

Convolutional Neural Networks IV

CS 4391 Introduction Computer Vision

Instructor Yu Xiang

The University of Texas at Dallas

Supervised Learning



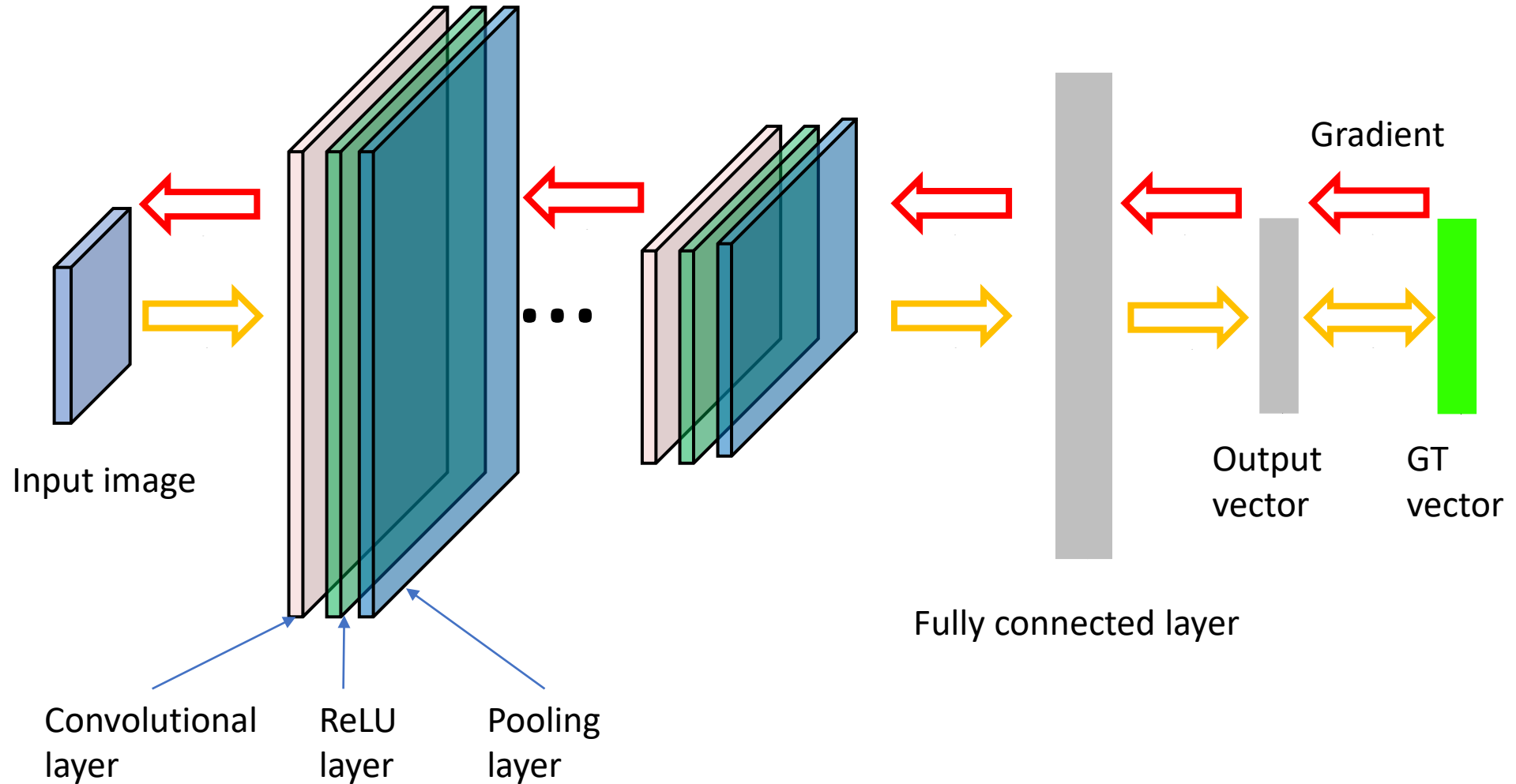
$$f(\mathbf{x})$$

Training Data $\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^N$

Input

Output

Training: back-propagate errors



Classification Loss Functions

- Cross entropy loss

$$H(p, q) = -\mathbb{E}_p[\log q]$$

$$H(p, q) = -\sum_{x \in \mathcal{X}} p(x) \log q(x)$$

$$L_{CE} = - \sum_{i=0}^{m-1} \underset{\substack{\uparrow \\ \text{Binary} \\ \text{ground} \\ \text{truth label}}}{t_i} \log \underset{\substack{\uparrow \\ \text{Logit}}}{\sigma(\mathbf{y})_i}$$

Regression Loss Functions

- Mean Absolute Loss or L1 loss

$$L_1(x) = |x|$$

$$f(y, \hat{y}) = \sum_{i=1}^N |y_i - \hat{y}_i|$$

- Mean Square Loss or L2 loss

$$L_2(x) = x^2$$

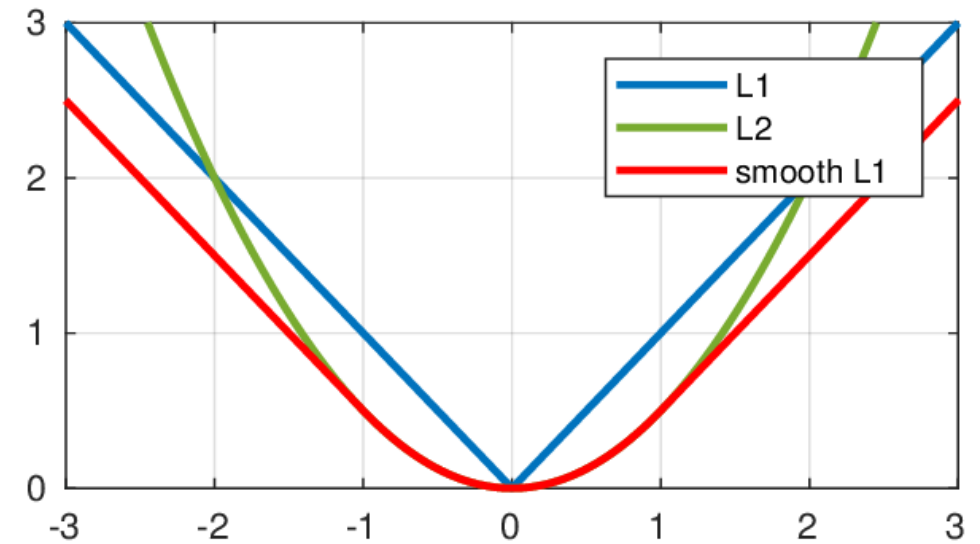
$$f(y, \hat{y}) = \sum_{i=1}^N (y_i - \hat{y}_i)^2$$

