Convolutional Neural Networks IV: Loss Function and Optimization

CS 4391 Introduction Computer Vision

Instructor Yu Xiang

The University of Texas at Dallas

Supervised Learning

$\mathbf{x}$ ⇒  ⇒ $\mathbf{y}$

Image

Object class

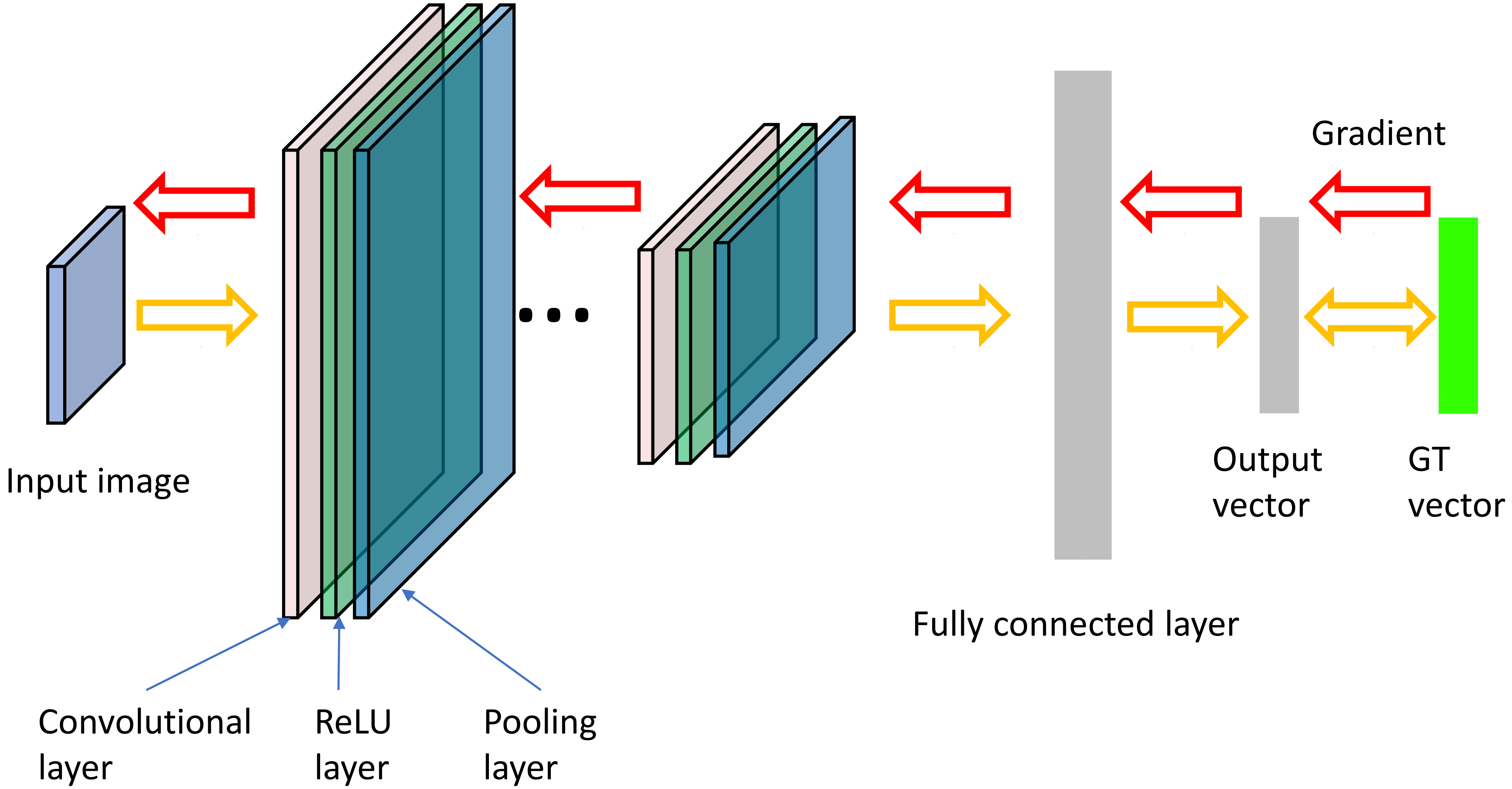
$$f(\mathbf{x})$$

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

Input

Output

Training: back-propagate errors



Classification Loss Functions

- Cross entropy loss

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

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

$$L_{CE} = -\sum_{i=0}^{m-1} t_i \log \sigma(\mathbf{y})_i$$

Binary ground truth label

Logit

Regression Loss Functions

- Mean Absolute Loss or L1 loss

$$L_1(x) = |x| \qquad f(y, \hat{y}) = \sum_{i=1}^{N} |y_i - \hat{y}_i|$$

- Mean Square Loss or L2 loss

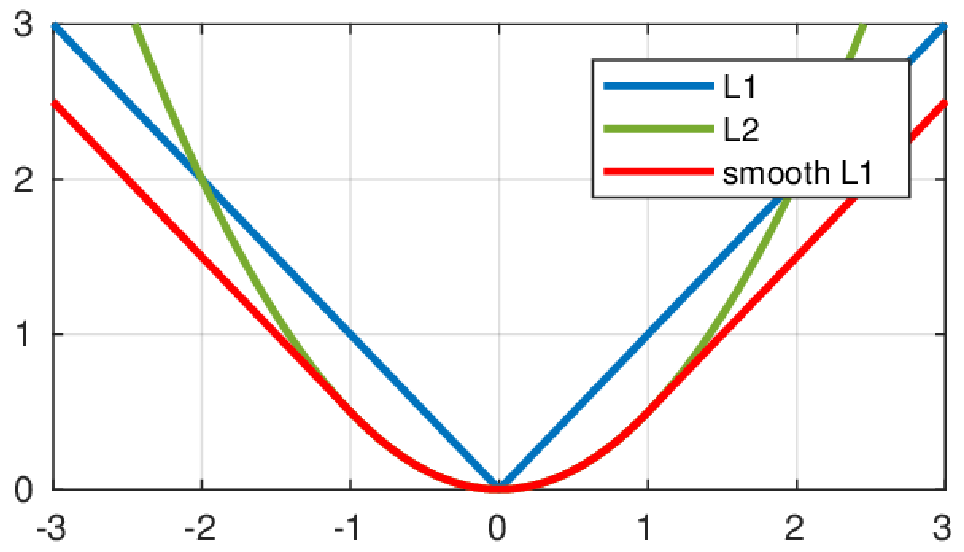
$$L_2(x) = x^2 \qquad f(y, \hat{y}) = \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

