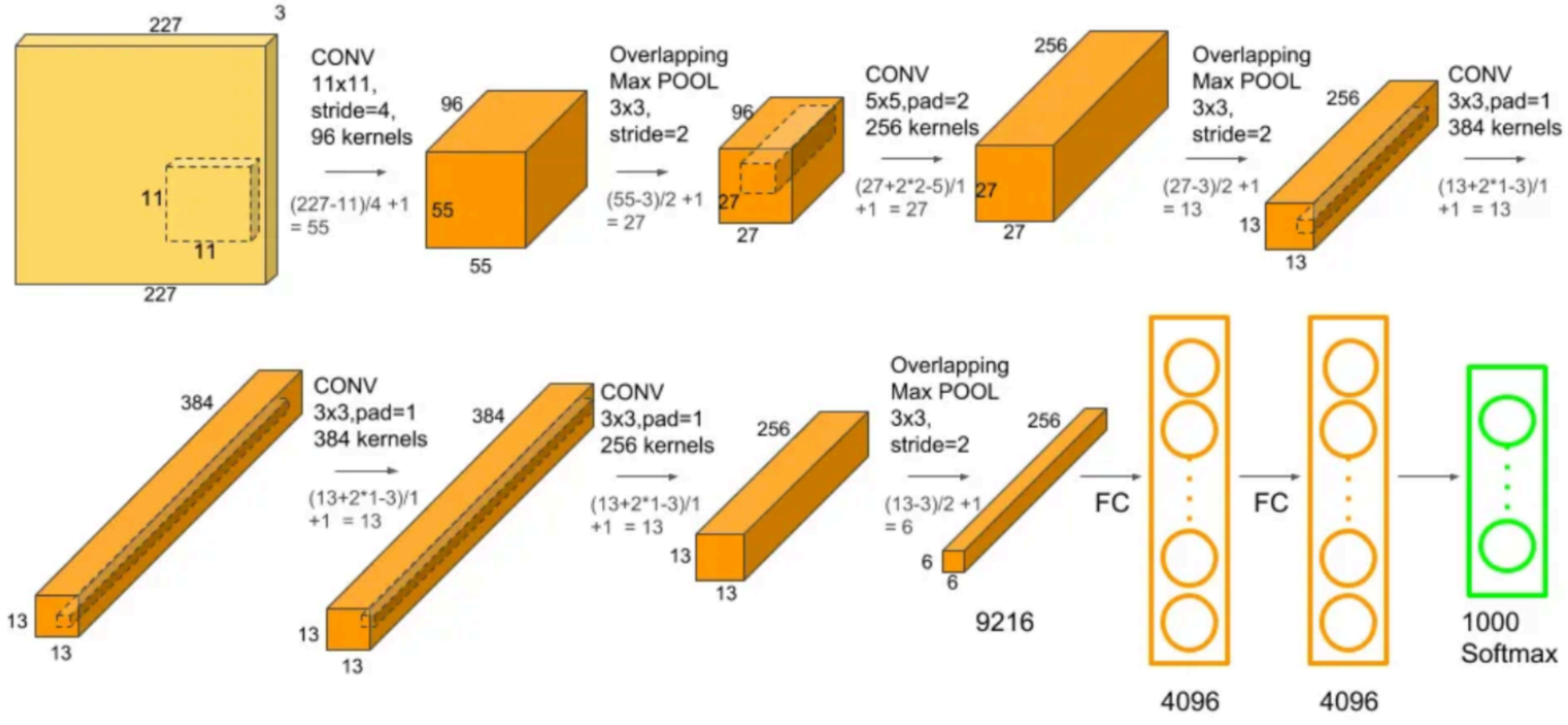
# Week2 CNN Architectures

**Tutor:**
**Email:**
**Tutorial:**
**Code:** https://github.com/Jinxu-Lin/COMP5329

# AlexNet

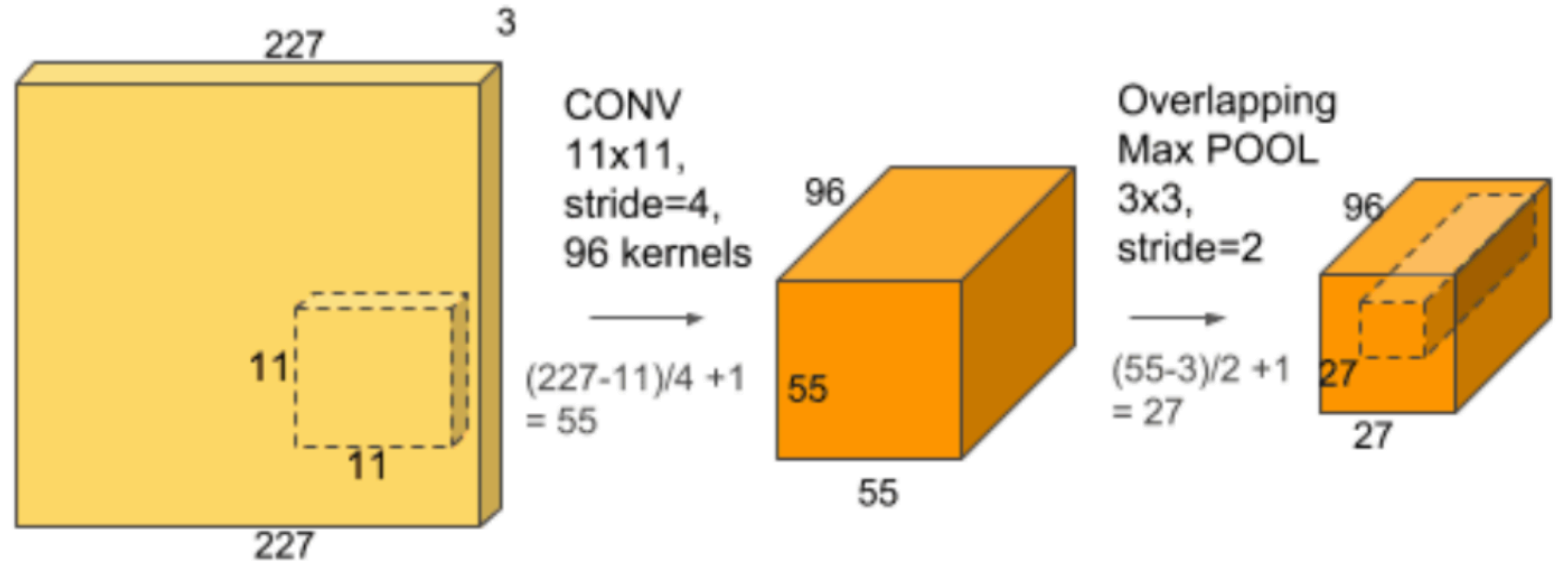
# AlexNet

- Dataset: ImageNet1K
  - $(3 \times 227 \times 227, 1000)$
- AlexNet
  - Block 1
  - Block 2
  - Block 3,4
  - Block 5
  - Linear Block



