EBU7240

Computer Vision

- Multi-layer Perceptron (MLP)-

Semester 1, 2021

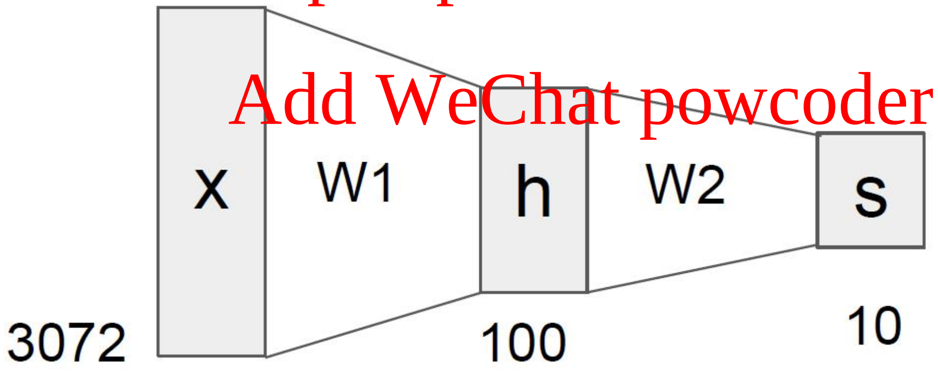
Changjae Oh

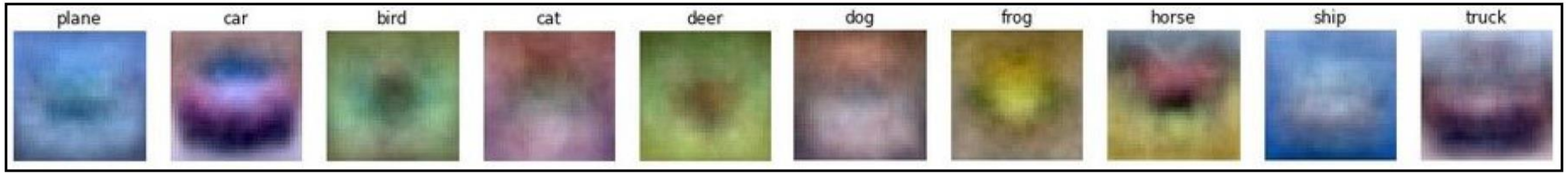
Neural networks

(**Before**) Linear score function: $\boldsymbol{f} = \mathbf{W}\boldsymbol{x}$

(**Now**) 2-layer Neural Network: $\boldsymbol{f} = \mathbf{W_2}\max(\mathbf{0}, \mathbf{W_1}\boldsymbol{x})$

3-layer Neural Network: $\boldsymbol{f} = \mathbf{W_3}\max\big(\mathbf{0}, \mathbf{W_2}\max(\mathbf{0}, \mathbf{W_1}\boldsymbol{x})\big)$





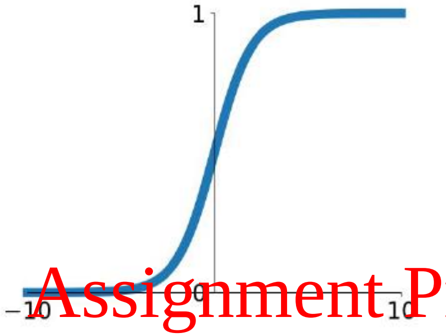
Activation functions

- **Adding non-linearities into neural networks, allowing the neural networks to learn powerful operations.**
- **A crucial component of deep learning**
  - If the activation functions were to be removed from a feedforward neural network, the entire network could be re-factored to a simple linear operation or matrix transformation on its input
  - It would no longer be capable of performing complex tasks such as image recognition.

Activation functions

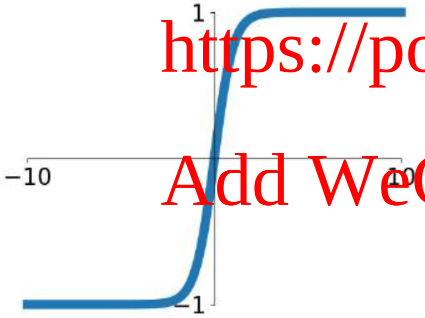
Sigmoid

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



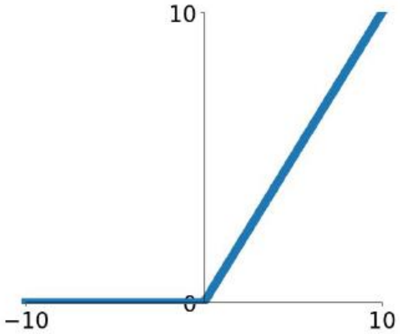
tanh

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



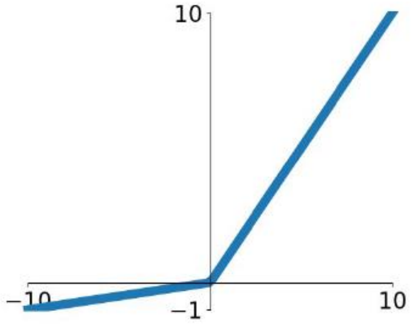
ReLU

$$\max(0, x)$$



Leaky ReLU

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

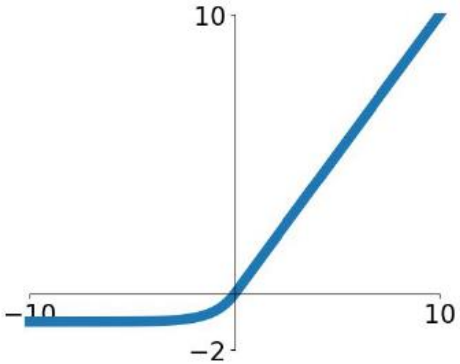


Maxout

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

ELU

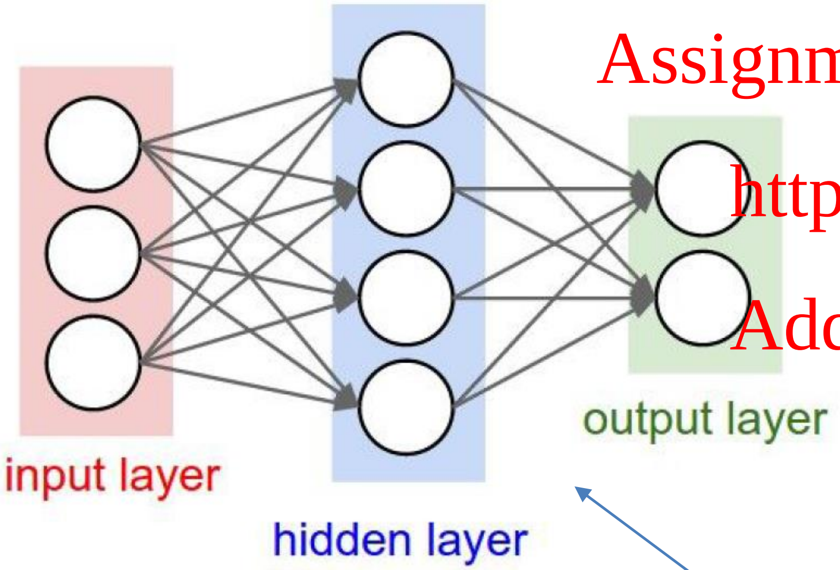
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

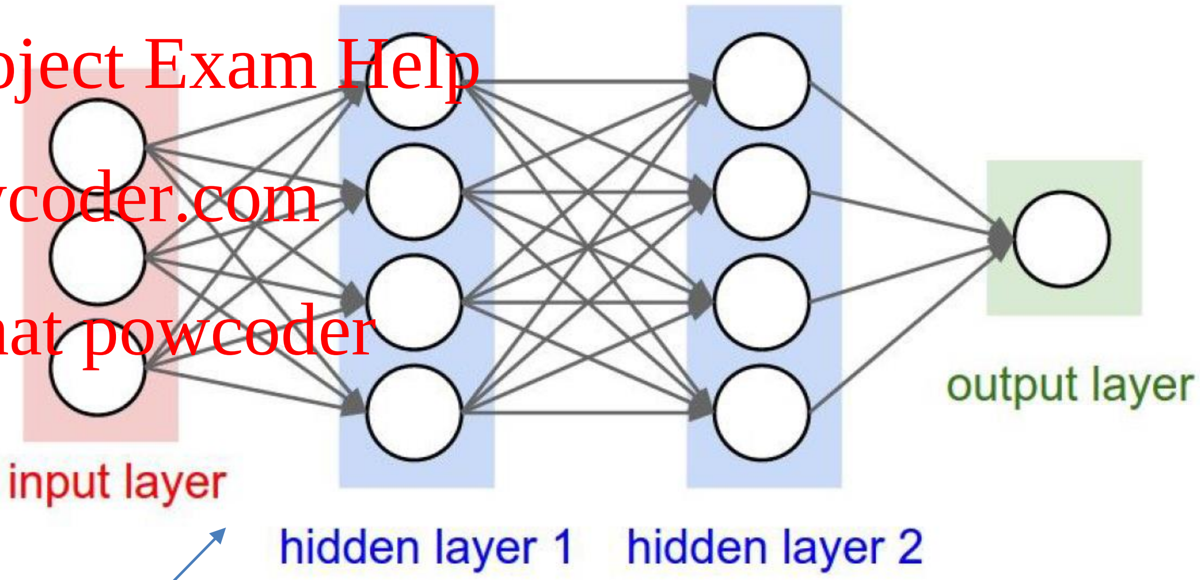


Neural networks: Architectures

“2-layer Neural Net”, or
“1-hidden-layer Neural Net”

“3-layer Neural Net”, or
“2-hidden-layer Neural Net”



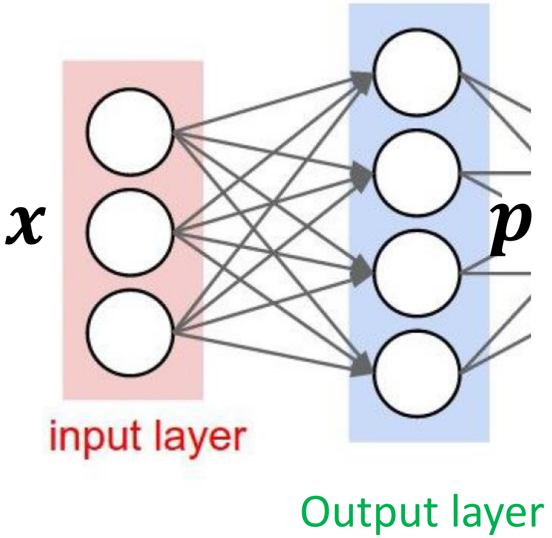


“Fully-connected” layers

Derivative of Neural Net using Chain Rules

- **Example**
  - 1-layer Neural Net (L2 regression loss)
  - 2-layer Neural Net (L2 regression loss)
  - 1-layer Neural Net (Softmax classifier)
  - 2-layer Neural Net (Softmax classifier)

1. 1-layer Neural Net (L2 regression loss)



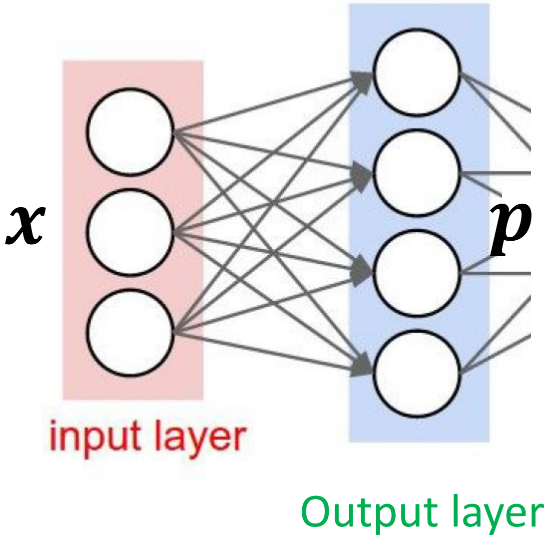
1. Linear score $\boldsymbol{s} = \mathbf{W}\boldsymbol{x} + \boldsymbol{b} \longleftrightarrow s_j = \boldsymbol{w}_j^T \boldsymbol{x} + b_j$

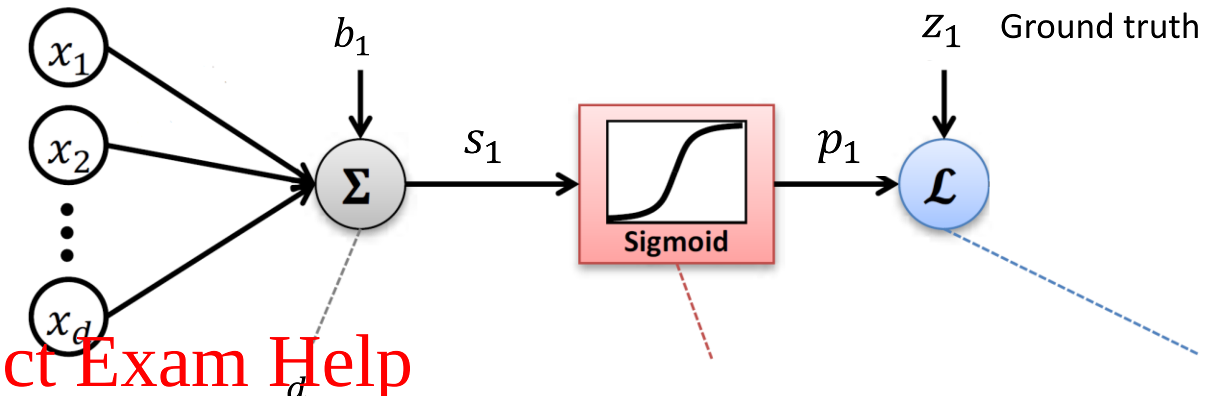
2. Activation function $\boldsymbol{p} = \sigma(\boldsymbol{s}) = \dfrac{1}{1 + e^{-\boldsymbol{s}}}$

3. Loss $L = (\boldsymbol{z} - \boldsymbol{p})^2$

$$\boldsymbol{s} = \begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_n \end{pmatrix} \quad \mathbf{W} = \begin{pmatrix} \boldsymbol{w}_1^T \\ \boldsymbol{w}_2^T \\ \vdots \\ \boldsymbol{w}_n^T \end{pmatrix} = \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1d} \\ w_{21} & w_{22} & \cdots & w_{2d} \\ & \vdots & & \\ w_{n1} & w_{n2} & \cdots & w_{nd} \end{pmatrix} \quad \boldsymbol{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} \quad \boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{pmatrix}$$

1. 1-layer Neural Net (L2 regression loss)



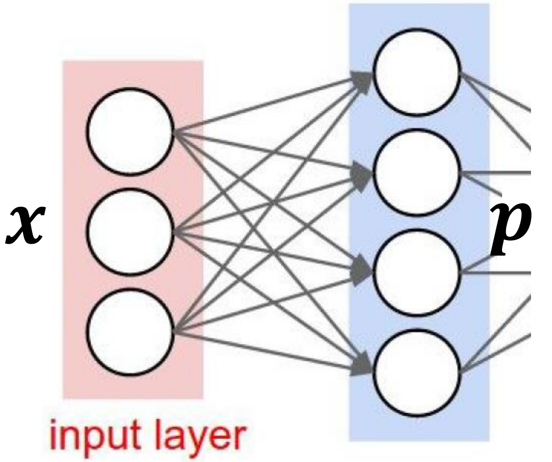


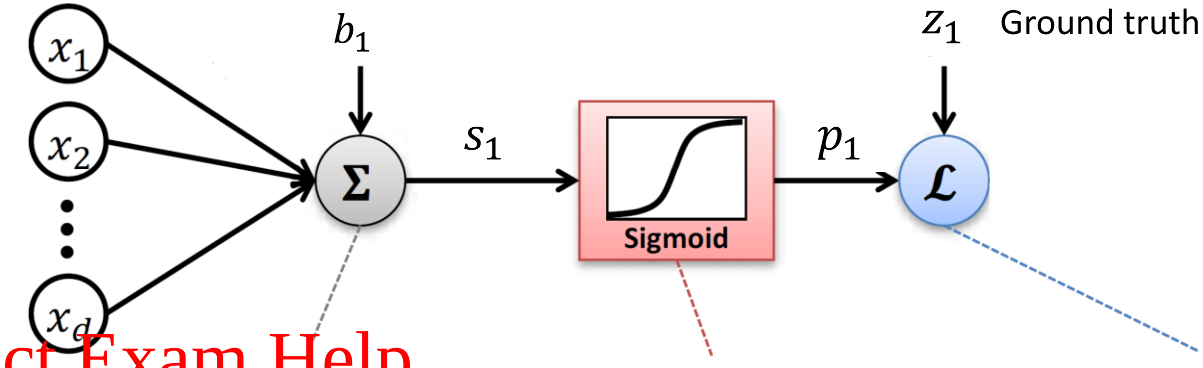
$$s_1 = \boldsymbol{w}_1^{\mathrm{T}}\boldsymbol{x} + b_1 = \sum_{k=1}^{d} w_{1k}x_k + b_1 \qquad p_1 = \frac{1}{1+e^{-s_1}} \qquad (z_1 - p_1)^2$$

1. Linear score $\boldsymbol{s} = \mathbf{W}\boldsymbol{x} + \boldsymbol{b} \longleftrightarrow s_j = \boldsymbol{w}_j^T\boldsymbol{x} + b_j$
2. Activation function $\boldsymbol{p} = \sigma(\boldsymbol{s}) = \dfrac{1}{1+e^{-\boldsymbol{s}}}$
3. Loss $L = (\boldsymbol{z} - \boldsymbol{p})^2$

$$\boldsymbol{s} = \begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_n \end{pmatrix} \quad \mathbf{W} = \begin{pmatrix} \boldsymbol{w}_1^T \\ \boldsymbol{w}_2^T \\ \vdots \\ \boldsymbol{w}_n^T \end{pmatrix} = \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1d} \\ w_{21} & w_{22} & \cdots & w_{2d} \\ & \vdots & & \\ w_{n1} & w_{n2} & \cdots & w_{nd} \end{pmatrix} \quad \boldsymbol{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} \quad \boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{pmatrix}$$

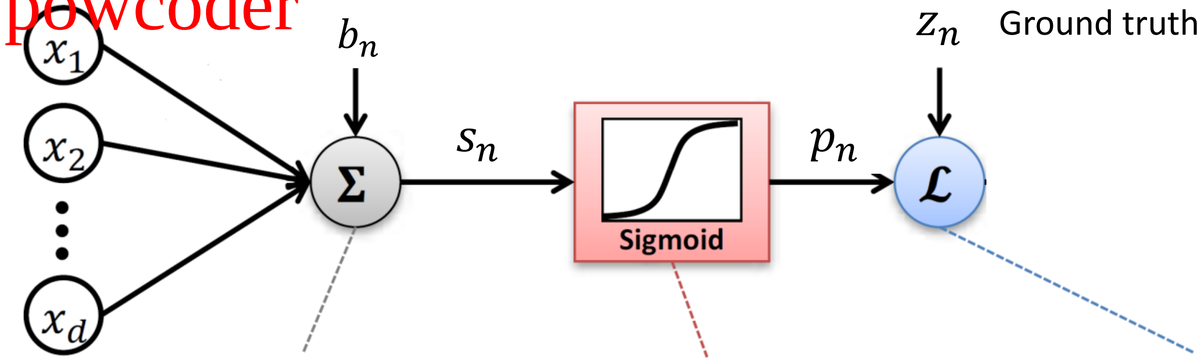
1. 1-layer Neural Net (L2 regression loss)





$$s_1 = \mathbf{w}_1^{\mathrm{T}}\boldsymbol{x} + b_1 = \sum_{k=1}^{d} w_{1k}x_k + b_1 \qquad p_1 = \frac{1}{1+e^{-s_1}} \qquad (z_1 - p_1)^2$$

$\vdots$

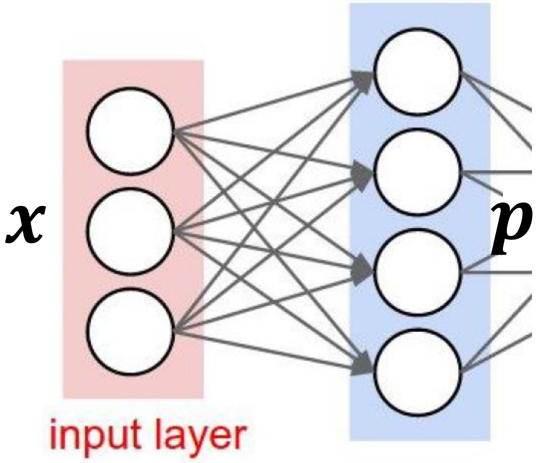


1. Linear score $\boldsymbol{s} = \mathbf{W}\boldsymbol{x} + \boldsymbol{b} \longleftrightarrow s_j = \boldsymbol{w}_j^T\boldsymbol{x} + b_j$
2. Activation function $\boldsymbol{p} = \sigma(\boldsymbol{s}) = \dfrac{1}{1+e^{-\boldsymbol{s}}}$
3. Loss $L = (\boldsymbol{z} - \boldsymbol{p})^2$

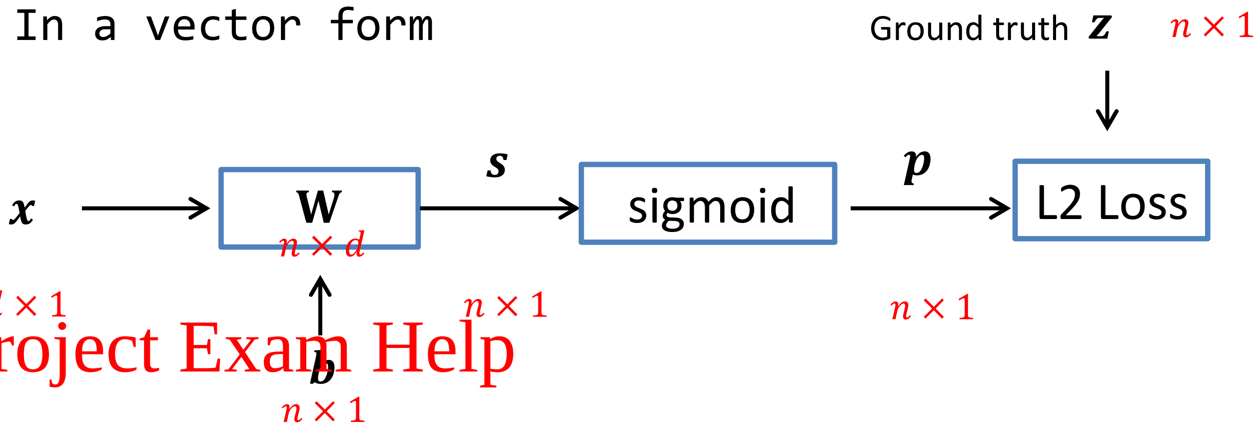
$$\boldsymbol{s} = \begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_n \end{pmatrix} \quad \mathbf{W} = \begin{pmatrix} \boldsymbol{w}_1^T \\ \boldsymbol{w}_2^T \\ \vdots \\ \boldsymbol{w}_n^T \end{pmatrix} = \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1d} \\ w_{21} & w_{22} & \cdots & w_{2d} \\ & \vdots & & \\ w_{n1} & w_{n2} & \cdots & w_{nd} \end{pmatrix} \quad \boldsymbol{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} \quad \boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{pmatrix}$$

$$s_n = \mathbf{w}_n^{\mathrm{T}}\boldsymbol{x} + b_n = \sum_{k=1}^{d} w_{nk}x_k + b_n \qquad p_n = \frac{1}{1+e^{-s_n}} \qquad (z_n - p_n)^2$$

1. 1-layer Neural Net (L2 regression loss)



Output layer



We need to compute gradients of $\mathbf{W}, \boldsymbol{b}, \boldsymbol{s}, \boldsymbol{p}$ with respect to the loss function L.

1. Linear score $\boldsymbol{s} = \mathbf{W}\boldsymbol{x} + \boldsymbol{b} \longleftrightarrow s_j = \boldsymbol{w}_j^T \boldsymbol{x} + b_j$
2. Activation function $\boldsymbol{p} = \sigma(\boldsymbol{s}) = \dfrac{1}{1 + e^{-\boldsymbol{s}}}$
3. Loss $L = (\boldsymbol{z} - \boldsymbol{p})^2$

$$\boldsymbol{s} = \begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_n \end{pmatrix} \quad \mathbf{W} = \begin{pmatrix} \boldsymbol{w}_1^T \\ \boldsymbol{w}_2^T \\ \vdots \\ \boldsymbol{w}_n^T \end{pmatrix} = \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1d} \\ w_{21} & w_{22} & \cdots & w_{2d} \\ & \vdots & & \\ w_{n1} & w_{n2} & \cdots & w_{nd} \end{pmatrix} \quad \boldsymbol{b} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} \quad \boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{pmatrix}$$

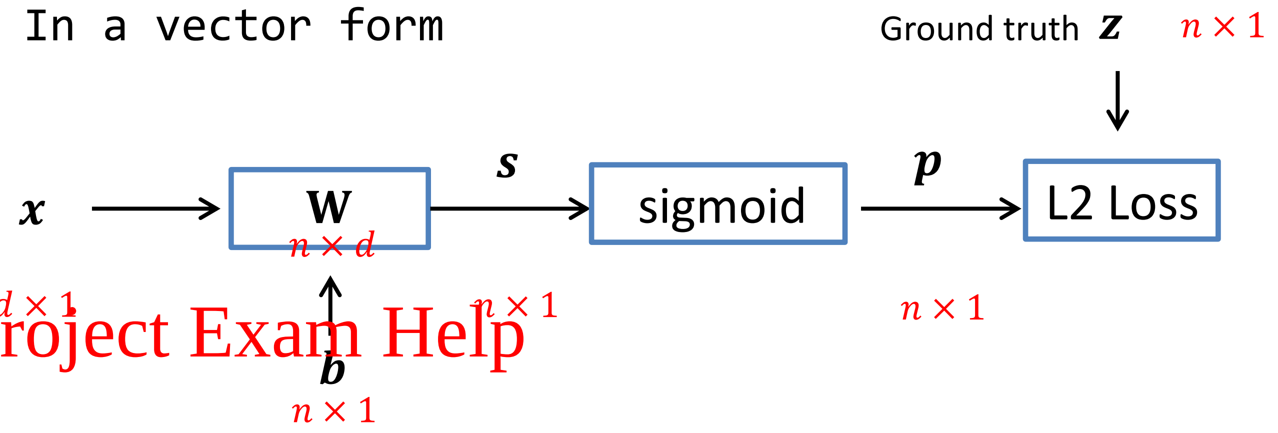
1. 1-layer Neural Net (L2 regression loss)

- **Gradient Descent**
  - The simplest approach to minimizing a loss function

    $$\mathbf{W}^{\mathrm{T}+1} = \mathbf{W}^{\mathrm{T}} - \alpha \frac{\partial L}{\partial \mathbf{W}^{\mathrm{T}}}$$

  - α: step size (a.k.a. learning rate)

In a vector form



$$\frac{\partial L}{\partial \boldsymbol{p}} = -2(\boldsymbol{z} - \boldsymbol{p})$$

$$\frac{\partial L}{\partial \boldsymbol{s}} = \frac{\partial \boldsymbol{p}}{\partial \boldsymbol{s}}\frac{\partial L}{\partial \boldsymbol{p}} = diag\big((1-\sigma(s_j))\sigma(s_j)\big)\frac{\partial L}{\partial \boldsymbol{p}} = -2\begin{bmatrix}(1-\sigma(s_1))\sigma(s_1)(z_1-p_1)\\(1-\sigma(s_2))\sigma(s_2)(z_2-p_2)\\\vdots\\(1-\sigma(s_n))\sigma(s_n)(z_n-p_n)\end{bmatrix} = (1-\sigma(\boldsymbol{s})) \otimes \sigma(\boldsymbol{s}) \otimes \frac{\partial L}{\partial \boldsymbol{p}}$$

$\otimes$: element-wise multiplication

j^th column

$$\frac{\partial L}{\partial \boldsymbol{w}_j} = \frac{\partial \boldsymbol{s}}{\partial \boldsymbol{w}_j}\frac{\partial L}{\partial \boldsymbol{s}} = \mathbf{X}_j \frac{\partial L}{\partial \boldsymbol{s}} = [\mathbf{0}\ \mathbf{0}\ \boldsymbol{x}\ \cdots\ \mathbf{0}]\frac{\partial L}{\partial \boldsymbol{s}} = \left(\frac{\partial L}{\partial \boldsymbol{s}}\right)_j \boldsymbol{x}$$