Regression Loss Functions

- Smooth L1 loss

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

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

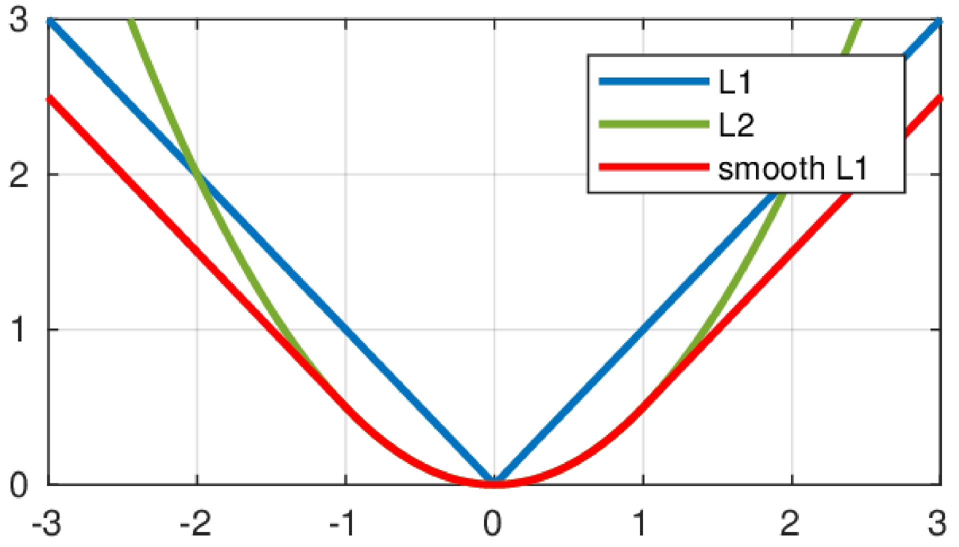


Regression Loss Functions

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

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

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



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)

$$W \leftarrow W - \gamma \frac{\partial L}{\partial W}$$

Learning rate

Optimization

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

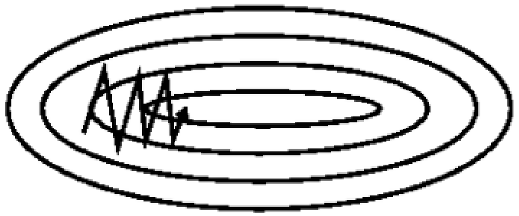


Image 2: SGD without momentum

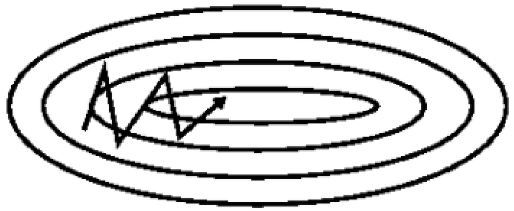


Image 3: SGD with momentum

$$v_t = \overset{\text{momentum}}{\gamma} v_{t-1} + \overset{\text{Learning rate}}{\eta} \nabla_\theta J(\theta)$$
$$\theta = \theta - v_t$$

