Deep Learning 2
深度学习 2

Jian Li

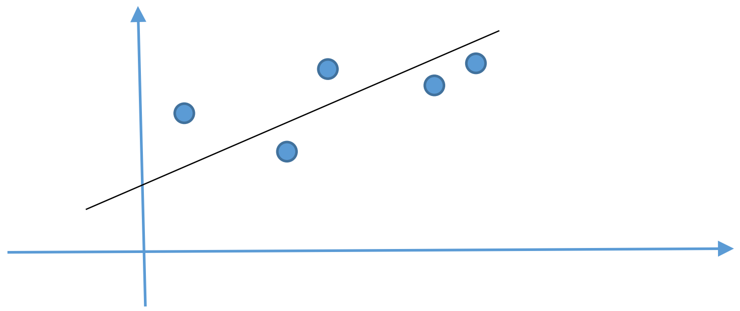
IIIS, Tsinghua

Some Linear Algebra, PCA, Eigenface
线性代数 主成分分析 本征脸

最小二乘法Least Square

问题描述Least square problem (LS)

$$\inf_{x \in R^n} ||Ax - b||_2^2 \quad A \in R^{m \times n}$$ (m points, n dimension)



$$f(x) = (Ax - b)^T(Ax - b) = x^T A^T Ax - 2b^T Ax + b^T b$$

Let $\nabla f = 2A^T Ax - 2A^T b = 0$

$$A^T Ax = A^T b$$

If $\text{rank}(A) = n, A^T A$ is invertible, so $x = (A^T A)^{-1} A^T b$

Moore-Penrose Pseudoinverse if rank(A)=n

Note: if not full col rank, we need to solve the problem in the row subspace.

Can do it via SVD.

如果不是列满秩，我们需要在行空间中解决问题，SVD！

矩阵奇异值分解SVD

Moore-Penrose inverse

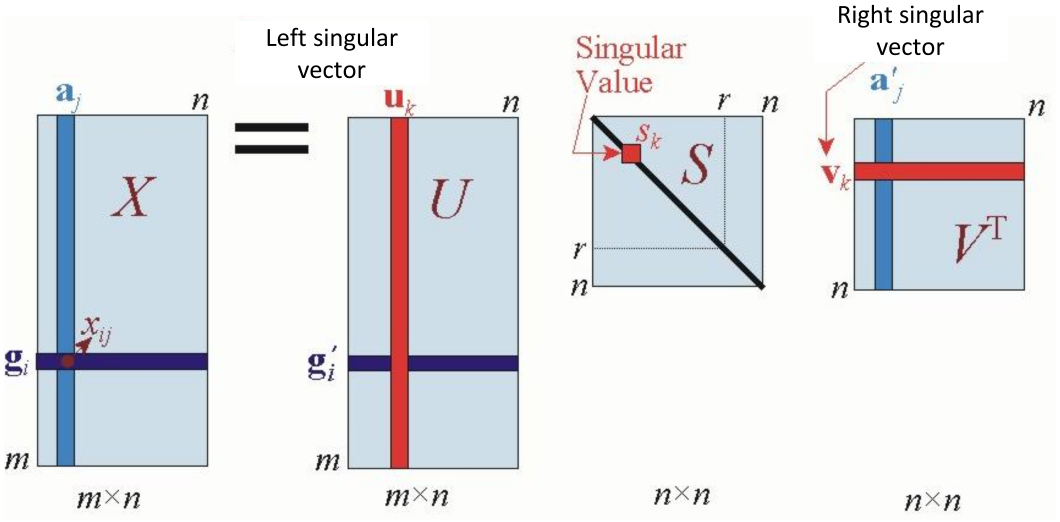
SVD: X $= USV^T$

$X^{inv} = VSU^T$

The solution to LS problem

$$inf\|Xt - b\|$$

is still $X^{inv}b$



Orthonormal rows

$V^TV = I$

行都是正交的

是行空间的一组基

Orthonormal cols

$U^TU = I$

A basis of col(X)

列都是正交的

是列空间的一组基

可以跳过

几何观点Geometric View

Now let us derive it geometrically. Consider the subspace spanned by col of X: col(X). 考虑由X列空间的子空间

So $Xt \in col(X)$. $\|Xt - b\|_2$ is the dist between Xt and b

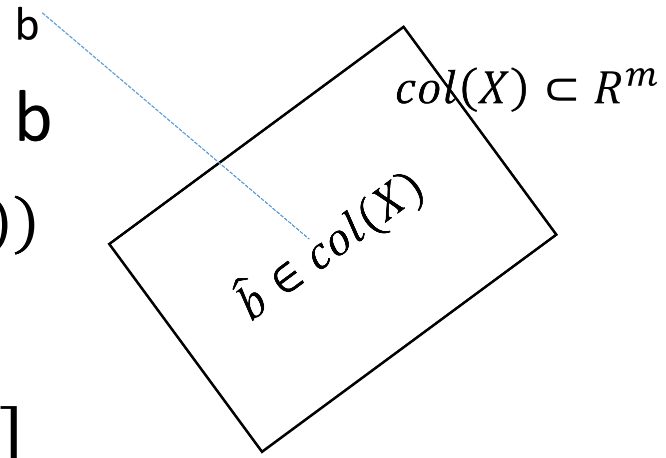
Def: 正交投影算子orthogonal projection operator (onto $col(X)$)

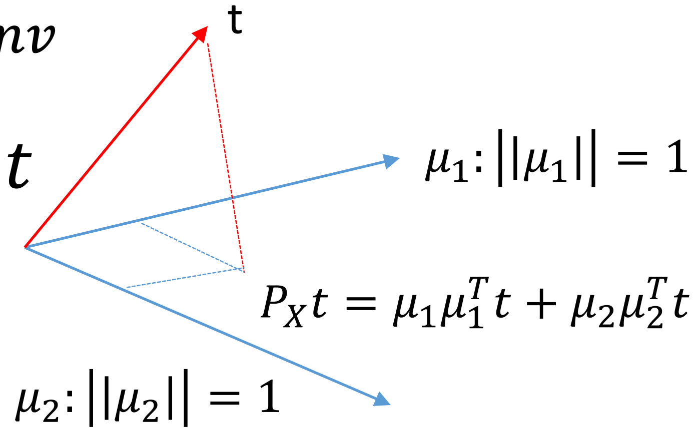
(projection: $PP = P$; orthogonal proj: $P = P^T$)

$$P_X = UU^T [= A(A^T A)^{inv} A^T = \mathrm{USV^T}(\mathrm{VSV^T})\mathrm{VSU^T}]$$

here we use $(A^T A)^{inv} = A^{inv}(A^T)^{inv}$

几何直觉Geometric intuition: consider $P_X t = UU^T t$





可以跳过

几何观点Geometric View

正交性Orthogonality

$t - P_X t = (I - P_X)t$ should be orthogonal to $col(X)$

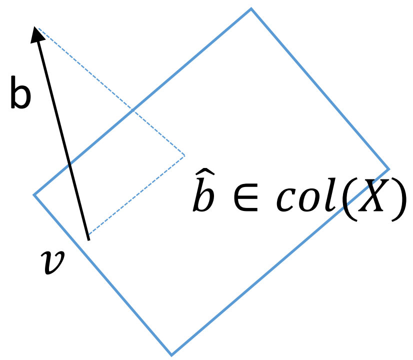
For every μ_i:

$$\mu_i^T(I - P_X)t = \left(\mu_i^T - \mu_i^T U U^T\right)t = \left(\mu_i^T - (0,0,\cdots,1,\cdots,0)U^T\right)t = 0$$

为什么正交投影可以最小化呢Why orthogonal Projection is the minimizer?

for any vector $v \in col(X)$,

$$\|v - b\|_2^2 = \|P_X b + (v - P_X b) - b\|_2^2 = \|P_X b - b\|_2^2 + \|v - P_X b\|_2^2$$



$col(X) \subset R^m$

$\overline{bv}^2 = \overline{b\hat{b}}^2 + \overline{\hat{b}v}^2$

(Pythagoras THM)

主成分分析Principle Component Analysis

- First principle component: the direction that maximizes the variance (which is the first eigen-vector of the covariance matrix X^TX) $X \in R^{m\times n}$ (m points, n dimension)

  第一个主元素，方向是最大化该方向的方差；是协方差矩阵的第一个特征向量

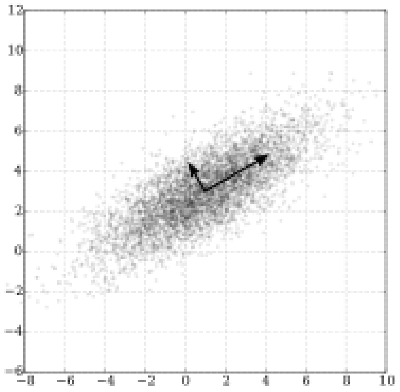
$$\mathbf{w}_{(1)} = \arg\max \left\{ \frac{\mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w}}{\mathbf{w}^T\mathbf{w}} \right\}$$

- 2nd principle component: the direction orthogonal to 1st PC and maximizes the variance

  第二个主元素，与第一个主元素垂直，方向是最大化该方向的方差；是协方差矩阵的第二个特征向量

- Dimension reduction: project to the first few PC

  降维：投影到前几个主元素上



本征脸Eigen-face [Turk, Pentland '91]

- Treat each face as a vector
- Eigen face: just principle components 主成分

1. Detect whether a figure is a face (see the distance from it to the subspace spanned by the first few PC

   检测一个图片是否是个脸

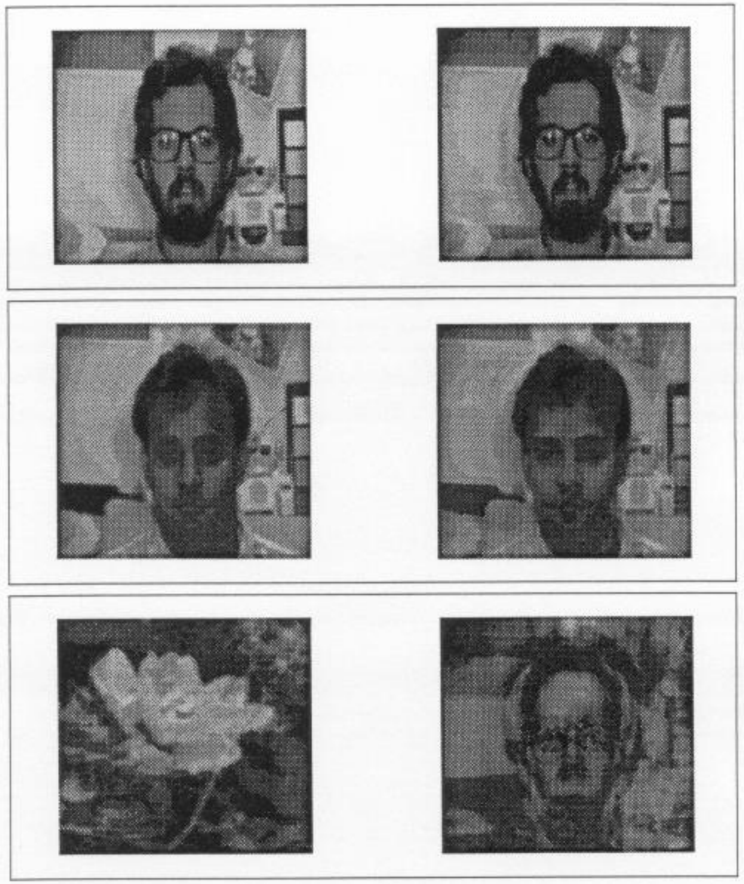


Figure 4. Three images and their projections onto the face space defined by the eigenfaces of Figure 2. The relative measures of distance from face space are **(a)** 29.8, **(b)** 58.5, **(c)** 5217.4. Images **(a)** and **(b)** are in the original training set.

本征脸Eigen-face

1. Detect and locate a face in a figure (like CNN) 检测定位某张脸
2. Tracking movement of a face 跟踪脸的运动
3. Reconstruct occluded image (ask student) 重建有遮挡的图片
    - Dictionary learning 字典学习

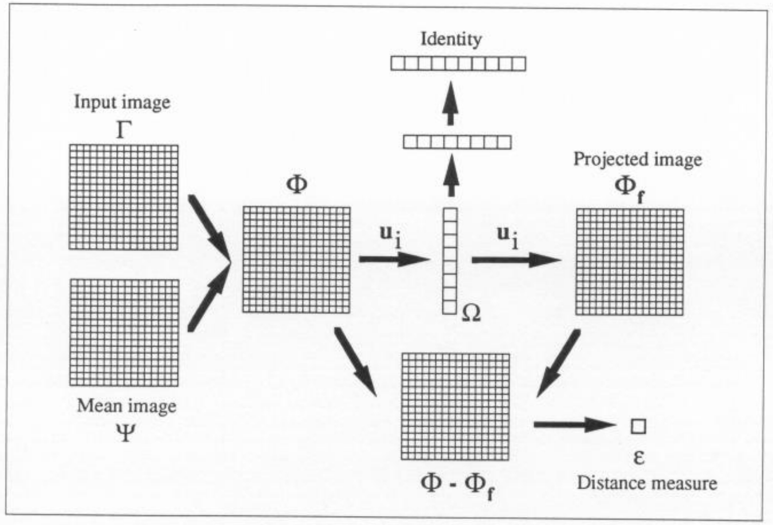


Figure 12. Collection of networks to implement computation of the pattern vector, projection into face space, distance from face space measure, and identification.

Figure 13. **(a)** Partially occluded face image and **(b)** its reconstruction using the eigenfaces.