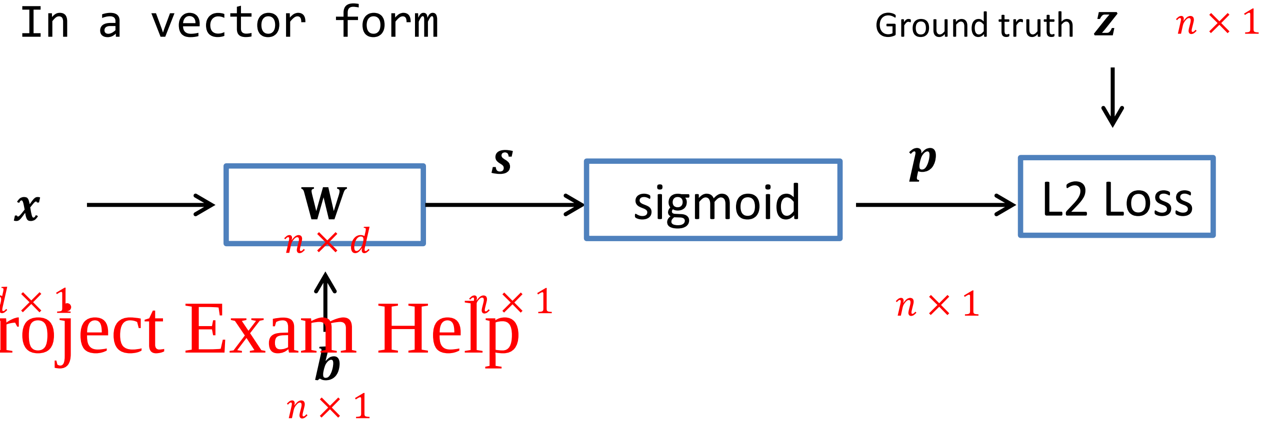
$(\boldsymbol{a})_j$: j^{th} element at vector $\boldsymbol{a}$

$$\frac{\partial L}{\partial \mathbf{W}} = \left(\frac{\partial L}{\partial \boldsymbol{w}_1}\quad \frac{\partial L}{\partial \boldsymbol{w}_2}\quad \cdots \quad \frac{\partial L}{\partial \boldsymbol{w}_n}\right)^{\mathrm{T}} = \frac{\partial L}{\partial \boldsymbol{s}}\boldsymbol{x}^{\mathrm{T}}$$

$$\frac{\partial L}{\partial \boldsymbol{b}} = \frac{\partial \boldsymbol{s}}{\partial \boldsymbol{b}}\frac{\partial L}{\partial \boldsymbol{s}} = \frac{\partial L}{\partial \boldsymbol{s}}$$

1. 1-layer Neural Net (L2 regression loss)

In a vector form



Summary

$$\frac{\partial L}{\partial \boldsymbol{p}} = -2(\boldsymbol{z} - \boldsymbol{p})$$

$$\frac{\partial L}{\partial \boldsymbol{s}} = \frac{\partial \boldsymbol{p}}{\partial \boldsymbol{s}} \frac{\partial L}{\partial \boldsymbol{p}} = (1 - \sigma(\boldsymbol{s})) \otimes \sigma(\boldsymbol{s}) \otimes \frac{\partial L}{\partial \boldsymbol{p}}$$

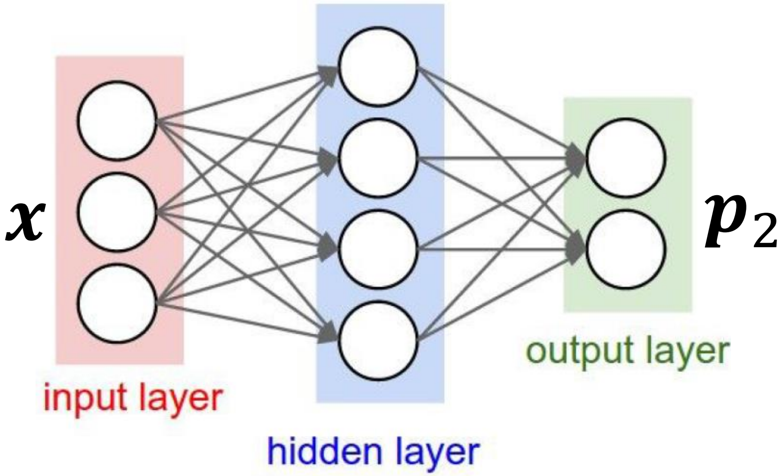
$$\frac{\partial L}{\partial \mathbf{W}} = \frac{\partial L}{\partial \boldsymbol{s}} \boldsymbol{x}^{\mathrm{T}}$$

$$\frac{\partial L}{\partial \boldsymbol{b}} = \frac{\partial L}{\partial \boldsymbol{s}}$$

Note that the following derivative can also be computed, but here $\boldsymbol{x}$ is an input data that is fixed during training. Thus, it is not necessary to compute its derivative.

$$\frac{\partial L}{\partial \boldsymbol{x}} = \frac{\partial \boldsymbol{s}}{\partial \boldsymbol{x}} \frac{\partial L}{\partial \boldsymbol{s}} = \mathbf{W}^{\mathrm{T}} \frac{\partial L}{\partial \boldsymbol{s}}$$

2. 2-layer Neural Net (L2 regression loss)



In a vector form

Ground truth $\boldsymbol{z}$ $m \times 1$

$\boldsymbol{x} \xrightarrow{} \boxed{\mathbf{W}_1} \xrightarrow{\boldsymbol{s}_1} \boxed{\text{sigmoid}} \xrightarrow{\boldsymbol{p}_1} \boxed{\mathbf{W}_2} \xrightarrow{\boldsymbol{s}_2} \boxed{\text{sigmoid}} \xrightarrow{\boldsymbol{p}_2} \boxed{\text{L2 Loss}}$

$\mathbf{W}_1$: $n \times d$; $\mathbf{W}_2$: $m \times n$

$\boldsymbol{x}$: $d \times 1$; $\boldsymbol{s}_1$: $n \times 1$; $\boldsymbol{p}_1$: $n \times 1$; $\boldsymbol{s}_2$: $m \times 1$; $\boldsymbol{p}_2$: $m \times 1$

$\boldsymbol{b}_1$ $n \times 1$; $\boldsymbol{b}_2$ $m \times 1$

$$\frac{\partial L}{\partial \boldsymbol{p_2}} = -2(\boldsymbol{z} - \boldsymbol{p_2})$$

$$\frac{\partial L}{\partial \boldsymbol{s}_2} = \frac{\partial \boldsymbol{p}_2}{\partial \boldsymbol{s}_2}\frac{\partial L}{\partial \boldsymbol{p}_2} = diag\big((1-\sigma(s_{2,j}))\sigma(s_{2,j})\big)\frac{\partial L}{\partial \boldsymbol{p}_2}$$

$$\frac{\partial L}{\partial \mathbf{W}_2} = \frac{\partial L}{\partial \boldsymbol{s}_2}\boldsymbol{p}_1^{\mathrm{T}} \qquad \frac{\partial L}{\partial \boldsymbol{b}_2} = \frac{\partial L}{\partial \boldsymbol{s}_2}$$

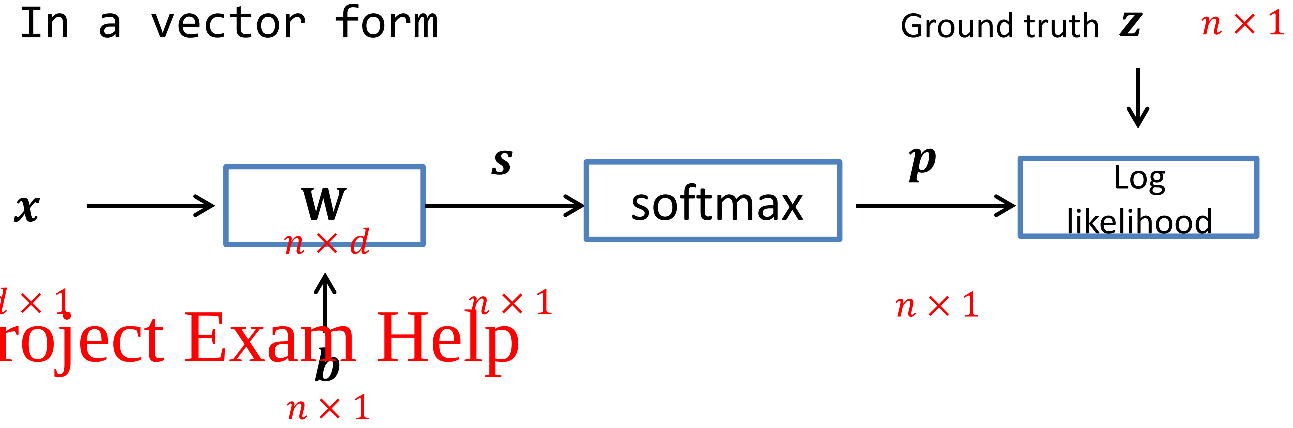
$$\frac{\partial L}{\partial \boldsymbol{p}_1} = \frac{\partial \boldsymbol{s}_2}{\partial \boldsymbol{p}_1}\frac{\partial L}{\partial \boldsymbol{s}_2} = \mathbf{W}_2^{\mathrm{T}}\frac{\partial L}{\partial \boldsymbol{s}_2}$$

$$\frac{\partial L}{\partial \boldsymbol{s}_1} = \frac{\partial \boldsymbol{p}_1}{\partial \boldsymbol{s}_1}\frac{\partial L}{\partial \boldsymbol{p}_1} = diag\big((1-\sigma(s_{1,j}))\sigma(s_{1,j})\big)\frac{\partial L}{\partial \boldsymbol{p}_1}$$

$$\frac{\partial L}{\partial \mathbf{W}_1} = \frac{\partial L}{\partial \boldsymbol{s}_1}\boldsymbol{x}^{\mathrm{T}} \qquad \frac{\partial L}{\partial \boldsymbol{b}_1} = \frac{\partial L}{\partial \boldsymbol{s}_1}$$

3. 1-layer Neural Net (Softmax classifier)

In a vector form



$$\frac{\partial L}{\partial \boldsymbol{p}} = \begin{array}{cl} \begin{bmatrix} 0 \\ 0 \\ -1/p_y \\ \vdots \\ 0 \end{bmatrix} & \text{y}^{\text{th}} \text{ row} \end{array}$$

$$\frac{\partial L}{\partial \boldsymbol{s}} = \frac{\partial \boldsymbol{p}}{\partial \boldsymbol{s}} \frac{\partial L}{\partial \boldsymbol{p}} = \mathbf{D} \frac{\partial L}{\partial \boldsymbol{p}} = -\frac{1}{p_y} \begin{bmatrix} D_{1y} \\ D_{2y} \\ \vdots \\ D_{ny} \end{bmatrix} = \boldsymbol{p} - \boldsymbol{z}$$

$$D_{ab} = p_a(\delta_{ab} - p_b)$$

$$\delta_{ab} = \begin{cases} 1 & a = b \\ 0 & \text{otherwise} \end{cases}$$

$$\frac{\partial L}{\partial \boldsymbol{w}_j} = \frac{\partial \boldsymbol{s}}{\partial \boldsymbol{w}_j} \frac{\partial L}{\partial \boldsymbol{s}} = \mathbf{X}_j \frac{\partial L}{\partial \boldsymbol{s}} = \begin{array}{c} \text{j}^{\text{th}} \text{ column} \\ [\mathbf{0}\ \mathbf{0}\ \boldsymbol{x}\ \cdots \mathbf{0}] \end{array} \frac{\partial L}{\partial \boldsymbol{s}} = \left(\frac{\partial L}{\partial \boldsymbol{s}}\right)_j \boldsymbol{x}$$

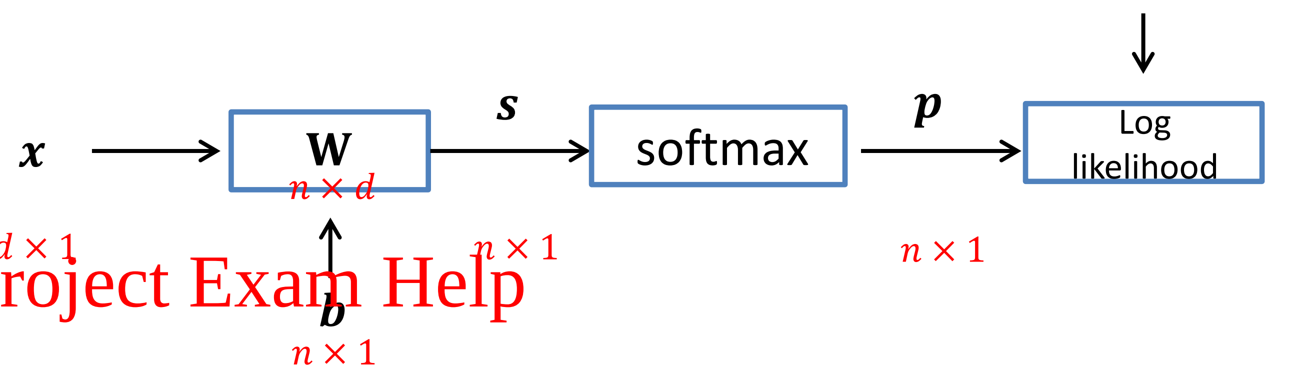
$(\boldsymbol{a})_j$: j$^{\text{th}}$ element at vector $\boldsymbol{a}$

$$\frac{\partial L}{\partial \mathbf{W}} = \left(\frac{\partial L}{\partial \boldsymbol{w}_1} \quad \frac{\partial L}{\partial \boldsymbol{w}_2} \quad \cdots \quad \frac{\partial L}{\partial \boldsymbol{w}_n}\right)^{\mathrm{T}} = \frac{\partial L}{\partial \boldsymbol{s}} \boldsymbol{x}^{\mathrm{T}}$$

$$\frac{\partial L}{\partial \boldsymbol{b}} = \frac{\partial \boldsymbol{s}}{\partial \boldsymbol{b}} \frac{\partial L}{\partial \boldsymbol{s}} = \frac{\partial L}{\partial \boldsymbol{s}}$$

3. 1-layer Neural Net (Softmax classifier)

In a vector form



Summary

$$\frac{\partial L}{\partial \boldsymbol{p}} = \begin{bmatrix} 0 \\ 0 \\ -1/p_y \\ \vdots \\ 0 \end{bmatrix} \leftarrow y^{\text{th}} \text{ row}$$

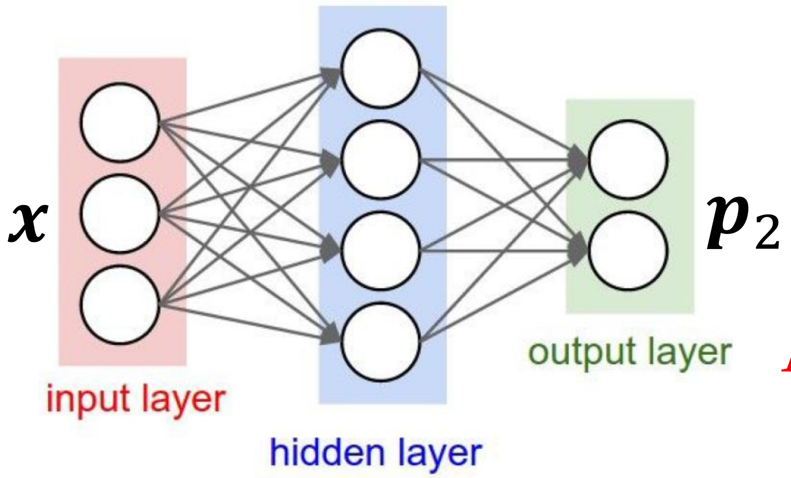
$$\frac{\partial L}{\partial \boldsymbol{s}} = \frac{\partial \boldsymbol{p}}{\partial \boldsymbol{s}} \frac{\partial L}{\partial \boldsymbol{p}} = \mathbf{D} \frac{\partial L}{\partial \boldsymbol{p}} = -\frac{1}{p_y} \begin{bmatrix} D_{1y} \\ D_{2y} \\ \vdots \\ D_{ny} \end{bmatrix} = \boldsymbol{p} - \boldsymbol{z}$$

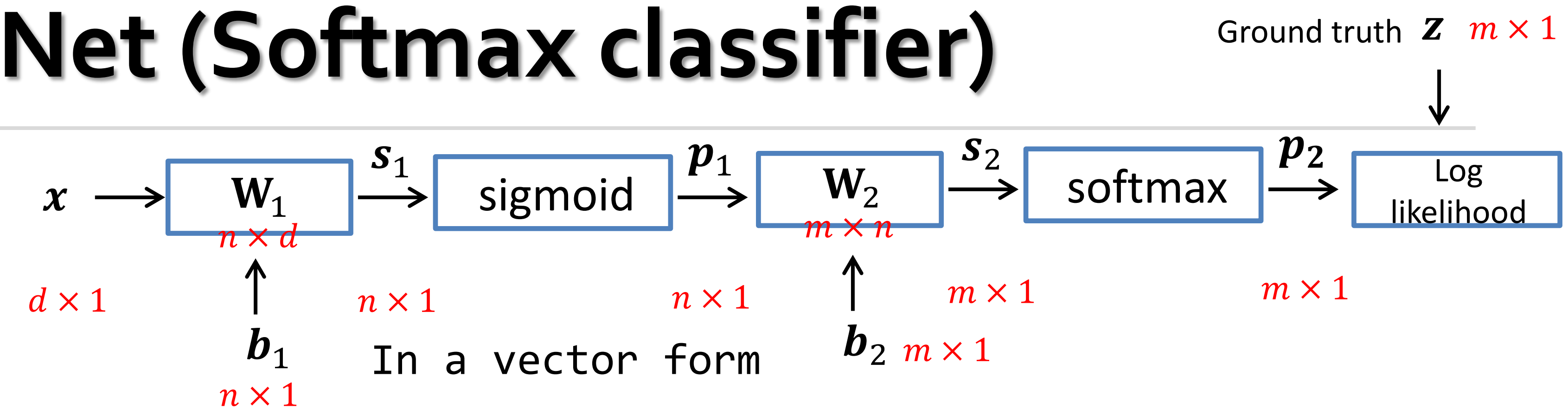
$$\frac{\partial L}{\partial \mathbf{W}} = \frac{\partial L}{\partial \boldsymbol{s}} \boldsymbol{x}^{\mathrm{T}} \qquad \frac{\partial L}{\partial \boldsymbol{b}} = \frac{\partial L}{\partial \boldsymbol{s}}$$

Note that the following derivative can also be computed, but here $\boldsymbol{x}$ is an input data that is fixed during training. Thus, it is not necessary to compute its derivative.

$$\frac{\partial L}{\partial \boldsymbol{x}} = \frac{\partial \boldsymbol{s}}{\partial \boldsymbol{x}} \frac{\partial L}{\partial \boldsymbol{s}} = \mathbf{W}^{\mathrm{T}} \frac{\partial L}{\partial \boldsymbol{s}}$$

4. 2-layer Neural Net (Softmax classifier)





$$\frac{\partial L}{\partial \boldsymbol{p}_2} = \begin{bmatrix} 0 \\ 0 \\ -1/p_y \\ \vdots \\ 0 \end{bmatrix}$$

y^{th} row

$$\frac{\partial L}{\partial \boldsymbol{s}_2} = \frac{\partial \boldsymbol{p}_2}{\partial \boldsymbol{s}_2}\frac{\partial L}{\partial \boldsymbol{p}_2} = \mathbf{D}\frac{\partial L}{\partial \boldsymbol{p}_2}$$

$$D_{ab} = p_a(\delta_{ab} - p_b)$$

$$\delta_{ab} = \begin{cases} 1 & a = b \\ 0 & \text{otherwise} \end{cases}$$

$$\frac{\partial L}{\partial \mathbf{W}_2} = \frac{\partial L}{\partial \boldsymbol{s}_2}\boldsymbol{p}_1^{\mathrm{T}}$$

$$\frac{\partial L}{\partial \boldsymbol{p}_1} = \frac{\partial \boldsymbol{s}_2}{\partial \boldsymbol{p}_1}\frac{\partial L}{\partial \boldsymbol{s}_2} = \mathbf{W}_2^{\mathrm{T}}\frac{\partial L}{\partial \boldsymbol{s}_2} \qquad \frac{\partial L}{\partial \boldsymbol{b}_2} = \frac{\partial L}{\partial \boldsymbol{s}_2}$$

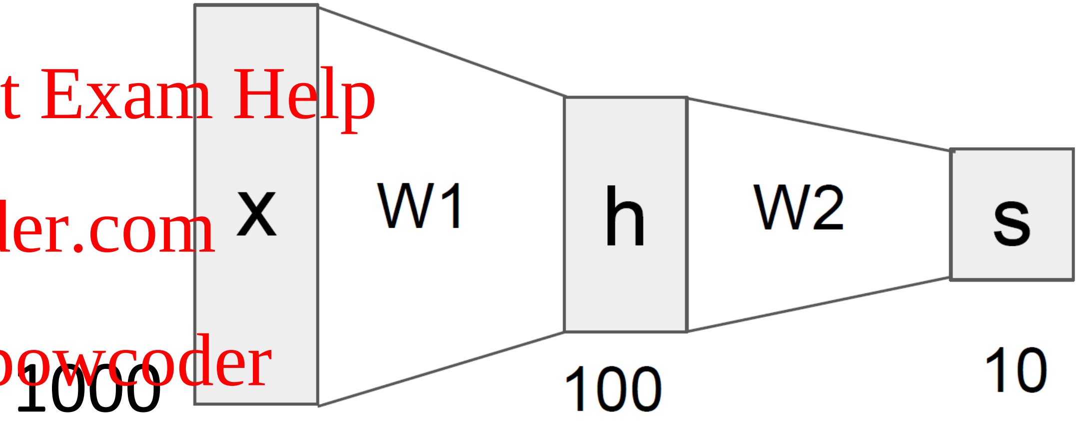
$$\frac{\partial L}{\partial \boldsymbol{s}_1} = \frac{\partial \boldsymbol{p}_1}{\partial \boldsymbol{s}_1}\frac{\partial L}{\partial \boldsymbol{p}_1} = diag\big((1-\sigma(s_{1,j}))\sigma(s_{1,j})\big)\frac{\partial L}{\partial \boldsymbol{p}_1}$$

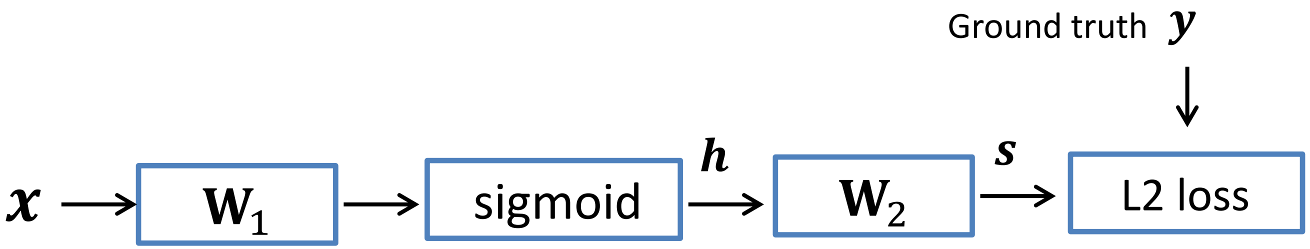
$$\frac{\partial L}{\partial \mathbf{W}_1} = \frac{\partial L}{\partial \boldsymbol{s}_1}\boldsymbol{x}^{\mathrm{T}} \qquad \frac{\partial L}{\partial \boldsymbol{b}_1} = \frac{\partial L}{\partial \boldsymbol{s}_1}$$

Full implementation of training a 2-layer Neural Network

```python
import numpy as np
from numpy.random import randn

N, D_in, H, D_out = 64, 1000, 100, 10
x, y = randn(N, D_in), randn(N, D_out)
w1, w2 = randn(D_in, H), randn(H, D_out)

for t in range(2000):
  h = 1 / (1 + np.exp(-x.dot(w1)))
  y_pred = h.dot(w2)
  loss = np.square(y_pred - y).sum()
  print(t, loss)

  grad_y_pred = 2.0 * (y_pred - y)
  grad_w2 = h.T.dot(grad_y_pred)
  grad_h = grad_y_pred.dot(w2.T)
  grad_w1 = x.T.dot(grad_h * h * (1 - h))

  w1 -= 1e-4 * grad_w1
  w2 -= 1e-4 * grad_w2
```

N: batch size
D_in: input feature size
H: input feature size of the second layer
D_out: output feature size





Neural networks: Pros and cons

- **Pros**
  - Flexible and general function approximation framework
  - Can build extremely powerful models by adding more layers
- **Cons**
  - Hard to analyze theoretically (e.g., training is prone to local optima)
  - Huge amount of training data, computing power may be required to get good perfor mance
  - The space of implementation choices is huge (network architectures, parameters)

Summary

- **We arrange neurons into fully-connected layers**

- **The layer allows us to use efficient vectorized code (e.g. matrix multiplication)**
  - Using back-propagation

EBU7240

Computer Vision

- Convolutional Neural Networks -

Semester 1, 2021

Changjae Oh

CNN Introduction

- **Image Recognition**
  - Recognizing the object class in the image

http://www.cs.toronto.edu/~fritz/absps/imagenet.pdf

CNN (= ConvNet)

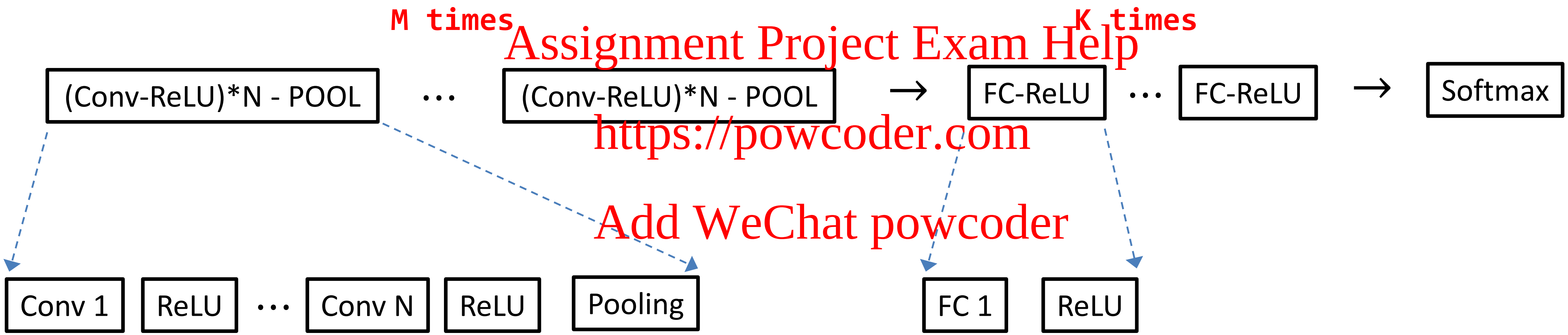
- **ConvNet**
  - is a sequence of layers
  - Every layer of a ConvNet transforms one volume of activations to another through a differentiable function.

