深度学习技术与应用（6）

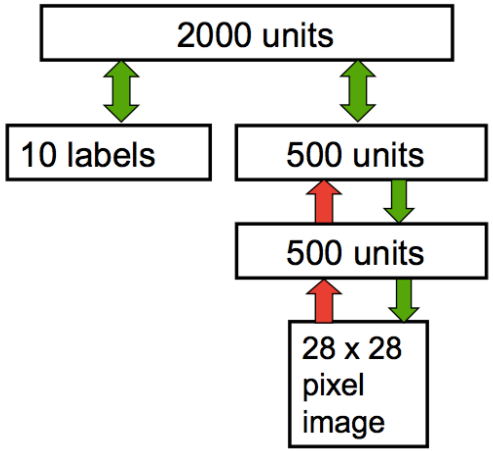
Deep Learning: Techniques and Applications (6)

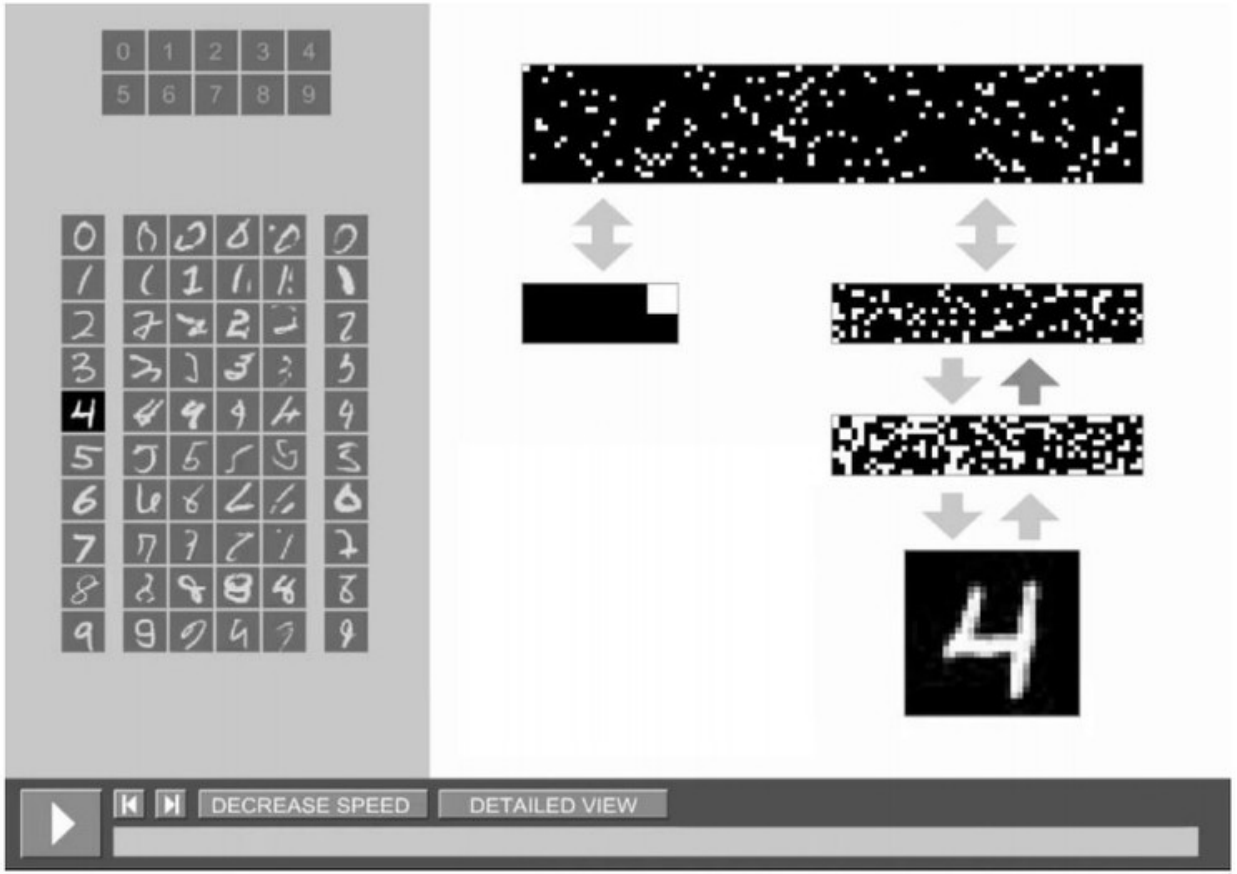
Ge Li

Peking University

Problems for Fully Connected Neural Networks

- Hinton in 2006





Problems for Fully Connected Neural Networks

- Fully Connect Networks
  - With small images, it was computationally feasible to learn features on the entire image.
    - 28x28 images for the MNIST dataset
  - With larger images, learning features that span the entire image is very computationally expensive.
    - With 96x96 images, you would have about 10^4 input units, and assuming you want to learn 100 features, you would have on the order of 10^6 parameters to learn.
    - The feedforward and backpropagation computations would also be about 10^2 times slower, compared to 28x28 images.