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

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

  3. Updating rule

Learning rate

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t \qquad \epsilon = 10^{-8}$$

Adaptive learning rate

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

PyTorch Example

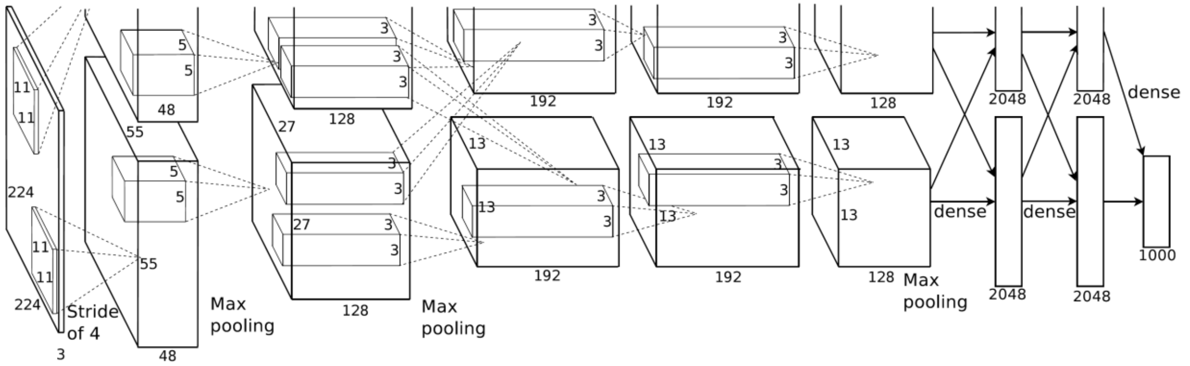
```
optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.9)
optimizer = optim.Adam([var1, var2], lr=0.0001)
```

```
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



