UC San Diego
JACOBS SCHOOL OF ENGINEERING

ECE/SIOC 228 Machine Learning for Physical Applications
Lecture 5: Convolutional Neural Network

Yuanyuan Shi

Assistant Professor, ECE

University of California, San Diego

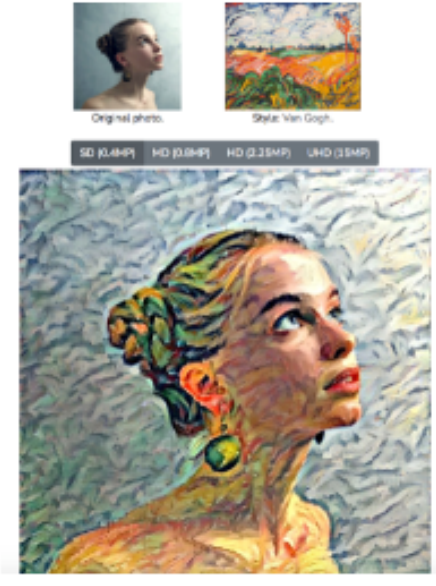
Reference & Acknowledgement: Stanford CS230 Deep Learning Lecture Note, Andrew Ng

Computer Vision Problems

UC San Diego

Image classification

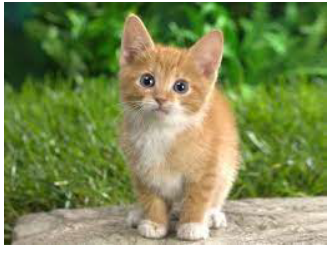
→Cat? (0/1)



https://neuralstyle.art/

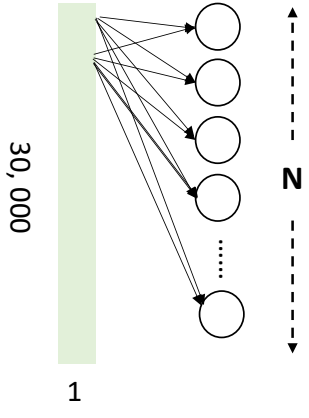
Using MLP?

UC San Diego



100*100*3 = 30, 000

Reshape



1 Layer of MLP

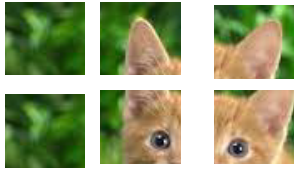
N	# parameters
1, 000	30,000,000
10, 000	300,000,000
100, 000	3,000,000,000

# of parameters become huge even if we only have 1 layer !

Spatial Structure

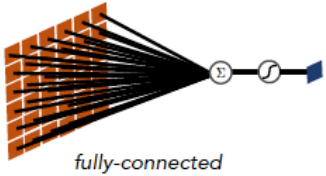
UC San Diego

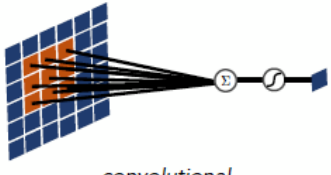
- We can exploit the spatial structure of image data to reduce the amount of learning



✓ nearby areas tend to contain stronger patterns

✓ nearby pixels tend to share similar characteristics and are combined in certain ways





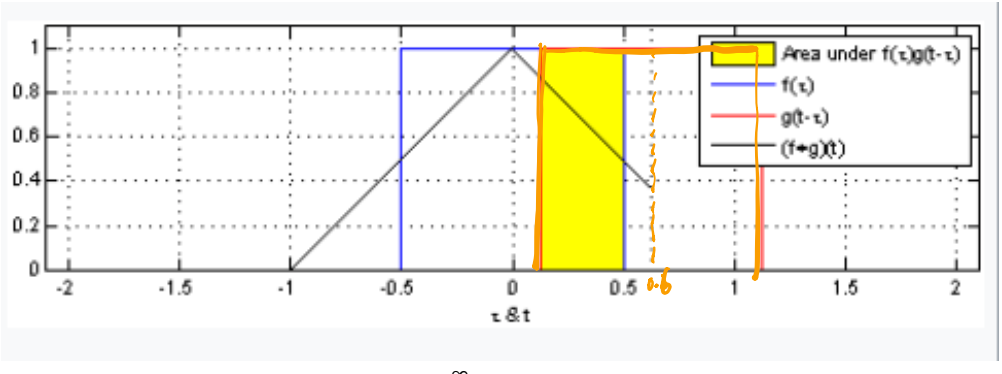
weight savings

Convolution

UC San Diego

- **Convolution** is a mathematical operation on two functions (f and g) that produces a third function ($f * g$) that expresses how the shape of one is modified by the other.

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau = (g * f)(t)$$

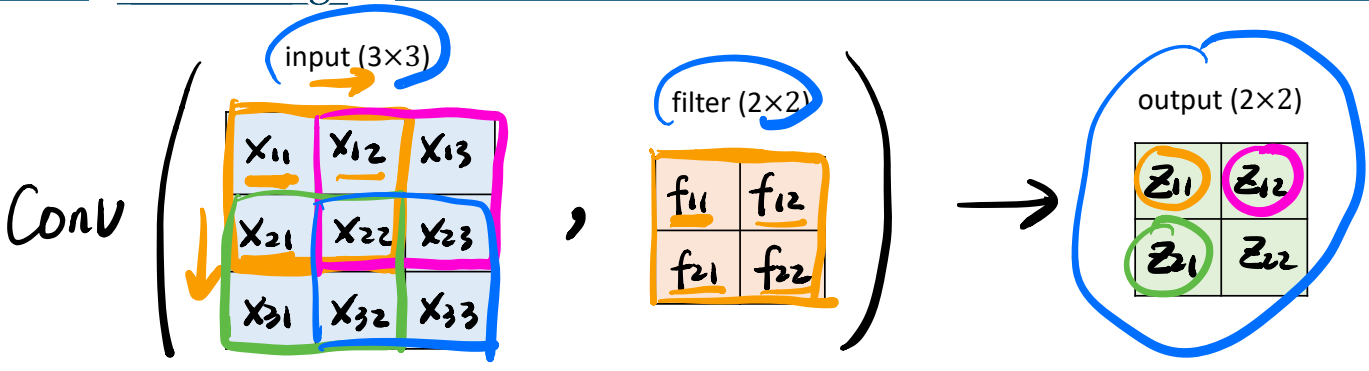


The amount of yellow is the area of $\int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau$ for the current value of t .

https://en.wikipedia.org/wiki/Convolution

Convolution in Neural Networks

UC San Diego



$$z_{11} = f_{11}x_{11} + f_{12}x_{12} + f_{21}x_{21} + f_{22}x_{22}$$

$$z_{12} = f_{11}x_{12} + f_{12}x_{13} + f_{21}x_{22} + f_{22}x_{23}$$

$$z_{21} = f_{11}x_{21} + f_{12}x_{22} + f_{21}x_{31} + f_{22}x_{32}$$

$$z_{22} = f_{11}x_{22} + f_{12}x_{23} + f_{21}x_{32} + f_{22}x_{33}$$

Convolution Example: Vertical Edge Detection

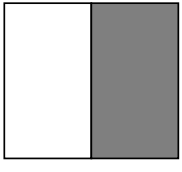
UC San Diego

Conv

Input (6×6)

10	10 ×1	10 ×0	0 ×-1	0	0
10	10 ×1	10 ×0	0 ×-1	0	0
10	10 ×1	10 ×0	0 ×-1	0	0
10	10	10	0 ×1	0 ×0	0 ×-1
10	10	10	0 ×1	0 ×0	0 ×-1
10	10	10	0 ×1	0 ×0	0 ×-1

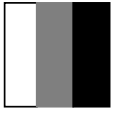
×



Filter (3×3)