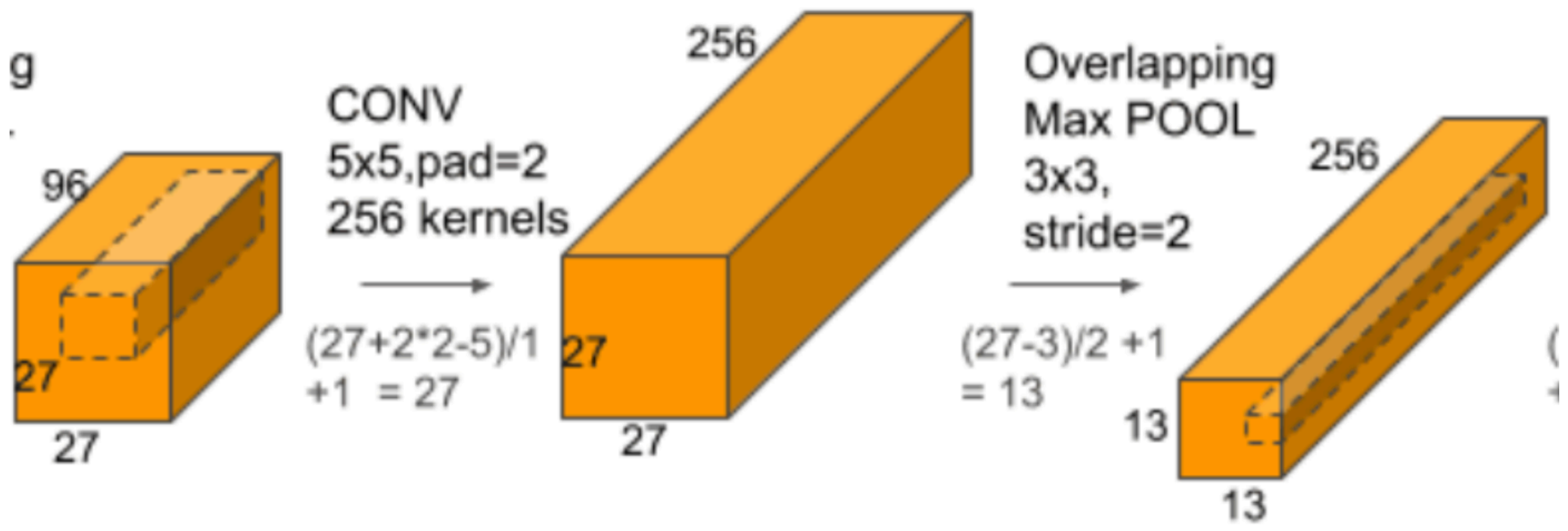
# AlexNet-Block 1

- 2d Convolution Layer: 96*(11*11), s=4, p=0
  - Input Chanels: 3; Input Size: 227*227
  - Output Shape: ?
- ReLU
- MaxPooling