  - **Convolutional Layer**: computes the output of neurons that are connected to local regions in the input

  - **ReLU (nonlinear) layer**: activates relevant responses
  - **Pooling Layer**: performs a downsampling operation along the spatial dimensions
  - **Fully-Connected Layer**: each neuron in this layer will be connected to all the numbers in the previous volume

Typical architectures of ConvNet

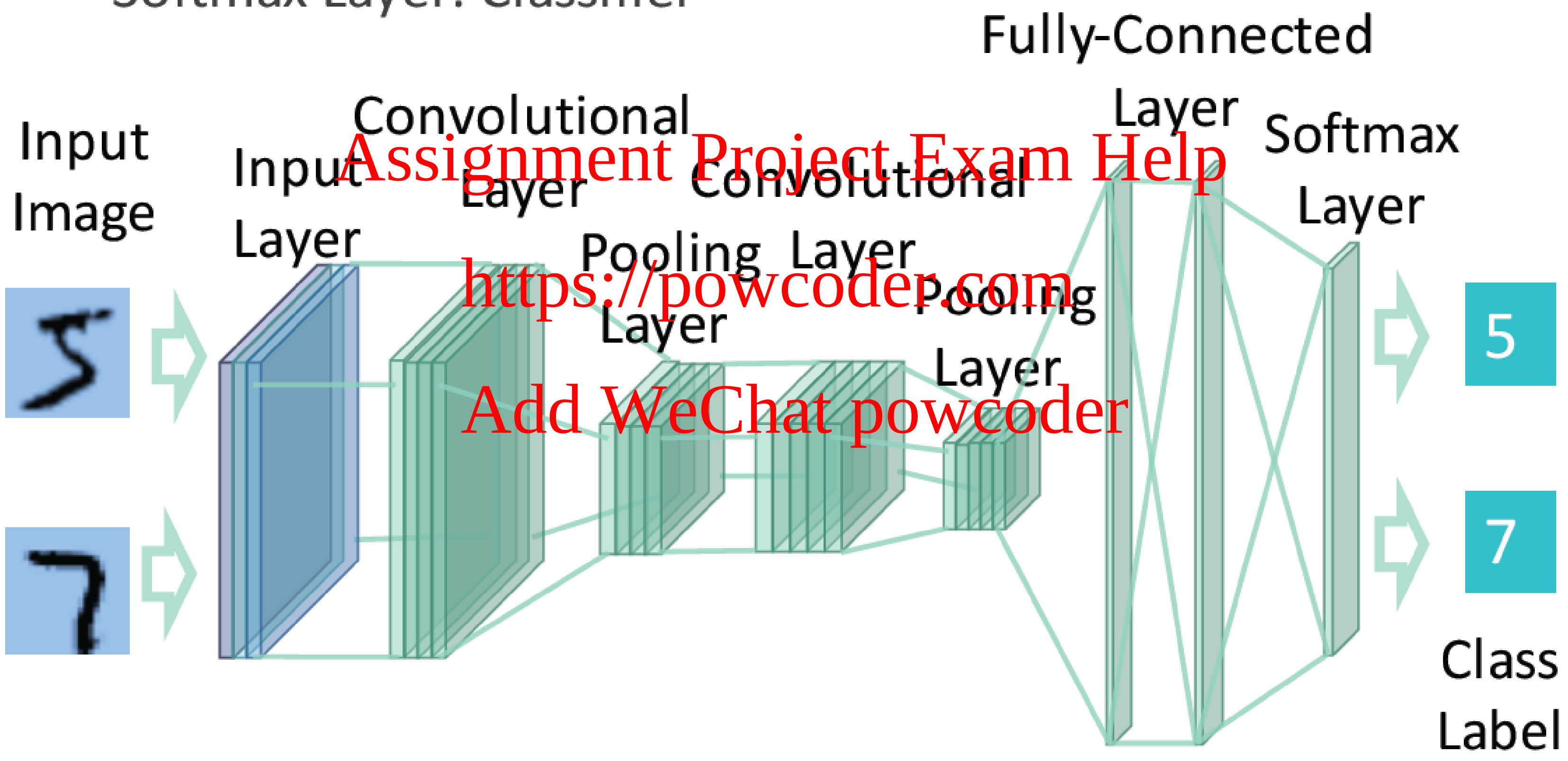
`[(Conv-ReLU)*N - POOL]*M - (FC-RELU)*K - Softmax`



N is usually up to ~5, M is large, 0 <= K <= 2
but some advances such as ResNet/GoogLeNet challenge this paradigm

Typical architectures of ConvNet

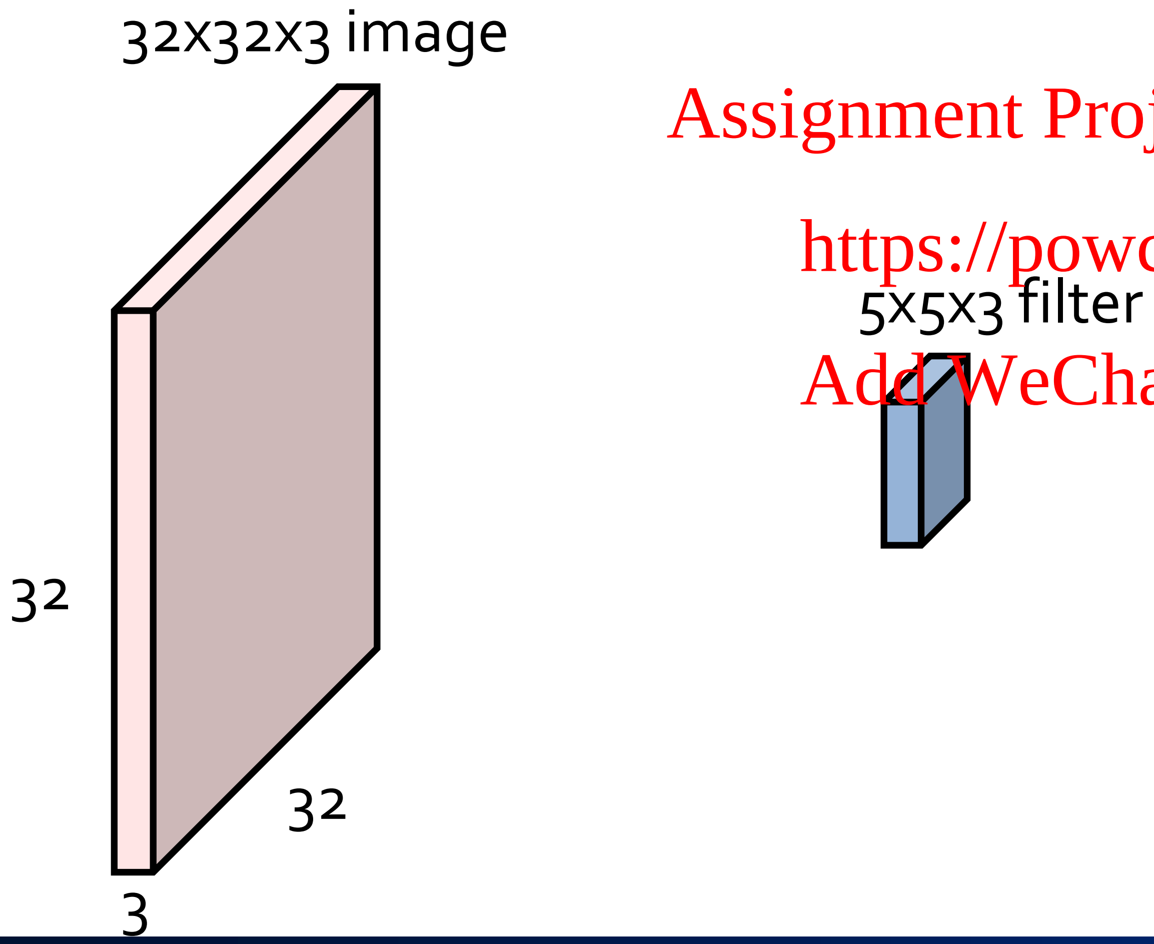
- Fully-Connected Layers : Global feature extraction
- Softmax Layer: Classifier



Convolutional Layer

Convolutional Layer

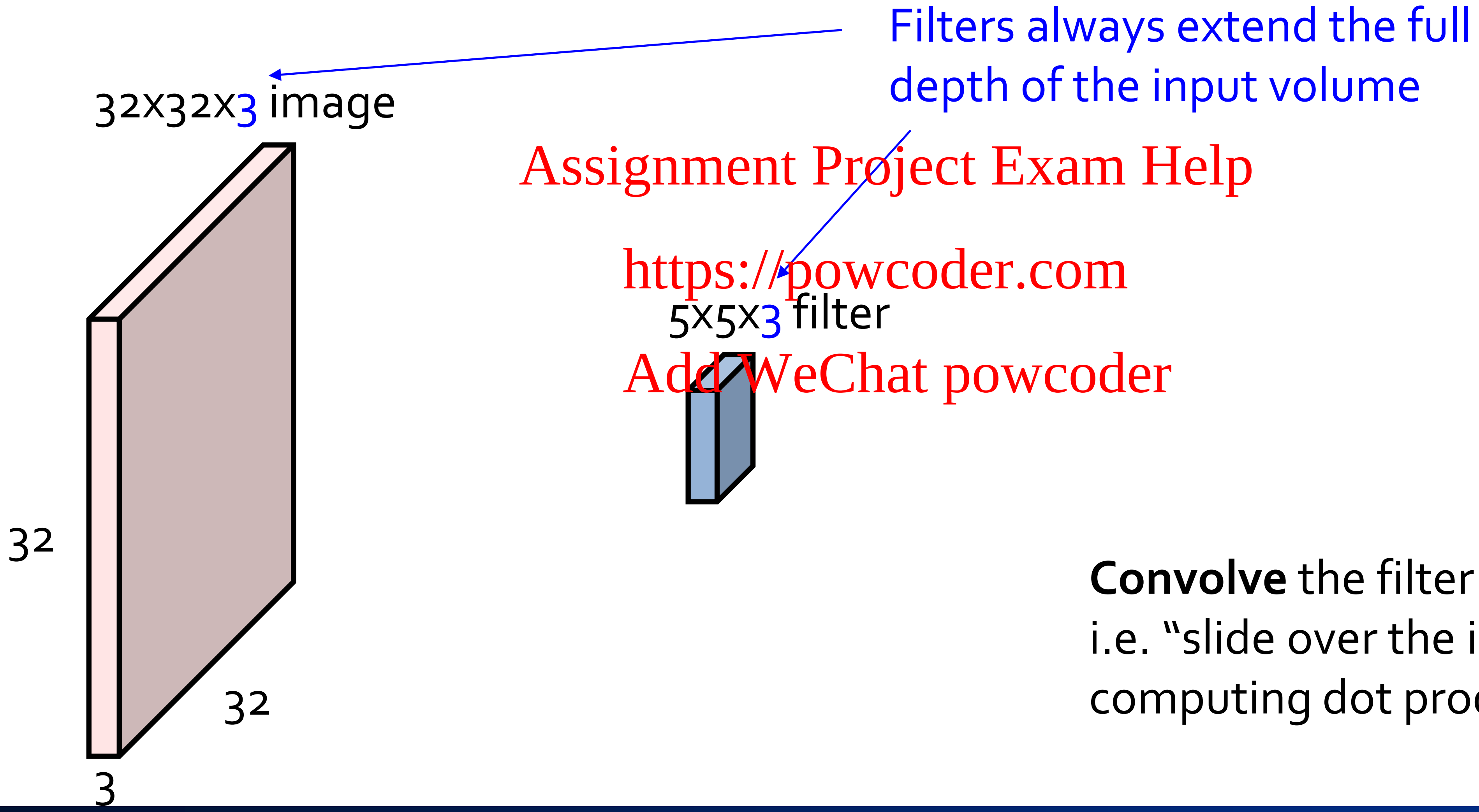
To preserve spatial structure, use an original 2D image



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

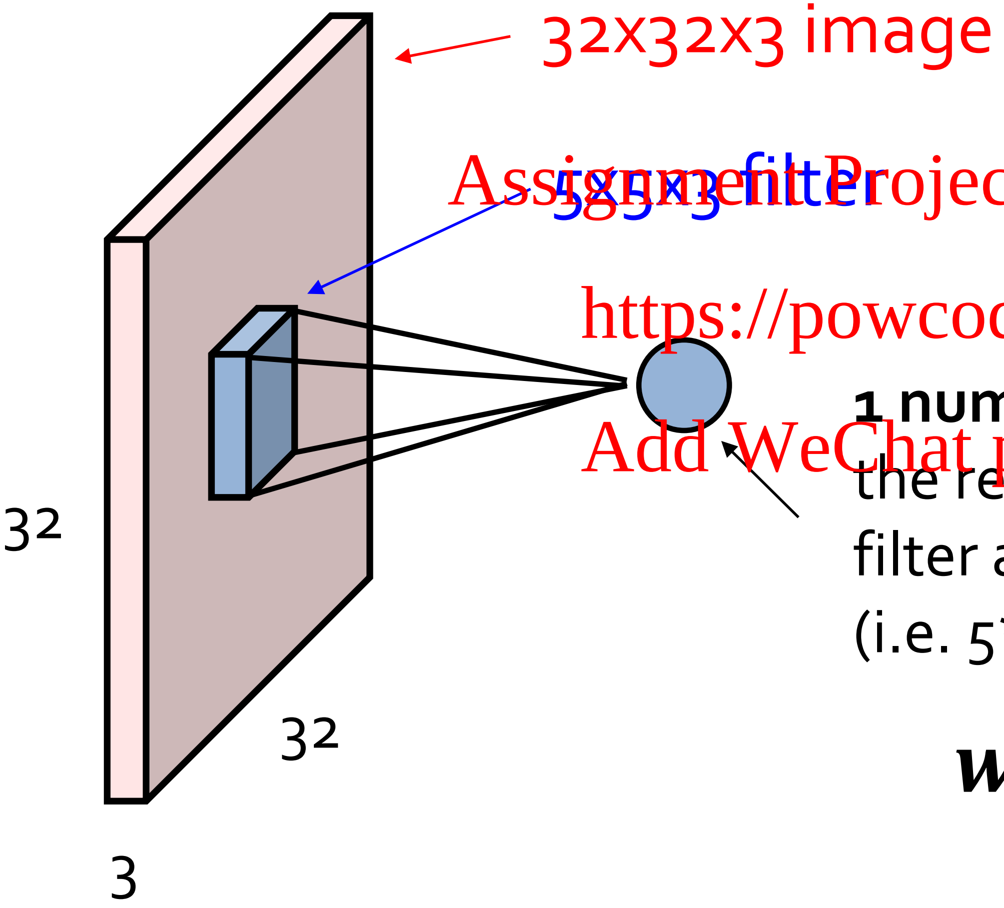
Convolutional Layer

To preserve spatial structure, use an original 2D image



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

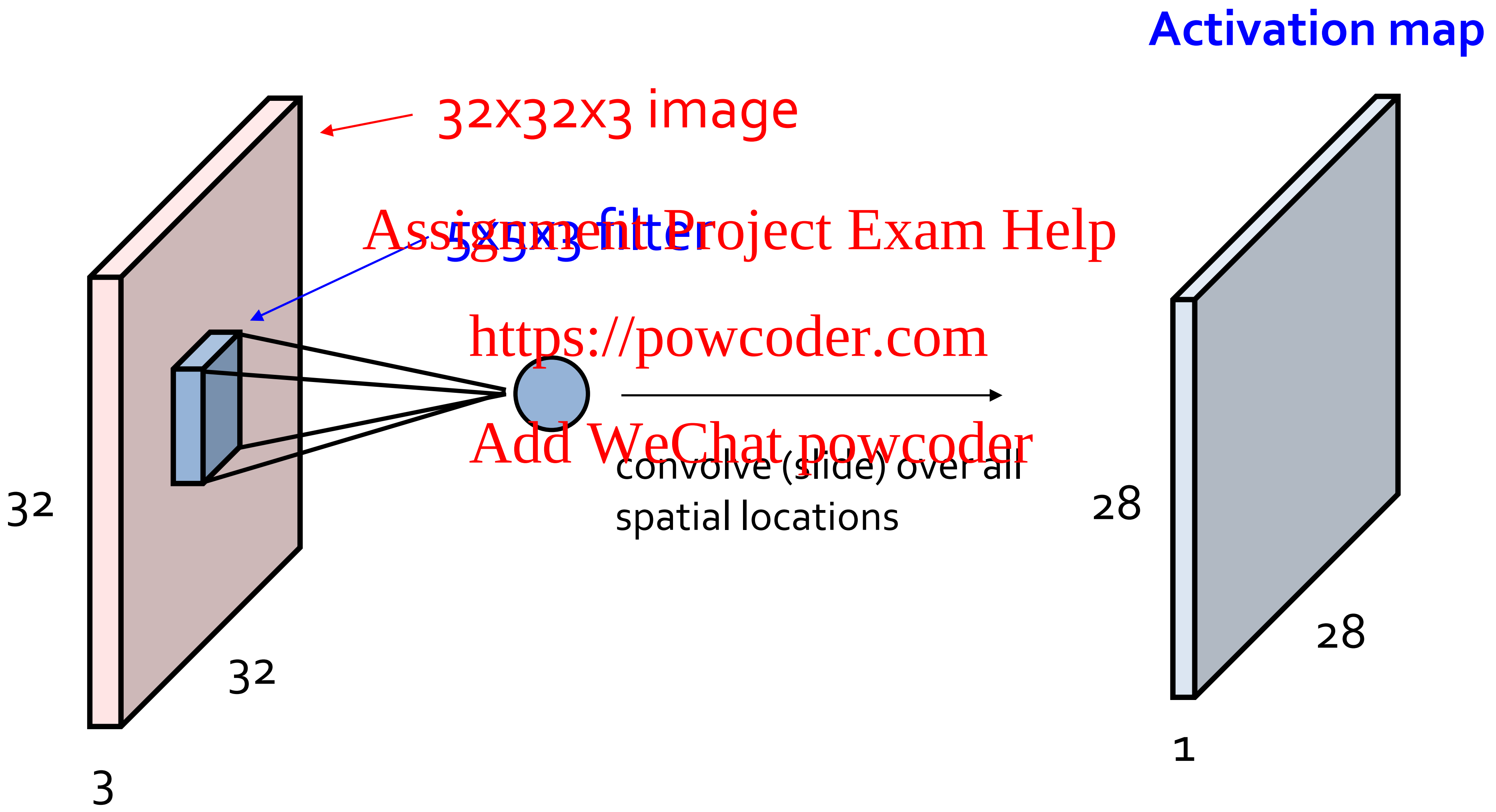
Convolutional Layer



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)

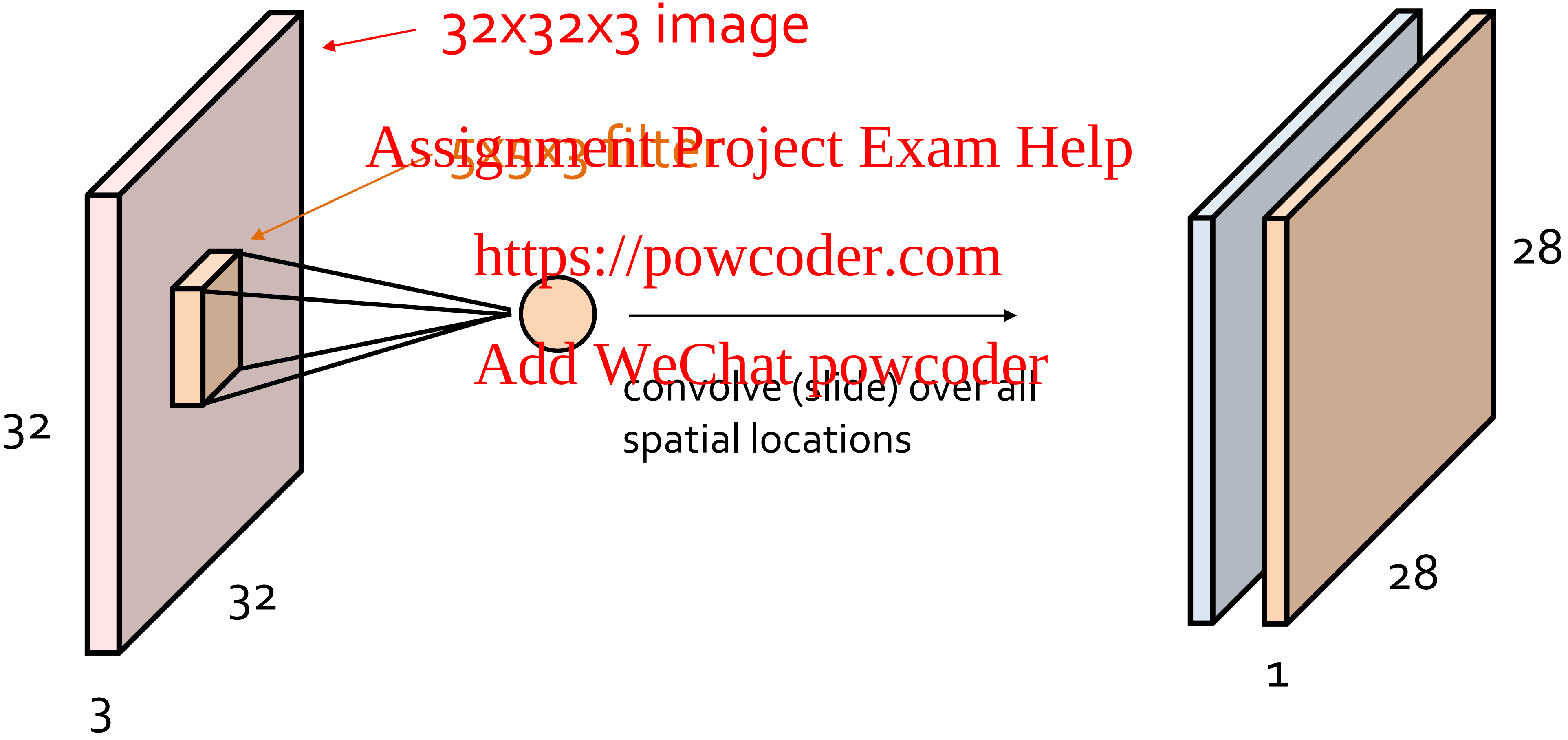
$$\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + \boldsymbol{b}$$

Convolutional Layer



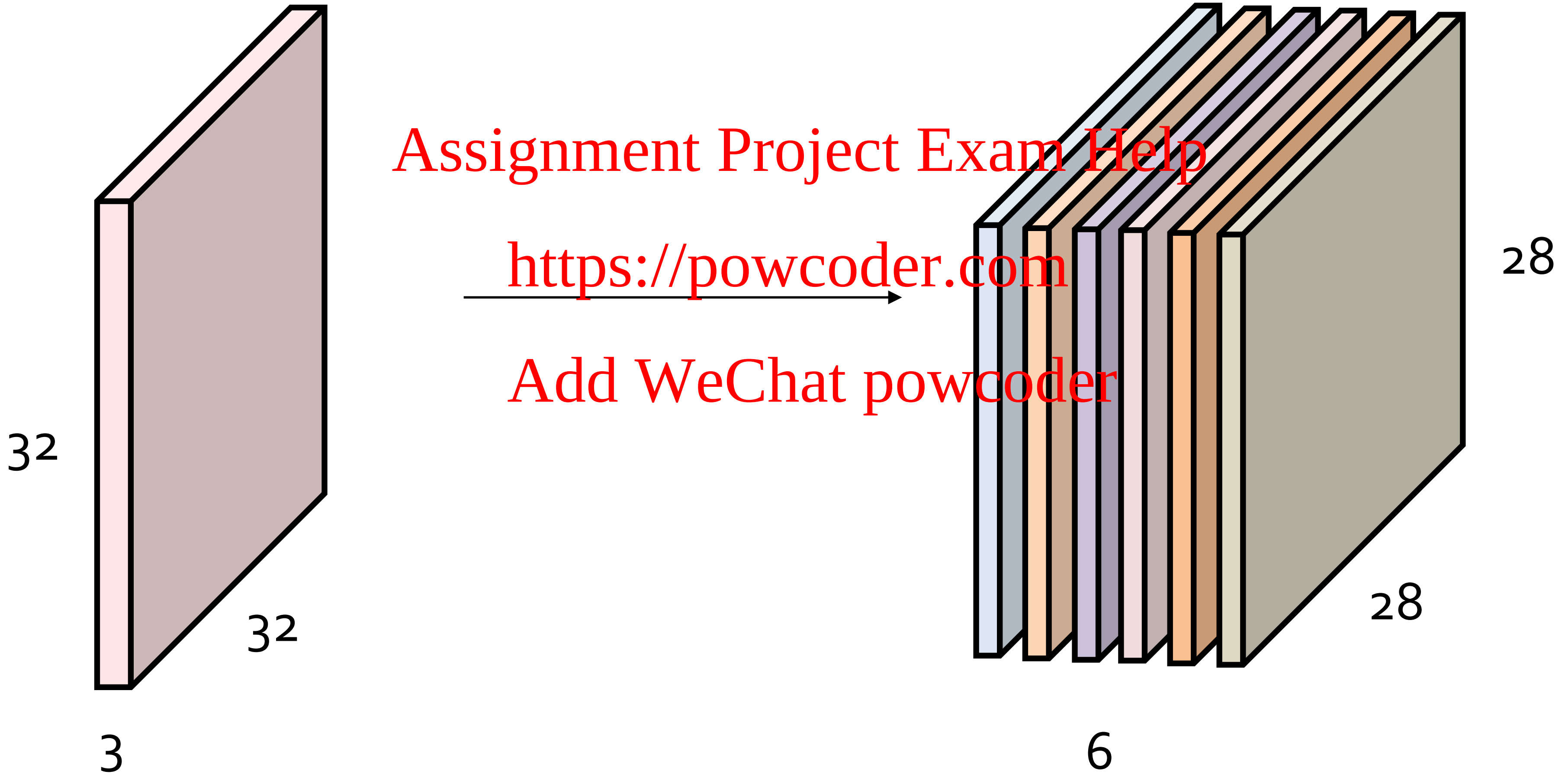
Convolutional Layer

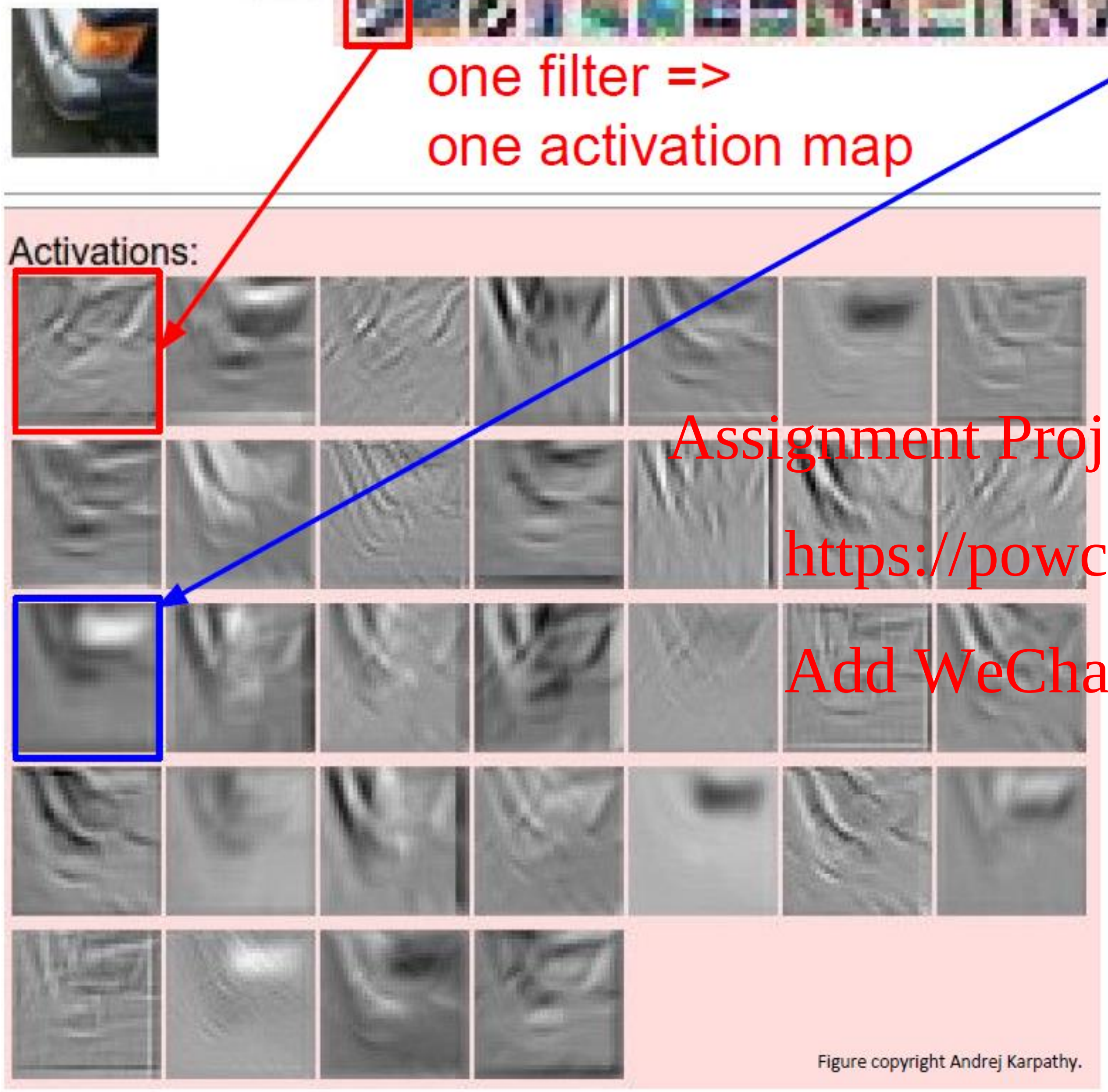
consider a second 5x5x3 (orange) filter



Convolutional Layer

If we had **six** 5x5x3 filters, we'll get **six** separate activation maps:





example 5x5 filters
(32 total)

We call the layer convolutional because it is related to convolution of two signals:

$$f[x,y]*g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1, y-n_2]$$

elementwise multiplication and sum of a filter and the signal (image)