1	0	-1
1	0	-1
1	0	-1

f

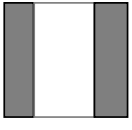


=

Output (4×4)

0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0

=



Convolution Example: Horizontal Edge Detection

UC San Diego

Input (6×6)

10	10	10	10	10	10
10	10	10	10	10	10
10	10	10	10	10	10
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

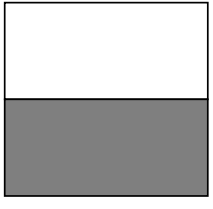
Filter (3×3)

1	1	1
0	0	0
-1	-1	-1

=

Output (4×4)

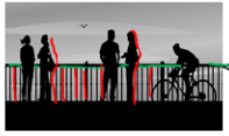
0	0	0	0
30	30	30	30
30	30	30	30
0	0	0	0



=

Learning to Detect Edges

UC San Diego



vertical edges

1	0	-1
1	0	-1
1	0	-1

horizontal edges

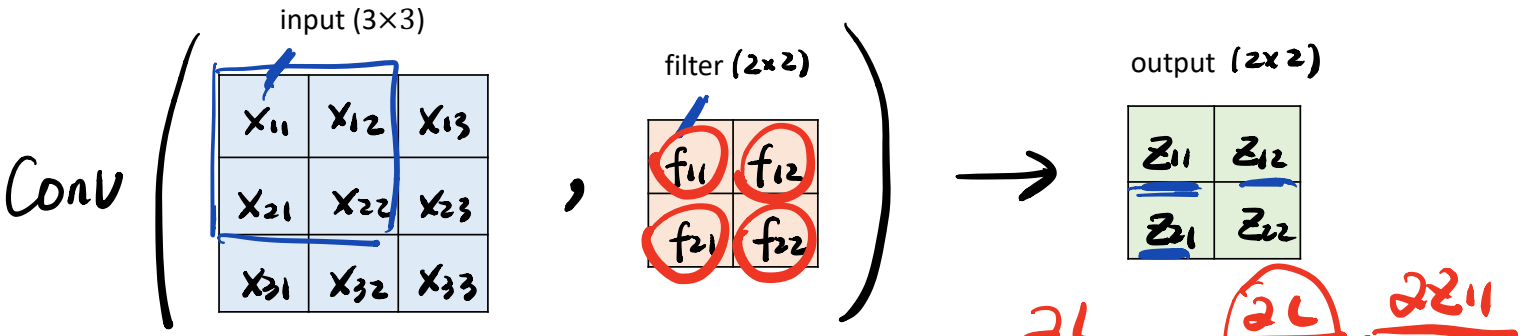
1	1	1
0	0	0
-1	-1	-1

f_{11}	f_{12}	f_{13}
f_{21}	f_{22}	f_{23}
f_{31}	f_{32}	f_{33}

Filter (3×3)

Forward Computation of Convolutional Layer

UC San Diego



$$z_{11} = f_{11}x_{11} + f_{12}x_{12} + f_{21}x_{21} + f_{22}x_{22}$$

$$z_{12} = f_{11}x_{12} + f_{12}x_{13} + f_{21}x_{22} + f_{22}x_{23}$$

$$z_{21} = f_{11}x_{21} + f_{12}x_{22} + f_{21}x_{31} + f_{22}x_{32}$$

$$z_{22} = f_{11}x_{22} + f_{12}x_{23} + f_{21}x_{32} + f_{22}x_{33}$$

$$\frac{\partial L}{\partial f_{11}} = \frac{\partial L}{\partial z_{11}} \cdot \frac{\partial z_{11}}{\partial f_{11}} + \frac{\partial L}{\partial z_{12}} \cdot \frac{\partial z_{12}}{\partial f_{11}} + \frac{\partial L}{\partial z_{21}} \cdot \frac{\partial z_{21}}{\partial f_{11}} + \frac{\partial L}{\partial z_{22}} \cdot \frac{\partial z_{22}}{\partial f_{11}}$$

Backward Computation of Convolutional Layer

UC San Diego

- Now, Compute the derivatives of loss "L" with respect to filter "f", assume. $dZ = \frac{\partial L}{\partial Z}$ known

$$\frac{\partial L}{\partial f_{11}} = \frac{\partial L}{\partial z_{11}} \cdot \frac{\partial z_{11}}{\partial f_{11}} + \frac{\partial L}{\partial z_{12}} \cdot \frac{\partial z_{12}}{\partial f_{11}} + \frac{\partial L}{\partial z_{21}} \cdot \frac{\partial z_{21}}{\partial f_{11}} + \frac{\partial L}{\partial z_{22}} \cdot \frac{\partial z_{22}}{\partial f_{11}}$$

$$\frac{\partial L}{\partial f_{12}} = \frac{\partial L}{\partial z_{11}} \cdot \frac{\partial z_{11}}{\partial f_{12}} + \frac{\partial L}{\partial z_{12}} \cdot \frac{\partial z_{12}}{\partial f_{12}} + \frac{\partial L}{\partial z_{21}} \cdot \frac{\partial z_{21}}{\partial f_{12}} + \frac{\partial L}{\partial z_{22}} \cdot \frac{\partial z_{22}}{\partial f_{12}}$$

$$\frac{\partial L}{\partial f_{21}} = \frac{\partial L}{\partial z_{11}} \cdot \frac{\partial z_{11}}{\partial f_{21}} + \frac{\partial L}{\partial z_{12}} \cdot \frac{\partial z_{12}}{\partial f_{21}} + \frac{\partial L}{\partial z_{21}} \cdot \frac{\partial z_{21}}{\partial f_{21}} + \frac{\partial L}{\partial z_{22}} \cdot \frac{\partial z_{22}}{\partial f_{21}}$$

$$\frac{\partial L}{\partial f_{22}} = \frac{\partial L}{\partial z_{11}} \cdot \frac{\partial z_{11}}{\partial f_{22}} + \frac{\partial L}{\partial z_{12}} \cdot \frac{\partial z_{12}}{\partial f_{22}} + \frac{\partial L}{\partial z_{21}} \cdot \frac{\partial z_{21}}{\partial f_{22}} + \frac{\partial L}{\partial z_{22}} \cdot \frac{\partial z_{22}}{\partial f_{22}}$$

- Which can be written as,

$$\frac{\partial L}{\partial f_{11}} = \frac{\partial L}{\partial z_{11}} x_{11} + \frac{\partial L}{\partial z_{12}} x_{12} + \frac{\partial L}{\partial z_{21}} \cdot x_{21} + \frac{\partial L}{\partial z_{22}} \cdot x_{22}$$

$$\frac{\partial L}{\partial f_{12}} = \frac{\partial L}{\partial z_{11}} \cdot x_{12} + \frac{\partial L}{\partial z_{12}} \cdot x_{13} + \frac{\partial L}{\partial z_{21}} \cdot x_{22} + \frac{\partial L}{\partial z_{22}} \cdot x_{23}$$

$$\frac{\partial L}{\partial f_{21}} = \frac{\partial L}{\partial z_{11}} \cdot x_{21} + \frac{\partial L}{\partial z_{12}} \cdot x_{22} + \frac{\partial L}{\partial z_{21}} \cdot x_{31} + \frac{\partial L}{\partial z_{22}} \cdot x_{32}$$

$$\frac{\partial L}{\partial f_{22}} = \frac{\partial L}{\partial z_{11}} \cdot x_{22} + \frac{\partial L}{\partial z_{12}} \cdot x_{23} + \frac{\partial L}{\partial z_{21}} \cdot x_{32} + \frac{\partial L}{\partial z_{22}} \cdot x_{33}$$

Backward Computation of Convolutional Layer

UC San Diego

- The derivative can also be written in the form of convolution!