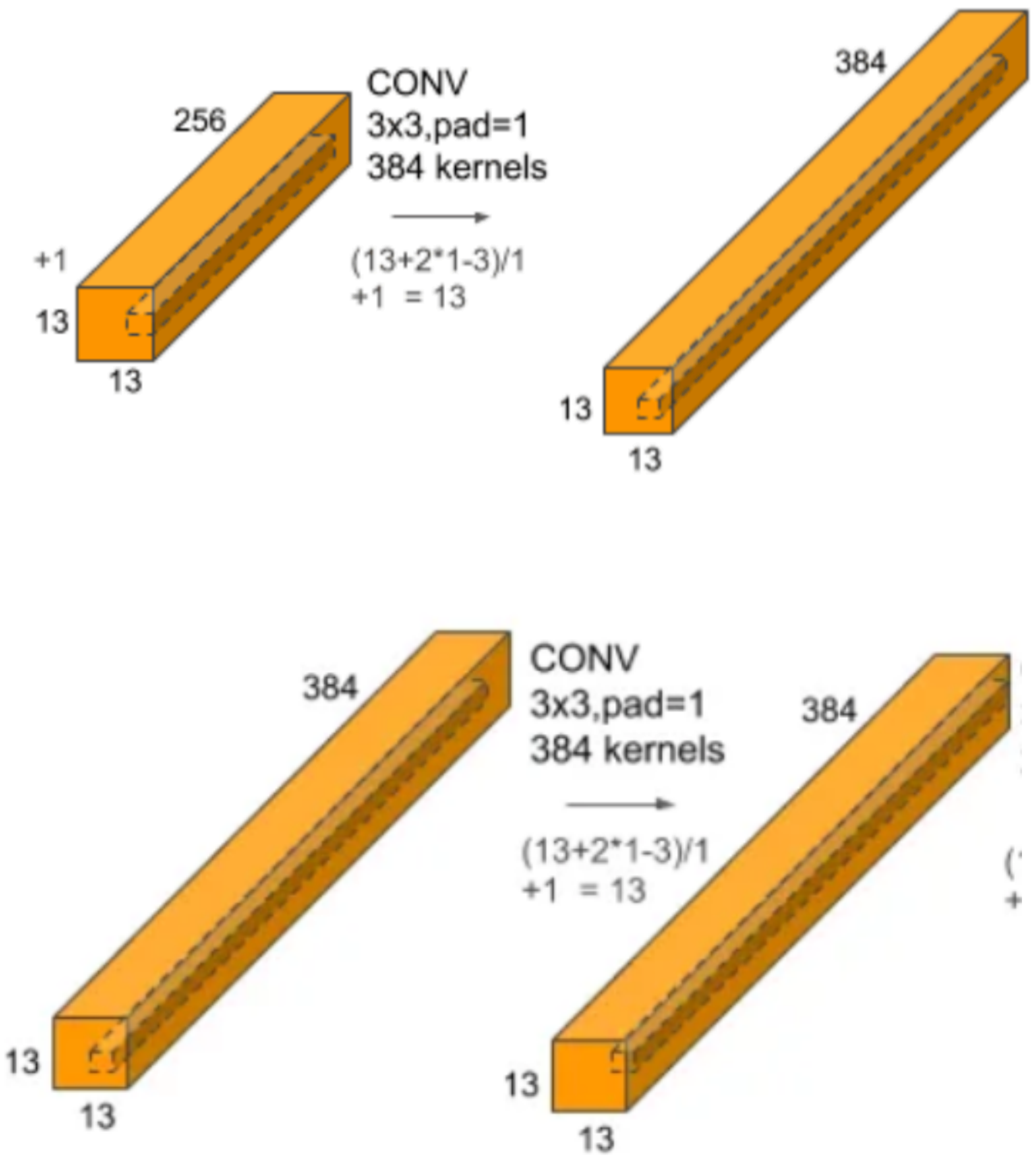
# AlexNet-Block 2

- 2d Convolution Layer: 256*(5*5), s=1, p=2
    - Input Chanels: 96; Input Size: 27*27
    - Output Shape: ?
- ReLU
- MaxPooling