$$[I_{xy}^R, I_{xy}^G, I_{xy}^B]^T$$

$$+ \quad [\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3][\alpha_1\lambda_1, \alpha_2\lambda_2, \alpha_3\lambda_3]^T$$

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

Random variable N(0, 0.1)

Eigen values

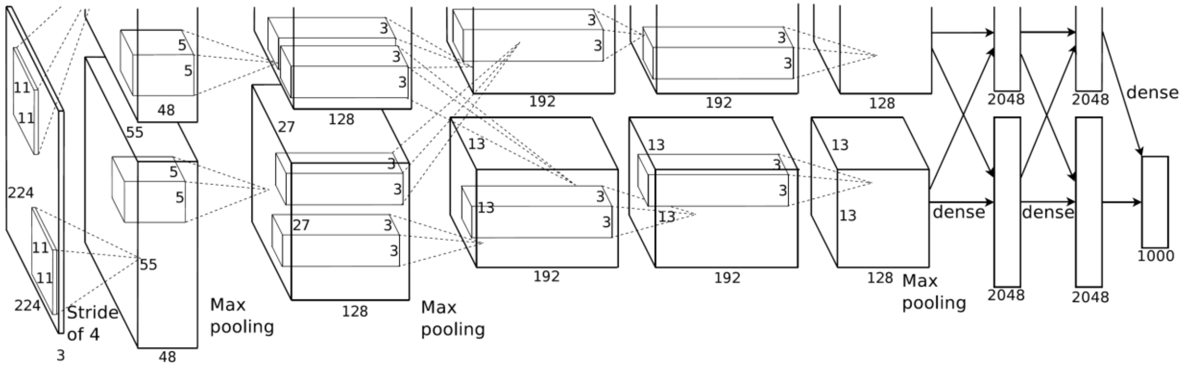
covariance matrix

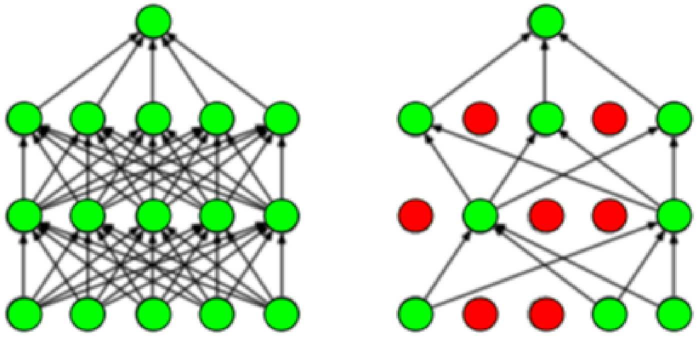
$$S = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \bar{X})(X_i - \bar{X})'$$

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