Convolutional Layer

- **The number of parameters in convolutional layer**

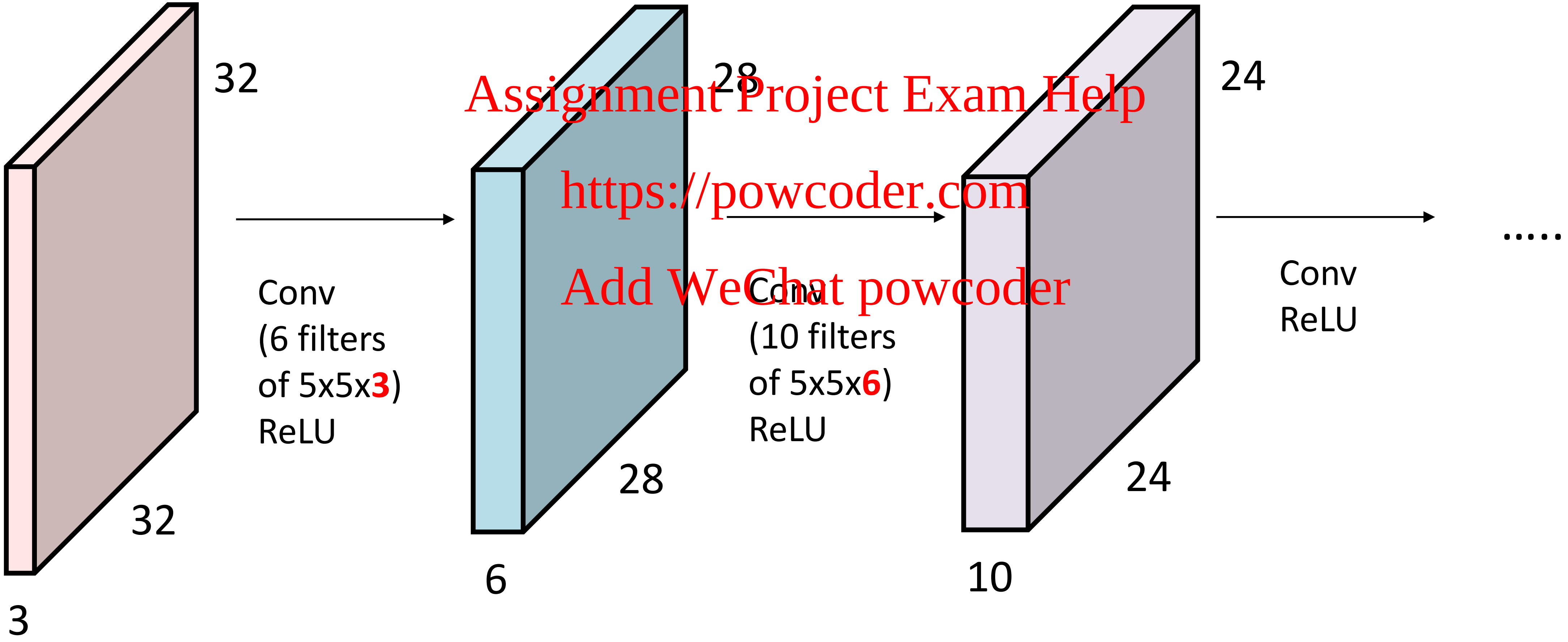
Input	Weight	Output
$H_1 \times W_1 \times C_1$	C_2 filters of $F_h \times F_v \times C_1$	$H_1 \times W_1 \times C_2$

The number of weights: $C_2 \times (F_h \times F_v \times C_1)$

The number of bias: C_2

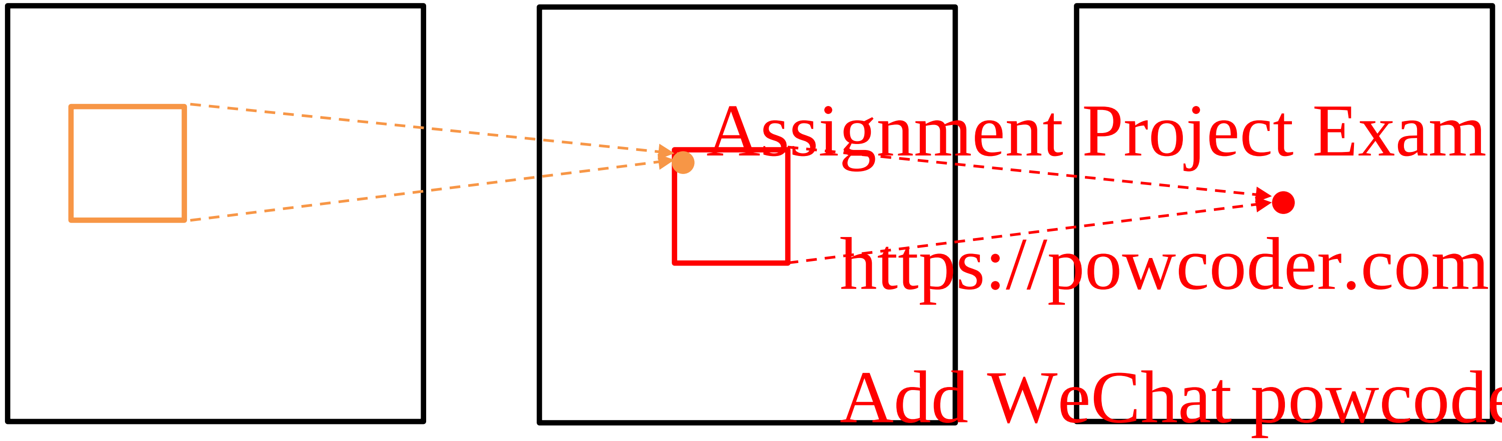
ConvNet

- A sequence of Convolutional Layers, interspersed with activation functions



ConvNet

- **Receptive field**
  - The region of the input space that affects a particular unit of the network



$5 \times 5 \times 1$ filter $\qquad$ $5 \times 5 \times 1$ filter

Effective receptive field at $: 9 \times 9 \ (= 5 + 5 - 1)$

From the convolution property,

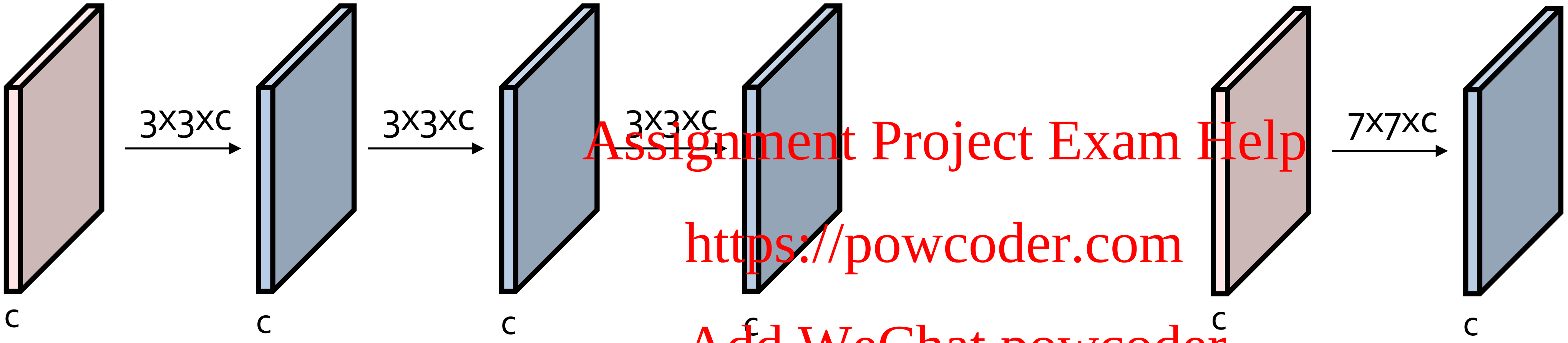
$$y = x * h_1$$
$$z = y * h_2$$

$$\rightarrow z = x * h_1 * h_2 = x * h$$

$$\boxed{h = h_1 * h_2}$$

Convolutional Filter Size

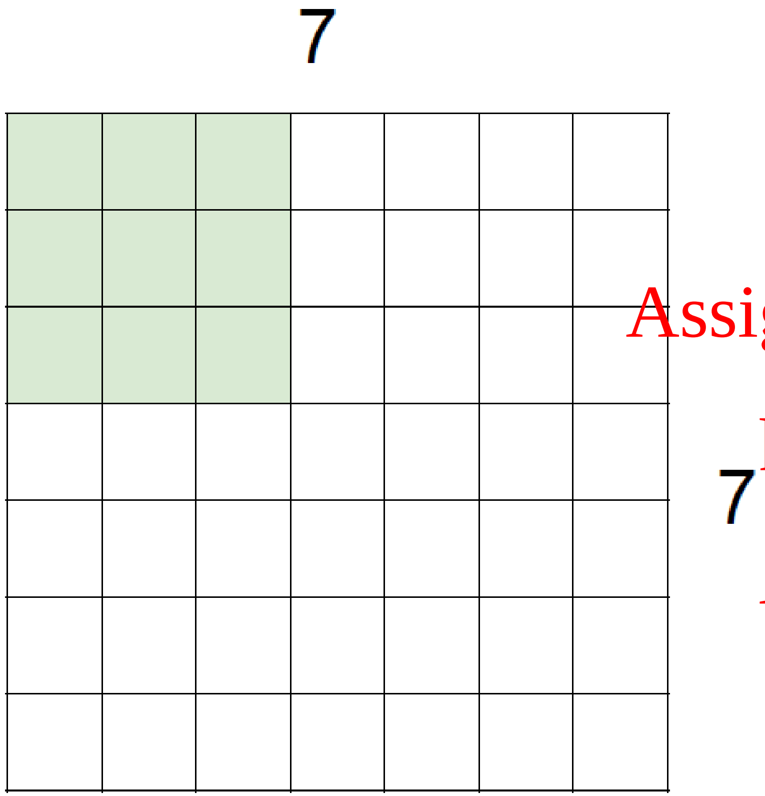
Three 3x3 Conv layers vs. Single 7x7 Conv layer



Assume that zero-padding is applied to preserve a spatial resolution

- **Receptive fields are equal. For three 3x3 Conv layers, 3+3-1+3-1 = 7.**
- **# of Conv parameters:** $3 \times C \times (3 \times 3 \times C) = 27C^2$ **vs.** $C \times (7 \times 7 \times C) = 49C^2$
- **The three stacks of CONV layers produce more *expressive* activation maps**

Spatial Dimension at Convolution Layer



7x7 input (spatially)

assume 3x3 filter applied **with stride 1**

-> 5x5 output

assume 3x3 filter applied **with stride 2**

-> 3x3 output

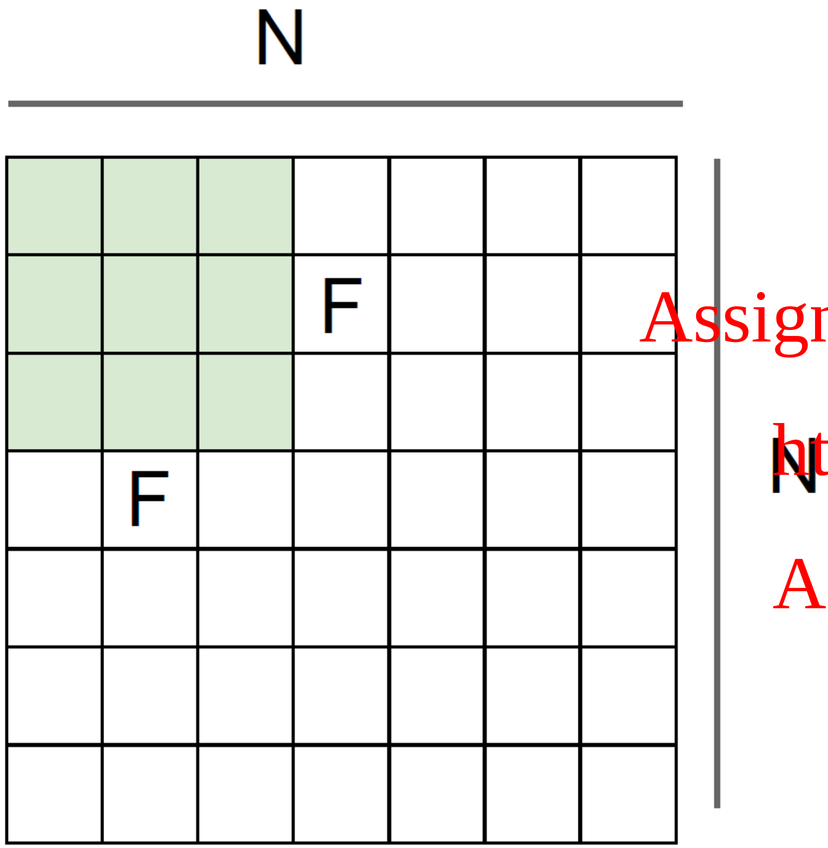
assume 3x3 filter applied **with stride 3**

-> **doesn't fit!**

cannot apply 3x3 filter on 7x7 input with stride 3.

***Stride:** is the number of pixels shifts over the input matrix

Spatial Dimension at Convolution Layer



Output size:

(N - F) / stride + 1

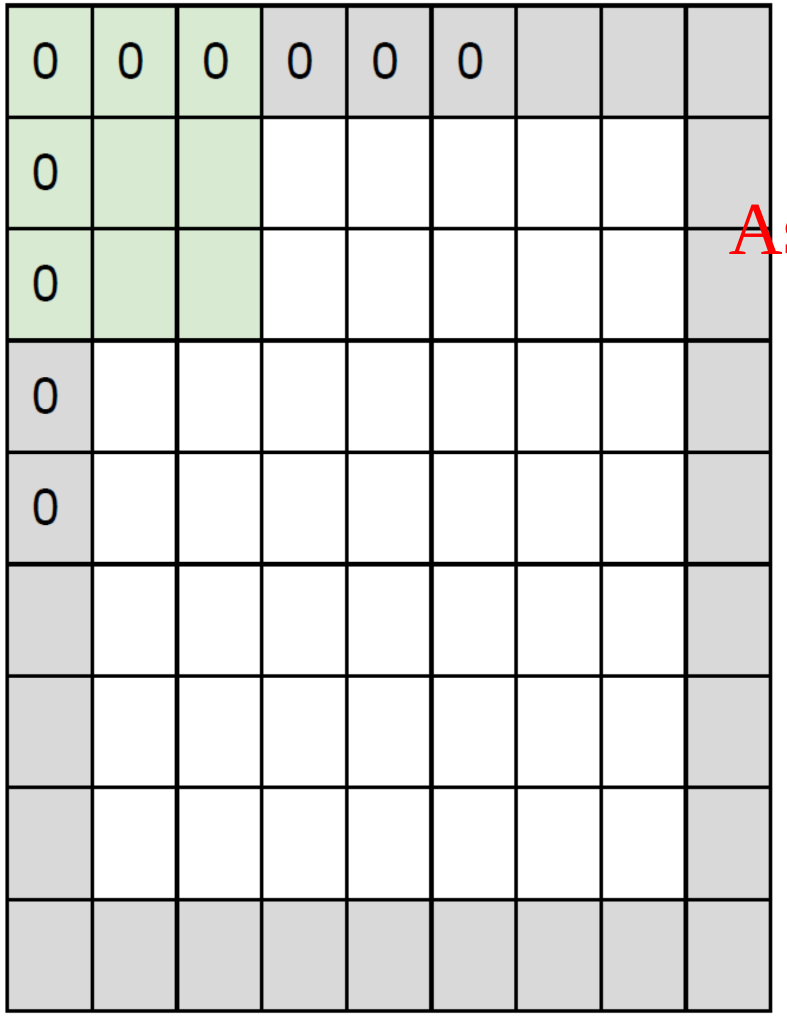
e.g. N = 7, F = 3:

stride 1 => (7 - 3)/1 + 1 = 5

stride 2 => (7 - 3)/2 + 1 = 3

stride 3 => (7 - 3)/3 + 1 = 2.33

Spatial Dimension at Convolution Layer



In practice: Common to zero pad the border

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

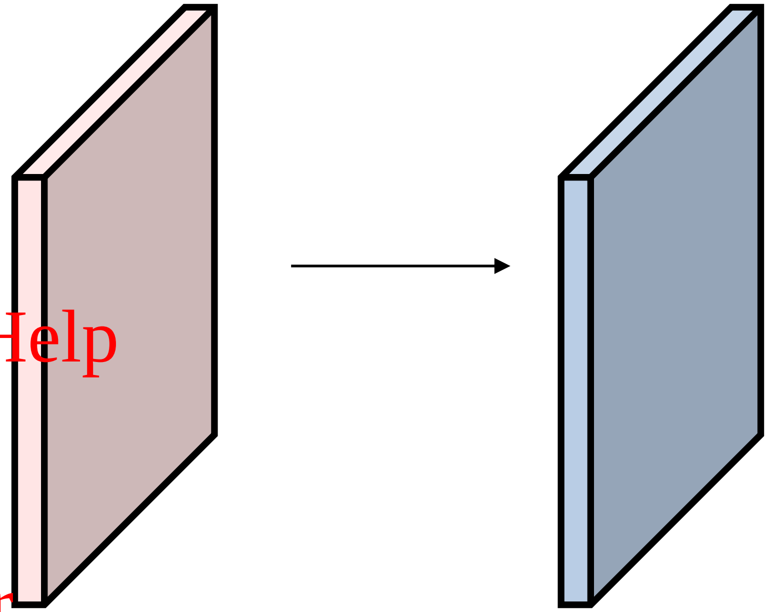
in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)

e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

Spatial Dimension at Convolution Layer

Input volume: **32x32x3**

10 5x5x3 filters with stride 1, pad 2

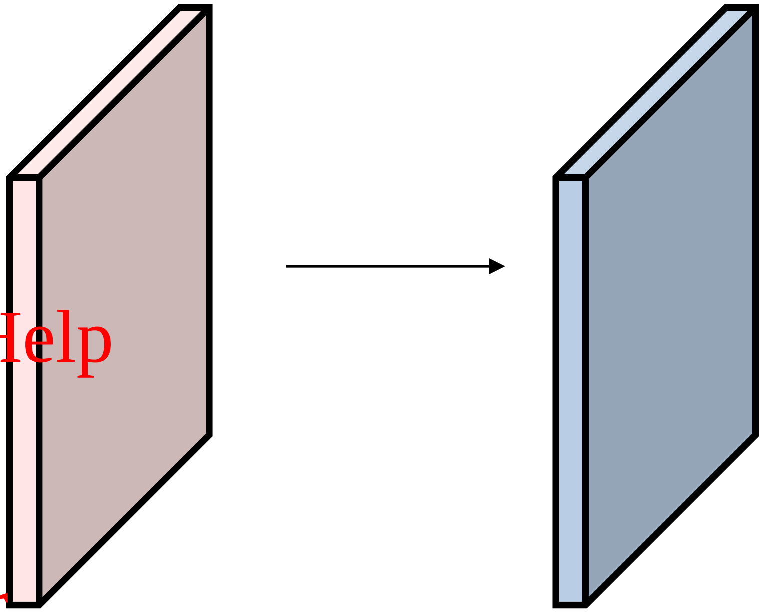


Output volume size:

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

32x32x10

Spatial Dimension at Convolution Layer



Input volume: 32x32x3

10 5x5x3 filters with stride 1, pad 2

Number of parameters in this layer?

each filter has 5*5*3 + 1 = 76 params (+1 for bias)

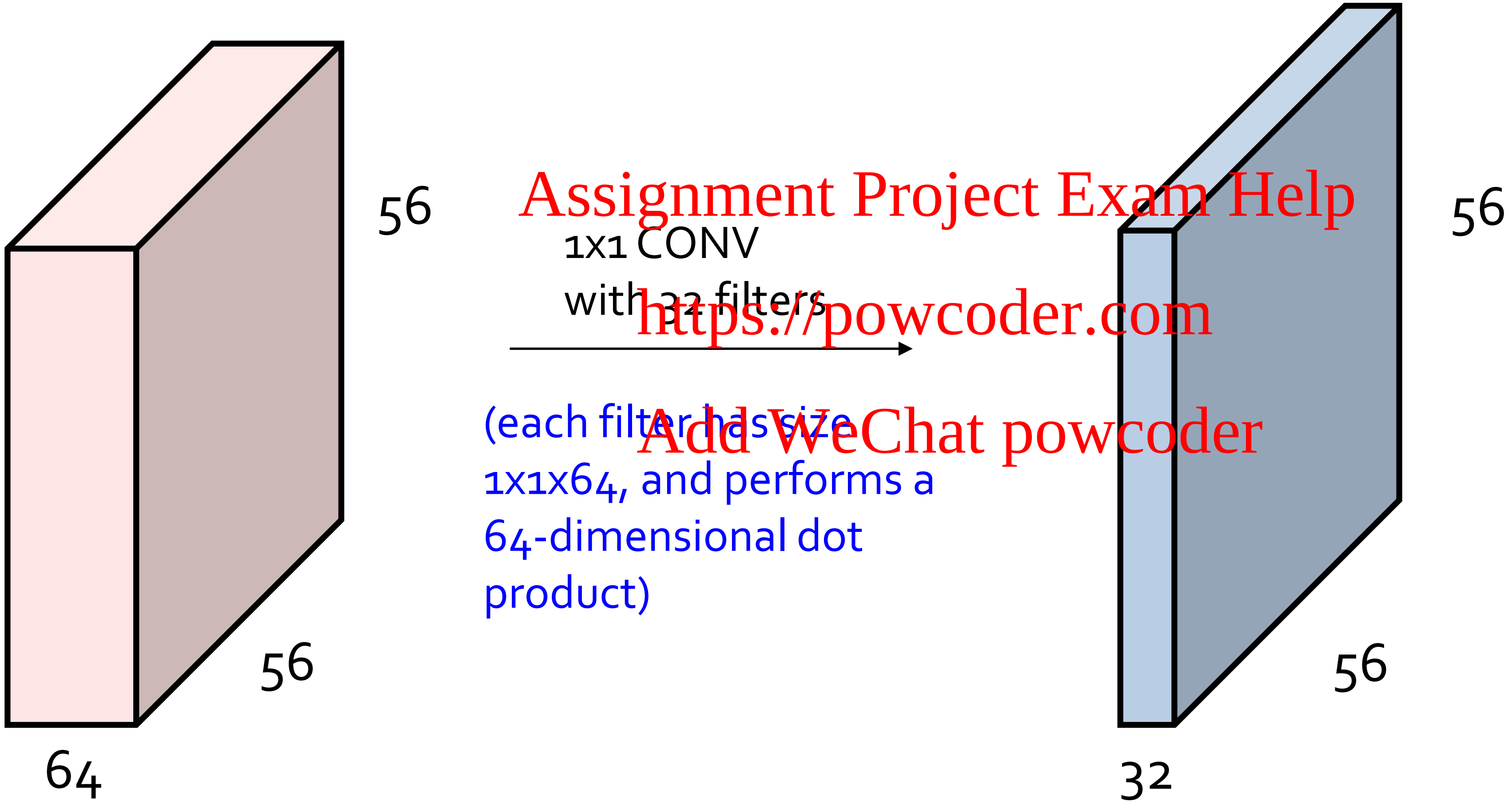
-> 76*10 = **760**

Spatial Dimension at Convolution 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.

1x1 Convolution

1x1 convolution layers make perfect sense



Convolutional Layer

Implementation and Backpropagation

Implementation as Matrix Multiplication

- Convolution: dot products between the filters and local regions of the input
- Conv layer: the forward pass of a convolutional layer as one big matrix multiply

Input: [227x227x3]
Conv filter: 96 filters of [11x11x3]
Conv bias: 96x1 vector
Stride: 4
Padding: 0

Output: (227-11)/4+1 = 55
-> [55x55x96]

- Example of feed-forward process

1. Convert the input into X_col by taking a block of 11x11x3 (=363) pixels in the input for 55x55 (=3025) times
   X_col: [363x3025]

2. Reshape the conv filter into W_row: [96x363]
   Reshape the conv bias (96x1 vector) into b_col: [96x3025] by stacking it for 3025 times

3. Perform matrix multiplication O = W_row * X_col + b_col

4. Reshape O: [96x3025] into [55x55x96]

$$\boxed{\underbrace{O}_{96 \times 3025} = \underbrace{W\_row}_{96 \times 363} * \underbrace{X\_col}_{363 \times 3025} + \underbrace{b\_col}_{96 \times 3025}}$$

Backpropagation of Convolution Layer

Input: [227x227x3]
Conv filter: 96 filters of [11x11x3]
Stride: 4
Padding: 0

Output: (227-11)/4+1 = 55
-> [55x55x96]

Backpropagation of $s = \mathrm{W}x + b$

$\boldsymbol{x}$ → **W** ($n \times d$) → $\boldsymbol{s}$

$d \times 1$ $\quad$ $n \times 1$

↑ $\boldsymbol{b}$ ($n \times 1$)

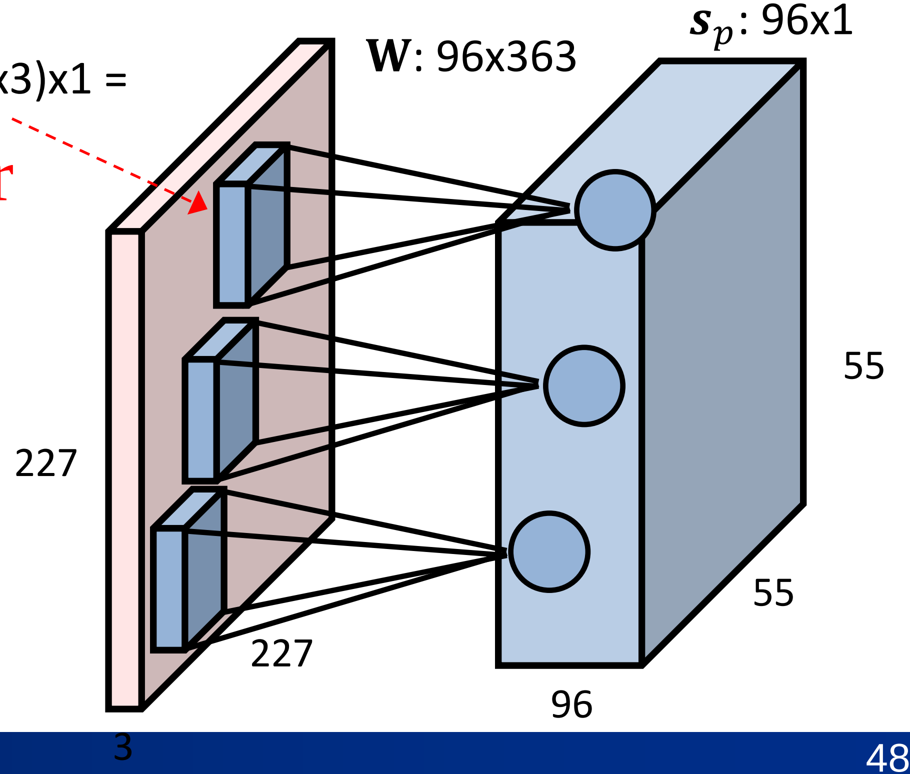
$$\frac{\partial L}{\partial \boldsymbol{x}} = \frac{\partial \boldsymbol{s}}{\partial \boldsymbol{x}} \frac{\partial L}{\partial \boldsymbol{s}} = \mathbf{W}^{\mathrm{T}} \frac{\partial L}{\partial \boldsymbol{s}}$$

$$\frac{\partial L}{\partial \mathbf{W}} = \frac{\partial L}{\partial \boldsymbol{s}} \boldsymbol{x}^{\mathrm{T}} \qquad \frac{\partial L}{\partial \boldsymbol{b}} = \frac{\partial L}{\partial \boldsymbol{s}}$$

Convolution layer shares **W** for all neurons of current activation map.
For each neuron,

$$\frac{\partial L}{\partial \boldsymbol{x}_p} = \mathbf{W}^{\mathrm{T}} \frac{\partial L}{\partial \boldsymbol{s}_p} \qquad \frac{\partial L}{\partial \mathbf{W}} = \sum_p \frac{\partial L}{\partial \boldsymbol{s}_p} \boldsymbol{x}_p^{\mathrm{T}} \qquad \frac{\partial L}{\partial \boldsymbol{b}} = \sum_p \frac{\partial L}{\partial \boldsymbol{s}_p}$$

$p = 1, \ldots, 3025 (= 55 \times 55)$



Backpropagation of Convolution Layer

Input: [227x227x3]
Conv filter: 96 filters of [11x11x3]
Stride: 4
Padding: 0

Output: (227-11)/4+1 = 55
-> [55x55x96]

Convolution layer shares $\mathbf{W}$ for all neurons of current activation map.
For all neurons,

$$\frac{\partial L}{\partial \boldsymbol{x}} = \mathbf{W}^{\mathrm{T}} \frac{\partial L}{\partial \boldsymbol{s}} \qquad \frac{\partial L}{\partial \mathbf{W}} = \frac{\partial L}{\partial \boldsymbol{s}} \boldsymbol{x}^{\mathrm{T}} \qquad \frac{\partial L}{\partial \boldsymbol{b}} = \frac{\partial L}{\partial \boldsymbol{s}} \mathbf{1}^{\mathrm{T}}$$