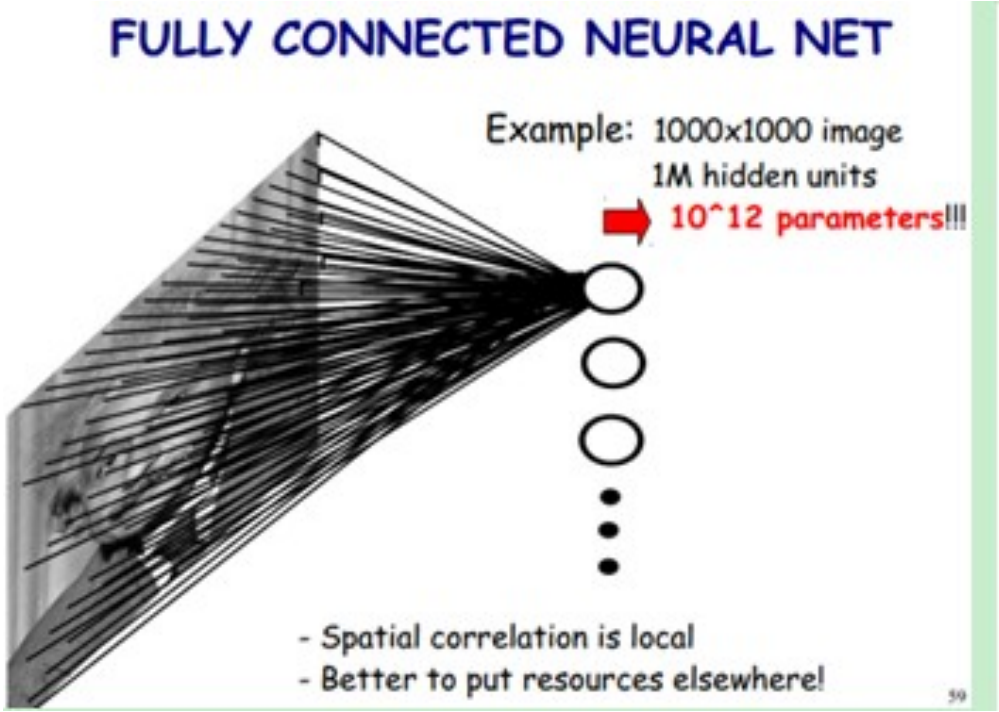
Locally Connected Networks

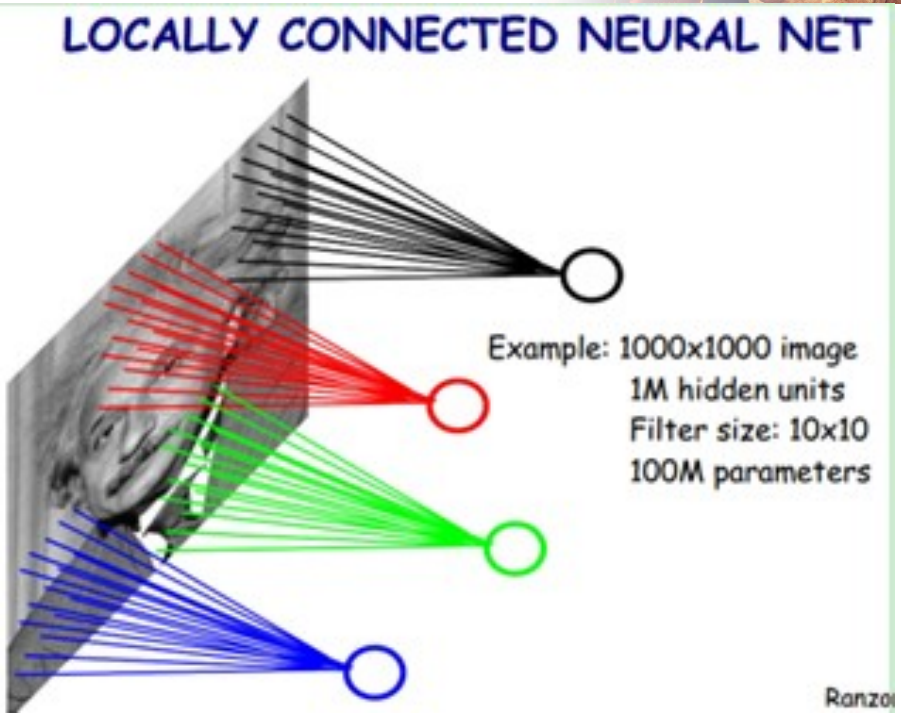
- One simple solution:
  - to restrict the connections between the hidden units and the input units, allowing each hidden unit to connect to only a small subset of the input units.
  - Specifically, **each hidden unit will connect to only a small contiguous region of pixels in the input**.
    - there is often also a natural way to select "contiguous groups" of input units to connect to a single hidden unit as well;
  - This idea of having locally connected networks also draws inspiration from how the early visual system is wired up in biology.
    - Specifically, neurons in the visual cortex have localized receptive fields (i.e., they respond only to stimuli in a certain location).