Regression Loss Functions

- Smooth L1 loss

$$\text{smooth } L_1(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise} \end{cases}$$

$$f(y, \hat{y}) = \begin{cases} 0.5(y - \hat{y})^2 & \text{if } |y - \hat{y}| < 1 \\ |y - \hat{y}| - 0.5 & \text{otherwise} \end{cases}$$

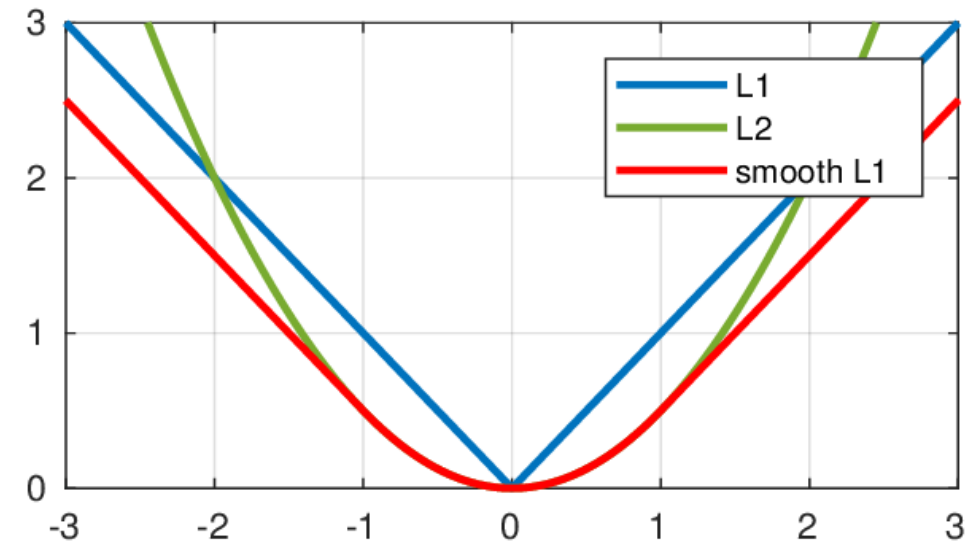


Regression Loss Functions

- Huber loss
 - Generalization of smooth L1 loss ($\delta = 1$)

$$L_{\delta}(a) = \begin{cases} \frac{1}{2}a^2 & \text{for } |a| \leq \delta, \\ \delta(|a| - \frac{1}{2}\delta), & \text{otherwise.} \end{cases}$$

$$L_{\delta}(y, f(x)) = \begin{cases} \frac{1}{2}(y - f(x))^2 & \text{for } |y - f(x)| \leq \delta, \\ \delta(|y - f(x)| - \frac{1}{2}\delta), & \text{otherwise.} \end{cases}$$



Optimization

- Gradient descent
 - Gradient direction: steepest direction to increase the objective
 - Can only find local minimum
 - Widely used for neural network training (works in practice)
 - Compute gradient with a mini-batch (Stochastic Gradient Descent, SGD)

$$W \leftarrow W - \underset{\substack{\text{Learning rate}}}{\gamma} \frac{\partial L}{\partial W}$$

Optimization

- Gradient descent with momentum
 - Add a fraction of the update vector from previous time step (momentum)
 - Accelerated SGD, reduced oscillation



Image 2: SGD without momentum



Image 3: SGD with momentum

momentum Learning rate

$$v_t = \gamma v_{t-1} + \eta \nabla_{\theta} J(\theta)$$
$$\theta = \theta - v_t$$

<https://ruder.io/optimizing-gradient-descent/>

Optimization

- Adam: Adaptive Moment Estimation

1. Exponentially decaying average of gradients and squared gradients

$$g_t = \nabla_{\theta} f_t(\theta_t)$$

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

$$\beta_1 = 0.9, \beta_2 = 0.999$$

Start m and v from 0s

2. Bias-corrected 1st and 2nd moment estimates

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

3. Updating rule

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

Learning rate

$$\epsilon = 10^{-8}$$

Adaptive learning rate

<https://arxiv.org/pdf/1412.6980.pdf>

PyTorch Example

```
optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.9)  
optimizer = optim.Adam([var1, var2], lr=0.0001)
```

```
for input, target in dataset:  
    optimizer.zero_grad()  
    output = model(input)  
    loss = loss_fn(output, target)  
    loss.backward()  
    optimizer.step()
```

<https://pytorch.org/docs/stable/optim.html>

Case Study: Training AlexNet

- Data augmentation
 - Extracting random 224x224 patches from 256x256 images

- Change RGB intensities

$$\begin{bmatrix} I_{xy}^R, I_{xy}^G, I_{xy}^B \end{bmatrix}^T + \begin{bmatrix} \mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3 \end{bmatrix} \begin{bmatrix} \alpha_1 \lambda_1, \alpha_2 \lambda_2, \alpha_3 \lambda_3 \end{bmatrix}^T$$