1x3025 vector

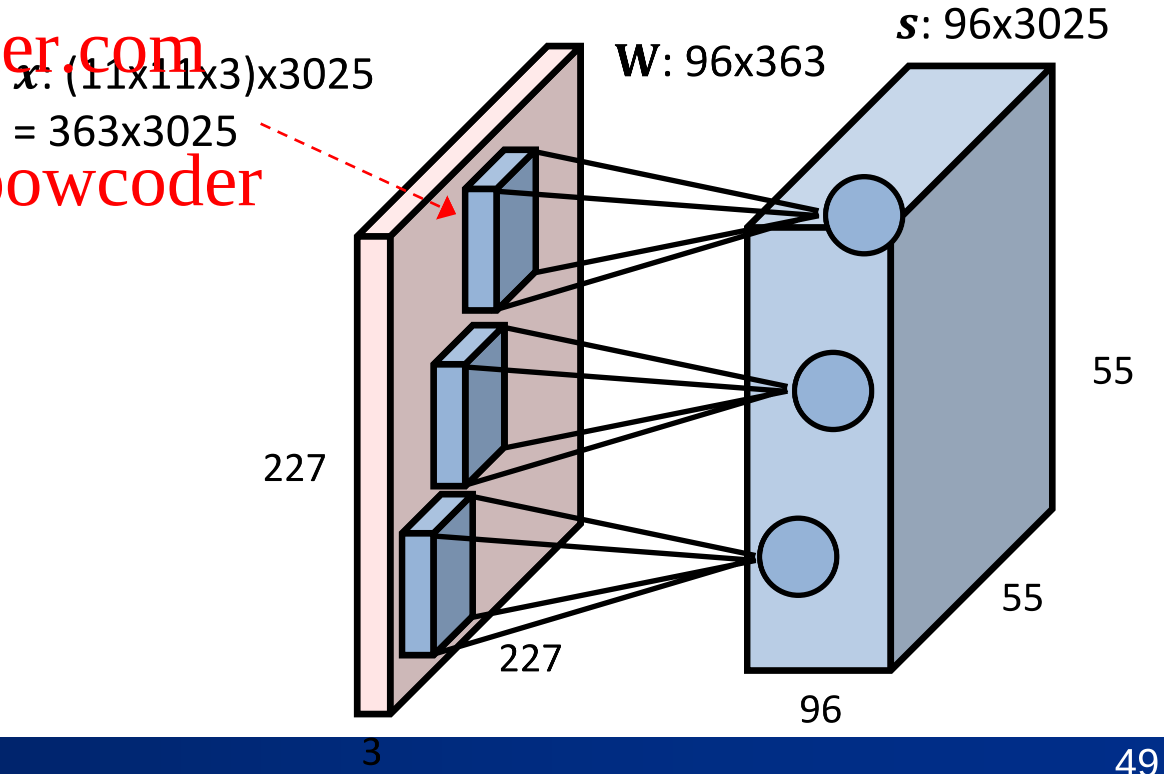
Backpropagation of $s = \mathrm{W}x + b$

$$\underset{d \times 1}{\boldsymbol{x}} \longrightarrow \boxed{\underset{n \times d}{\mathbf{W}}} \longrightarrow \underset{n \times 1}{\boldsymbol{s}}$$

$$\underset{n \times 1}{\boldsymbol{b}} \uparrow$$

$$\frac{\partial L}{\partial \boldsymbol{x}} = \frac{\partial \boldsymbol{s}}{\partial \boldsymbol{x}} \frac{\partial L}{\partial \boldsymbol{s}} = \mathbf{W}^{\mathrm{T}} \frac{\partial L}{\partial \boldsymbol{s}}$$

$$\frac{\partial L}{\partial \mathbf{W}} = \frac{\partial L}{\partial \boldsymbol{s}} \boldsymbol{x}^{\mathrm{T}} \qquad \frac{\partial L}{\partial \boldsymbol{b}} = \frac{\partial L}{\partial \boldsymbol{s}}$$

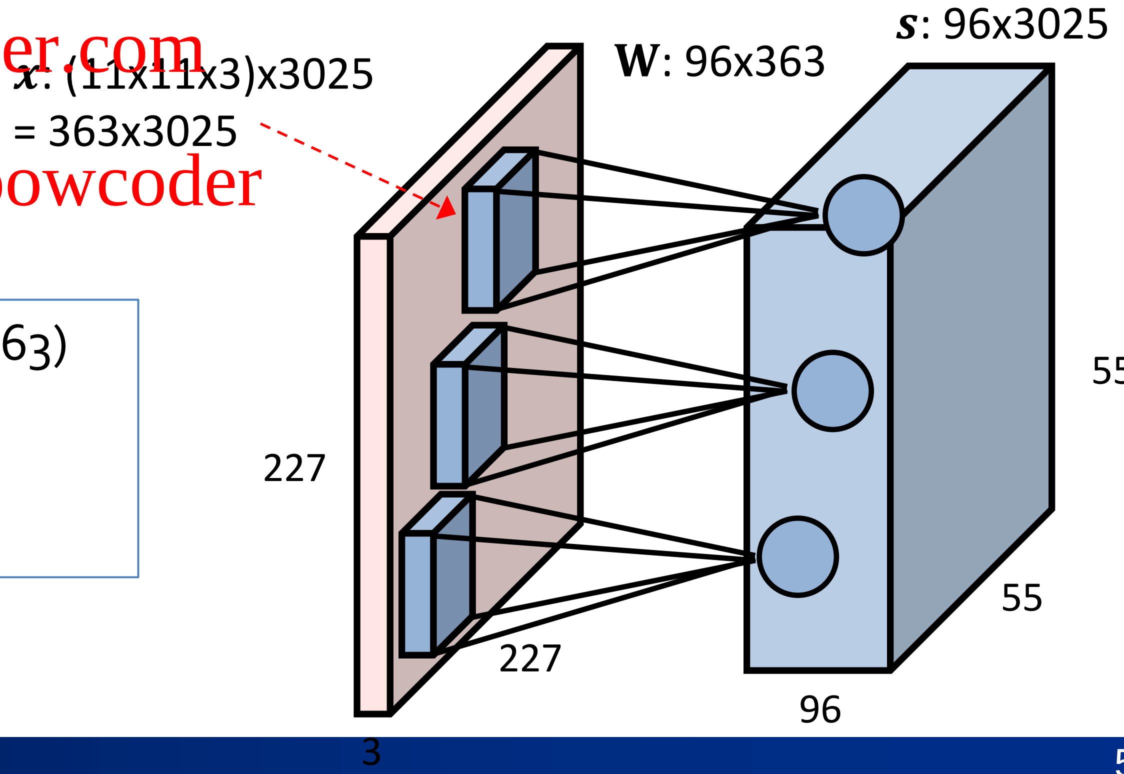


Backpropagation of Convolution Layer

$$\frac{\partial L}{\partial \boldsymbol{x}} = \mathbf{W}^{\mathrm{T}} \frac{\partial L}{\partial \boldsymbol{s}}$$

1. Perform $\frac{\partial L}{\partial \boldsymbol{x}} = \mathbf{W}^{\mathrm{T}} \frac{\partial L}{\partial \boldsymbol{s}}$
2. Reshape $\frac{\partial L}{\partial \boldsymbol{x}}$ (363x3025) into 3025 gradients of 11x11x3
3. Overlay the reshaped gradient into 3D matrix [227x227x3] in which overlapped gradients are accumulated.

$$\frac{\partial L}{\partial \mathbf{W}} = \frac{\partial L}{\partial \boldsymbol{s}} \boldsymbol{x}^{\mathrm{T}} \qquad \frac{\partial L}{\partial \boldsymbol{b}} = \frac{\partial L}{\partial \boldsymbol{s}} \mathbf{1}^{\mathrm{T}}$$



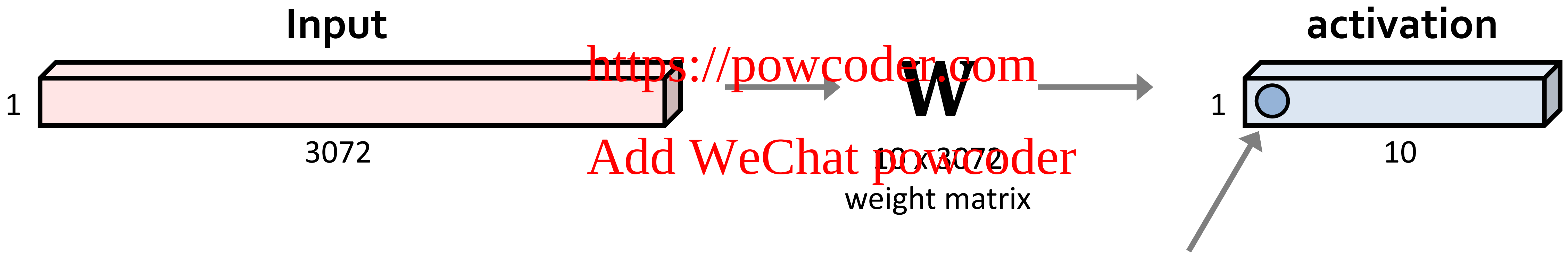
1. Convert the input into $\boldsymbol{x}$ by taking a block of 11x11x3 (=363) pixels in the input for 55x55 (=3025) times. $\boldsymbol{x}$: [363x3025]
2. Perform $\frac{\partial L}{\partial \mathbf{W}} = \frac{\partial L}{\partial \boldsymbol{s}} \boldsymbol{x}^{\mathrm{T}}$ and $\frac{\partial L}{\partial \boldsymbol{b}} = \frac{\partial L}{\partial \boldsymbol{s}} \mathbf{1}^{\mathrm{T}}$

Fully Connected Layer

Fully Connected Layer

Each neuron looks at the full input volume

32x32x3 image -> stretch to 3072 x 1



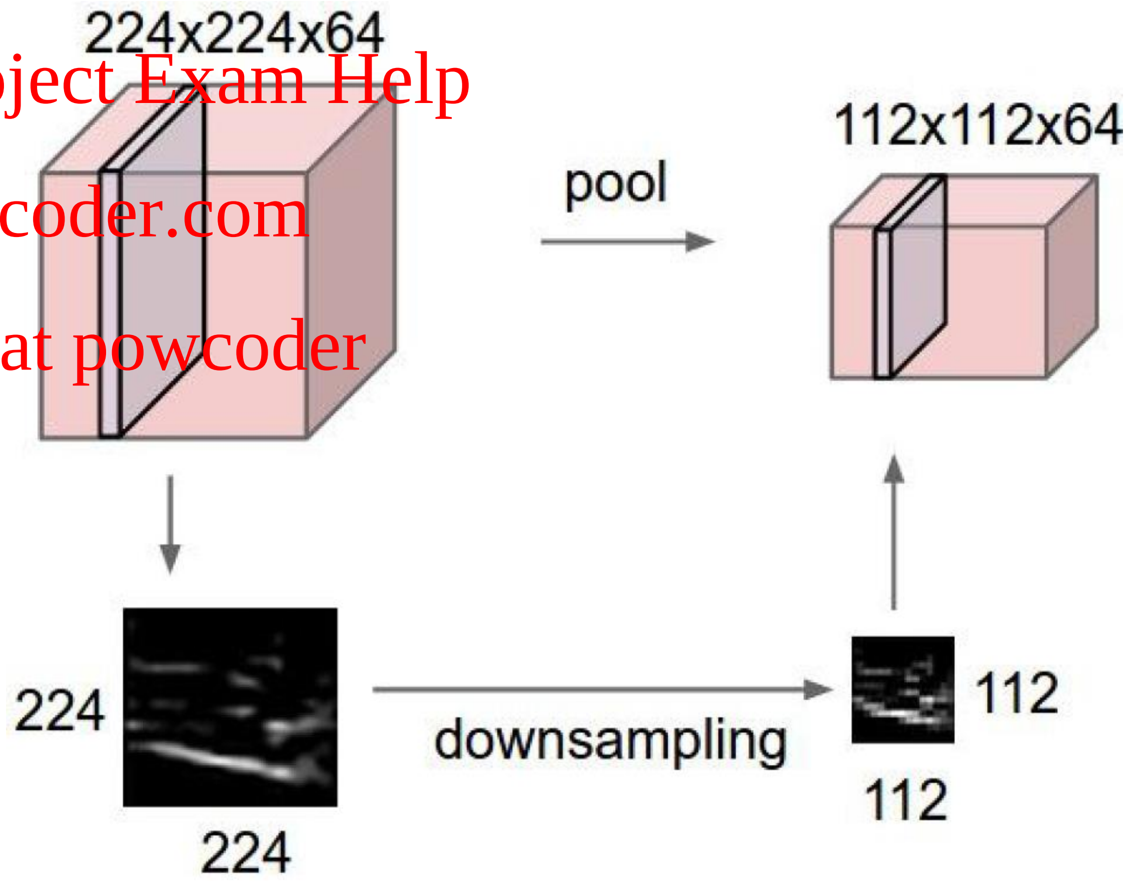
1 number:
the result of taking a dot product between a row of W and the input (a 3072-dimensional dot product)

Pooling Layer

Pooling layer

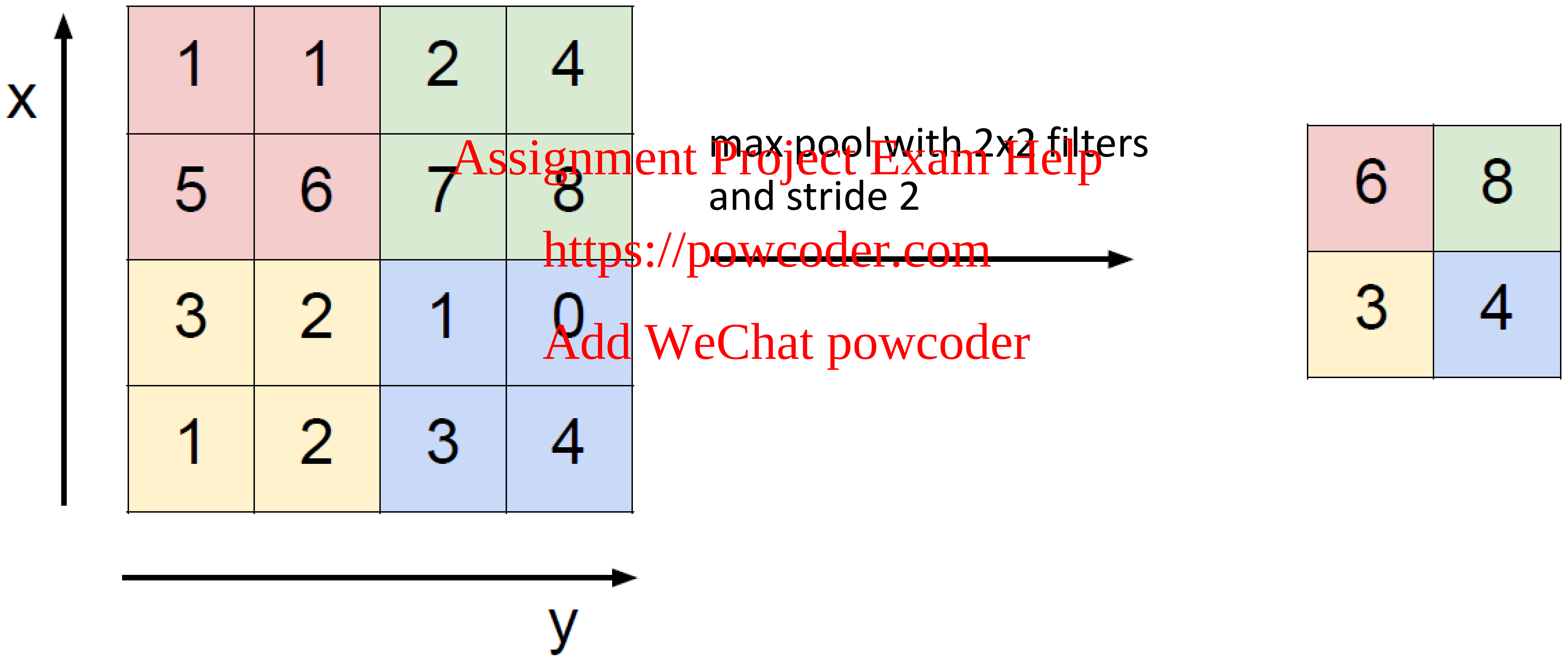
- makes the representations smaller and more manageable
- operates over each activation map independently

Pooling layer
- Max pooling
- Average pooling (rarely used)
- L2 norm pooling (rarely used)



MAX POOLING

Single depth slice



EBU7240

Computer Vision

- CNN Architectures -

Semester 1, 2021

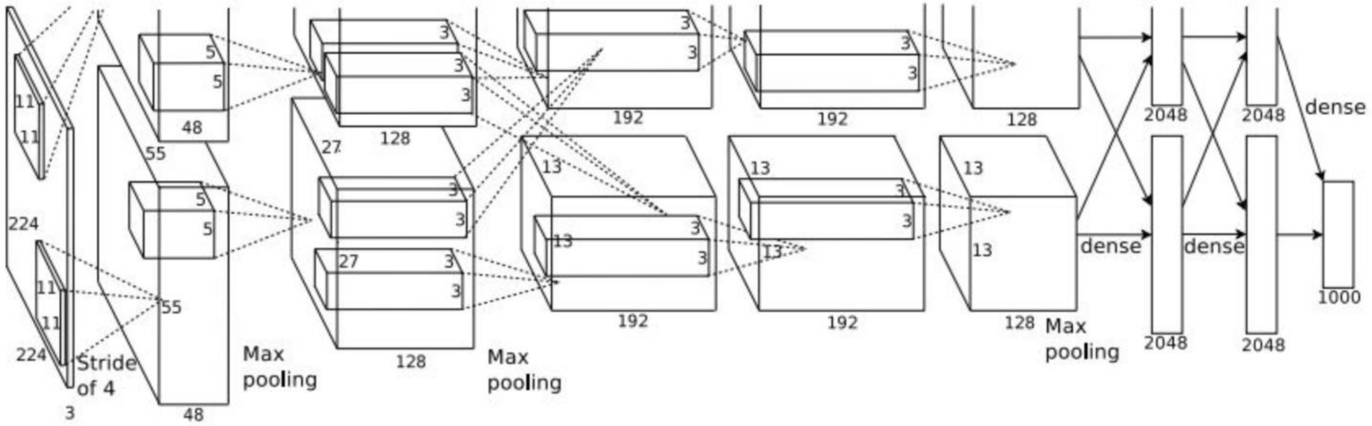
Changjae Oh

CNN Architectures

- **Case Studies**
  - AlexNet
  - VGG
  - GoogLeNet
  - ResNet

Case Study: AlexNet

[Krizhevsky et al. 2012]

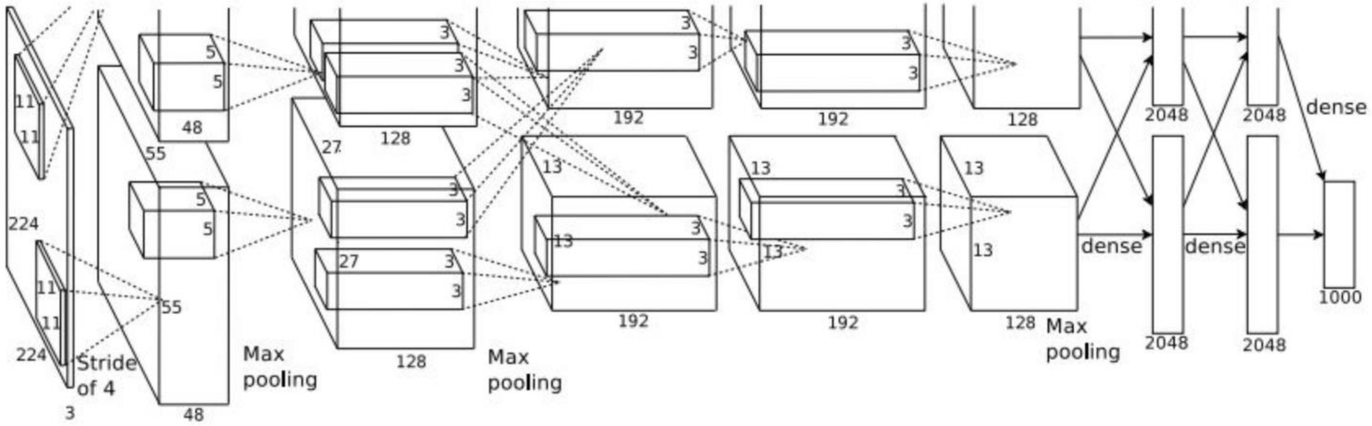


Architecture:

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

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

First layer (CONV1): 96 11x11 filters applied at stride 4

=>

The output volume size: (227-11)/4+1 = 55

Output volume [55x55x96]

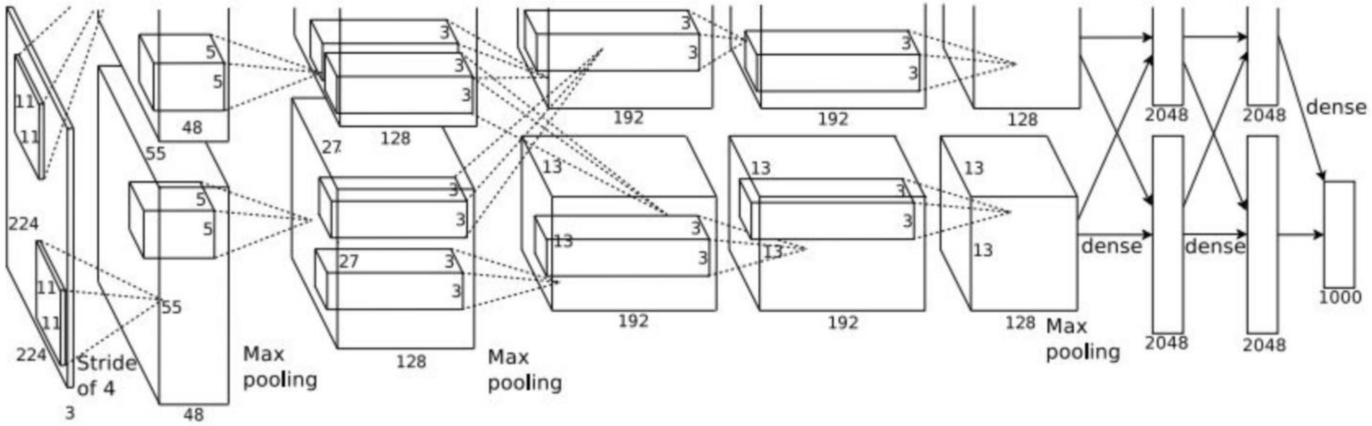
Total number of parameters in this layer

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

Bias: 96

Case Study: AlexNet

[Krizhevsky et al. 2012]



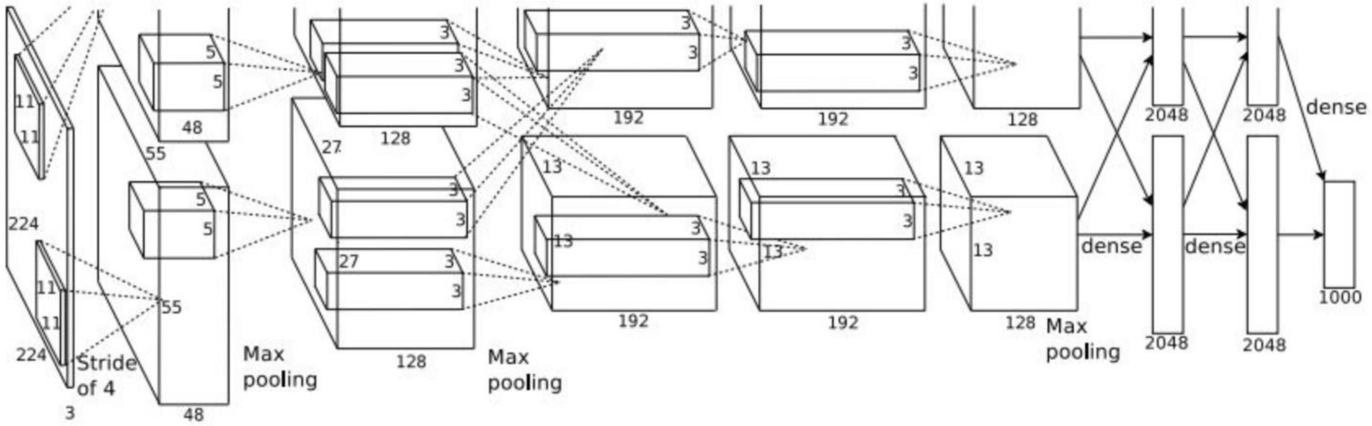
Input: 227x227x3 images
After CONV1: 55x55x96

Second layer (POOL1): 3x3 filters applied at stride 2
The output volume size: (55-3)/2+1 = 27
Output volume: 27x27x96

The number of parameters in this layer
Parameters: 0!

Case Study: AlexNet

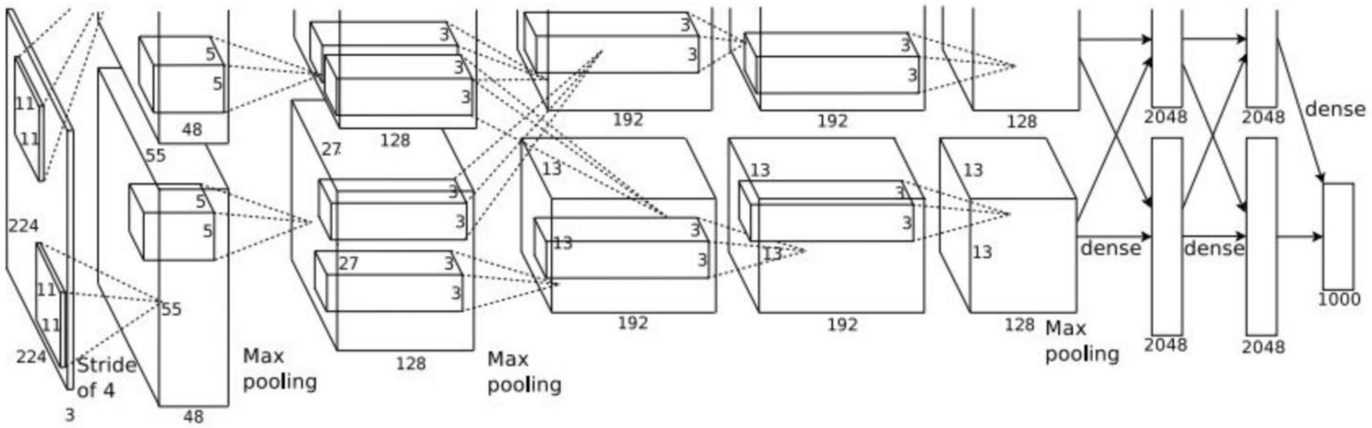
[Krizhevsky et al. 2012]



Input: 227x227x3 images
After CONV1: 55x55x96
After POOL1: 27x27x96
...