代码 Code for SVD and PCA

- Let X be the training samples

```
U, S, V = np.linalg.svd(X)
cov = X.T.dot(X)/X.shape[0]
U_cov, _, _ = np.linalg.svd(cov)
X_reduced = X.dot(U_cov[:, :k])
```

SVD for X
SVD分解X

Compute covariance matrix
计算协方差矩阵

SVD for covariance matrix
SVD分界协方差矩阵

Projection to the first k cols of U
PCA for X where k is the number of features that you want to reserve
投影到U的前k列，k是保留的维数

卷积神经网络Convolutional Neural Network

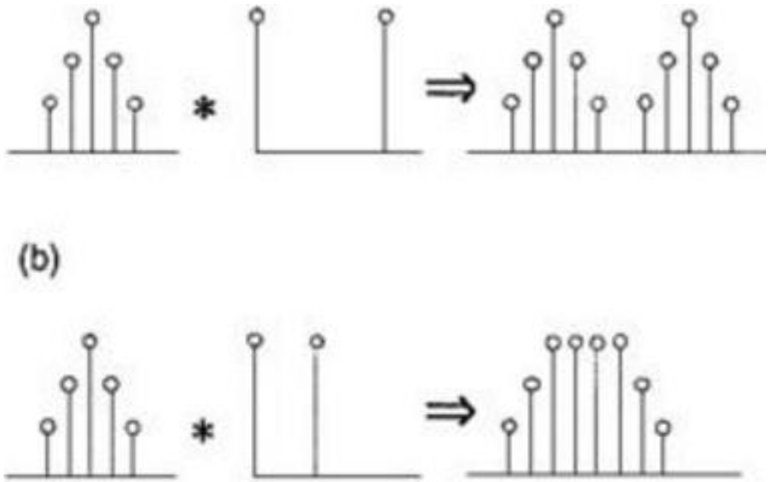
卷积Convolution

- 连续一维卷积1d convolution (continuous):

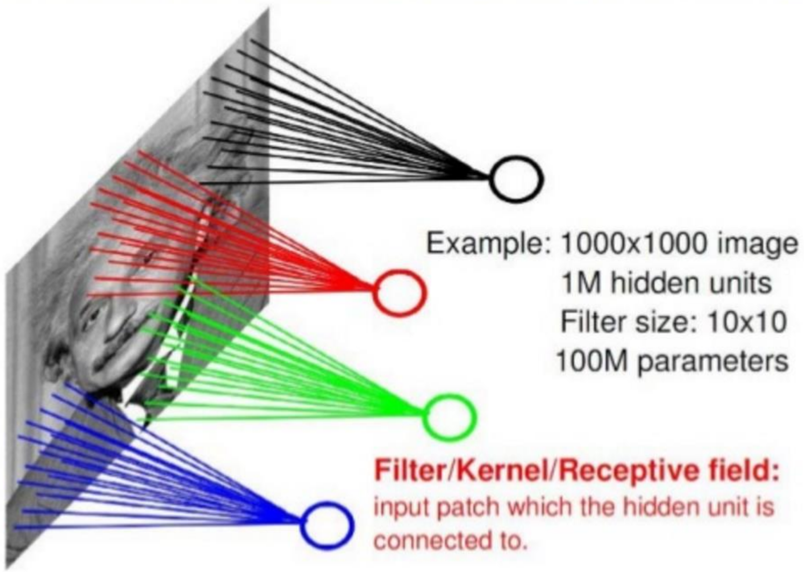
$$s(t) = \int x(a)w(t-a)da \qquad s(t) = (x * w)(t)$$

- 离散一维卷积1d convolution (discrete):

$$s[t] = (x * w)(t) = \sum_{a=-\infty}^{\infty} x[a]w[t-a]$$



卷积Convolution

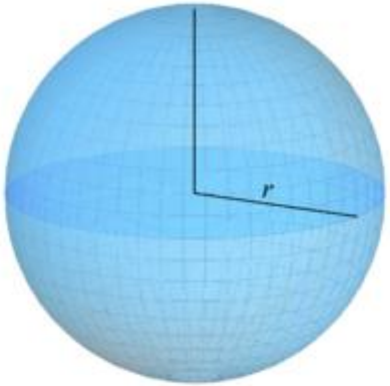


For a 2-D image **H** and a 2-D kernel **F**,

- Convolution Operator: $G = H \star F$

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i-u,j-v]$$

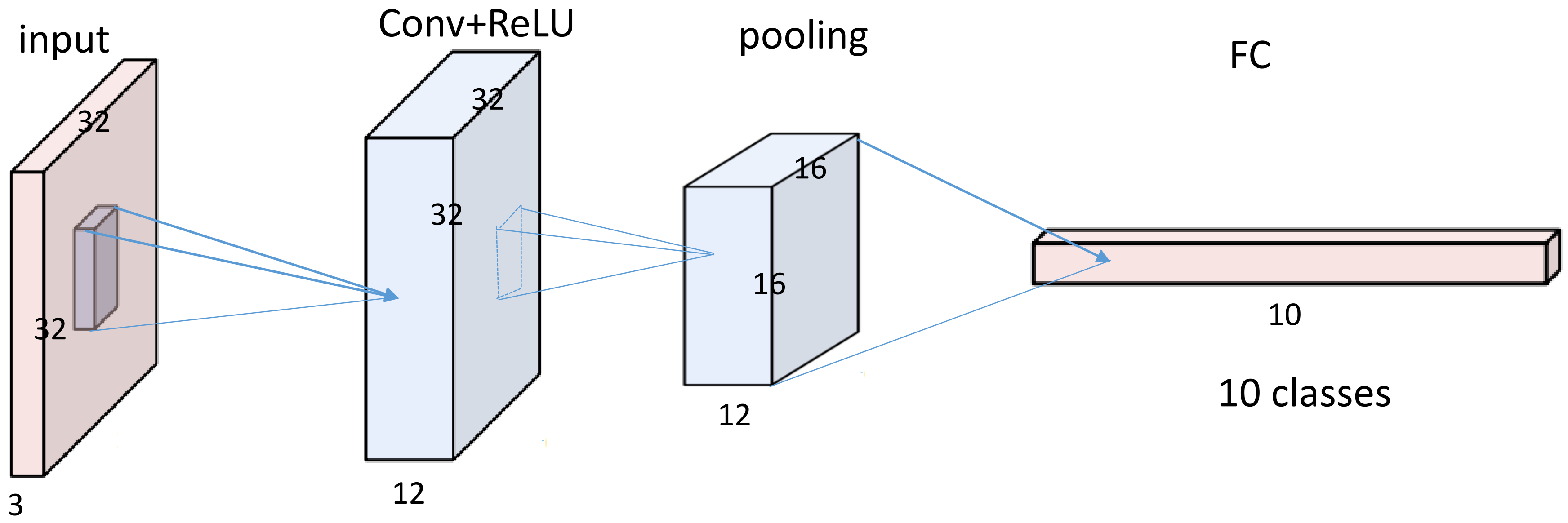
高维空间的随机向量Random Vectors in High Dimension



- Pick two i.i.d. n-dimensional Gaussian N(0,I) X, Y
  在n维高斯分布中采样两个点
  As n becomes large, X and Y are nearly orthogonal (i.e., $< X, Y >\approx 0$)
  随着n变大，两个点基本正交
- Pick two points X, Y uniformly randomly from n-dimensional unit sphere
  在n维球的均匀分布中采样两个点
  As n becomes large, X and Y are nearly orthogonal (i.e., $< X, Y >\approx 0$)
  随着n变大，两个点基本正交
- For two points X, Y, if $< X, Y >$ is far away from 0, they must be highly correlated.
  如果$< X, Y >$不接近0，说明X, Y关联性很强

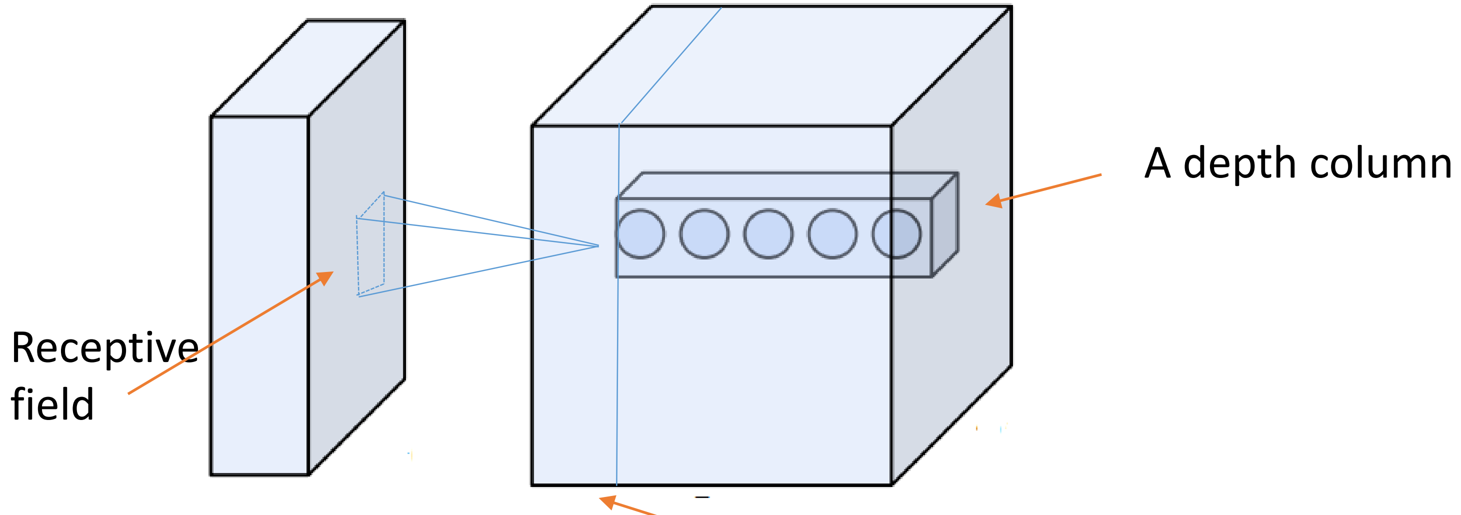
High dimension phenomena – not true in low dimensions 这个是高维空间的现象，在低维空间不成立

基本架构Basic architecture

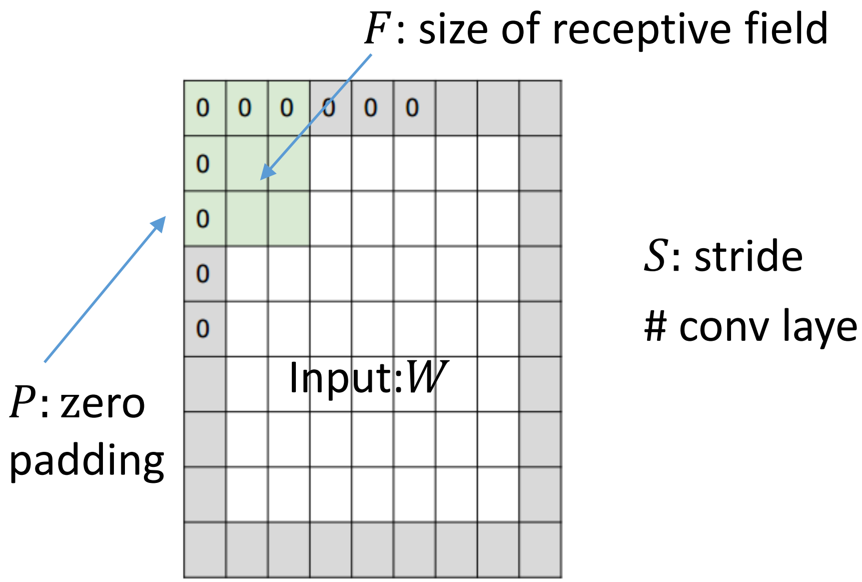


- *Example Architecture: Overview*. We will go into more details below, but a simple ConvNet for CIFAR-10 classification could have the architecture [INPUT - CONV - RELU - POOL - FC]. In more detail:
- INPUT [32x32x3] will hold the raw pixel values of the image, in this case an image of width 32, height 32, and with three color channels R,G,B.
- CONV layer will compute the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and the region they are connected to in the input volume. This may result in volume such as [32x32x12].
- RELU layer will apply an elementwise activation function, such as the max(0,x)max(0,x) thresholding at zero. This leaves the size of the volume unchanged ([32x32x12]).
- POOL layer will perform a downsampling operation along the spatial dimensions (width, height), resulting in volume such as [16x16x12].
- FC (i.e. fully-connected) layer will compute the class scores, resulting in volume of size [1x1x10], where each of the 10 numbers correspond to a class score, such as among the 10 categories of CIFAR-10. As with ordinary Neural Networks and as the name implies, each neuron in this layer will be connected to all the numbers in the previous volume.