Eigen vectors
of 3x3 covariance
matrix of RGB values
on training set

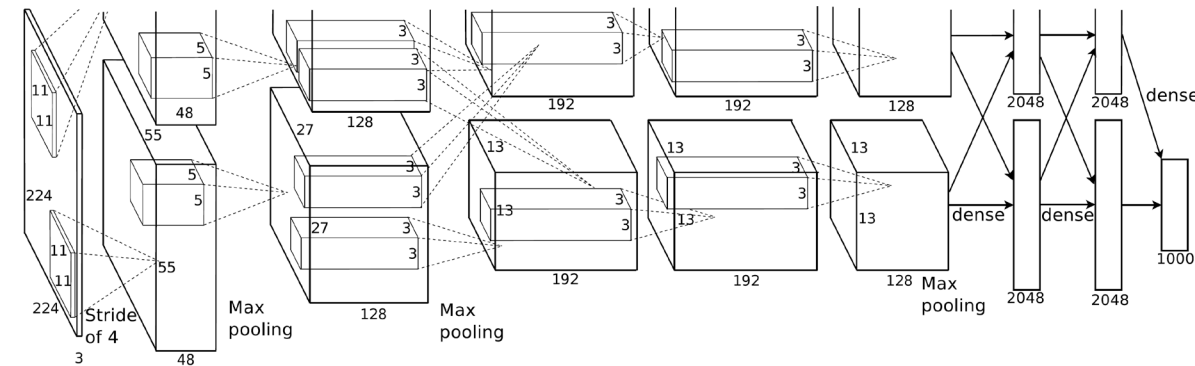
Random variable
 $N(0, 0.1)$

Eigen values

covariance matrix

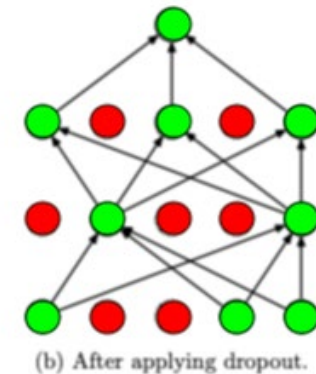
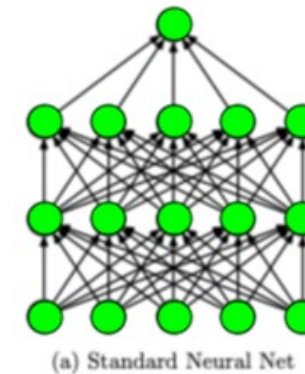
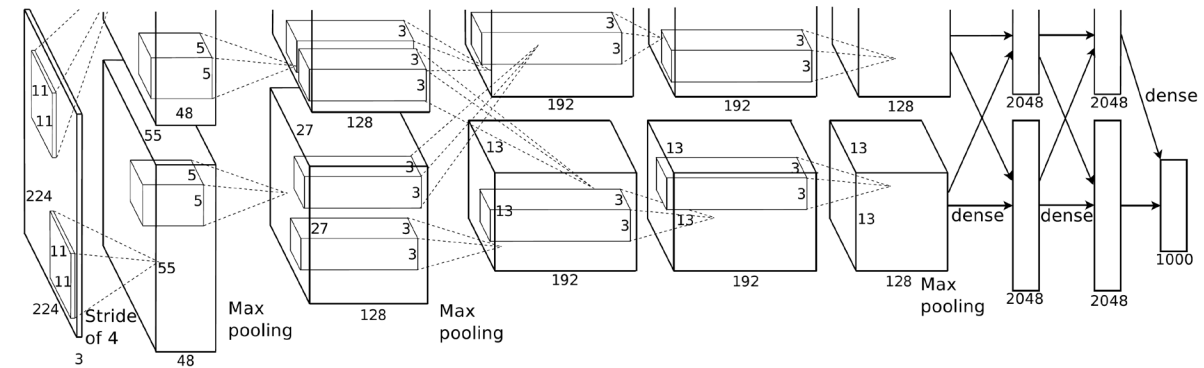
$$S = \frac{1}{n-1} \sum_{i=1}^n (X_i - \bar{X})(X_i - \bar{X})'$$

<https://papers.nips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html>



Case Study: Training AlexNet

- Dropout
 - Set to zero the output of each hidden neuron with probability 0.5
 - Apply to the first two FC layers
 - Prevent overfitting



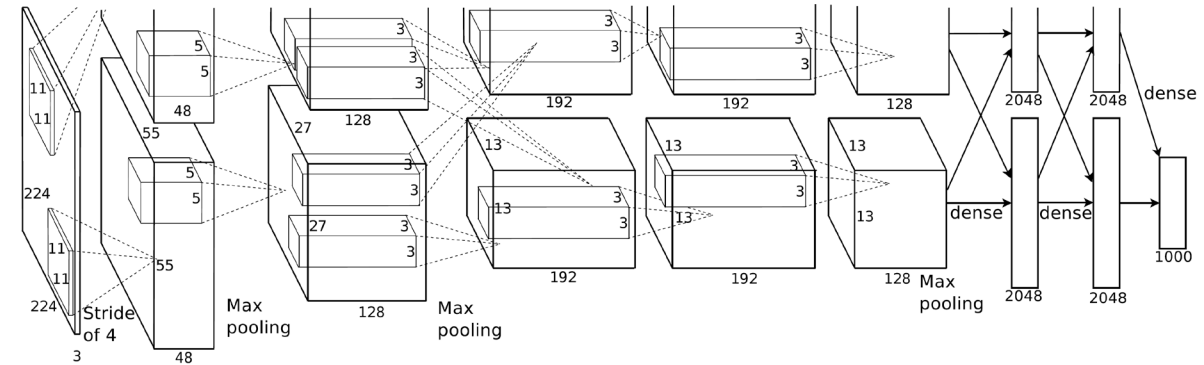
<https://papers.nips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html>