$$\begin{bmatrix} \frac{\partial L}{\partial f_{11}} & \frac{\partial L}{\partial f_{12}} \\ \frac{\partial L}{\partial f_{21}} & \frac{\partial L}{\partial f_{22}} \end{bmatrix} = \text{Conv}\left(\begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{bmatrix}, \begin{bmatrix} \frac{\partial L}{\partial z_{11}} & \frac{\partial L}{\partial z_{12}} \\ \frac{\partial L}{\partial z_{21}} & \frac{\partial L}{\partial z_{22}} \end{bmatrix} \right)$$

Similarly, we can find the derivatives of the input matrix X w.r.t. error L

$$\frac{\partial L}{\partial X_{11}} = \frac{\partial L}{\partial Z_{11}} \cdot \frac{\partial Z_{11}}{\partial X_{11}} + \frac{\partial L}{\partial Z_{12}} \cdot \frac{\partial Z_{12}}{\partial X_{11}} + \frac{\partial L}{\partial Z_{21}} \cdot \frac{\partial Z_{21}}{\partial X_{11}} + \frac{\partial L}{\partial Z_{22}} \cdot \frac{\partial Z_{22}}{\partial X_{11}}$$

$$= \frac{\partial L}{\partial Z_{11}} \cdot f_{11} + \frac{\partial L}{\partial Z_{12}} \cdot 0 + \frac{\partial L}{\partial Z_{21}} \cdot 0 + \frac{\partial L}{\partial Z_{22}} \cdot 0$$

$$\frac{\partial L}{\partial X_{12}} = \frac{\partial L}{\partial Z_{11}} \cdot f_{12} + \frac{\partial L}{\partial Z_{21}} f_{11}$$

Backward Computation of Convolutional Layer

UC San Diego

$$\frac{\partial L}{\partial x_{13}} = \frac{\partial L}{\partial z_{12}} \cdot f_{12}$$

$$\frac{\partial L}{\partial x_{21}} = \frac{\partial L}{\partial z_{11}} \cdot f_{21} + \frac{\partial L}{\partial z_{21}} \cdot f_{11}$$

$$\frac{\partial L}{\partial x_{22}} = \frac{\partial L}{\partial z_{11}} \cdot f_{22} + \frac{\partial L}{\partial z_{12}} \cdot f_{21} + \frac{\partial L}{\partial z_{21}} \cdot f_{12} + \frac{\partial}{\partial z_{22}} \cdot f_{11}$$

$$\frac{\partial L}{\partial x_{23}} = \frac{\partial L}{\partial z_{12}} \cdot f_{22} + \frac{\partial L}{\partial z_{22}} \cdot f_{11}$$

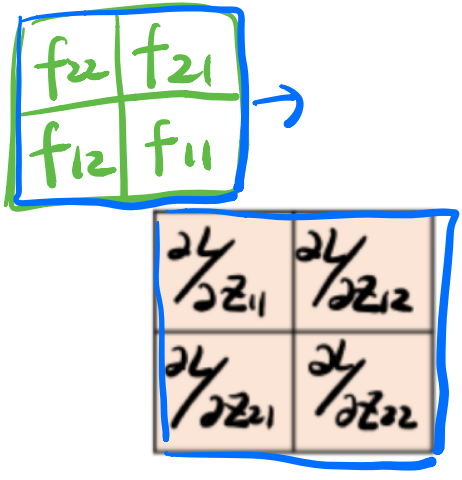
$$\frac{\partial L}{\partial x_{31}} = \frac{\partial L}{\partial z_{21}} \cdot f_{21}$$

$$\frac{\partial L}{\partial x_{32}} = \frac{\partial L}{\partial z_{21}} \cdot f_{22} + \frac{\partial L}{\partial z_{22}} \cdot f_{21}$$

$$\frac{\partial L}{\partial x_{33}} = \frac{\partial L}{\partial z_{22}} \cdot f_{22}$$

Backward Computation of Convolutional Layer

UC San Diego



(full convolution)

$$\frac{\partial L}{\partial x_{11}} = f_{11} \cdot \frac{\partial L}{\partial z_{11}}$$

$$\frac{\partial L}{\partial x_{12}} = f_{12} \frac{\partial L}{\partial z_{11}} + f_{11} \cdot \frac{\partial L}{\partial z_{12}}$$

$$\frac{\partial L}{\partial x_{13}} = f_{12} \cdot \frac{\partial L}{\partial z_{12}}$$

$$\frac{\partial L}{\partial x_{21}} = f_{21} \cdot \frac{\partial L}{\partial z_{11}} + f_{11} \frac{\partial L}{\partial z_{21}}$$

$$\frac{\partial L}{\partial x_{22}} = f_{22} \cdot \frac{\partial L}{\partial z_{11}} + f_{21} \frac{\partial L}{\partial z_{12}} + f_{12} \cdot \frac{\partial L}{\partial z_{21}} + f_{11} \cdot \frac{\partial L}{\partial z_{22}}$$

$$\frac{\partial L}{\partial x_{23}} = f_{22} \cdot \frac{\partial L}{\partial z_{12}} + f_{12} \cdot f_{12} \cdot \frac{\partial L}{\partial z_{22}}$$

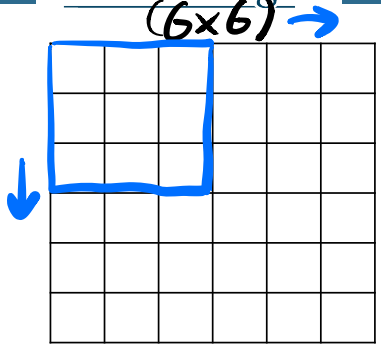
$$\frac{\partial L}{\partial x_{31}} = f_{21} \cdot \frac{\partial L}{\partial z_{21}}$$

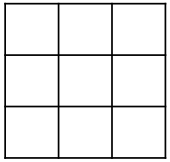
$$\frac{\partial L}{\partial x_{32}} = f_{22} \cdot \frac{\partial L}{\partial z_{21}} + f_{21} \frac{\partial L}{\partial z_{22}}$$

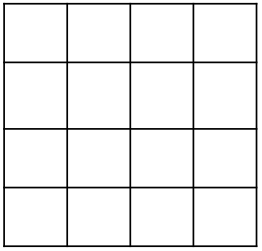
$$\frac{\partial L}{\partial x_{33}} = f_{22} \cdot \frac{\partial L}{\partial z_{22}}$$

Padding

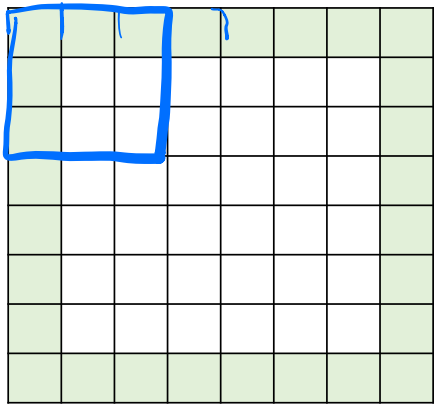
UC San Diego

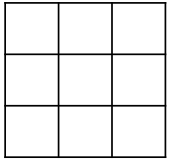




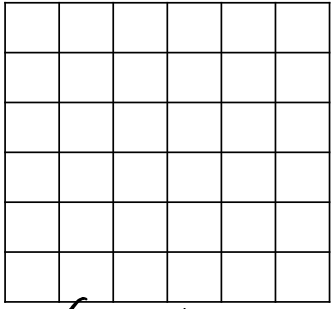


- dimension shrinks





- padding zero

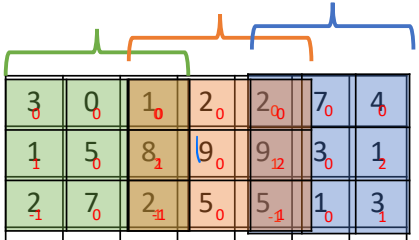


Same padding:
dimension stays the same after Convolution

Strided Convolution

UC San Diego

- Only compute output at some integer interval to downsample image



3	0	1	2	2	7	4
1	5	8	9	9	3	1
2	7	2	5	5	1	3
0	1	3	1	1	7	8
4	2	1	6	6	2	8
2	4	5	2	2	3	9
1	3	4	5	7	8	3