http://cs231n.github.io/convolutional-networks/



在输入的边界加零 Zero padding the boundary of input

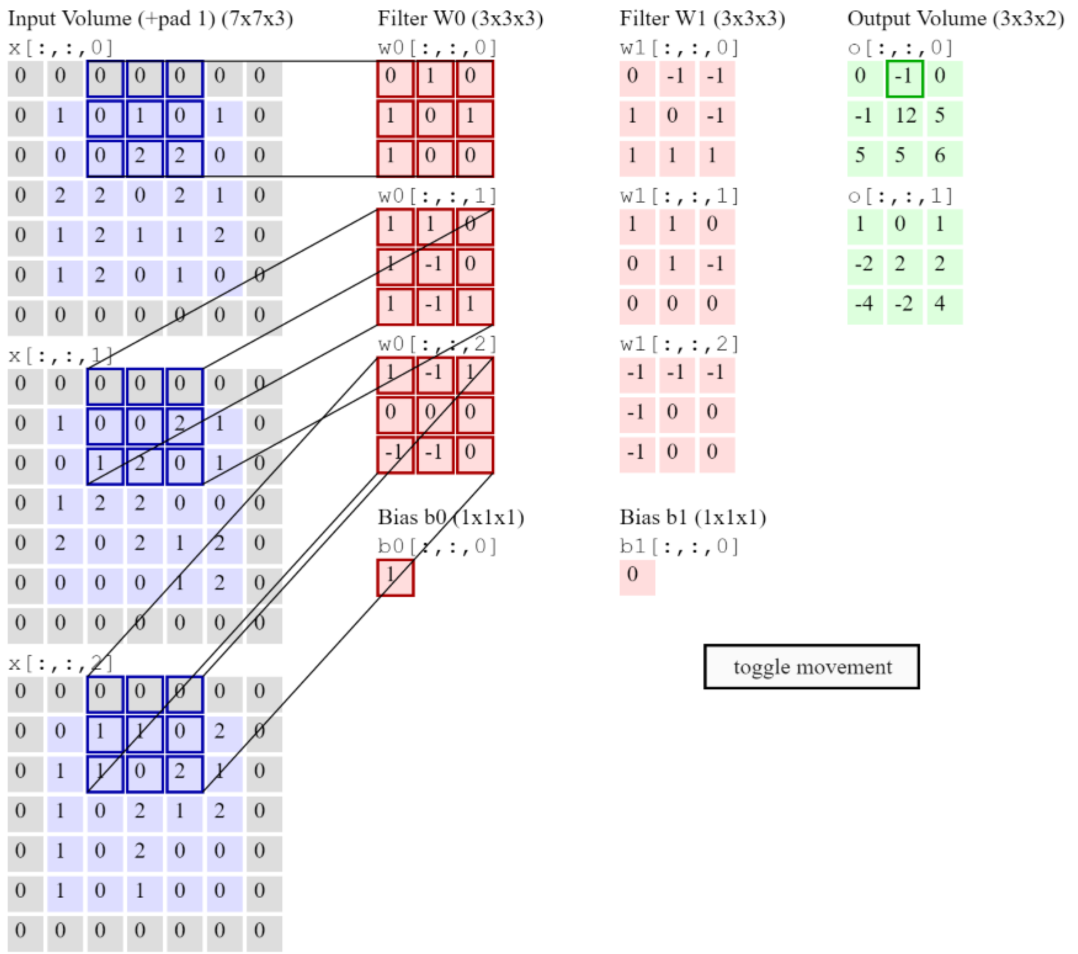


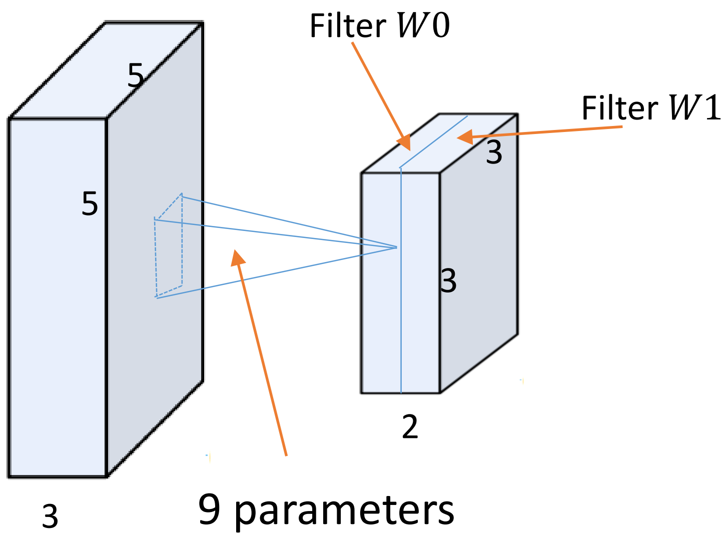
S: stride

conv layer neurons=$\frac{W+2P-F}{S}+1$

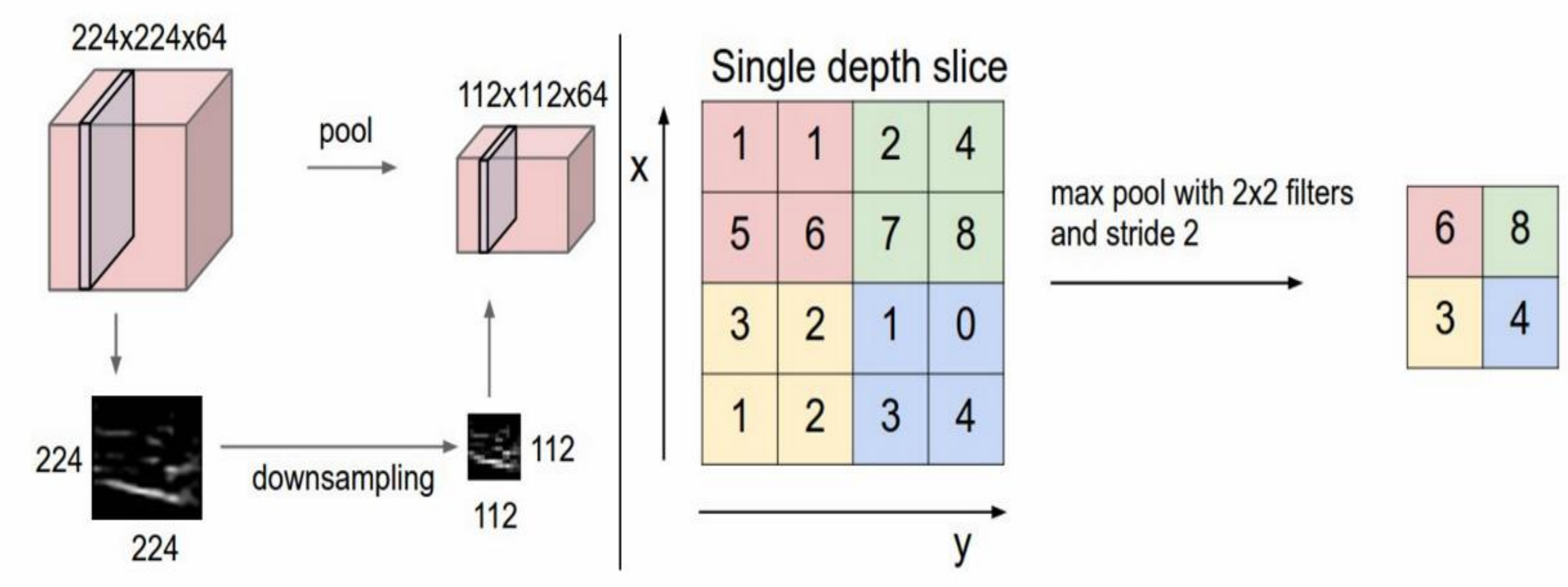
卷积层Convolution Layer

Stride=2





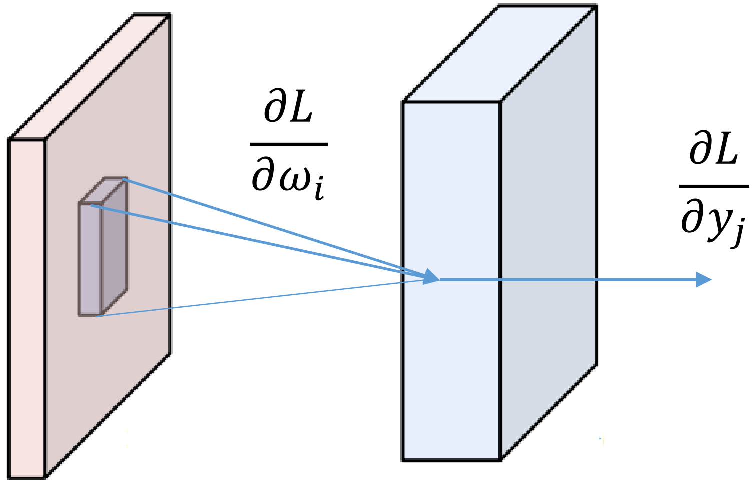
Pooling Layer



-fractional pooling: randomized $1 \times 1, 1 \times 2, 2 \times 1, 2 \times 2$ pooling 分数pooling

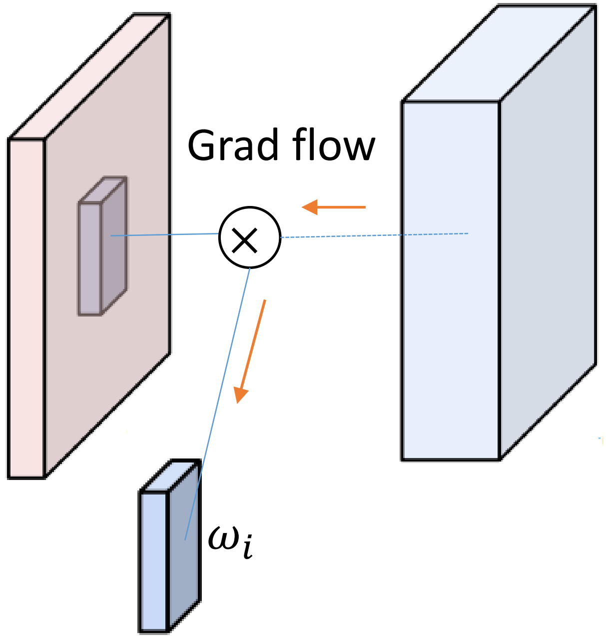
-all convolutional Net 全卷积网络（无pooling）

卷积神经网络中的梯度 BP in CNN



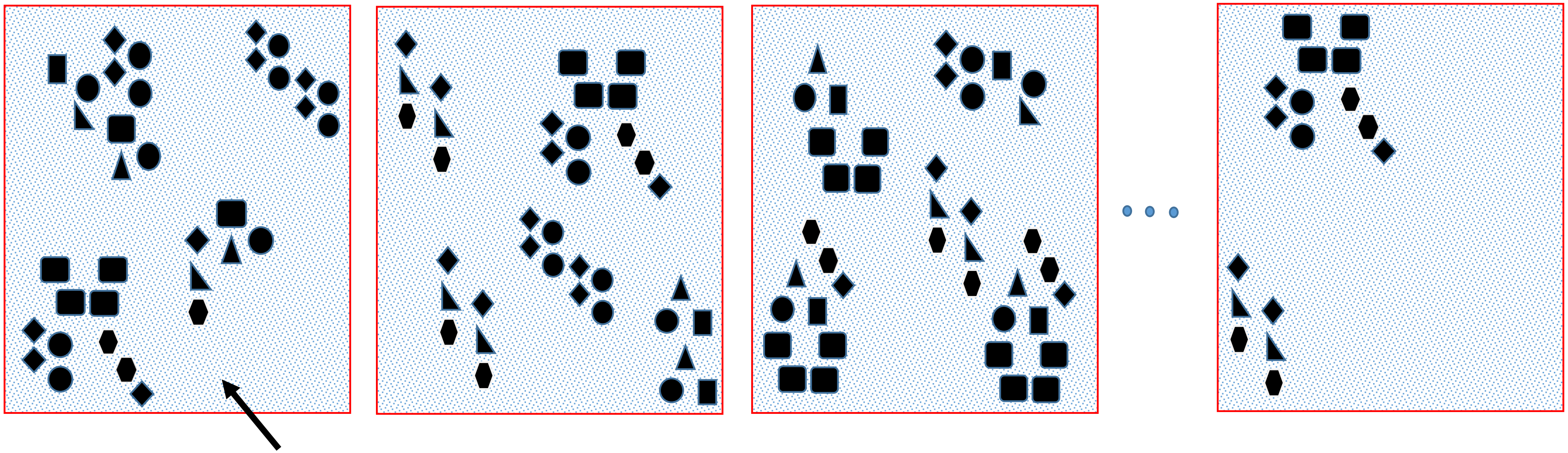
$$\frac{\partial L}{\partial \omega_i} = \sum_j \frac{\partial L}{\partial y_j} \cdot \frac{\partial y_j}{\partial \omega_i}$$

