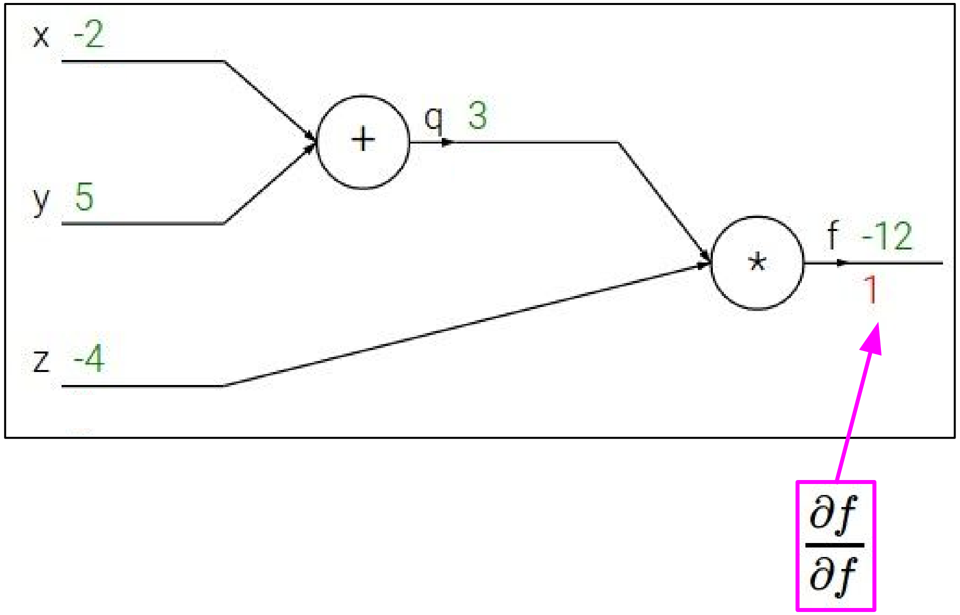
$$f(x, y, z) = (x + y)z$$

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



UNIVERSITY OF SURREY

Analytic gradient

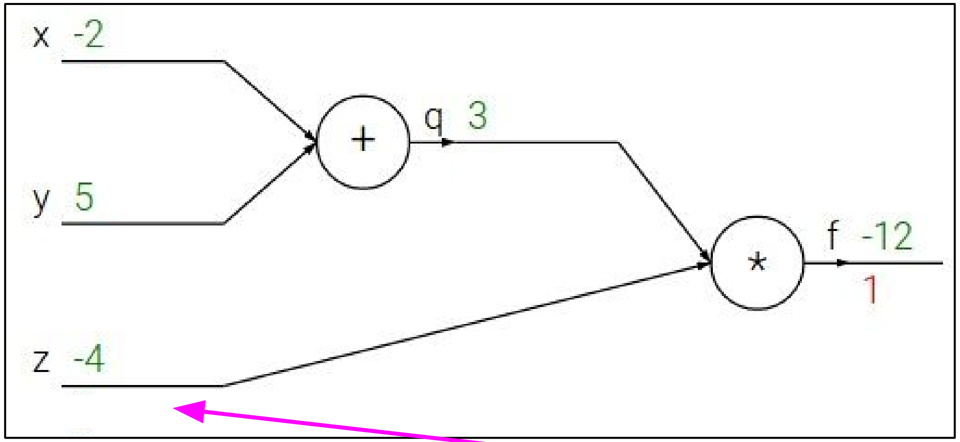
$$f(x, y, z) = (x + y)z$$

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



$$\frac{\partial f}{\partial z}$$

Analytic gradient

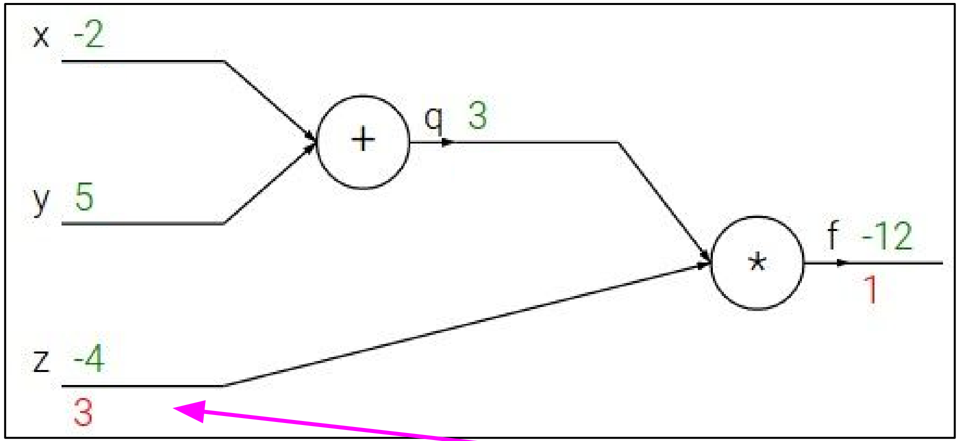
$$f(x,y,z) = (x+y)z$$

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



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

Analytic gradient

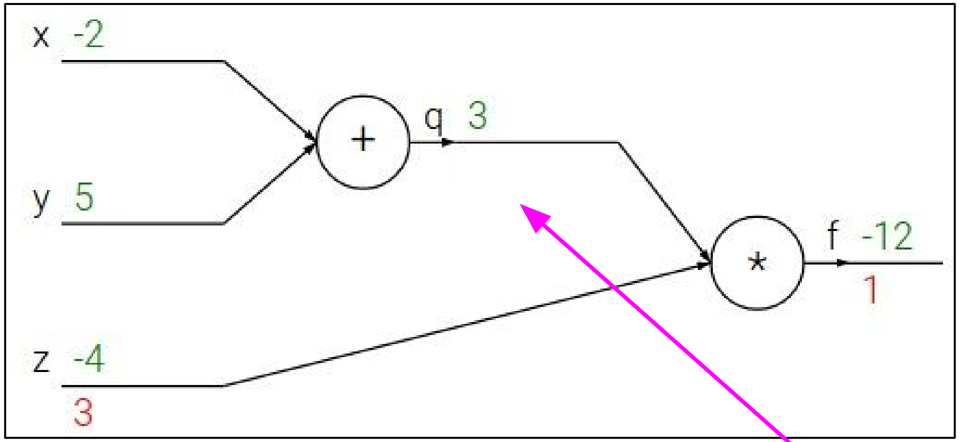
$$f(x, y, z) = (x + y)z$$

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

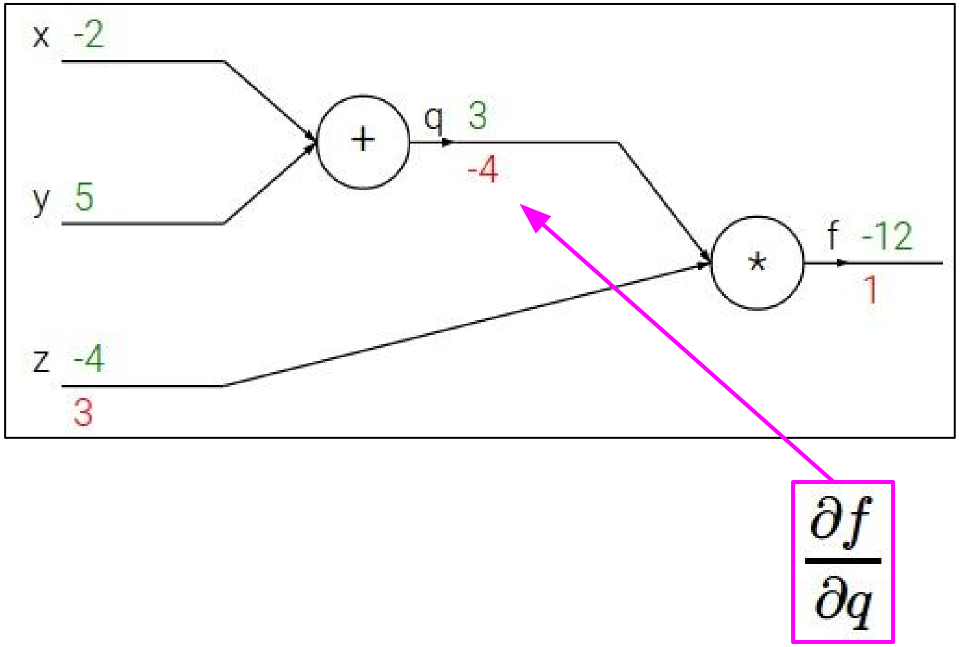


$$\frac{\partial f}{\partial q}$$

Analytic gradient

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

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

Analytic gradient

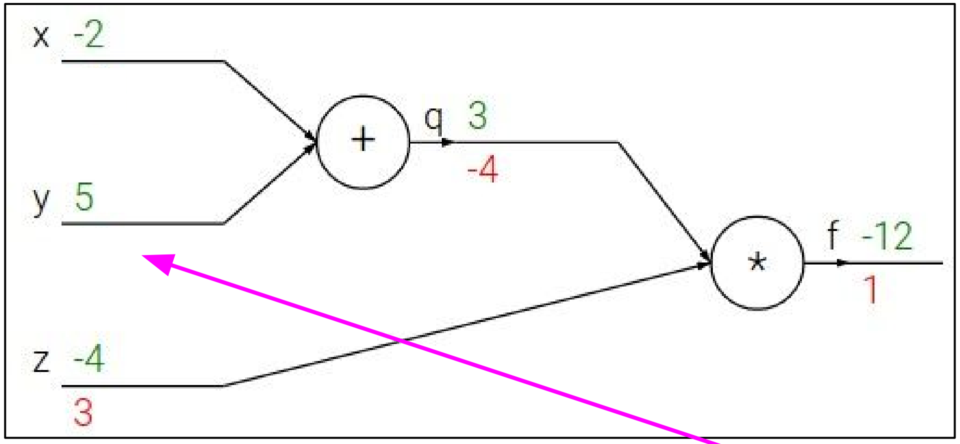
$$f(x,y,z) = (x+y)z$$

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



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

UNIVERSITY OF SURREY

Analytic gradient

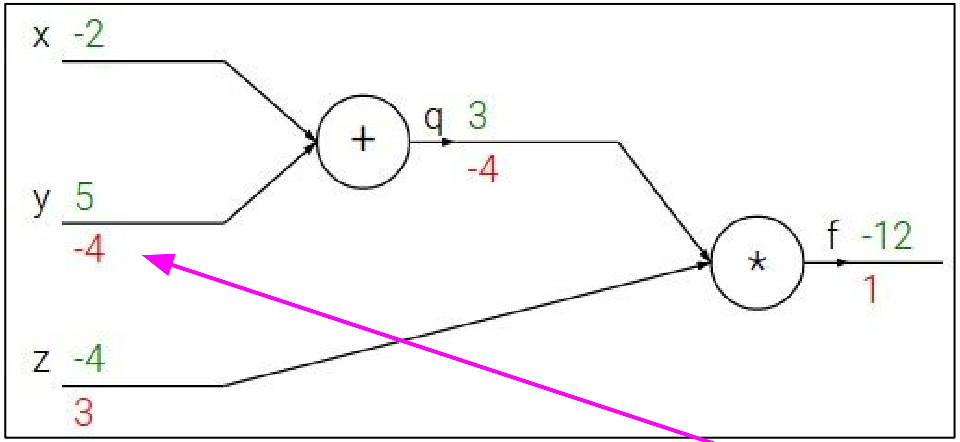
$$f(x,y,z) = (x+y)z$$

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:

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

Analytic gradient

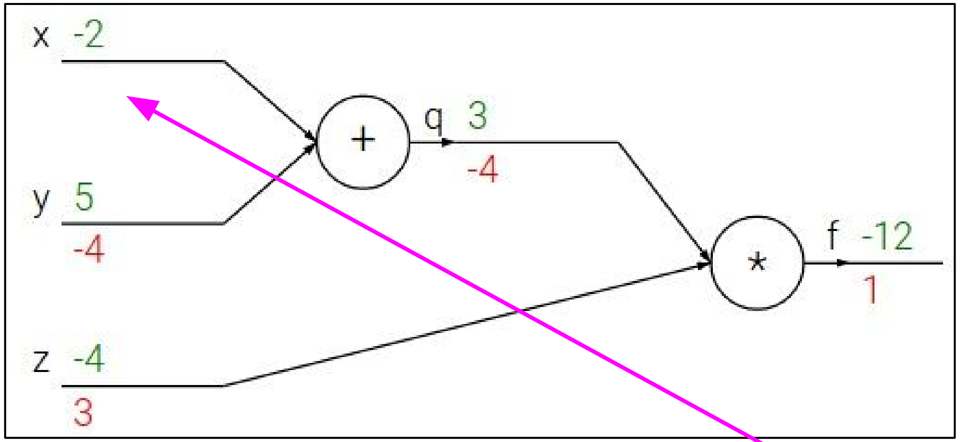
$$f(x,y,z) = (x+y)z$$

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



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

Analytic gradient

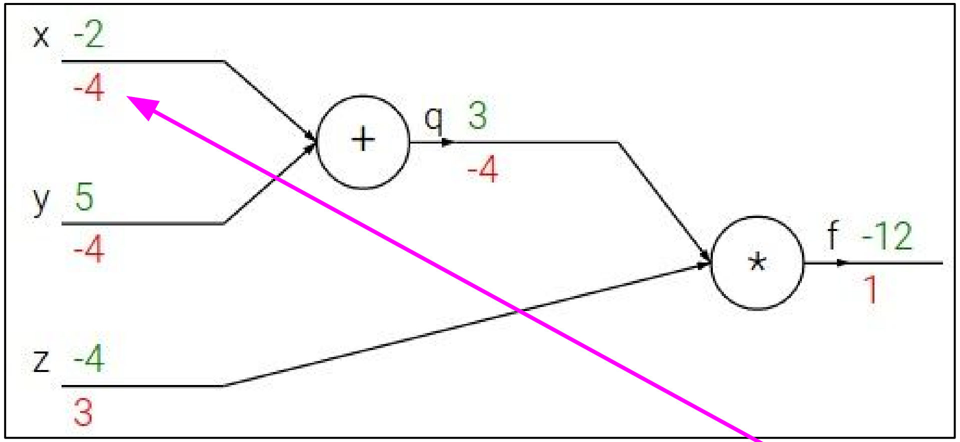
$$f(x, y, z) = (x + y)z$$

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



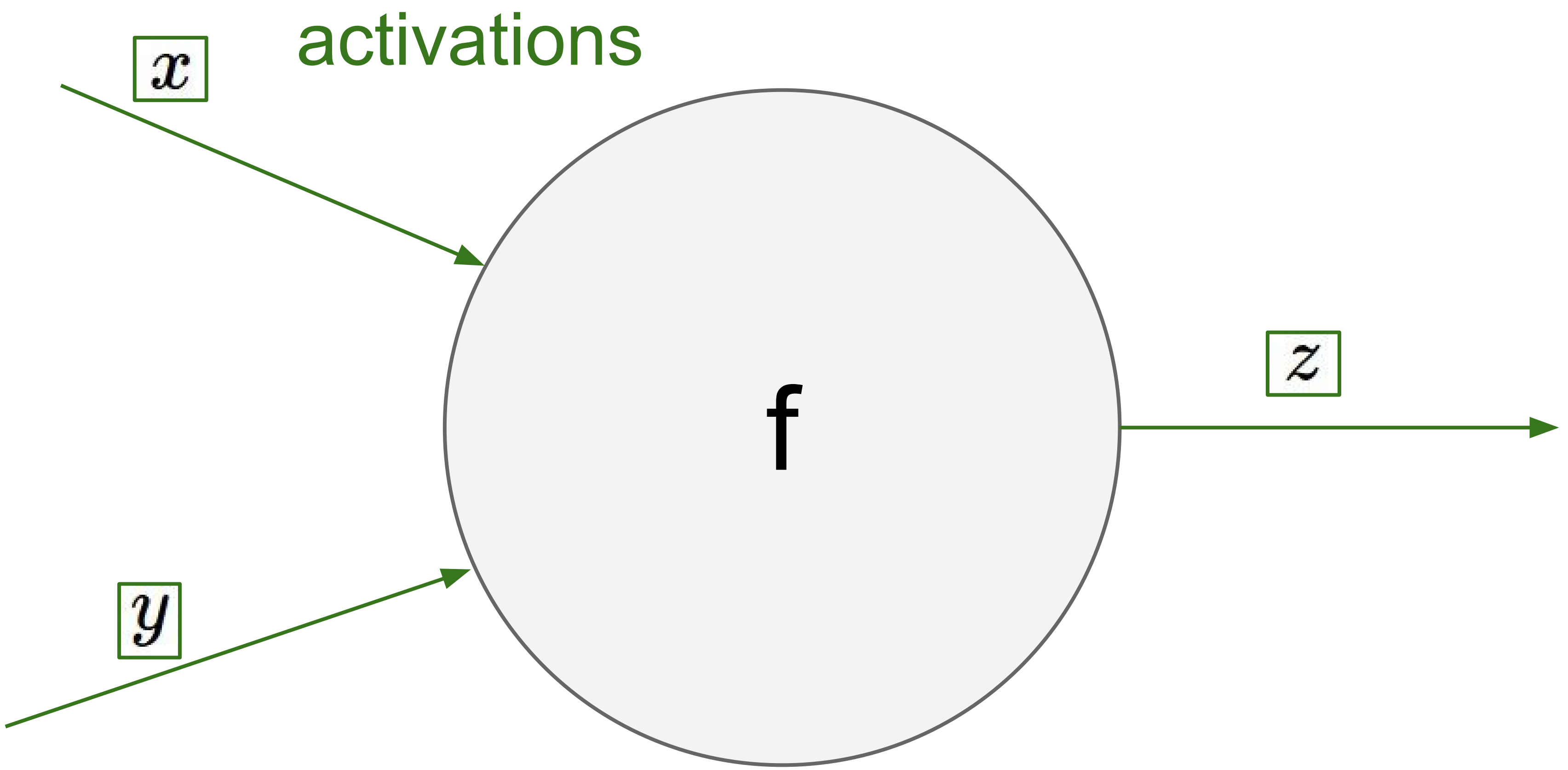
$$\frac{\partial f}{\partial x}$$

Chain rule:

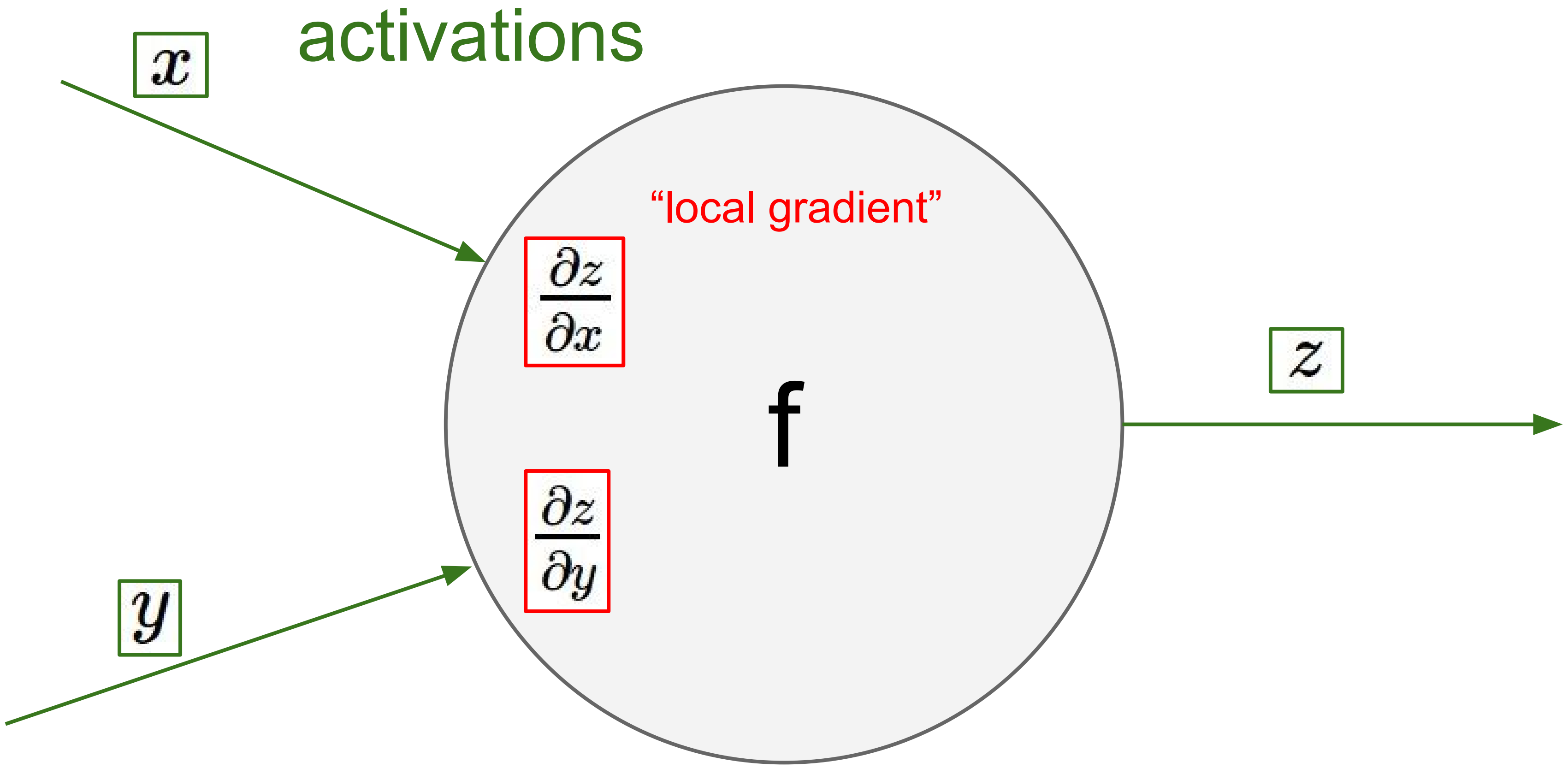
$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}$$

