Lecture 6 Convolutional Neural Networks

Lei Li and Yuxiang Wang
UCSB

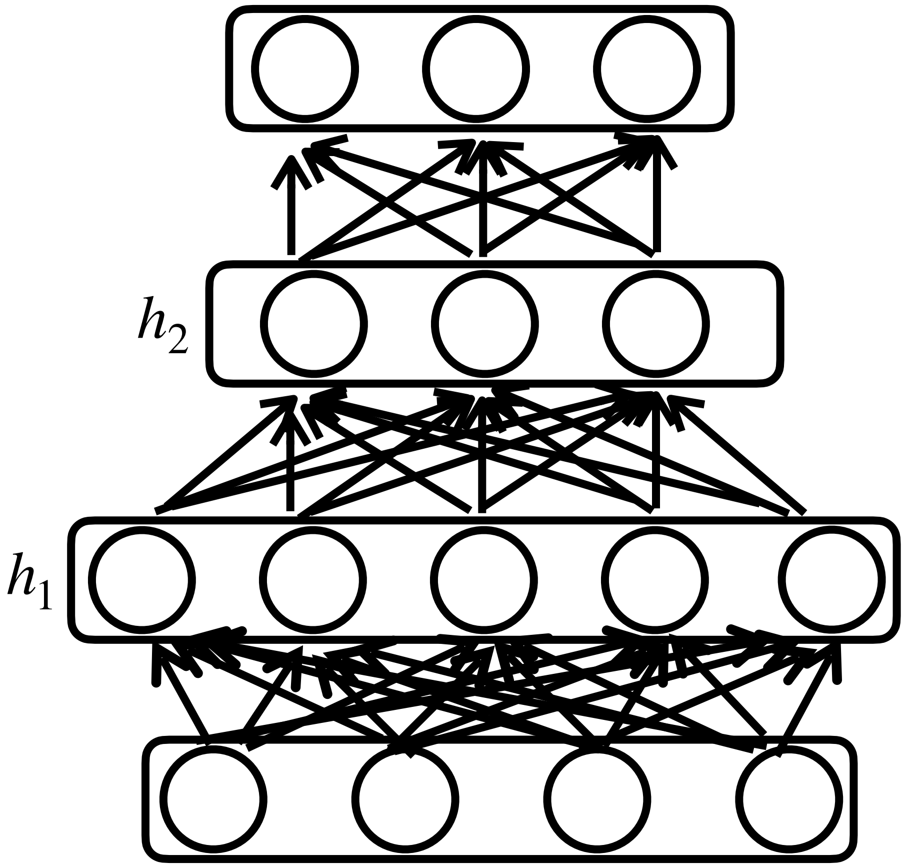
Acknowledgement: Slides borrowed from Bhiksha Raj's 11485 and Mu Li & Alex Smola's 157 courses on Deep Learning, with modification

Recap

- Single artificial neuron to mimic biological neurons
  - each with simple operations
- Logistic Regression and its limitation
- Feedforward neural network (multilayer perceptron)
  - Massive combination of simple units
- Successful example of FFN
  - Deep&Wide model for recommendation system
- Computing Gradient for FFN — backpropagation

Feedforward Neural Net (FFN)

- also known as multilayer perceptron (MLP)
- Layers are connected sequentially
- Each layer has full-connection (each unit is connected to all units of next layer)
  - Linear project followed by
  - an element-wise nonlinear activation function
- There is no connection from output to input



Learning FFN: Stochastic Gradient Descent

```
learning rate eta.
1. set initial parameter θ ← θ_0
2. for epoch = 1 to maxEpoch or until converge:
3.   random_shuffle data
4.   for each data batch (x, y):
5.     compute error err(f(x; θ) - y) using forward
6.     compute gradient g = ∂err(θ)/∂θ using backpropagation
7.     total_g += g
8.   update θ = θ - eta * total_g / batch_size
```

Forward "Pass"

- Input: D dimensional vector $\mathbf{x} = [x_j,\ \ j = 1 \ldots D]$
- Set:
  - $D_0 = D,$ is the width of the 0th (input) layer
  - $y_j^{(0)} = x_j,\ \ j = 1 \ldots D;\qquad y_0^{(k=1\ldots N)} = x_0 = 1$
- For layer $k = 1 \ldots N$
  - For $j = 1 \ldots D_k$

    D_k is the size of the kth layer

    - $z_j^{(k)} = \sum_{i=0}^{D_{k-1}} w_{i,j}^{(k)} y_i^{(k-1)}$
    - $y_j^{(k)} = f_k\left(z_j^{(k)}\right)$
- Output:
  - $Y = y_j^{(N)},\ j = 1..D_N$

Backward Pass

- Output layer (N) :
  - For $i = 1 \dots D_N$
    - $$\frac{\partial \ell}{\partial z_i^{(N)}} = f_N'(z_i^{(N)}) \frac{\partial \ell}{\partial \hat{y}_i^{(N)}}$$
    - $$\frac{\partial \ell}{\partial w_{ij}^{(N)}} = y_i^{(N-1)} \frac{\partial \ell}{\partial z_j^{(N)}} \text{ for each j}$$

Called "Backpropagation" because the derivative of the loss is propagated "backwards" through the network

- For layer $k = N-1$ *downto*

Very analogous to the forward pass:

  - For $i = 1 \dots D_k$
    - $$\frac{\partial \ell}{\partial y_i^{(k-1)}} = \sum_j w_{ij}^{(k)} \frac{\partial \ell}{\partial z_j^{(k)}}$$
    - $$\frac{\partial \ell}{\partial z_i^{(k)}} = f_k'(z_i^{(k)}) \frac{\partial \ell}{\partial y_i^{(k)}}$$
    - $$\frac{\partial \ell}{\partial w_{ij}^{(k)}} = y_i^{(k-1)} \frac{\partial \ell}{\partial z_j^{(k)}} \text{ for each j}$$

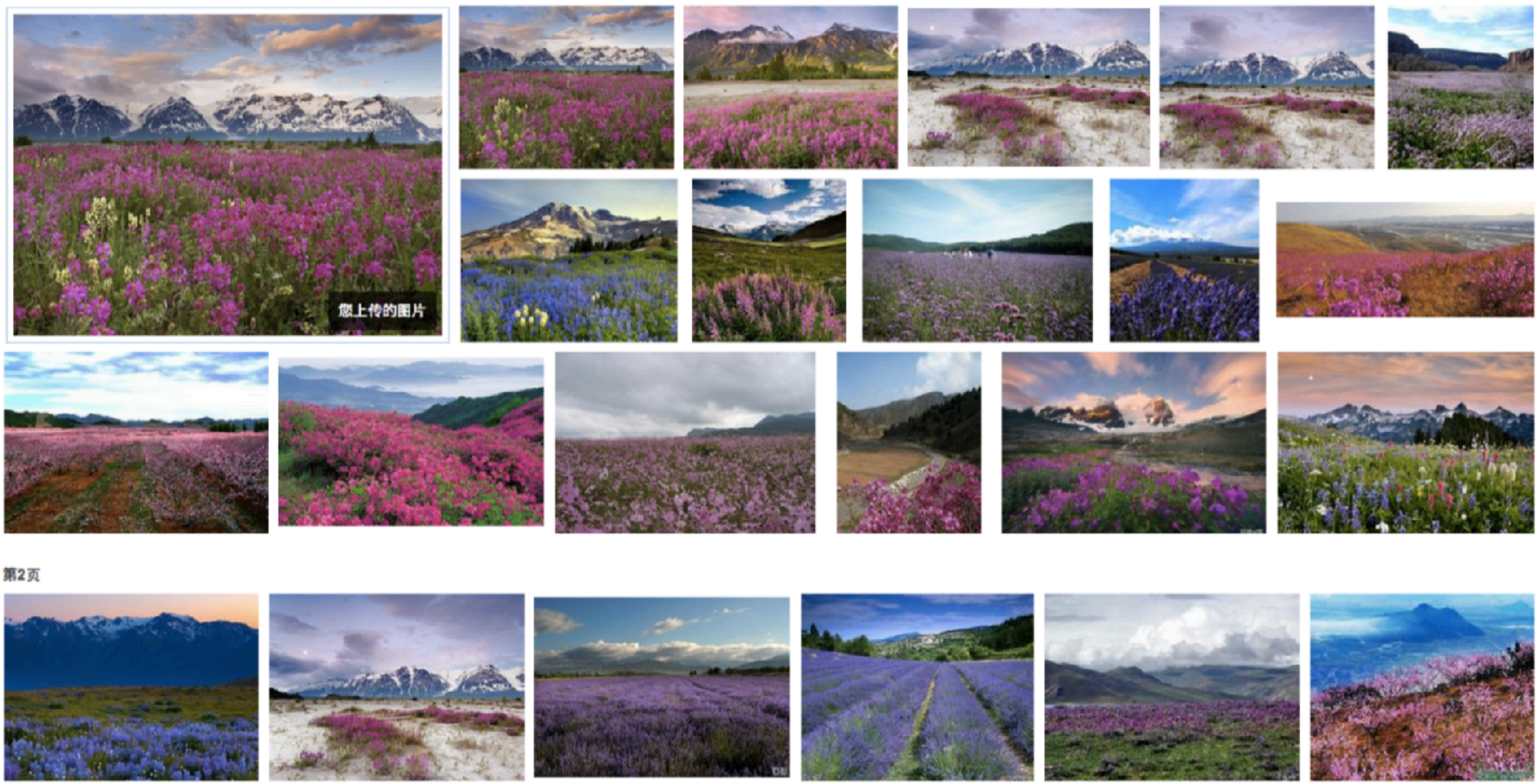
Backward weighted combination of next layer

Backward equivalent of activation

Why Learning CNN?

- A fundamental class of models for image recognition
- Vast applications:
  - Autonomous driving vehicle
  - Image search
  - E-commerce recommendation
  - Face identification (iphone faceID)

Visual Search



Answering question about image

Q: what is the color of the bus?
A: yellow

Q: what are there hanging up?
A: umbrellas

Q: What is the color of the cake?
A: red

ABC-CNN
[Chen, Wang et al 2015]

Autonomous Driving in 2015

Convolution

Problem: Classifying Dog and Cat Images

- Use a good camera
- RGB image has 36M elements
- What is the size of a FFN with a single hidden layer (100 hidden units)?
- How to reduce parameter size?

Where
is
Waldo?

Two Principles

- Translation Invariance
- Locality

Full Projection in Tensor Form

- Input image: a matrix with size (h, w)
- Projection weights: a 4-D tensors (h,w) by (h',w')

$$h_{i,j} = \sum_{k,l} w_{i,j,k,l} x_{k,l} = \sum_{a,b} v_{i,j,a,b} x_{i+a,j+b}$$

V is re-indexes W such as that $v_{i,j,a,b} = w_{i,j,i+a,j+b}$

Tensor is a generalization of matrix

Idea #1 - Translation Invariance

$$h_{i,j} = \sum_{a,b} v_{i,j,a,b} x_{i+a,j+b}$$

- A shift in *x* also leads to a shift in *h*
- *v* should not depend on *(i,j)*. Fix via

$$v_{i,j,a,b} = v_{a,b}$$

$$h_{i,j} = \sum_{a,b} v_{a,b} x_{i+a,j+b}$$

Idea #2 - Locality

$$h_{i,j} = \sum_{a,b} v_{a,b} x_{i+a,j+b}$$

- We shouldn't look very far from x(i,j) in order to assess what's going on at h(i,j)
- Outside range $|a|,|b| > \Delta$ parameters vanish $v_{a,b} = 0$

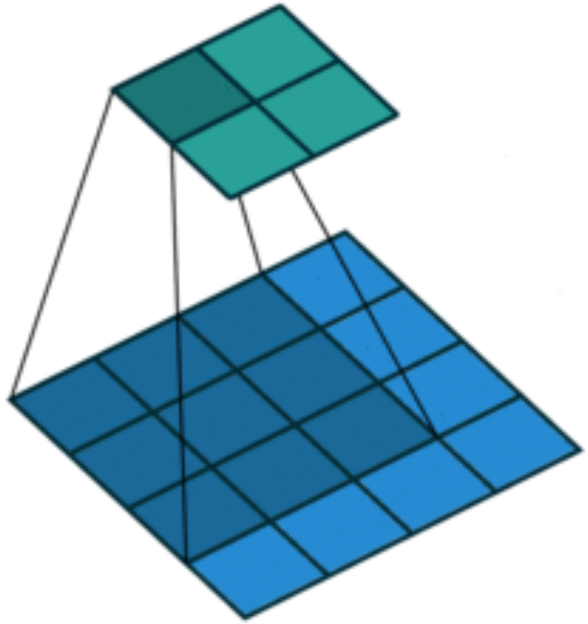
$$h_{i,j} = \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} v_{a,b} x_{i+a,j+b}$$

2-D Convolution Layer

- input matrix $\mathbf{X} : n_h \times n_w$
- kernel matrix $\mathbf{W} : k_h \times k_w$
- b: scalar bias
- output matrix
  $\mathbf{Y} : (n_h - k_h + 1) \times (n_w - k_w + 1)$
  $\mathbf{Y} = \mathbf{X} \star \mathbf{W} + b$

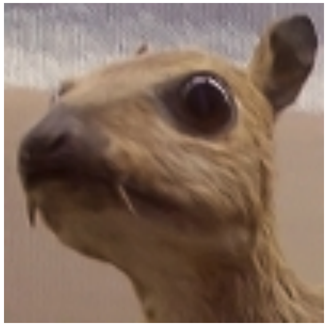
$$y_{i,j} = \sum_{a=1}^{h} \sum_{b=1}^{w} w_{a,b} x_{i+a,j+b}$$

- **W** and *b* are learnable parameters



$$\begin{bmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{bmatrix} * \begin{bmatrix} 0 & 1 \\ 2 & 3 \end{bmatrix} = \begin{bmatrix} 19 & 25 \\ 37 & 43 \end{bmatrix}$$

Examples

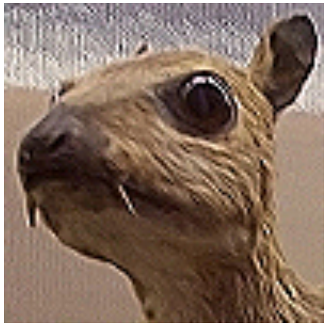


(wikipedia)