Can be viewed as

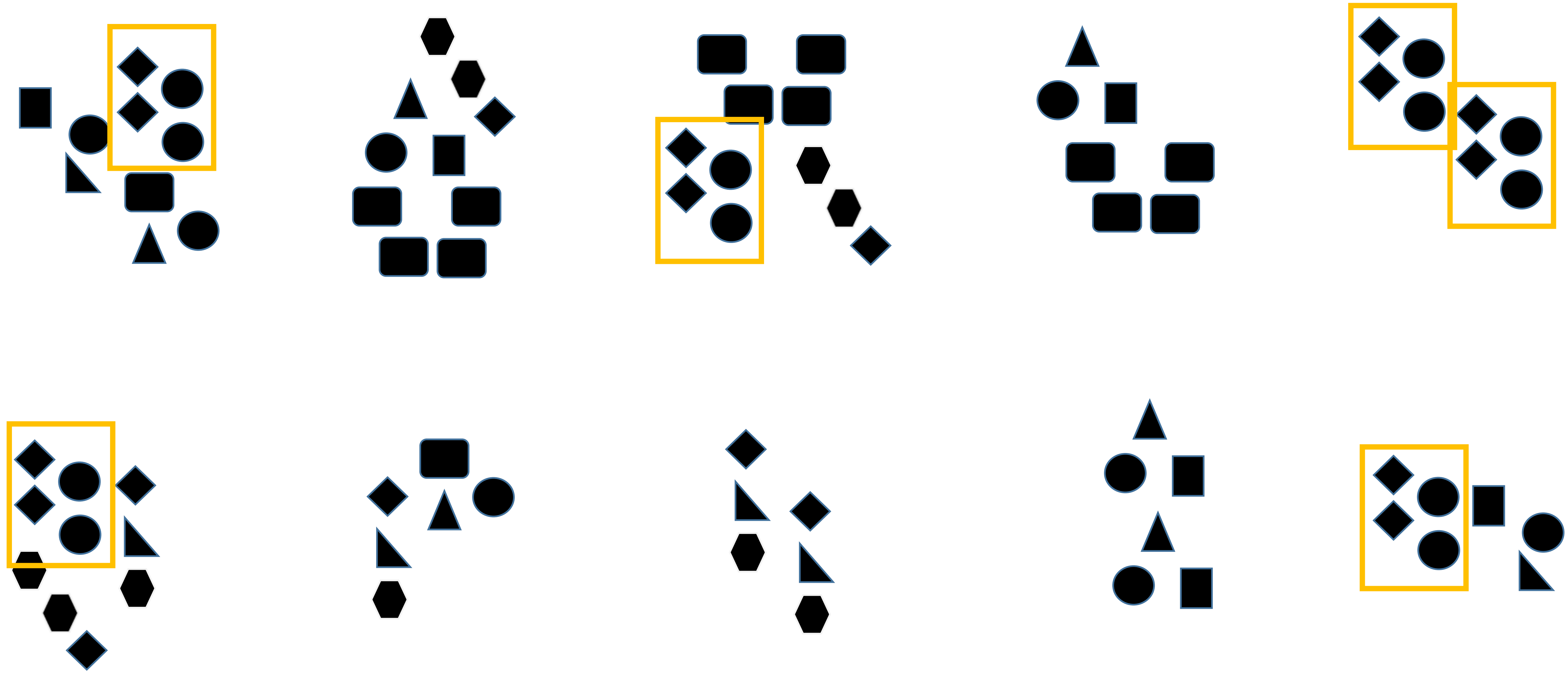


特征的层级 A Hierarchy of Features

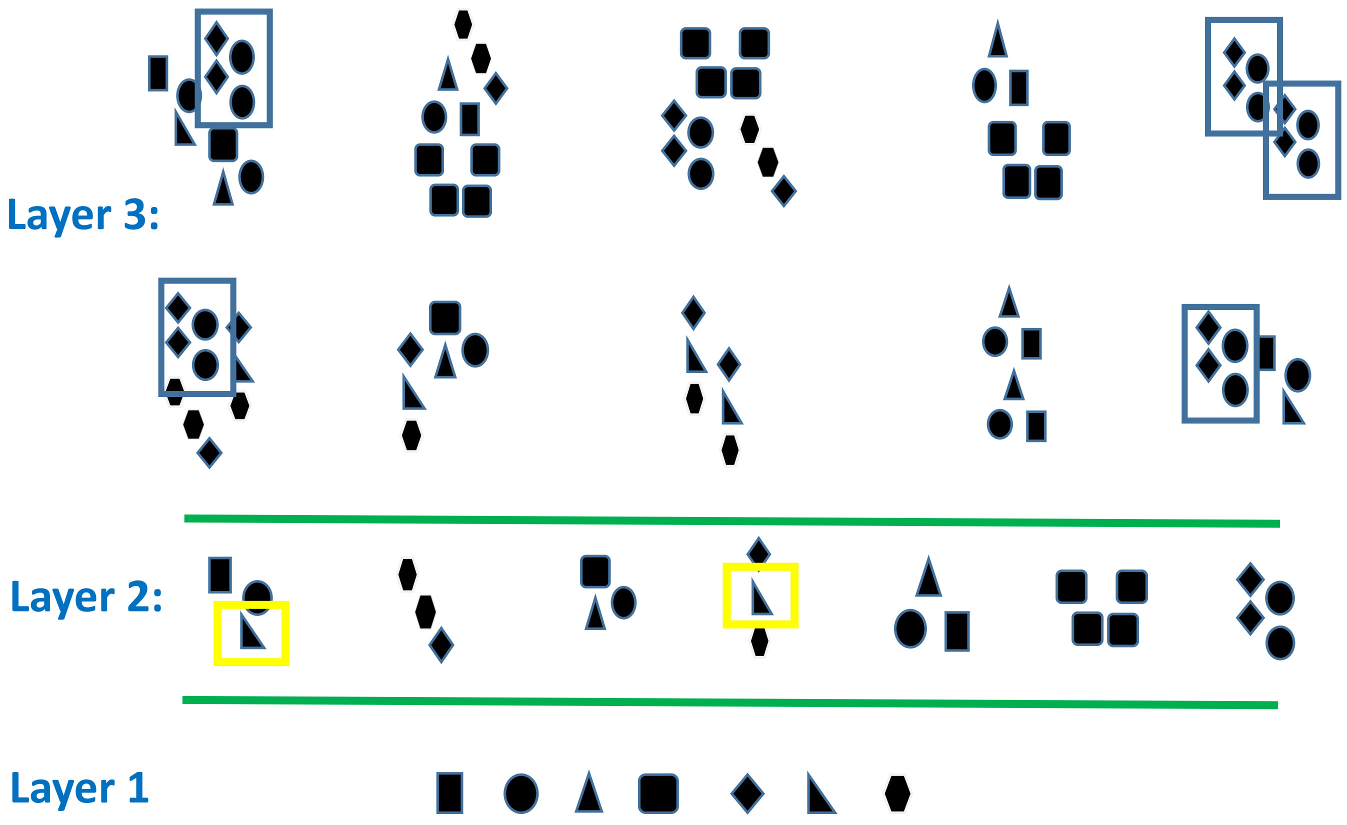
- Toy training images



特征的层级A Hierarchy of Features

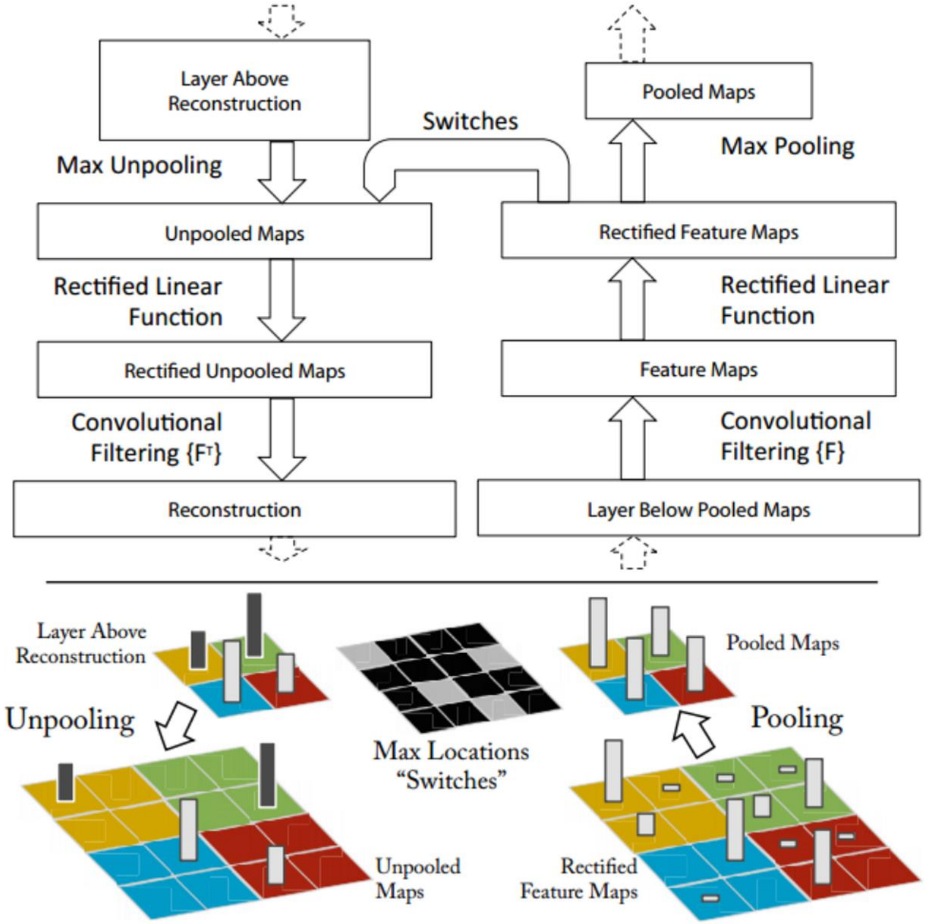


特征的层级A Hierarchy of Features

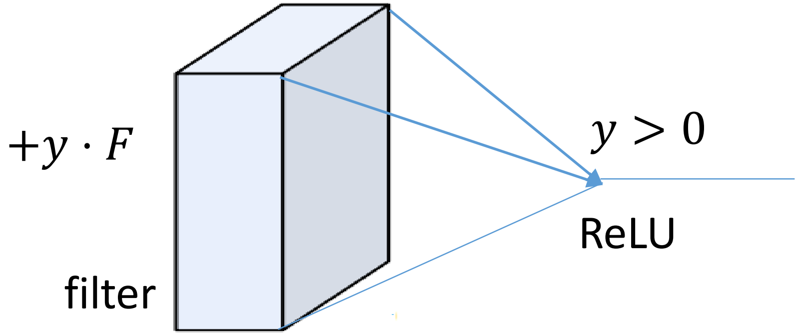


Visualizing CNN
卷积神经网络可视化

Deconv Net and Visualizing CNN [Matthew D. Zeiler and Rob Fergus]
反卷积和卷积神经网络可视化

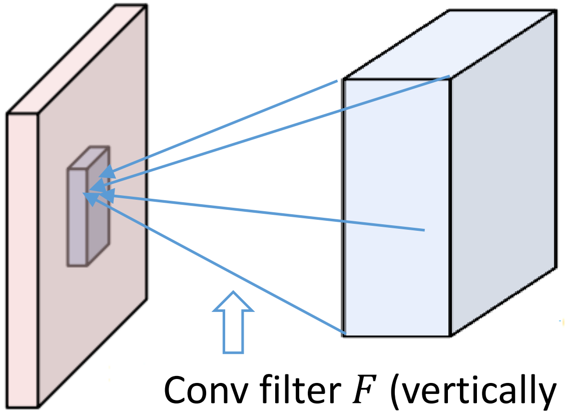


Deconv a ReLU layer



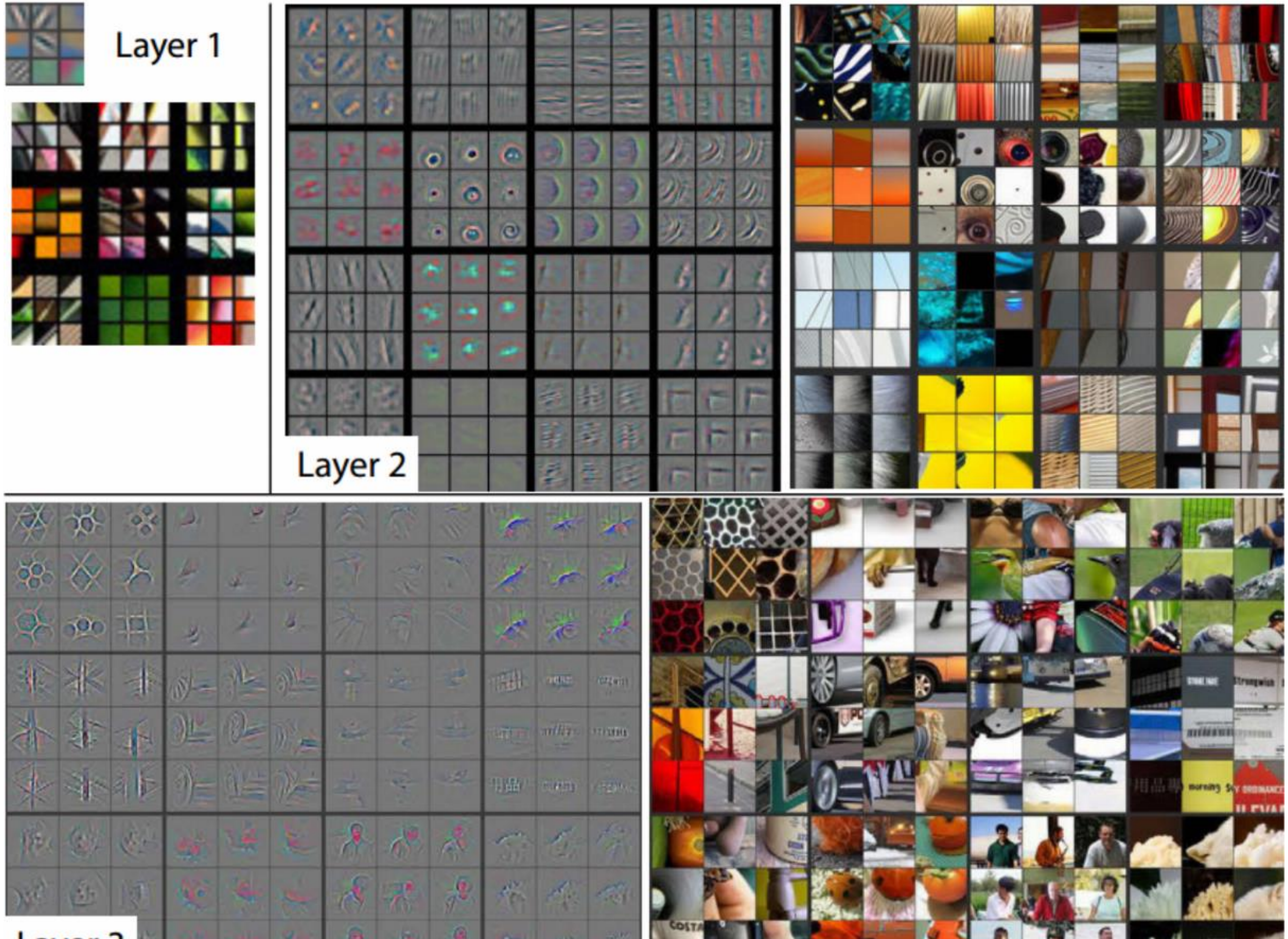
If the gate is activated, the previous layer $+y \cdot F$

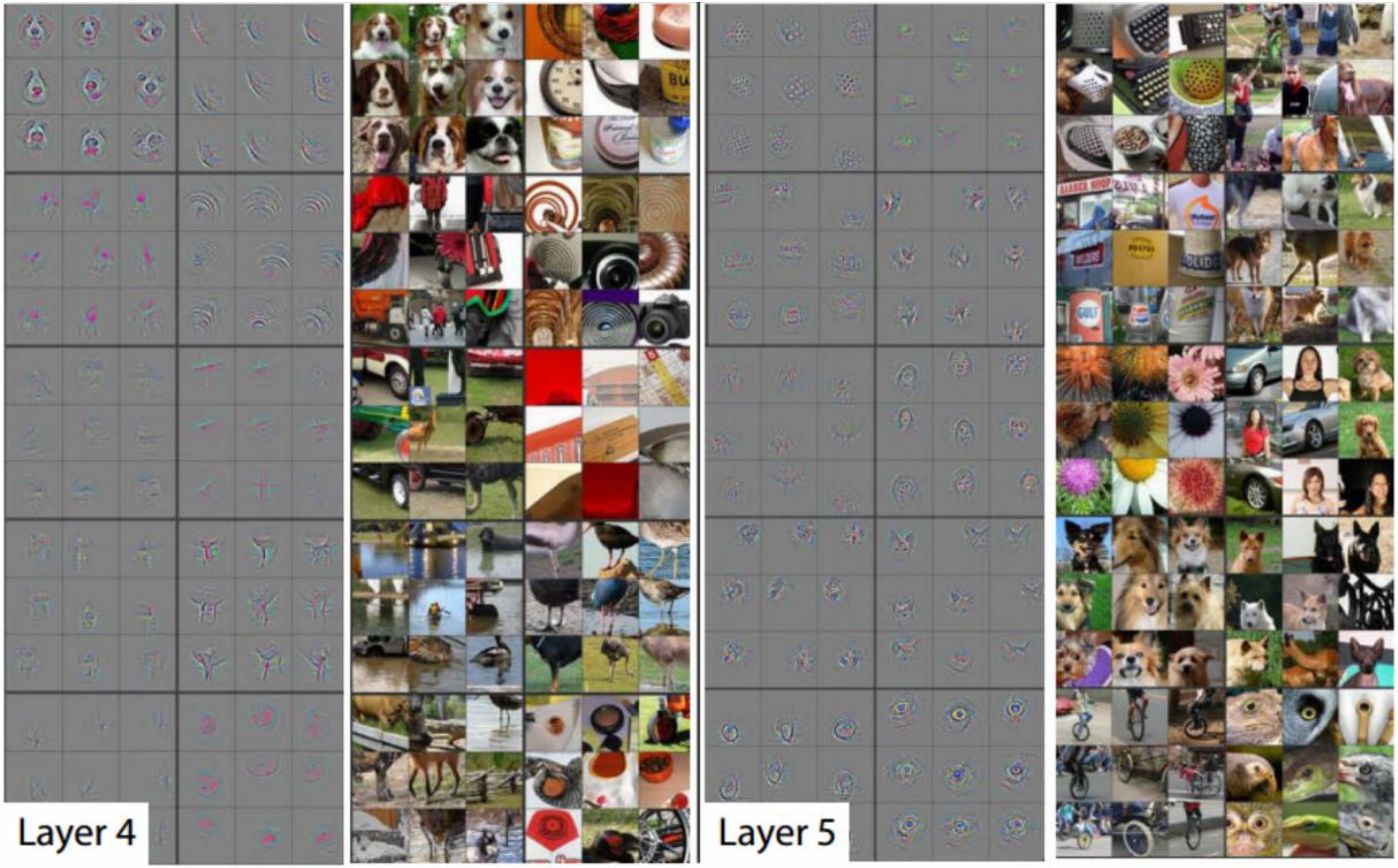
In fact, this is a convolution operation again



Conv filter F (vertically & horizontally reversed)

Try to figure this by yourself!


Layer 1
Layer 2

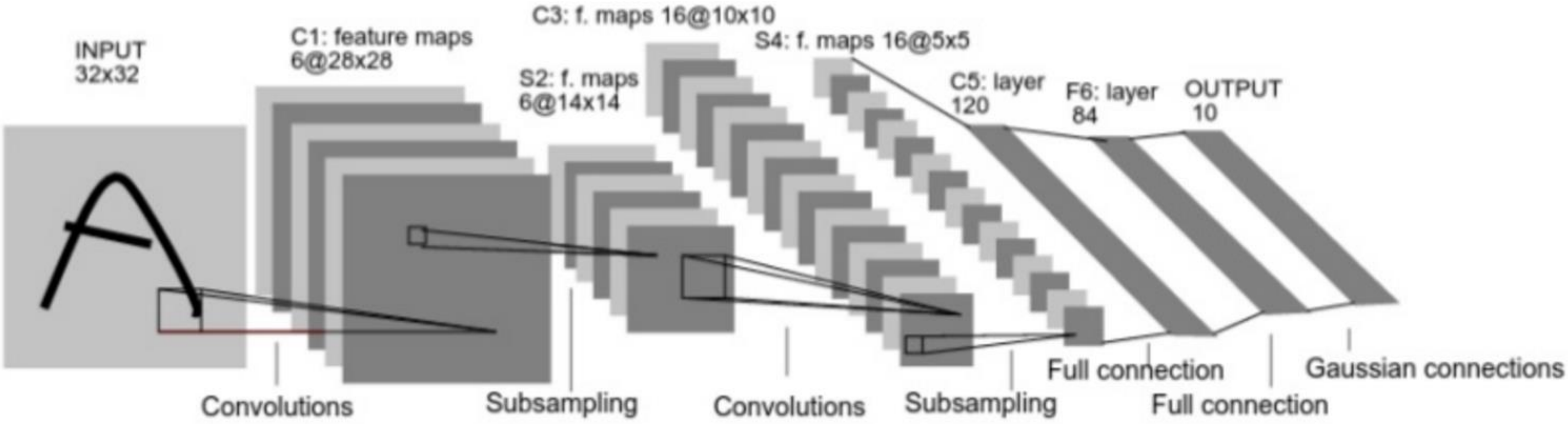

Layer 4
Layer 5

T-SNE [van der Maaten, Hinton]

- t-distributed stochastic neighbor embedding
  - A nonlinear dimension reduction
- Think the CNN code of an image as its feature vector (highly nonlinear features)
- Two images are closer if their CNN codes are closer in the feature space