Case Study: Training AlexNet

- Batch size: 128
- Updating rule

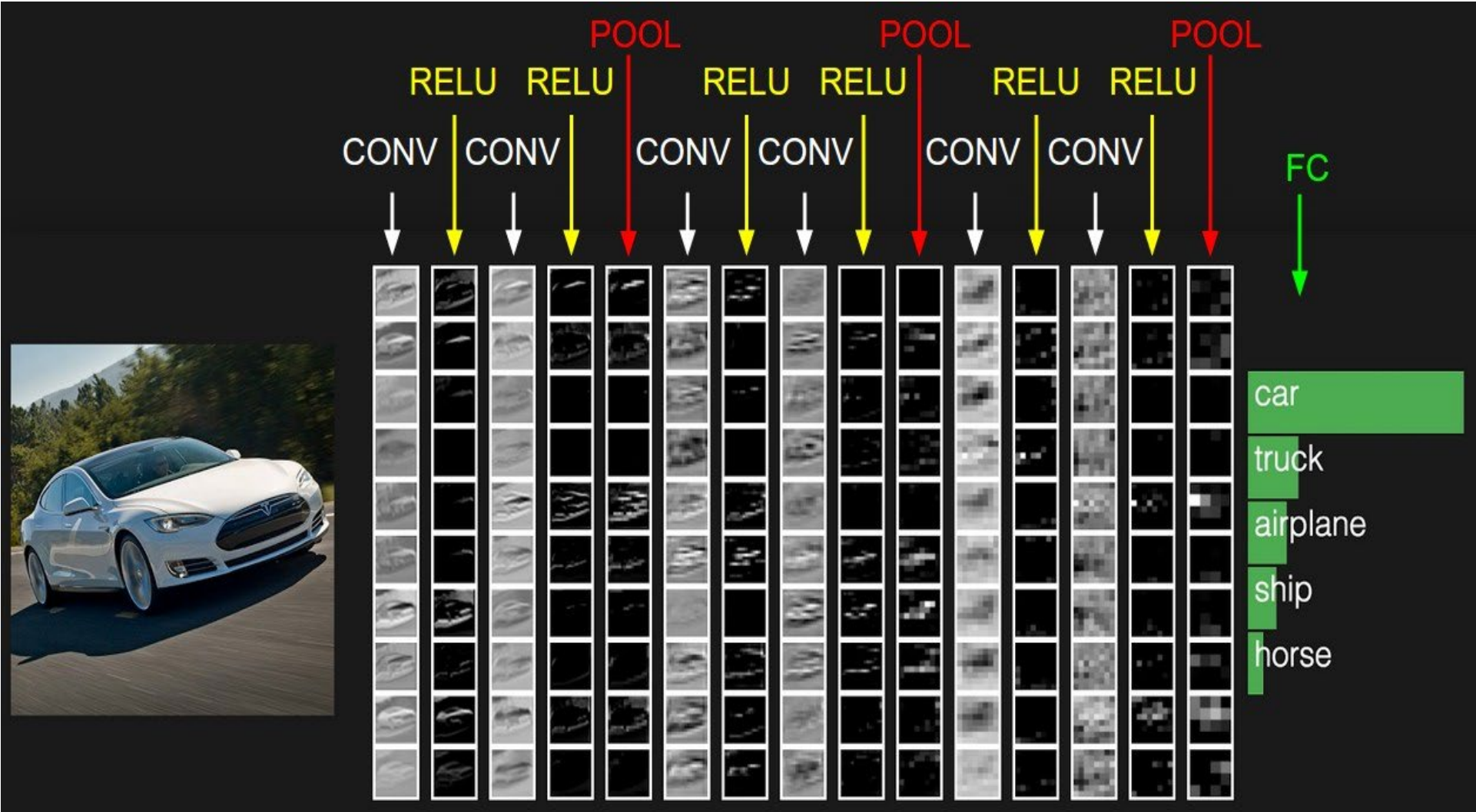
$$w_{i+1} := w_i + v_{i+1}$$

$$v_{i+1} := \underset{\text{Momentum}}{0.9} \cdot v_i - \underset{\text{Weight Decay}}{0.0005} \cdot \epsilon \cdot w_i - \underset{\text{Learning rate}}{\epsilon} \cdot \underset{\text{Gradient}}{\left\langle \frac{\partial L}{\partial w} \Big|_{w_i} \right\rangle_{D_i}}$$



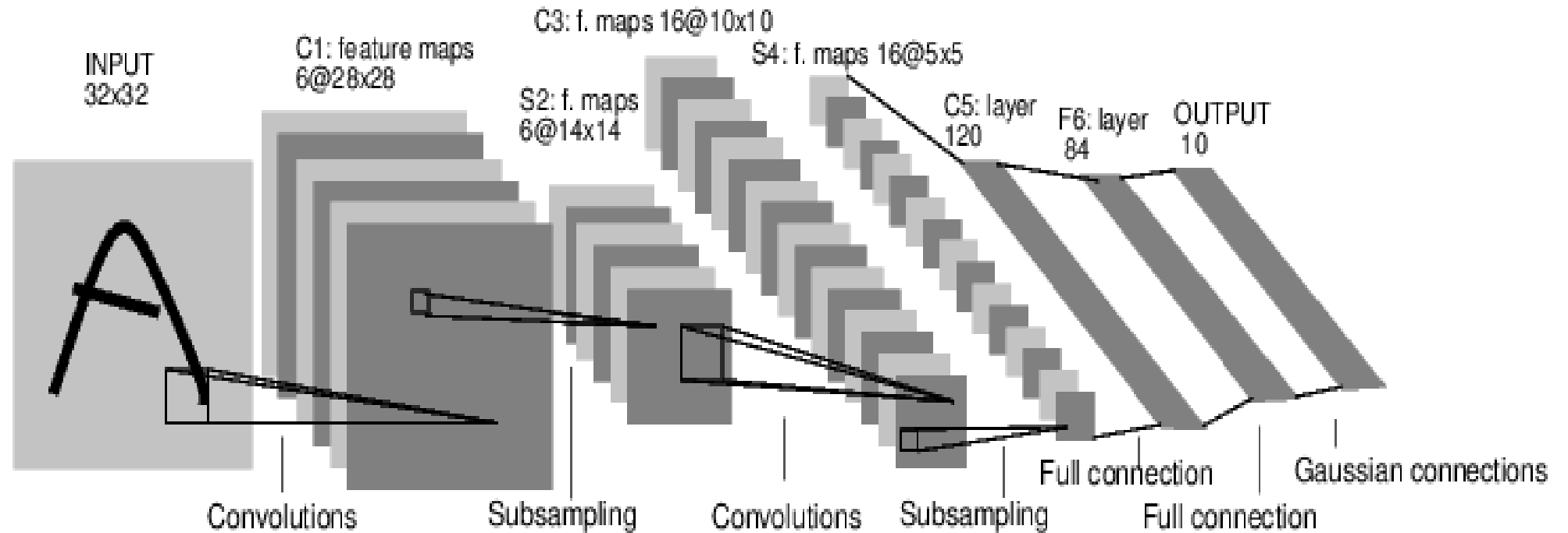
Five to six days on two NVIDIA GTX 580 3GB GPUs, 2012

<https://papers.nips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html>



Case Study: LeNet-5

[LeCun et al., 1998]



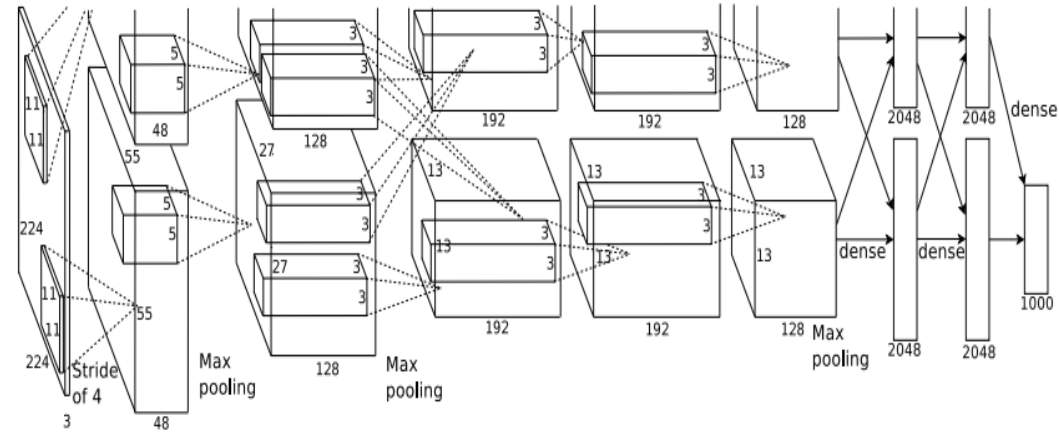
Conv filters were 5x5, applied at stride 1

Subsampling (Pooling) layers were 2x2 applied at stride 2

i.e. architecture is [CONV-POOL-CONV-POOL-CONV-FC]