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Edge Detection

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$



Sharpen

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Gaussian Blur

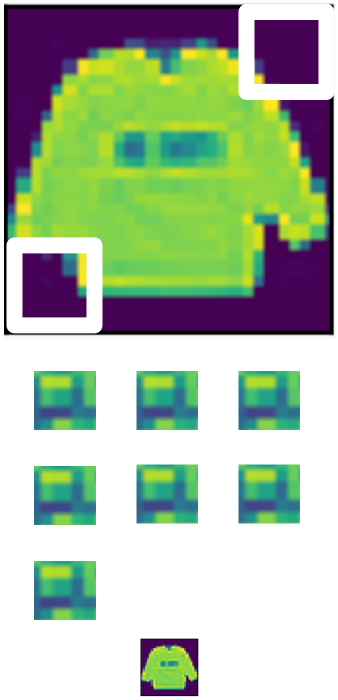
Examples

(Rob Fergus)

Padding and Stride

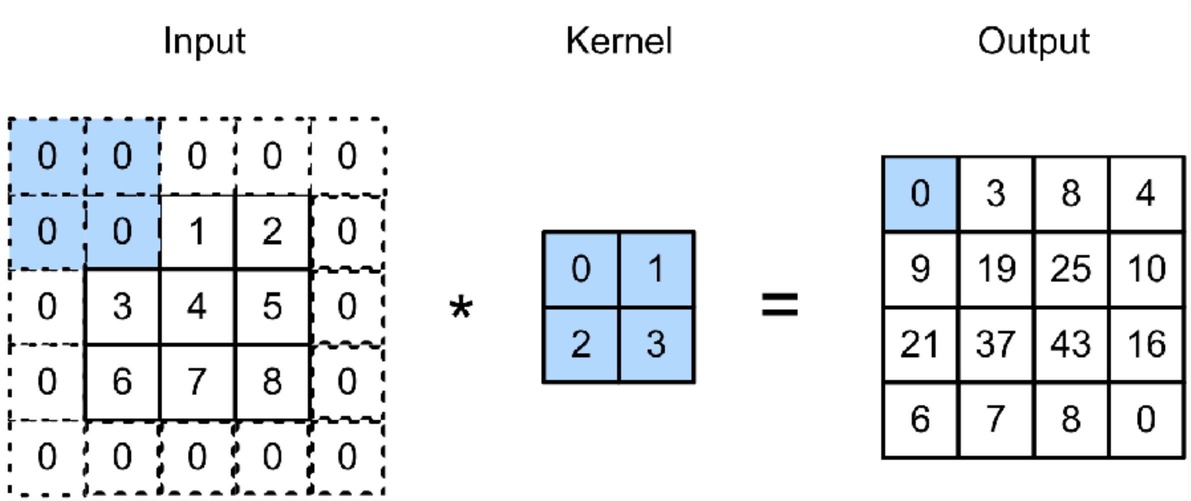
Padding

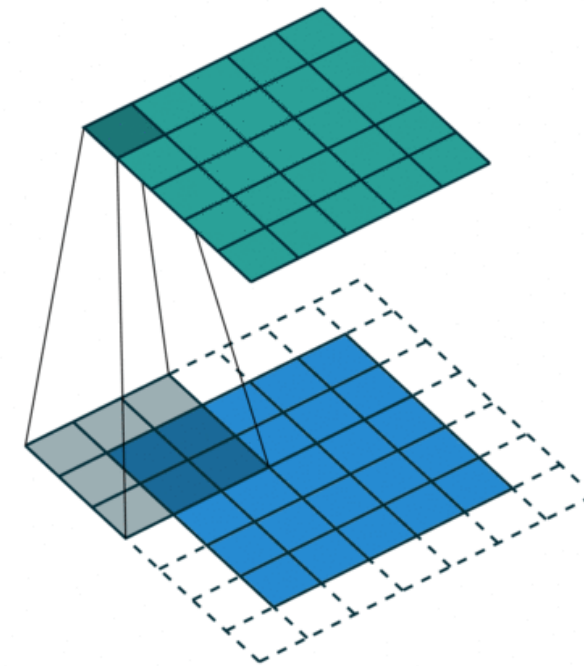
- Given a 32 x 32 input image
- Apply convolutional layer with 5 x 5 kernel
  - 28 x 28 output with 1 layer
  - 4 x 4 output with 7 layers
- Shape decreases faster with larger kernels
  - Shape reduces from $n_h \times n_w$ to $(n_h - k_h + 1) \times (n_w - k_w + 1)$



Padding

Padding adds rows/columns around input





$$0 \times 0 + 0 \times 1 + 0 \times 2 + 0 \times 3 = 0$$

Padding

- Padding p_h rows and p_w columns, output shape will be
  $$(n_h - k_h + p_h + 1) \times (n_w - k_w + p_w + 1)$$
- A common choice is $p_h = k_h - 1$ and $p_w = k_w - 1$
  - Odd k_h : pad $p_h/2$ on both sides
  - Even k_h : pad $\lceil p_h/2 \rceil$ on top, $\lfloor p_h/2 \rfloor$ on bottom

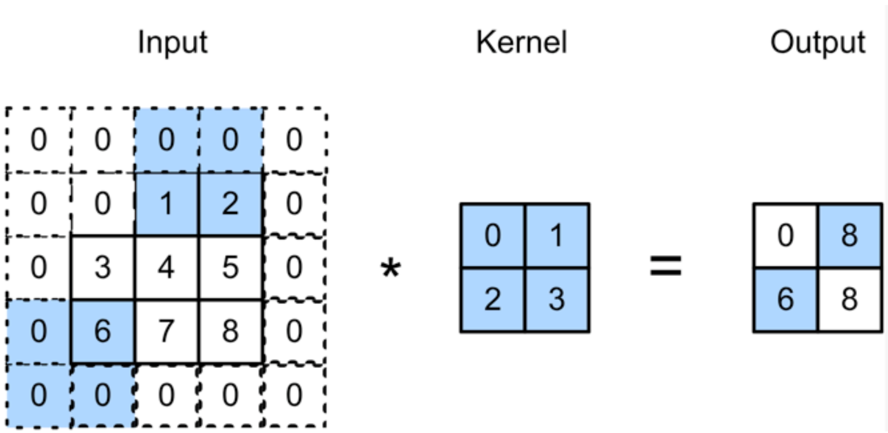
Stride

- Padding reduces shape linearly with #layers
  - Given a 224 x 224 input with a 5 x 5 kernel, needs 44 layers to reduce the shape to 4 x 4
  - Requires a large amount of computation

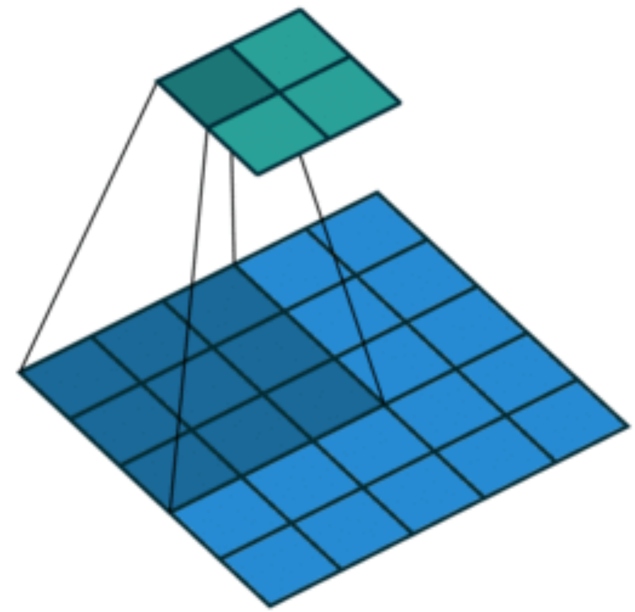
Stride

- Stride is the #rows/#column

Strides of 3 and 2 for height and width



$$0 \times 0 + 0 \times 1 + 1 \times 2 + 2 \times 3 = 8$$
$$0 \times 0 + 6 \times 1 + 0 \times 2 + 0 \times 3 = 6$$



Stride

- Given stride s_h for the height and stride s_w for the width,
  the output shape is

$$\lfloor (n_h - k_h + p_h + s_h)/s_h \rfloor \times \lfloor (n_w - k_w + p_w + s_w)/s_w \rfloor$$

- With $p_h = k_h - 1$ and $p_w = k_w - 1$

$$\lfloor (n_h + s_h - 1)/s_h \rfloor \times \lfloor (n_w + s_w - 1)/s_w \rfloor$$

- If input height/width are divisible by strides

$$(n_h/s_h) \times (n_w/s_w)$$


Multiple Channels

Multiple Input Channels

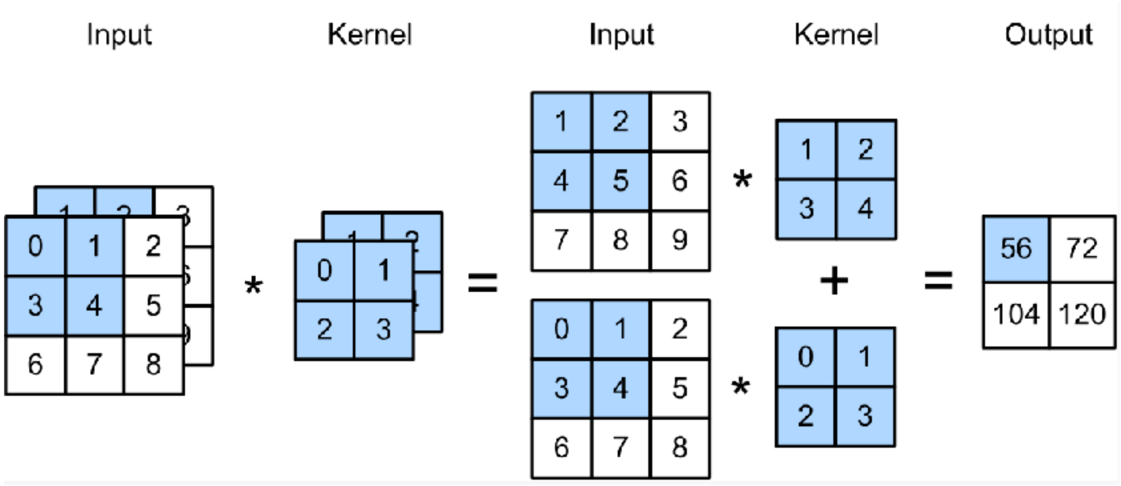
- Color image may have three RGB channels
- Converting to grayscale loses information

Multiple Input Channels

- Color image may have three RGB channels
- Converting to grayscale loses information

Multiple Input Channels

- Input is a tensor
- Have a kernel for each channel, and then sum results over channels



$$(1\times 1+2\times 2+4\times 3+5\times 4)$$
$$+(0\times 0+1\times 1+3\times 2+4\times 3)$$
$$=56$$

Multiple Input Channels

- $\mathbf{X} : c_i \times n_h \times n_w$ input tensor
- $\mathbf{W} : c_i \times k_h \times k_w$ kernel tensor
- $\mathbf{Y} : m_h \times m_w$ output

$$\mathbf{Y} = \sum_{i=0}^{c_i} \mathbf{X}_{i,:,:} \star \mathbf{W}_{i,:,:}$$

Multiple Output Channels

- No matter how many inputs channels, so far we always get single output channel
- We can have multiple 3-D kernels, each one generates a output channel
- Input $\mathbf{X} : c_i \times n_h \times n_w$
- Kernel $\mathbf{W} : c_o \times c_i \times k_h \times k_w$
- Output $\mathbf{Y} : c_o \times m_h \times m_w$

$$\mathbf{Y}_{i,:,:} = \mathbf{X} \star \mathbf{W}_{i,:,:,:}$$

for $i = 1, \ldots, c_o$

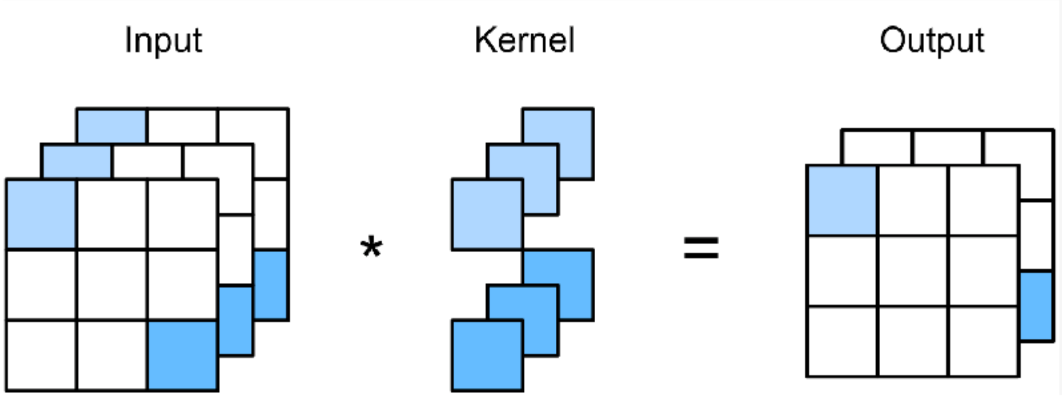
Multiple Input/Output Channels

- Each output channel may recognize a particular pattern

- Input channels kernels recognize and combines patterns in inputs

1 x 1 Convolutional Layer

$k_h = k_w = 1$ is a popular choice. It doesn't recognize spatial patterns, but fuse channels.



Equal to a dense layer with $n_h n_w \times c_i$ input and $c_o \times c_i$ weight.