UNIVERSITY OF SURREY

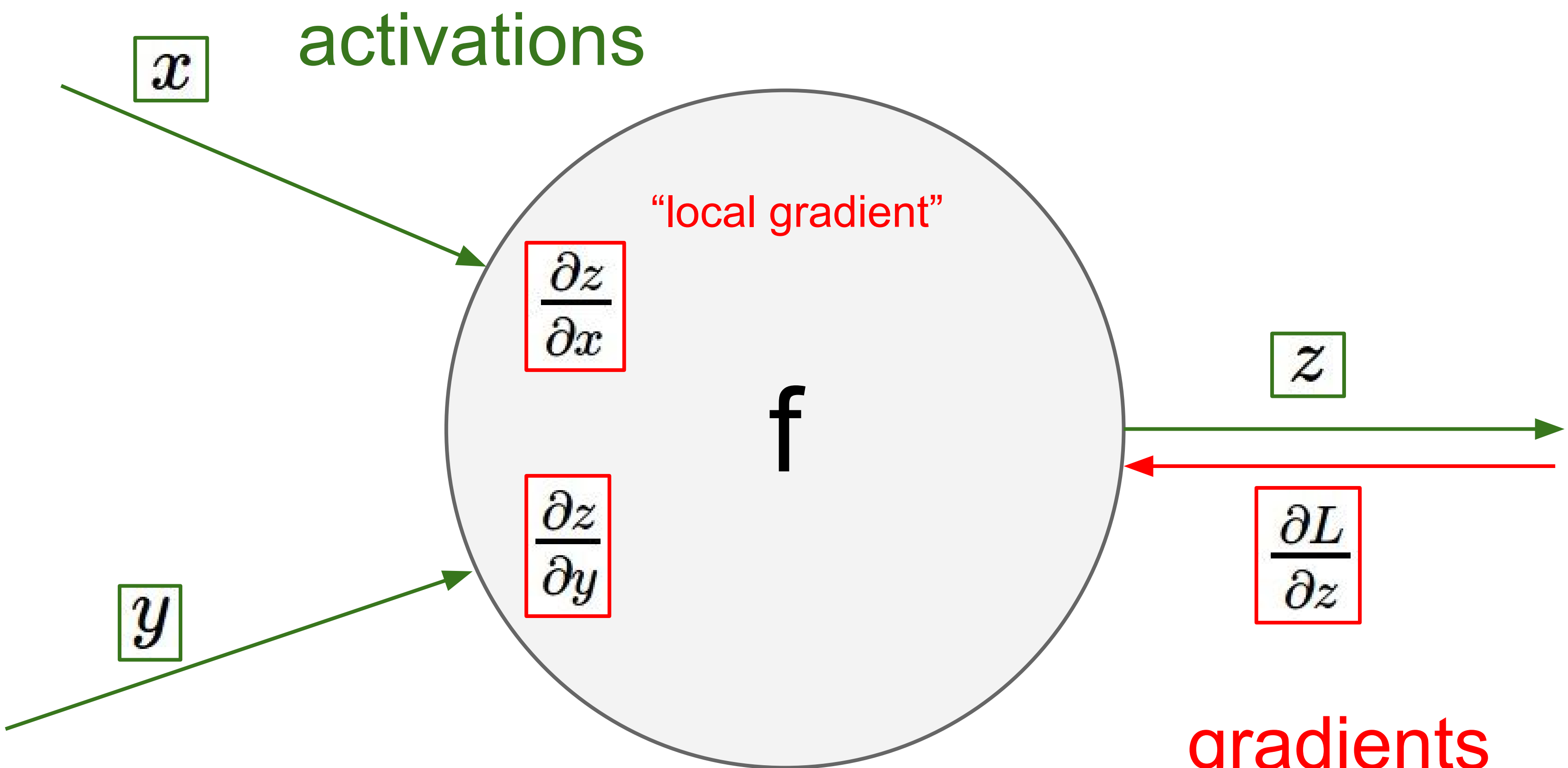
Analytic gradient in deep learning libraries



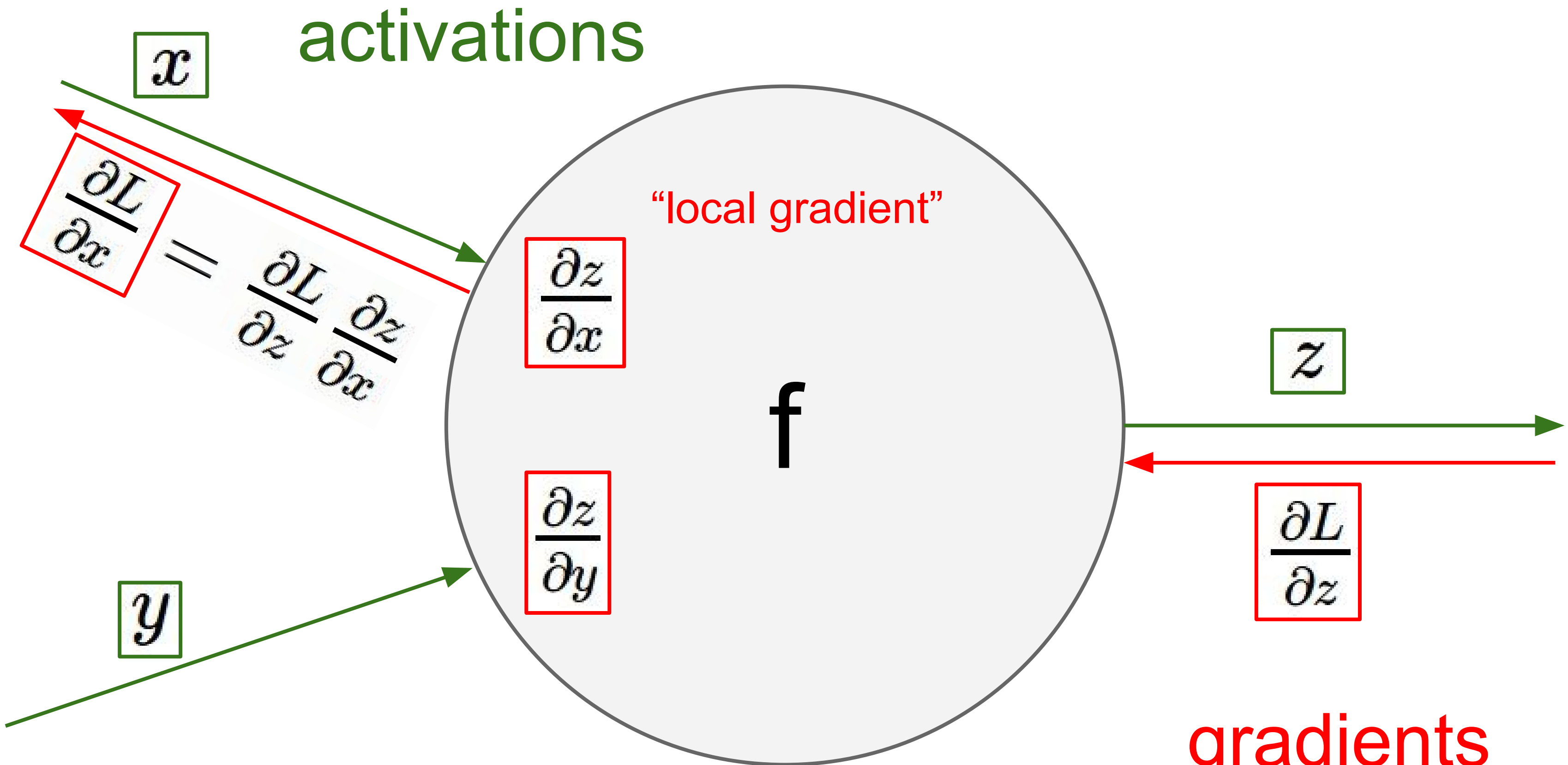
Analytic gradient in deep learning libraries



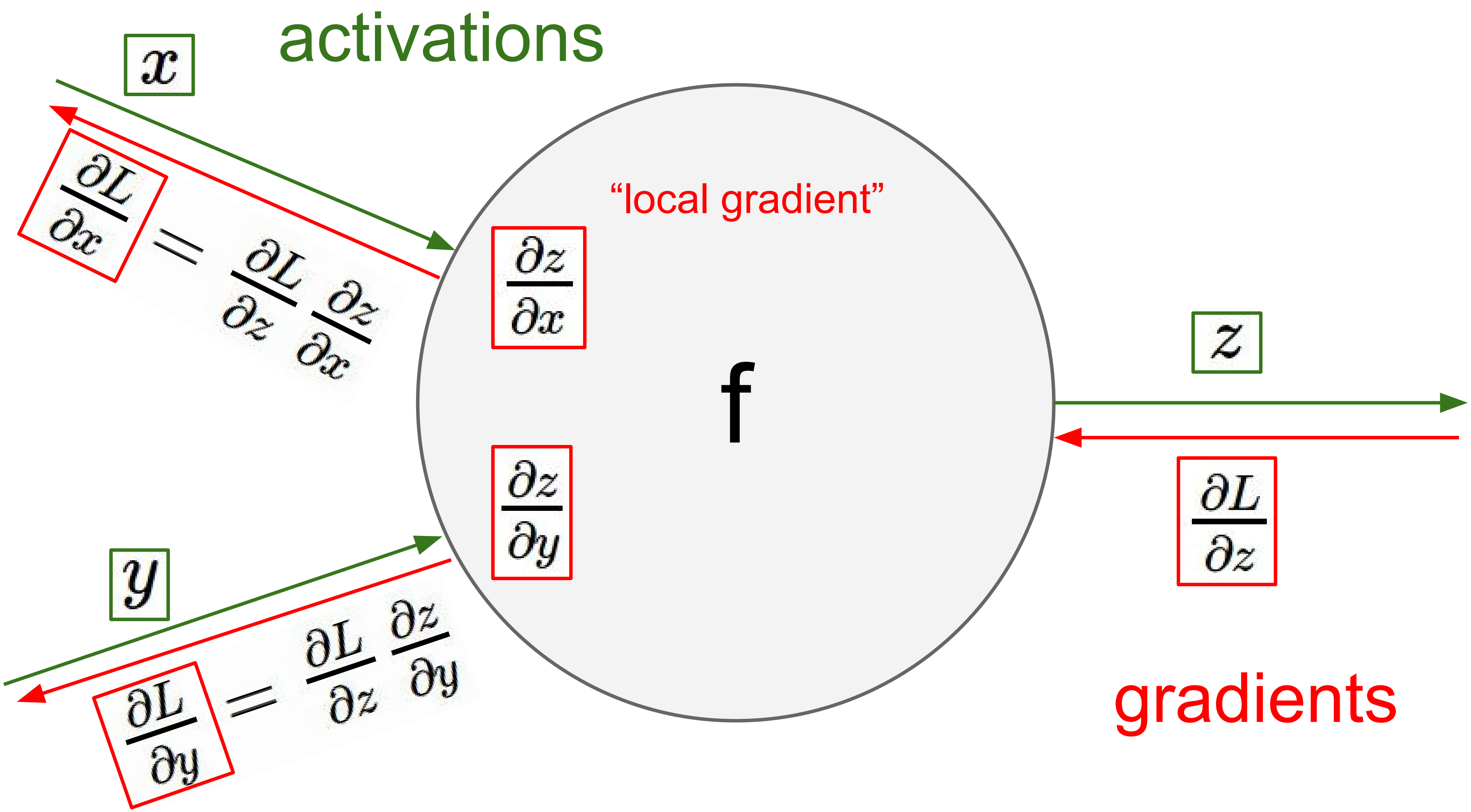
Analytic gradient in deep learning libraries



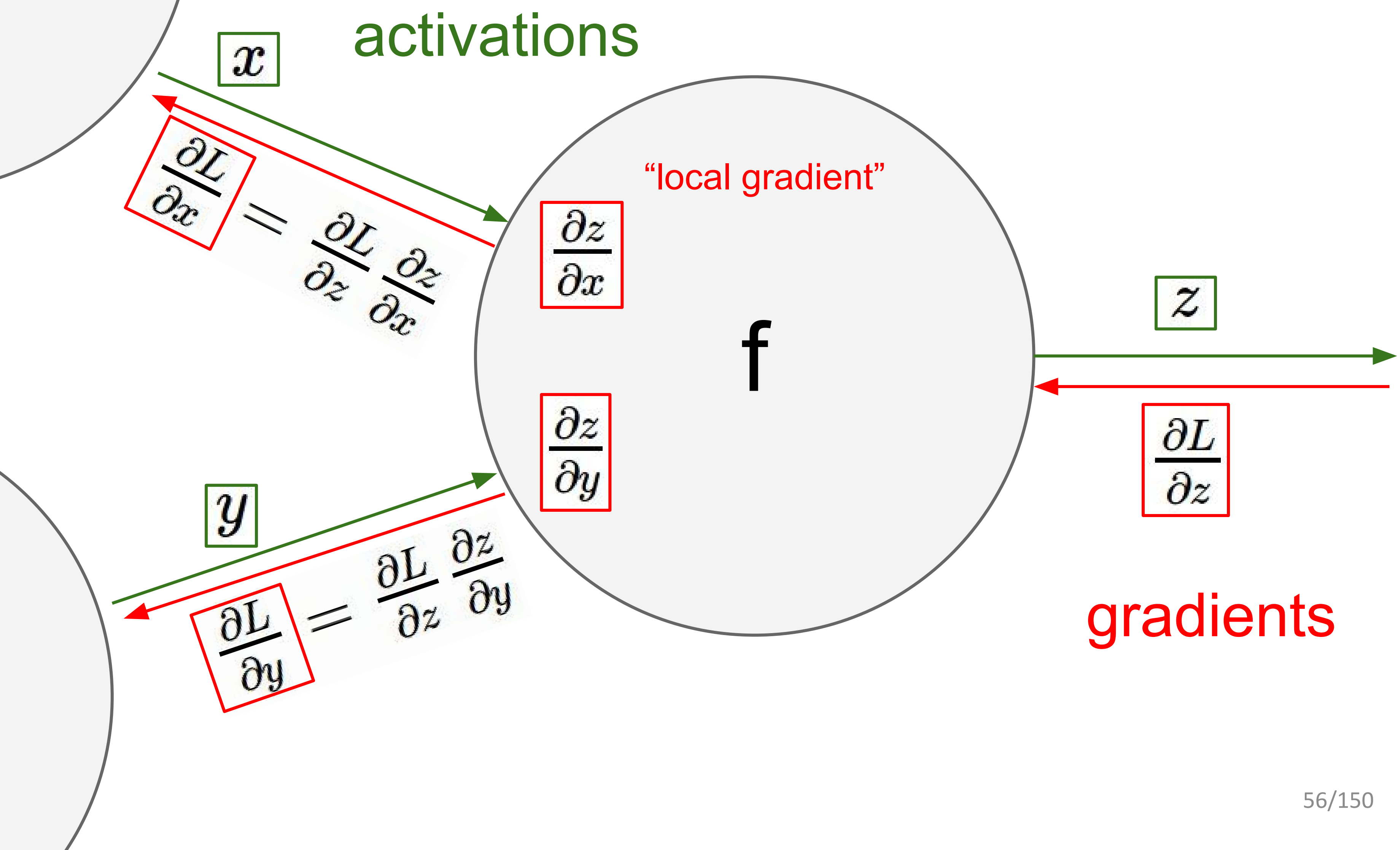
Analytic gradient in deep learning libraries



Analytic gradient in deep learning libraries



Analytic gradient in deep learning libraries

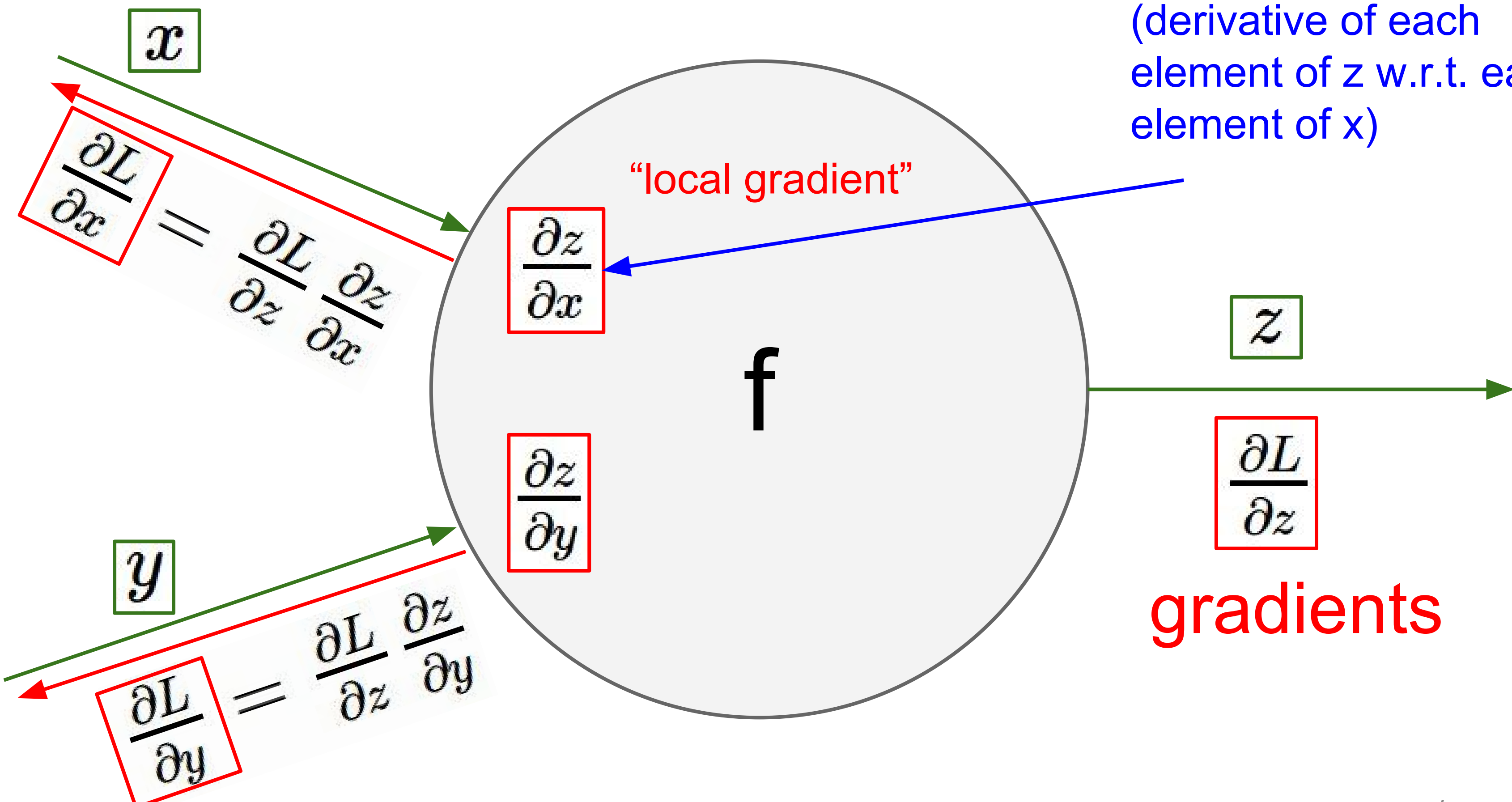


Analytic gradient in deep learning libraries

Gradients for vectorized code

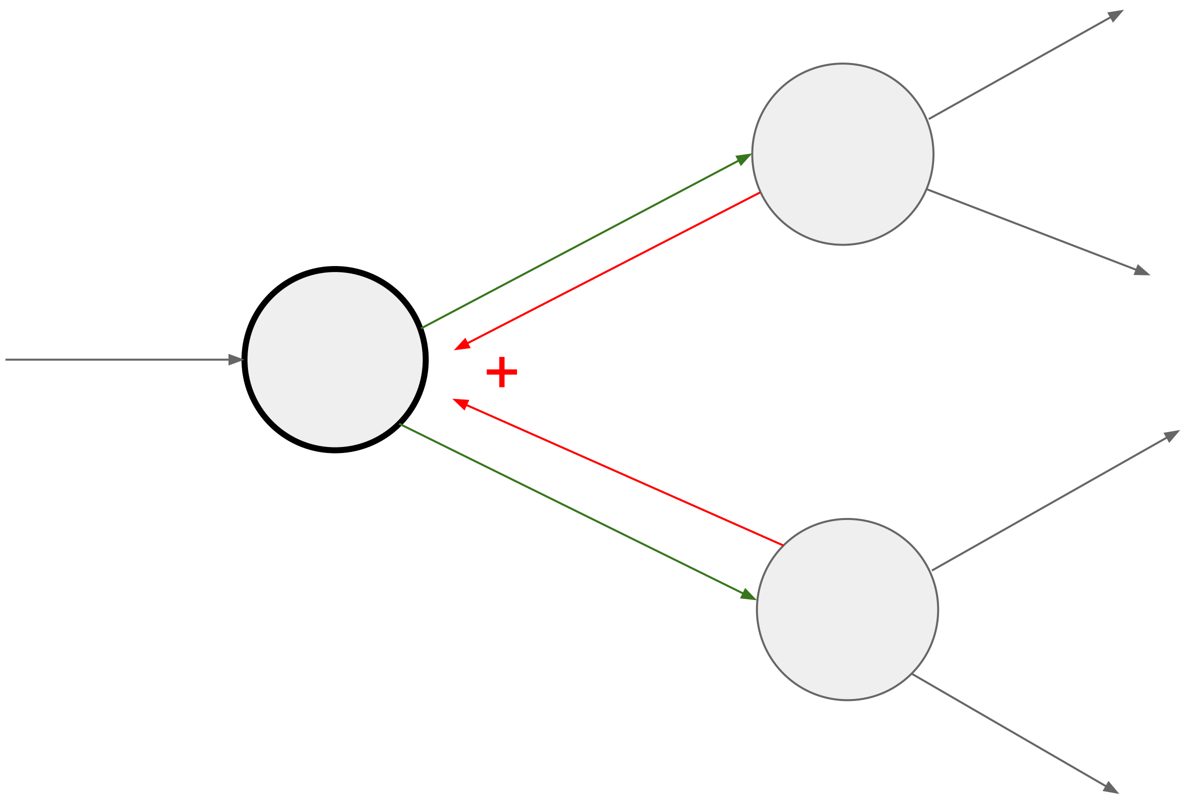
(x,y,z are now vectors)

This is now the **Jacobian matrix** (derivative of each element of z w.r.t. each element of x)



Analytic gradient in deep learning libraries

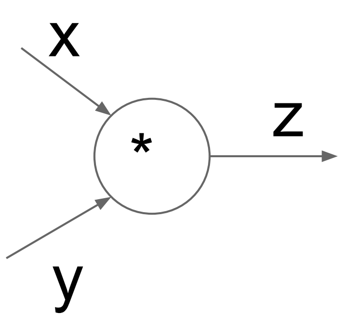
Gradients add at branches



UNIVERSITY OF SURREY

Modules Implementation in Deep Learning Libraries

Implementation: forward/backward API



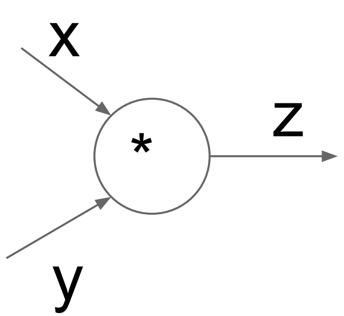
Graph (or Net) object. *(Rough psuedo code)*

```
class ComputationalGraph(object):
    #...
    def forward(inputs):
        # 1. [pass inputs to input gates...]
        # 2. forward the computational graph:
        for gate in self.graph.nodes_topologically_sorted():
            gate.forward()
        return loss # the final gate in the graph outputs the loss
    def backward():
        for gate in reversed(self.graph.nodes_topologically_sorted()):
            gate.backward() # little piece of backprop (chain rule applied)
        return inputs_gradients
```

UNIVERSITY OF SURREY

Modules Implementation in Deep Learning Libraries

Implementation: forward/backward API



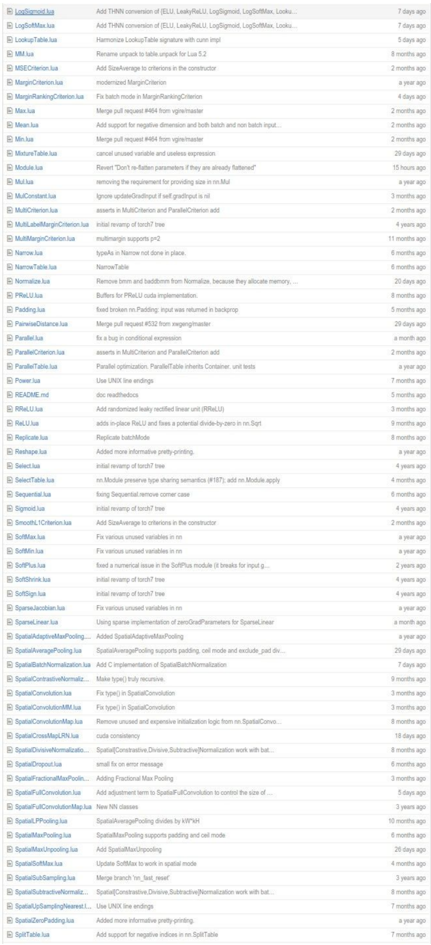
(x,y,z are scalars)

```
class MultiplyGate(object):
    def forward(x,y):
        z = x*y
        self.x = x # must keep these around!
        self.y = y
        return z
    def backward(dz):
        dx = self.y * dz # [dz/dx * dL/dz]
        dy = self.x * dz # [dz/dy * dL/dz]
        return [dx, dy]
```