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

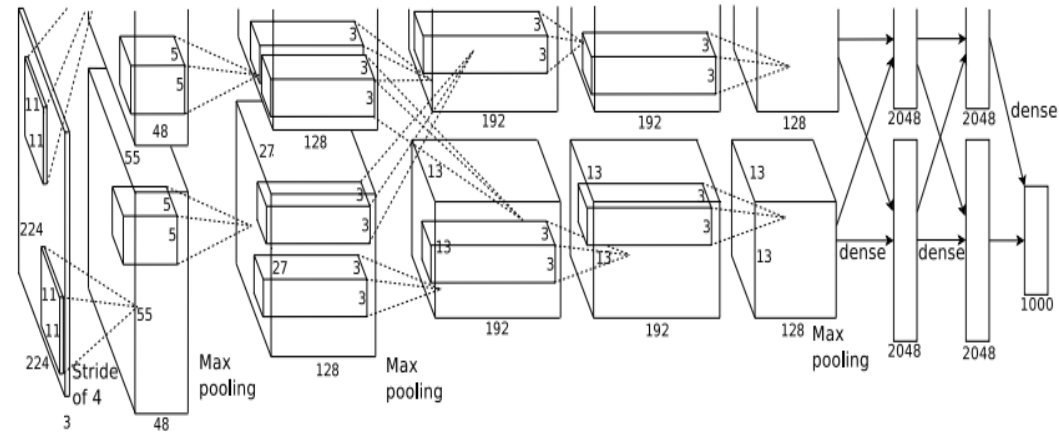
First layer (CONV1): 96 11x11 filters applied at stride 4

=>

Q: what is the output volume size? Hint: $(227-11)/4+1 = 55$

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

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

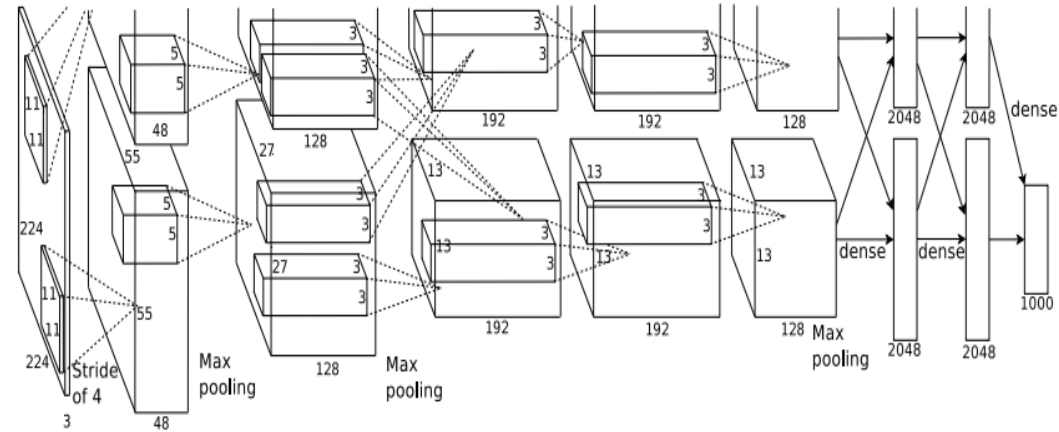
=>

Output volume [55x55x96]

Q: What is the total number of parameters in this layer?