Input (7×7)

0	0	0
1	0	2
-1	0	1

Filter (3×3)

Stride 2

=

17	29	9

Output (3×3)

Summary of Convolutions

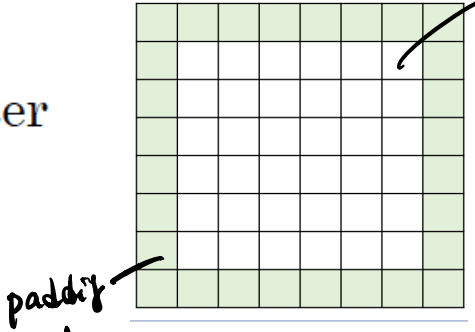
UC San Diego

$n \times n$ image

$f \times f$ filter

padding p

stride s



0	0	0
1	0	2
-1	0	1

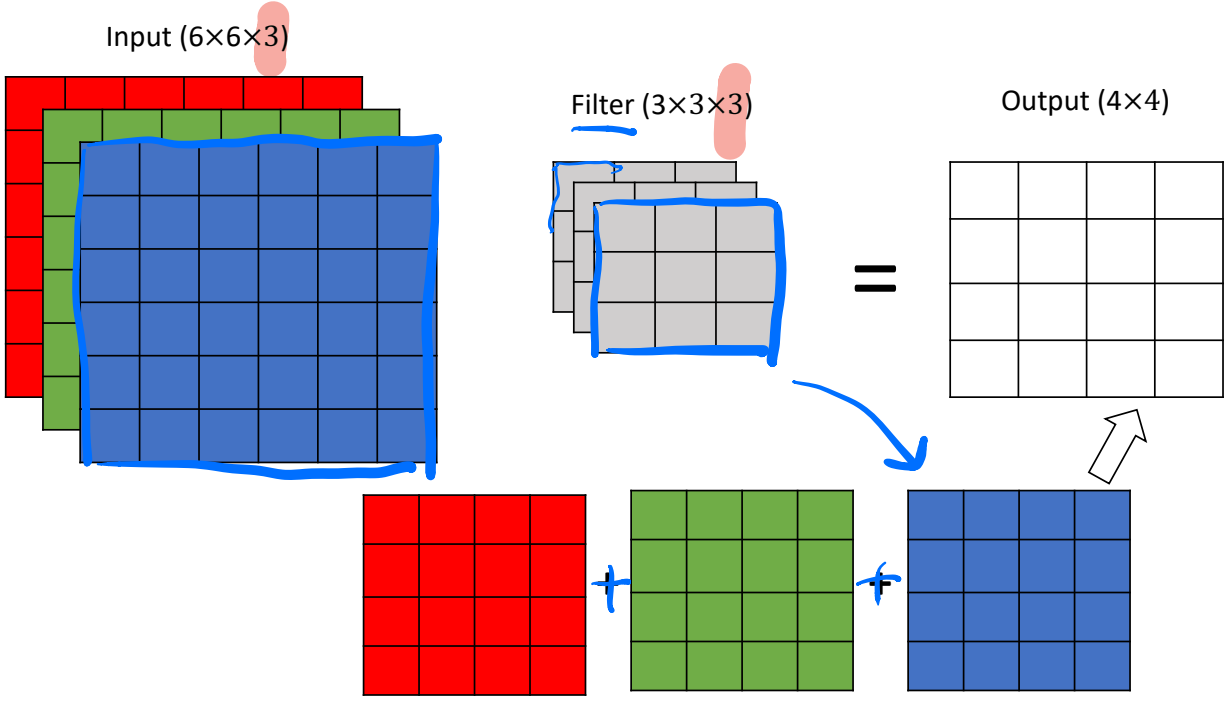
Filter (3×3)

$$\Rightarrow \left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor$$

floor function

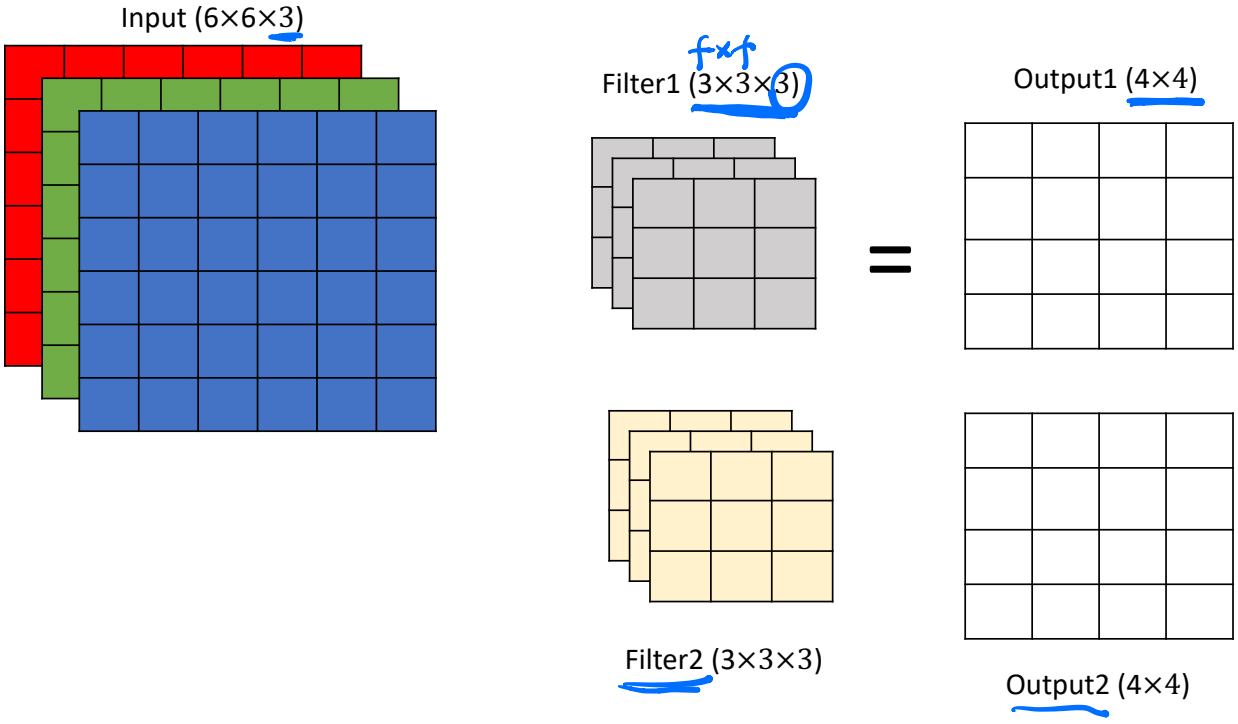
Convolutions on RGB images

UC San Diego



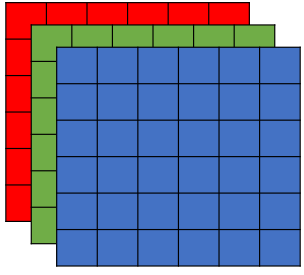
More Filters

UC San Diego



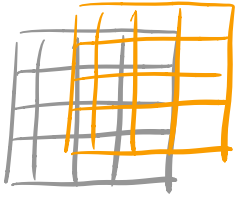
One layer of Convolutional Neural Nets

UC San Diego



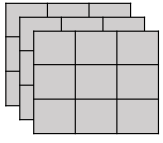
6x6x3

↓ 1 layer of CNN



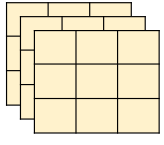
4x4x2

filter 1



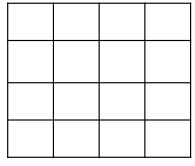
3x3x3

filters



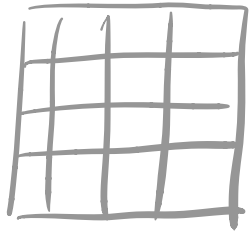
3x3x3

ReLU

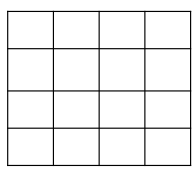


4x4

$+ b_1$ (R)