[local gradient] x [gradient from top]

Modules Implementation in Deep Learning Libraries

Example: Torch Layers



UNIVERSITY OF SURREY

Modules Implementation in Deep Learning Libraries

```
local MulConstant, parent = torch.class('nn.MulConstant', 'nn.Module')

function MulConstant:__init(constant_scalar,ip)
  parent.__init(self)
  assert(type(constant_scalar) == 'number', 'input is not scalar!')
  self.constant_scalar = constant_scalar

  -- default for inplace is false
   self.inplace = ip or false
   if (ip and type(ip) ~= 'boolean') then
      error('in-place flag must be boolean')
   end
end

function MulConstant:updateOutput(input)
  if self.inplace then
    input:mul(self.constant_scalar)
    self.output = input
  else
    self.output:resizeAs(input)
    self.output:copy(input)
    self.output:mul(self.constant_scalar)
  end
  return self.output
end

function MulConstant:updateGradInput(input, gradOutput)
  if self.gradInput then
    if self.inplace then
      gradOutput:mul(self.constant_scalar)
      self.gradInput = gradOutput
      -- restore previous input value
      input:div(self.constant_scalar)
    else
      self.gradInput:resizeAs(gradOutput)
      self.gradInput:copy(gradOutput)
      self.gradInput:mul(self.constant_scalar)
    end
    return self.gradInput
  end
end
```

Example: Torch MulConstant

$$f(X) = aX$$

initialization

forward()

backward()

Modules Implementation in Deep Learning Libraries

Example: Caffe Layers

Branch: **master** caffe / src / caffe / **layers** / — New file | Find file | History

longjon enable dilated deconvolution … — Latest commit bbc4e57 16 days ago

..

absval_layer.cpp	dismantle layer headers	a month ago
absval_layer.cu	dismantle layer headers	a month ago
accuracy_layer.cpp	dismantle layer headers	a month ago
argmax_layer.cpp	dismantle layer headers	a month ago
base_conv_layer.cpp	enable dilated deconvolution	15 days ago
base_data_layer.cpp	dismantle layer headers	a month ago
base_data_layer.cu	dismantle layer headers	a month ago
batch_norm_layer.cpp	dismantle layer headers	a month ago
batch_norm_layer.cu	dismantle layer headers	a month ago
batch_reindex_layer.cpp	dismantle layer headers	a month ago
batch_reindex_layer.cu	dismantle layer headers	a month ago
bnll_layer.cpp	dismantle layer headers	a month ago
bnll_layer.cu	dismantle layer headers	a month ago
concat_layer.cpp	dismantle layer headers	a month ago
concat_layer.cu	dismantle layer headers	a month ago
contrastive_loss_layer.cpp	dismantle layer headers	a month ago
contrastive_loss_layer.cu	dismantle layer headers	a month ago
conv_layer.cpp	add support for 2D dilated convolution	15 days ago
conv_layer.cu	dismantle layer headers	a month ago
cudnn_conv_layer.cpp	dismantle layer headers	a month ago
cudnn_conv_layer.cu	Fix CuDNNConvolutionLayer for cuDNN v4	a month ago
cudnn_lcn_layer.cpp	dismantle layer headers	a month ago
cudnn_lcn_layer.cu	dismantle layer headers	a month ago
cudnn_lrn_layer.cpp	dismantle layer headers	a month ago
cudnn_lrn_layer.cu	dismantle layer headers	a month ago
cudnn_pooling_layer.cpp	dismantle layer headers	a month ago
cudnn_pooling_layer.cu	dismantle layer headers	a month ago
cudnn_relu_layer.cpp	dismantle layer headers	a month ago
cudnn_relu_layer.cu	dismantle layer headers	a month ago
cudnn_sigmoid_layer.cpp	dismantle layer headers	a month ago
cudnn_sigmoid_layer.cu	dismantle layer headers	a month ago
cudnn_softmax_layer.cpp	dismantle layer headers	a month ago
cudnn_softmax_layer.cu	dismantle layer headers	a month ago
cudnn_tanh_layer.cpp	dismantle layer headers	a month ago
cudnn_tanh_layer.cu	dismantle layer headers	a month ago
data_layer.cpp	dismantle layer headers	a month ago
deconv_layer.cpp	enable dilated deconvolution	15 days ago
deconv_layer.cu	dismantle layer headers	a month ago
dropout_layer.cpp	dismantle layer headers	a month ago
dropout_layer.cu	dismantle layer headers	a month ago
dummy_data_layer.cpp	dismantle layer headers	a month ago
eltwise_layer.cpp	dismantle layer headers	a month ago
eltwise_layer.cu	dismantle layer headers	a month ago
embed_layer.cpp	dismantle layer headers	a month ago
embed_layer.cu	dismantle layer headers	a month ago
euclidean_loss_layer.cpp	dismantle layer headers	a month ago
euclidean_loss_layer.cu	dismantle layer headers	a month ago
exp_layer.cpp	dismantle layer headers	a month ago
exp_layer.cu	dismantle layer headers	a month ago
filter_layer.cpp	dismantle layer headers	a month ago
filter_layer.cu	dismantle layer headers	a month ago
flatten_layer.cpp	dismantle layer headers	a month ago

memory_data_layer.cpp	dismantle layer headers	a month ago
multinomial_logistic_loss_la...	dismantle layer headers	a month ago
mvn_layer.cpp	dismantle layer headers	a month ago
mvn_layer.cu	dismantle layer headers	a month ago
neuron_layer.cpp	dismantle layer headers	a month ago
pooling_layer.cpp	dismantle layer headers	a month ago
pooling_layer.cu	dismantle layer headers	a month ago
power_layer.cpp	dismantle layer headers	a month ago
power_layer.cu	dismantle layer headers	a month ago
prelu_layer.cpp	dismantle layer headers	a month ago
prelu_layer.cu	dismantle layer headers	a month ago
reduction_layer.cpp	dismantle layer headers	a month ago
reduction_layer.cu	dismantle layer headers	a month ago
relu_layer.cpp	dismantle layer headers	a month ago
relu_layer.cu	dismantle layer headers	a month ago
reshape_layer.cpp	dismantle layer headers	a month ago
sigmoid_cross_entropy_los...	dismantle layer headers	a month ago
sigmoid_cross_entropy_los...	dismantle layer headers	a month ago
sigmoid_layer.cpp	dismantle layer headers	a month ago
sigmoid_layer.cu	dismantle layer headers	a month ago
silence_layer.cpp	dismantle layer headers	a month ago
silence_layer.cu	dismantle layer headers	a month ago
slice_layer.cpp	dismantle layer headers	a month ago
slice_layer.cu	dismantle layer headers	a month ago
softmax_layer.cpp	dismantle layer headers	a month ago
softmax_layer.cu	dismantle layer headers	a month ago
softmax_loss_layer.cpp	dismantle layer headers	a month ago
softmax_loss_layer.cu	dismantle layer headers	a month ago
split_layer.cpp	dismantle layer headers	a month ago
split_layer.cu	dismantle layer headers	a month ago
spp_layer.cpp	dismantle layer headers	a month ago
tanh_layer.cpp	dismantle layer headers	a month ago
tanh_layer.cu	dismantle layer headers	a month ago
threshold_layer.cpp	dismantle layer headers	a month ago
threshold_layer.cu	dismantle layer headers	a month ago
tile_layer.cpp	dismantle layer headers	a month ago
tile_layer.cu	dismantle layer headers	a month ago
window_data_layer.cpp	dismantle layer headers	a month ago

Modules Implementation in Deep Learning Libraries

Caffe Sigmoid Layer

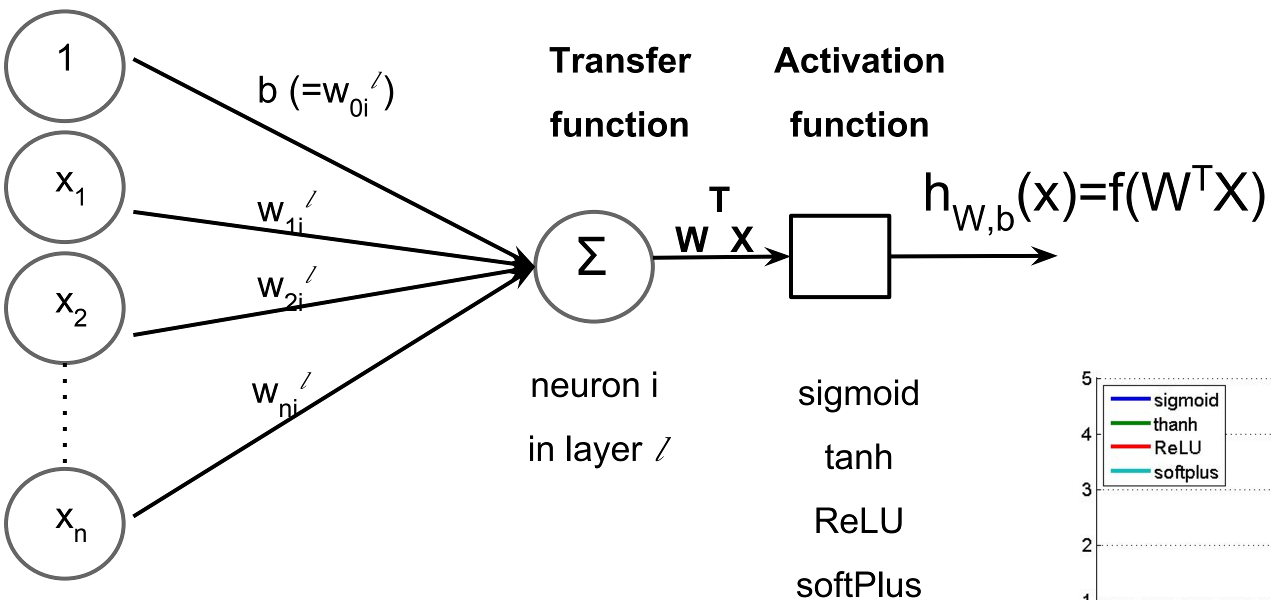
```
#include <cmath>
#include <vector>

#include "caffe/layers/sigmoid_layer.hpp"

namespace caffe {

template <typename Dtype>
inline Dtype sigmoid(Dtype x) {
  return 1. / (1. + exp(-x));
}

template <typename Dtype>
void SigmoidLayer<Dtype>::Forward_cpu(const vector<Blob<Dtype>*>& bottom,
    const vector<Blob<Dtype>*>& top) {
  const Dtype* bottom_data = bottom[0]->cpu_data();
  Dtype* top_data = top[0]->mutable_cpu_data();
  const int count = bottom[0]->count();
  for (int i = 0; i < count; ++i) {
    top_data[i] = sigmoid(bottom_data[i]);
  }
}

template <typename Dtype>
void SigmoidLayer<Dtype>::Backward_cpu(const vector<Blob<Dtype>*>& top,
    const vector<bool>& propagate_down,
    const vector<Blob<Dtype>*>& bottom) {
  if (propagate_down[0]) {
    const Dtype* top_data = top[0]->cpu_data();
    const Dtype* top_diff = top[0]->cpu_diff();
    Dtype* bottom_diff = bottom[0]->mutable_cpu_diff();
    const int count = bottom[0]->count();
    for (int i = 0; i < count; ++i) {
      const Dtype sigmoid_x = top_data[i];
      bottom_diff[i] = top_diff[i] * sigmoid_x * (1. - sigmoid_x);
    }
  }
}

#ifdef CPU_ONLY
STUB_GPU(SigmoidLayer);
#endif

INSTANTIATE_CLASS(SigmoidLayer);


}  // namespace caffe
```

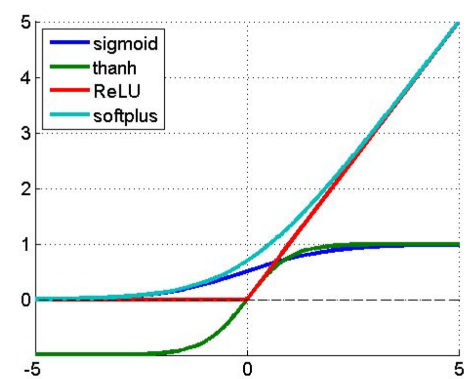
$$\boxed{\sigma(x) = \frac{1}{1+e^{-x}}}$$

$$\boxed{(1-\sigma(x))\sigma(x)}$$ *top_diff   (chain rule)

Neuron Model

Simple Neuron model

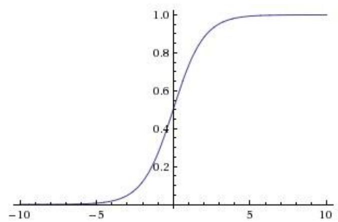




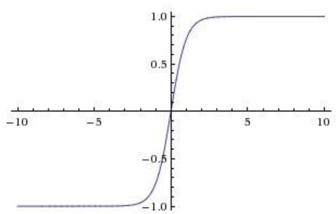
Activation Functions

Sigmoid

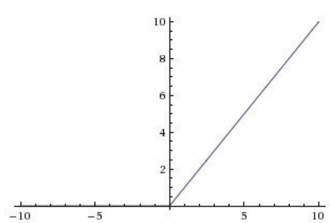
$$\sigma(x) = 1/(1 + e^{-x})$$



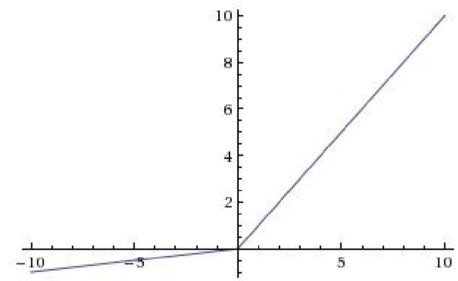
tanh tanh(x)



ReLU max(0,x)

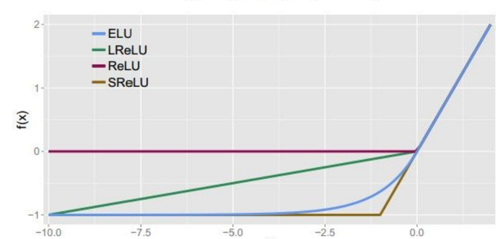


Leaky ReLU
max(0.1x, x)

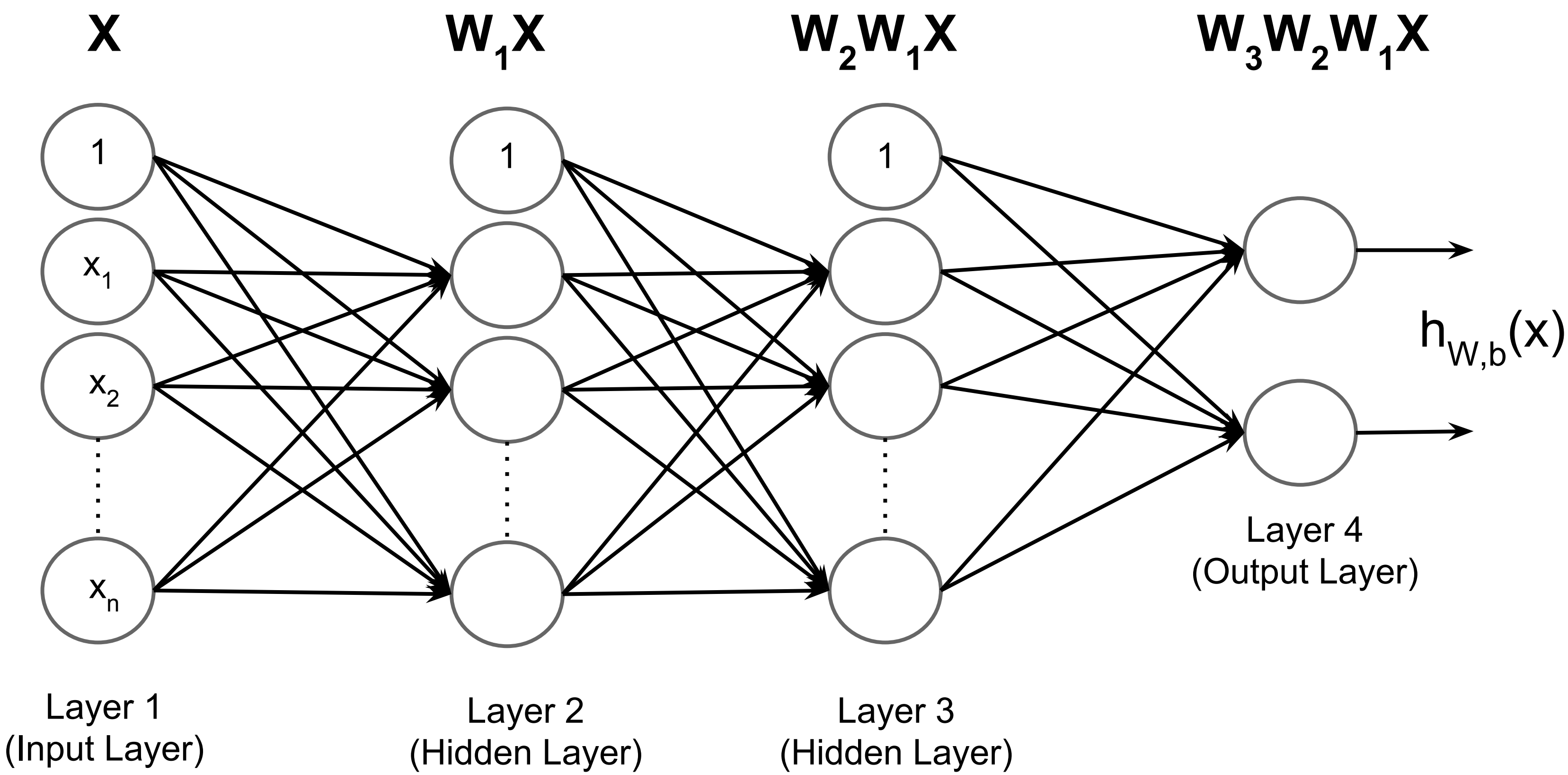


Maxout $\max(w_1^T x + b_1, w_2^T x + b_2)$

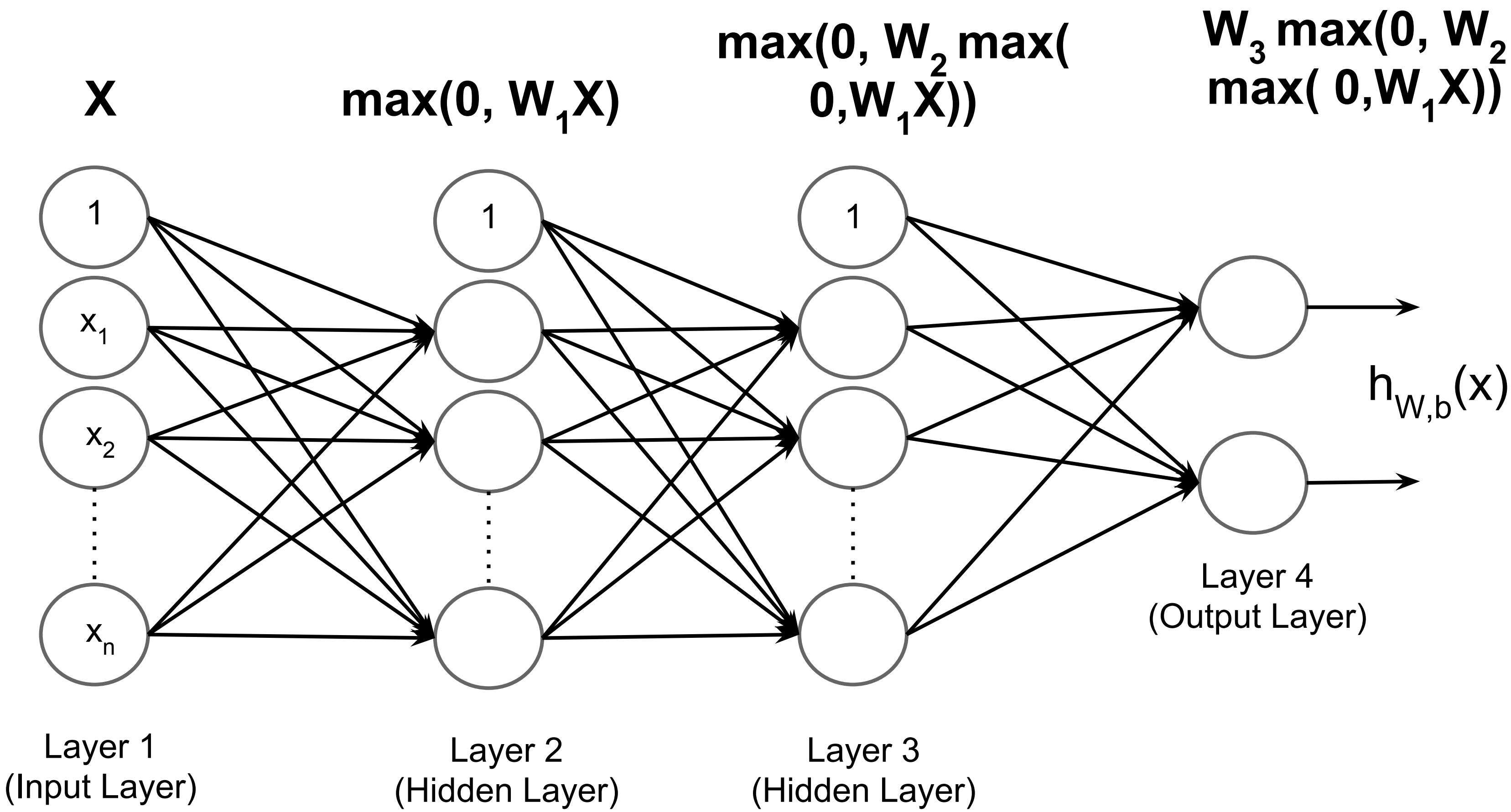
ELU $f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha\,(\exp(x) - 1) & \text{if } x \le 0 \end{cases}$