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

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

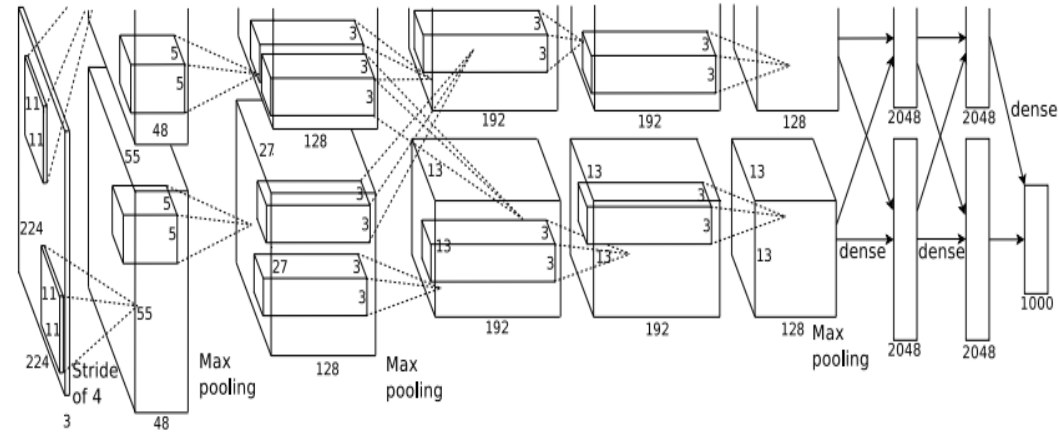
=>

Output volume **[55x55x96]**

Parameters: $(11*11*3)*96 = \mathbf{35K}$

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

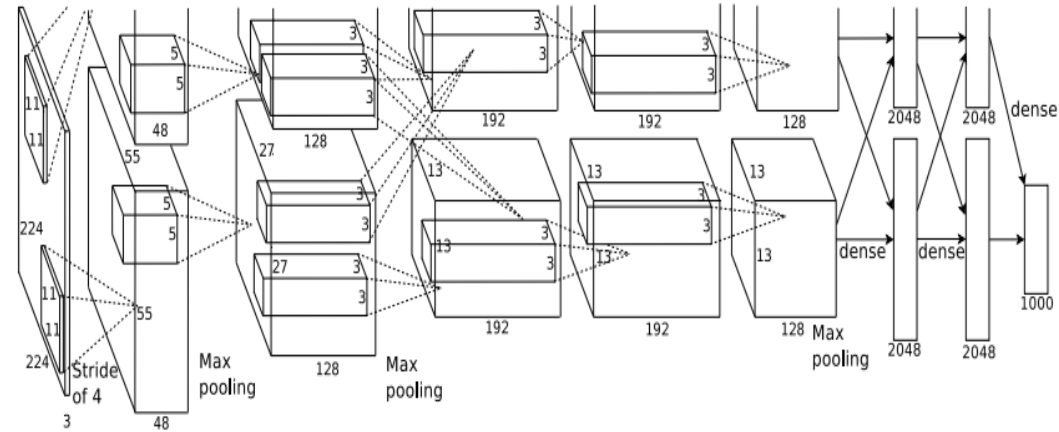
After CONV1: 55x55x96

Second layer (POOL1): 3x3 filters applied at stride 2

Q: what is the output volume size? Hint: $(55-3)/2+1 = 27$

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

After CONV1: 55x55x96

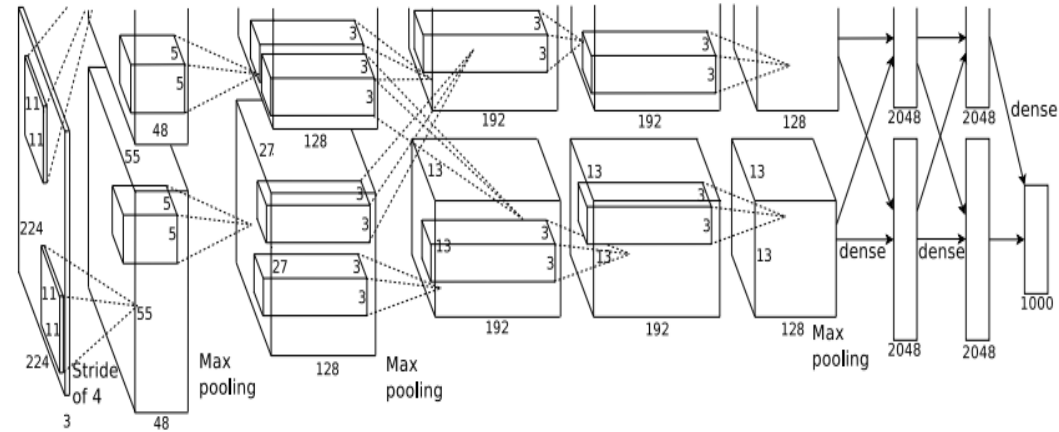
Second layer (POOL1): 3x3 filters applied at stride 2

Output volume: 27x27x96

Q: what is the number of parameters in this layer?

Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

After CONV1: 55x55x96

Second layer (POOL1): 3x3 filters applied at stride 2

Output volume: 27x27x96

Parameters: 0!

Case Study: AlexNet

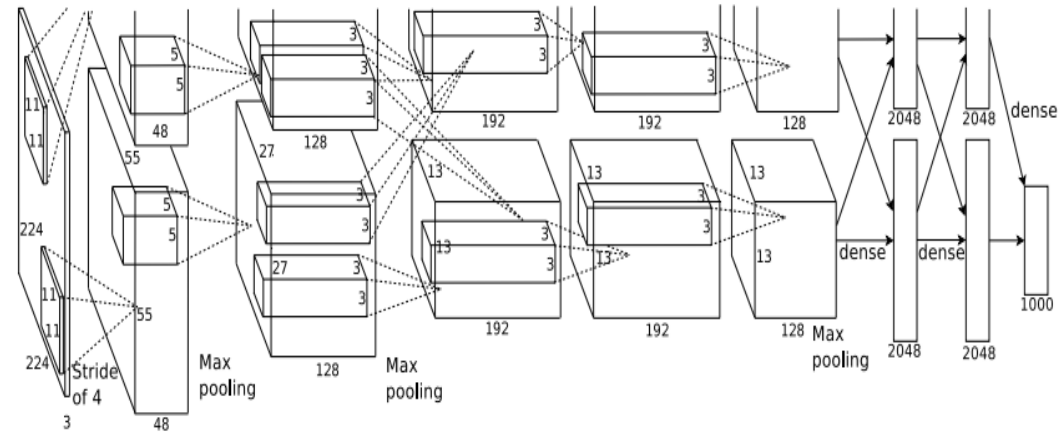
[Krizhevsky et al. 2012]

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

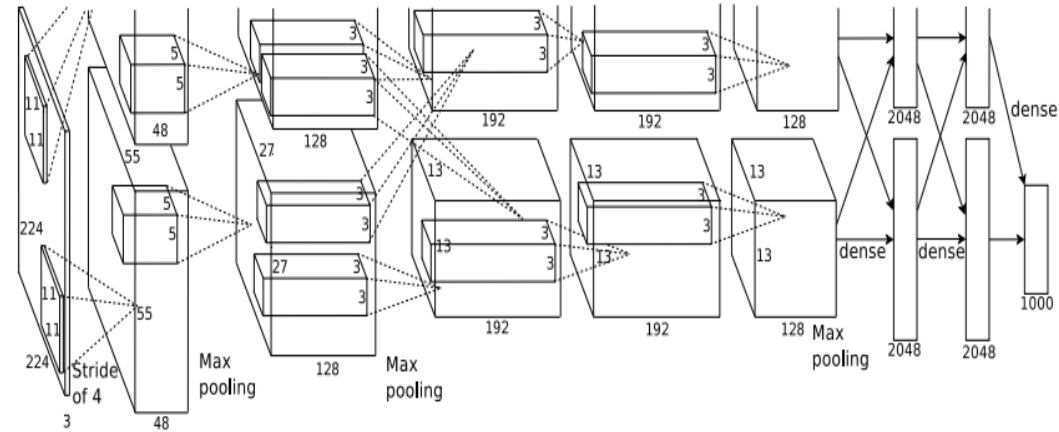
[13x13x256] **CONV5**: 256 3x3 filters at stride 1, pad 1

[6x6x256] **MAX POOL3**: 3x3 filters at stride 2

[4096] **FC6**: 4096 neurons

[4096] **FC7**: 4096 neurons

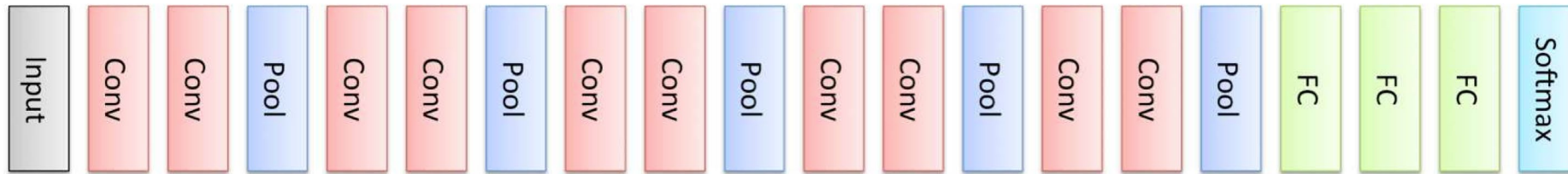
[1000] **FC8**: 1000 neurons (class scores)



Case Study: VGGNet

[Simonyan and Zisserman, 2014]