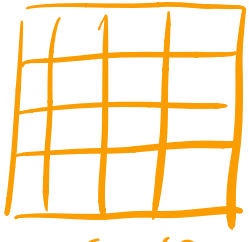
4x4



4x4

$+ b_2$ (R)

ReLU



4x4

use as input for next Convolutional layer

Summary of Notation

UC San Diego

For 1 Convolutional layer:

$f^{[l]}$ = filter size

$p^{[l]}$ = padding

$s^{[l]}$ = stride

$n_c^{[l]}$ = number of conv filters

Input: $n_H^{[l-1]} \times n_W^{[l-1]} \times n_c^{[l-1]}$

Output: $n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$

Parameters in each filter

$f^{[l]} \times f^{[l]} \times n_c^{[l-1]}$

Total # of parameters in 1 conv layer

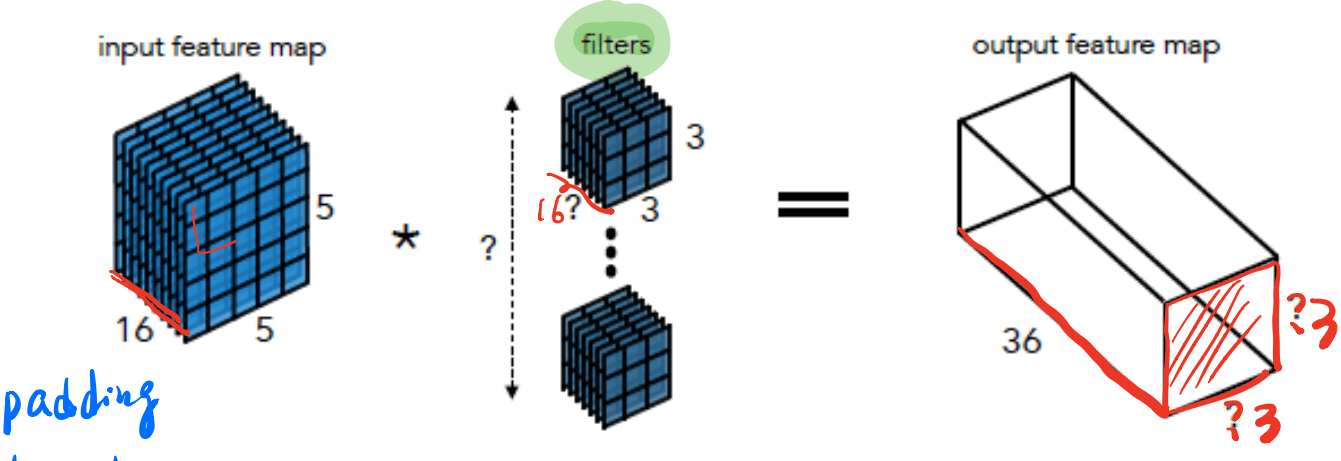
$f^{[l]} \times f^{[l]} \times n_c^{[l-1]} \times n_c^{[l]}$

number of filters

$$n_H^{[l]} = \left\lfloor \frac{n_H^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor \qquad n_W^{[l]} = \left\lfloor \frac{n_W^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor$$

Pop Quiz

UC San Diego



no padding

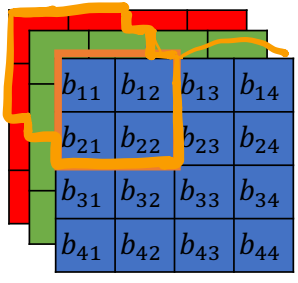
stride = 1

- What is the size of each filter? 3×3×16
- How many filters do we have? 36

Implementation

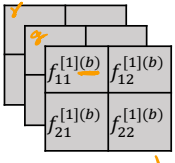
UC San Diego

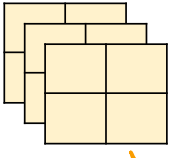
Input (4×4×3)

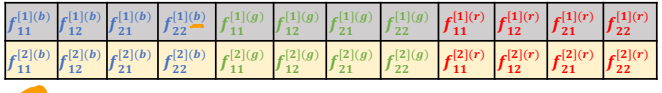


X

Filter1 (2×2×3)







W

Filter2 (2×2×3)

stride = 2

- Take each [2×2×3] blocks of pixels in the input and stretch each block into a column vector of size 12
- Iterating this process with s = 2 gives (4-2)/2+1 = 2 locations along both width and height, leading to a input matrix with size [12×4]
- The weights of the CONV layer are similarly stretched out into rows. Since we have 2 filters of size [2×2×3], this gives us a matrix W of size [2×12]
- We can express convolution as WX
- Output of this operation is [2×4]
- We can reshape the output to [2×2×2]

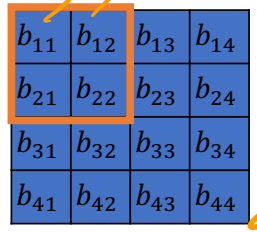
b_{11}	b_{13}	b_{31}	b_{33}
b_{12}	b_{14}	b_{32}	b_{34}
b_{21}	b_{23}	b_{41}	b_{43}
b_{22}	b_{24}	b_{42}	b_{44}
g_{11}	g_{13}	g_{31}	g_{33}
g_{12}	g_{14}	g_{32}	g_{34}
g_{21}	g_{23}	g_{41}	g_{43}
g_{22}	g_{24}	g_{42}	g_{44}
r_{11}	r_{13}	r_{31}	r_{33}
r_{12}	r_{14}	r_{32}	r_{34}
r_{21}	r_{23}	r_{41}	r_{43}
r_{22}	r_{24}	r_{42}	r_{44}

X_{col}

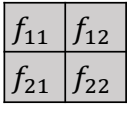
Convolution v.s. Fully Connected Layer

UC San Diego

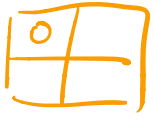
Input (consider 1 channel)



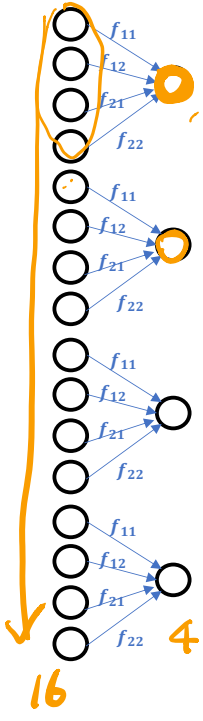
X



consider 1 filter



Convolution



local receptive field

In convolution layers, we enforce the following prior on learning:

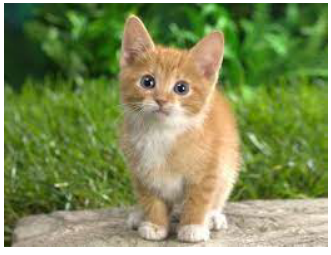
- Connections are restricted to nearby regions
- Same filters (weights) are applied throughout the input
- We only have **4** rather than 64 parameters to learn!

weight sharing

16x4=64

Convolution v.s. Fully Connected Layer

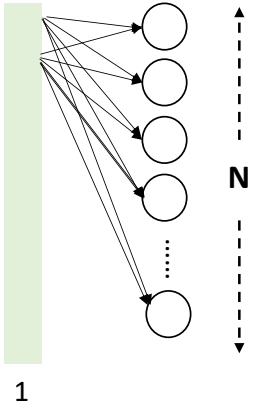
UC San Diego



100*100*3 = 30, 000

Reshape

30, 000



1 Layer of MLP

N	# parameters
1, 000	30,000,000
10, 000	300,000,000
100, 000	3,000,000,000

- If you have 100 filters that are 3x3x3 in one layer of a neural network, how many parameters does that layer have?