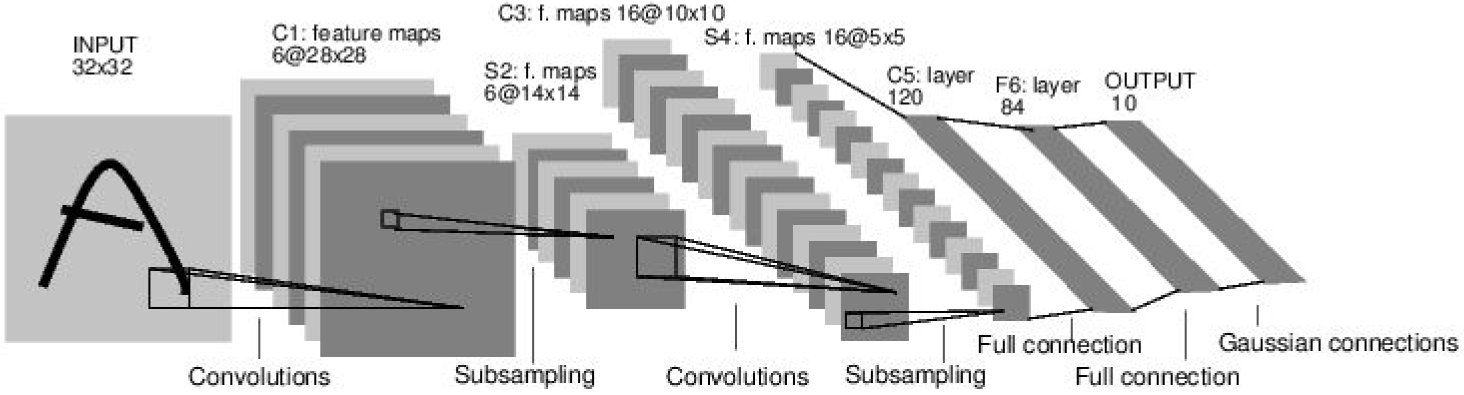
Multi-Layer Neural Networks



Multi-Layer Neural Networks with ReLU

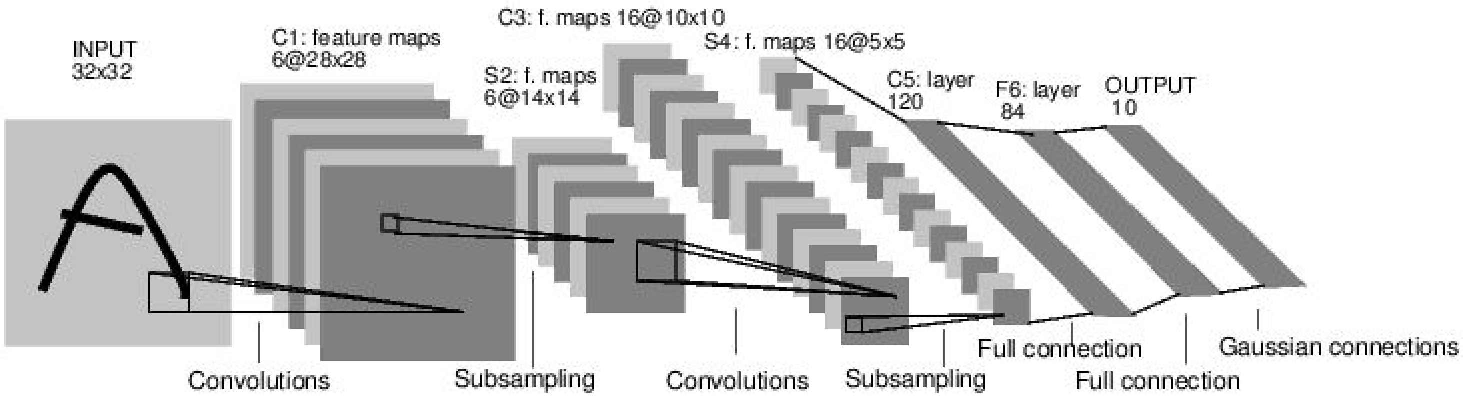


Convolutional Neural Networks

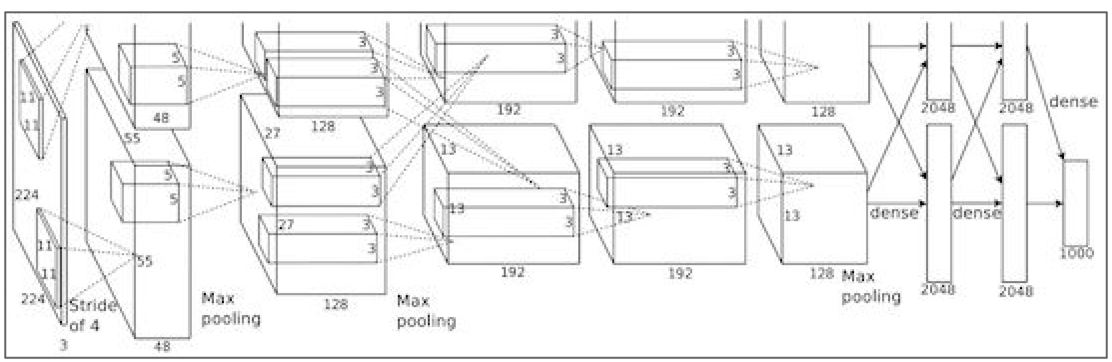


[LeNet-5, LeCun 1980]

Convolutional Neural Networks



[LeNet-5, LeCun 1980]



"AlexNet" [Krizhevsky, Sutskever, Hinton, 2012]

Convolutional Neural Networks

Image

1	2	1	2	1	0	0
0	1	2	1	1	0	1
0	0	1	0	1	1	0
1	0	0	0	1	0	1
2	0	1	0	1	2	2
0	2	1	0	1	0	1
2	2	2	0	0	1	1

Convolved feature

8				

7x7 input (spatially)
assume 3x3 filter

Convolutional Neural Networks

Image

1	2	1	2	1	0	0
0	1	2	1	1	0	1
0	0	1	0	1	1	0
1	0	0	0	1	0	1
2	0	1	0	1	2	2
0	2	1	0	1	0	1
2	2	2	0	0	1	1

Convolved feature

8	10			

7x7 input (spatially)
assume 3x3 filter

Convolutional Neural Networks

Image

1	2	1	2	1	0	0
0	1	2	1	1	0	1
0	0	1	0	1	1	0
1	0	0	0	1	0	1
2	0	1	0	1	2	2
0	2	1	0	1	0	1
2	2	2	0	0	1	1

Convolved feature

8	10	10		

7x7 input (spatially)
assume 3x3 filter

Convolutional Neural Networks

Image

1	2	1	2	1	0	0
0	1	2	1	1	0	1
0	0	1	0	1	1	0
1	0	0	0	1	0	1
2	0	1	0	1	2	2
0	2	1	0	1	0	1
2	2	2	0	0	1	1

Convolved feature

8	10	10	7	5
5				

7x7 input (spatially)
assume 3x3 filter

Convolutional Neural Networks

Image

1	2	1	2	1	0	0
0	1	2	1	1	0	1
0	0	1	0	1	1	0
1	0	0	0	1	0	1
2	0	1	0	1	2	2
0	2	1	0	1	0	1
2	2	2	0	0	1	1

Convolved feature

8	10	10	7	5
5	5	7	5	6
5	2	5	6	9
7	4	5	5	9
12	8	6	5	9

7x7 input (spatially)
assume 3x3 filter

Convolutional Neural Networks

Image

1	2	1	2	1	0	0
0	1	2	1	1	0	1
0	0	1	0	1	1	0
1	0	0	0	1	0	1
2	0	1	0	1	2	2
0	2	1	0	1	0	1
2	2	2	0	0	1	1

Convolved feature

8	10	10	7	5
5	5	7	5	6
5	2	5	6	9
7	4	5	5	9
12	8	6	5	9

7x7 input (spatially)
assume 3x3 filter

=> 5x5 output

Convolutional Neural Networks

Image

1	2	1	2	1	0	0
0	1	2	1	1	0	1
0	0	1	0	1	1	0
1	0	0	0	1	0	1
2	0	1	0	1	2	2
0	2	1	0	1	0	1
2	2	2	0	0	1	1

Convolved feature

8	10	5
5	5	9
12	6	9

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

Convolutional Neural Networks

Image

1	2	1	2	1	0	0
0	1	2	1	1	0	1
0	0	1	0	1	1	0
1	0	0	0	1	0	1
2	0	1	0	1	2	2
0	2	1	0	1	0	1
2	2	2	0	0	1	1

Convolved feature

8	10	5
5	5	9
12	6	9

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

Convolutional Neural Networks

Image

1	2	1	2	1	0	0
0	1	2	1	1	0	1
0	0	1	0	1	1	0
1	0	0	0	1	0	1
2	0	1	0	1	2	2
0	2	1	0	1	0	1
2	2	2	0	0	1	1

Convolved feature

8	10	5
5	5	9
12	6	9

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

Convolutional Neural Networks

Image

1	2	1	2	1	0	0
0	1	2	1	1	0	1
0	0	1	0	1	1	0
1	0	0	0	1	0	1
2	0	1	0	1	2	2
0	2	1	0	1	0	1
2	2	2	0	0	1	1

Convolved feature

8	10	5
5	5	9
12	6	9

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

Convolutional Neural Networks

1	2	1	2	1	0	0
0	1	2	1	1	0	1
0	0	1	0	1	1	0
1	0	0	0	1	0	1
2	0	1	0	1	2	2
0	2	1	0	1	0	1
2	2	2	0	0	1	1

8	10	5
5	5	9
12	6	9

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

=> 3x3 output

UNIVERSITY OF SURREY

Convolutional Neural Networks

F

F

1	2	1	2	1	0	0
0	1	2	1	1	0	1
0	0	1	0	1	1	0
1	0	0	0	1	0	1
2	0	1	0	1	2	2
0	2	1	0	1	0	1
2	2	2	0	0	1	1

N

N

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 :\

UNIVERSITY OF SURREY

Convolutional Neural Networks

0	0	0	0	0	0			
0	1	2	1	2	1	0	0	
0	0	1	2	1	1	0	1	
0	0	0	1	0	1	1	0	
0	1	0	0	0	1	0	1	
	2	0	1	0	1	2	2	
	0	2	1	0	1	0	1	
	2	2	2	0	0	1	1	

In practice: Common to zero pad the border

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

Convolutional Neural Networks

0	0	0	0	0	0			
0	1	2	1	2	1	0	0	
0	0	1	2	1	1	0	1	
0	0	0	1	0	1	1	0	
0	1	0	0	0	1	0	1	
	2	0	1	0	1	2	2	
	0	2	1	0	1	0	1	
	2	2	2	0	0	1	1	

4	7	9	8	5	3	1
4	8	10	10	7	5	2
2	5	5	7	5	6	3
3	5	2	5	6	9	6
5	7	4	5	5	9	6
8	12	8	6	5	9	7
6	9	7	4	2	4	3

=> 7x7 output

UNIVERSITY OF SURREY

Convolutional Neural Networks

0	0	0	0	0	0			
0	1	2	1	2	1	0	0	
0	0	1	2	1	1	0	1	
0	0	0	1	0	1	1	0	
0	1	0	0	0	1	0	1	
	2	0	1	0	1	2	2	
	0	2	1	0	1	0	1	
	2	2	2	0	0	1	1	

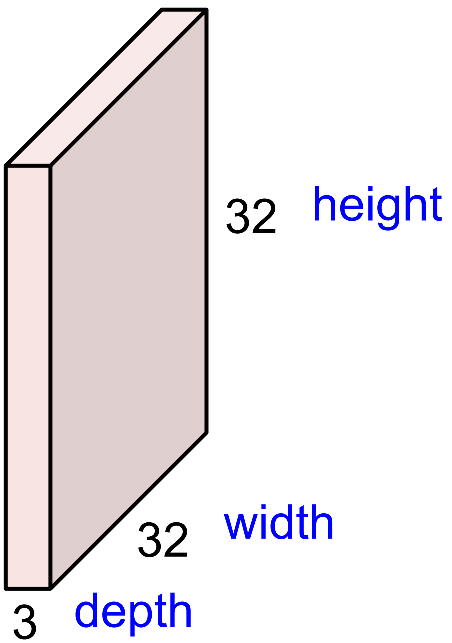
In practice: Common to zero pad the border

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

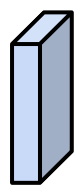
Convolutional Neural Networks

Filters always extend the full depth of the input volume

32x32x3 image

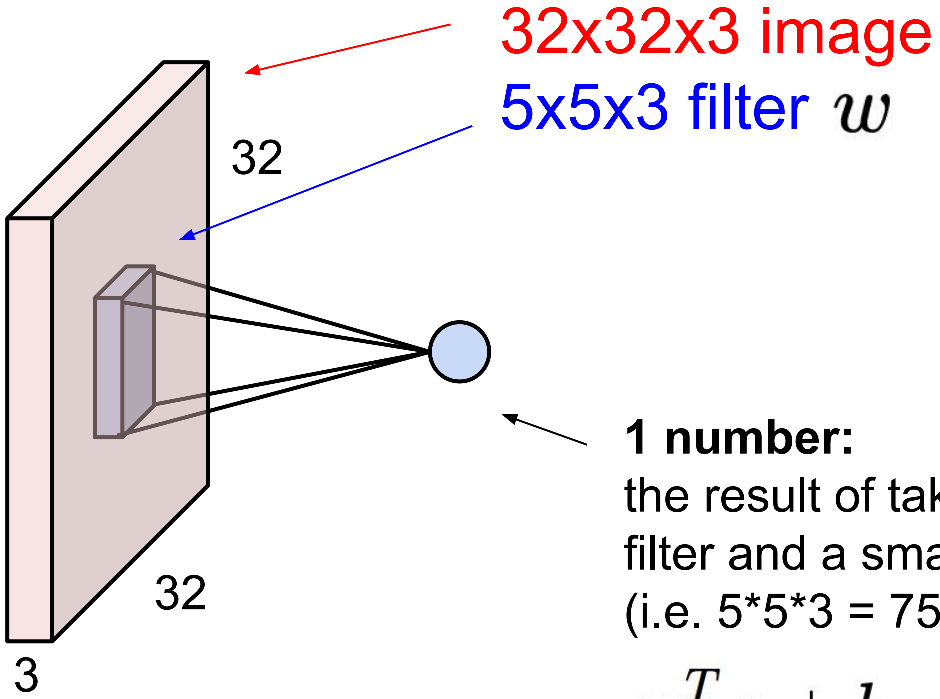


5x5x3 filter



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

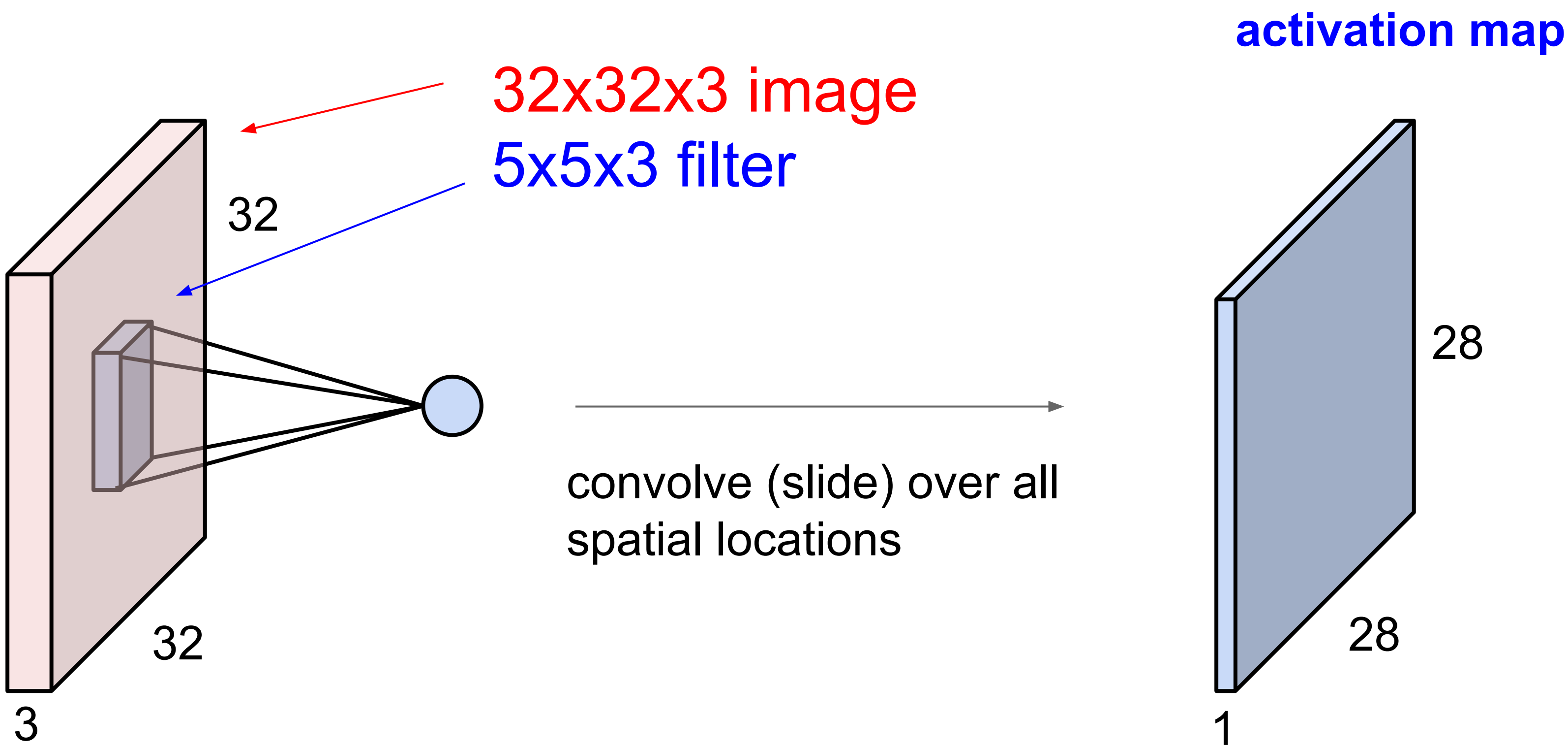
Convolutional Neural Networks



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)

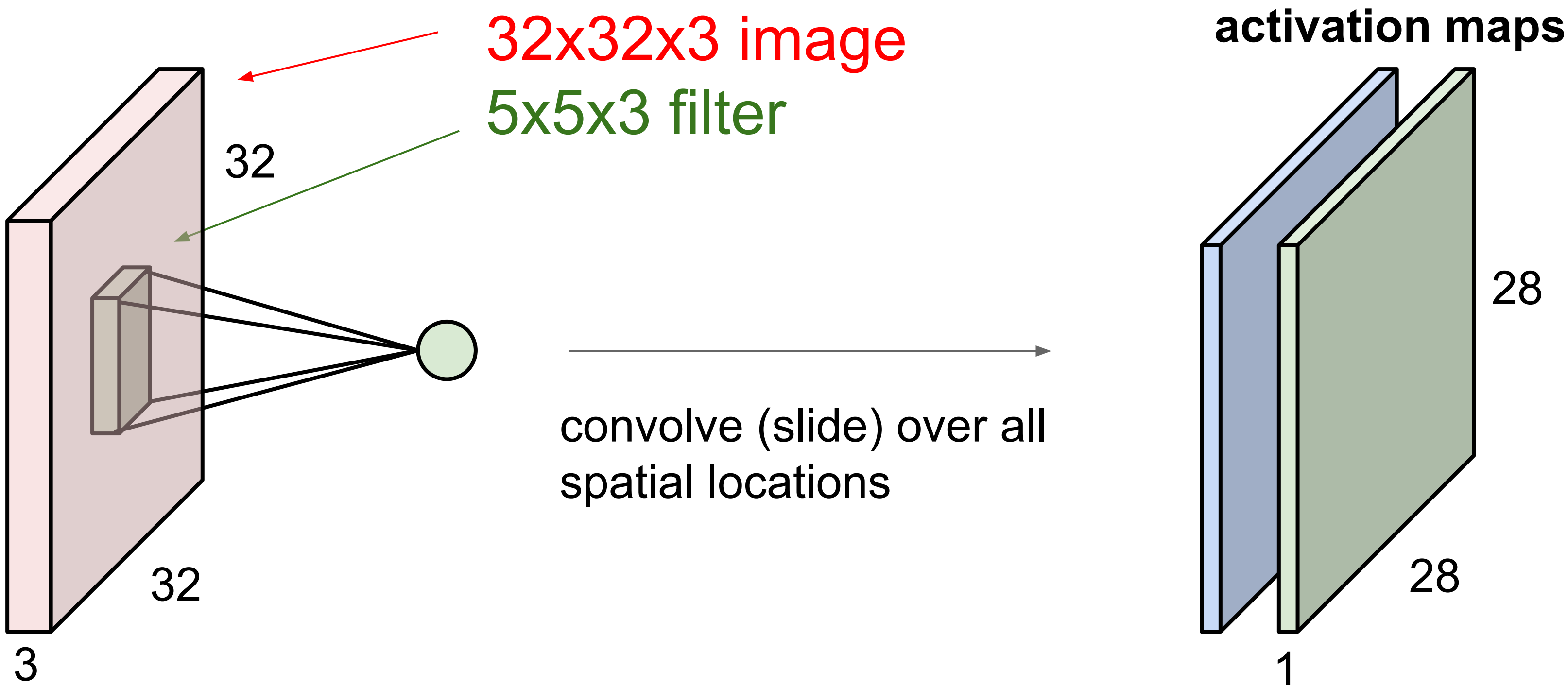
$$w^T x + b$$

Convolutional Neural Networks



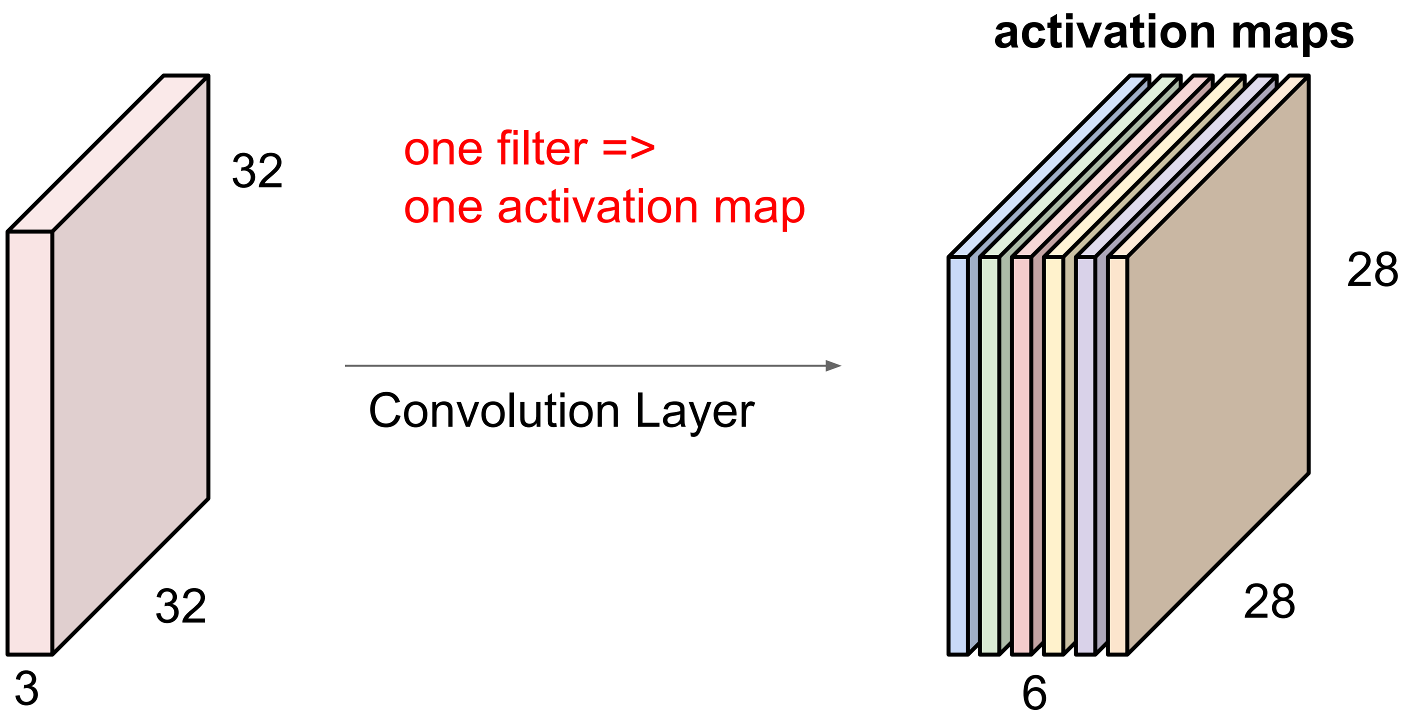
Convolutional Neural Networks

consider a second, green filter



Convolutional Neural Networks

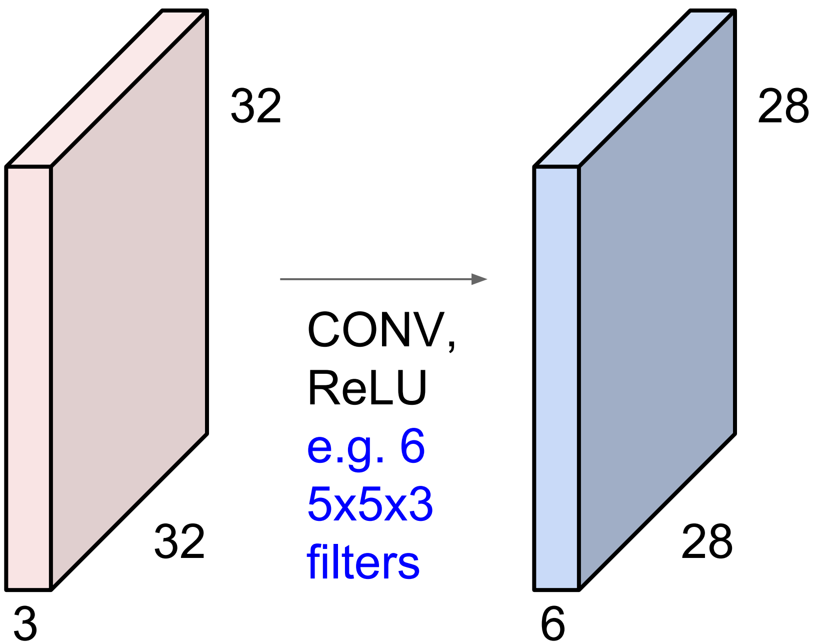
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