VGGNet



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

11.2% top 5 error in ILSVRC 2013

->

7.3% top 5 error

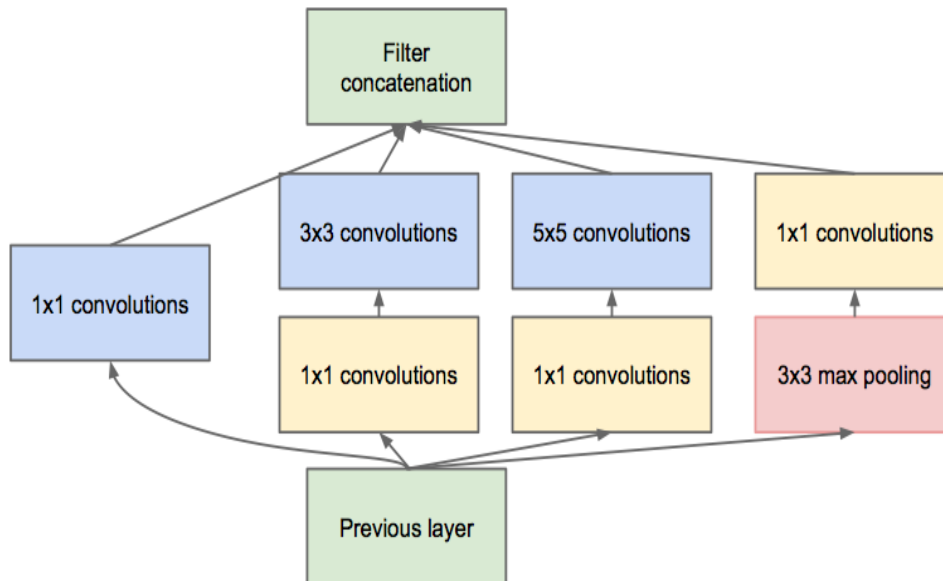
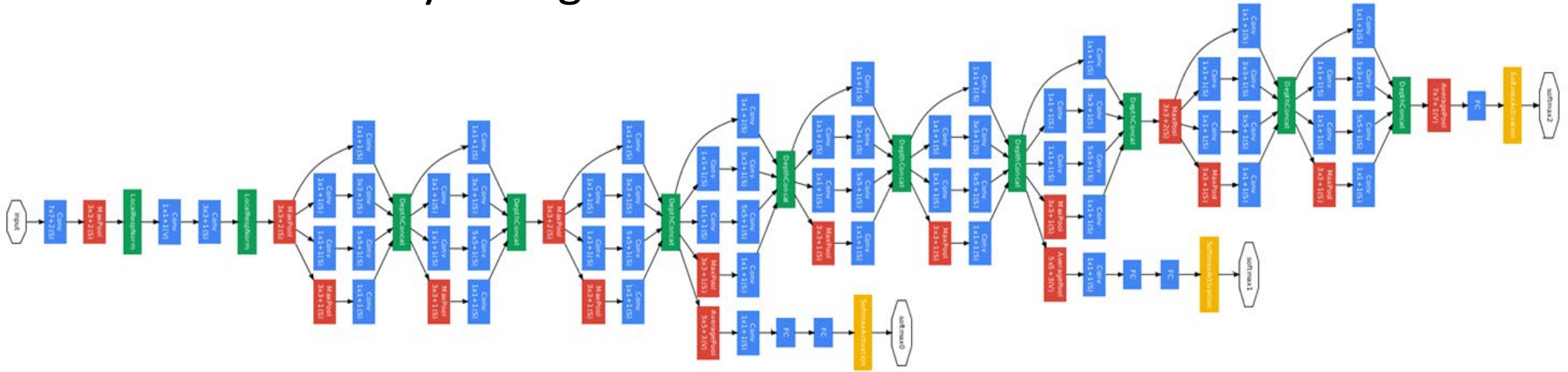
Case Study: VGGNet

[Simonyan and Zisserman, 2014]

INPUT: [224x224x3] memory: $224*224*3=150\text{K}$ params: 0
CONV3-64: [224x224x64] memory: $224*224*64=3.2\text{M}$ params: $(3*3*3)*64 = 1,728$
CONV3-64: [224x224x64] memory: $224*224*64=3.2\text{M}$ params: $(3*3*64)*64 = 36,864$
POOL2: [112x112x64] memory: $112*112*64=800\text{K}$ params: 0
CONV3-128: [112x112x128] memory: $112*112*128=1.6\text{M}$ params: $(3*3*64)*128 = 73,728$
CONV3-128: [112x112x128] memory: $112*112*128=1.6\text{M}$ params: $(3*3*128)*128 = 147,456$
POOL2: [56x56x128] memory: $56*56*128=400\text{K}$ params: 0
CONV3-256: [56x56x256] memory: $56*56*256=800\text{K}$ params: $(3*3*128)*256 = 294,912$
CONV3-256: [56x56x256] memory: $56*56*256=800\text{K}$ params: $(3*3*256)*256 = 589,824$
CONV3-256: [56x56x256] memory: $56*56*256=800\text{K}$ params: $(3*3*256)*256 = 589,824$
POOL2: [28x28x256] memory: $28*28*256=200\text{K}$ params: 0
CONV3-512: [28x28x512] memory: $28*28*512=400\text{K}$ params: $(3*3*256)*512 = 1,179,648$
CONV3-512: [28x28x512] memory: $28*28*512=400\text{K}$ params: $(3*3*512)*512 = 2,359,296$
CONV3-512: [28x28x512] memory: $28*28*512=400\text{K}$ params: $(3*3*512)*512 = 2,359,296$
POOL2: [14x14x512] memory: $14*14*512=100\text{K}$ params: 0
CONV3-512: [14x14x512] memory: $14*14*512=100\text{K}$ params: $(3*3*512)*512 = 2,359,296$
CONV3-512: [14x14x512] memory: $14*14*512=100\text{K}$ params: $(3*3*512)*512 = 2,359,296$
CONV3-512: [14x14x512] memory: $14*14*512=100\text{K}$ params: $(3*3*512)*512 = 2,359,296$
POOL2: [7x7x512] memory: $7*7*512=25\text{K}$ 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: $4096*1000 = 4,096,000$ (not counting biases)

Case Study: GoogLeNet

[Szegedy et al., 2014]



Inception module

ILSVRC 2014 winner (6.7% top 5 error)

Case Study: 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!

Compared to AlexNet:

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

Case Study: ResNet

[He et al., 2015]

ILSVRC 2015 winner (3.6% top 5 error)



LeNet (5 layers)



AlexNet
(8 layers)



VGGNet
(19 layers)