2-D Convolution Layer Summary

- Input $\mathbf{X} : c_i \times n_h \times n_w$
- Kernel $\mathbf{W} : c_o \times c_i \times k_h \times k_w$
- Bias $\mathbf{B} : c_o$
- Output $\mathbf{Y} : c_o \times m_h \times m_w$
- Complexity (number of floating point operations FLOP)

$$\mathbf{Y} = \mathbf{X} \star \mathbf{W} + \mathbf{B}$$

$$c_i = c_o = 100$$
$$k_h = h_w = 5$$
$$m_h = m_w = 64$$

$$O(c_i c_o k_h k_w m_h m_w)$$

1GFLOP

- 10 layers, 1M examples: 10PF
(CPU: 0.15 TF = 18h, GPU: 12 TF = 14min)

Pooling Layer

Pooling

- Convolution is sensitive to position
  - Detect vertical edges

0 output with 1

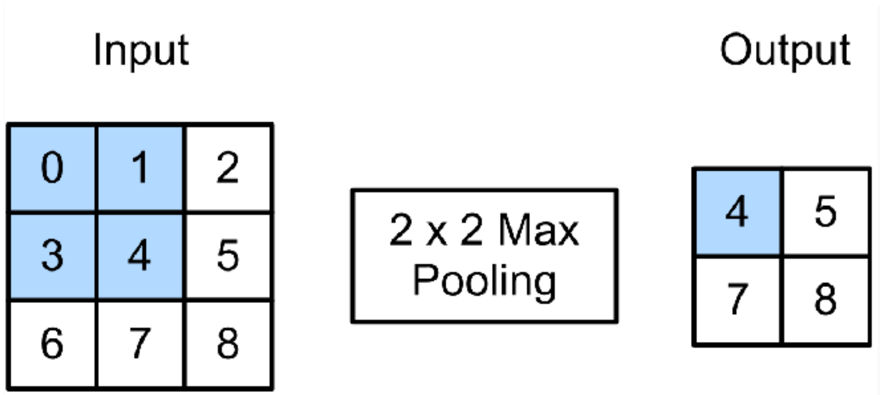
X
```
[[1.  1.  0.  0.  0.
 [1.  1.  0.  0.  0.
 [1.  1.  0.  0.  0.
 [1.  1.  0.  0.  0.
```

Y
```
[[ 0.   1.   0.   0.
 [ 0.   1.   0.   0.
 [ 0.   1.   0.   0.
 [ 0.   1.   0.   0.
```

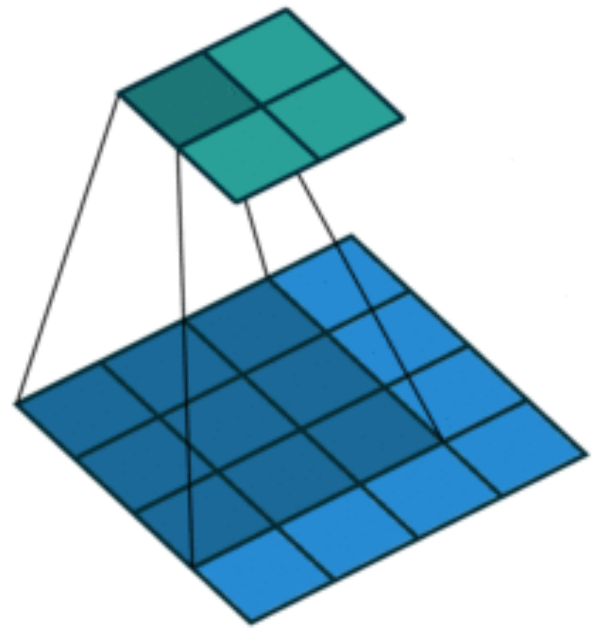
- We need some degree of invariance to translation
  - Lighting, object positions, scales, appearance vary among images

2-D Max Pooling

- Returns the maximal value in the sliding window



$$\max(0,1,3,4) = 4$$



2-D Max Pooling

- Returns the maximal value in the sliding window

Vertical edge detection

```
[[1. 1. 0. 0. 0.
 [1. 1. 0. 0. 0.
 [1. 1. 0. 0. 0.
 [1. 1. 0. 0. 0.
```

Conv output

```
[[ 0.   1.   0.   0.
 [ 0.   1.   0.   0.
 [ 0.   1.   0.   0.
 [ 0.   1.   0.   0.
```

2 x 2 max pooling

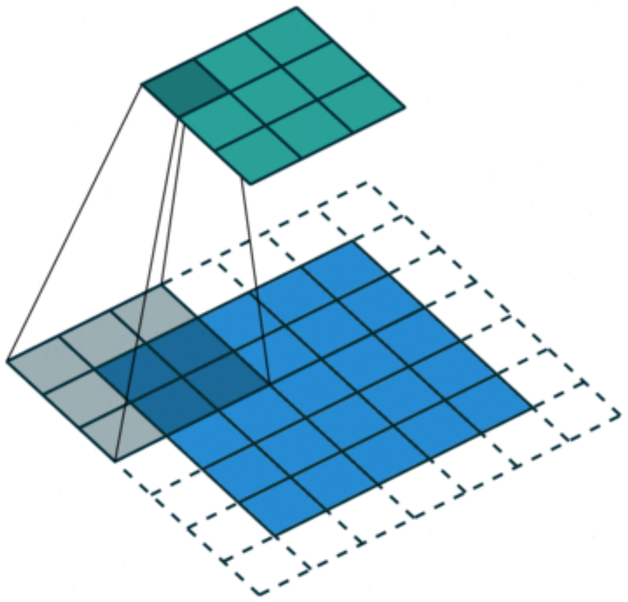
```
[[ 1.  1.   1.   0.
 [ 1.  1.   1.   0.
 [ 1.  1.   1.   0.
 [ 1.  1.   1.   0.
```

Tolerant to 1 pixel

Padding, Stride, and Multiple Channels

- Pooling layers have similar padding and stride as convolutional layers
- No learnable parameters
- Apply pooling for each input channel to obtain the corresponding output channel

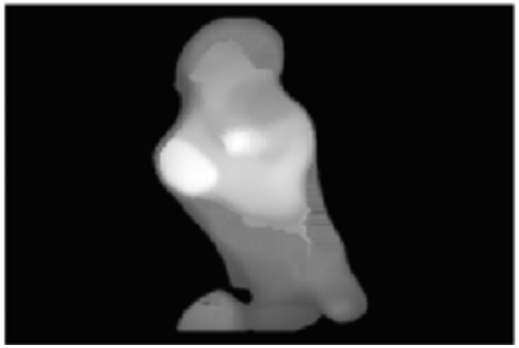
#output channels = #input channels



Average Pooling

- Max pooling: the strongest pattern signal in a window
- Average pooling: replace max with mean in max pooling
  - The average signal strength in a window

Max pooling

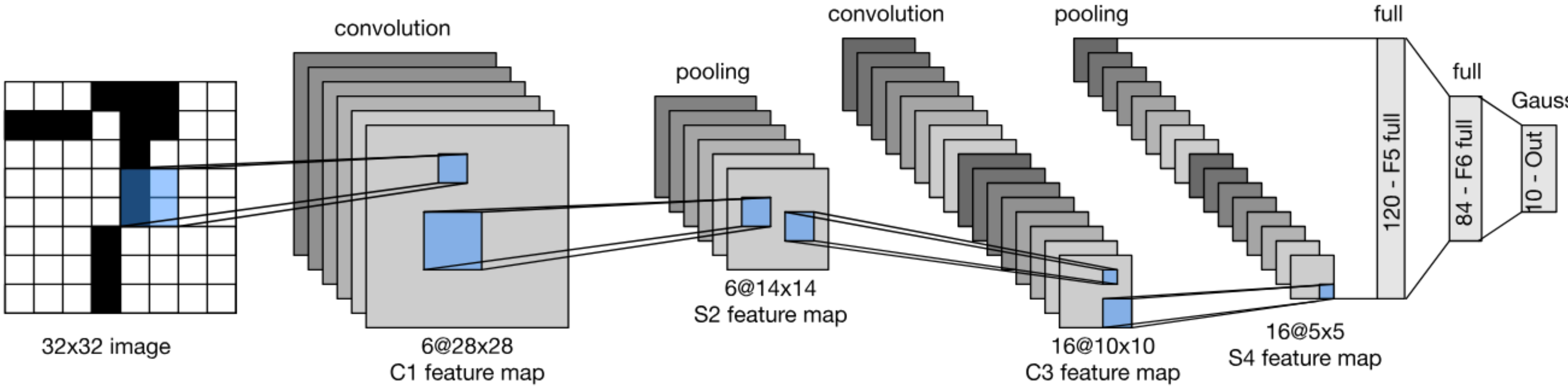


Average pooling

Quiz

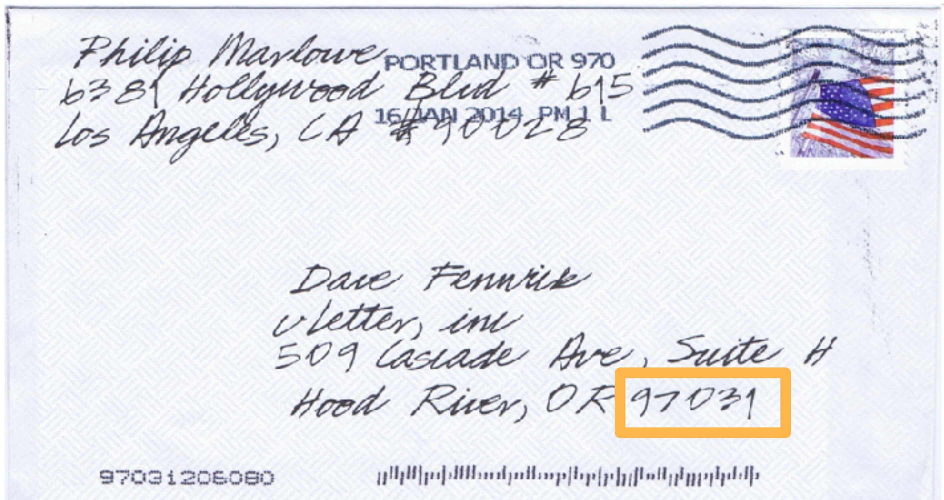
- https://edstem.org/us/courses/22801/lessons/45024/slides/257680

LeNet Architecture



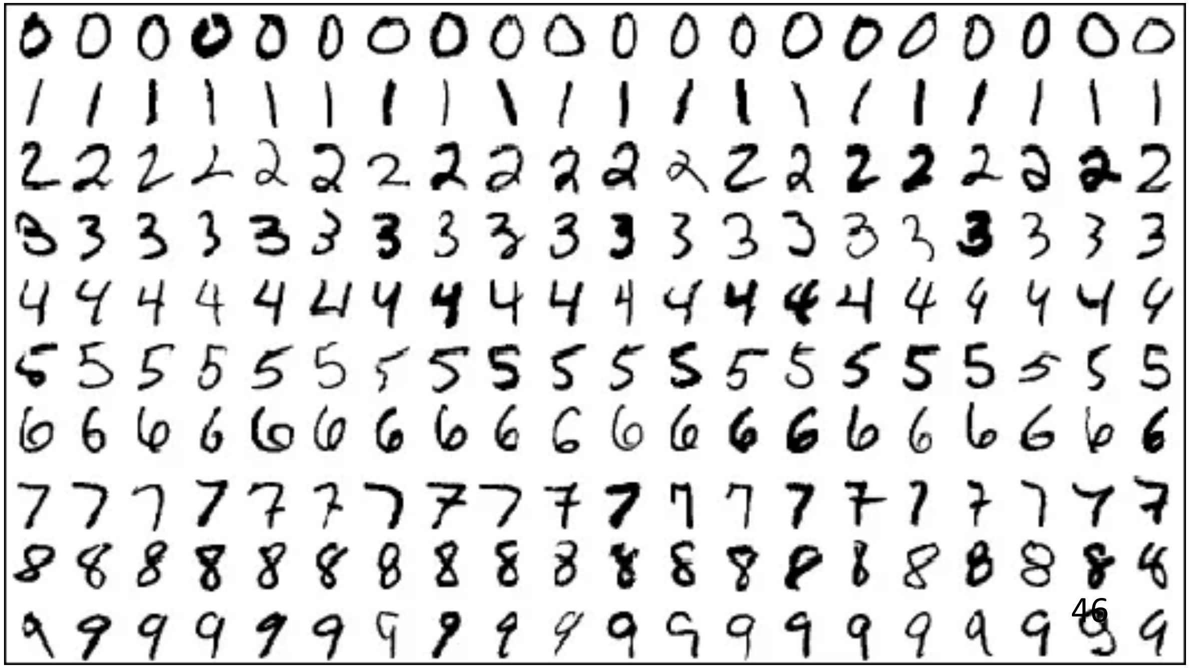
Handwritten Digit Recognition

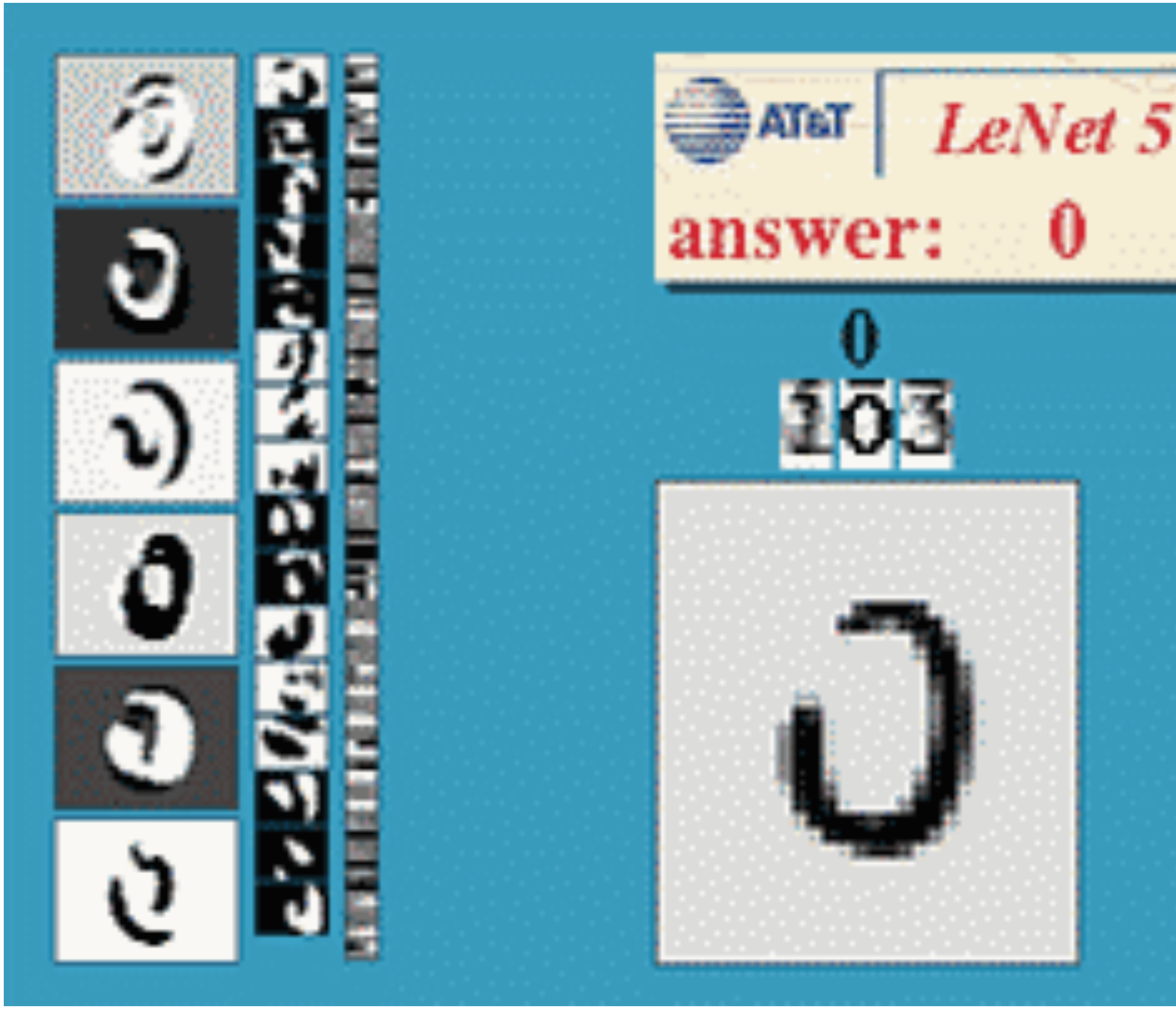
An instance of optical character recognition (OCR)