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

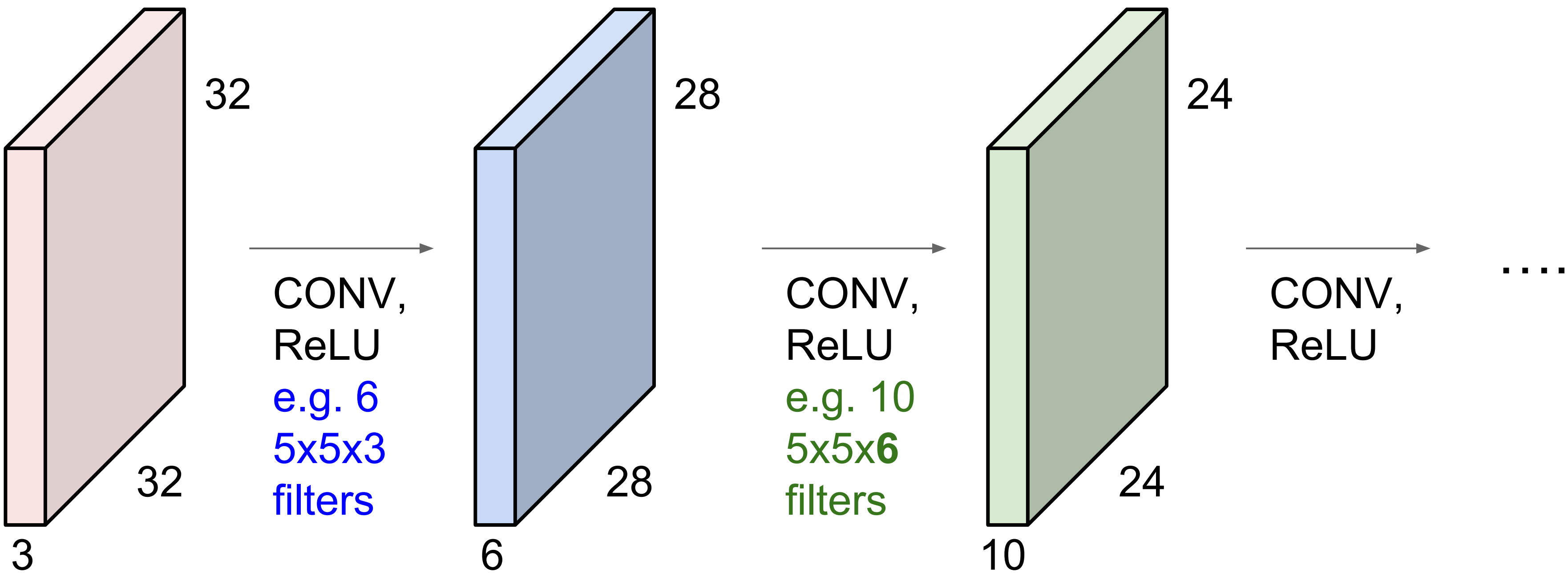
Convolutional Neural Networks

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

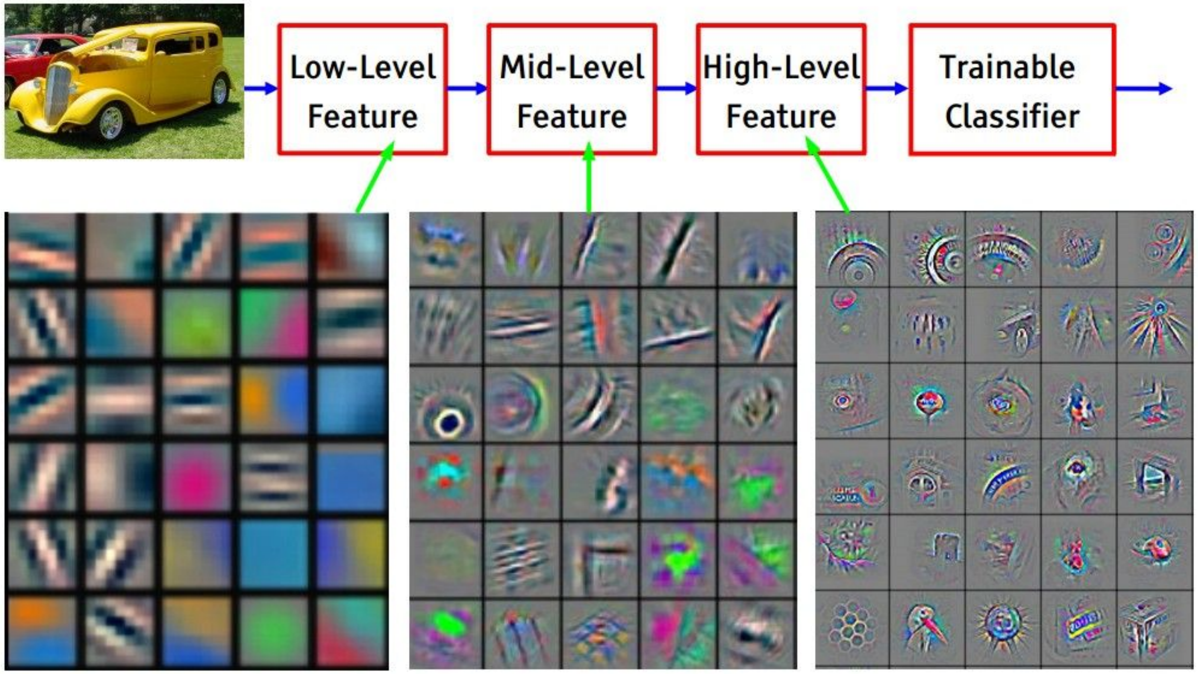


Convolutional Neural Networks

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

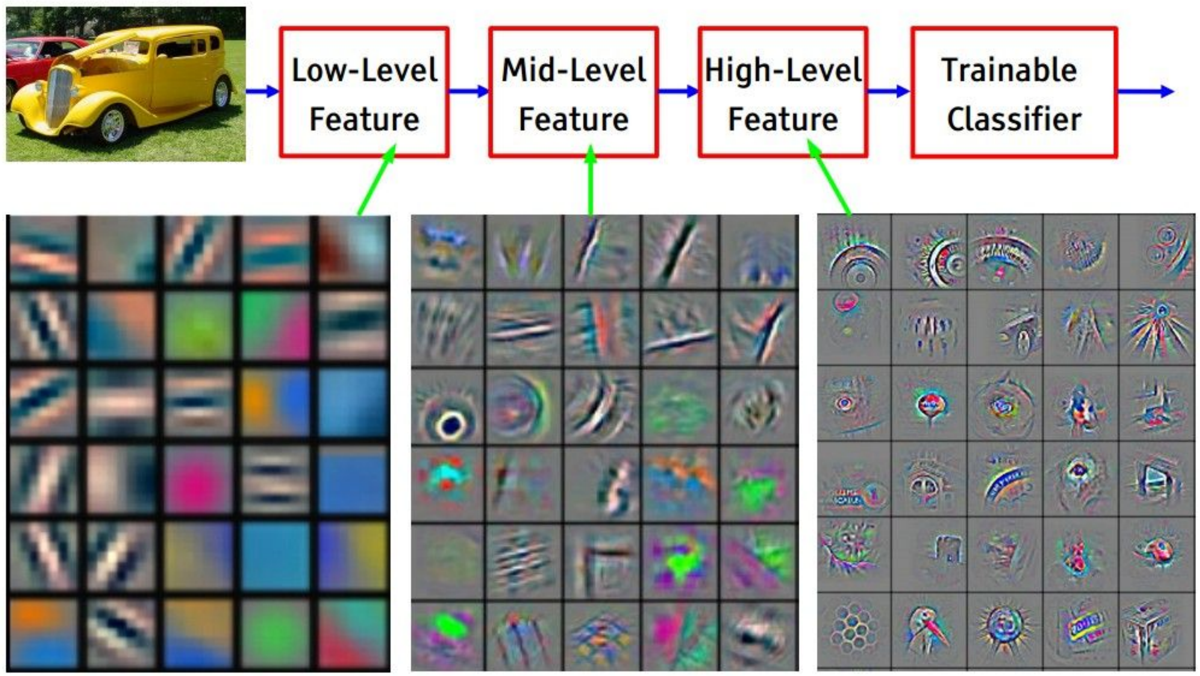


Convolutional Neural Networks

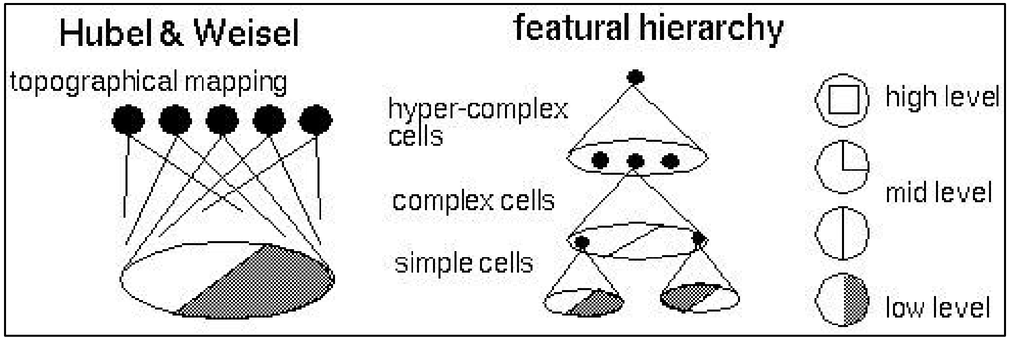


Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

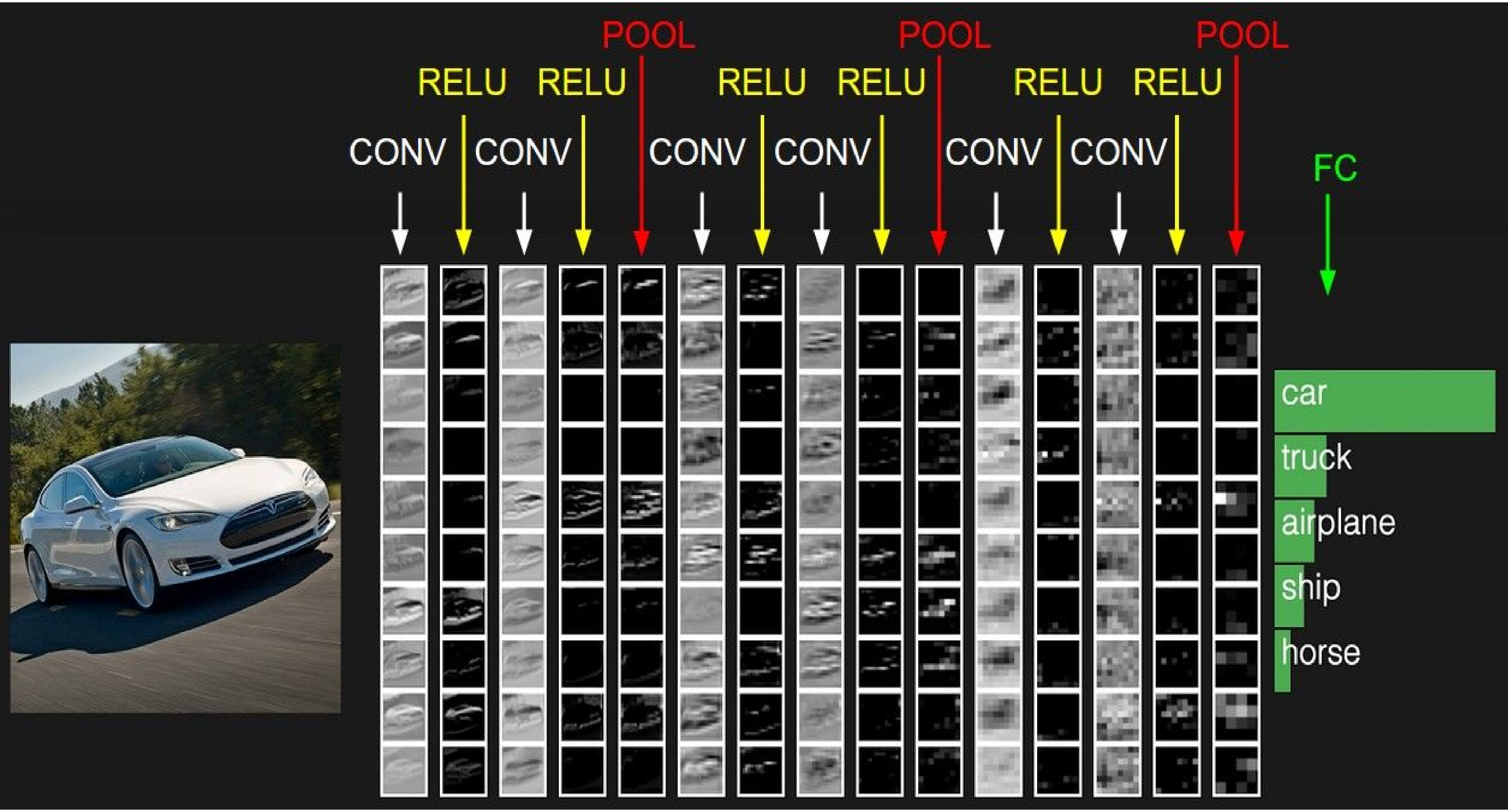
Convolutional Neural Networks



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]



Convolutional Neural Networks



Convolutional Neural Networks

Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

Convolutional Neural Networks

Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

Common settings:

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

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

UNIVERSITY OF SURREY

Convolutional Neural Networks

Convolutional layer in Torch

SpatialConvolution

```
module = nn.SpatialConvolution(nInputPlane, nOutputPlane, kW, kH, [dW], [dH], [padW], [padH])
```

Applies a 2D convolution over an input image composed of several input planes. The `input` tensor in `forward(input)` is expected to be a 3D tensor (`nInputPlane x height x width`).

The parameters are the following:

- `nInputPlane` : The number of expected input planes in the image given into `forward()` .
- `nOutputPlane` : The number of output planes the convolution layer will produce.
- `kW` : The kernel width of the convolution
- `kH` : The kernel height of the convolution
- `dW` : The step of the convolution in the width dimension. Default is `1` .
- `dH` : The step of the convolution in the height dimension. Default is `1` .
- `padW` : The additional zeros added per width to the input planes. Default is `0` , a good number is `(kW-1)/2` .
- `padH` : The additional zeros added per height to the input planes. Default is `padW` , a good number is `(kH-1)/2` .

Note that depending of the size of your kernel, several (of the last) columns or rows of the input image might be lost. It is up to the user to add proper padding in images.

If the input image is a 3D tensor `nInputPlane x height x width` , the output image size will be `nOutputPlane x oheight x owidth` where

```
owidth  = floor((width  + 2*padW - kW) / dW + 1)
oheight = floor((height + 2*padH - kH) / dH + 1)
```

Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.

Convolutional Neural Networks

Convolutional layer in Torch

```
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  # learning rate and decay multipliers for the filters
  param { lr_mult: 1 decay_mult: 1 }
  # learning rate and decay multipliers for the biases
  param { lr_mult: 2 decay_mult: 0 }
  convolution_param {
    num_output: 96     # learn 96 filters
    kernel_size: 11    # each filter is 11x11
    stride: 4          # step 4 pixels between each filter application
    weight_filler {
      type: "gaussian" # initialize the filters from a Gaussian
      std: 0.01        # distribution with stdev 0.01 (default mean: 0)
    }
    bias_filler {
      type: "constant" # initialize the biases to zero (0)
      value: 0
    }
  }
}
```

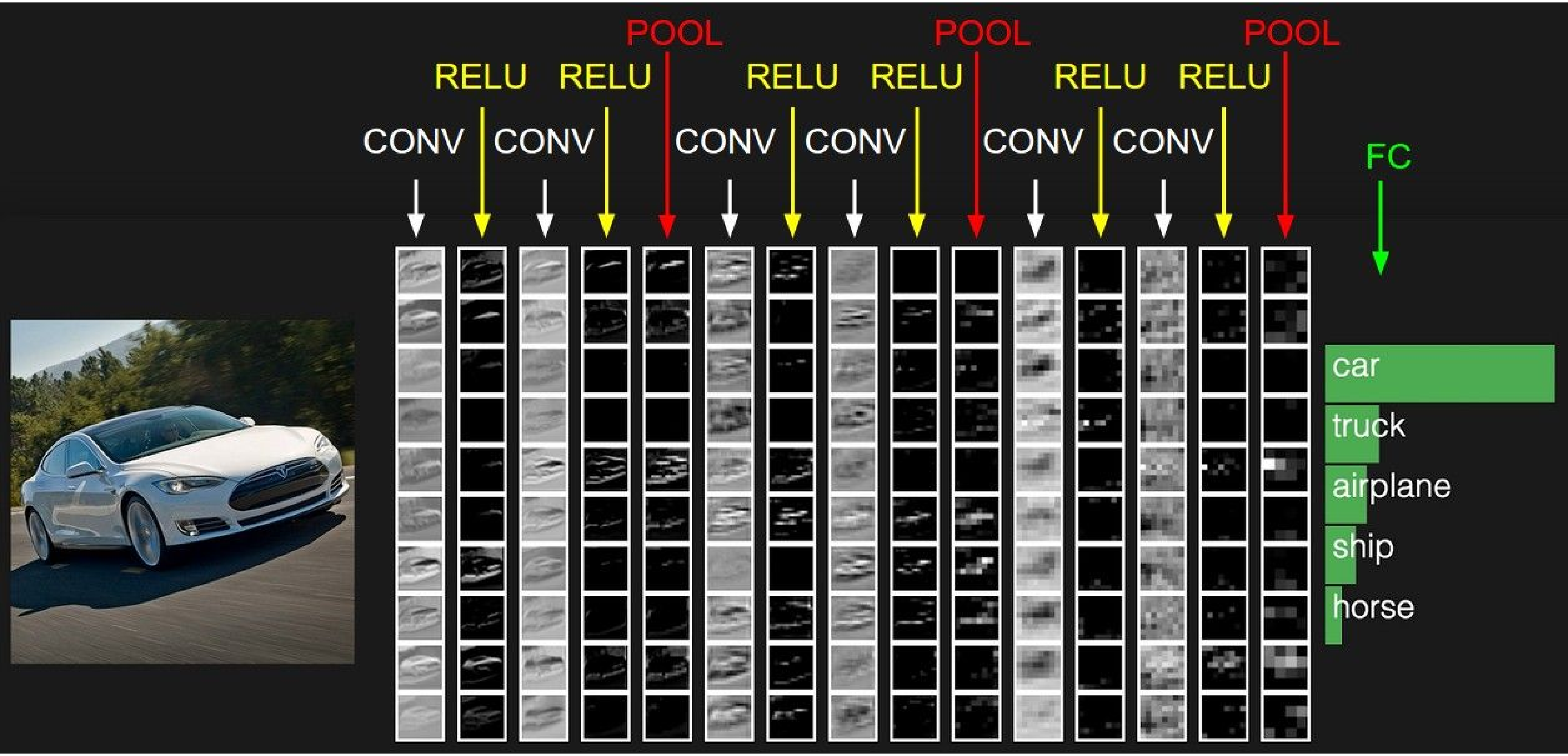
Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.