Some popular CNN architectures
一些流行的卷积神经网络架构

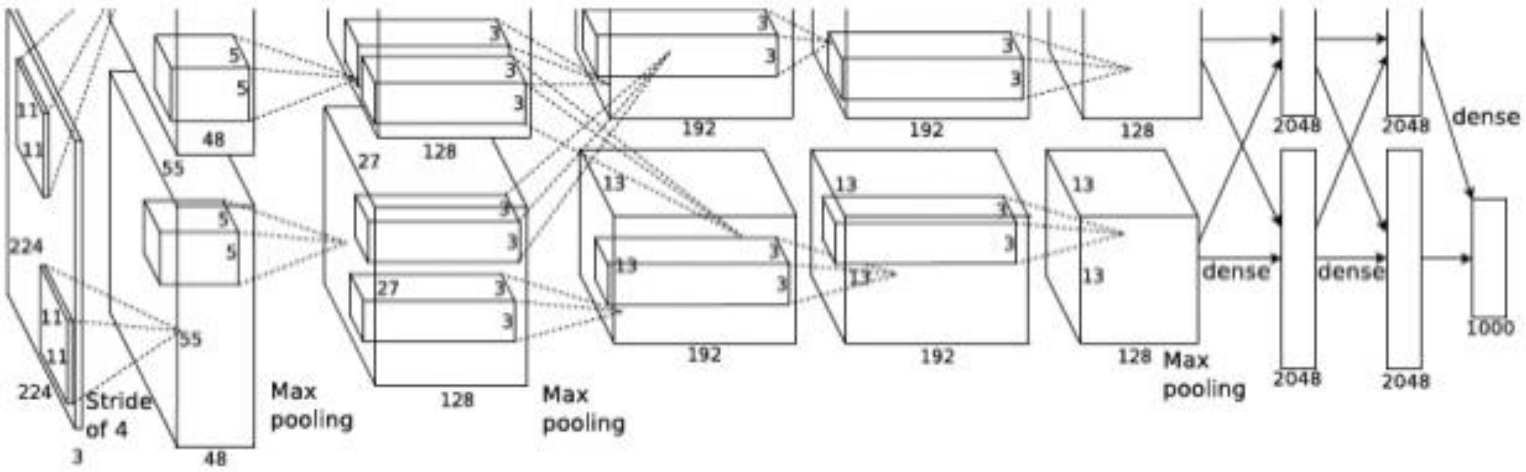
LeNet (Lecun-98)



Lenet-5 (Lecun-98), Convolutional Neural Network for digits recognition

Alexnet

- Similar framework to LeCun'98 but:
  - Bigger model (7 hidden layers, 650,000 units, 60,000,000 params)
  - More data (10^6 vs. 10^3 images)
  - GPU implementation (50x speedup over CPU)
    - Trained on two GPUs for a week



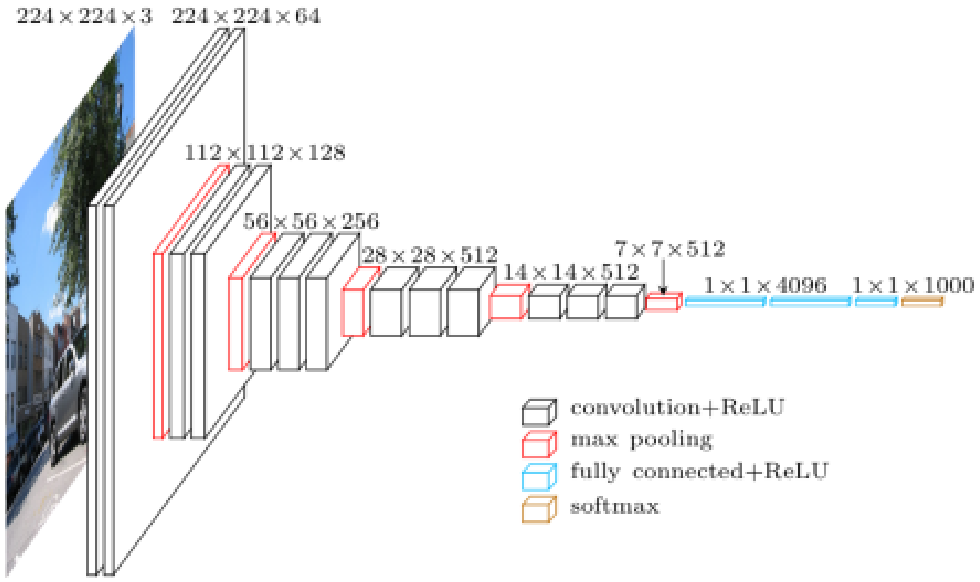
A. Krizhevsky, I. Sutskever, and G. Hinton, ImageNet Classification with Deep Convolutional Neural Networks, NIPS 2012

- https://github.com/BVLC/caffe/tree/master/models/bvlc_alexnet

VGG Net [Simonyan, Zisserman]

Model	top-5 classification error on ILSVRC-2012 (%)	
	validation set	test set
16-layer	7.5%	7.4%
19-layer	7.5%	7.3%
model fusion	7.1%	7.0%

Top-5 error in ImageNet (1000 classes)



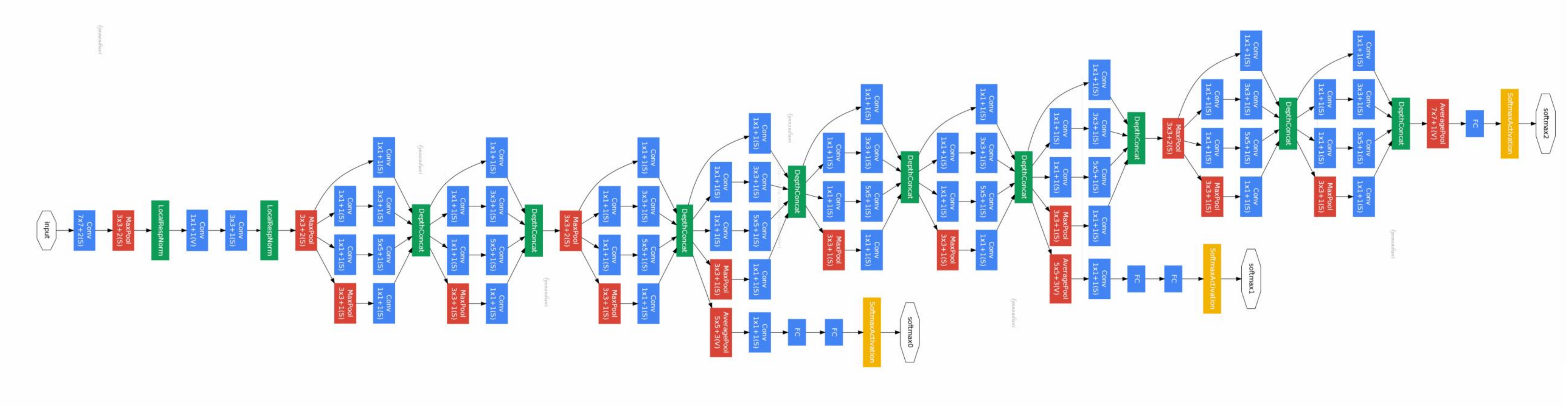
VGG16

Table 1: **ConvNet configurations** (shown in columns). The depth of the configurations increases from the left (A) to the right (E), as more layers are added (the added layers are shown in bold). The convolutional layer parameters are denoted as "conv⟨receptive field size⟩-⟨number of channels⟩". The ReLU activation function is not shown for brevity.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

- Implemented in Caffe
- You can download the weight from http://www.robots.ox.ac.uk/~vgg/research/very_deep/
- In Tensorflow: https://www.cs.toronto.edu/~frossard/post/vgg16/

GoogleNet [Szegedy et al.]



- https://github.com/BVLC/caffe/tree/master/models/bvlc_googlenet

ResNet [He et al.]

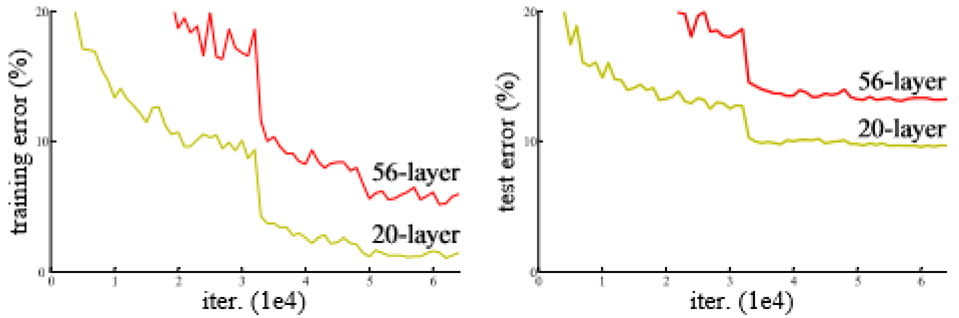
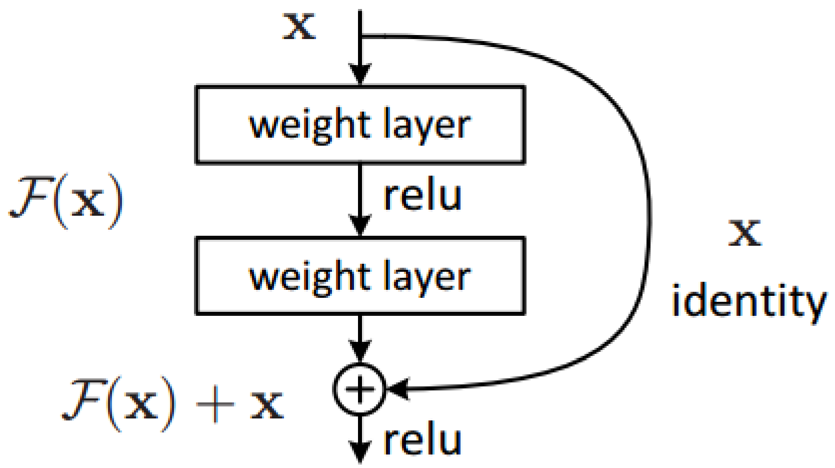


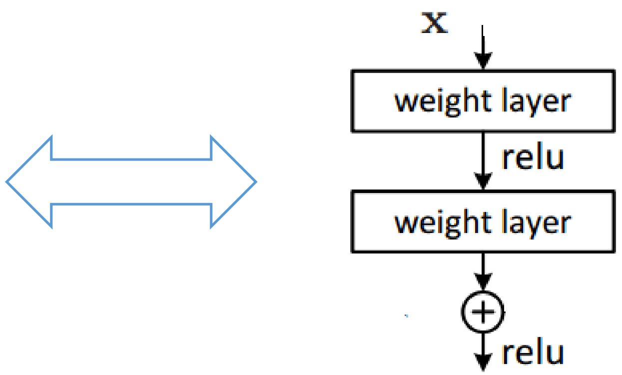
Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

ResNet [He et al.]

Stack many plain layers may even increase the training error叠加很多的层甚至会增加训练误差

ResNet

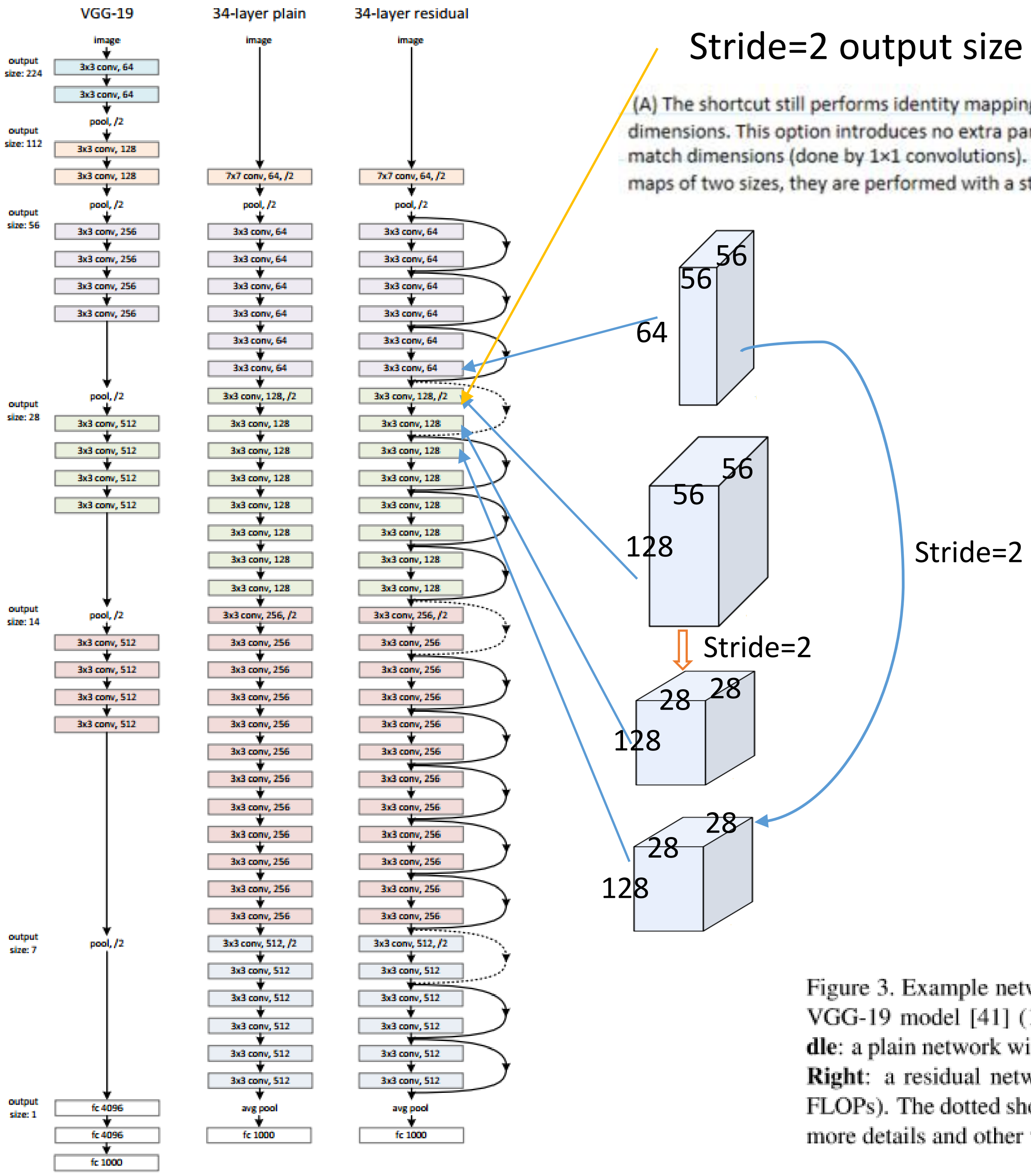




We hypothesis $H(x)$ is close to x

More generally $y = F(x, \{W_i\}) + x$
where F can be a general function e.g. $F = W_2\delta(W_1 x)$ in above

ResNet



Stride=2 output size halves

(A) The shortcut still performs identity mapping, with extra zero entries padded for increasing dimensions. This option introduces no extra parameter; (B) The projection shortcut in Eqn.(2) is used to match dimensions (done by 1×1 convolutions). For both options, when the shortcuts go across feature maps of two sizes, they are performed with a stride of 2.

$$y = F(x, \{W_i\}) + W_s x$$

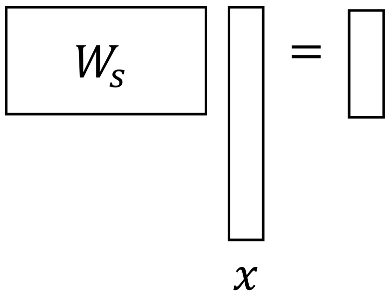


Figure 3. Example network architectures for ImageNet. **Left**: the VGG-19 model [41] (19.6 billion FLOPs) as a reference. **Middle**: a plain network with 34 parameter layers (3.6 billion FLOPs). **Right**: a residual network with 34 parameter layers (3.6 billion FLOPs). The dotted shortcuts increase dimensions. **Table 1** shows more details and other variants.