MNIST

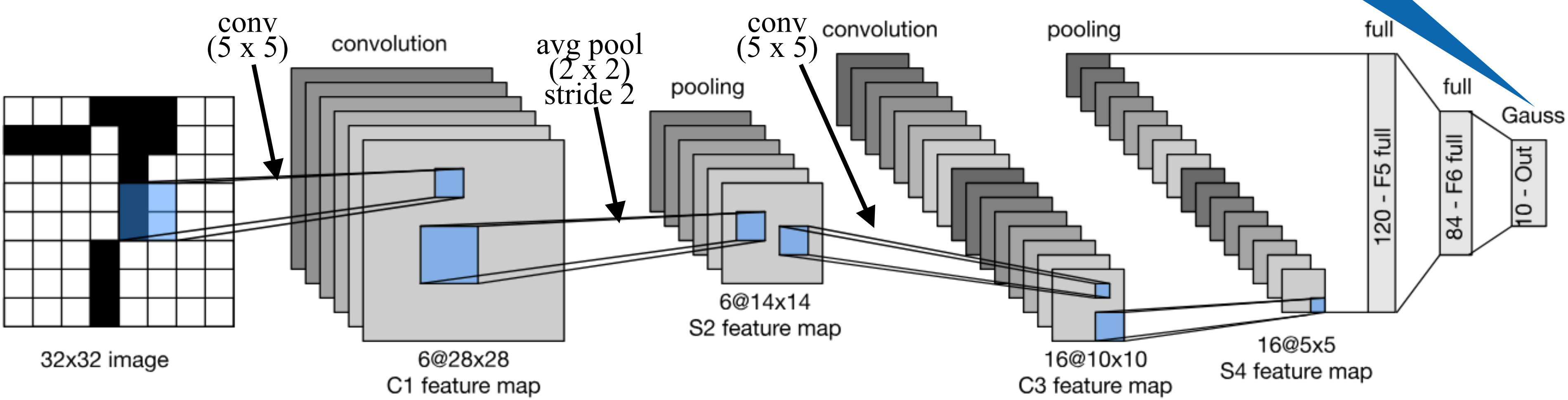
- Centered and scaled
- 50,000 training data
- 10,000 test data
- 28 x 28 images
- 10 classes





Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, 1998 Gradient-based learning applied to document recognition

Expensive if we have many outputs



LeNet-5

Layer	#channels	kernel size	stride	activation	feature map size
Input					32 x 32 x 1
Conv 1	6	5 x 5	1	tanh	28 x 28 x 6
Avg Pooling 1		2 x 2	2		14 x 14 x 6
Conv 2	16	5 x 5	1	tanh	10 x 10 x 16
Avg Pooling 2		2 x 2	2		5 x 5 x 16
Conv 3	120	5 x 5	1	tanh	120
FC 1					84
FC 2					10

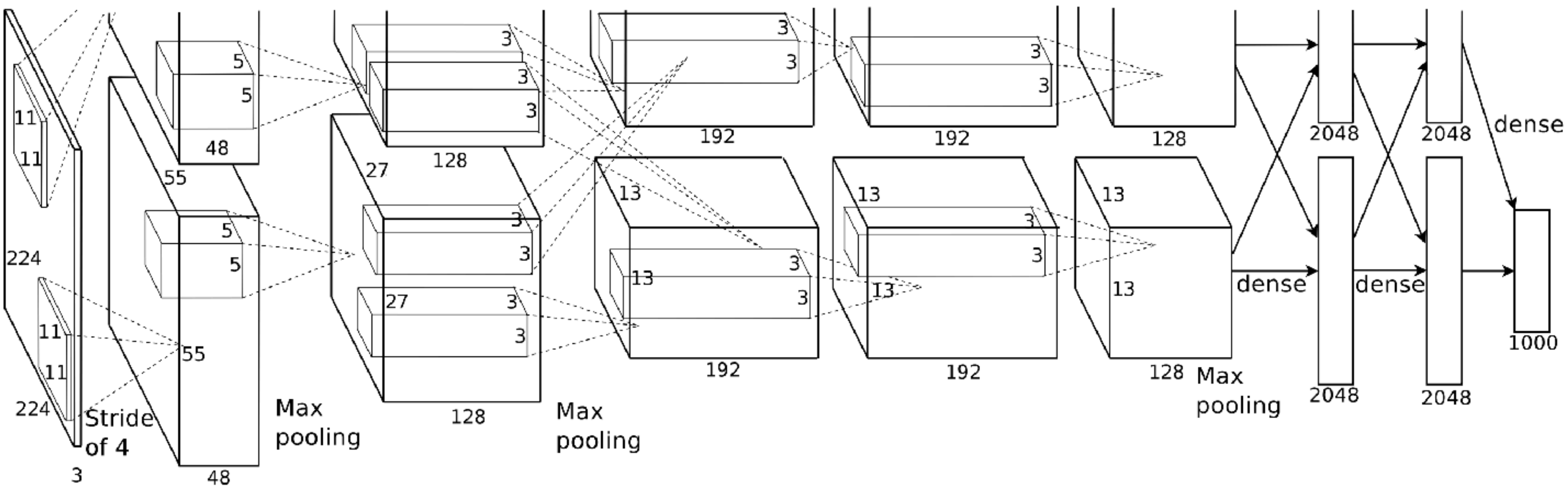
LeNet in Pytorch

```
class LeNet(nn.Module):

  def __init__(self):
    super(LeNet, self).__init__()
    self.model = nn.Sequential(
      nn.Conv2d(in_channels = 1, out_channels = 6, kernel_size = 5, stride = 1,
padding = 0),
      nn.Tanh(),
      nn.AvgPool2d(kernel_size = 2, stride = 2),
      nn.Conv2d(in_channels = 6, out_channels = 16, kernel_size = 5, stride = 1,
padding = 0),
      nn.Tanh(),
      nn.AvgPool2d(kernel_size = 2, stride = 2),
      nn.Conv2d(in_channels = 16, out_channels = 120, kernel_size = 5, stride =
1, padding = 0),
      nn.Flatten(),
      nn.Linear(120, 84),
      nn.Tanh(),
      nn.Linear(84, 10))

  def forward(self, x):
    y = self.model(x)
    return y
```

Recap

- Convolutional layer
  - Reduced model capacity compared to dense layer
  - Efficient at detecting spatial pattens
  - High computation complexity
  - Control output shape via padding, strides and channels
- Max/Average Pooling layer
  - Provides some degree of invariance to translation

AlexNet



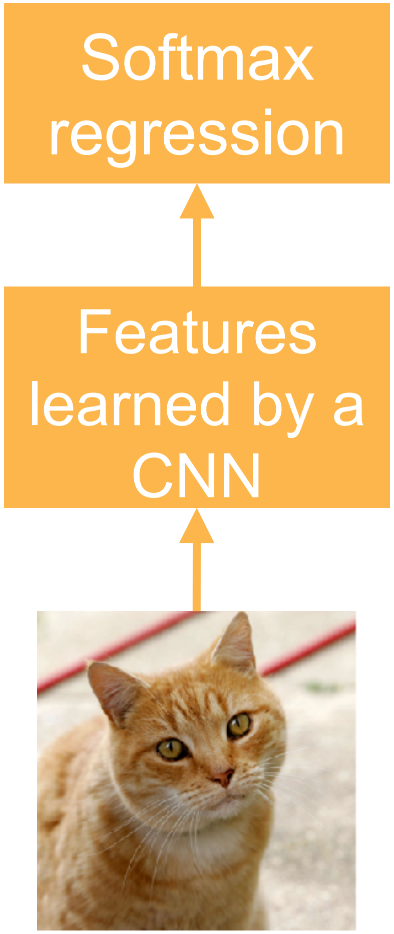
11
11
224
224
11
11
Stride of 4
3
55
5
5
48
5
5
55
48
Max pooling
27
3
3
3
128
27
3
3
3
3
128
Max pooling
3
3
192
13
13
3
3
192
3
3
192
13
13
3
3
192
128
13
13
128
Max pooling
2048
2048
dense
dense
dense
2048
2048
1000

ImageNet (2010)

Images	Color images with nature objects	Gray image for hand-written digits
Size	469 x 387	28 x 28
# examples	1.2 M	60 K
# classes	1,000	10

AlexNet

- AlexNet won ImageNet competition in 2012
- Deeper and bigger LeNet
- Key modifications
  - Dropout (regularization)
  - ReLu (training)
  - MaxPooling
- Paradigm shift for computer vision

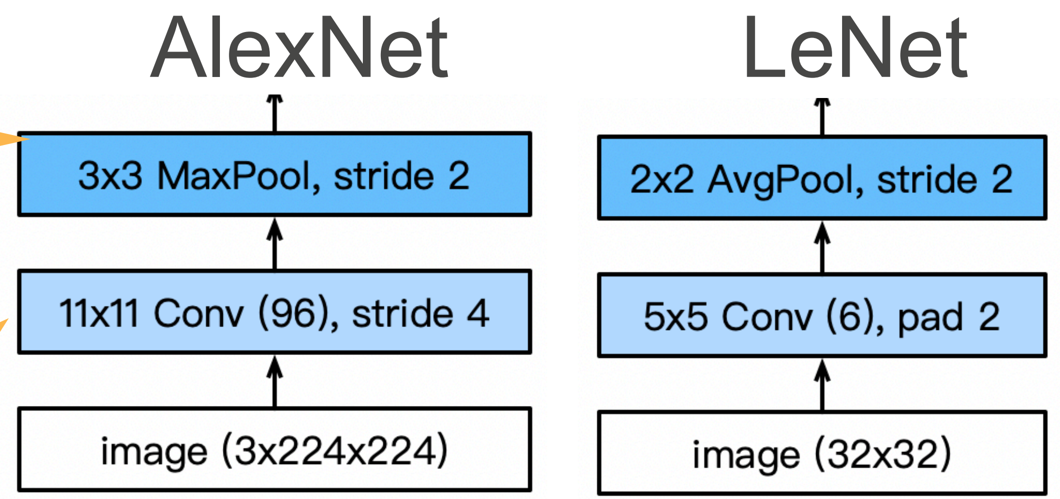


Alex Krizhevsky, Ilya Sutskever, Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. 2012

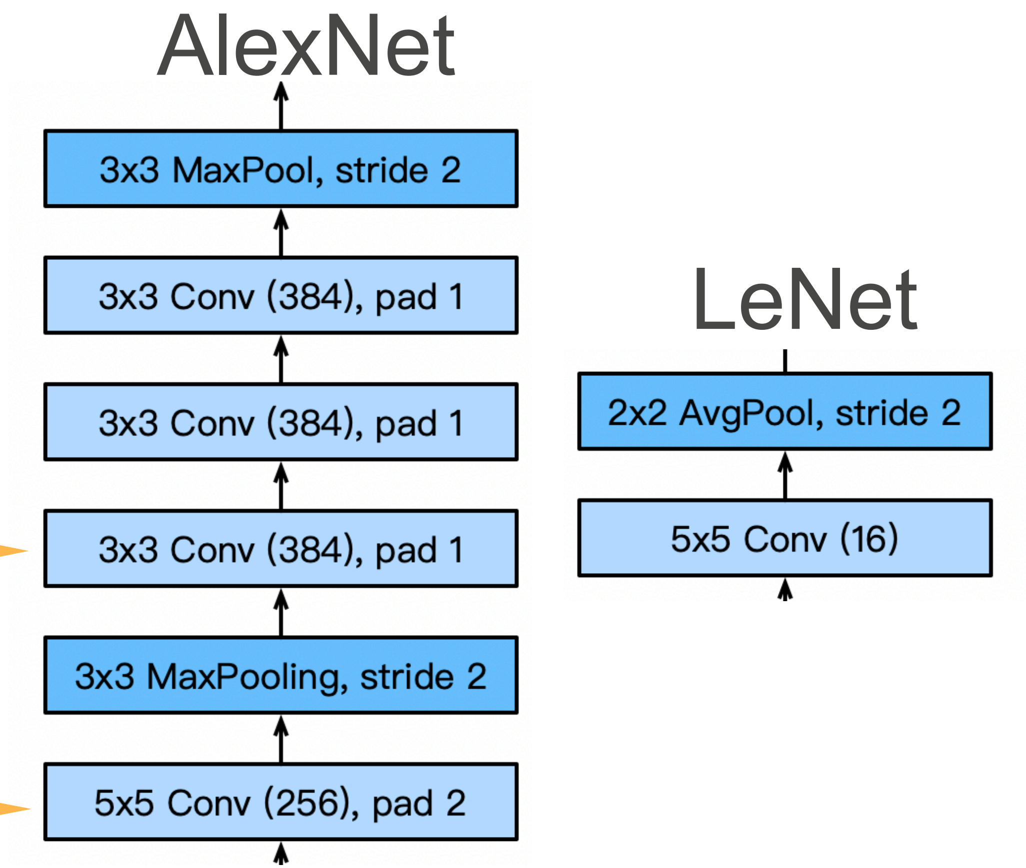
AlexNet Architecture

Larger pool size, change to max pooling

Larger kernel size, stride because of the increased image size, and more output channels.