GoogleNet

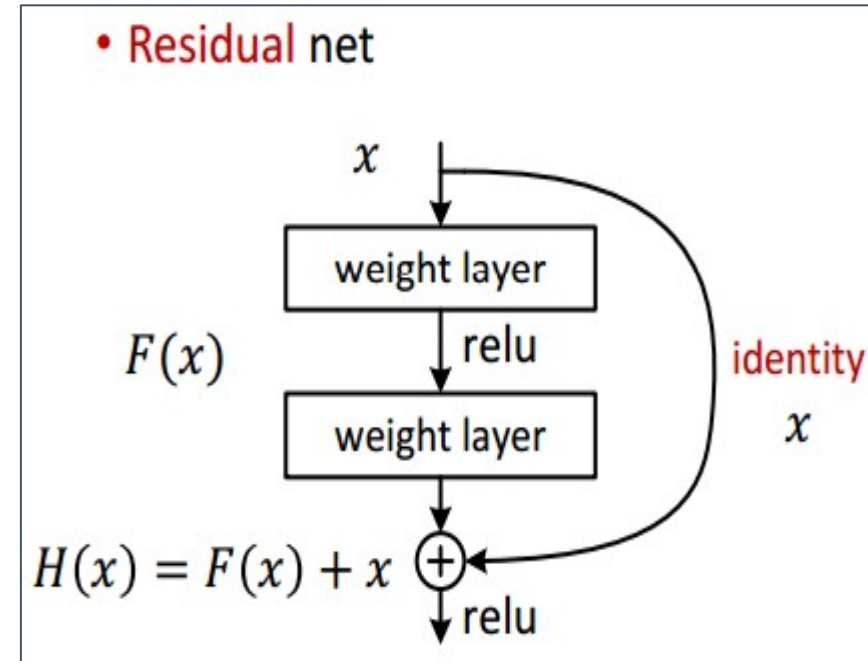
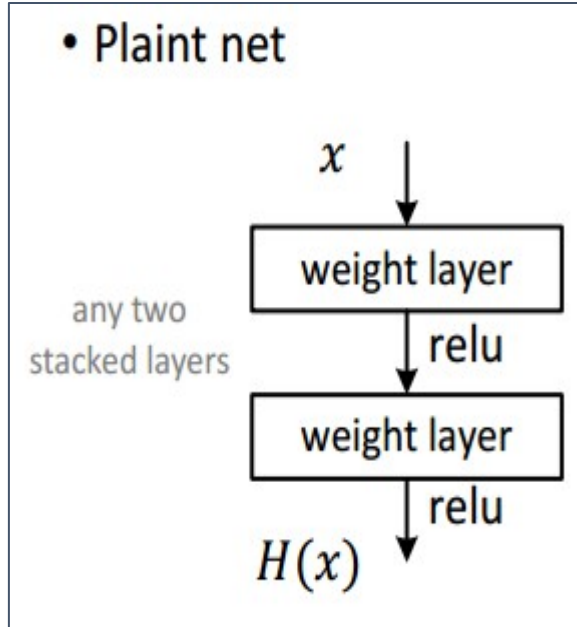
ResNet
(152 layers)

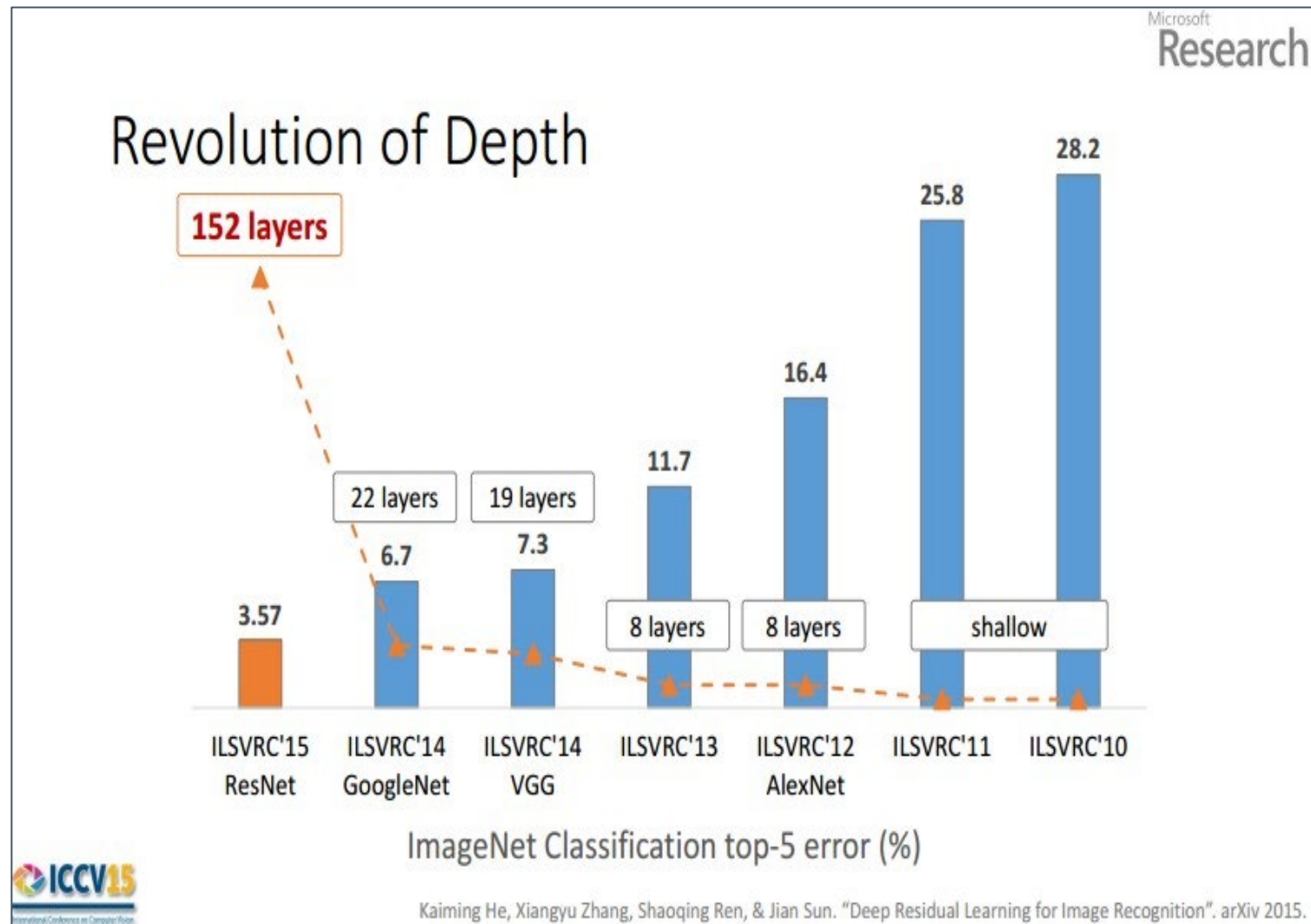
2-3 weeks of
training on 8
GPU machine

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

Case Study: ResNet

[He et al., 2015]





(slide from Kaiming He)

Further Reading

- Stanford CS231n, lecture 3 and lecture 4, <http://cs231n.stanford.edu/schedule.html>
- Deep learning with PyTorch https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html
- Dropout: A Simple Way to Prevent Neural Networks from Overfitting <https://jmlr.org/papers/v15/srivastava14a.html>
- Matrix Calculus: <https://explained.ai/matrix-calculus/>