UNIVERSITY OF SURREY

Convolutional Neural Networks

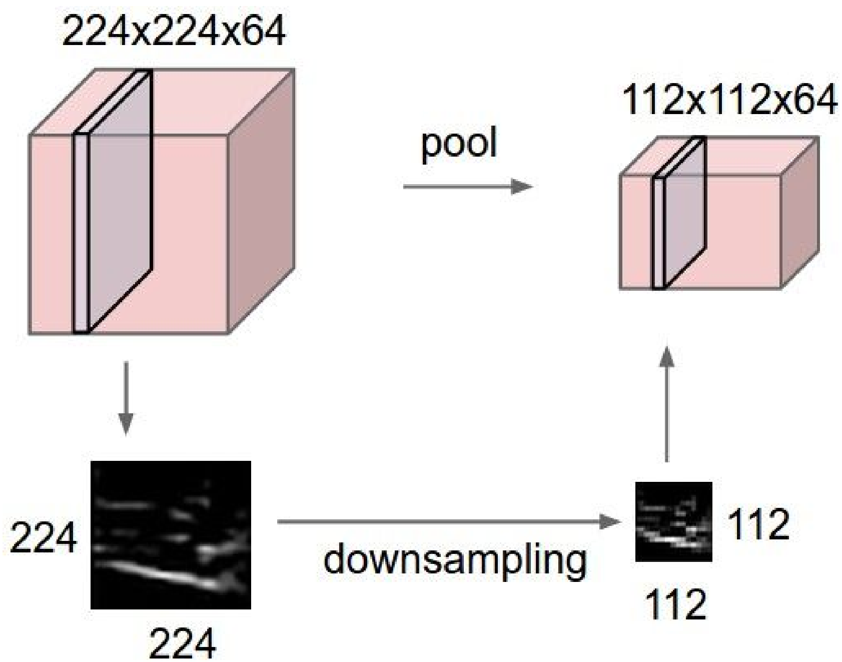
Pooling layer



Convolutional Neural Networks

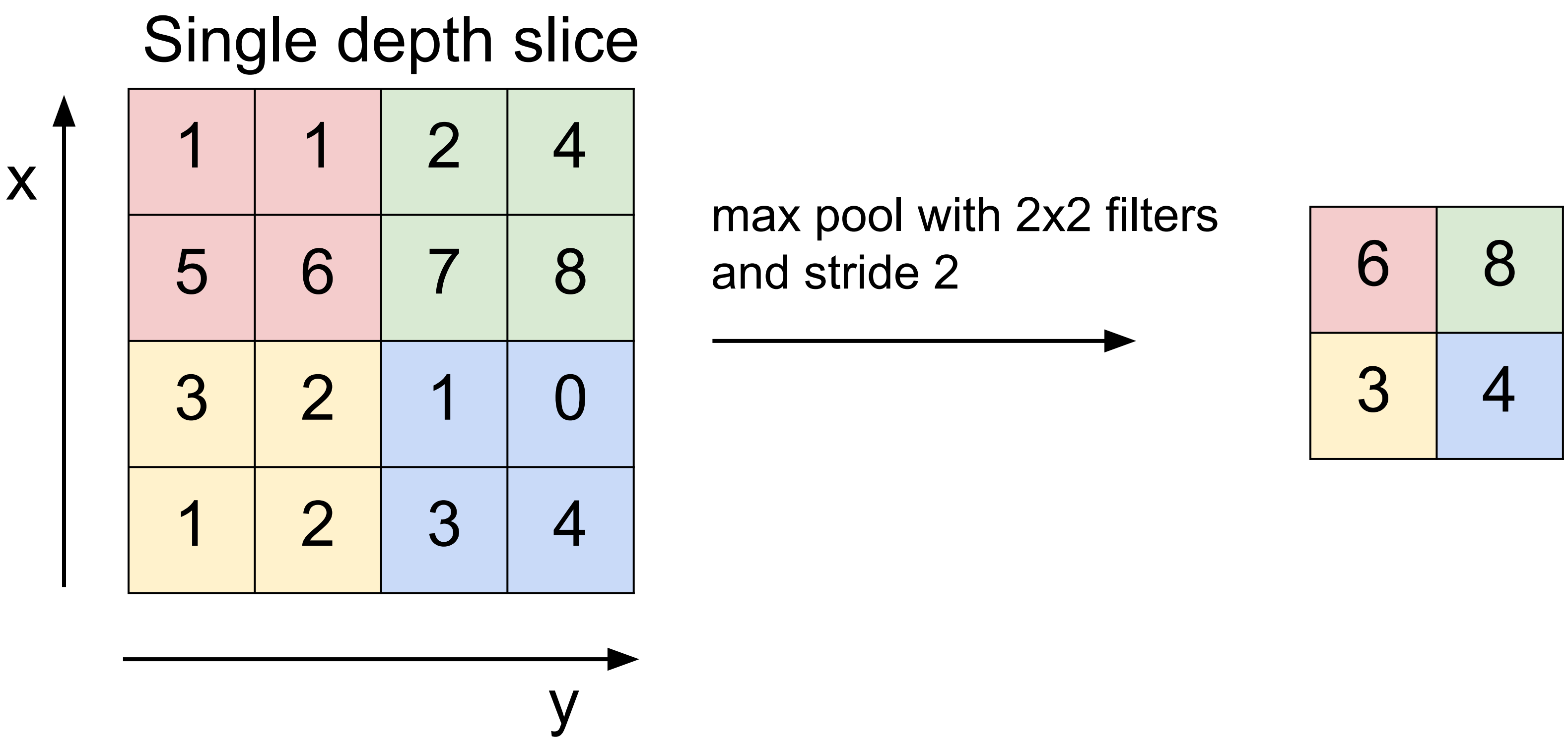
Pooling layer

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



Convolutional Neural Networks

Pooling layer



UNIVERSITY OF SURREY

Convolutional Neural Networks

Pooling layer

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires three hyperparameters:
  - their spatial extent F,
  - the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for Pooling layers

Convolutional Neural Networks

Pooling layer

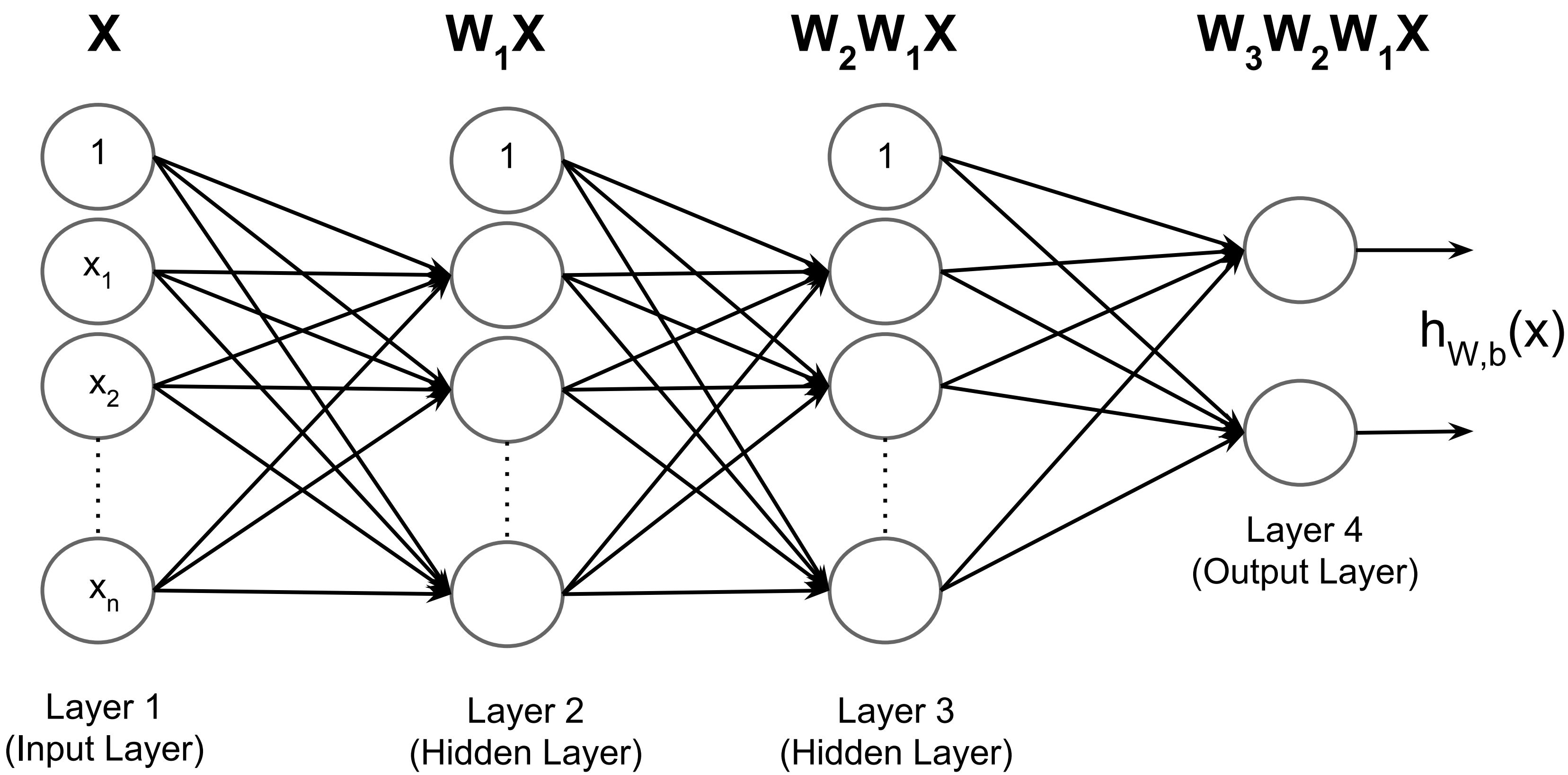
- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires three hyperparameters:
  - their spatial extent F,
  - the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for Pooling layers

Common settings:

F = 2, S = 2
F = 3, S = 2

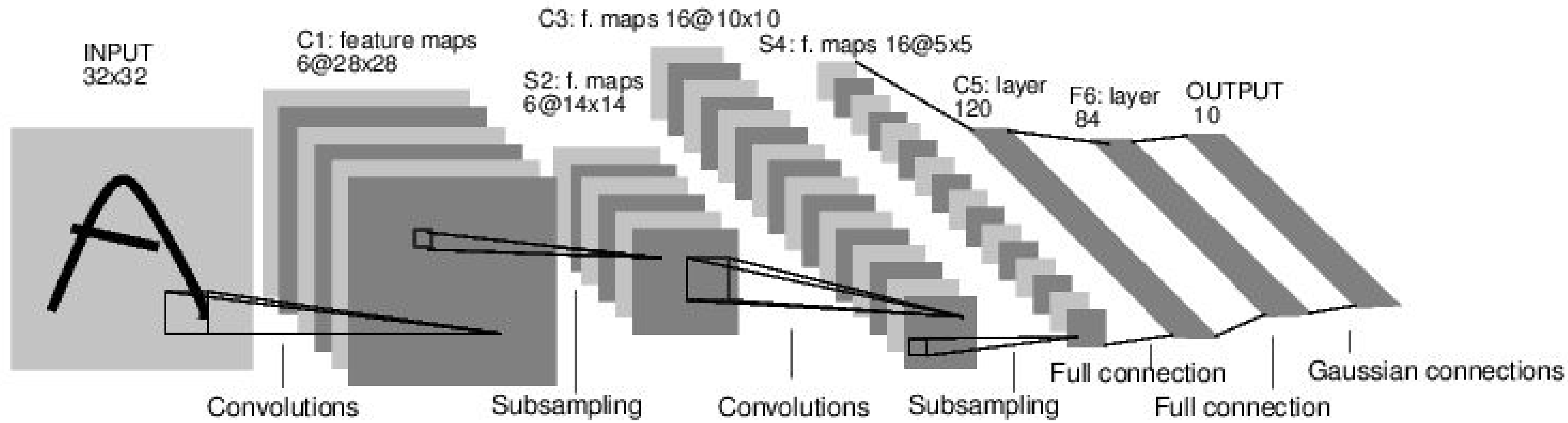
Convolutional Neural Networks

Fully Connected Layer (FC layer)



Famous CNN architectures (LeNet)

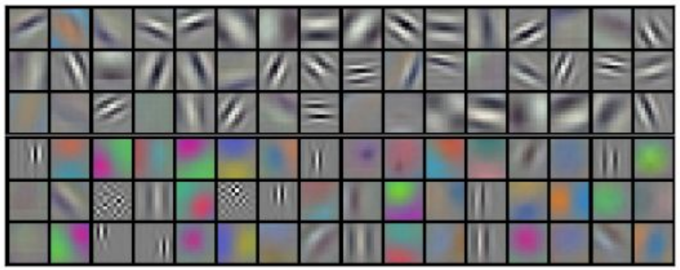
[LeCun et al., 1998]

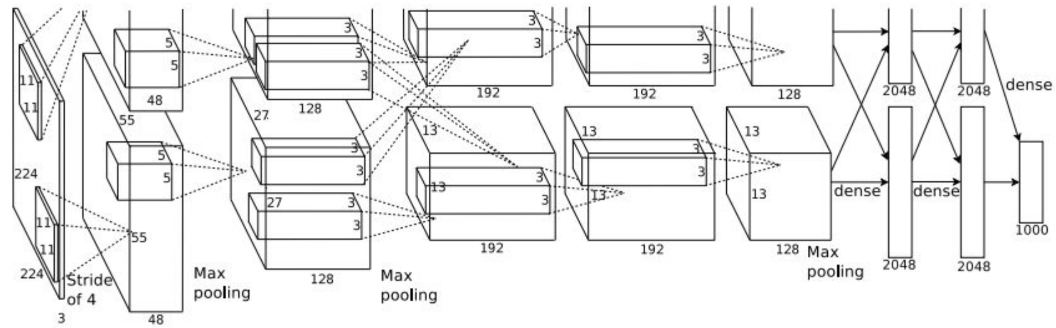


Conv filters were 5x5, applied at stride 1
Subsampling (Pooling) layers were 2x2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-CONV-FC]

Famous CNN architectures (AlexNet)

[Krizhevsky et al. 2012]

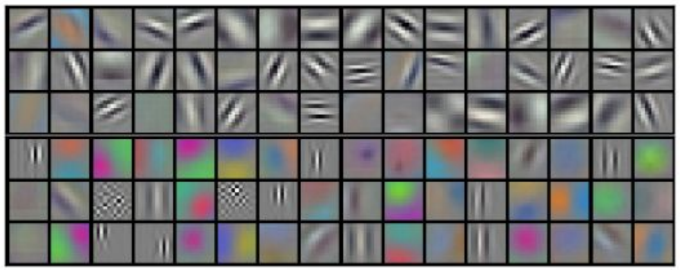


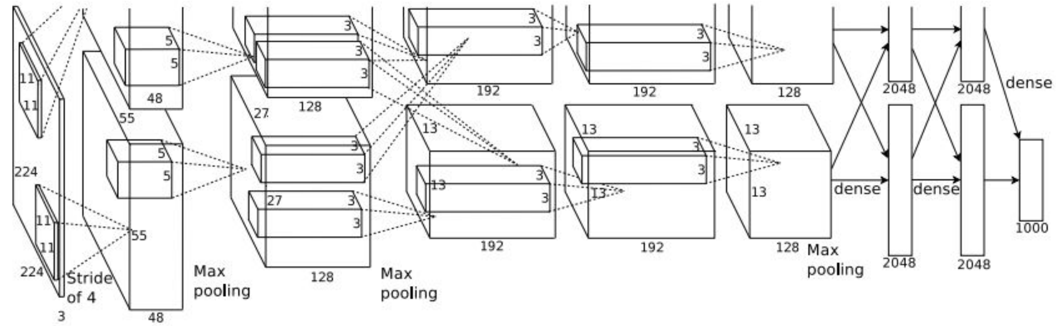


Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

Famous CNN architectures (AlexNet)

[Krizhevsky et al. 2012]

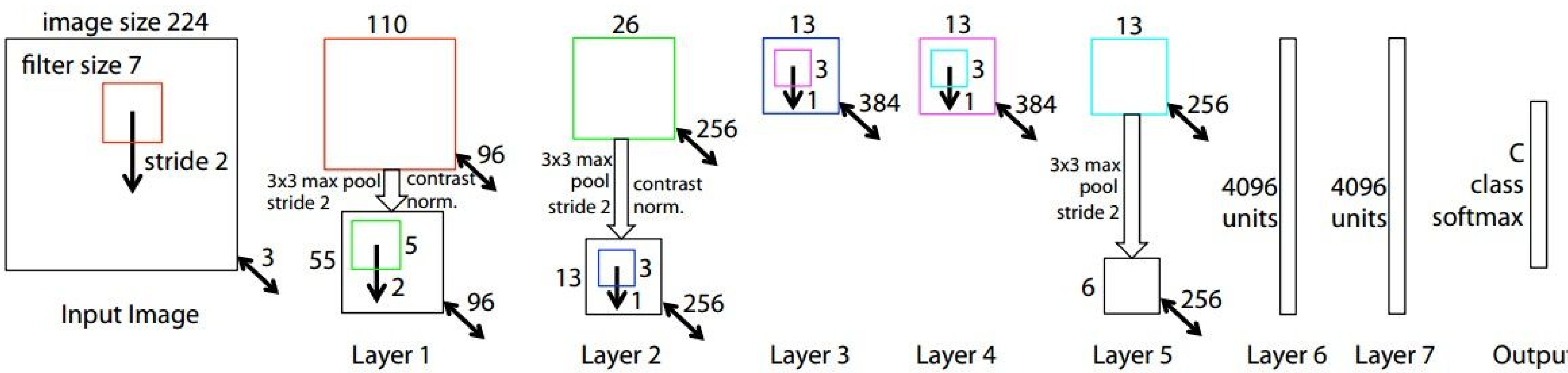




Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

Details/Retrospectives:
- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- SGD Momentum 0.9
- Learning rate 1e-2, reduced by 10 manually when val accuracy plateaus
- L2 weight decay 5e-4
- 7 CNN ensemble: 18.2% -> 15.4%

Famous CNN architectures (ZFNet)



[Zeiler and Fergus, 2013]

AlexNet but:
CONV1: change from (11x11 stride 4) to (7x7 stride 2)
CONV3,4,5: instead of 384, 384, 256 filters use 512, 1024, 512

ImageNet top 5 error: 15.4% -> 14.8%

Famous CNN architectures (VGGNet)

[Simonyan and Zisserman, 2014]

Only 3x3 CONV stride 1, pad 1
and 2x2 MAX POOL stride 2

best model

11.2% top 5 error in ILSVRC 2013
->
7.3% top 5 error

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

Famous CNN architectures (VGGNet)

(not counting biases)

INPUT: [224x224x3] memory: 224*224*3=150K params: 0
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = 102,760,448
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

TOTAL memory: 24M * 4 bytes ~= 93MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters

ConvNet Configuration			
B	C	D	
13 weight layers	16 weight layers	16 weight layers	19
put (224 × 224 RGB image)			
conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	cc cc
maxpool			
conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	co co
maxpool			
conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	co co co **co**
maxpool			
conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	co co co **co**
maxpool			
conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	co co co **co**
maxpool			
FC-4096			
FC-4096			
FC-1000			
soft-max			

Famous CNN architectures (VGGNet)

(not counting biases)

INPUT: [224x224x3] memory: 224*224*3=150K params: 0
CONV3-64: [224x224x64] memory: **224*224*64=3.2M** params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: **224*224*64=3.2M** params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = **102,760,448**
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

Note:

Most memory is in early CONV

Most params are in late FC

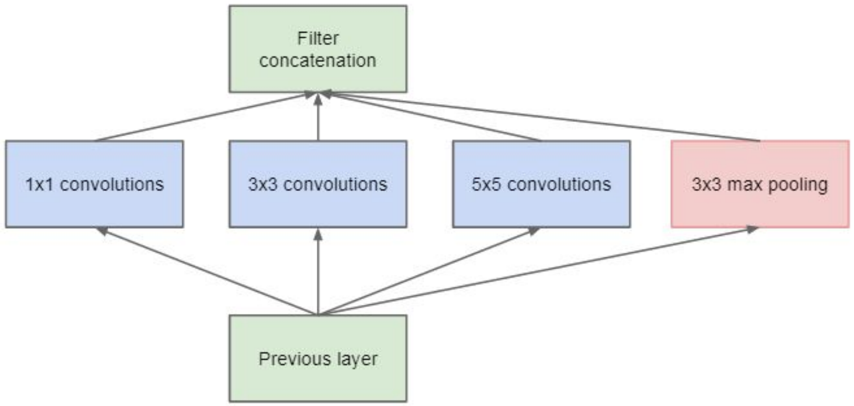
TOTAL memory: 24M * 4 bytes ~= 93MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters

Famous CNN architectures (GoogLeNet)

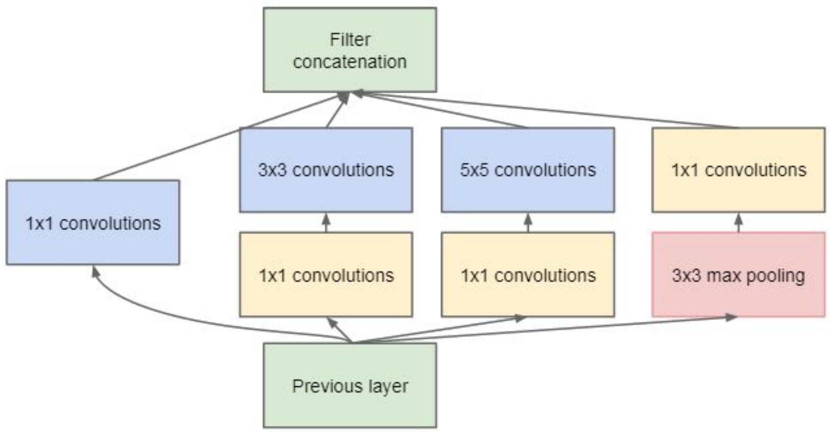
[Szegedy et al., 2014]

Neural Network Architecture

Inception modules

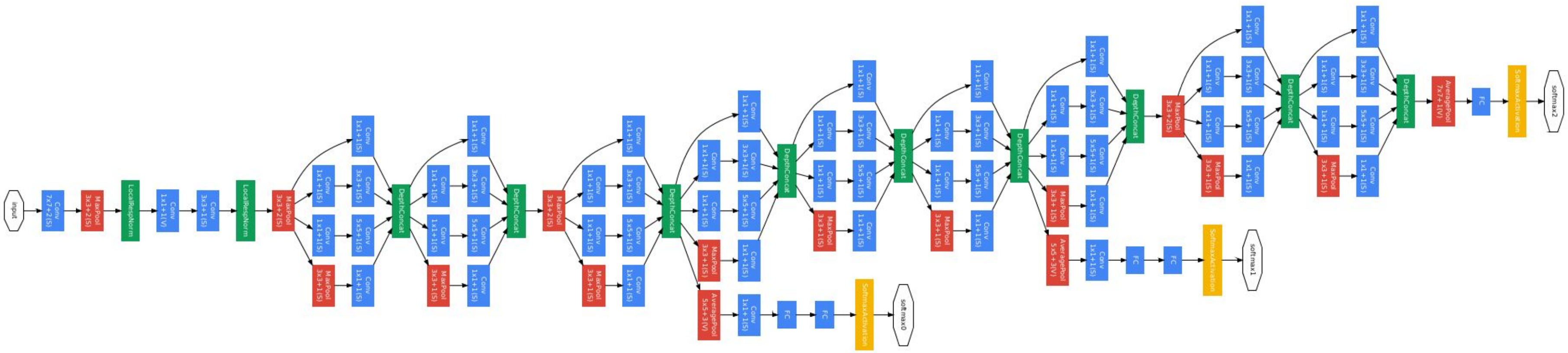


(a) Inception module, naïve version



(b) Inception module with dimension reductions

Famous CNN architectures (GoogLeNet)



ILSVRC 2014 winner (6.7% top 5 error)

UNIVERSITY OF SURREY

Famous CNN architectures (GoogLeNet)

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

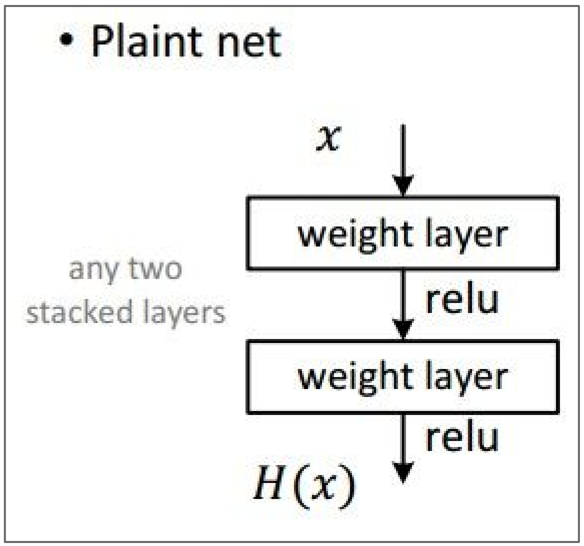
Fun features:

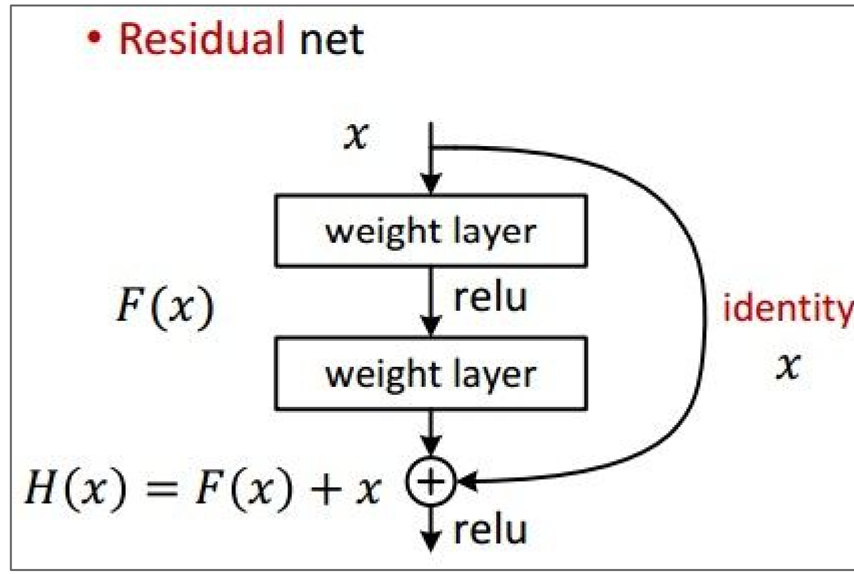
- Only 5 million params! (Removes FC layers completely)