AlexNet Architecture



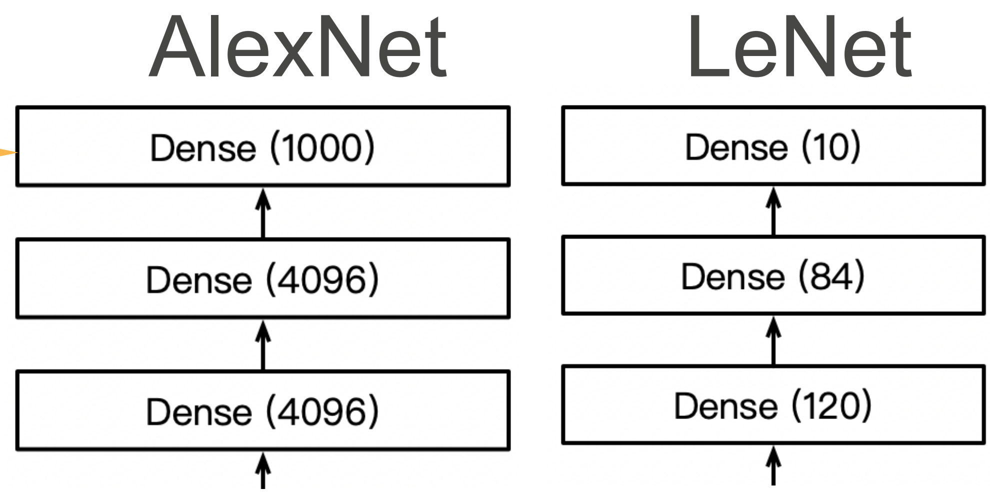
3 additional convolutional layers

More output channels.

AlexNet Architecture

1000 classes output

Increase hidden size from 120 to 4096



More Tricks

- Change activation function from sigmoid to ReLu (no more vanishing gradient)
- Add a dropout layer after two hidden FFN layers (better robustness / regularization)
- Data augmentation

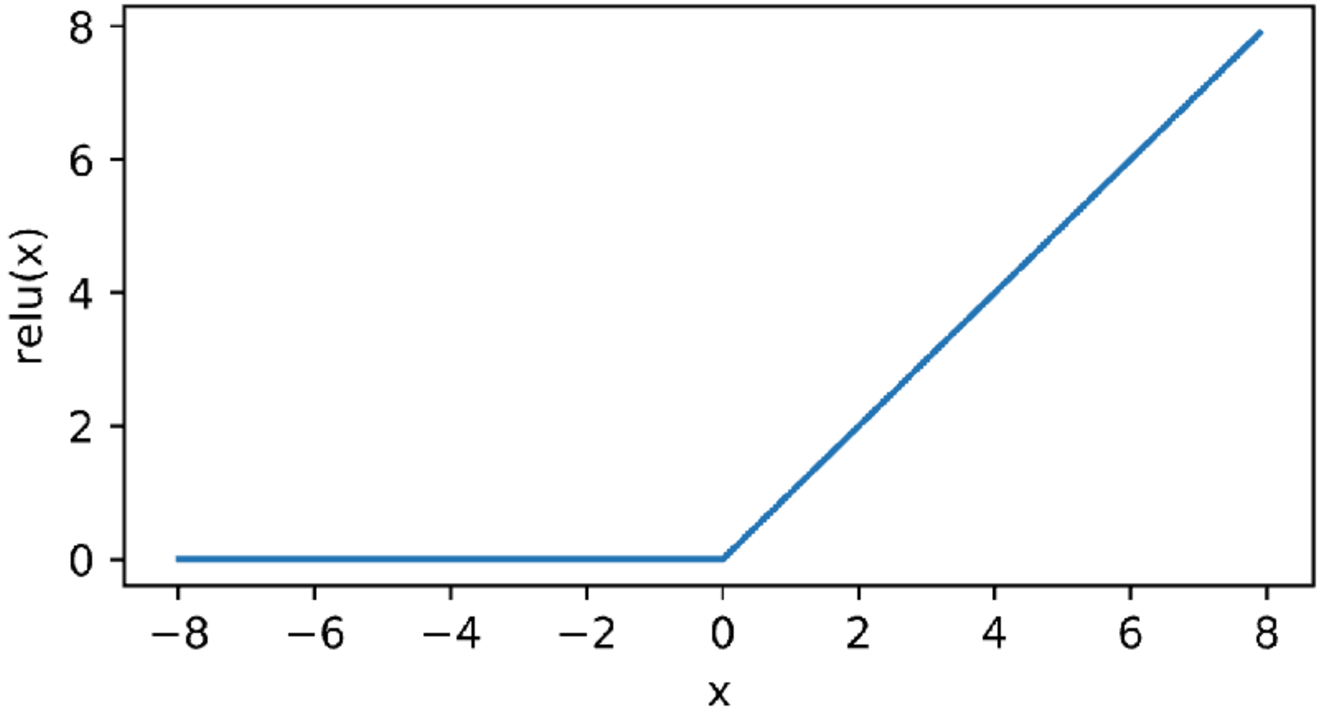
Data Augmentation

- Create additional training data with existing data

ReLU Activation

ReLU: rectified linear unit

$$\text{ReLU}(x) = \max(x, 0)$$

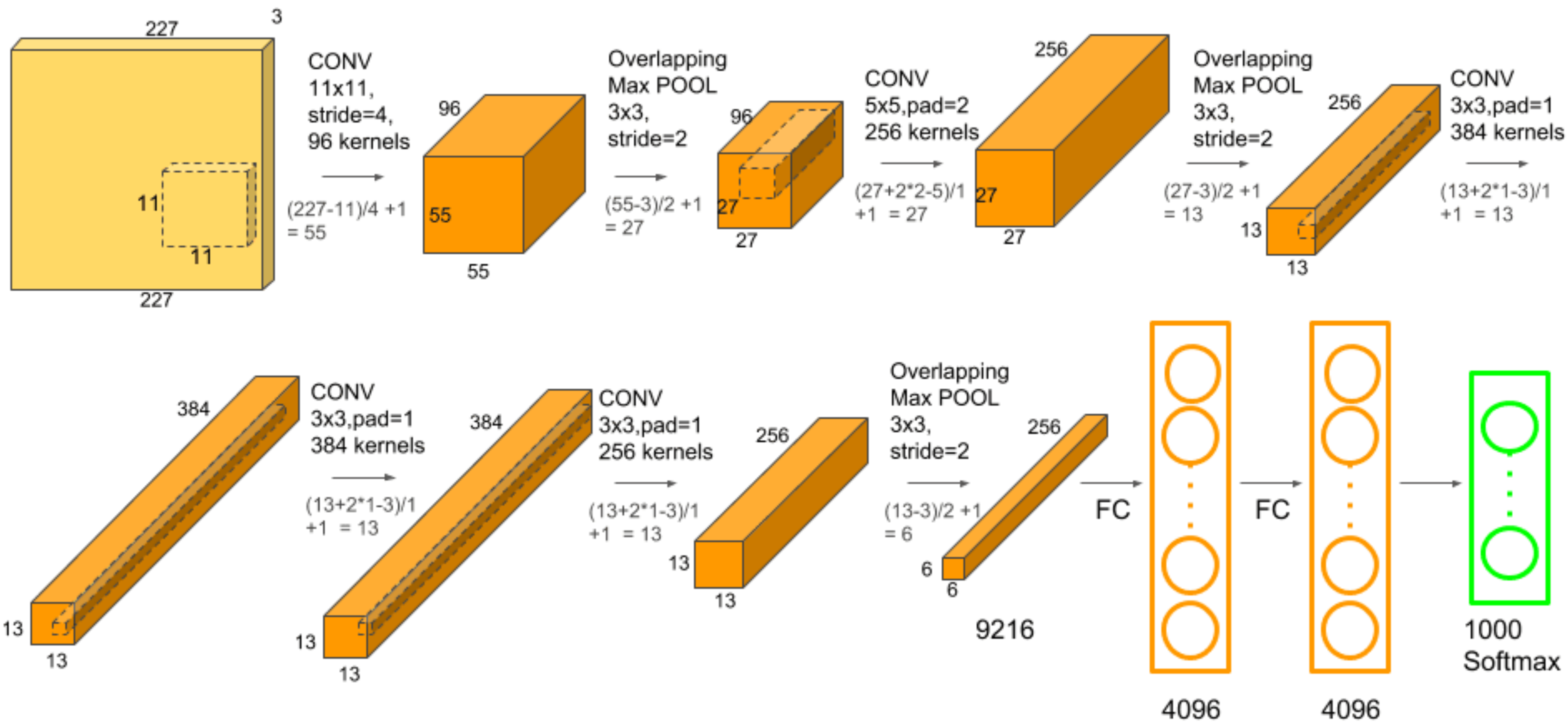


Dropout Layer

- For every input x_i, Dropout produces

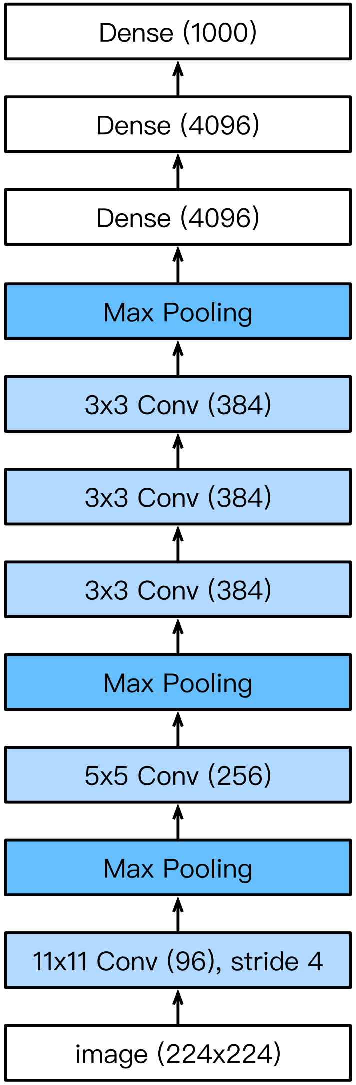
$$x_i' = \begin{cases} 0 & \text{with probablity } p \\ \dfrac{x_i}{1-p} & \text{otherise} \end{cases}$$

AlexNet

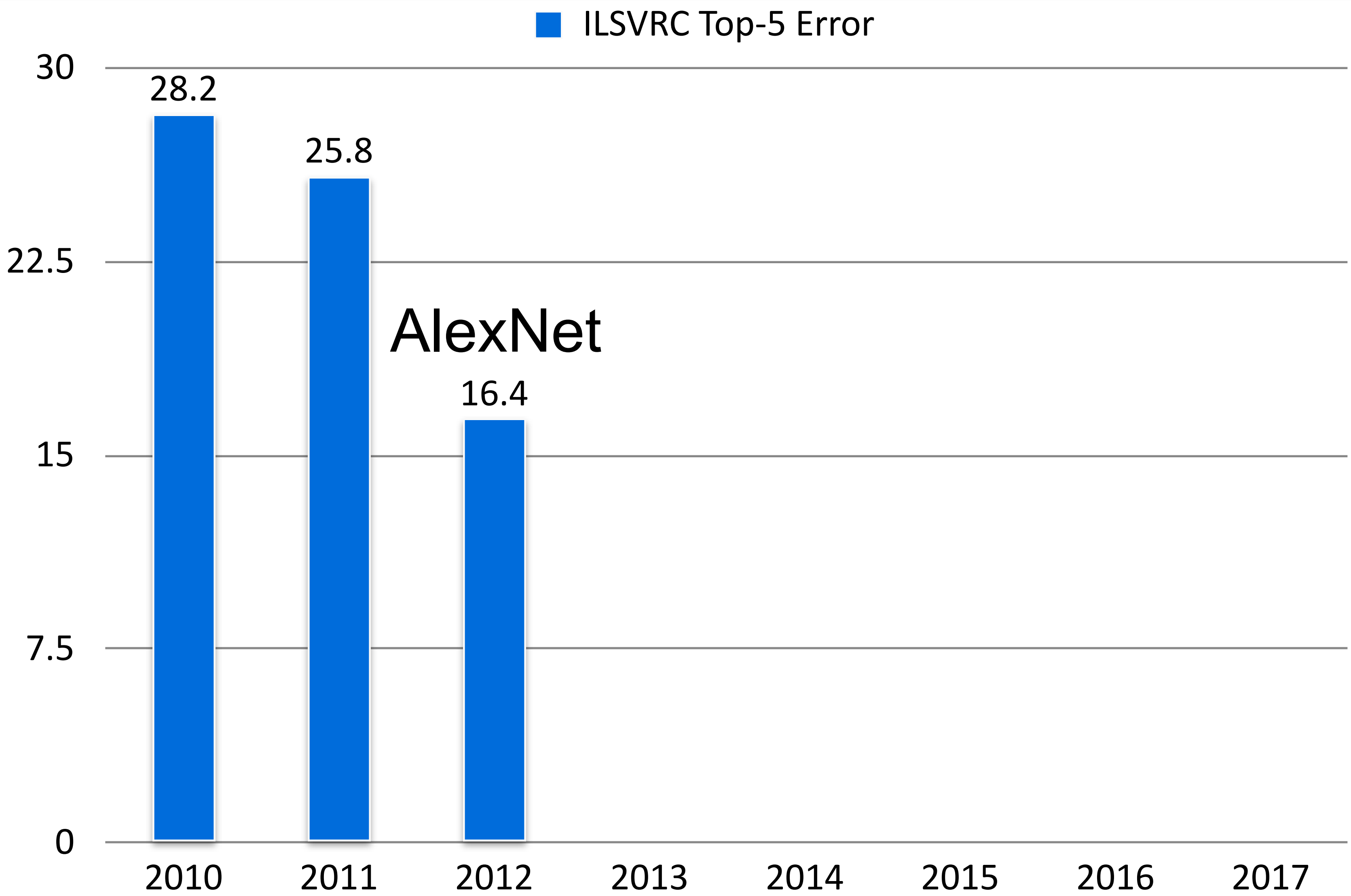

227
3
227
11
11
CONV
11x11,
stride=4,
96 kernels
(227-11)/4 +1
= 55
96
55
55
Overlapping
Max POOL
3x3,
stride=2
(55-3)/2 +1
= 27
96
27
27
CONV
5x5,pad=2
256 kernels
(27+2*2-5)/1
+1 = 27
256
27
27
Overlapping
Max POOL
3x3,
stride=2
(27-3)/2 +1
= 13
256
13
13
CONV
3x3,pad=1
384 kernels
(13+2*1-3)/1
+1 = 13
384
13
13
CONV
3x3,pad=1
384 kernels
(13+2*1-3)/1
+1 = 13
384
13
13
CONV
3x3,pad=1
256 kernels
(13+2*1-3)/1
+1 = 13
256
13
13
Overlapping
Max POOL
3x3,
stride=2
(13-3)/2 +1
= 6
256
6
6
9216
FC
4096
FC
4096
1000
Softmax