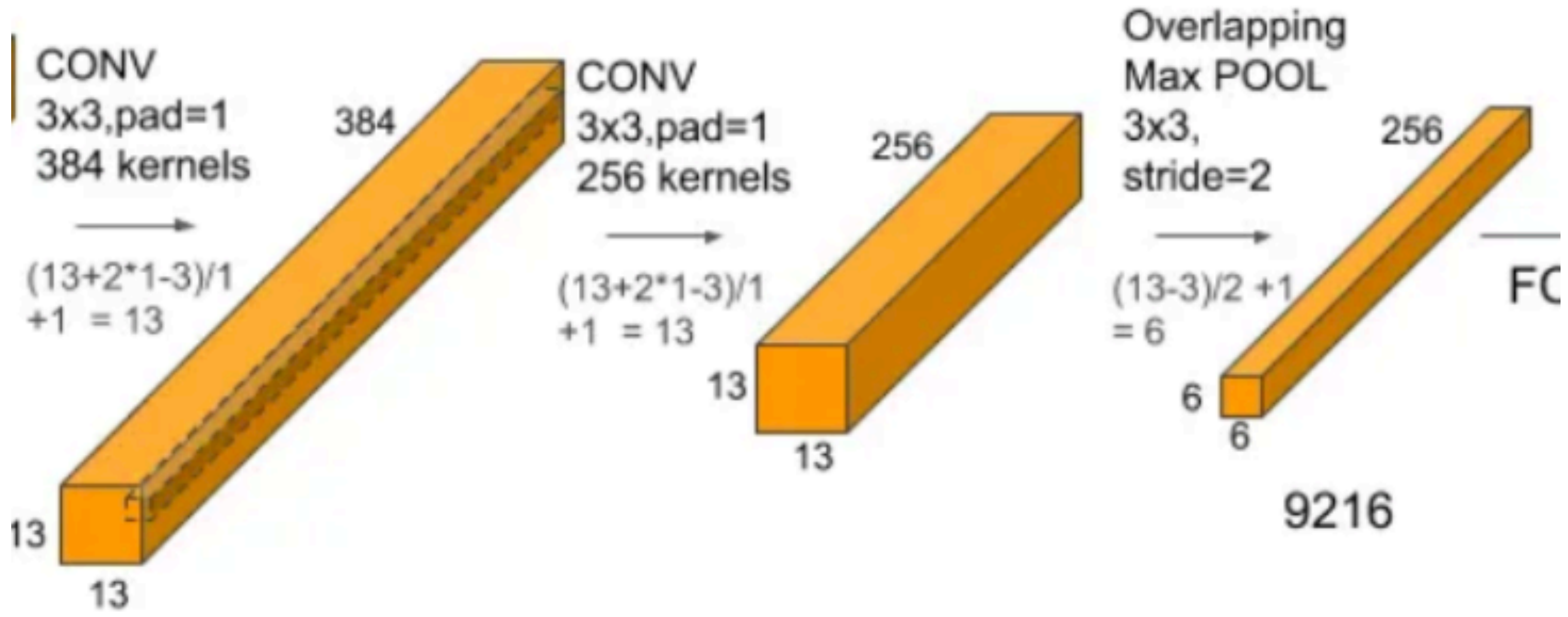
# AlexNet-Block 3,4

- 2d Convolution Layer: 384*(3*3), s=1, p=1
  - Input Chanels: 256; Input Size: 13*13
  - Output Shape: ?
- ReLU