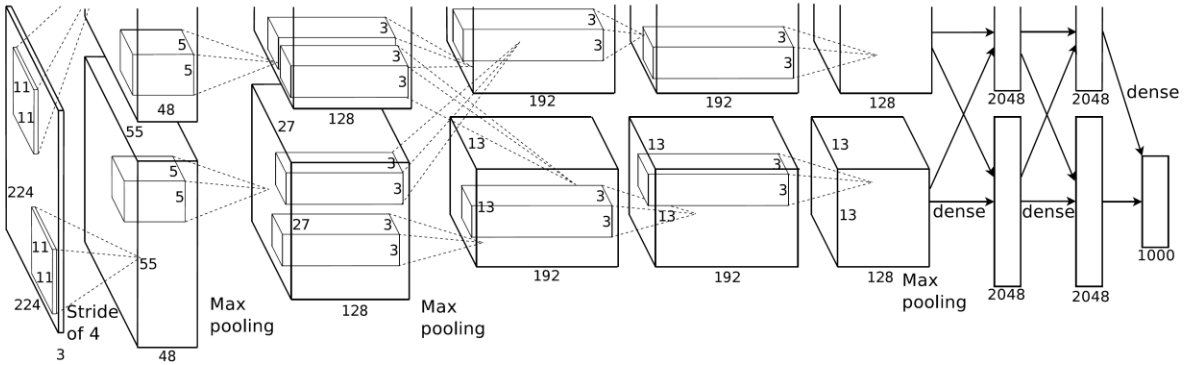


(a) Standard Neural Net (b) After applying dropout.

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

Case Study: Training AlexNet

- Batch size: 128



- Updating rule

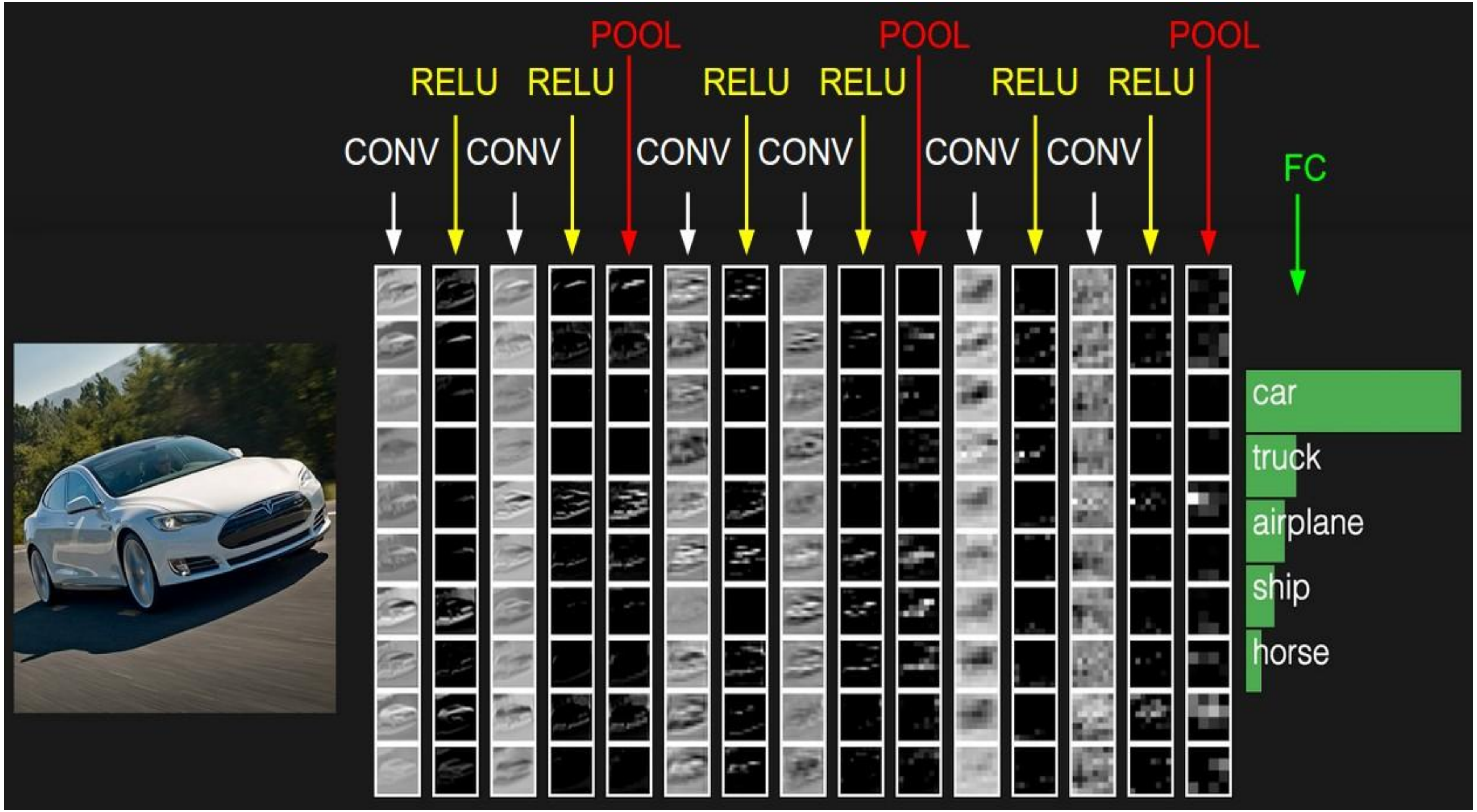
$$w_{i+1} := w_i + v_{i+1}$$

$$v_{i+1} := 0.9 \cdot v_i - 0.0005 \cdot \epsilon \cdot w_i - \epsilon \cdot \left\langle \frac{\partial L}{\partial w}\Big|_{w_i} \right\rangle_{D_i}$$

Momentum Weight Decay Learning rate Gradient

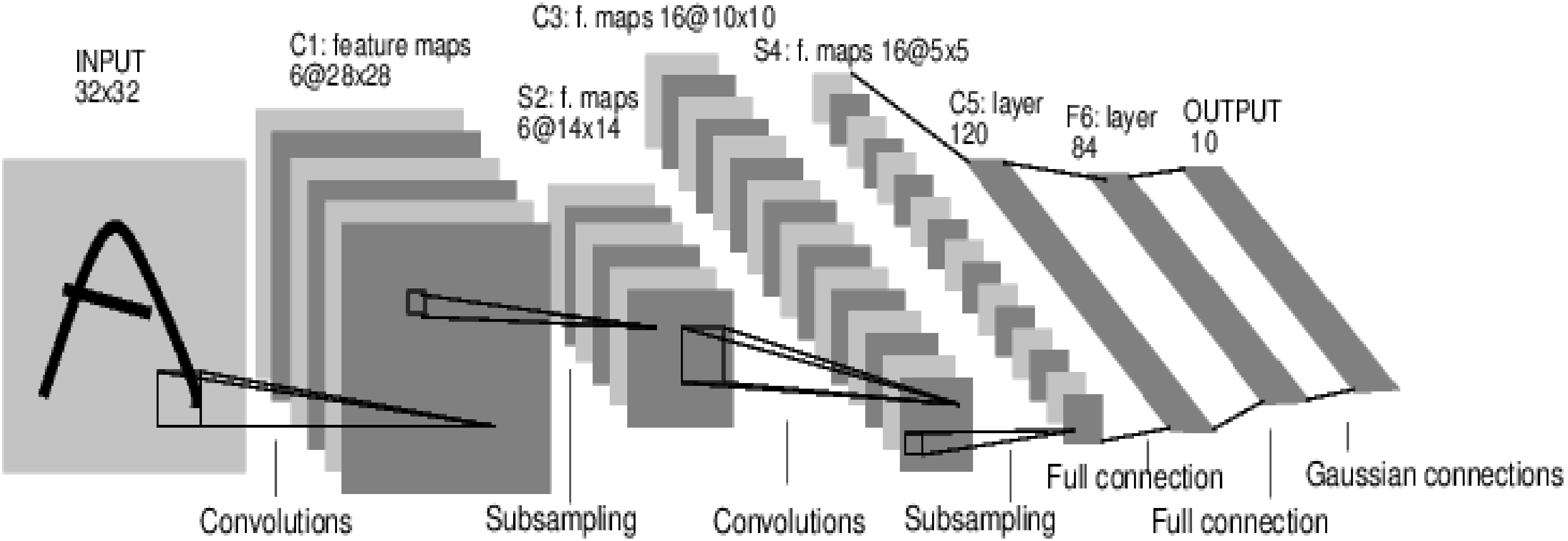