Compared to AlexNet:
- 12X less params
- 2x more compute
- 6.67% (vs. 16.4%)

Famous CNN architectures (ResNet)

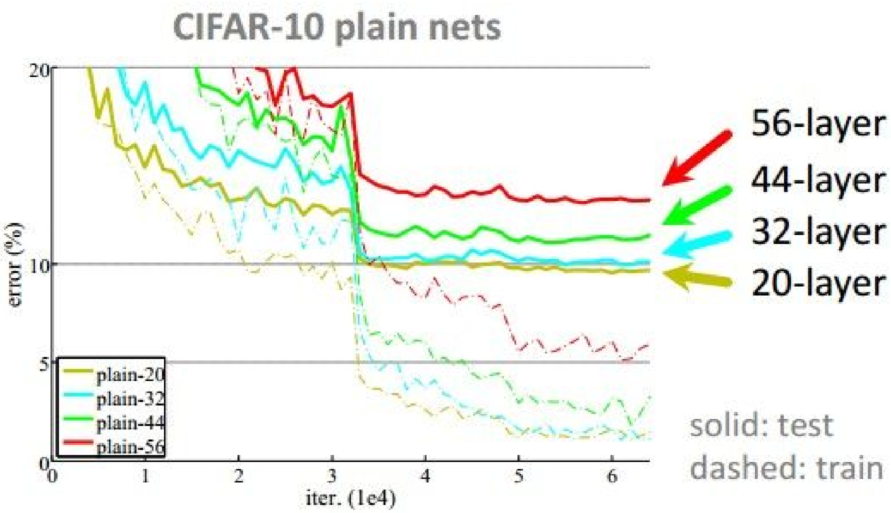
[He et al., 2015]

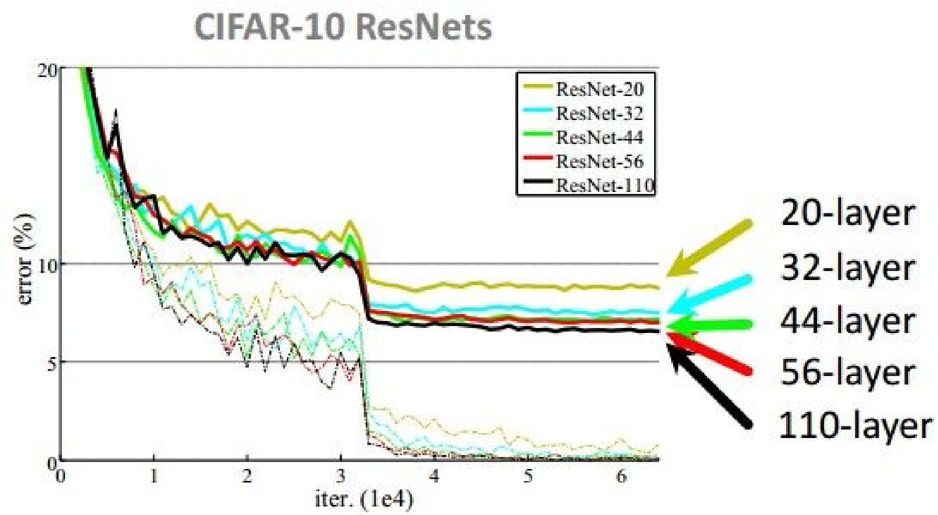




Famous CNN architectures (ResNet)

CIFAR-10 experiments

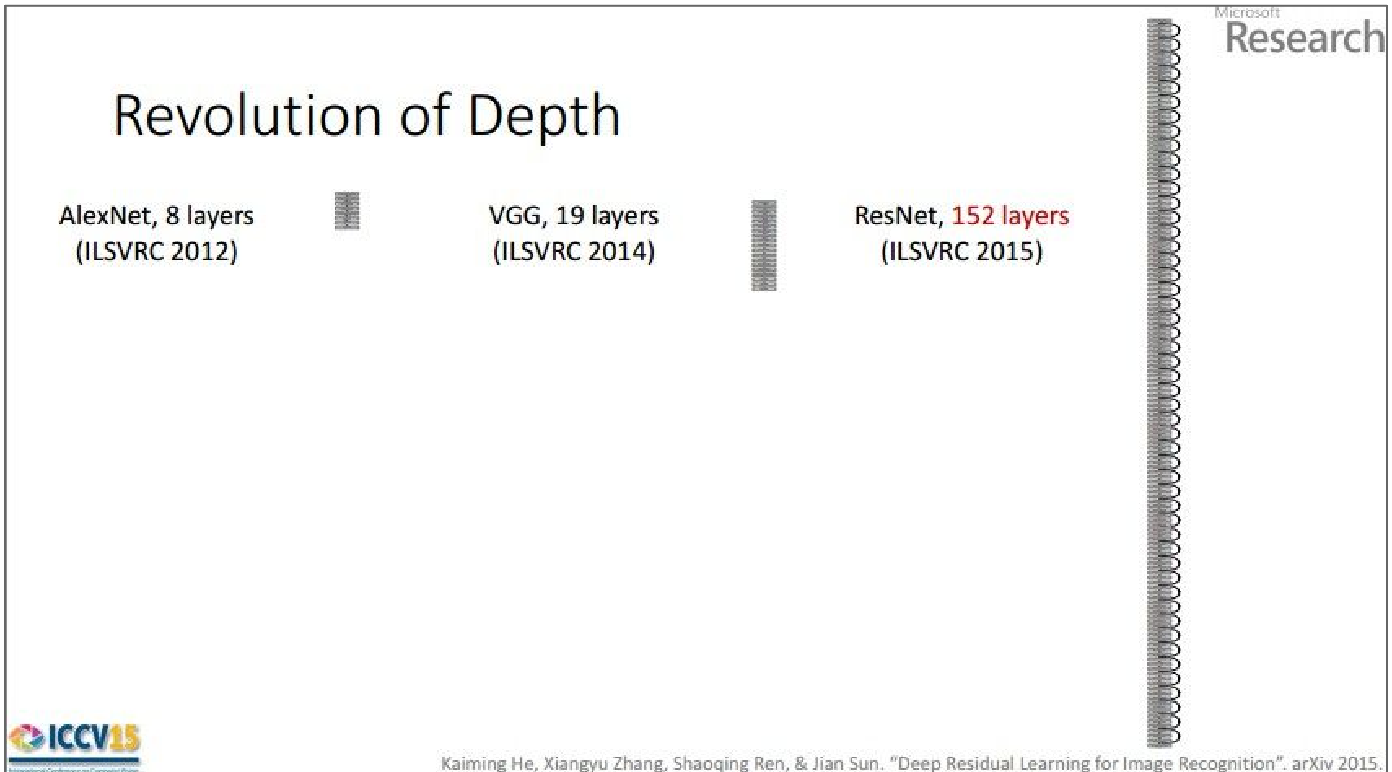




Famous CNN architectures (ResNet)

[He et al., 2015]

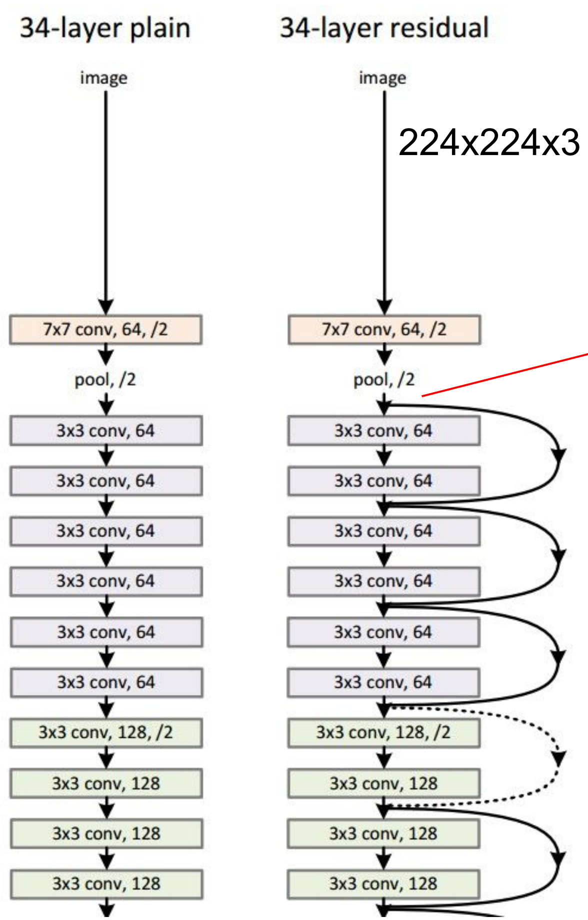
ILSVRC 2015 winner (3.6% top 5 error)



2-3 weeks of training on 8 GPU machine

at runtime: faster than a VGGNet! (even though it has 8x more layers)

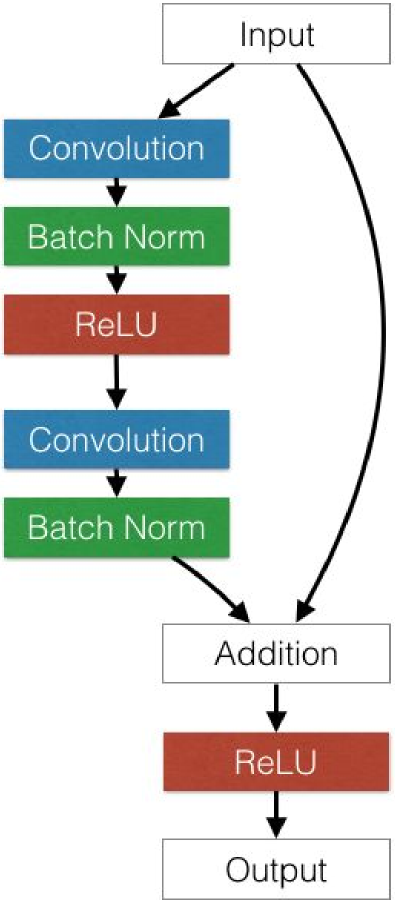
Famous CNN architectures (ResNet)

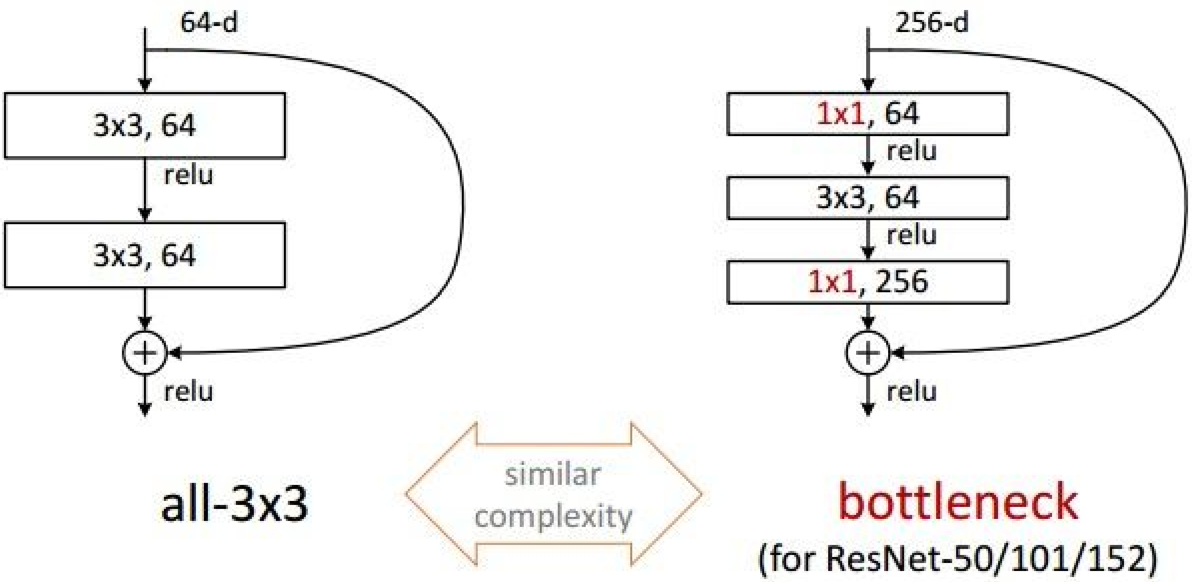


spatial dimension
only 56x56!

Famous CNN architectures (ResNet)

[He et al., 2015]

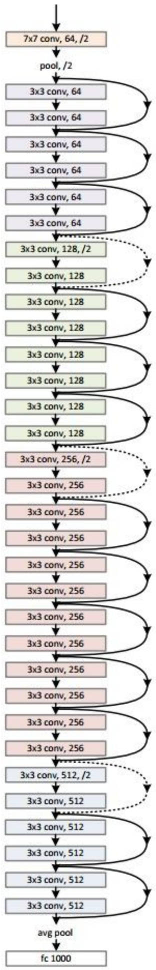




Famous CNN architectures (ResNet)

- Batch Normalization after every CONV layer
- Xavier/2 initialization from He et al.
- SGD + Momentum (0.9)
- Learning rate: 0.1, divided by 10 when validation error plateaus
- Mini-batch size 256
- Weight decay of 1e-5
- No dropout used

Famous CNN architectures (ResNet)

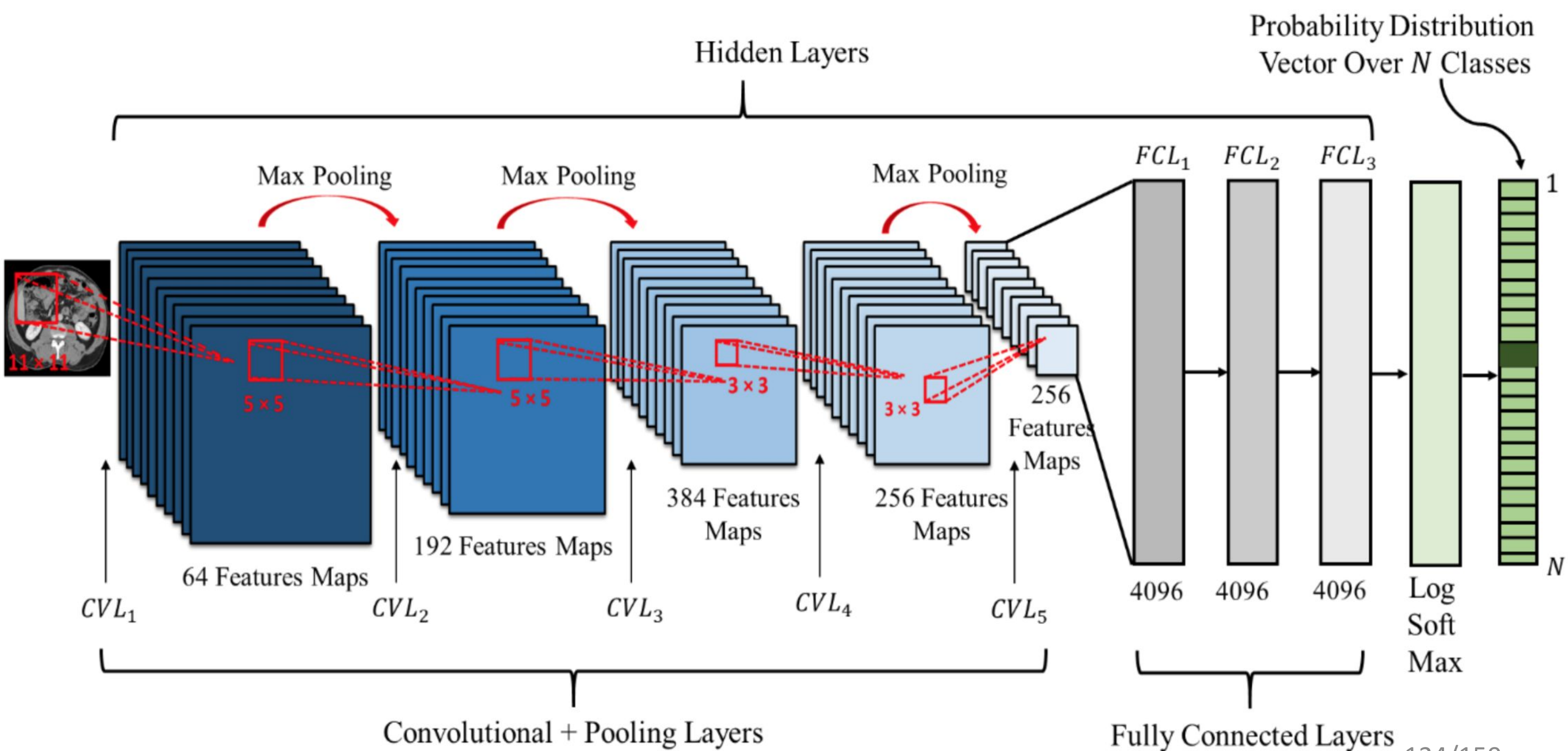


layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times23$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Deep Learning Applications

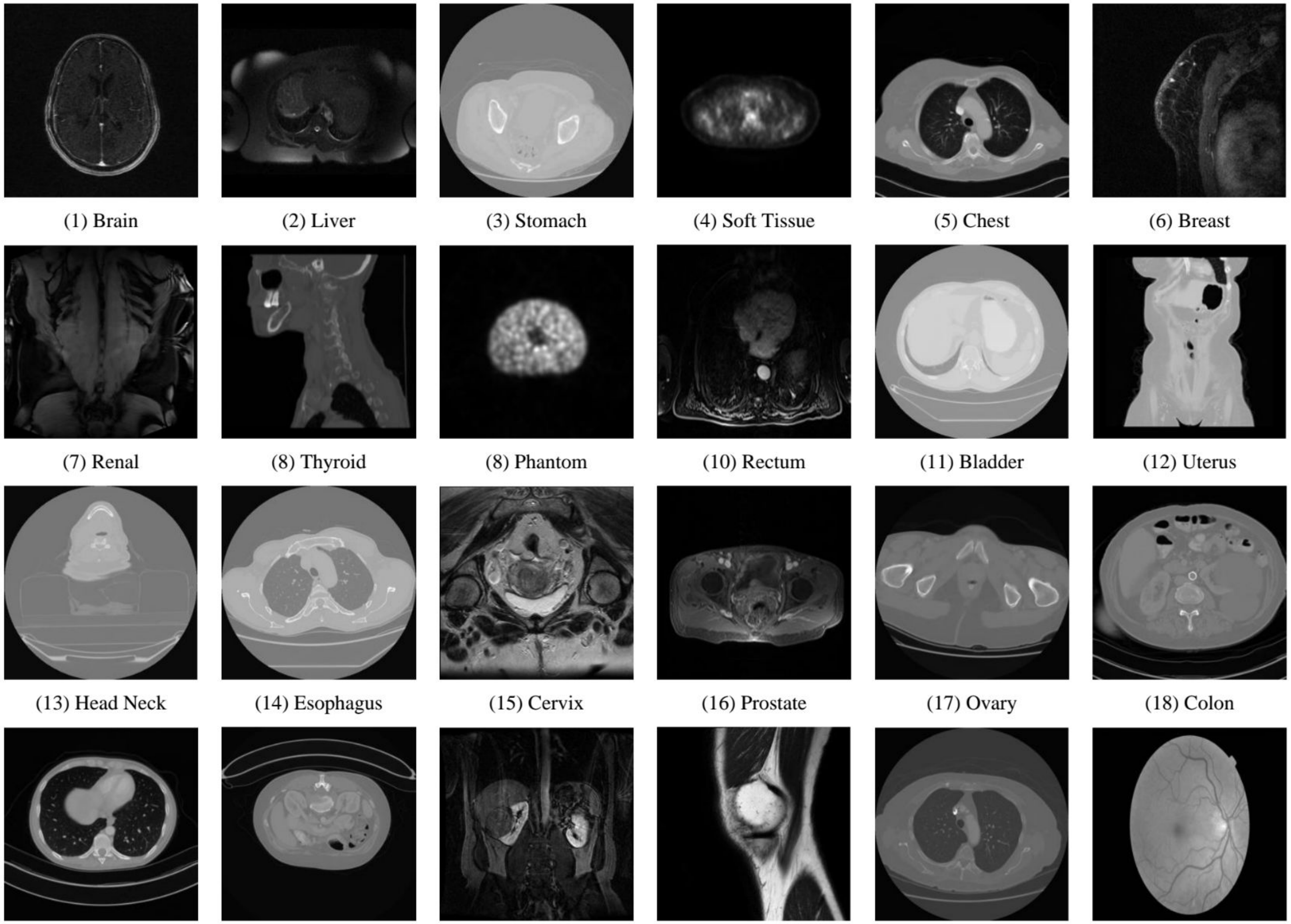
Deep Learning Applications

➢DCNN architecture used for CBMIR task



Deep Learning Applications

➢Example image from each class (interclass variation)



(1) Brain (2) Liver (3) Stomach (4) Soft Tissue (5) Chest (6) Breast

(7) Renal (8) Thyroid (8) Phantom (10) Rectum (11) Bladder (12) Uterus

(13) Head Neck (14) Esophagus (15) Cervix (16) Prostate (17) Ovary (18) Colon

(19) Lymph (20) Pancreas (21) Kidney (22) Knee (23) Lungs (24) Eye

UNIVERSITY OF SURREY

Deep Learning Applications

- ➢Confusion matrix for 24 classes using DCNN
  - ○99.82% accuracy (human accuracy is around 85%)

	Brain	Liver	Stomach	Soft Tissue	Chest	Breast	Renal	Thyroid	Phantom	Rectum	Bladder	Uterus	Head-Neck	Esophagus	Cervix	Prostate	Ovary	Colon	Abdomen	Pancreas	Kidney	Knee	Lungs	Eye
Brain	100.0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Liver	0	98.9	0	0	0	0	0	0	0	0	0	0	0	0	1.1	0	0	0	0	0	0	0	0	0
Stomach	0	0	96.7	0	0	0	0	3.3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Soft Tissue	0	0	0	100.0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Chest	0	0	0	0	100.0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Breast	0	0	0	0	0	100.0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Renal	0	0	0	0	0	0	100.0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Thyroid	0	0	0	0	0	0	0	100.0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Phantom	0	0	0	0	0	0	0	0	100.0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Rectum	0	0	0	0	0	0	0	0	0	100.0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bladder	0	0	0	0	0	0	0	0	0	0	100.0	0	0	0	0	0	0	0	0	0	0	0	0	0
Uterus	0	0	0	0	0	0	0	0	0	0	0	98.9	0	0	0	0	0	0	0	0	0	1.1	0	0
Head-Neck	0	0	0	0	0	0	0	0	0	0	0	0	100.0	0	0	0	0	0	0	0	0	0	0	0
Esophagus	0	0	0	0	0	0	0	0	0	0	0	0	0	100.0	0	0	0	0	0	0	0	0	0	0
Cervix	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100.0	0	0	0	0	0	0	0	0	0
Prostate	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100.0	0	0	0	0	0	0	0	0
Ovary	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100.0	0	0	0	0	0	0	0
Colon	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100.0	0	0	0	0	0	0
Abdomen	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100.0	0	0	0	0	0
Pancreas	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100.0	0	0	0	0
Kidney	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100.0	0	0	0
Knee	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100.0	0	0
Lungs	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100.0	0
Eye	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100.0