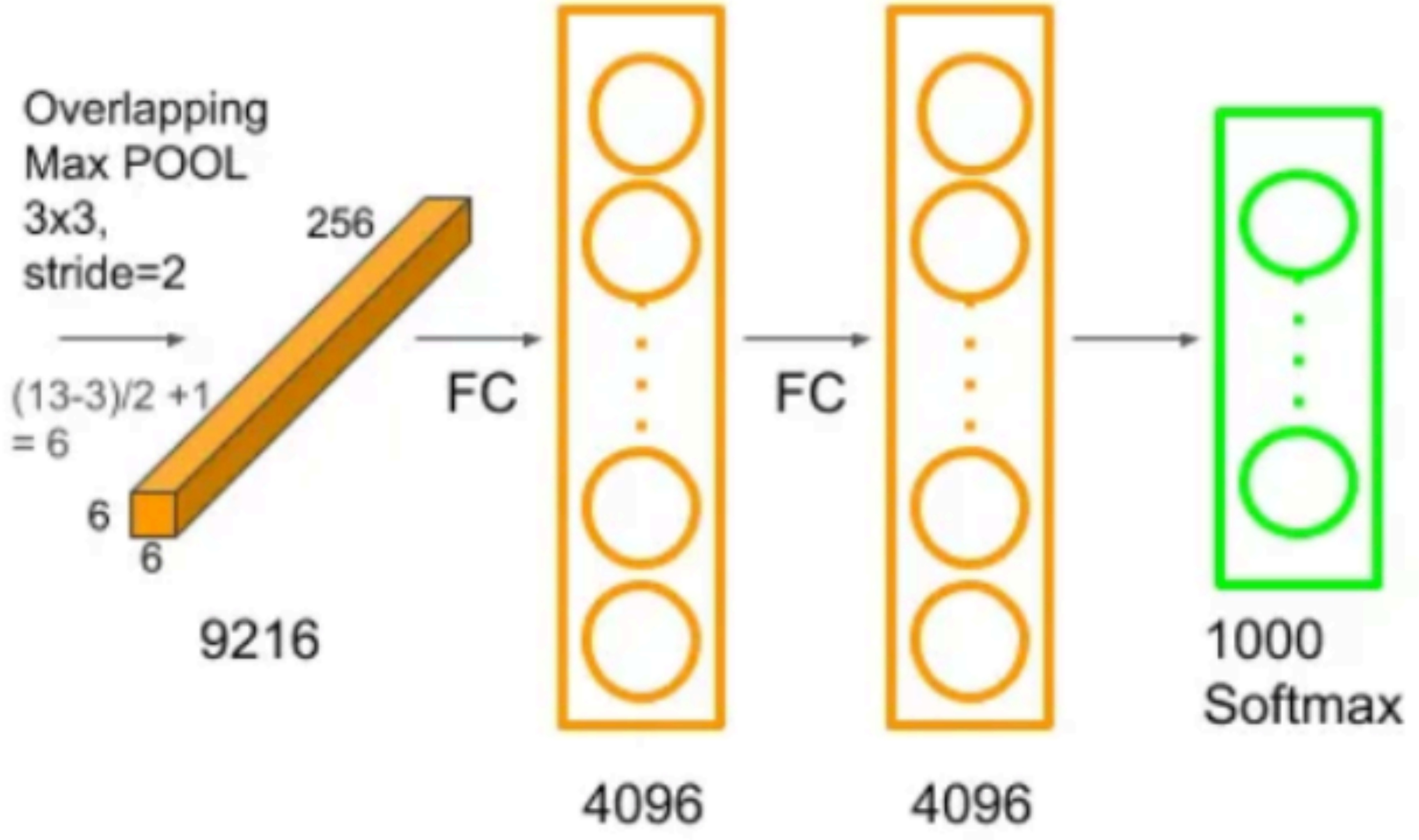
# AlexNet-Block 5

- 2d Convolution Layer: 256*(3*3), s=1, p=1
  - Input Chanels: 384; Input Size: 13*13
  - Output Shape: ?
- ReLU
- MaxPooling



# AlexNet-Full Connected Layer

- Flatten (256*6*6=9216)
- Linear: (9216,4096)
- ReLU