Complexity

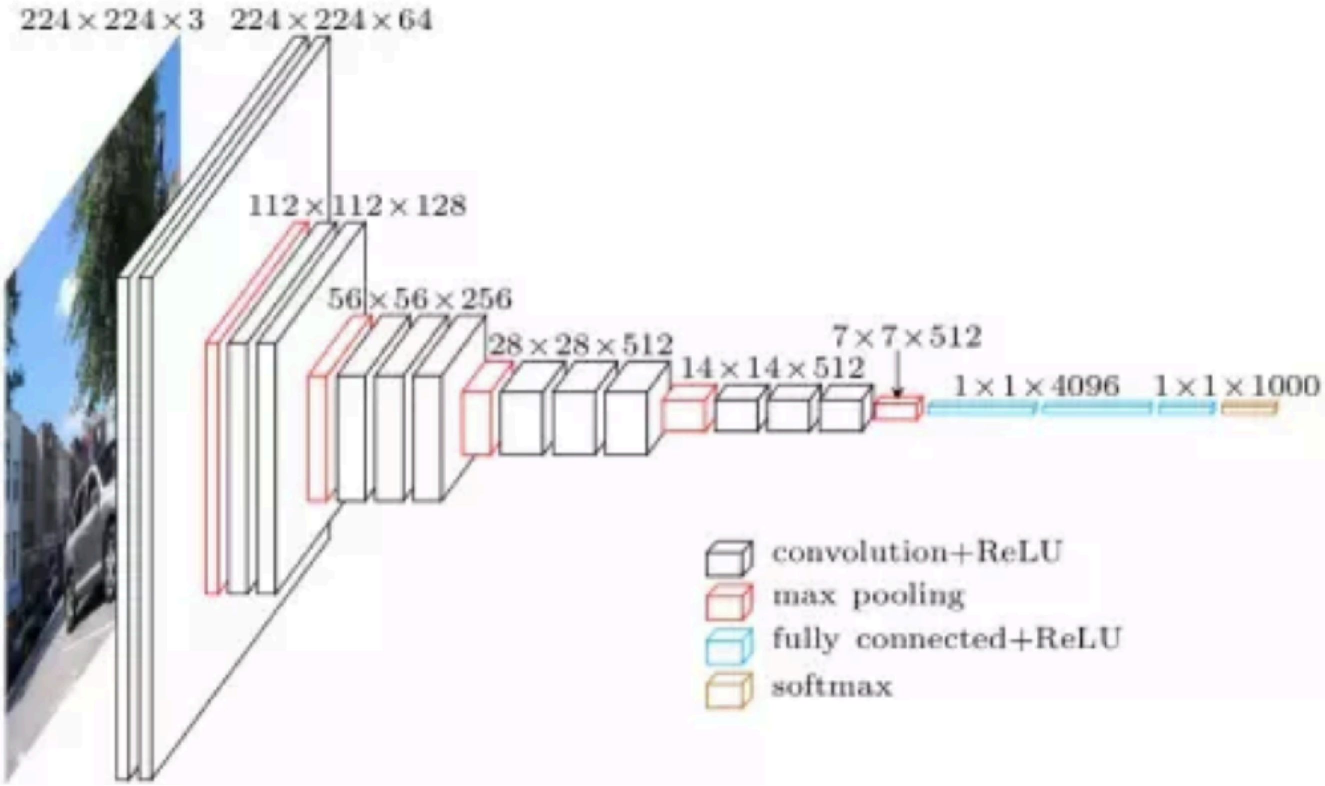
	#parameters		FLOP	
	AlexNet	LeNet	AlexNet	LeNet
Conv1	35K	150	101M	1.2M
Conv2	614K	2.4K	415M	2.4M
Conv3-5	3M		445M	
Dense1	26M	0.48M	26M	0.48M
Dense2	16M	0.1M	16M	0.1M
Total	46M	0.6M	1G	4M
Increase	11x	1x	250x	1x



ImageNet Results: ILSVRC Winners



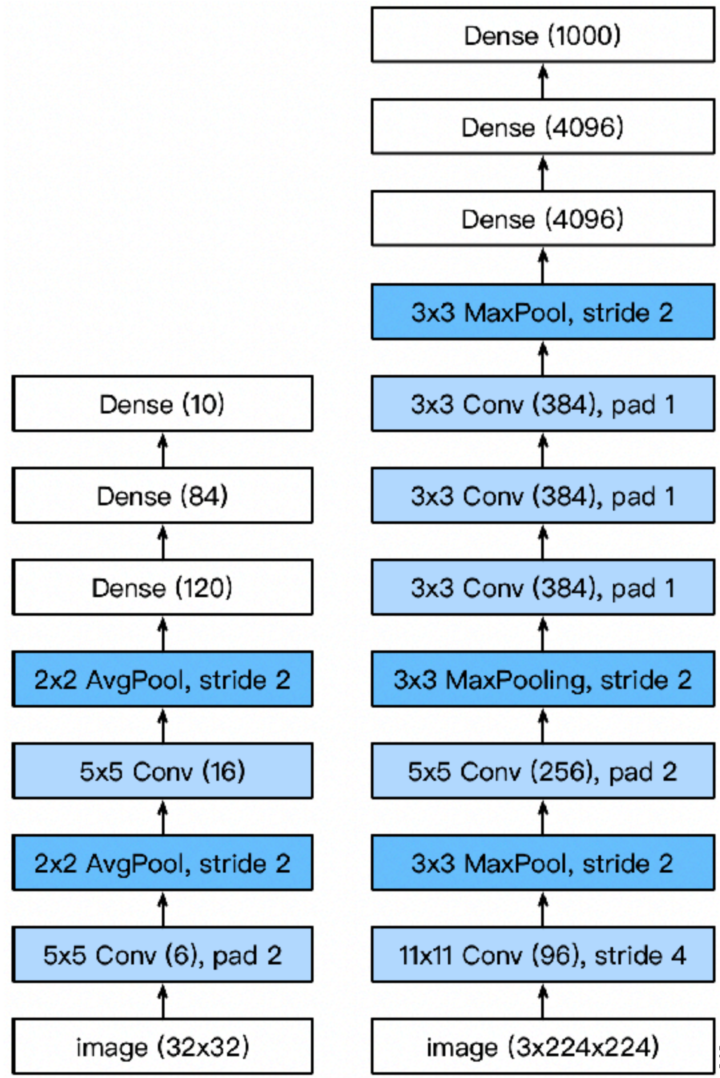
VGG



224 × 224 × 3
224 × 224 × 64
112 × 112 × 128
56 × 56 × 256
28 × 28 × 512
14 × 14 × 512
7 × 7 × 512
1 × 1 × 4096
1 × 1 × 1000
convolution+ReLU
max pooling
fully connected+ReLU
softmax

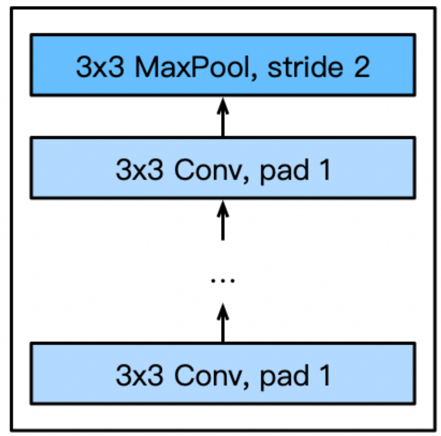
VGG

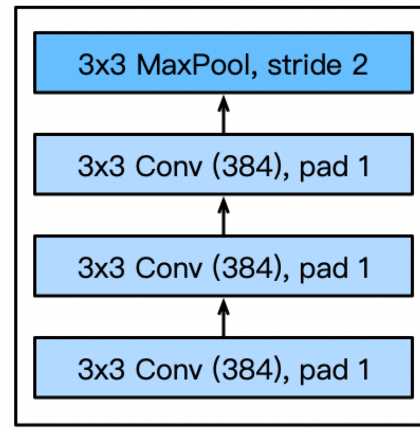
- AlexNet is deeper and bigger than LeNet to get performance
- Go even bigger & deeper?
- Options
  - More dense layers (too expensive)
  - **More** convolutions
  - Group into **blocks**



VGG Blocks

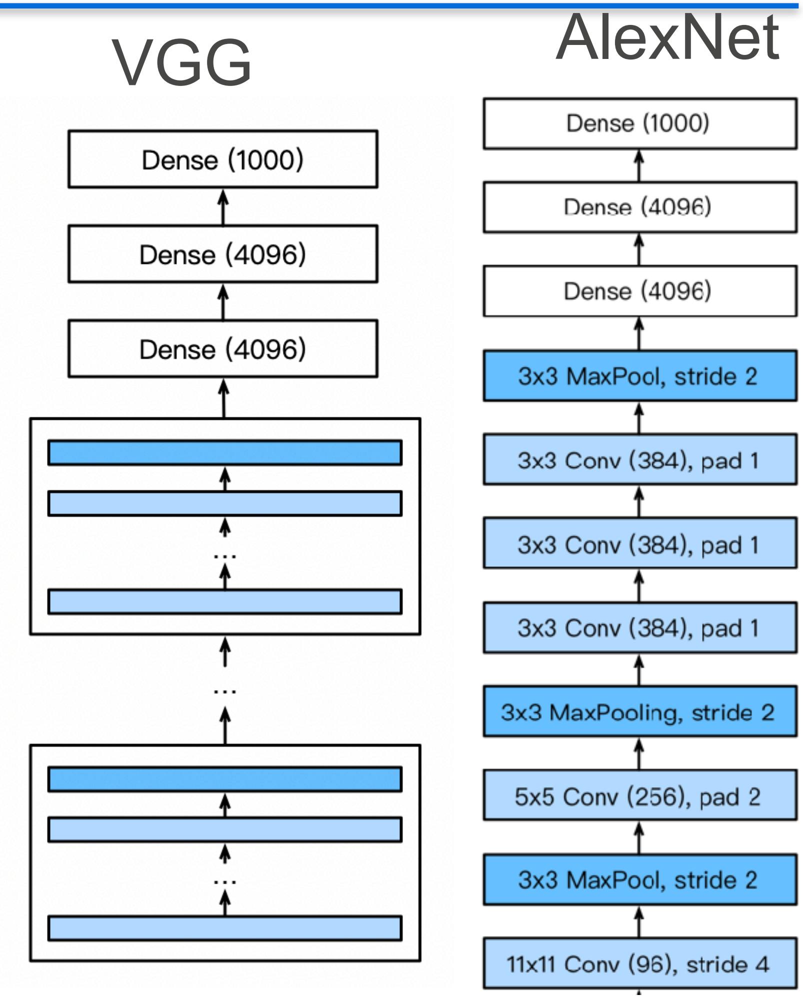
- Deeper vs. wider?
  - 5x5 convolutions
  - 3x3 convolutions (more)
  - **Deep & narrow better**
- VGG block
  - *3x3* convolutions (pad 1) **(n layers, m channels)**
  - 2x2 max-pooling (stride 2)





VGG Architecture

- Multiple VGG blocks followed by dense layers
- Vary the repeating number to get different architectures, such as VGG-16, VGG-19, …



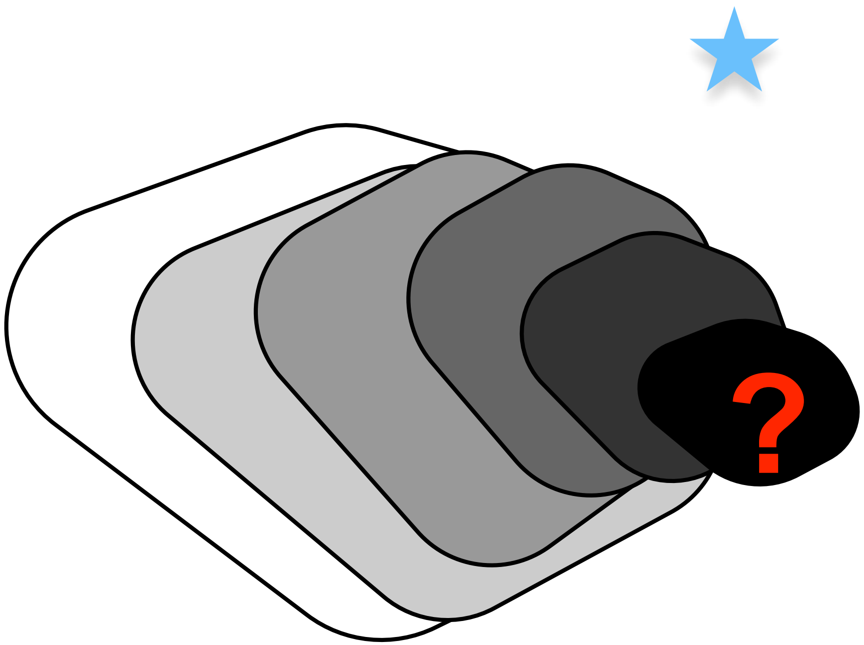
Going Deeper

- LeNet (1995)
  - 2 convolution + pooling layers
  - 2 hidden dense layers
- AlexNet
  - Bigger and deeper LeNet
  - ReLu, Dropout, preprocessing
- VGG
  - Bigger and deeper AlexNet (repeated VGG blocks)

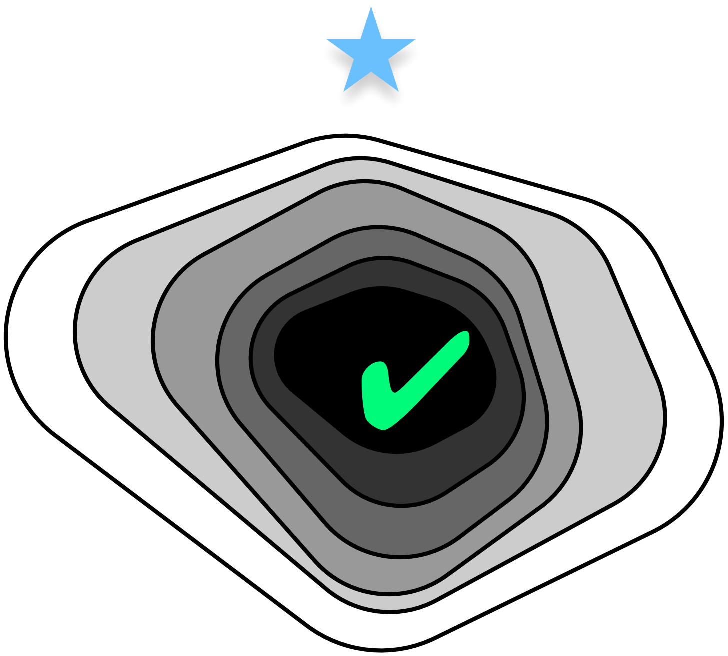
Residual Networks

Best paper CVPR 2016

Does adding layers improve accuracy?