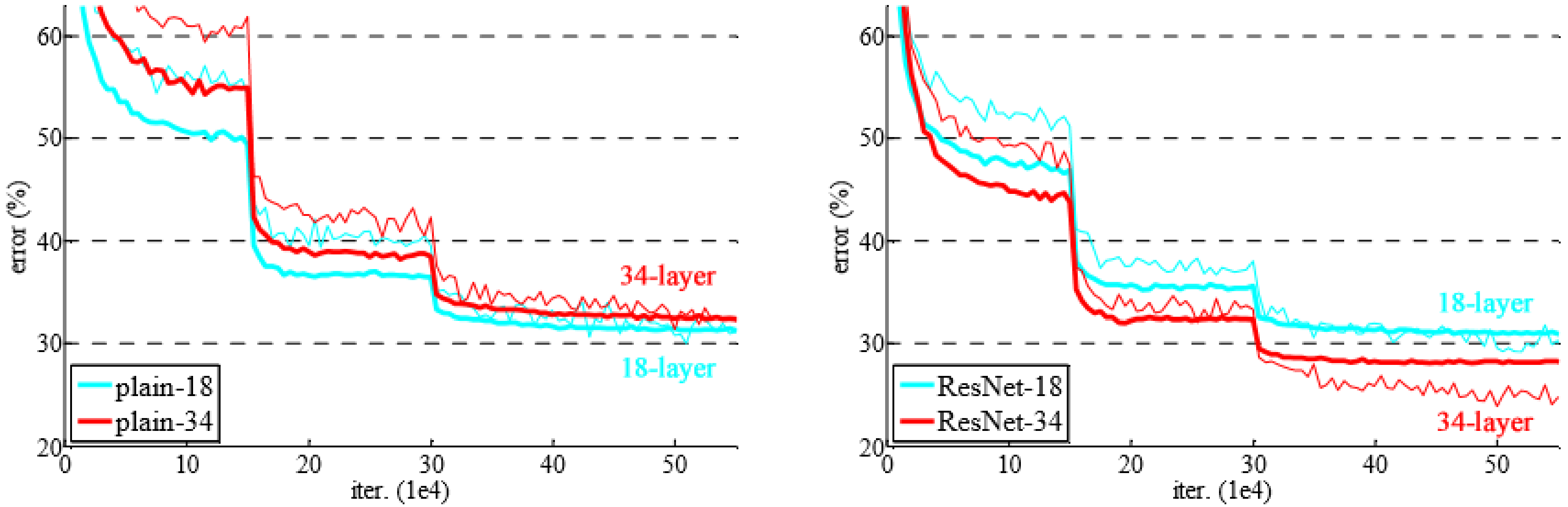


Figure 4. Training on **ImageNet**. Thin curves denote training error, and bold curves denote validation error of the center crops. Left: plain networks of 18 and 34 layers. Right: ResNets of 18 and 34 layers. In this plot, the residual networks have no extra parameter compared to their plain counterparts.

ResNet

- https://github.com/KaimingHe/deep-residual-networks
- A later improved model has 1000 layers

Fractal Net [Larsson et al.]

- The network is defined recursively

  递归定义

$$f_1(z) = \mathrm{conv}(z)$$

$$f_{C+1}(z) = [(f_C \circ f_C)(z)] \oplus [\mathrm{conv}(z)]$$

  $\circ$ denotes composition and $\oplus$ a join operation

- Instead of adding shortcut, FracNet provides a combination of short and long paths

  提供短的、长的路径的组合

  - neural information processing pathway

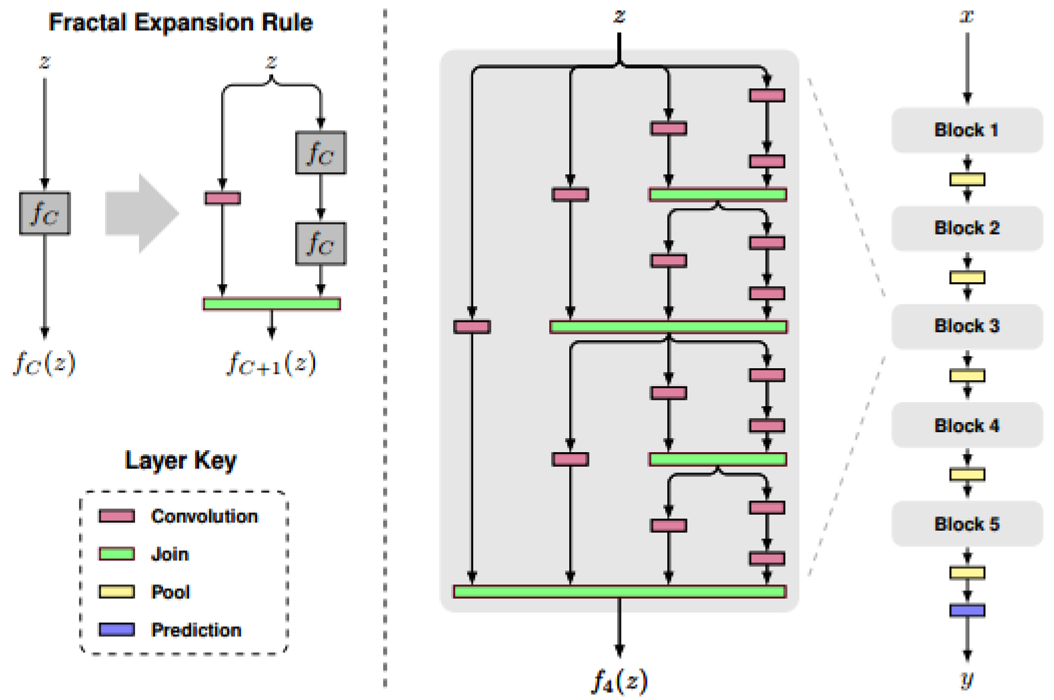


Figure 1: **Fractal architecture.** *Left:* A simple expansion rule generates a fractal architecture with C intertwined columns. The base case, $f_1(z)$, has a single layer of the chosen type (*e.g.* convolutional) between input and output. Join layers compute element-wise mean. *Right:* Deep convolutional networks periodically reduce spatial resolution via pooling. A fractal version uses f_C as a building block between pooling layers. Stacking B such blocks yields a network whose total depth, measured in terms of convolution layers, is $B \cdot 2^{C-1}$. This example has depth 40 ($B = 5$, $C = 4$).

Fractal Net

- Drop-path: a generalization of dropout dropout的推广

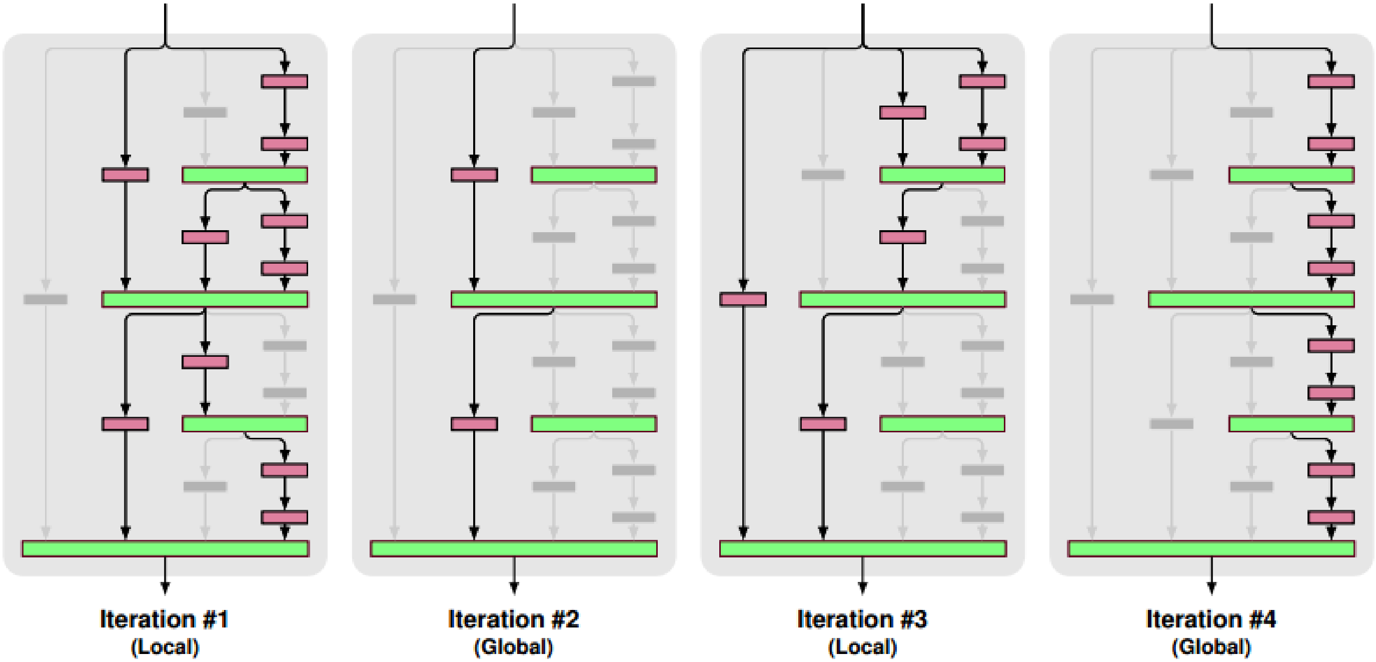


Figure 2: **Drop-path.** A fractal network block functions with some connections between layers disabled, provided some path from input to output is still available. Drop-path guarantees at least one such path, while sampling a subnetwork with many other paths disabled. During training, presenting a different active subnetwork to each mini-batch prevents co-adaptation of parallel paths. A global sampling strategy returns a single column as a subnetwork. Alternating it with local sampling encourages the development of individual columns as performant stand-alone subnetworks.

性能Performance

Method	C100	C100+	C100++	C10	C10+	C10++	SVHN
Network in Network [21]	35.68	-	-	10.41	8.81	-	2.35
Generalized Pooling [17]	32.37	-	-	7.62	6.05	-	1.69
Recurrent CNN [19]	31.75	-	-	8.69	7.09	-	1.77
Competitive Multi-scale [20]	27.56	-	-	6.87	-	-	1.76
FitNet [27]	-	35.04	-	-	8.39	-	2.42
Deeply Supervised [18]	-	34.57	-	9.69	7.97	-	1.92
All-CNN [30]	-	33.71	-	9.08	7.25	4.41	-
Highway Network [31]	-	32.39	-	-	7.72	-	-
ELU [2]	-	24.28	-	-	6.55	-	-
Scalable BO [29]	-	-	27.04	-	-	6.37	1.77
Fractional Max-Pooling [5]	-	-	26.32	-	-	3.47	-
FitResNet (LSUV) [23]	-	27.66	-	-	5.84	-	-
ResNet [8]	-	-	-	-	6.61	-	-
ResNet (reported by [11])	44.76	27.22	-	13.63	6.41	-	2.01
ResNet: Stochastic Depth [11]	37.80	24.58	-	11.66	5.23	-	1.75
ResNet: Identity Mapping [9]	-	22.68	-	-	4.69	-	-
ResNet in ResNet [33]	-	22.90	-	-	5.01	-	-
FractalNet	35.34	23.30	22.85	10.18	5.22	5.11	2.01
FractalNet+dropout/drop-path	28.20	23.73	23.36	7.33	4.60	4.59	1.87
↳ Deepest column alone	29.05	24.32	23.60	7.27	4.68	4.63	1.89

Table 1: **CIFAR-100/CIFAR-10/SVHN.** We compare test error (%) with other leading methods, trained with either no data augmentation, translation/mirroring (+), or more substantial augmentation (++). Our main point of comparison is ResNet. We closely match its state-of-the-art results using data augmentation, and outperform it by large margins without data augmentation. Training with drop-path, we can extract from FractalNet simple single-column networks that are highly competitive.

Stochastic Depth [Huang et al.]

- 深的ResNet非常难训练同时也很慢Very deep residual network: very hard and very slow to train
- Idea: randomly drop a subset of layers (treating them as Identity) (for each mini-batch)随机扔掉一些层（把他们视为恒等映射）
- Allow one to go beyond 1200 layers

p_l:survival probabilities

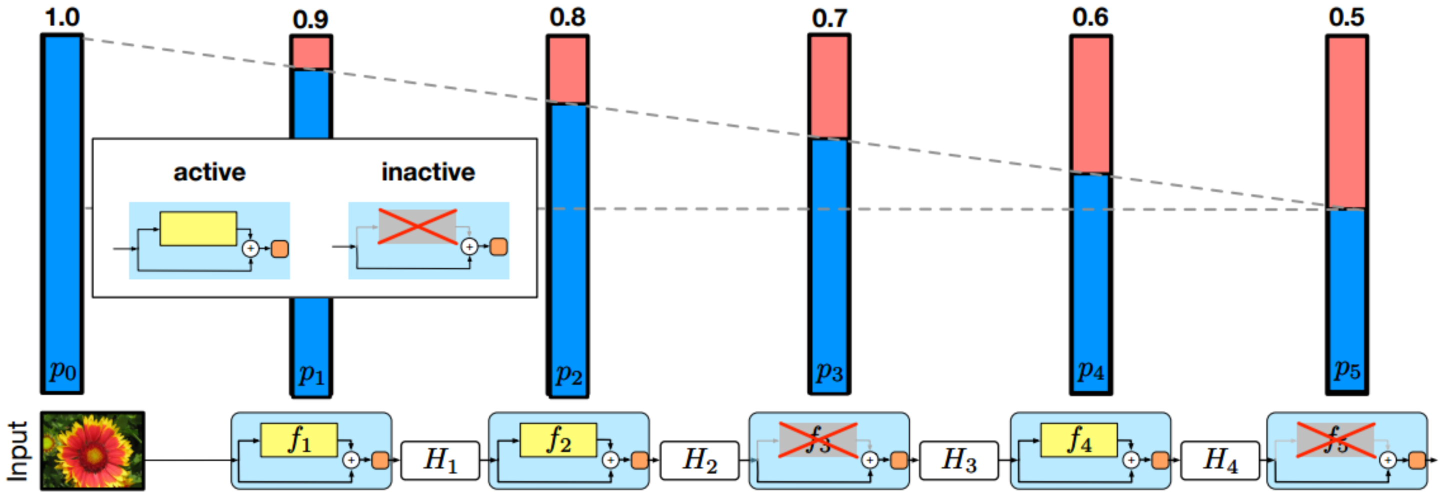


Fig. 2. The linear decay of p_ℓ illustrated on a ResNet with stochastic depth for $p_0 = 1$ and $p_L = 0.5$. Conceptually, we treat the input to the first ResBlock as H_0, which is always active.

Stochastic Depth

- https://github.com/yueatsprograms/Stochastic_Depth

Table 1. Test error (%) of ResNets trained with stochastic depth compared to other most competitive methods previously published (whenever available). A "+" in the name denotes standard data augmentation. ResNet with constant depth refers to our reproduction of the experiments by He et al.

	CIFAR10+	CIFAR100+	SVHN	ImageNet
Maxout [21]	9.38	-	2.47	-
DropConnect [20]	9.32	-	1.94	-
Net in Net [24]	8.81	-	2.35	-
Deeply Supervised [13]	7.97	-	1.92	33.70
Frac. Pool [25]	-	27.62	-	-
All-CNN [6]	7.25	-	-	41.20
Learning Activation [26]	7.51	30.83	-	-
R-CNN [27]	7.09	-	1.77	-
Scalable BO [28]	6.37	27.40	1.77	-
Highway Network [29]	7.60	32.24	-	-
Gen. Pool [30]	6.05	-	1.69	28.02
ResNet with constant depth	6.41	27.76	1.80	21.78
ResNet with stochastic depth	5.25	24.98	1.75	21.98

Table 2. Training time comparison on benchmark datasets.