$(27+1)\times 100 = 2800$

Pooling Layer- Max pooling

UC San Diego

Why pooling?

- Reduce feature dimension ⇒ reduce # of learn parameters
- Summarize features in a local region (robust to variations)

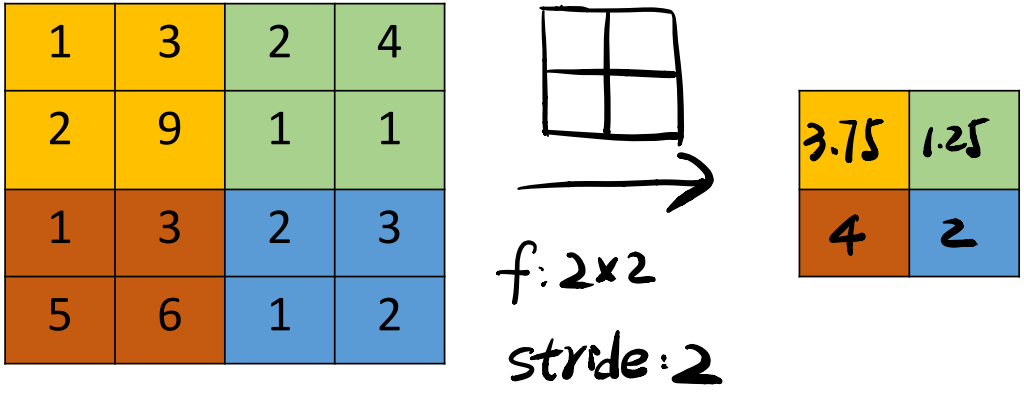
1	3	2	4
2	9	1	1
1	3	2	3
5	6	1	2

f: 2×2

stride: 2

9	4
6	3

Pooling Layer- Average pooling



☆ No parameters to be learned

Hyperparameters:

f : filter size

s : stride

Max or average pooling

✓ $f=2, s=2$ (shrink input size by 2)

✓ $f=3, s=2$

$$n_H \times n_W \times n_C \xrightarrow{\text{after pooling}} \left\lfloor \frac{n_H - f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n_W - f}{s} + 1 \right\rfloor \times n_C$$

Backward Computation of Pooling Layer

UC San Diego

1	3	2	4
2	9	1	1
1	3	2	3
5	6	1	2

max pooling

$$z_{11} = \max(x_{11}, x_{12}, x_{21}, x_{22})$$

$$= 9$$

$$\frac{\partial z_{11}}{\partial x_{11}} = 0, \quad \frac{\partial z_{11}}{\partial x_{12}} = 0$$

$$\frac{\partial z_{11}}{\partial z_{21}} = 0 \quad \frac{\partial z_{11}}{\partial x_{22}} = 1$$

Suppose $z = \max(x_1 \cdots x_n)$

$$\frac{\partial z}{\partial x} = \begin{cases} 1 & \text{if } x_i = \max(x_1 \cdots x_n) \\ 0 & \text{otherwise} \end{cases}$$

Backward Computation of Pooling Layer

UC San Diego

1	3	2	4
2	9	1	1
1	3	2	3
5	6	1	2

average pooling

$$z_{11} = \frac{1}{4}(x_{11} + x_{12} + x_{21} + x_{22})$$

$$\frac{\partial z_{11}}{\partial x_{11}} = \frac{1}{4}$$

$$\frac{\partial z_{11}}{\partial x_{12}} = \frac{1}{4}$$

$$\frac{\partial z_{11}}{\partial x_{21}} = \frac{1}{4}$$

$$\frac{\partial z_{11}}{\partial x_{22}} = \frac{1}{4}$$

Classic Networks

UC San Diego

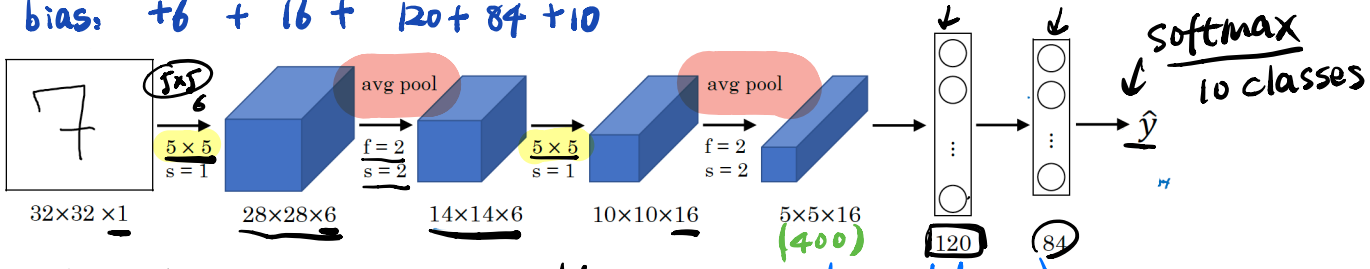
- Classic Networks
  - LeNet – 5
  - AlexNet
  - VGG -16
- ResNet (152 layer)

LeNet - 5

UC San Diego

parameter: 150 + 2400 + 48000 + 10080 + 840

bias: +6 + 16 + 120 + 84 + 10



- $(n_H, n_W)\downarrow$ $n_C\uparrow$, no padding (with padding)
- Conv pool conv pool fc fc output
- Activation function: tanh (V.S. ReLU)
- Average pooling (V.S. max pooling)

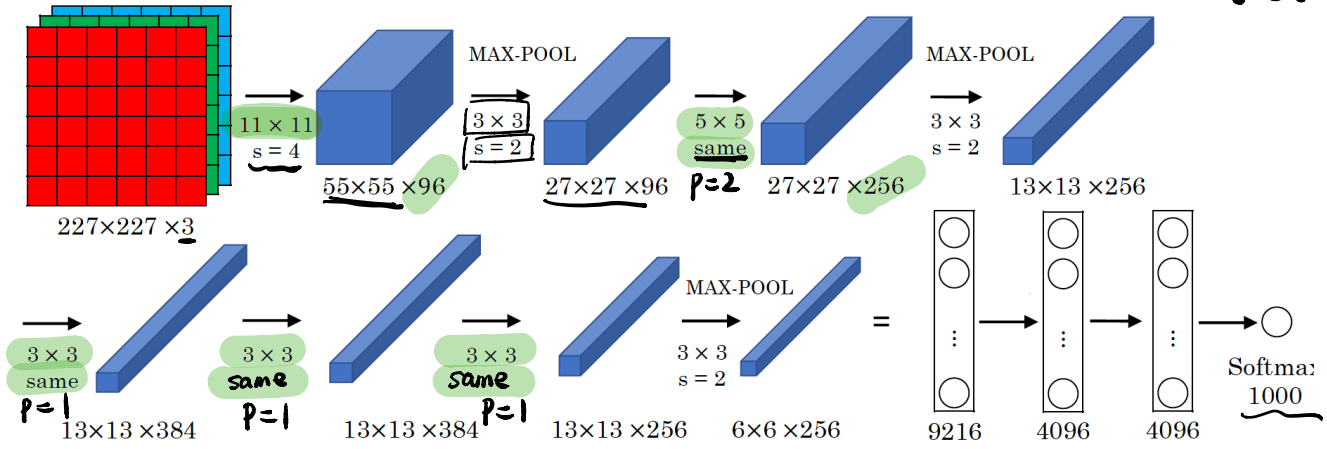
~60k parameters

Lecun et al., 1998. Gradient-based Learning Applied to Document Recognition, https://ieeexplore.ieee.org/document/726791