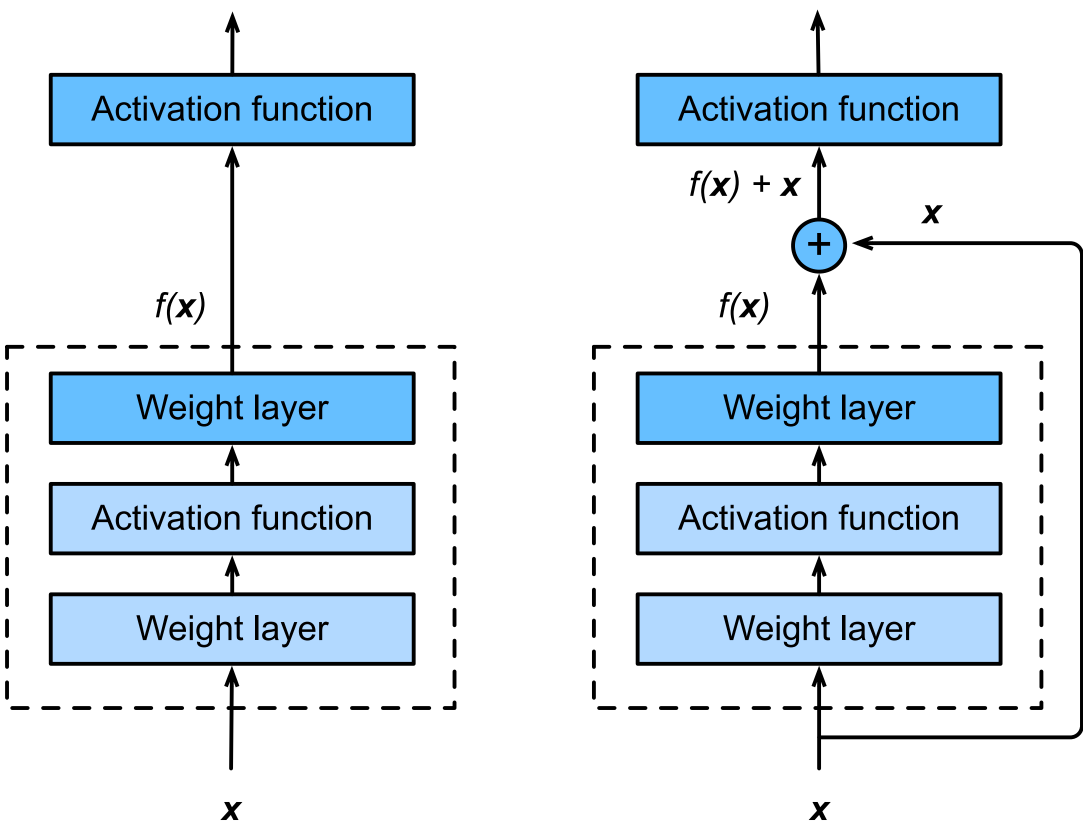
generic function classes



nested function classes

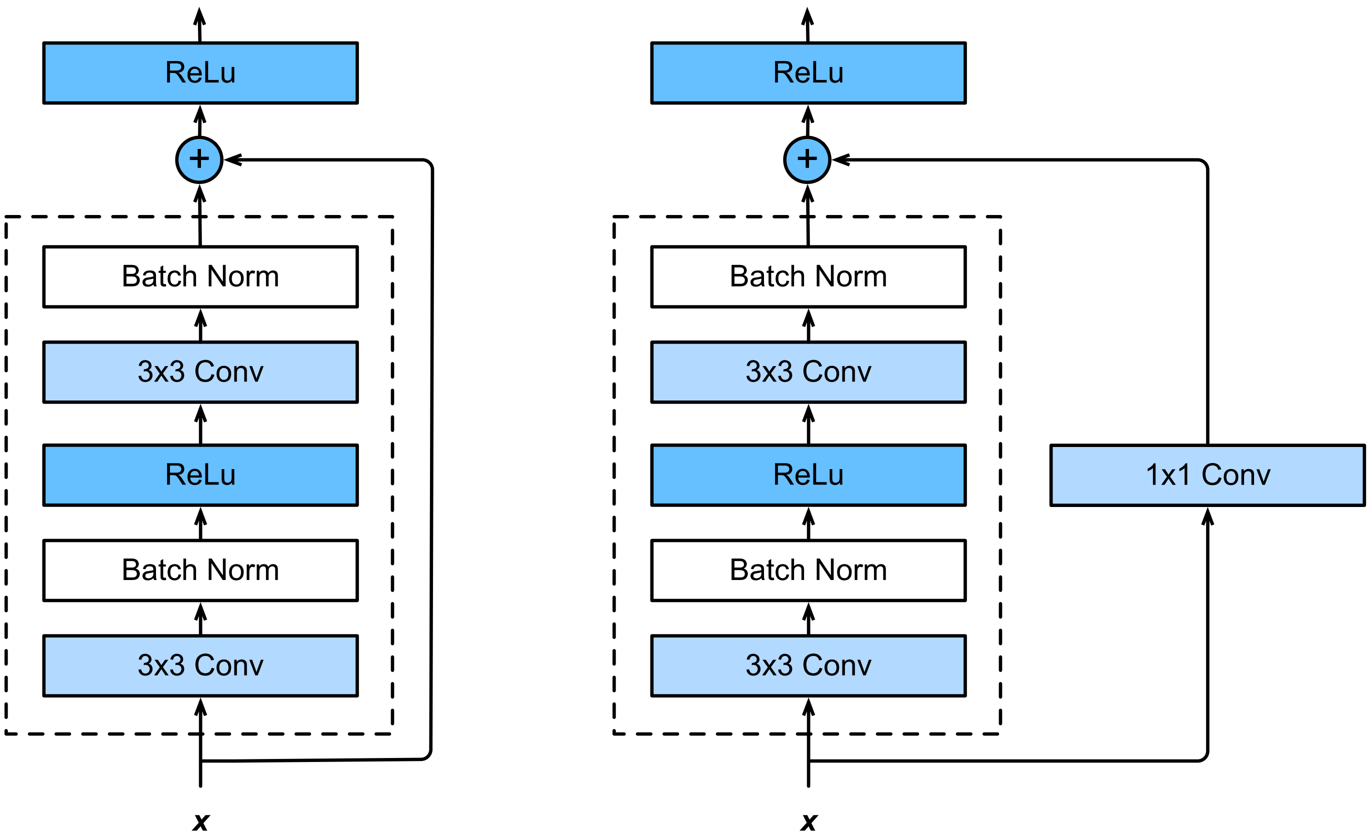
Residual Networks

- Adding a layer **changes** function class
- We want to **add to** the function class
- 'Taylor expansion' style $f(x) = x + g(x)$ parametrization

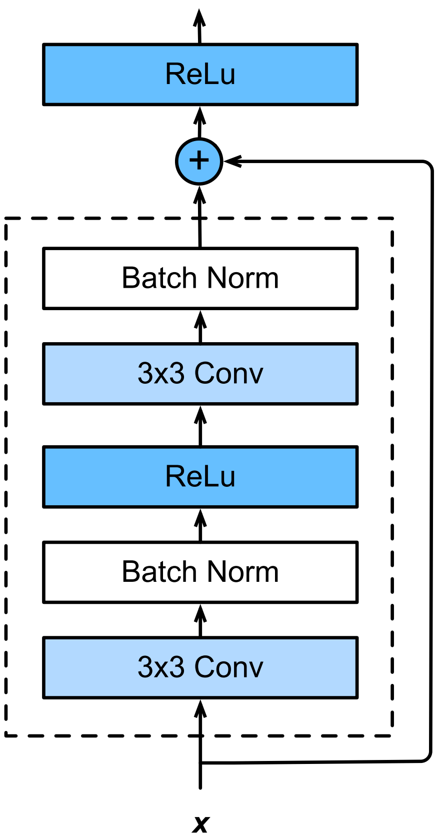


Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, Deep Residual Learning for Image Recognition. 2016

ResNet Block in detail



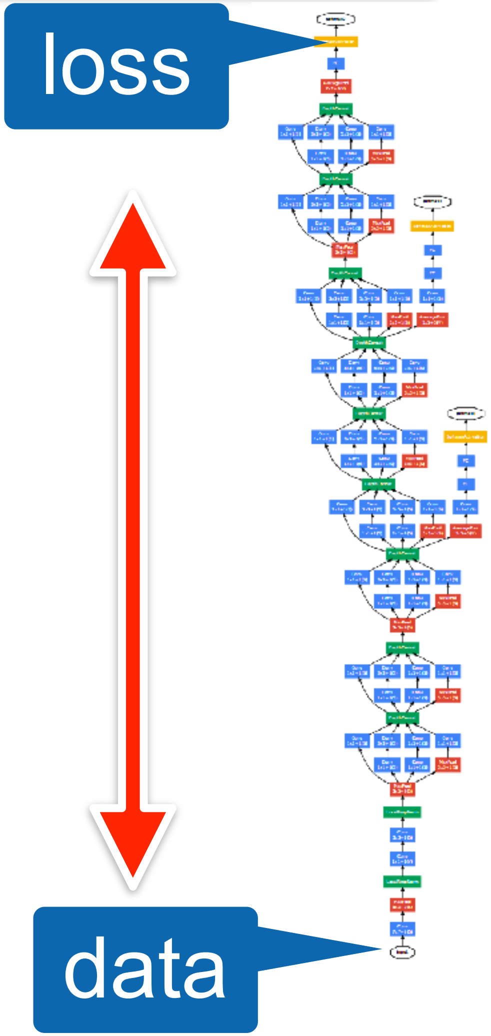
Code




```
# an essential block of layers which forms resnets
class ResBlock(nn.Module):
  #in_channels -> input channels,int_channels->intermediate channels
  def __init__(self,in_channels,int_channels,identity_downsample=None,stride=1):
    super(ResBlock,self).__init__()
    self.expansion = 4
    self.conv1 = nn.Conv2d(in_channels,int_channels,kernel_size=1,stride=1,padding=0)
    self.bn1 = nn.BatchNorm2d(int_channels)
    self.conv2 = nn.Conv2d(int_channels,int_channels,kernel_size=3,stride=stride,padding=1)
    self.bn2 = nn.BatchNorm2d(int_channels)
    self.conv3 = nn.Conv2d(int_channels,int_channels*self.expansion,kernel_size=1,stride=1,padding=
    self.bn3 = nn.BatchNorm2d(int_channels*self.expansion)
    self.relu = nn.ReLU()
    self.identity_downsample =  identity_downsample
    self.stride = stride

  def forward(self,x):
    identity = x.clone()
    x =  self.conv1(x)
    x =  self.bn1(x)
    x = self.relu(x)
    x = self.conv2(x)
    x = self.bn2(x)
    x = self.relu(x)
    x = self.conv3(x)
    x = self.bn3(x)
    #the so called skip connections
    if self.identity_downsample is not None:
      identity = self.identity_downsample(identity)
    x += identity
    x = self.relu(x)
    return x
```

Batch Normalization

- Loss occurs at last layer
  - Last layers learn quickly
- Data is inserted at first layer
  - Input layers change - **everything** changes
  - Last layers need to relearn many times
  - Slow convergence
- This is like covariate shift
  - The distribution of each layer shift across over training process



Batch Normalization

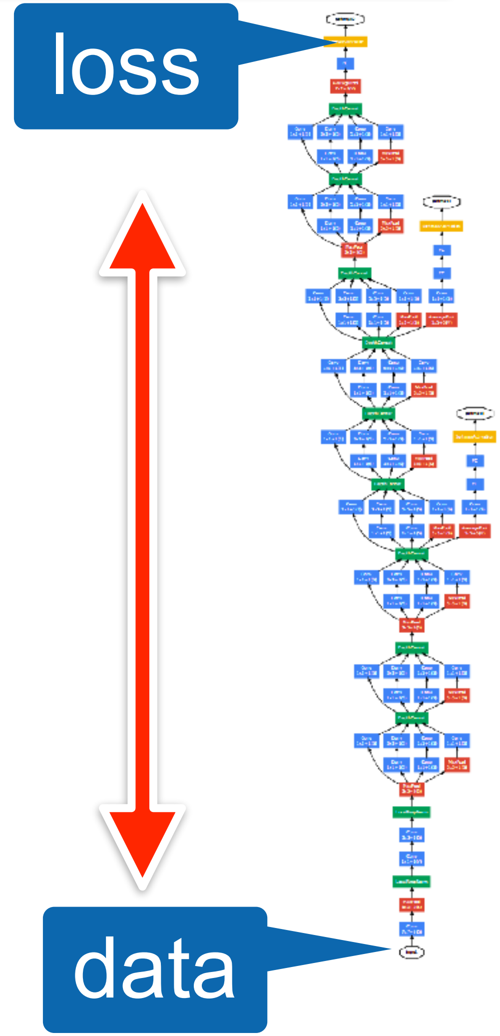
- For each layer, compute mean and variance

$$\mu_B = \frac{1}{|B|}\sum_{i\in B} x_i \text{ and } \sigma_B^2 = \frac{1}{|B|}\sum_{i\in B}(x_i - \mu_B)^2 + \epsilon$$

and adjust it separately

$$x_{i+1} = \gamma \frac{x_i - \mu_B}{\sigma_B} + \beta$$

- γ and β are learnable parameters



Sergey Ioffe, Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. 2015

This was the original motivation …

What Batch Norms really do

- Doesn't really reduce covariate shift (Lipton et al., 2018)
- Regularization by noise injection

$$x_{i+1} = \gamma \frac{x_i - \hat{\mu}_B}{\hat{\sigma}_B} + \beta$$

Random offset

Random scale

  - Random shift per minibatch
  - Random scale per minibatch
- No need to mix with dropout (both are capacity control)
- Ideal minibatch size of 64 to 256