- Linear: (4096,1000)
- Softmax



# VGG-11

# VGG-11

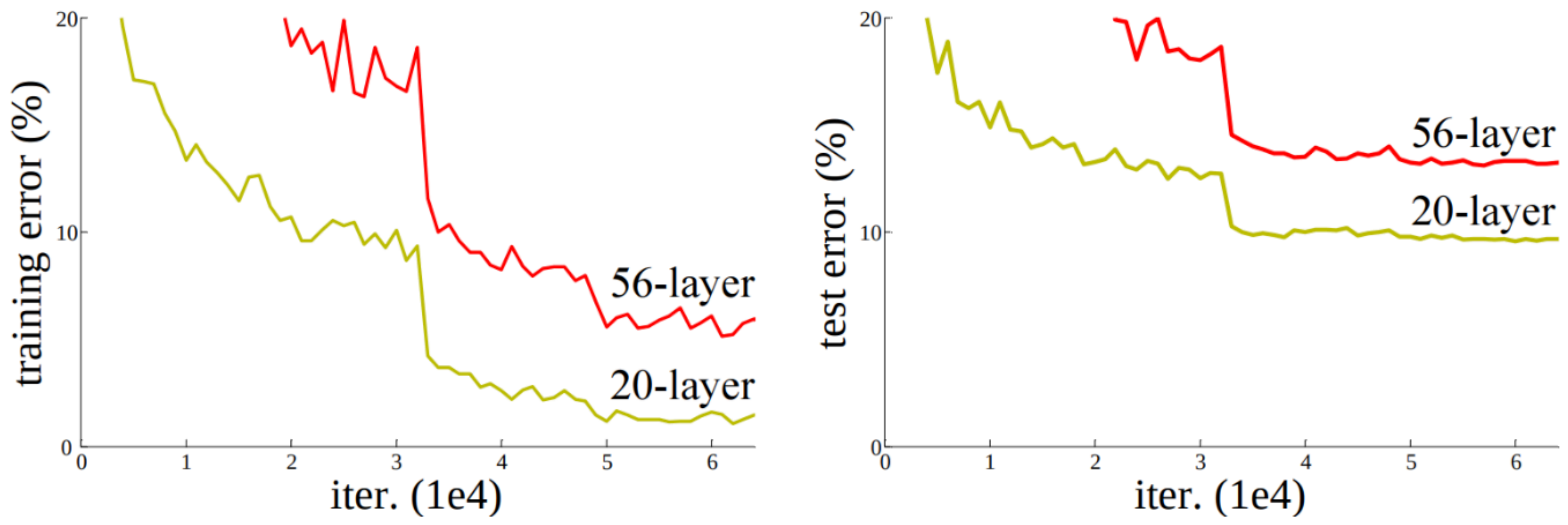
- VGG is also a popular CNN with a number of versions which are VGG-11, VGG-16 and VGG-19.
- In this class we are gonna build the VGG-11 according to the following diagram:

| VGG 11 | | | |
|---|---|---|---|
| | | Input 224 x 224 x3 | Input 32 x 32 x 3 |
| conv3-64 + ReLU | In | 224 x 224 x3 | 32 x 32 x 3 |
| | Out | 224 x 224 x64 | 32 x 32 x 64 |
| MaxPool | In | 224 x 224 x 64 | 32 x 32 x 64 |
| | Out | 112 x 112 x 64 | 16 x 16 x 64 |
| conv3-128 + ReLU | In | 112 x 112 x 64 | 16 x 16 x 64 |
| | Out | 112 x 112 x 128 | 16 x 16 x 128 |
| MaxPool | In | 112 x 112 x 128 | 16 x 16 x 128 |
| | Out | 56 x 56 x 128 | 8 x 8 x 128 |
| conv3-256 + ReLU | In | 56 x 56 x 128 | 8 x 8 x 128 |
| | Out | 56 x 56 x 256 | 8 x 8 x 256 |
| conv3-256 + ReLU | In | 56 x56 x 256 | 8 x 8 x 256 |
| | Out | 56 x 56 x 256 | 8 x 8 x 256 |
| MaxPool | In | 56 x 56 x 256 | 8 x 8 x 256 |
| | Out | 28 x 28 x 256 | 4 x 4 x 256 |
| conv3-512 + ReLU | In | 28 x 28 x 256 | 4 x 4 x 256 |
| | Out | 28 x 28 x 512 | 4 x 4 x 512 |
| conv3-512 + ReLU | In | 28 x 28 x 512 | 4 x 4 x 512 |
| | Out | 28 x 28 x 512 | 4 x 4 x 512 |
| MaxPool | In | 28 x28 x 512 | 4 x 4 x 512 |
| | Out | 14 x 14 x 512 | 2 x 2 x 512 |
| conv3-512 + ReLU | In | 14x14 x 512 | 2 x 2 x 512 |
| | Out | 14 x 14 x 512 | 2 x 2 x 512 |
| conv3-512 + ReLU | In | 14x14 x 512 | 2 x 2 x 512 |
| | Out | 14 x 14 x 512 | 2 x 2 x 512 |
| MaxPool | In | 14 x 14 x 512 | 2 x 2 x 512 |
| | Out | 7 x 7 x 512 | 1 x 1 x 512 |
| FC | In | 25088 | 512 |
| | Out | 4096 | 2048 |
| FC | In | 4096 | 2048 |
| | Out | 4096 | 2048 |
| FC | In | 4096 | 2048 |
| | Out | 1000 | 10 |
| SoftMax | | | |