Case Study: AlexNet

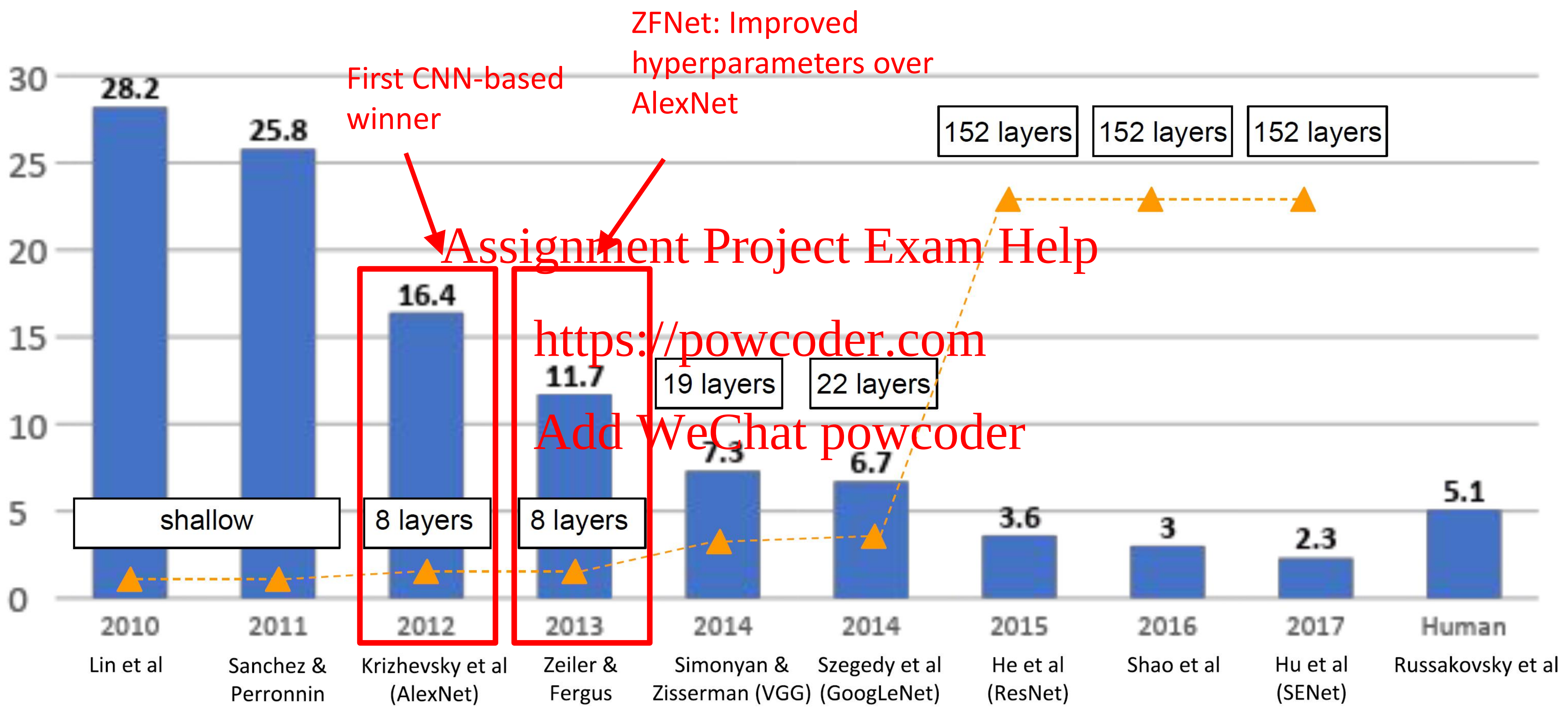
[Krizhevsky et al. 2012]



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%

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



Case Study: VGGNet

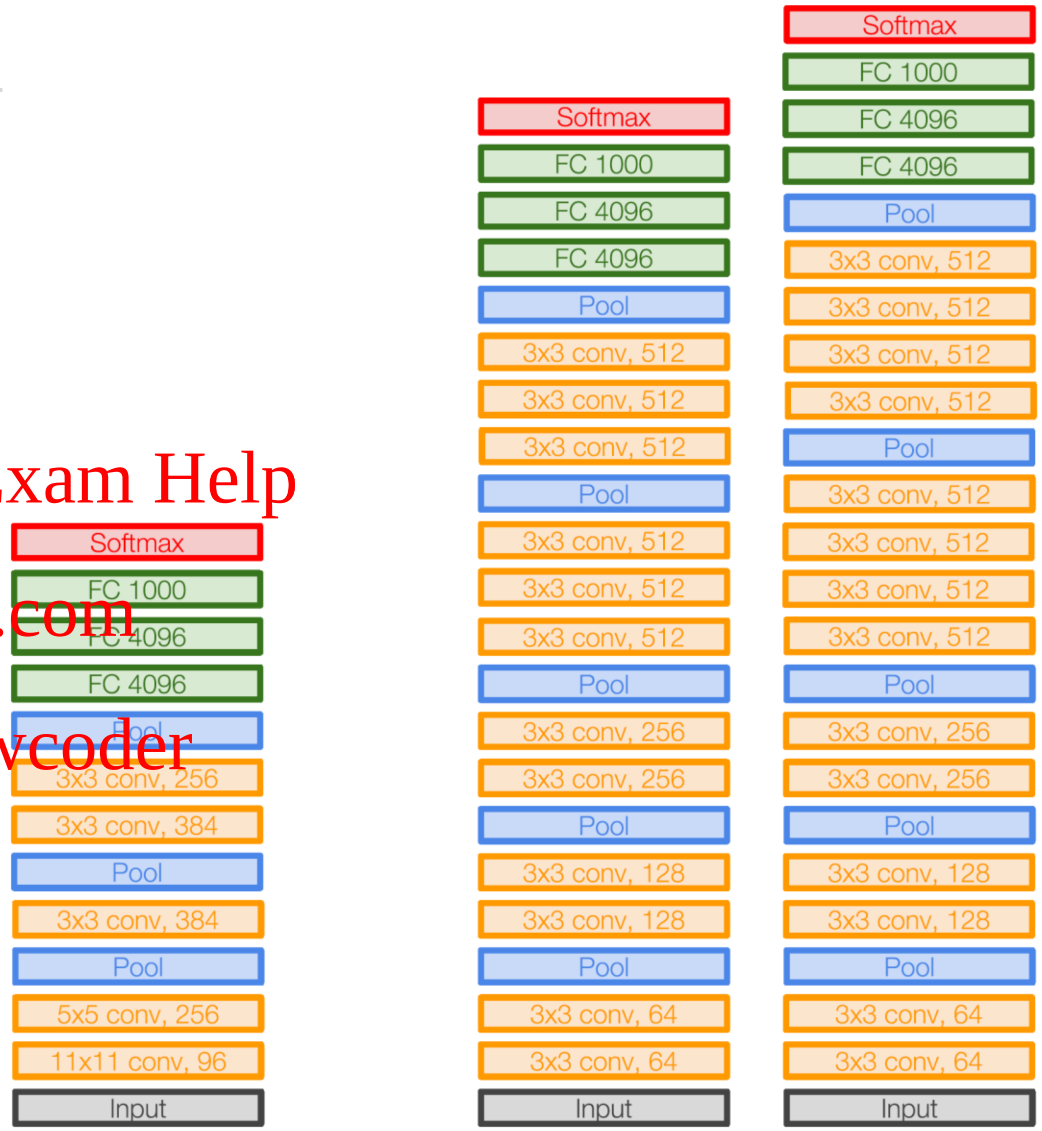
[Simonyan and Zisserman, 2014]

Small filters, Deeper networks

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

8 layers (AlexNet)
-> 16 - 19 layers (VGGNet)

11.7% top 5 error in ILSVRC’13
(ZFNet)
-> 7.3% top 5 error in ILSVRC’14



AlexNet VGG16 VGG19

Case Study: VGGNet

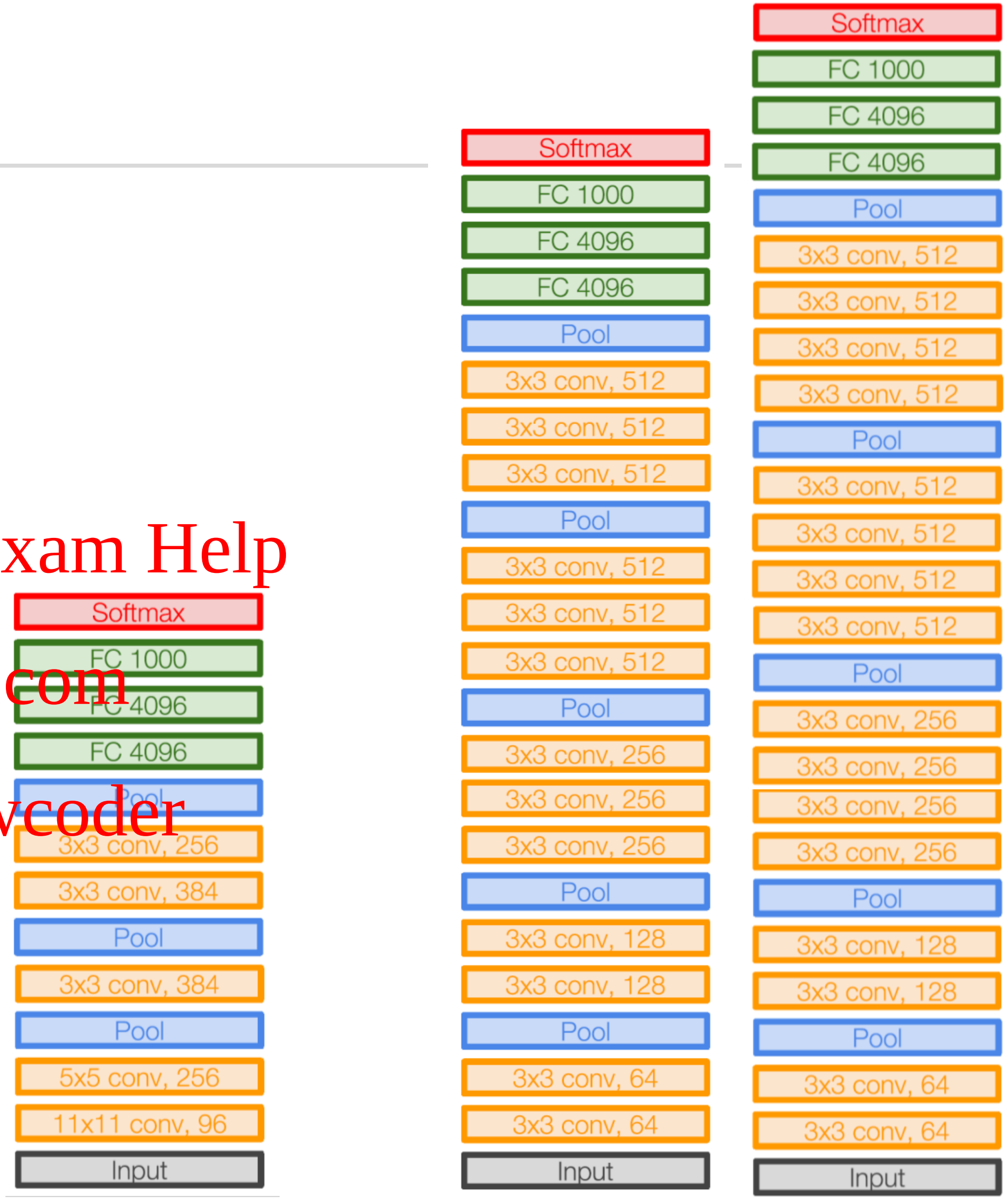
[Simonyan and Zisserman, 2014]

Q: Why use smaller filters? (3x3 conv)

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

But deeper, more non-linearity

And fewer parameters: 3 * (32C2) vs. 72C2 for C channels per layer



AlexNet VGG16 VGG19

(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

VGG16

TOTAL memory: 15.2M * 4 bytes ~= 61MB / image (for a forward pass)

TOTAL params: 138M parameters

(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

Most memory is in early CONV

Most params are in late FC

TOTAL memory: 15.2M * 4 bytes ~= 61MB / image (for a forward pass)

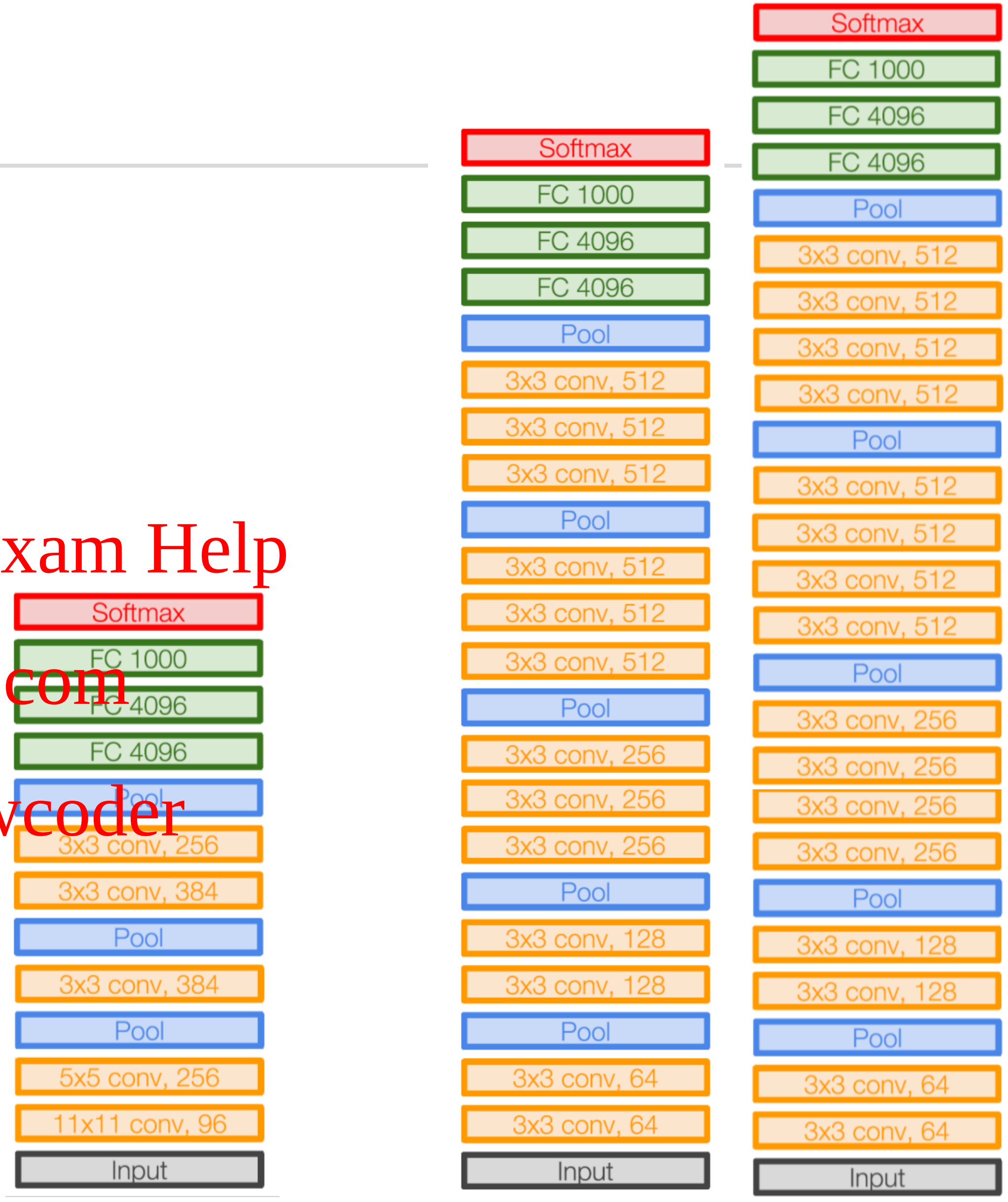
TOTAL params: 138M parameters

Case Study: VGGNet

[Simonyan and Zisserman, 2014]

Details:

- ILSVRC’14 2nd in classification, 1st in localization
- Similar training procedure as Krizhevsky 2012
- No Local Response Normalisation (LRN)
- Use VGG16 or VGG19 (VGG19 only slightly better, more memory)
- Use ensembles for best results
- FC7 features generalize well to other tasks



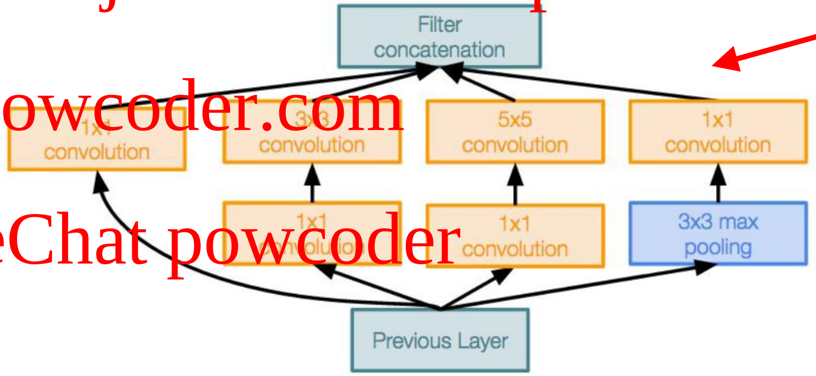
AlexNet VGG16 VGG19

Case Study: GoogLeNet

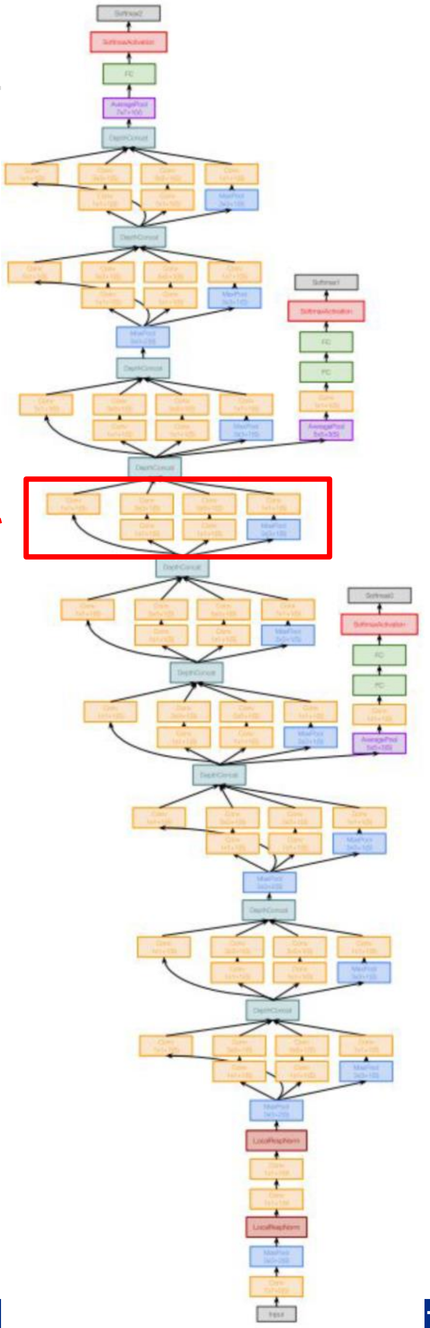
[Szegedy et al., 2014]

Deeper and wider networks,
with computational efficiency

- 22 layers
- Efficient “Inception” module
- No FC layers
- Only 5 million parameters!
  12x less than AlexNet
- ILSVRC’14 classification winner
  (6.7% top 5 error)



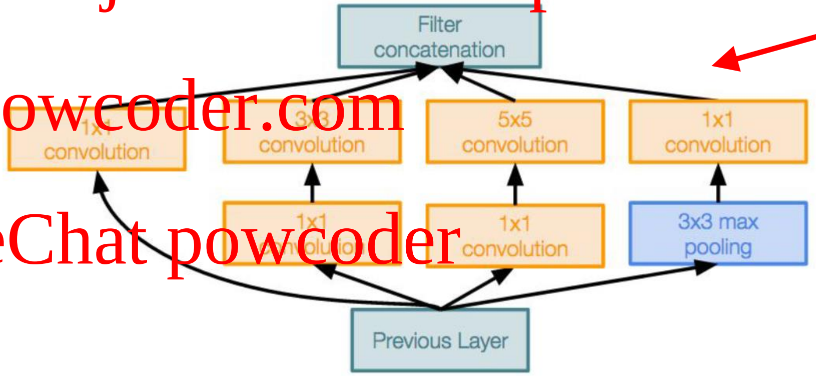
Inception module



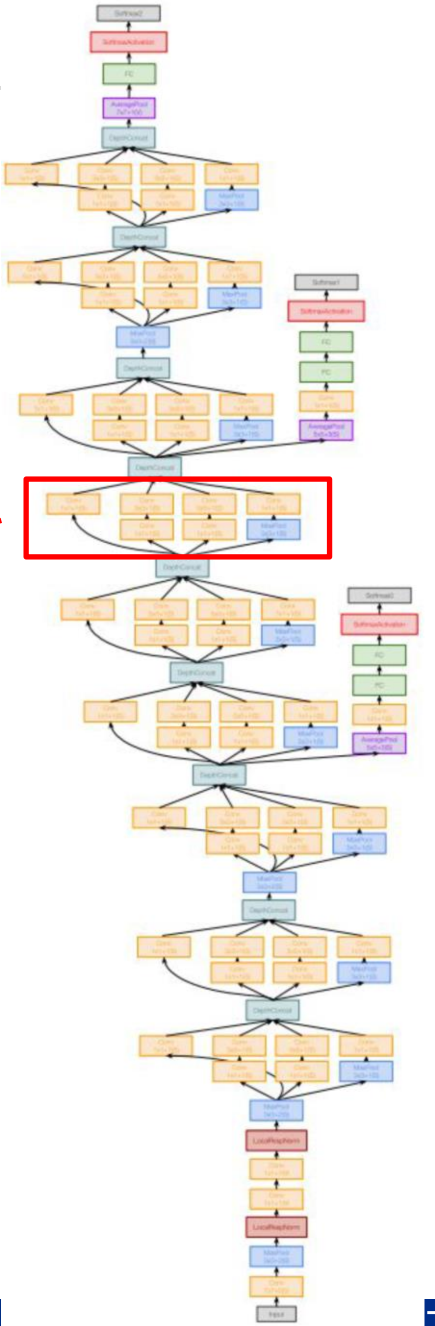
Case Study: GoogLeNet

[Szegedy et al., 2014]

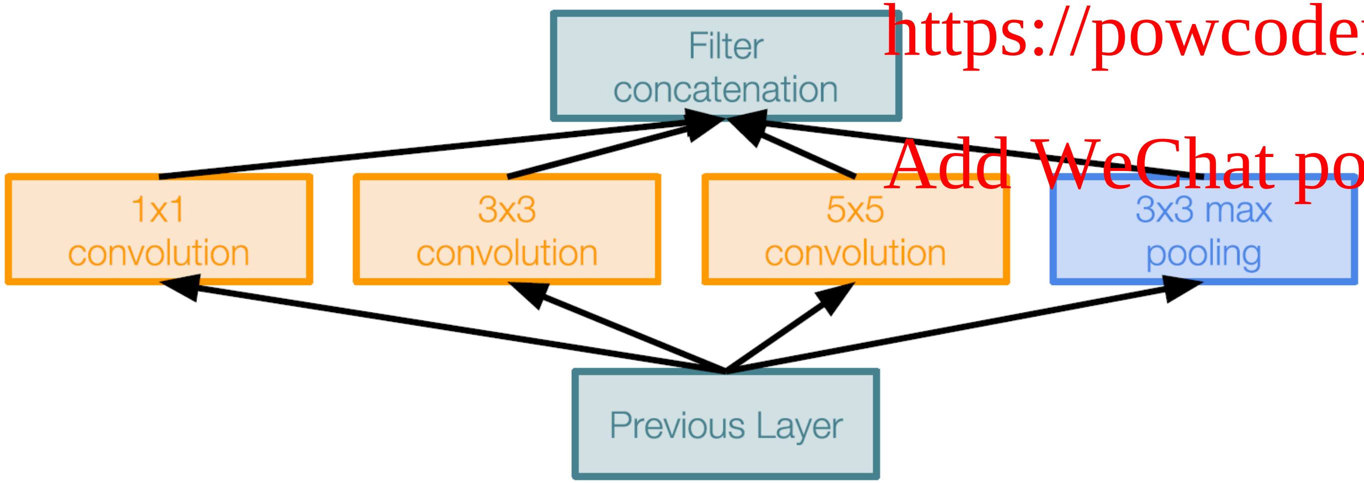
“Inception module”: design a good local network topology (network within a network) and then stack these modules on top of each other



Inception module



Case Study: GoogLeNet



Naive Inception module

Apply parallel filter operations on the input from previous layer:
- Multiple receptive field sizes for convolution (1x1, 3x3, 5x5)
- Pooling operation (3x3)

Concatenate all filter outputs together depth-wise

Q: What is the problem with this?
[Hint: Computational complexity]

Case Study: GoogLeNet

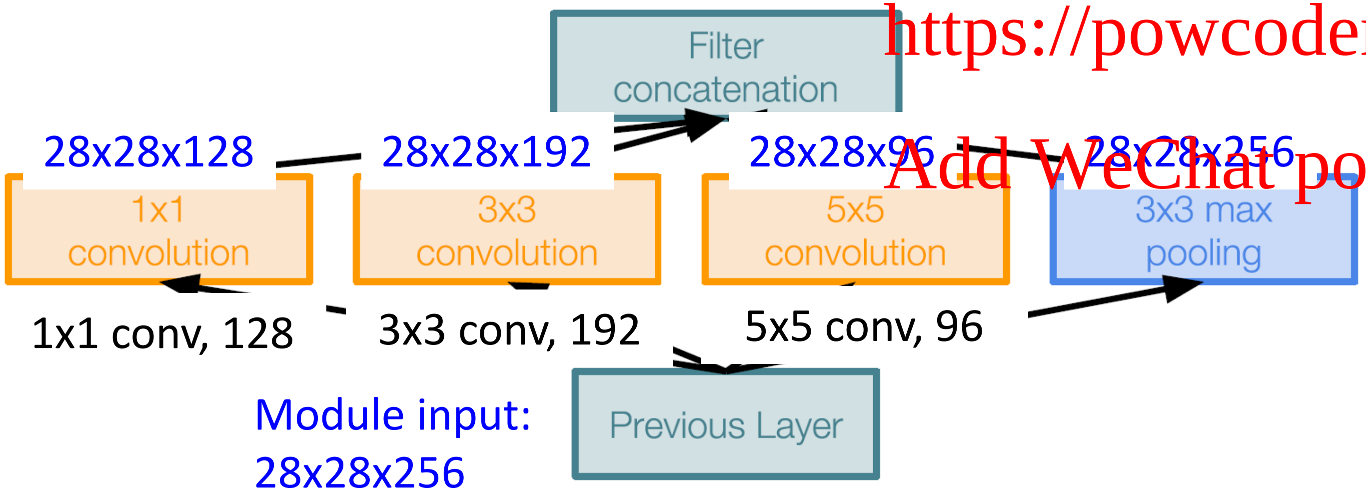
Example:

Q: What are the output size of filter concatenation?

(Assume that a proper size of zero-padding is used)

Q: What is the problem with this?
[Hint: Computational complexity]

28x28x(128+192+96+256) = 28x28x672



Module input: 28x28x256

Naive Inception module

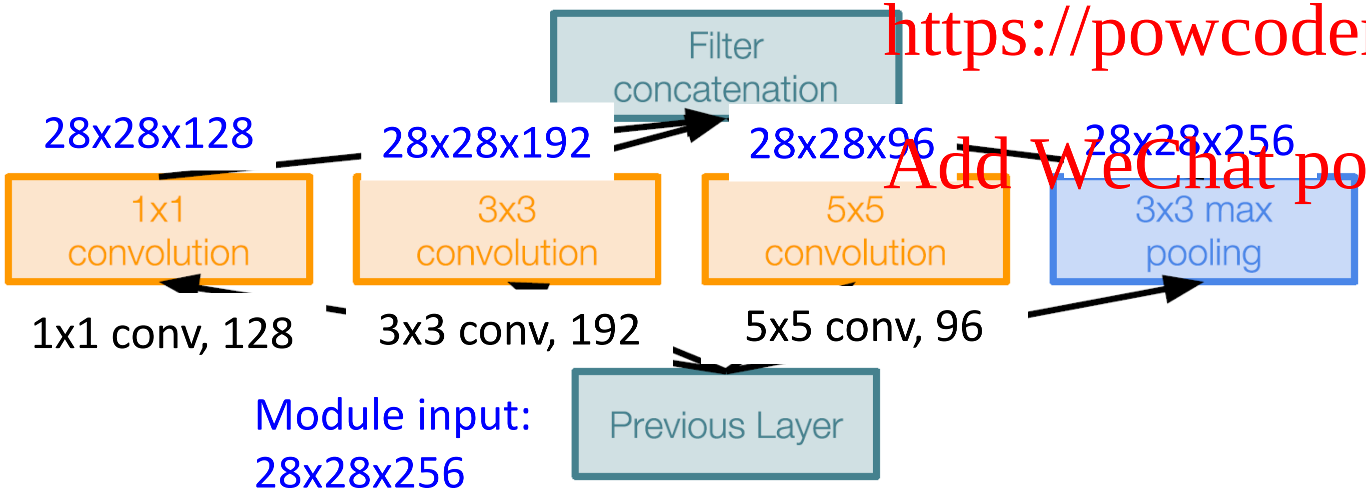
Case Study: GoogLeNet

Example:

Q: What are the output size of filter concatenation?
(Assume that a proper size of zero-padding is used)

Q: What is the problem with this?
[Hint: Computational complexity]

28x28x(128+192+96+256) = 28x28x**672**



Naive Inception module

Conv Ops:
[1x1 conv, 128] 28x28x128x1x1x256
[3x3 conv, 192] 28x28x192x3x3x256
[5x5 conv, 96] 28x28x96x5x5x256
Total: 854M ops

Very expensive compute

Pooling layer also preserves feature depth.
-> Total depth after concatenation can only grow at every layer!

Case Study: GoogLeNet

Example:

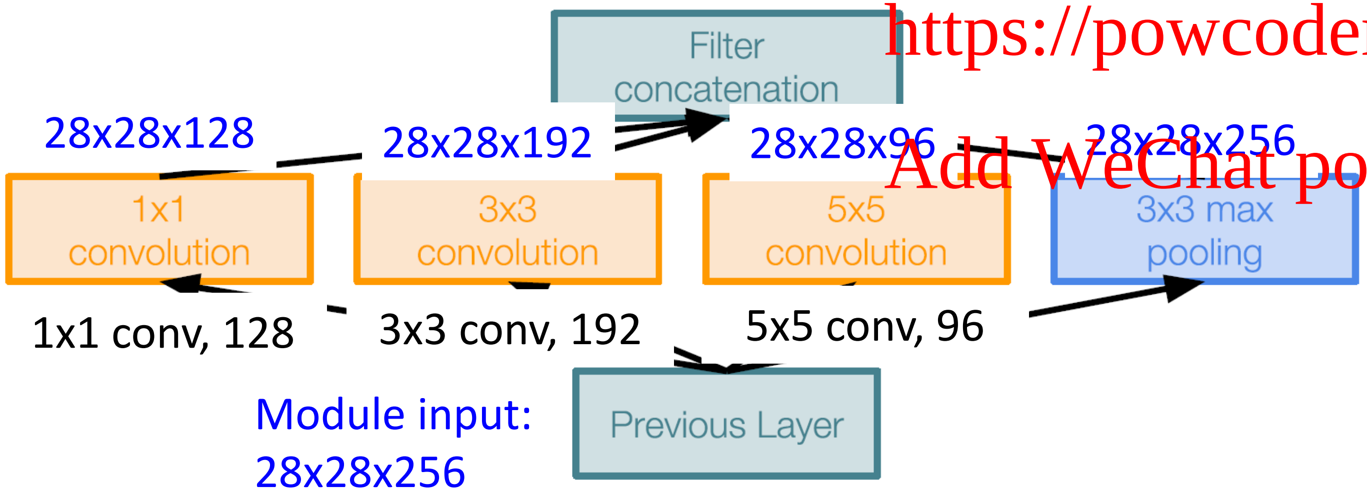
Q: What are the output size of filter concatenation?