	CIFAR10+	CIFAR100+	SVHN
Constant Depth	20h 42m	20h 51m	33h 43m
Stochastic Depth	15h 7m	15h 20m	25h 33m

Applications
应用

图像重构Image Reconstruction [Mahendran, Vedaldi 2014]

Find an image such that:
- Its code is similar to a given code
- It "looks natural" (image prior regularization)

$$\mathbf{x}^* = \underset{\mathbf{x}\in\mathbb{R}^{H\times W\times C}}{\operatorname{argmin}} \ell(\Phi(\mathbf{x}), \Phi_0) + \lambda\mathcal{R}(\mathbf{x})$$

$$\ell(\Phi(\mathbf{x}), \Phi_0) = \|\Phi(\mathbf{x}) - \Phi_0\|^2$$

$R(x)$: regularizer to encourage "natural image"

$R(x) = \|x\|_\alpha^\alpha$ (e.g. $\alpha = 6$)

$$R_{TV}(x) = \sum_{ij}\left(\left(x_{i+1,j} - x_{ij}\right)^2 + \left(x_{i,j+1} - x_{ij}\right)^2\right)^{\beta/2}$$

图像重构Image Reconstruction

Understanding Deep Image Representations by Inverting Them
[Mahendran and Vedaldi, 2014]

original image

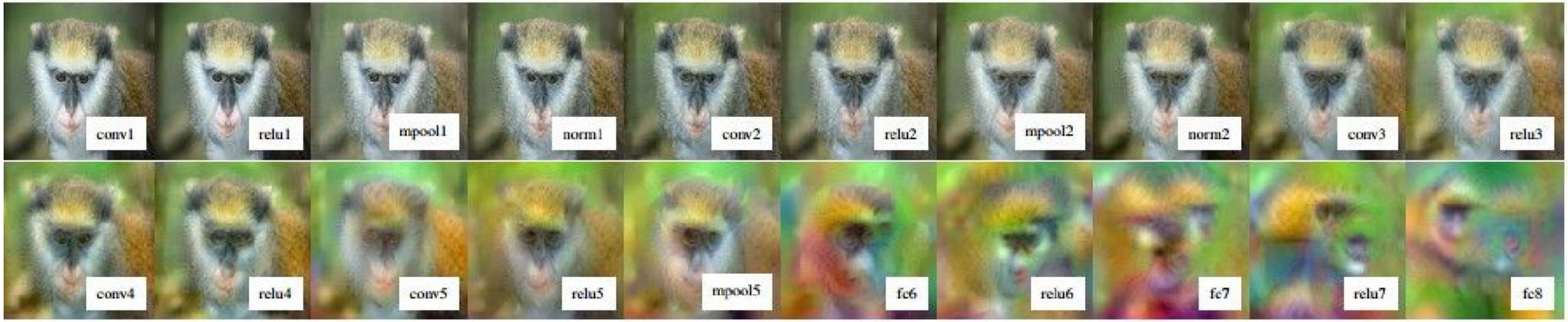
reconstructions from the 1000 log probabilities for ImageNet (ILSVRC) classes

图像重构Image Reconstruction

Reconstructions from the representation after last last pooling layer
(immediately before the first Fully Connected layer)

图像重构Image Reconstruction

Reconstructions from intermediate layers



图像重构Image Reconstruction

- https://github.com/aravindhm/deep-goggle

Deep Dream

inception_4c/output

Deep Dream

- caffe

IDEA: if a neuron is activated, activate it further!

we don't have a loss function

```
def objective_L2(dst):
    dst.diff[:] = dst.data

def make_step(net, step_size=1.5, end='inception_4c/output',
              jitter=32, clip=True, objective=objective_L2):
    '''Basic gradient ascent step.'''

    src = net.blobs['data'] # input image is stored in Net's 'data' blob
    dst = net.blobs[end]

    ox, oy = np.random.randint(-jitter, jitter+1, 2)
    src.data[0] = np.roll(np.roll(src.data[0], ox, -1), oy, -2) # apply jitter shift

    net.forward(end=end)
    objective(dst)  # specify the optimization objective
    net.backward(start=end)
    g = src.diff[0]
    # apply normalized ascent step to the input image
    src.data[:] += step_size/np.abs(g).mean() * g

    src.data[0] = np.roll(np.roll(src.data[0], -ox, -1), -oy, -2) # unshift image

    if clip:
        bias = net.transformer.mean['data']
        src.data[:] = np.clip(src.data, -bias, 255-bias)
```

DeepDream: set dx = x :)

a layer in googlenet

← a forward computation (from 'data' to 'end'), we get activation values at 'end'

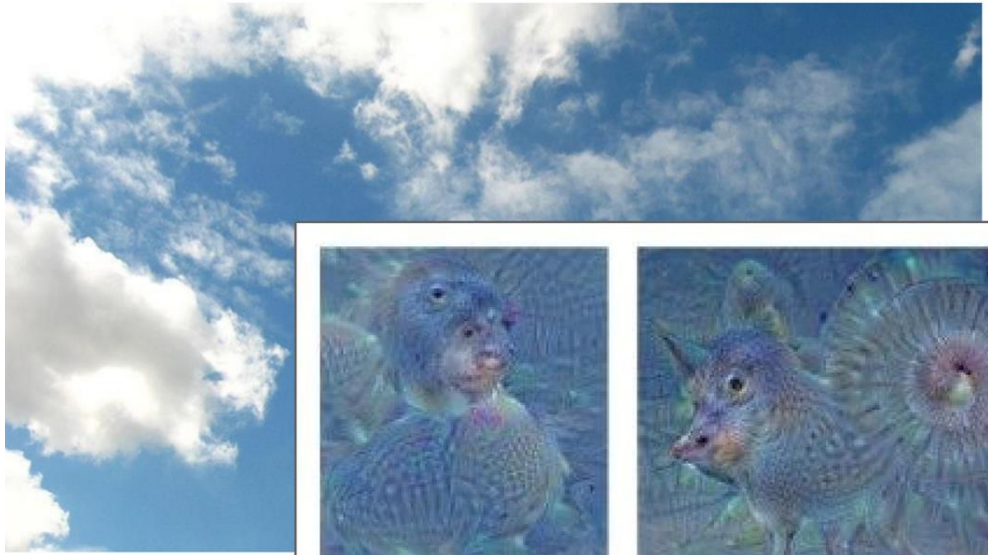
← backward computation, starting from 'end'

"image update"

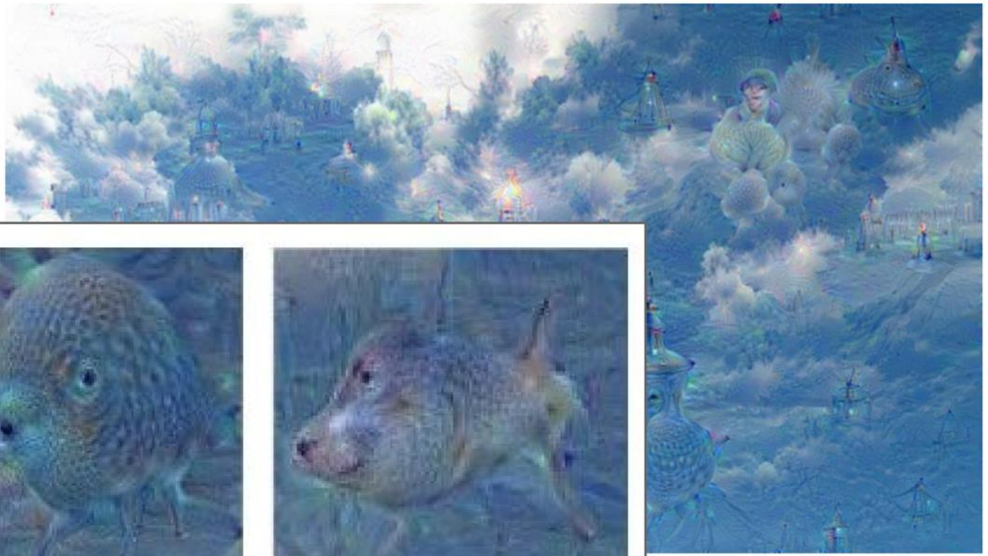
jitter regularizer

only marginally useful

Deep Dream



inception_4c/output

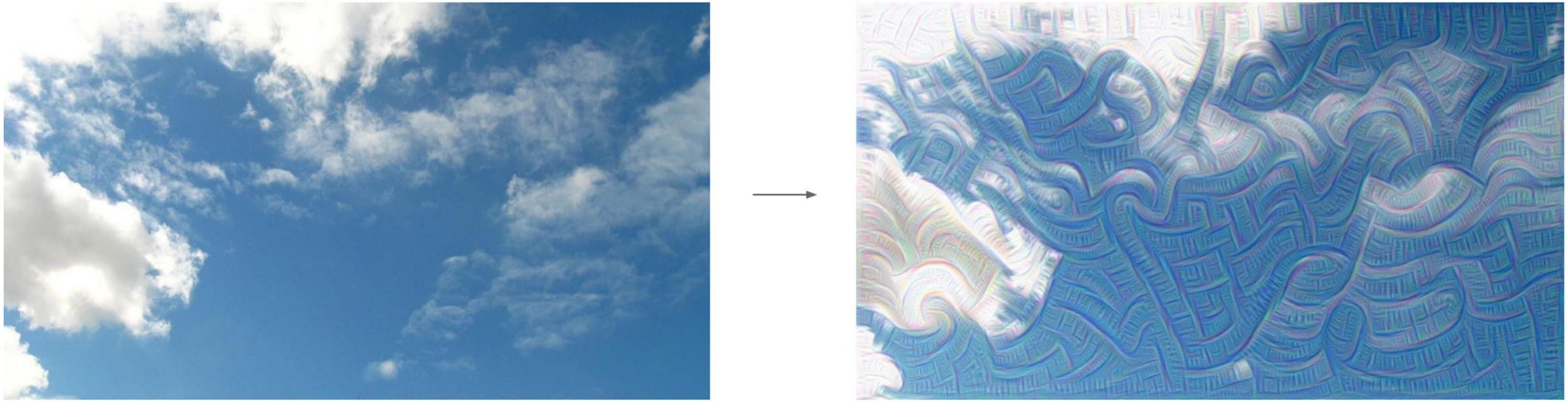


"Admiral Dog!" "The Pig-Snail" "The Camel-Bird" "The Dog-Fish"

DeepDream modifies the image in a way that "boosts" all activations, at any layer

Deep Dream

inception_3b/5x5_reduce



DeepDream modifies the image in a way that “boosts” all activations, at any layer

Deep Dream

Inceptionism!



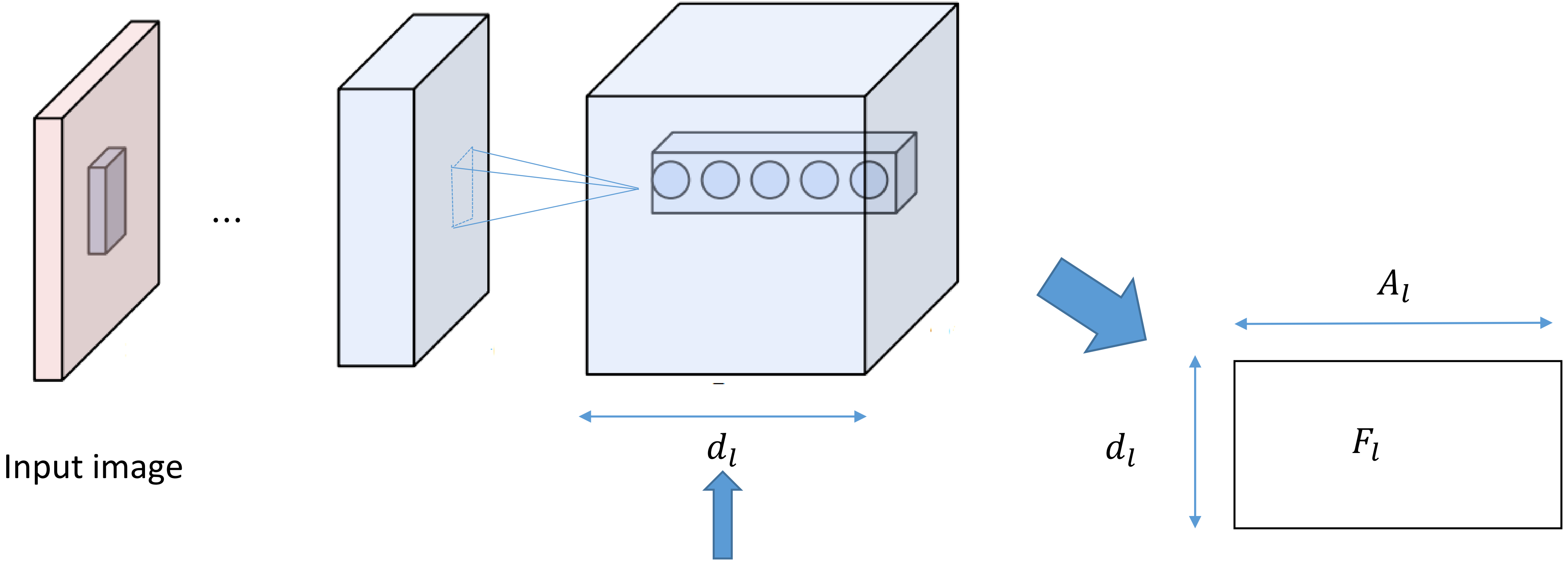
Neuralstyle [Gatys et al. 2015]

Neuralstyle

1. Try to match the content from the original figure 匹配原图的内容
2. Try to match the style from the art work 匹配艺术作品的风格

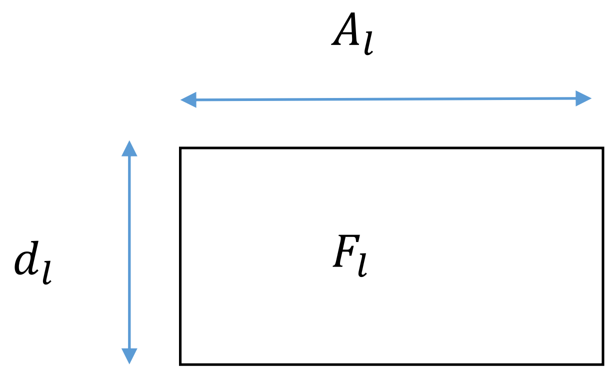
Correlation of filter response

Neuralstyle



Layler l: all response can be stored in tensor $R^{d_l \times \omega_l \times h_l}$
flatten it into $F_l \in R^{d_l \times A_l}$ $(A_l = \omega_l \times h_l)$

Neuralstyle



(find an image)

匹配内容 Matching the content.