# ResNet

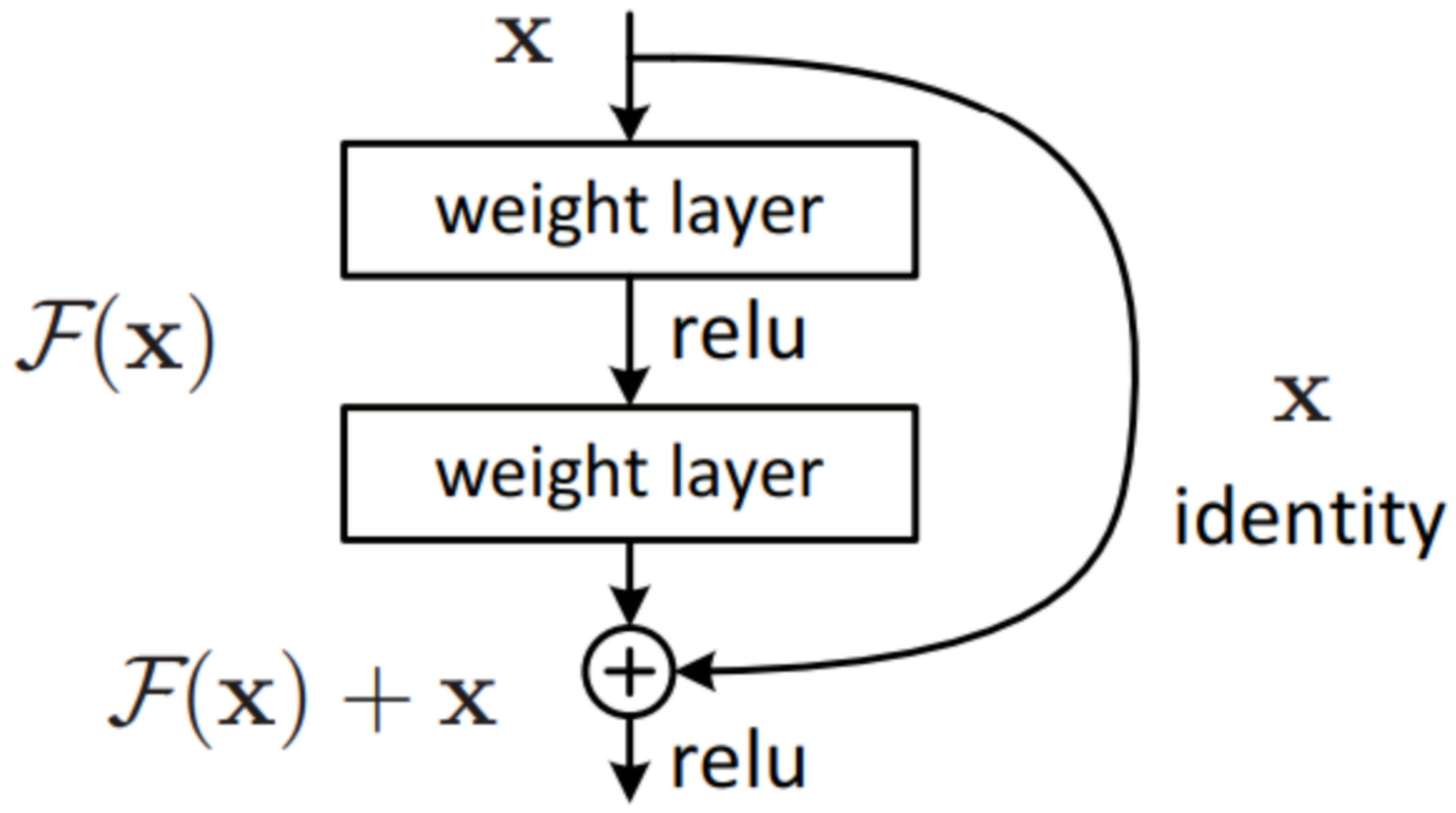
# ResNet

- Existing Problems
  - People all realize that deeper network means better results. However, it is not the whole story
- Reason
  - **Gradient vanishing**: gradient disappeared at very early layer, while very strong in the last layers.
  - **Saturation**: (Observed in the Resnet paper) Stacking more layers without gradient vanishing also have gradient vanishing problem.



# ResNet

- Skip Connection
- Intuition
  - If $F(x)$ is a saturated factor, $F(x)$ is optimized to 0. This will reduce the depth of the network. (saturation problem)
  - Strengthen signal from $x$. Which helps the gradient signals toward $x$ is stronger. (gradient vanishing)
- Means:
  - Reduce the depth of the network if saturation
  - Reduce gradient vanishing



# 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 | | $1.8\times10^9$ | $3.6\times10^9$ | $3.8\times10^9$ | $7.6\times10^9$ | $11.3\times10^9$ |

# Exam-Style Questions

# Code

# Code

- ./materials/Week6_CNN_Architectures/Week6_CNN_Architectures.ipynb
- ./ResNet/Models/resnet.py
- ./ResNet/train_utils.py: line32
- ./ResNet/train.py: line203