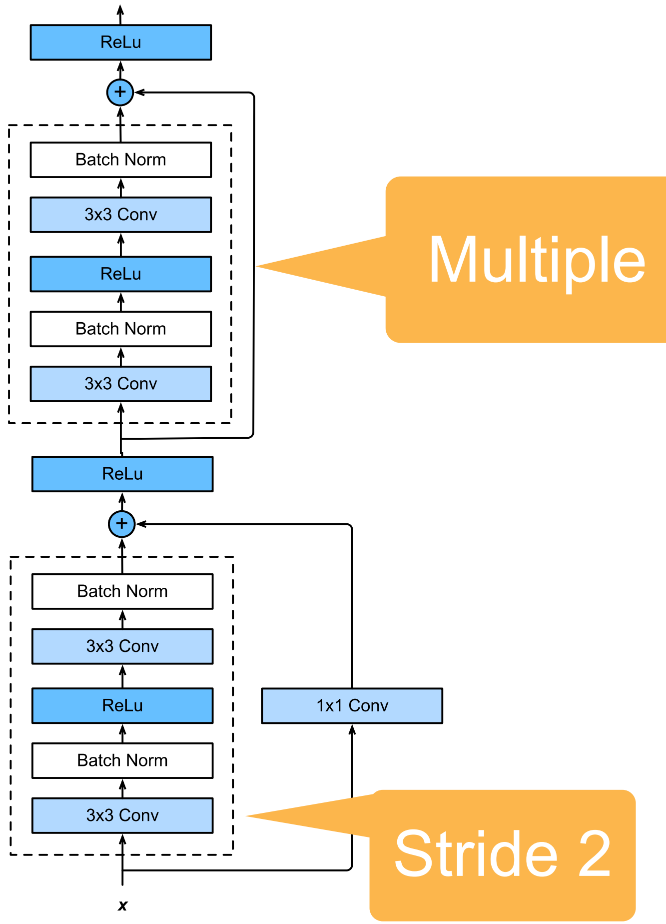
Code

```
torch.nn.BatchNorm1d(num_features)
```

```
torch.nn.BatchNorm2d(num_features)
>>> m = nn.BatchNorm2d(100)
>>> input = torch.randn(20, 100, 32, 32)
>>> output = m(input)
```

ResNet Module

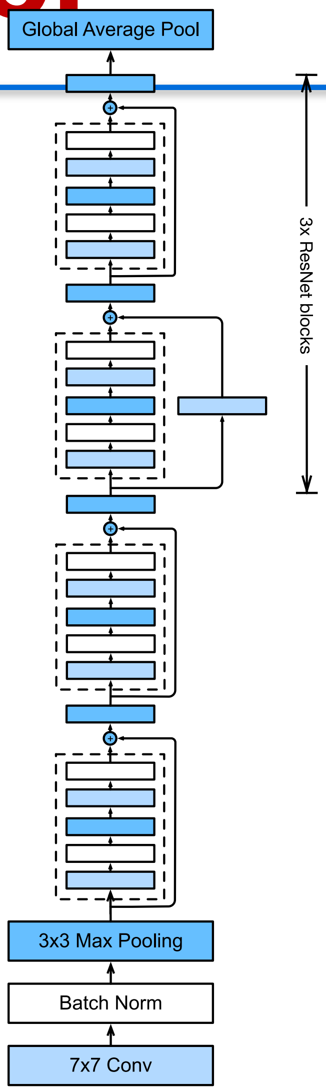
- Downsample per module (stride=2)
- Enforce some nontrivial nonlinearity per module (via 1x1 convolution)
- Stack up in blocks



Putting it all together

- Same block structure as e.g. VGG or GoogleNet
- Residual connection to add to expressiveness
- Pooling/stride for dimensionality reduction
- Batch Normalization for capacity control

… train it at scale …

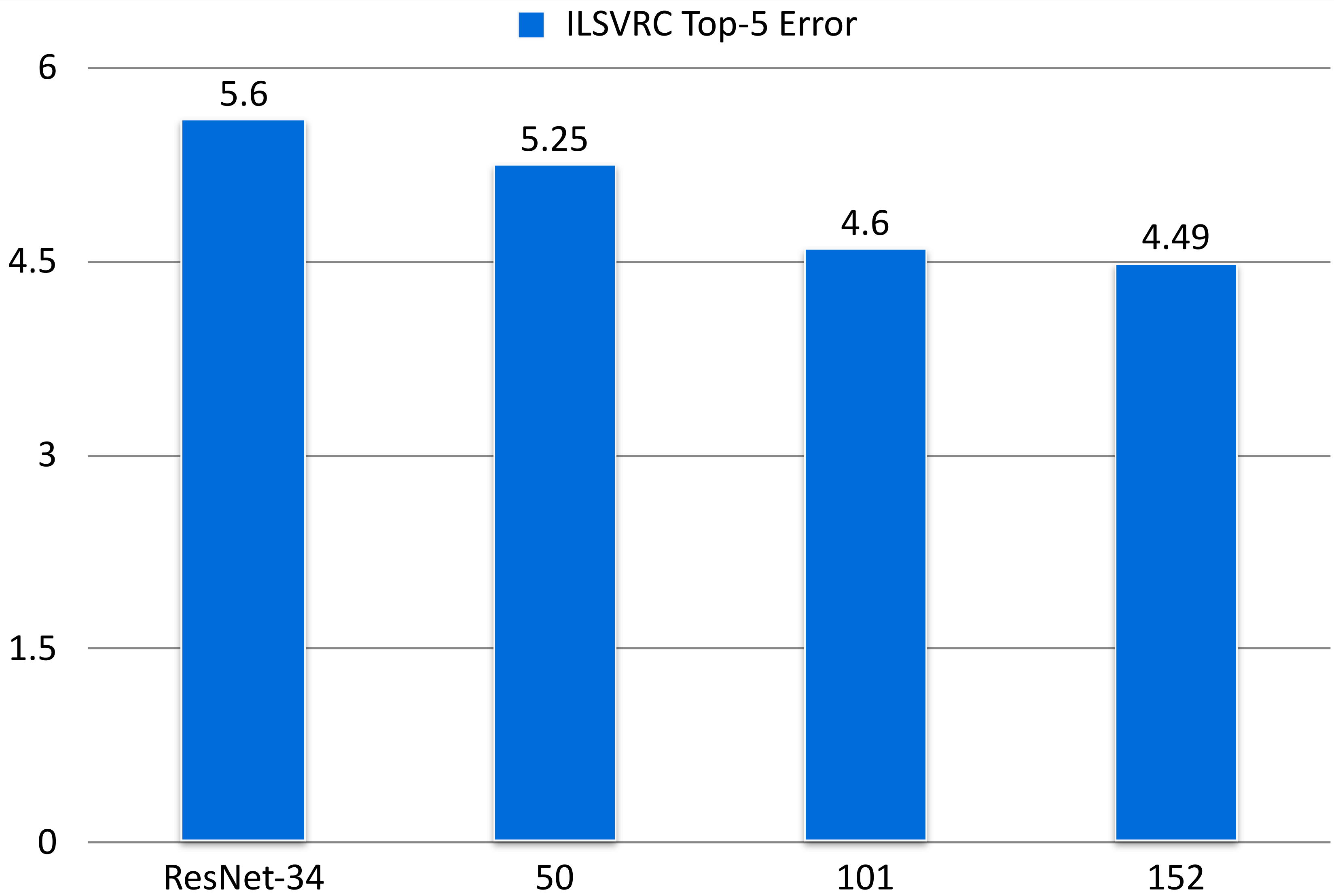


ResNet in Pytorch

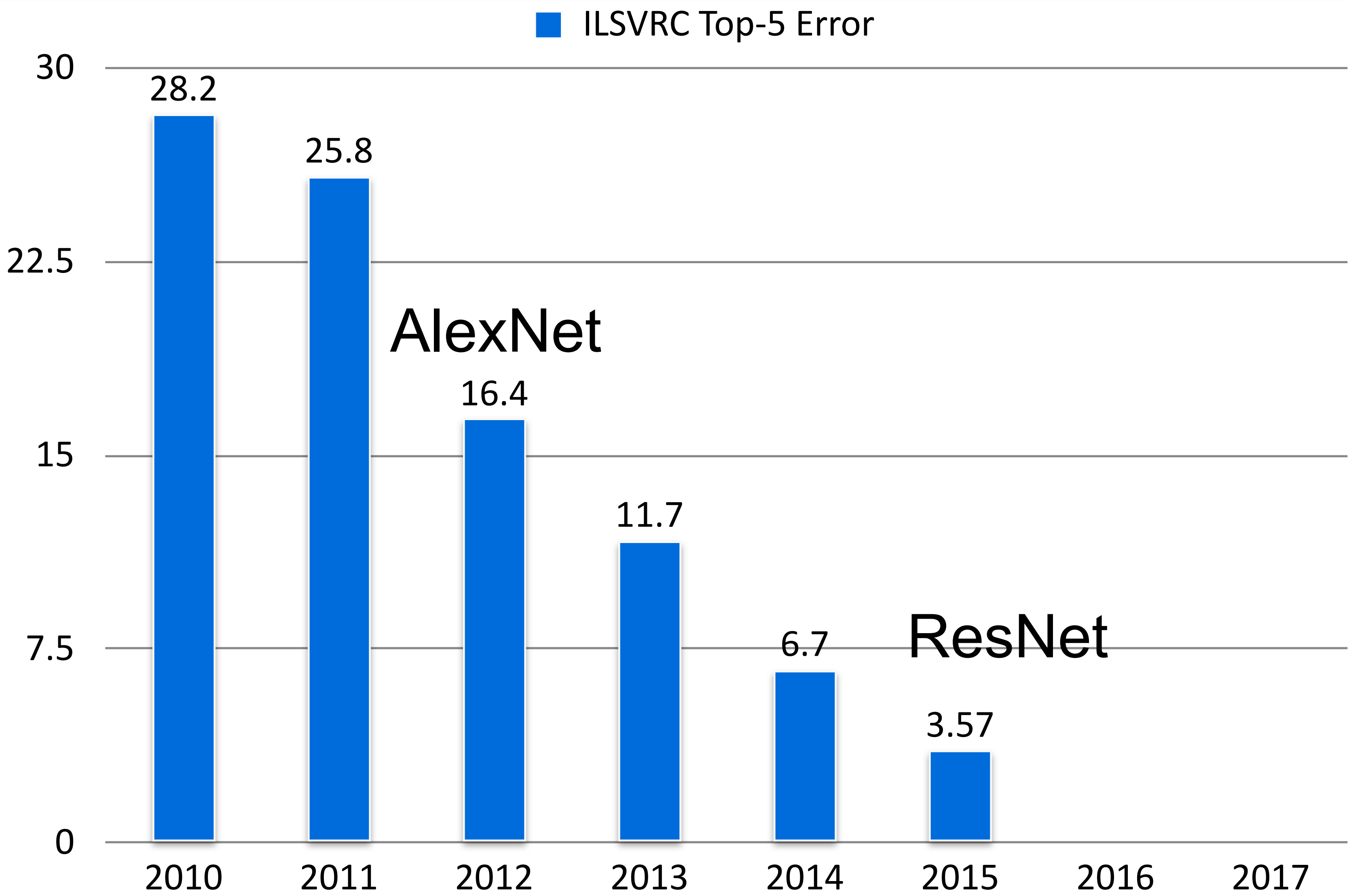
```
def _make_layer(self,block,num_res_blocks,int_channels,stride):
    identity_downsample =  None
    layers = []
    if stride!=1 or self.in_channels != int_channels*4:
      identity_downsample = nn.Sequential(nn.Conv2d(self.in_channels,int_channels*4,
                                                    kernel_size=1,stride=stride),
                                          nn.BatchNorm2d(int_channels*4))
layers.append(ResBlock(self.in_channels,int_channels,identity_downsample,stride))
      #this expansion size will always be 4 for all the types of ResNets
      self.in_channels =  int_channels*4
      for i in range(num_res_blocks-1):
        layers.append(ResBlock(self.in_channels,int_channels))
      return nn.Sequential(*layers)
```

https://medium.datadriveninvestor.com/cnn-architectures-from-scratch-c04d66ac20c2

Deeper is better



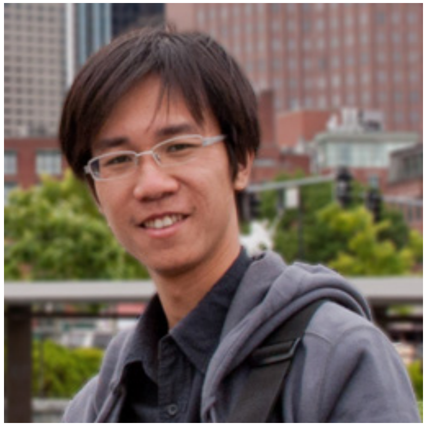
ImageNet Results: ILSVRC Winners



Notes

- ResNet won the champion for ILSVRC 2015
- The ResNet paper won the best paper award from CVPR 2016 (one of the leading CV conferences)
- Kaimin He won multiple best papers.

Papers of Kaimin He



- Exploring Simple Siamese Representation Learning. CVPR Best Paper Honorable Mention, 2021
- Group Normalization. ECCV Best Paper Honorable Mention, 2018
- Mask R-CNN. ICCV Best Paper Award (Marr Prize), 2017
- Focal Loss for Dense Object Detection. ICCV Best Student Paper Award, 2017
- Deep Residual Learning for Image Recognition. CVPR Best Paper Award, 2016
- Single Image Haze Removal using Dark Channel Prior. CVPR Best Paper Award, 2009

The first publication from Kaimin He

Discussion

- Your manager assigns a task for you: build a system to automatically select the cover photo for a short video on Tiktok
- Please discuss in groups how you plan to build the system

Summary

- Building blocks
  - Convolution
  - Stride
  - Padding
  - Channel
  - Pooling
  - Dropout
  - Batch Norm
  - Residual connection
- Data Augmentation
- Deeper is better — but still efficient