Five to six days on two NVIDIA GTX 580 3GB GPUs, 2012

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


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

Case Study: LeNet-5

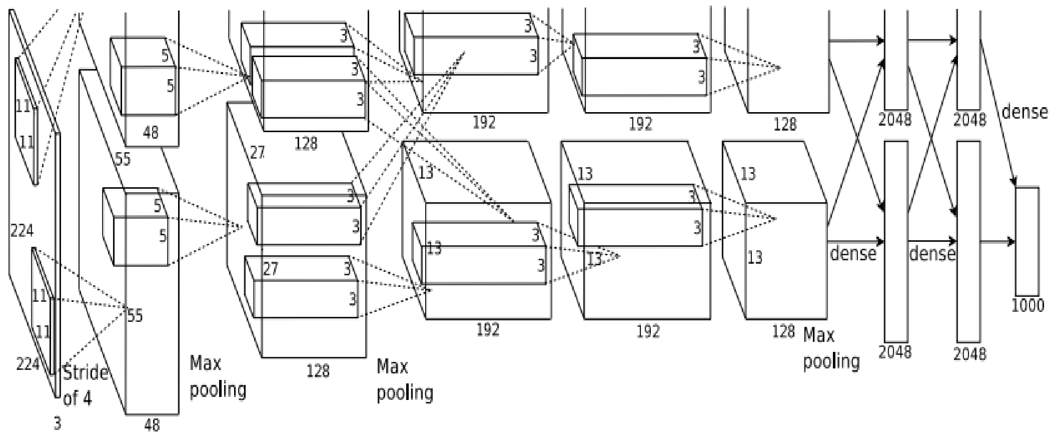
[LeCun et al., 1998]



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

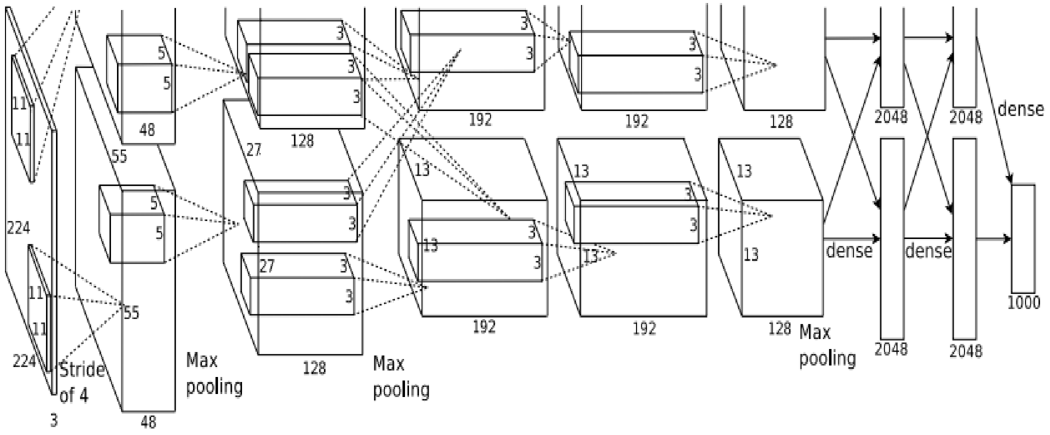
First layer (CONV1): 96 11x11 filters applied at stride 4

=>

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

Case Study: AlexNet

[Krizhevsky et al. 2012]