AlexNet

UC San Diego

- 90 epoch for 6 days in 2 GTX 580 GPUs.
- 2012 Winner: 17% top-5 error on test data



- bigger than LeNet-5 (~60M parameter)
- ReLU, max pooling, padding, + Dropout (5 conv + 3 FC layers)

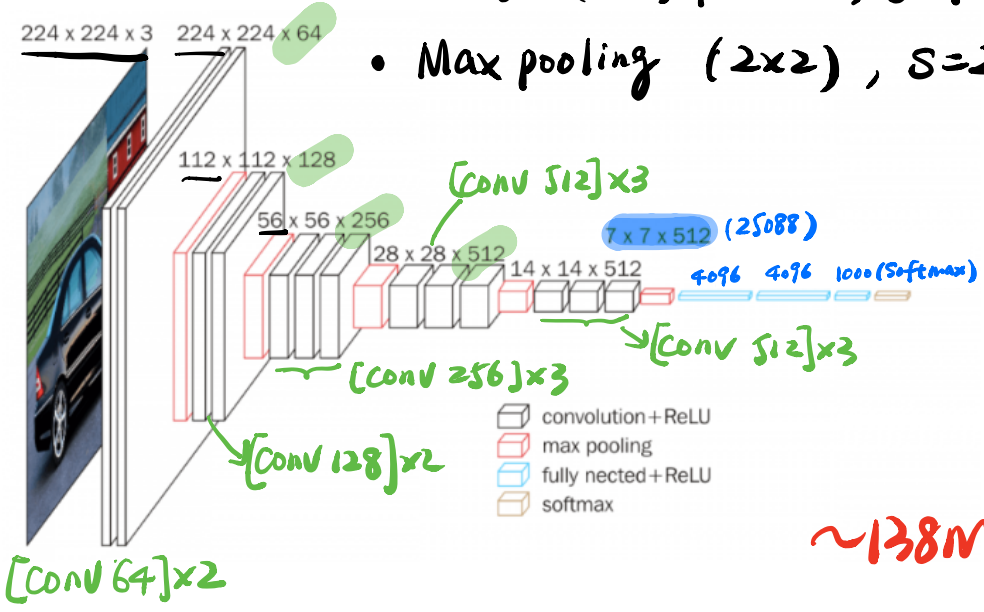
Krizhevsky et al., 2012. ImageNet Classification with Deep Convolutional Neural Networks, https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

VGG -16

UC San Diego

1st place for 2014 ImageNet challenge (7.32% top 5 error on test data)

- Conv = (3x3) filters, S=1 (keep same dimension)
- Max pooling (2x2), S=2



~138M parameter

Simonyan et al., 2014. Very Deep Convolutional Networks for Large-Scale Image Recognition, https://arxiv.org/abs/1409.1556

Residual Network (ResNet)

UC San Diego

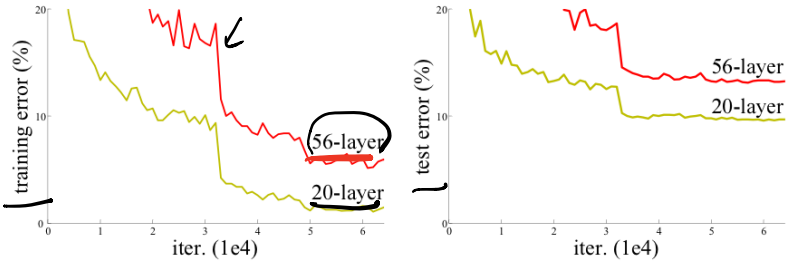
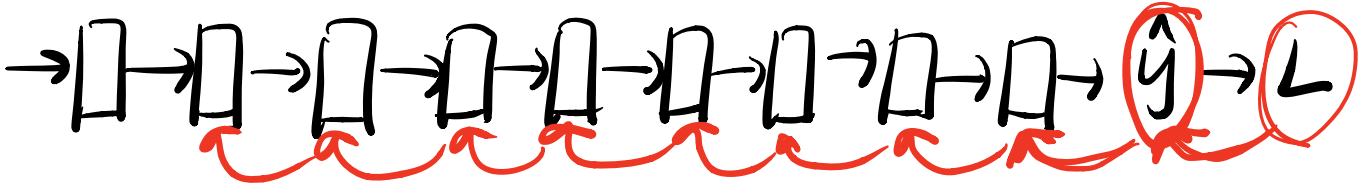


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

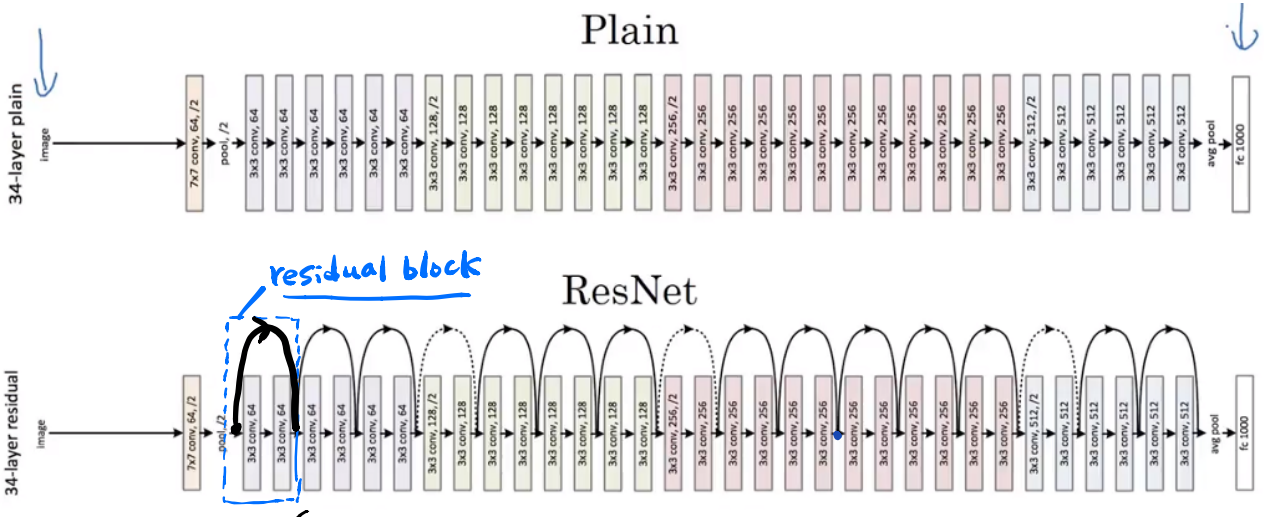
- Before ResNet, deeper network does not quarantee better performance
- Gradient vanishing after many layers



He et al., 2015. Deep Residual Learning for Image Recognition, https://arxiv.org/abs/1512.03385

Residual Network (ResNet)