UNIVERSITY OF SURREY

Deep Learning Applications

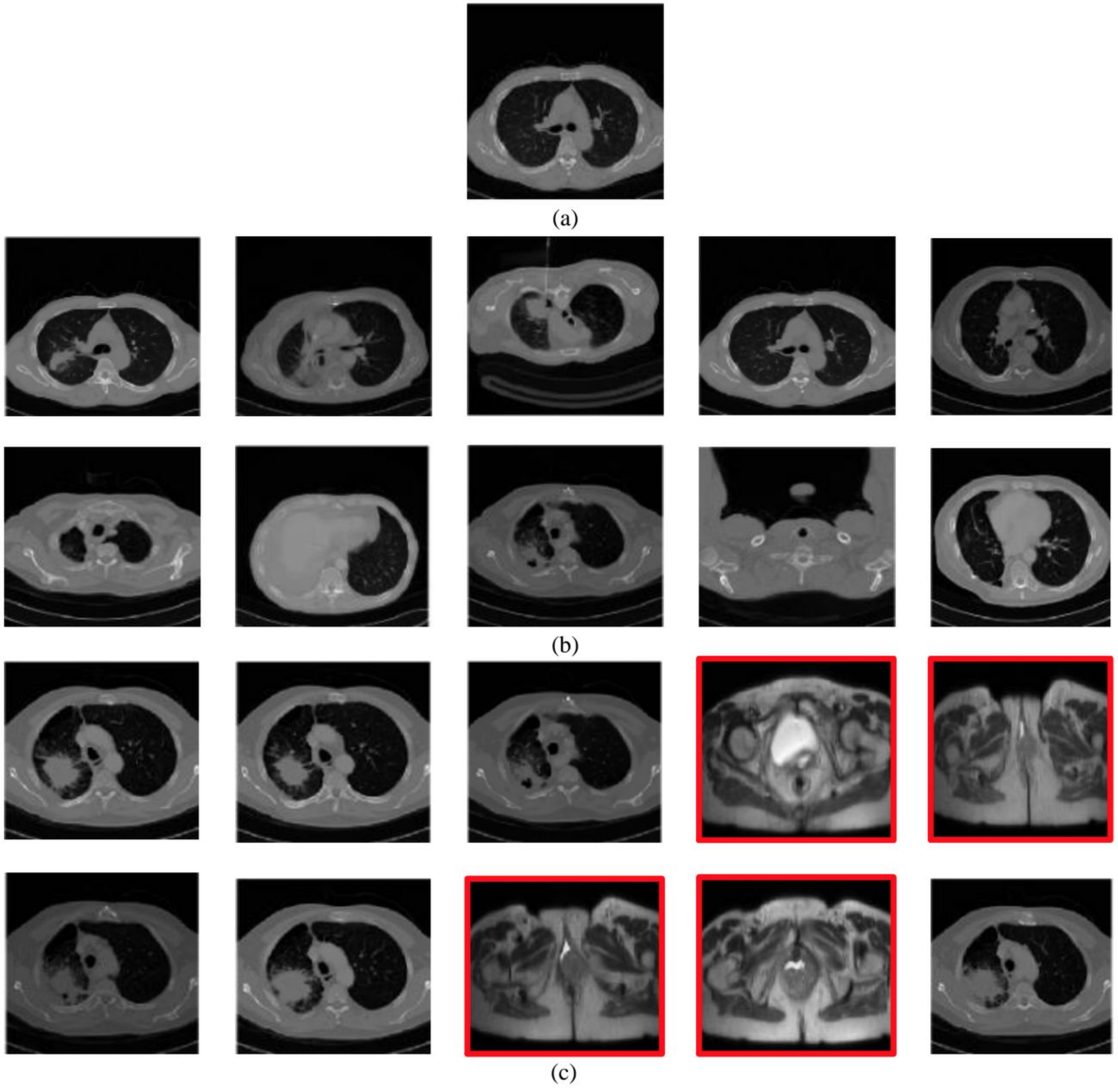
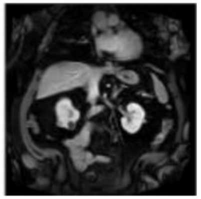
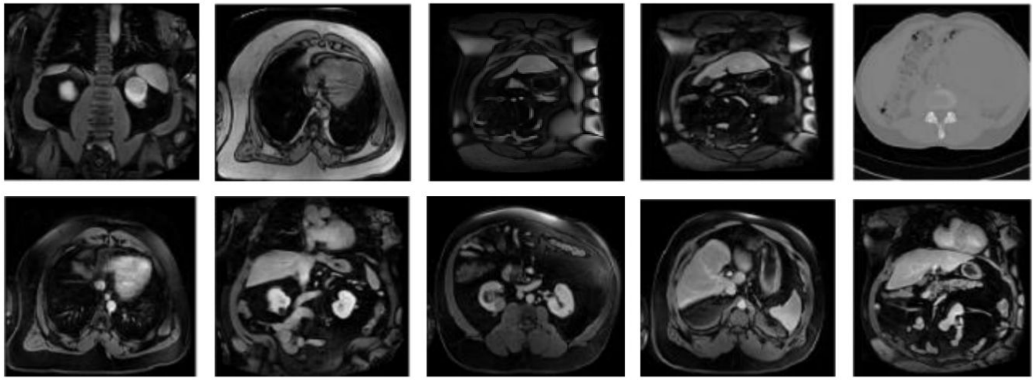


Fig. 9. Retrieval results for chest class (a) query image (b) retrieved images using class prediction (c) retrieved images without using class prediction.

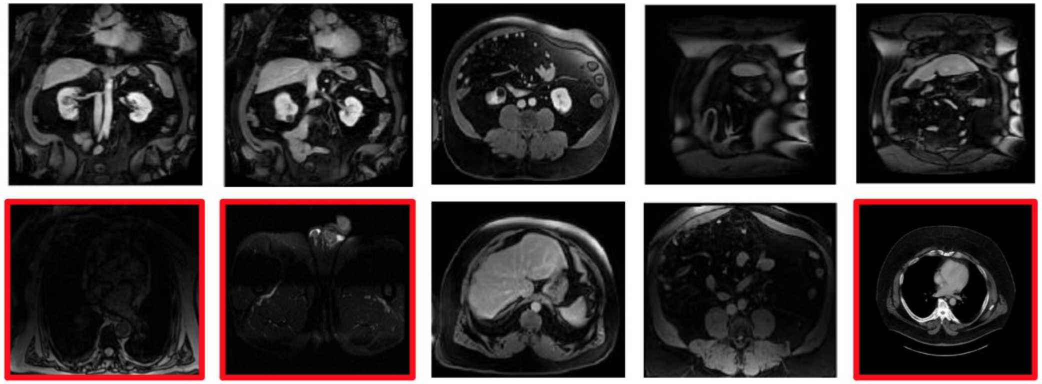
Deep Learning Applications



(a)



(b)



(c)

Fig. 9. Retrieval results for renal class (a) query images (b) retrieved images using class prediction c) retrieved images without using class prediction.

Deep Learning Applications

Classification

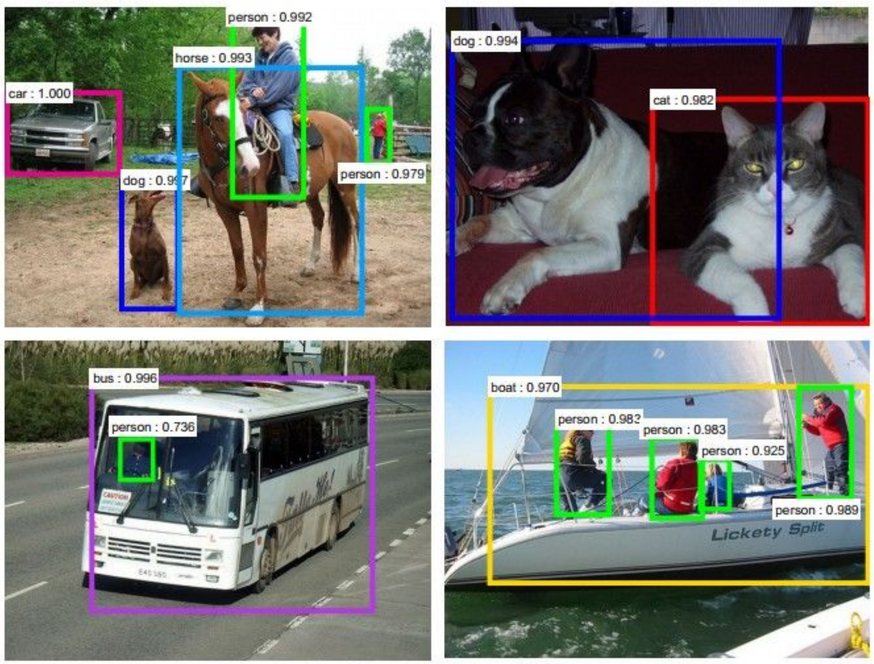
Retrieval

[Krizhevsky 2012]

UNIVERSITY OF SURREY

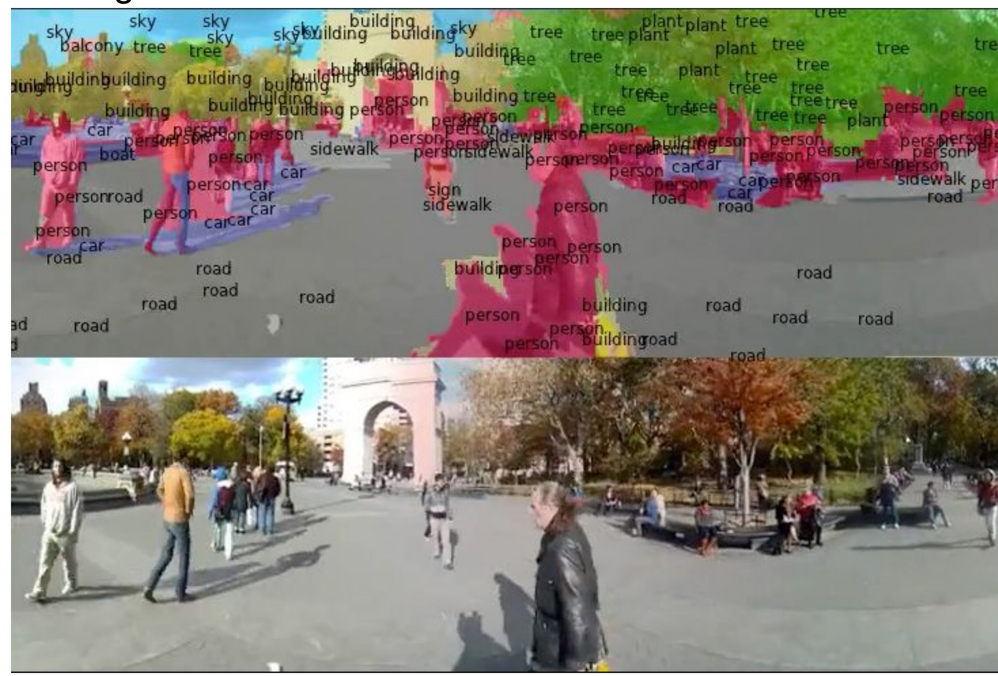
Deep Learning Applications

Detection



[Faster R-CNN: Ren, He, Girshick, Sun 2015]

Segmentation

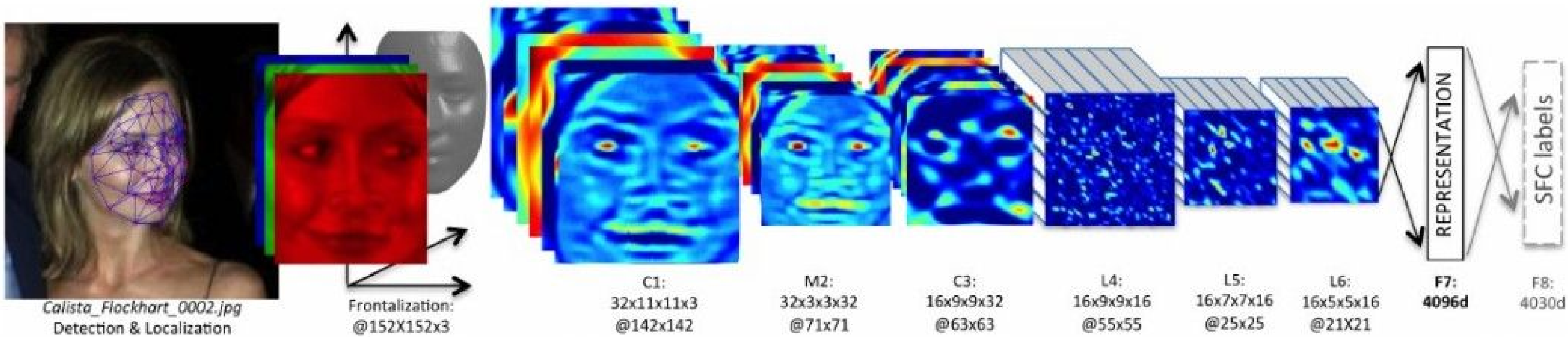


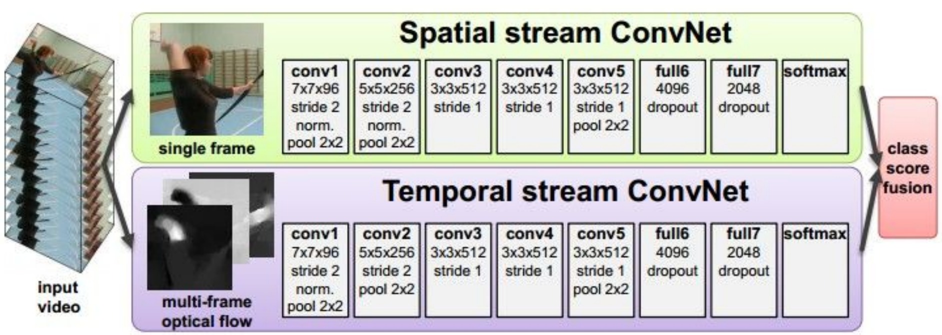
[Farabet et al., 2012]

Deep Learning Applications

self-driving cars

Deep Learning Applications





[Simonyan et al. 2014]

[Goodfellow 2014]

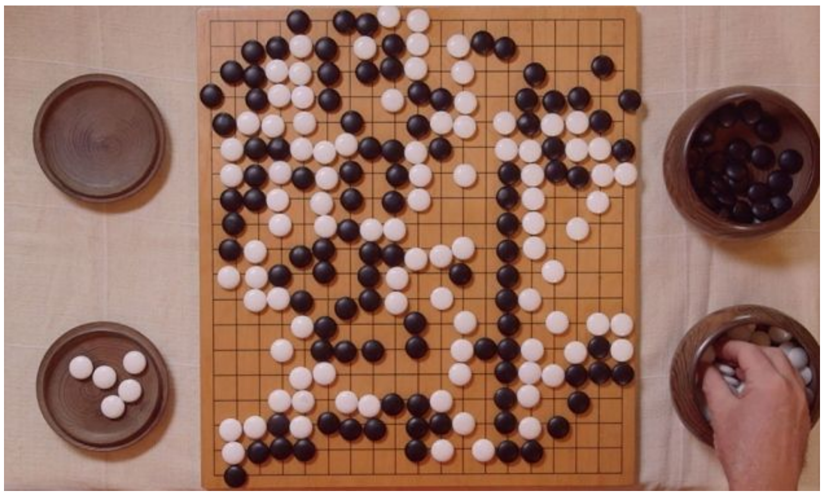
UNIVERSITY OF SURREY

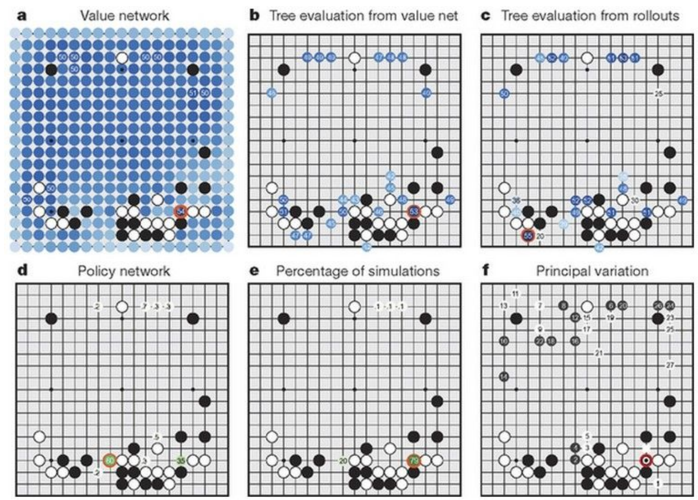
Deep Learning Applications

[Toshev, Szegedy 2014]

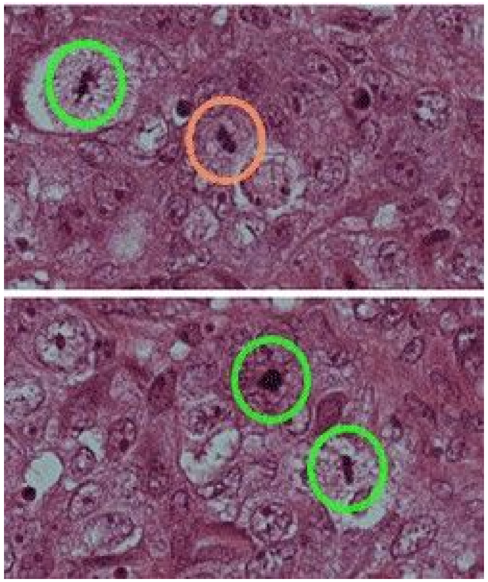
[Mnih 2013]

Deep Learning Applications





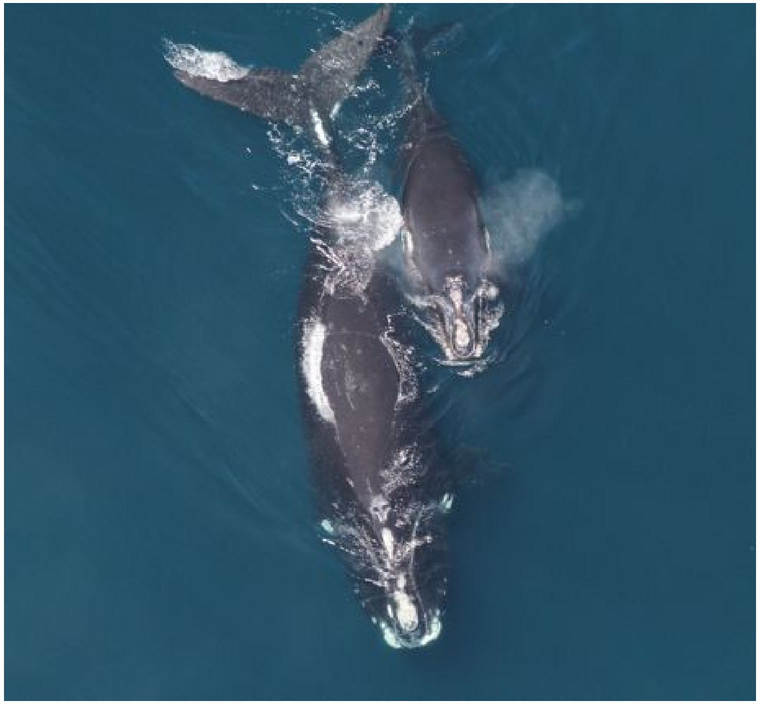
Deep Learning Applications



[Ciresan et al. 2013]

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

Deep Learning Applications



Whale recognition, Kaggle Challenge

Mnih and Hinton, 2010

Deep Learning Applications

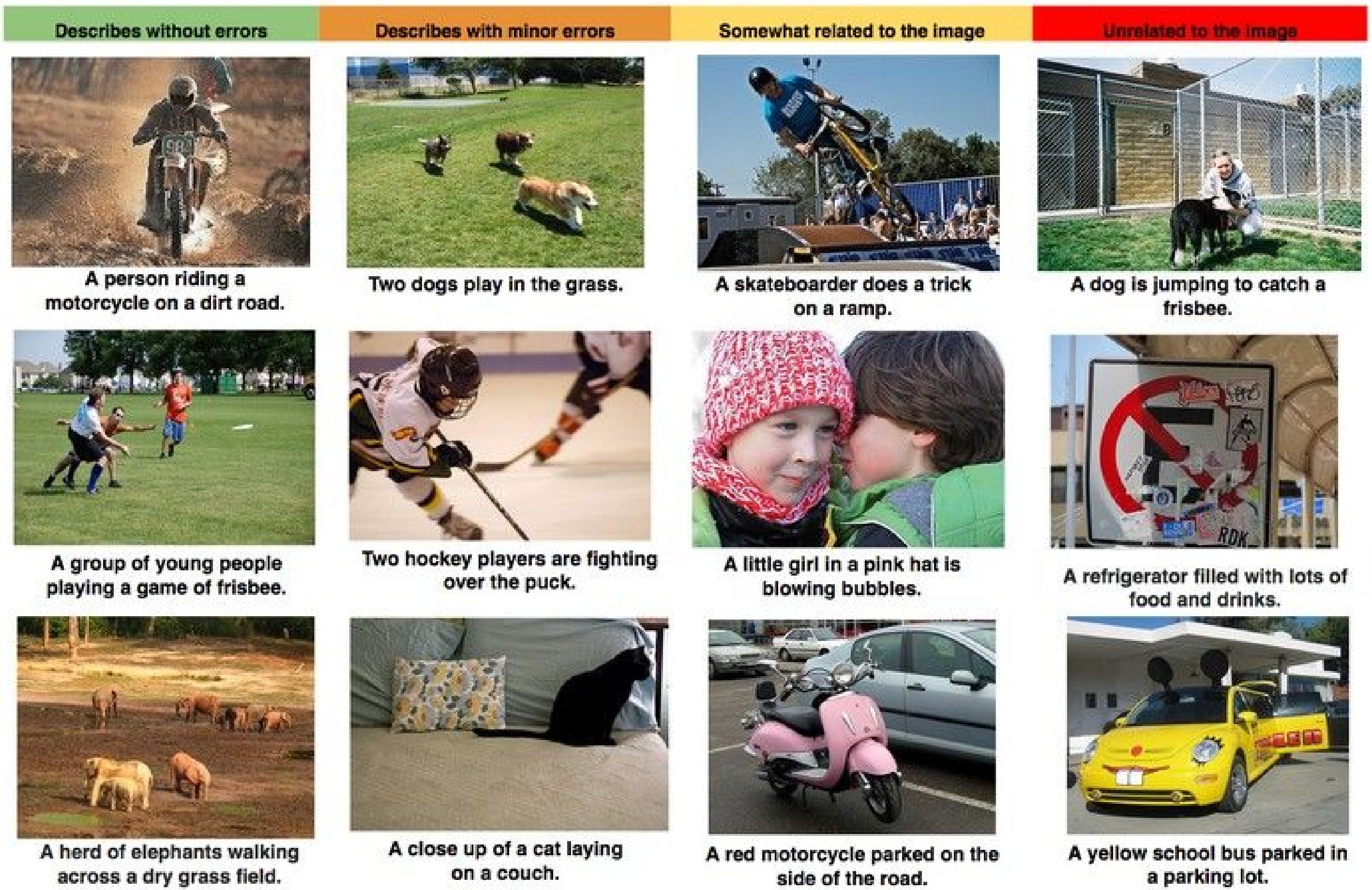


Image Captioning

[Vinyals et al., 2015]

Any Question???
Thanks