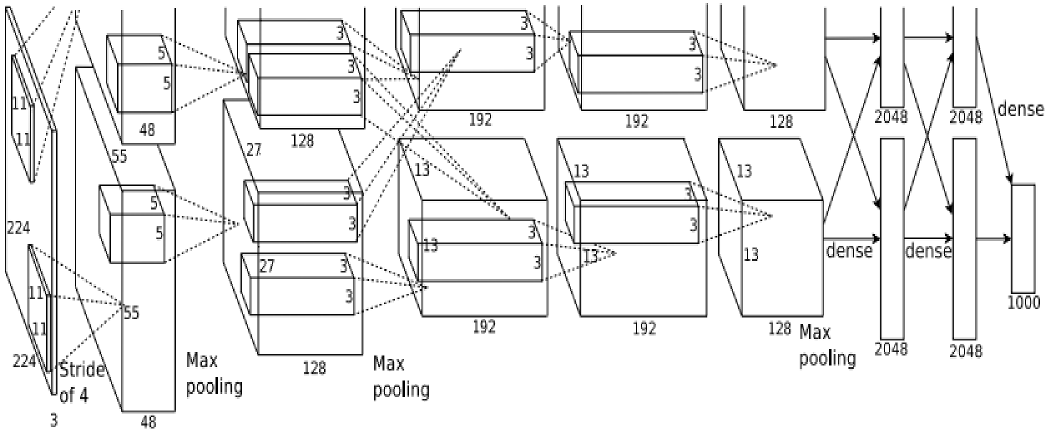
Input: 227x227x3 images

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]

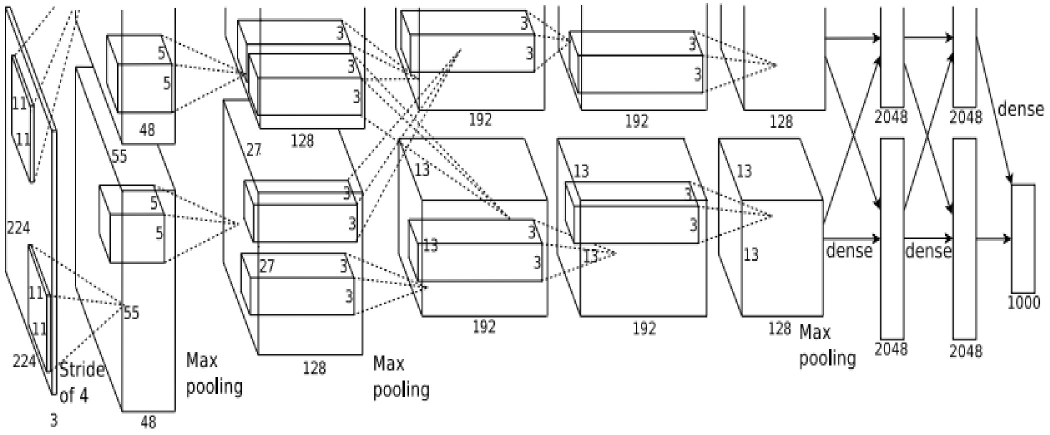


Input: 227x227x3 images

First layer (CONV1): 96 11x11 filters applied at stride 4
=>
Output volume **[55x55x96]**
Parameters: (11*11*3)*96 = **35K**

Case Study: AlexNet

[Krizhevsky et al. 2012]



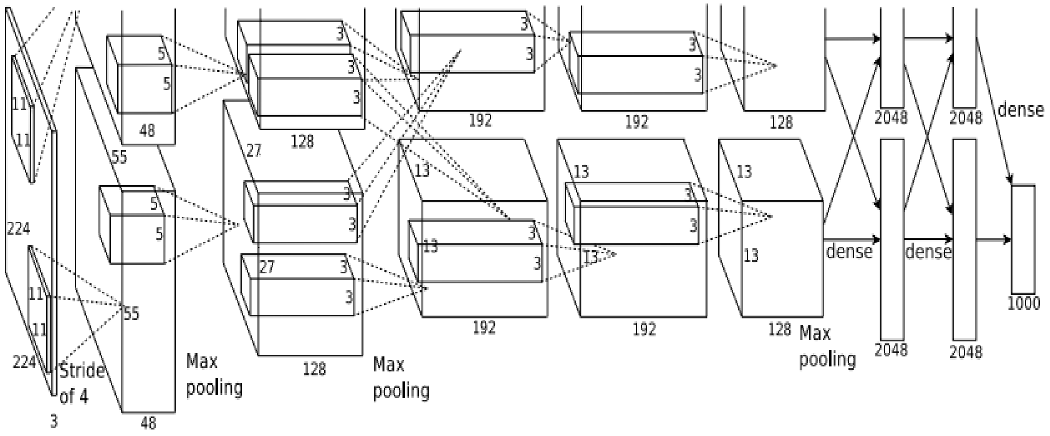
Input: 227x227x3 images
After CONV1: 55x55x96

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]