(Assume that a proper size of zero-padding is used)

Q: What is the problem with this?
[Hint: Computational complexity]

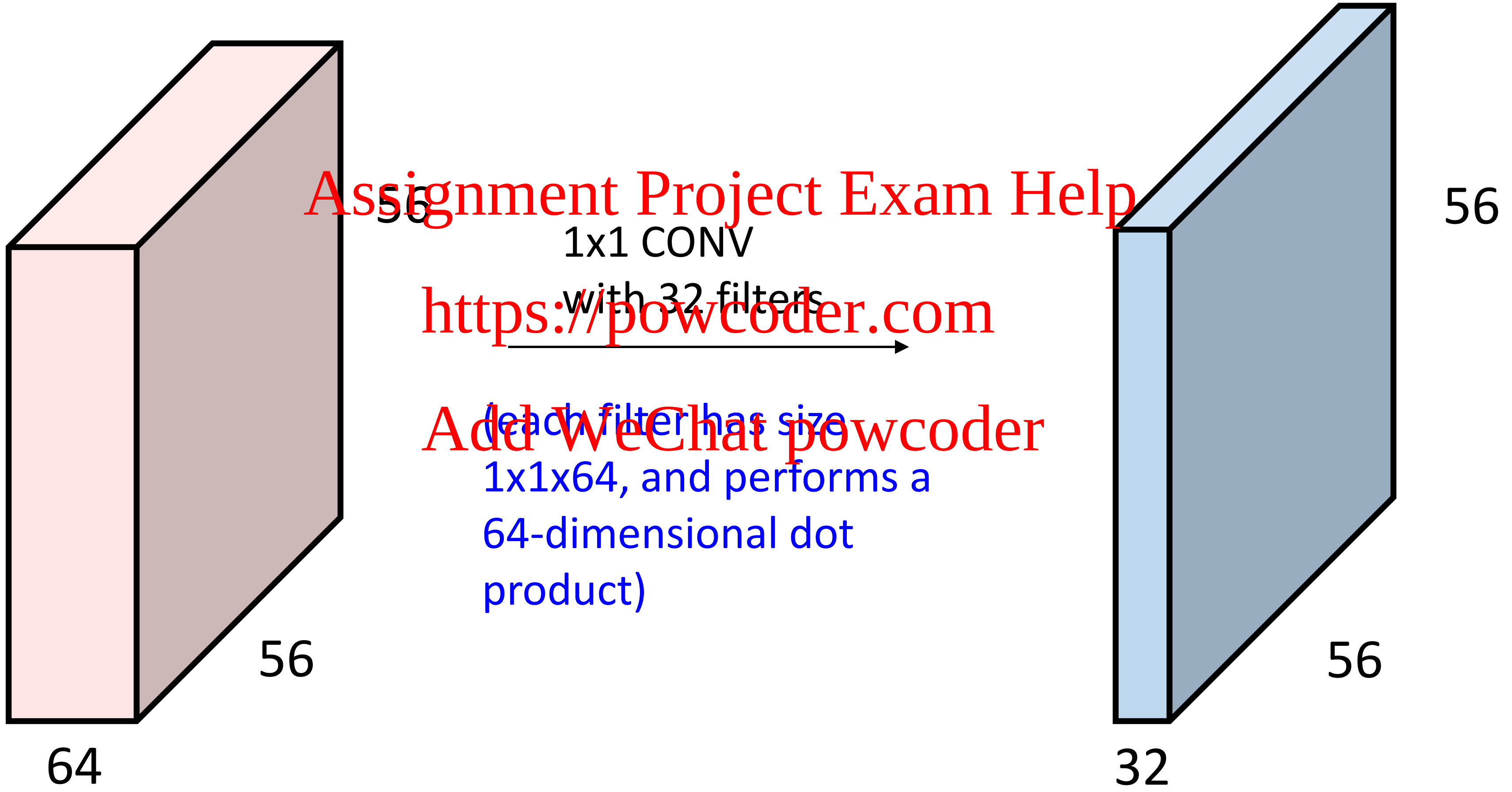
Solution:
“bottleneck” layers that use 1x1 convolutions to reduce feature depth

28x28x(128+192+96+256) = 28x28x**672**

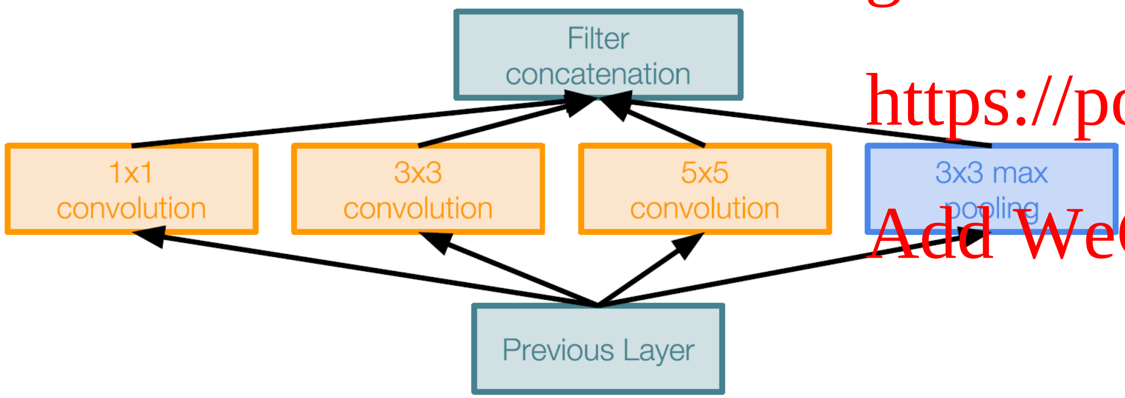


Naive Inception module

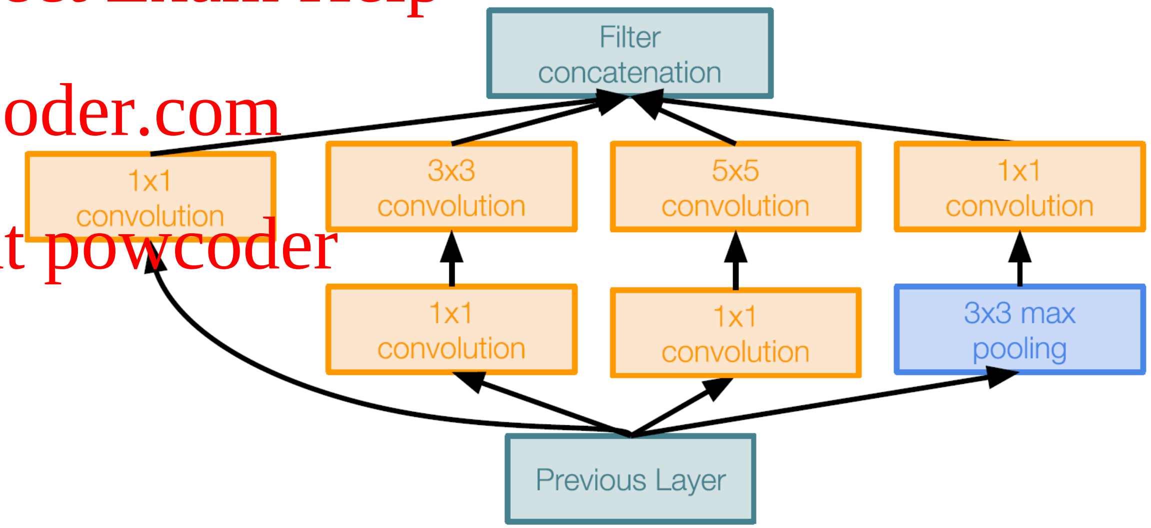
Reminder: 1x1 Convolution



Case Study: GoogLeNet



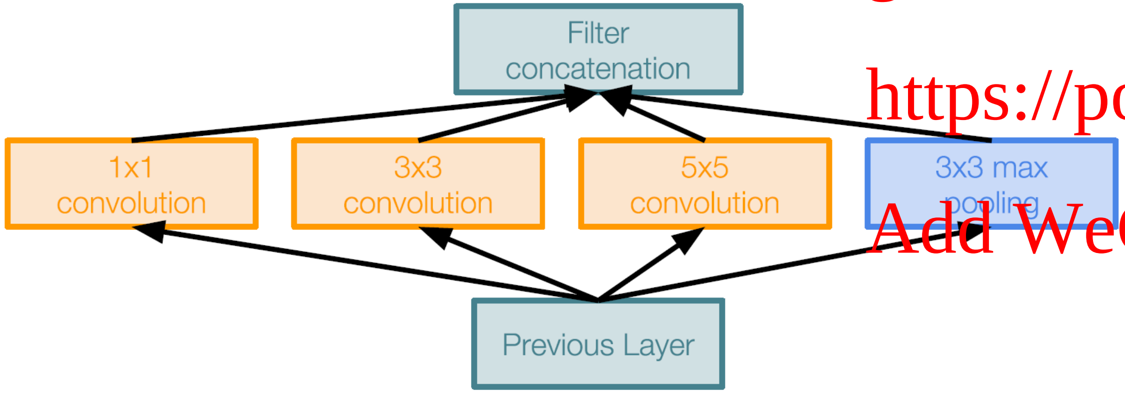
Naive Inception module



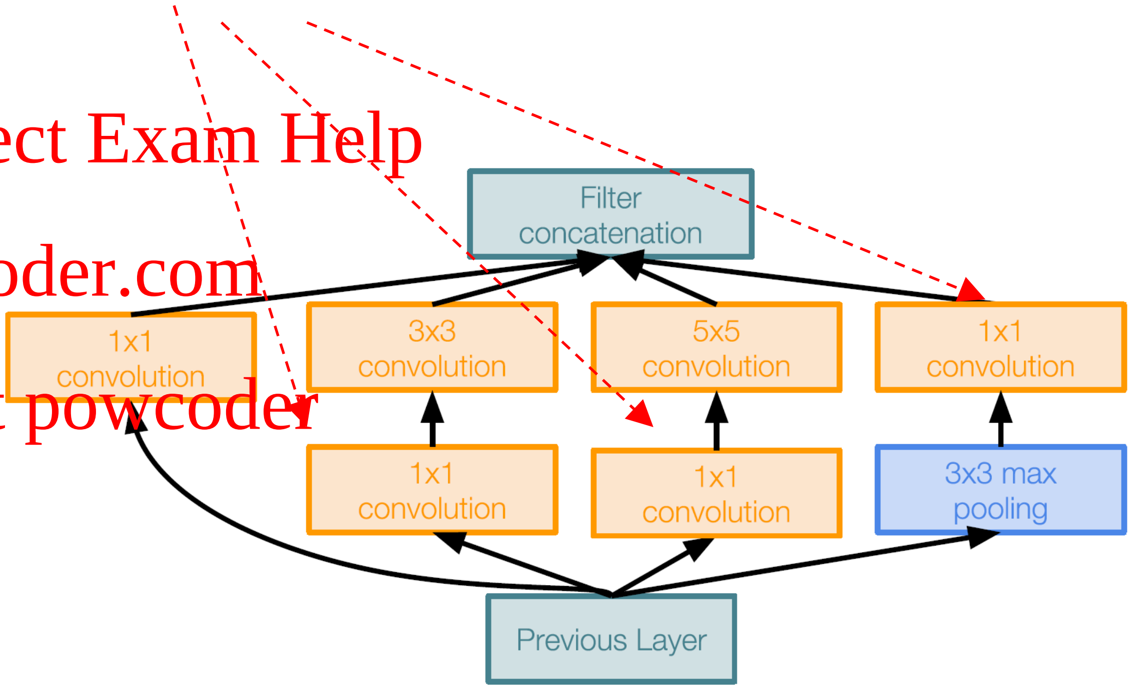
Inception module with dimension reduction

Case Study: GoogLeNet

1x1 conv “bottleneck” layers

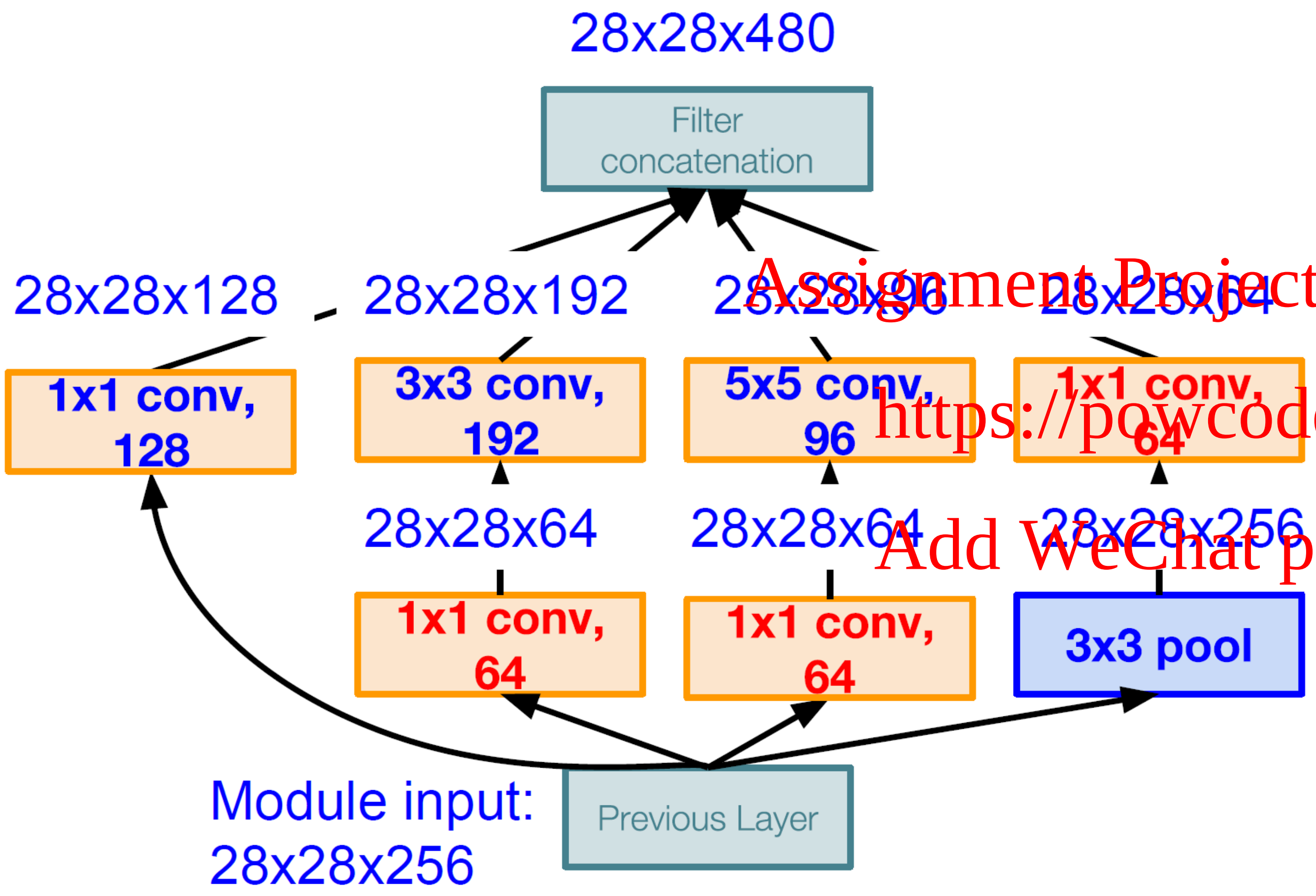


Naive Inception module



Inception module with dimension reduction

Case Study: GoogLeNet



Inception module with dimension reduction

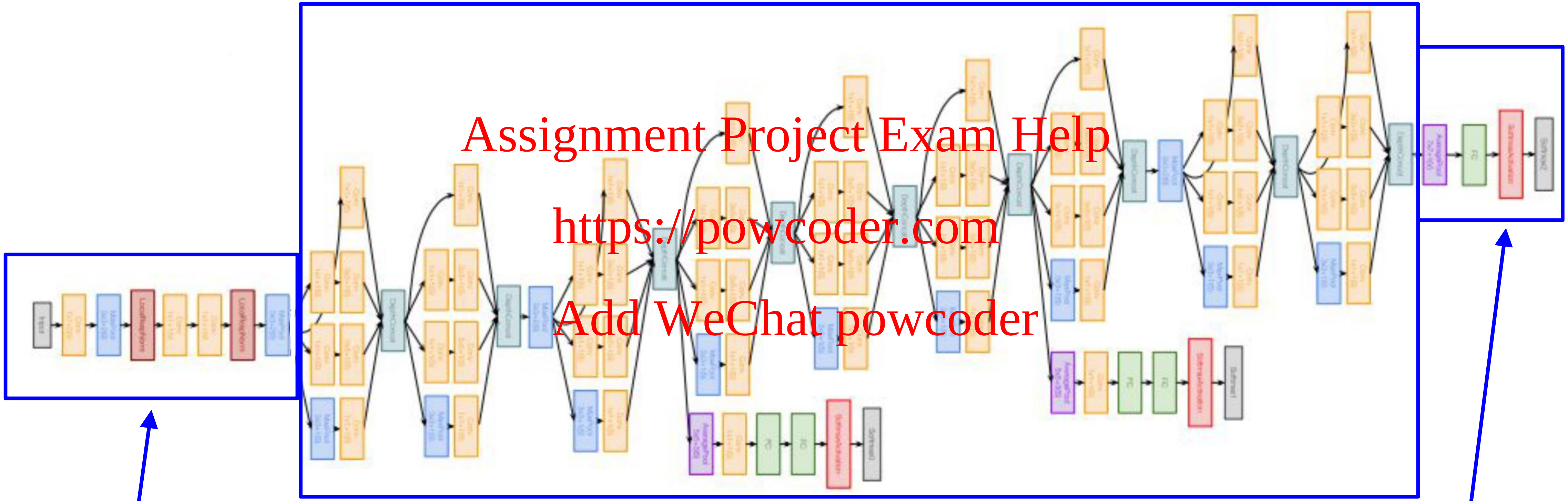
Using same parallel layers as naive example, and adding "1x1 conv, 64 filter" bottlenecks:

Conv Ops:
[1x1 conv, 64] 28x28x64x1x1x256
[1x1 conv, 64] 28x28x64x1x1x256
[1x1 conv, 128] 28x28x128x1x1x256
[3x3 conv, 192] 28x28x192x3x3x64
[5x5 conv, 96] 28x28x96x5x5x64
[1x1 conv, 64] 28x28x64x1x1x256
Total: 358M ops

Compared to 854M ops for naive version
Bottleneck can also reduce depth after pooling layer

Case Study: GoogLeNet

Full GoogLeNet architecture



Stem Network:
Conv-Pool-
2x Conv-Pool

Stacked Inception
Modules

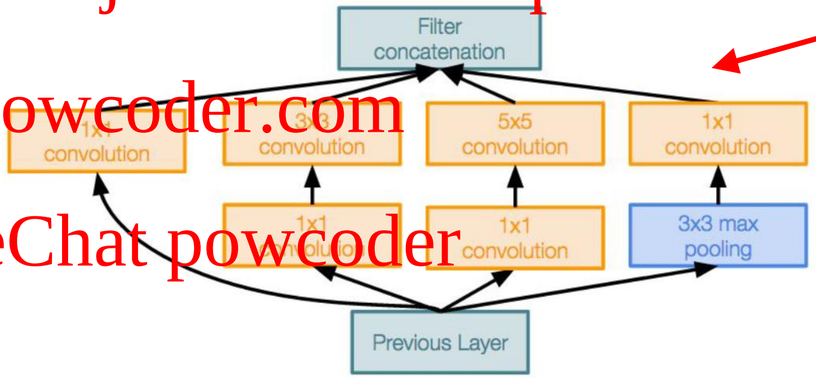
Classifier output
(removed expensive FC layers!)

Case Study: GoogLeNet

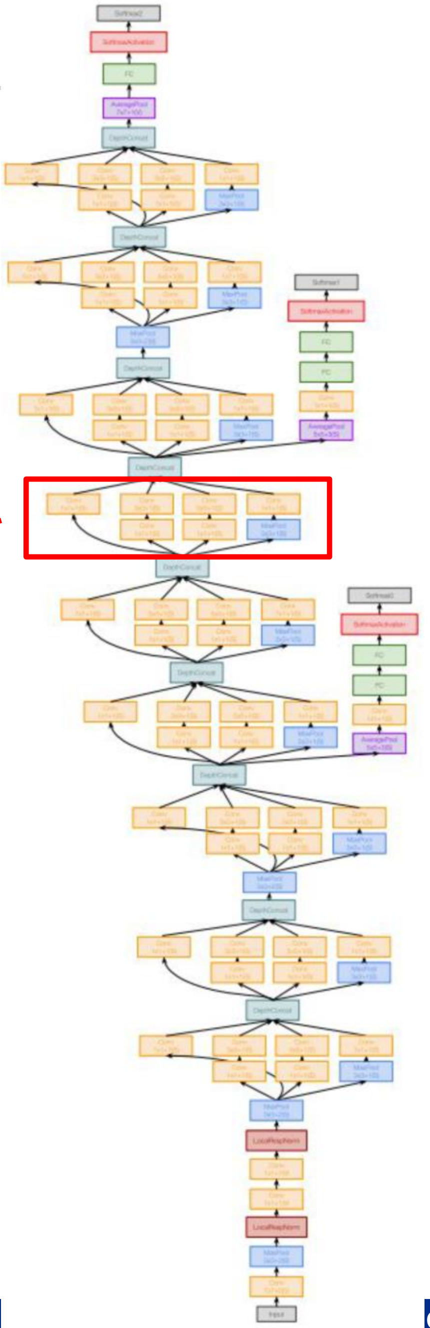
[Szegedy et al., 2014]

Deeper networks, with computational efficiency

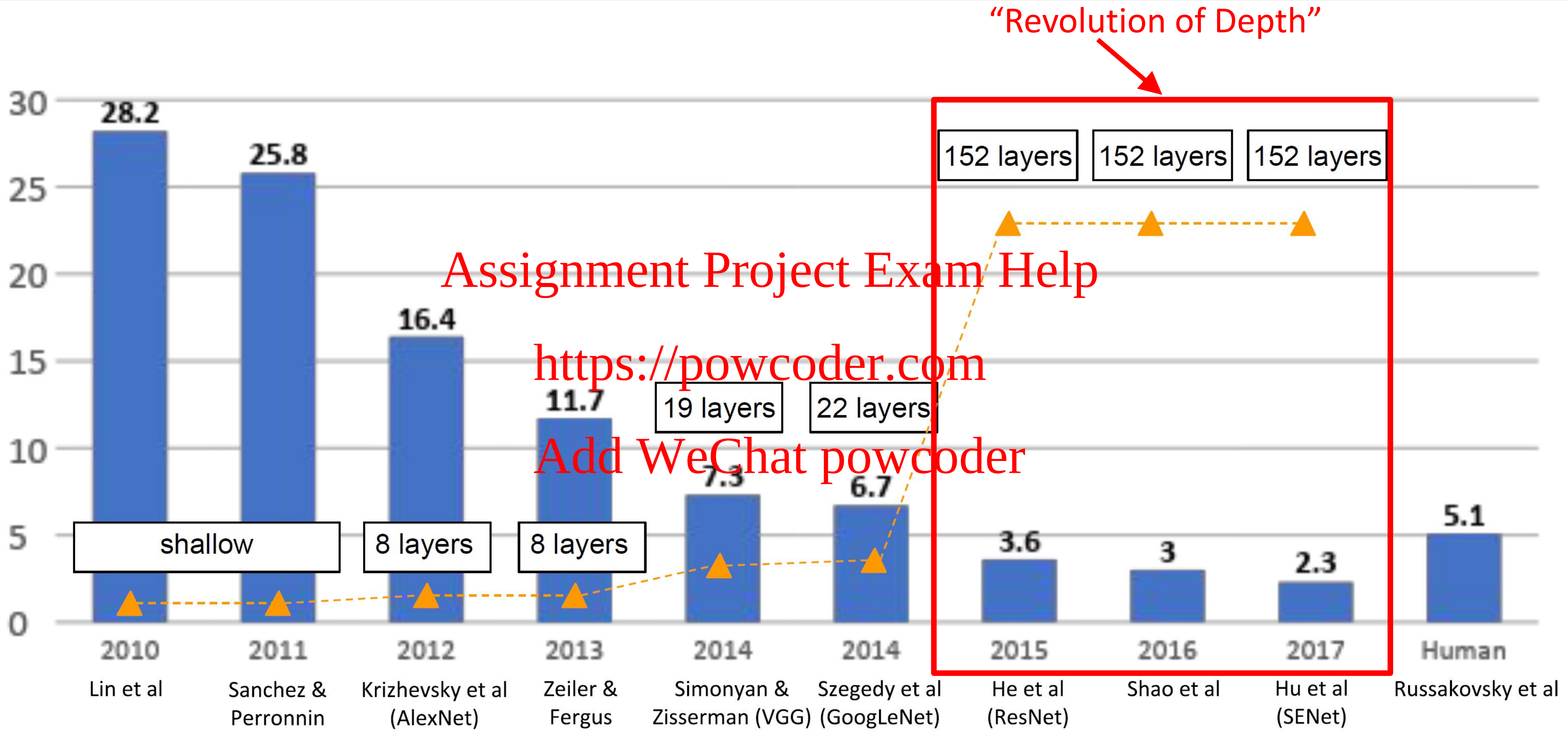
- 22 layers
- Efficient "Inception" module
- No FC layers
- Only 5 million parameters!
  12x less than AlexNet
- ILSVRC'14 classification winner
  (6.7% top 5 error)



Inception module



ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

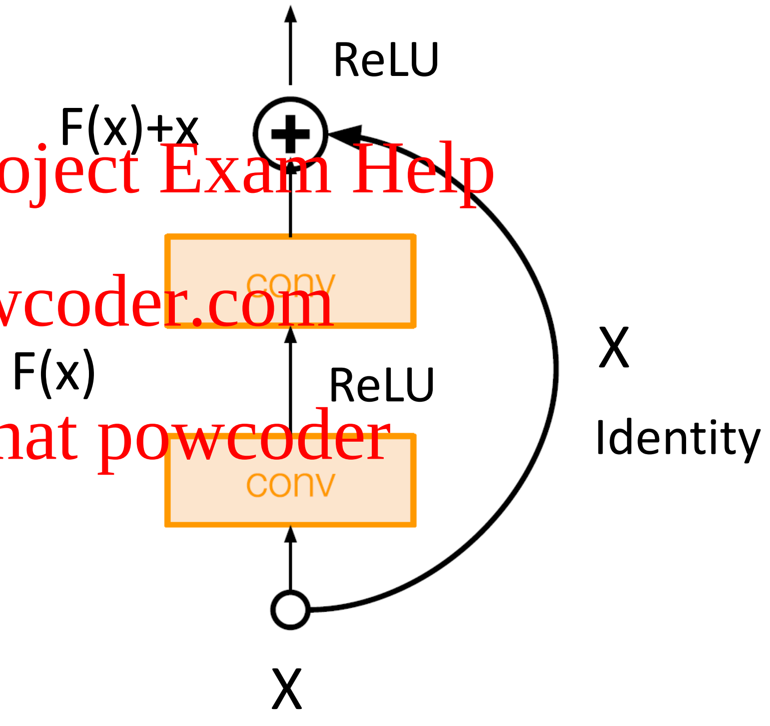


Case Study: ResNet

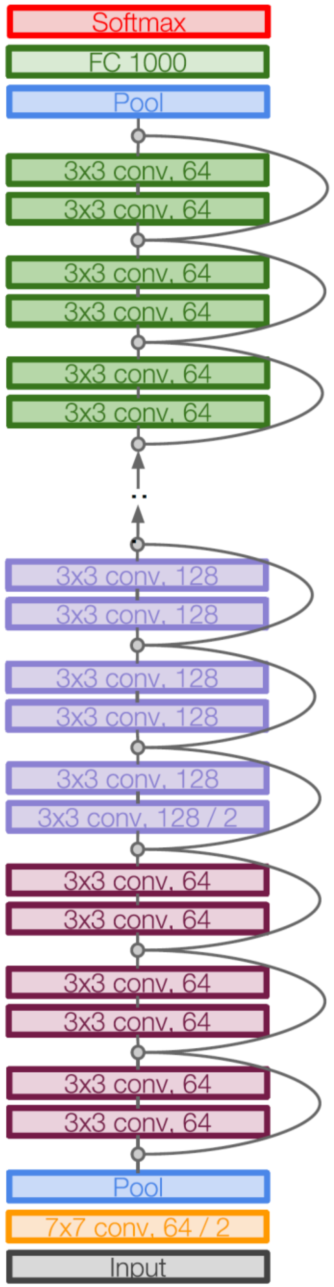
[He et al., 2015]

Very deep networks using residual connections

- 152-layer model for ImageNet
- ILSVRC'15 classification winner (3.57% top 5 error)
- Swept all classification and detection competitions in ILSVRC'15 and COCO'15!