loss: $L_{content}(p, x, l) = \frac{1}{2}\sum_{ij}\left(F_{ij}^{l} - p_{ij}^{l}\right)^2$

p： given input image, x: we want to generate x, l: layer l

F_{ij}^{l}: feature representation of x in layer l

p_{ij}^{l}: feature representation of p in layer l

How to get x? 用白噪声初始化 再通过梯度下降迭代

initially x ←white noise

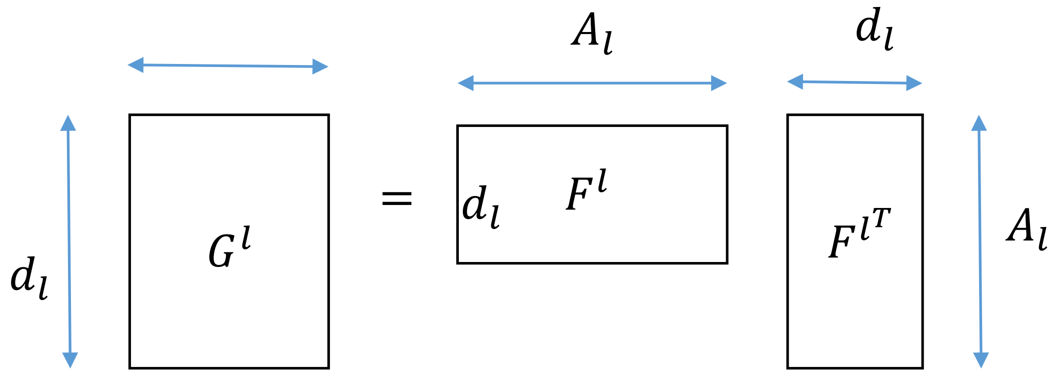
iterate GD (the network is fixed, but x is variable. So $\nabla_x L$ is well-defined

$$x_t \leftarrow x_{t-1} - \lambda_t \nabla_x L)$$

Neuralstyle

匹配风格 Matching the style

- 特征相关性 Feature correlation
  - $G^{l} \in R^{d_l \times d_l}$
- 损失函数Loss function:



$$L_{style}(a,x) = \sum_{l=0}^{L} \omega_l \left(\frac{1}{4d_l^2 A_l^2} \sum_{ij} \left(G_{ij}^l - A_{ij}^l\right)^2\right)$$

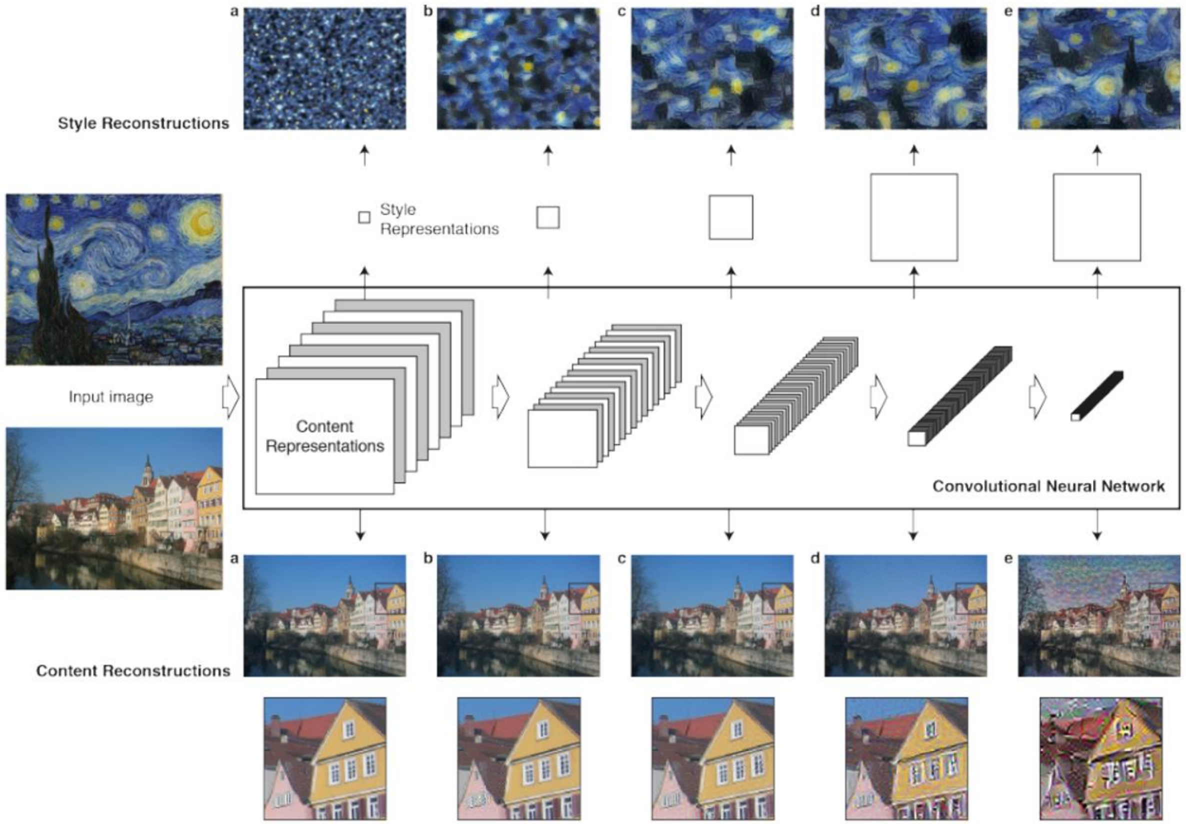
a: art work, x: we want to generate, ω_l: weight for layers

G_{ij}^l: feature correlation for x, A_{ij}^l: feature correlation for the art work

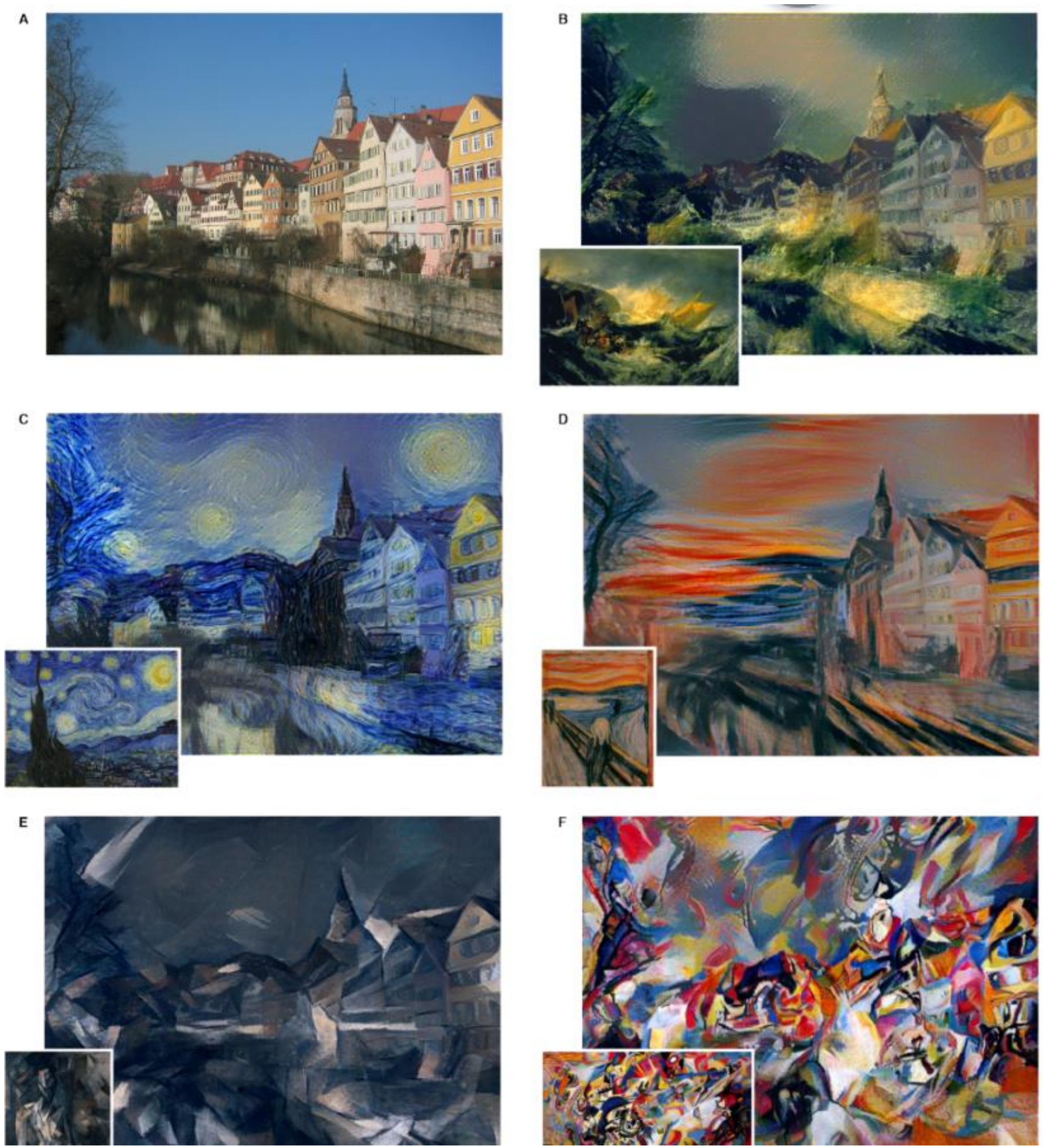
训练是一样的Training is the same (start from white noise)

Overall loss: $L_{total}(p,a,x) = \alpha L_{content}(p,x) + \beta L_{style}(a,x)$

Neuralstyle



Neuralstyle



Neuralstyle

More focus on style

More focus on content

Different α/β value

Which style layer to match

Style more Local

Depending on size of receptive Field

Style more global

Neuralstyle

- In tensorflow:
  - https://github.com/anishathalye/neural-style
- Mxnet
  - https://github.com/dmlc/mxnet/tree/master/example/neural-style

Convolution Layer in Keras

- Keras Code

```
keras.layers.convolutional.Conv2D(filters, kernel_size, strides=(1, 1), padding='valid',
data_format=None, dilation_rate=(1, 1), activation=None, use_bias=True,
kernel_initializer='glorot_uniform', bias_initializer='zeros', kernel_regularizer=None,
bias_regularizer=None, activity_regularizer=None, kernel_constraint=None, bias_constraint=None)
```

- When using this layer as the first layer in a model, provide the keyword argument input_shape
- Input dim: 4D tensor with shape: (samples, channels, rows, cols)
- Output dim: 4D tensor with shape: (samples, filters, new_rows, new_cols)

filters: Integer, the dimensionality of the output space (i.e. the number output of filters in the convolution).
kernel_size: An integer or tuple/list of 2 integers, specifying the width and height of the 2D convolution window. Can be a single integer to specify the same value for all spatial dimensions.
strides: An integer or tuple/list of 2 integers, specifying the strides of the convolution along the width and height. Can be a single integer to specify the same value for all spatial dimensions. Specifying any stride value != 1 is incompatible with specifying any `dilation_rate` value != 1.
padding: one of `"valid"` or `"same"` (case-insensitive).
activation: Activation function to use (see activations). If you don't specify anything, no activation is applied (ie. "linear" activation: `a(x) = x`).

https://keras.io/layers/convolutional/#conv2d

Pooling Layer in Keras

- Keras Code

Max pooling

```
keras.layers.pooling.MaxPooling2D(pool_size=(2, 2), strides=None, padding='valid',
data_format=None)
```

- Input dim: 4D tensor with shape: (batch_size, rows, cols, channels)
- Output dim: 4D tensor with shape: (batch_size, pooled_rows, pooled_cols, channels)

padding: one of `"valid"` or `"same"` (case-insensitive).

Average pooling

```
keras.layers.pooling.AveragePooling1D(pool_size=2, strides=None, padding='valid')
```

https://keras.io/layers/pooling/

用Keras实现CNN

Keras for CNN

```
# The data, shuffled and split between train and test sets:
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')
```

读入数据

```
# Convert class vectors to binary class matrices.
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

处理label，变成categorical类型

```
model = Sequential()
```

顺序模型

```
model.add(Conv2D(32, (3, 3), padding='same',
                 input_shape=x_train.shape[1:]))
model.add(Activation('relu'))
model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
```

2D Convolution，卷积核长宽3×3，补0使得输入输出长宽一样，32个输出channel

Keras for CNN

```
model.add(Flatten())
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes))
model.add(Activation('softmax'))

# initiate RMSprop optimizer
opt = keras.optimizers.rmsprop(lr=0.0001, decay=1e-6)

# Let's train the model using RMSprop
model.compile(loss='categorical_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])
```

把目前的tensor展开成1维

- **lr**: float >= 0. Learning rate.
- **rho**: float >= 0.
- **epsilon**: float >= 0. Fuzz factor.
- **decay**: float >= 0. Learning rate decay over each update.

用Tensorflow实现CNN

Tensorflow for CNN

```python
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data

def weight_varible(shape):
    initial = tf.truncated_normal(shape, stddev=0.1)
    return tf.Variable(initial)

def bias_variable(shape):
    initial = tf.constant(0.1, shape=shape)
    return tf.Variable(initial)

def conv2d(x, W):
    return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')

def max_pool_2x2(x):
    return tf.nn.max_pool(x, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')


mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
```

Import tensorflow and import mnist data

Define weight_variable function which return a tensor with given shape satisfying normal distribution with 0.1 variance

Define bias_variable function which return a constant tensor with given shape and value 0.1

Define conv2d function which does a convolution with stride 1 and zero padding

Define max_pool_2x2 function which do max poolin with stride 2 and fileter size 2

Load mnist data

Conv2d: x is an input tensor of shape [batch, in_height, in_width, in_channels] and a filter / kernel tensor W of shape [filter_height, filter_width, in_channels, out_channels]

Tensorflow for CNN

```
# paras
W_conv1 = weight_varible([5, 5, 1, 32])
b_conv1 = bias_variable([32])

# conv layer-1
x = tf.placeholder(tf.float32, [None, 784])
x_image = tf.reshape(x, [-1, 28, 28, 1])

h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)
h_pool1 = max_pool_2x2(h_conv1)

# conv layer-2
W_conv2 = weight_varible([5, 5, 32, 64])
b_conv2 = bias_variable([64])

h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) + b_conv2)
h_pool2 = max_pool_2x2(h_conv2)

# full connection
W_fc1 = weight_varible([7 * 7 * 64, 1024])
b_fc1 = bias_variable([1024])

h_pool2_flat = tf.reshape(h_pool2, [-1, 7 * 7 * 64])
h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)

# output layer: softmax
W_fc2 = weight_varible([1024, 10])
b_fc2 = bias_variable([10])
```

Produce first convolution layer parameters

Reshape data to image shape

Create first convolution layer followed by a relu nonlinear func and max pooling

Produce full connection layer parameters

Reshape data from image shape to a matrix shape

Create a full connection layer followed by a relu nonlinear func

Tensorflow for CNN

```
y_conv = tf.nn.softmax(tf.matmul(h_fc1, W_fc2) + b_fc2)
y_ = tf.placeholder(tf.float32, [None, 10])

# model training
cross_entropy = -tf.reduce_sum(y_ * tf.log(y_conv))
train_step = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)

correct_prediction = tf.equal(tf.arg_max(y_conv, 1), tf.arg_max(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

num_epoch = 10000
batchsz = 50
iters_per_epoch = num_epoch / batchsz
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for i in xrange(num_epoch):
        for iters in xrange(iters_per_epoch):
            batch = mnist.train.next_batch(batchsz)
            tf.run(train_step, feed_dict = {x: batch[0], y_: batch[1]})
        train_accuacy = accuracy.eval(feed_dict={x: batch[0], y_: batch[1], keep_prob: 1.0})
        print("step %d, training accuracy %g"%(i, train_accuacy))
```

Compute accuracy

- Some slides borrowed from Gaurav Mittal’s slides, Lawrence Carin’s slides, cs231n at Stanford