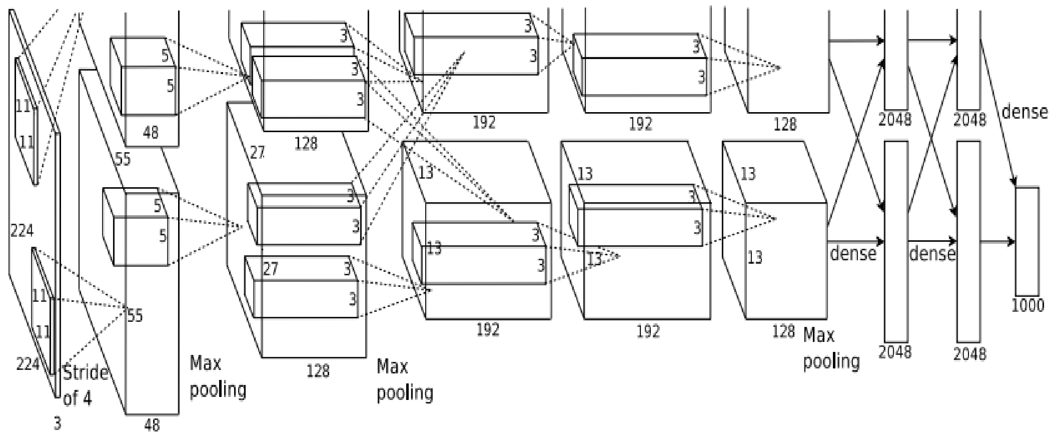
Input: 227x227x3 images
After CONV1: 55x55x96

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]

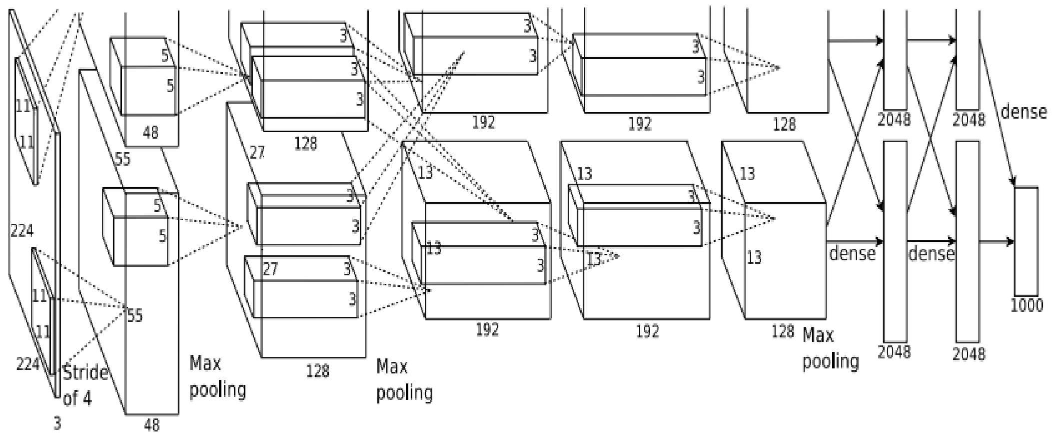


Input: 227x227x3 images
After CONV1: 55x55x96

Second layer (POOL1): 3x3 filters applied at stride 2
Output volume: 27x27x96
Parameters: 0!

Case Study: AlexNet

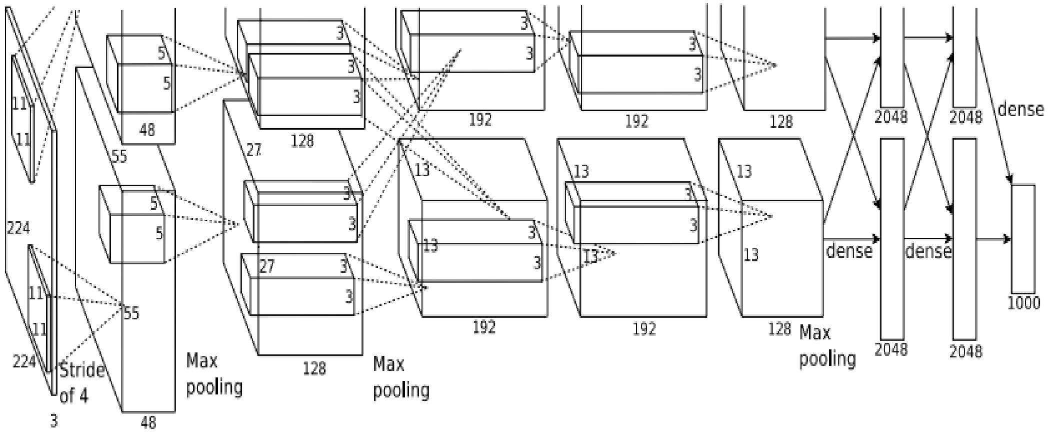
[Krizhevsky et al. 2012]



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)

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/