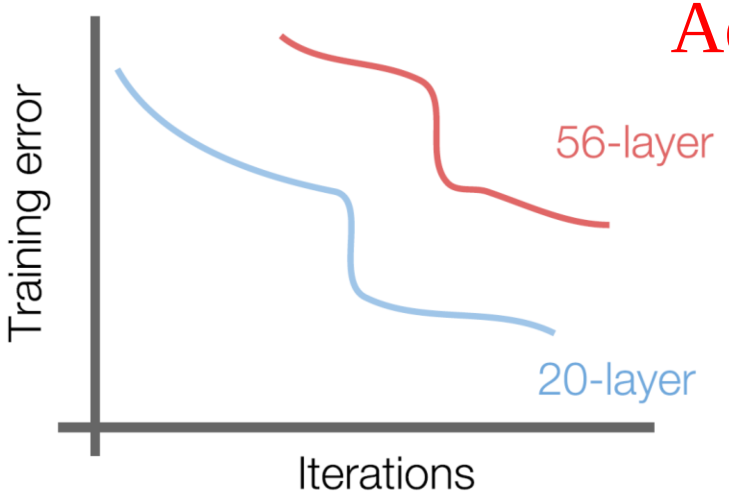
Residual block

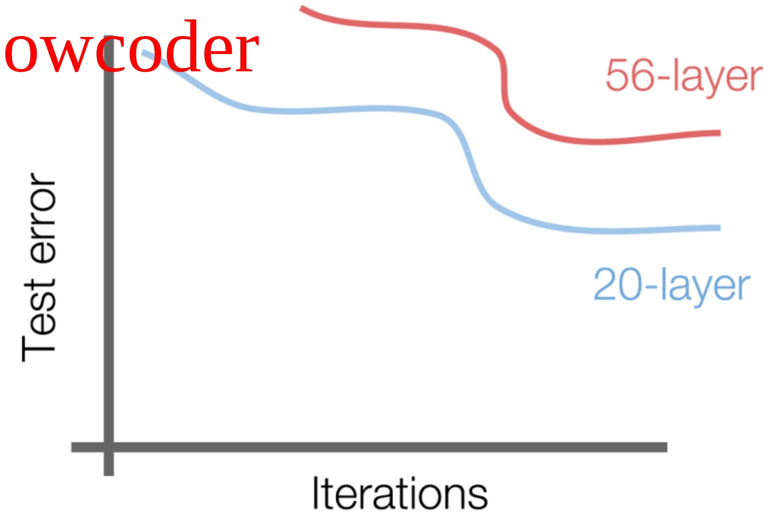


Case Study: ResNet

What happens when we continue stacking deeper layers on a “plain” convolutional neural network?

56-layer model performs worse on both training and test error
-> The deeper model performs worse, but it’s not caused by overfitting!





Case Study: ResNet

Hypothesis: the problem is an *optimization*, not *the model itself*,
-> deeper models are harder to optimize

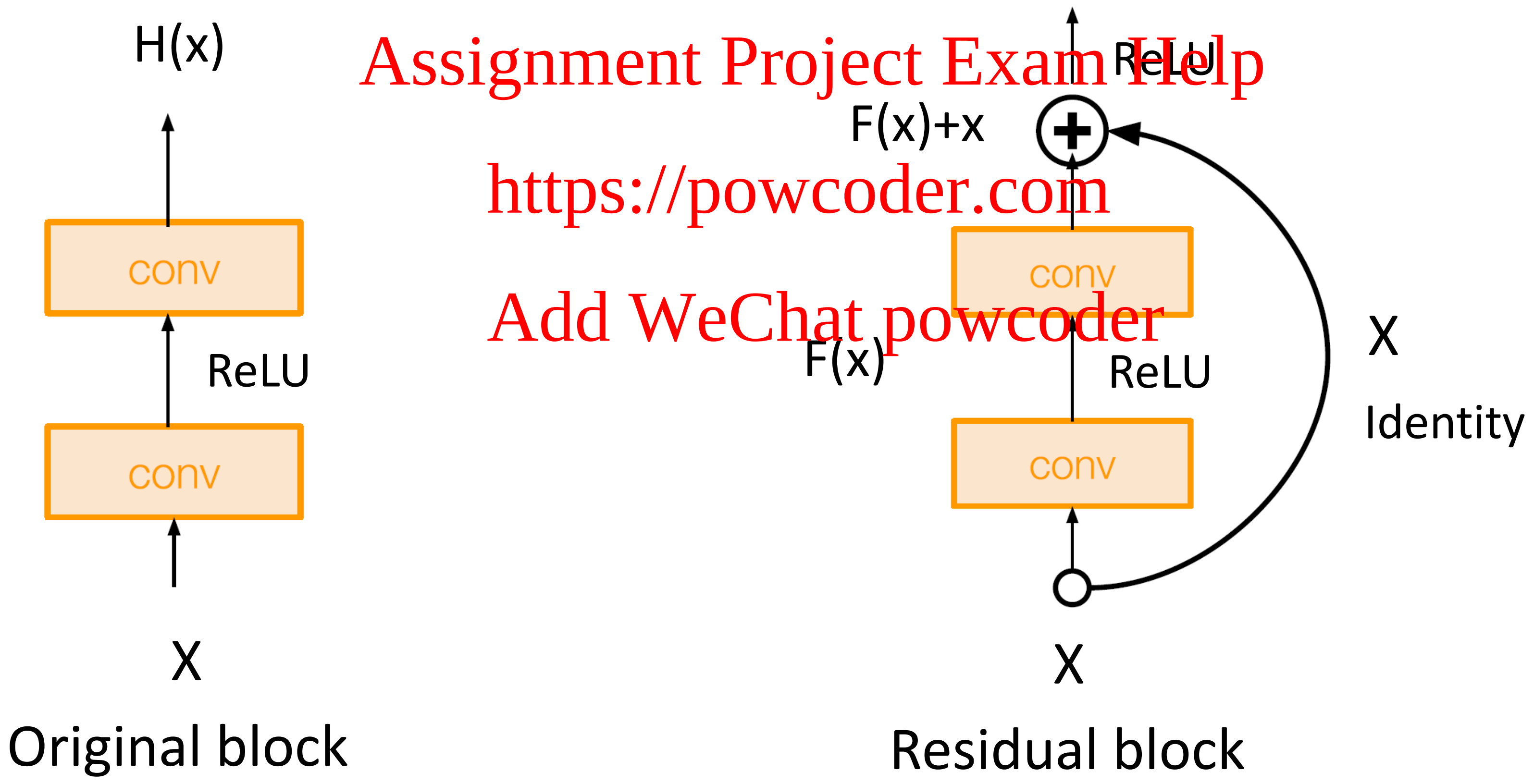
The deeper model should be able to perform at least as well as the shallower model.

-> Solution by construction is copying the learned layers from the shallower model and setting additional layers to identity mapping.

Case Study: ResNet

Solution

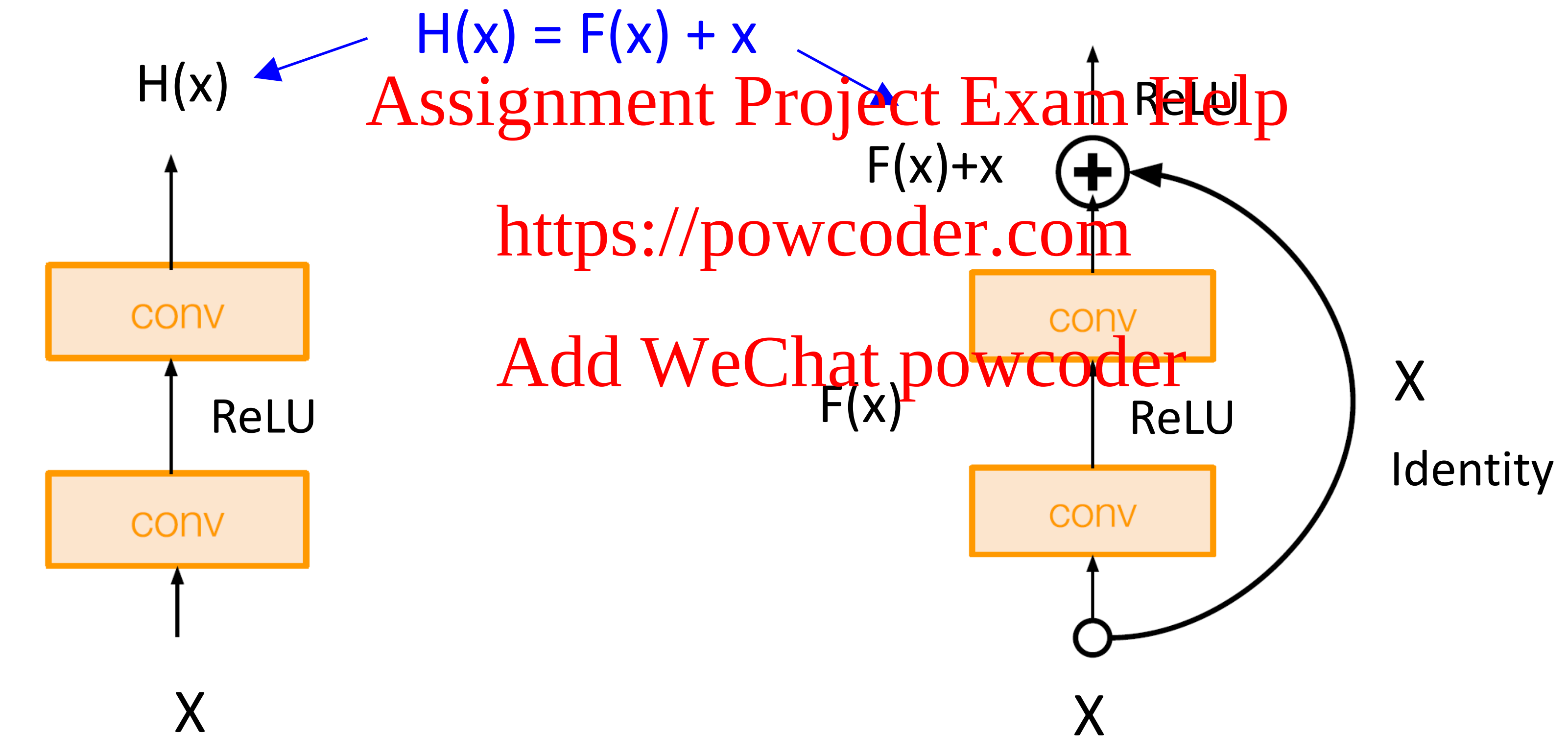
Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping



Case Study: ResNet

Solution

Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping

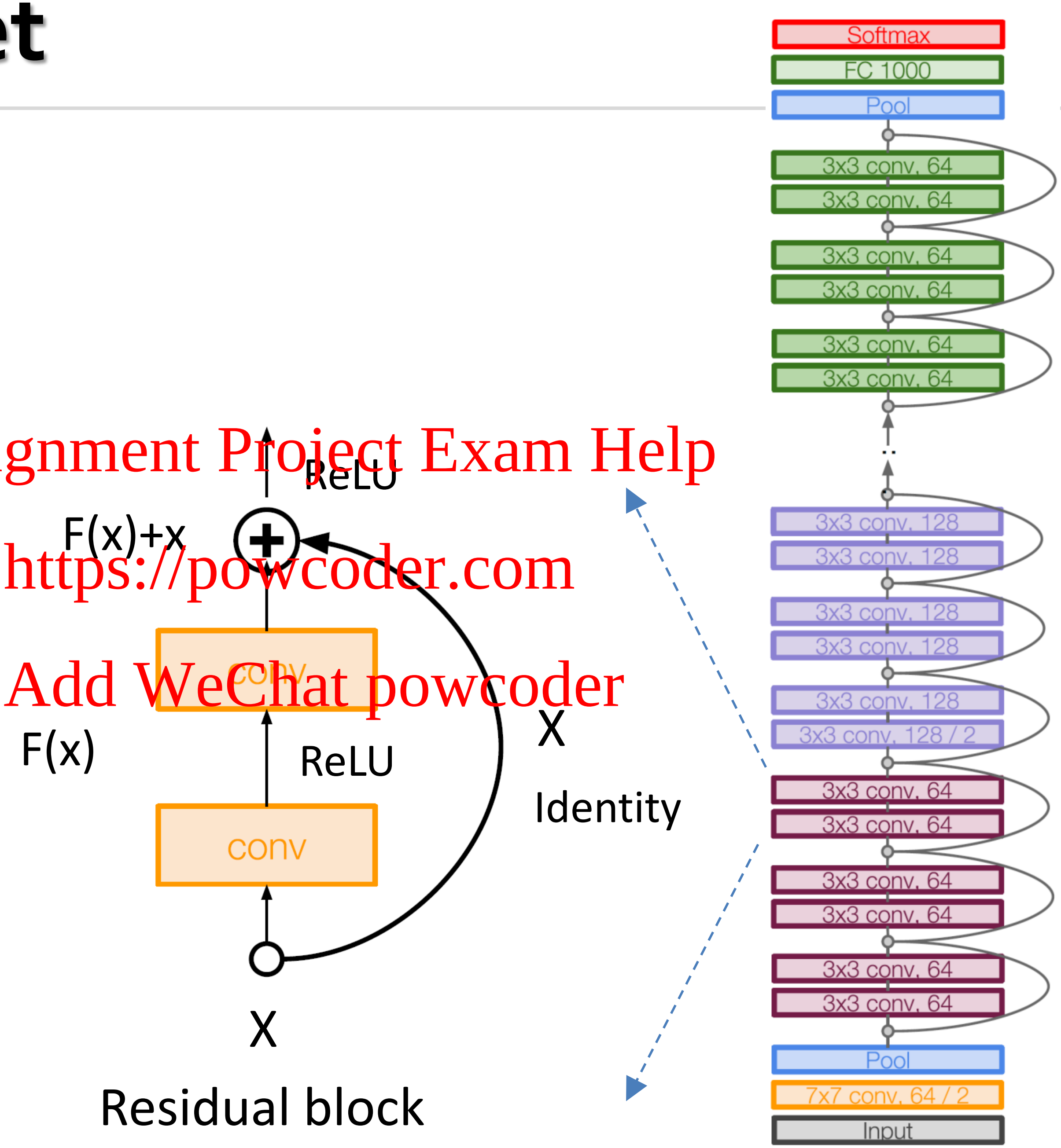


Original block   Residual block

Use layers to fit residual F(x) = H(x) - x instead of H(x) directly

Case Study: ResNet

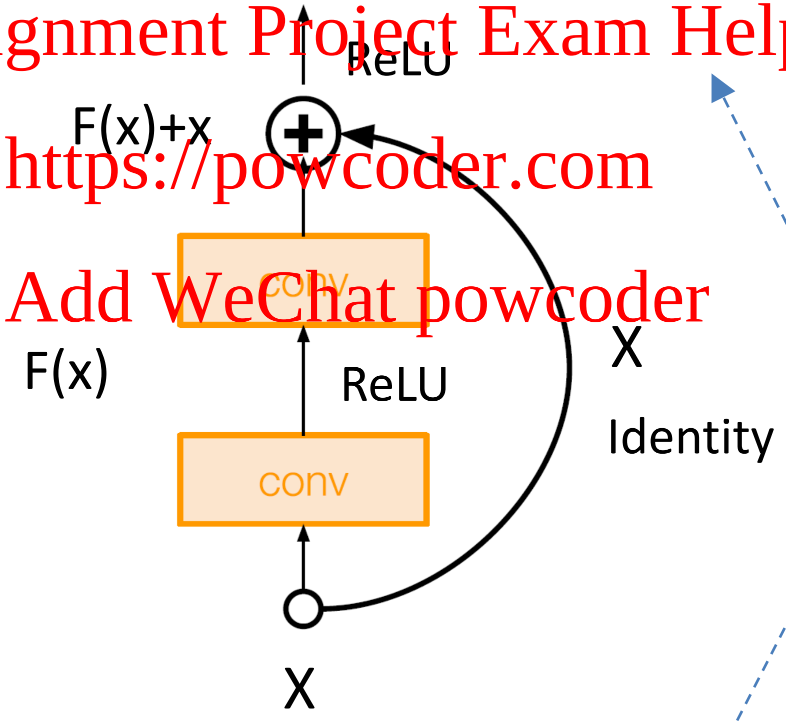
Full ResNet architecture:
- Stack residual blocks
- Every residual block has two 3x3 conv layers



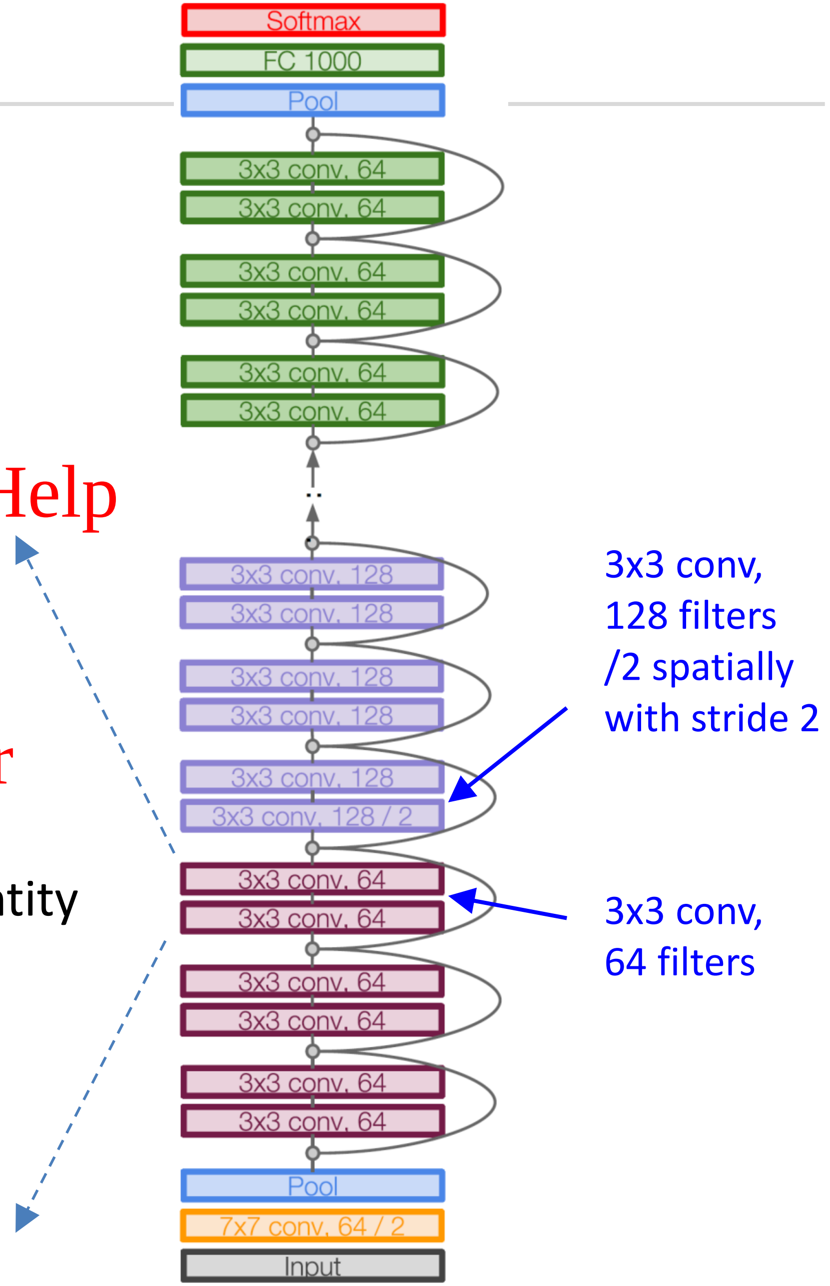
Case Study: ResNet

Full ResNet architecture:
- Stack residual blocks
- Every residual block has two 3x3 conv layers

- Periodically, double # of filters and downsample spatially using stride 2 (/2 in each dimension)

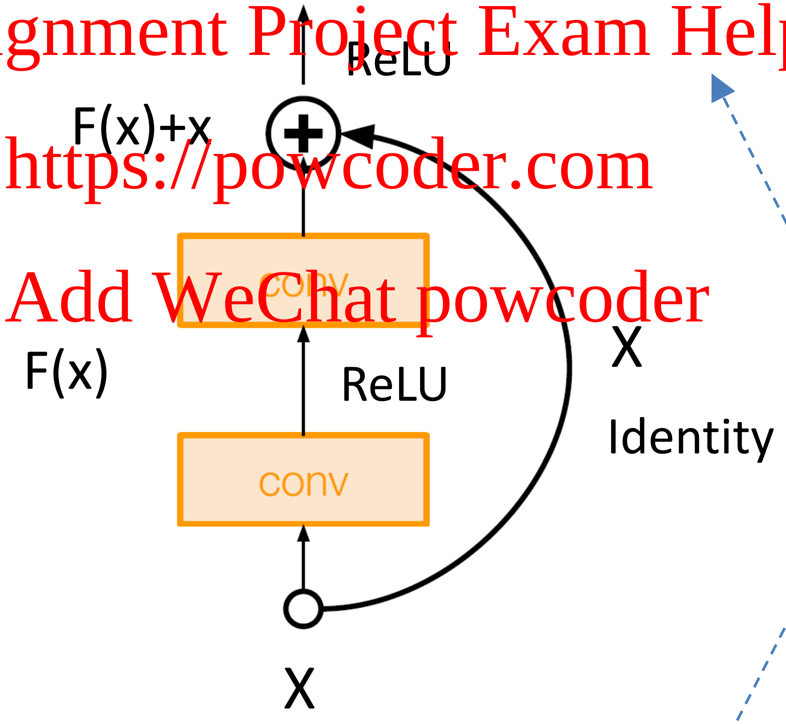


Residual block

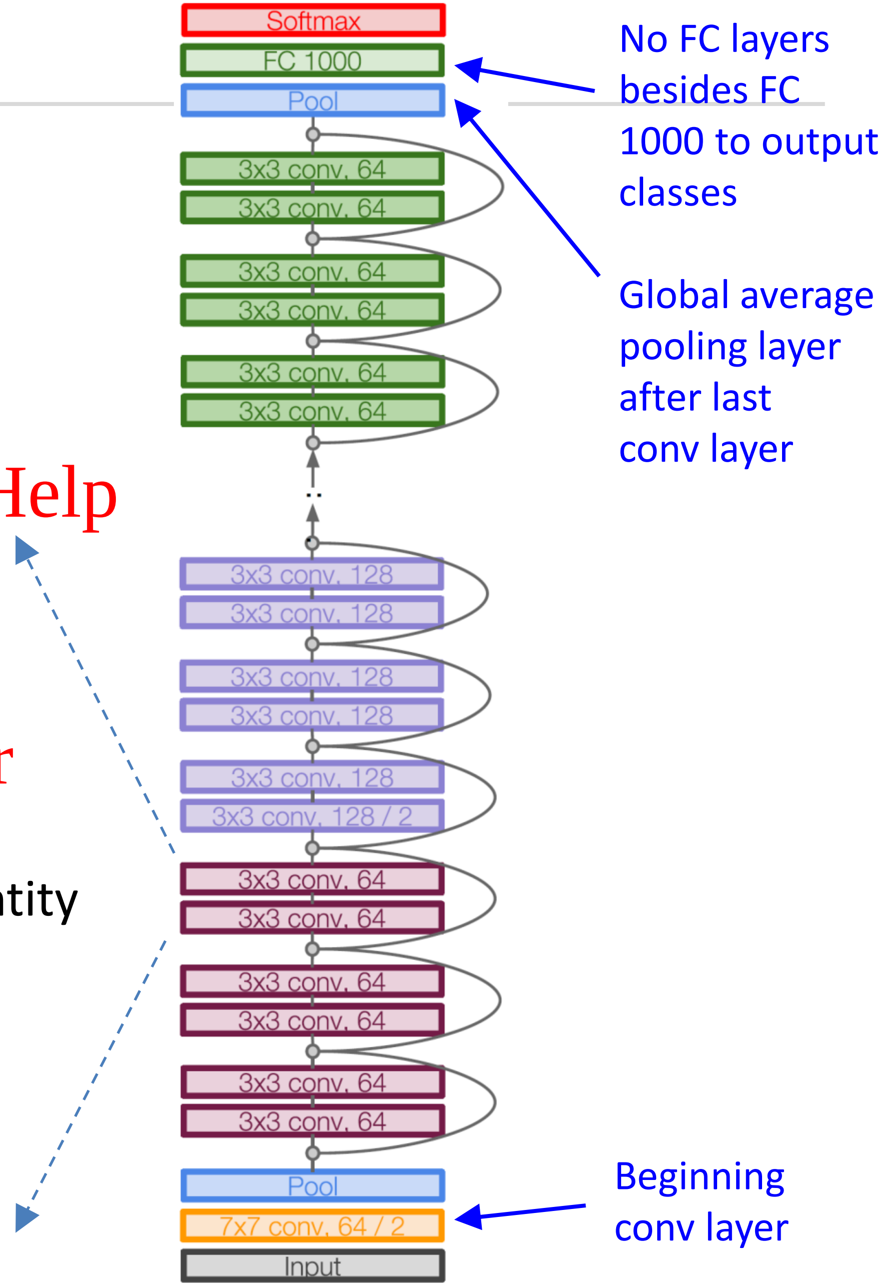


Case Study: ResNet

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



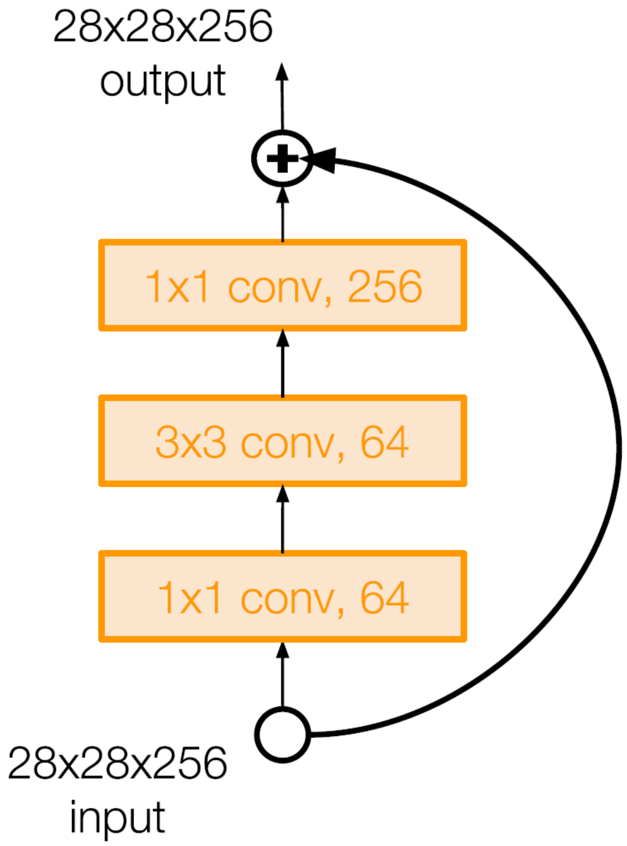
Residual block



Case Study: ResNet

Total depths of 18, 34, 50, 101, or 152 layers
(ResNet-18, ResNet-34, ResNet-50, ResNet-101, ResNet-152)

For deeper networks (ResNet-50+), use "bottleneck" layer to improve efficiency (similar to GoogLeNet)



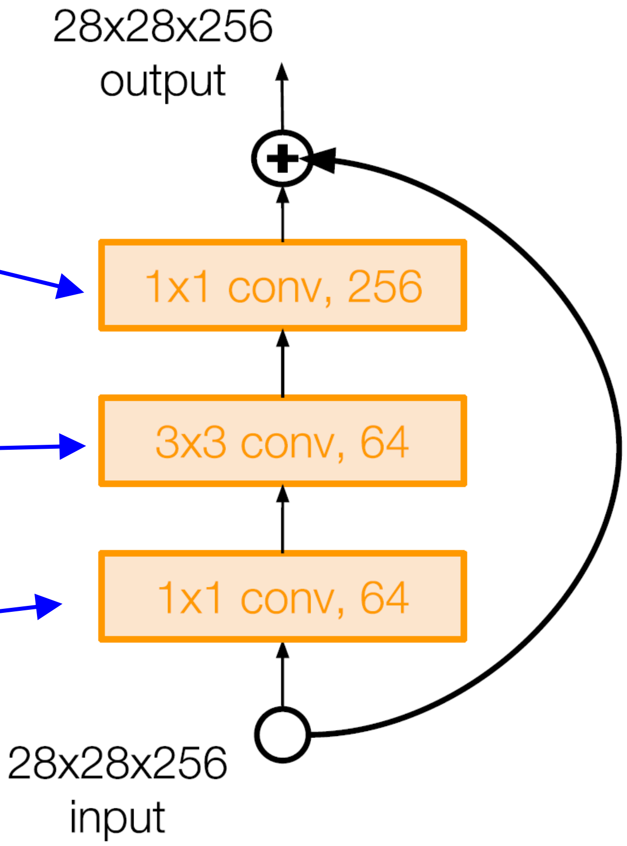
Case Study: ResNet

For deeper networks (ResNet-50+), use "bottleneck" layer to improve efficiency (similar to GoogLeNet)

1x1 conv, 256 filters projects back to 256 feature maps (28x28x256)

3x3 conv operates over only 64 feature maps

1x1 conv, 64 filters to project to 28x28x64



Case Study: ResNet

- Experimental Results
  - Able to train very deep networks without degrading (152 layers on ImageNet, 1202 on Cifar)
  - Deeper networks now achieve lowing training error as expected
  - Swept 1st place in all ILSVRC and COCO 2015 competitions

ILSVRC 2015 classification winner (3.6% top 5 error)
better than "human performance"!
(Russakovsky 2014)