UC San Diego



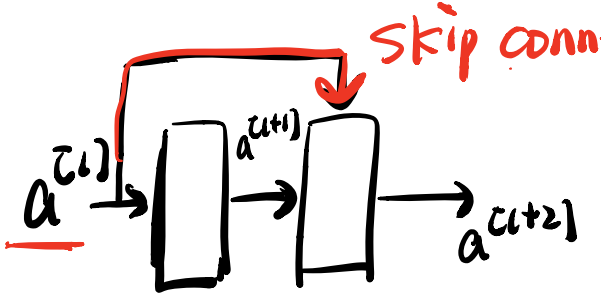
☆ 1st place in 2015 ImageNet (3.57% test error (ensemble of 5 models) in ImageNet test set using a 152-layer ResNet)

☆ Object detection: 28% improvement over VGG-16

He et al., 2015. Deep Residual Learning for Image Recognition, https://arxiv.org/abs/1512.03385

Residual Network (ResNet)

UC San Diego



$$z^{[l+1]} = W^{[l+1]} a^{[l]} + b^{[l+1]}$$
$$a^{[l+1]} = g(z^{[l+1]})$$
$$z^{[l+2]} = W^{[l+2]} a^{[l+1]} + b^{[l+2]}$$
$$a^{[l+2]} = g(z^{[l+2]})$$

Standard NN

↓ Residual block

$$z^{[l+2]} = W^{[l+2]} a^{[l+1]} + b^{[l+2]}$$
$$a^{[l+2]} = g(z^{[l+2]} + a^{[l]})$$

directly back-propogate gradients to early layers!

He et al., 2015. Deep Residual Learning for Image Recognition, https://arxiv.org/abs/1512.03385

Residual Network (ResNet)

UC San Diego

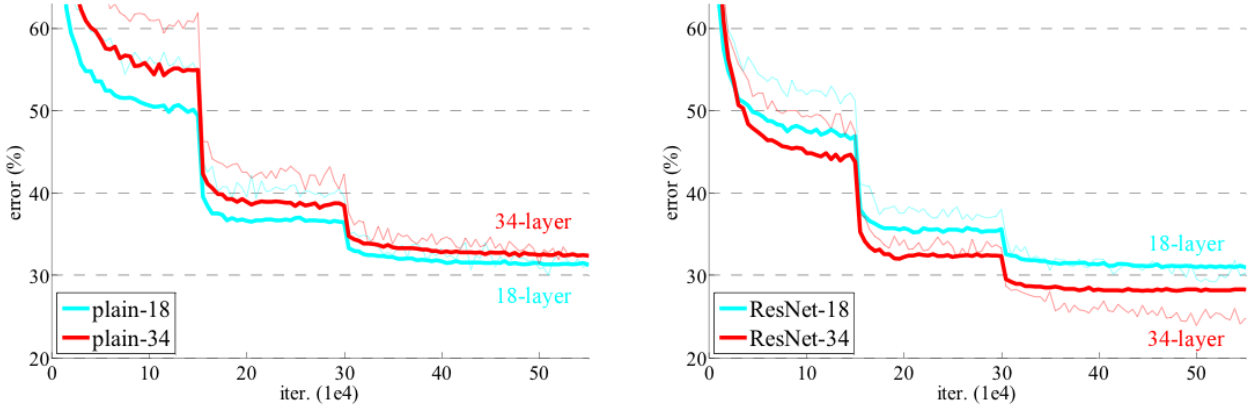


Figure 4. Training on **ImageNet**. Thin curves denote training error, and bold curves denote validation error of the center crops. Left: plain networks of 18 and 34 layers. Right: ResNets of 18 and 34 layers. In this plot, the residual networks have no extra parameter compared to their plain counterparts.

★ ResNet enables training very deep neural networks.

He et al., 2015. Deep Residual Learning for Image Recognition, https://arxiv.org/abs/1512.03385

Practical Tips

UC San Diego

- Use the classic architectures and architecture that works best on ImageNet
- Often times, you can download a pretrained model and finetune/do transfer learning with your own data (e.g., medical imaging or some applications with limited amount of data).

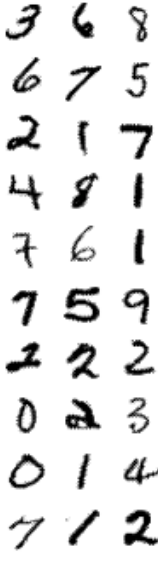
We provide pre-trained models, using the PyTorch `torch.utils.model_zoo`. These can be constructed by passing `pretrained=True`:

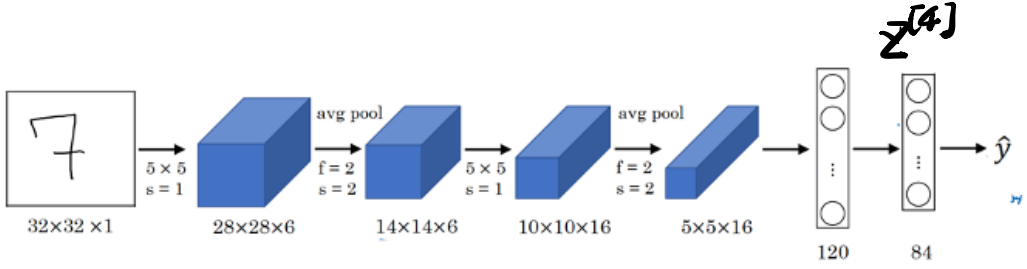
```
import torchvision.models as models
resnet18 = models.resnet18(pretrained=True)
alexnet = models.alexnet(pretrained=True)
squeezenet = models.squeezenet1_0(pretrained=True)
vgg16 = models.vgg16(pretrained=True)
densenet = models.densenet161(pretrained=True)
inception = models.inception_v3(pretrained=True)
googlenet = models.googlenet(pretrained=True)
shufflenet = models.shufflenet_v2_x1_0(pretrained=True)
mobilenet_v2 = models.mobilenet_v2(pretrained=True)
```

https://cs231n.github.io/convolutional-networks
https://pytorch.org/vision/stable/models.html

Multi-class Classification

UC San Diego





$z^{[4]} \in R^{84}$ $\quad w^{[5]} \in R^{10 \times 84}$

$z^{[5]} = w^{[5]} z^{[4]} + b^{[5]}$, $\quad z^{[5]} \in R^{10}$

$\hat{y} = a^{[5]} = \text{Softmax}(z^{[5]})$

Softmax: $\sigma(\vec{z})_i = \dfrac{e^{z_i}}{\sum_{i=1}^{K} e^{z_i}}$

- z: input vector to softmax
- K: number of elments (class) in z

(Continue on softmax function)

$$z = \begin{bmatrix} 9 \\ 5 \\ 0 \end{bmatrix}$$

$$e^{z_1} = e^9 = 8103$$

$$e^{z_2} = e^5 \approx 148$$

$$e^{z_3} = e^0 = 1$$

$$\sum_{j=1}^{K} = e^{z_1} + e^{z_2} + e^{z_3} = 8252$$

Softmax, $\sigma(z)_1 = \frac{8103}{8252} \approx 0.982$, $\sigma(z)_2 = \frac{148}{8252} \approx 0.0179$

$$\sigma(z)_3 = \frac{1}{8252} \approx 0.0001$$

Binary classification

Sigmoid activation

$$\hat{y} = a^{[L]} = \text{sigmoid}(z^{[L]}) = \frac{1}{1+e^{-z^{[L]}}}, \quad z^{[L]} \in R$$

Loss:

$$L = -[y \log \hat{y} + (1-y) \log (1-\hat{y})]$$

Multi-class classification

Softmax activation

$$\hat{y} = a^{[L]} = \text{softmax}(z^{[L]}) = \frac{e^{z_i^{[L]}}}{\sum_{j=1}^{K} e^{z_j^{[L]}}}, \quad z^{[L]} \in R^K$$

Loss

$$L = -\sum_{j=1}^{K} y_j \log \hat{y}_j$$