Locally Connected Networks

- Natural images have the property of being "**stationary**"
  - meaning that the statistics of one part of the image are the same as any other part.
- So, **the features that we learn at one part of the image can also be applied to other parts of the image**, and we can use the same features at all locations.
  - More precisely, having learned features over small (say 8x8) patches sampled randomly from the larger image, we can then apply this learned 8x8 feature detector anywhere in the image.
  - Specifically, we can take the learned 8x8 features and **convolve** them with the larger image, thus obtaining a different feature activation value at each location in the image.

Locally Connected Networks



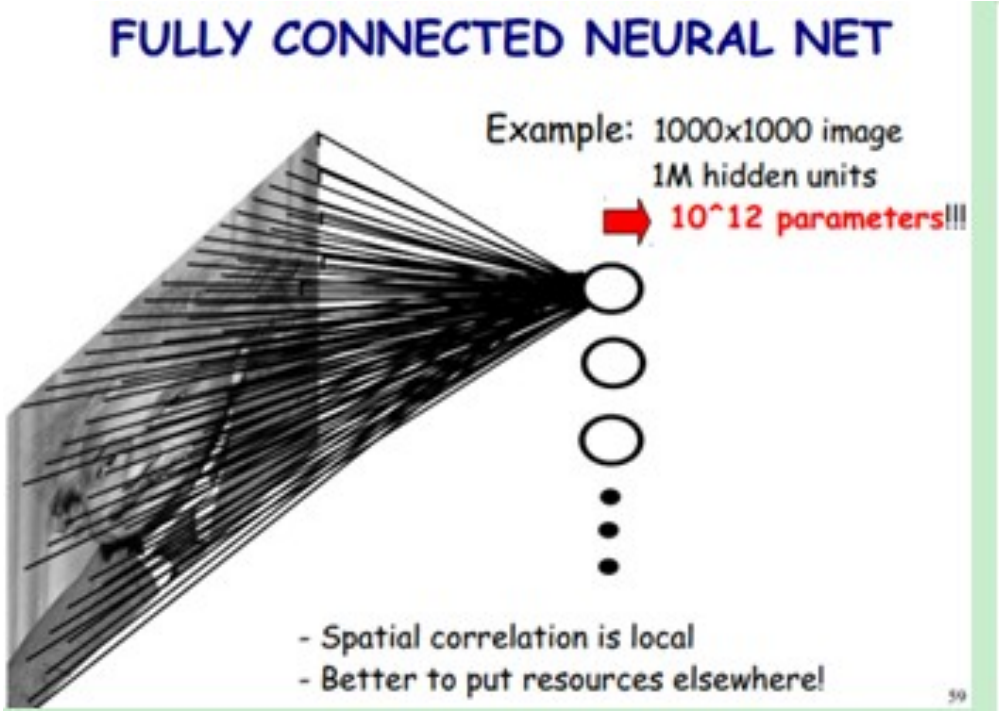


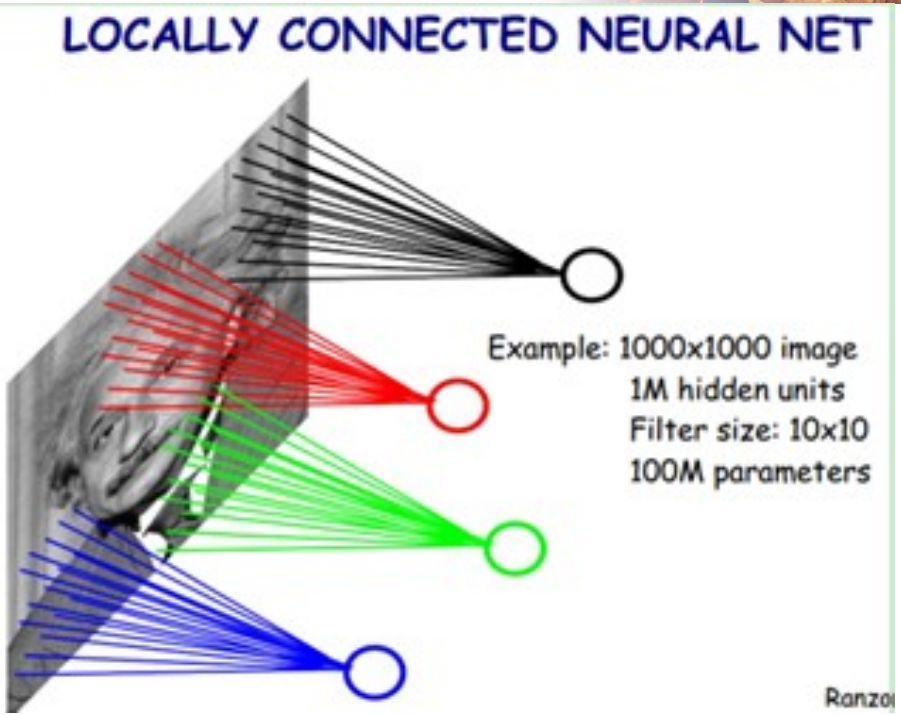
Suppose there are 1M hidden units:

Left: $1000 \times 1000 \times 1M = 10^{12}$

Right: $10 \times 10 \times 1M = 10^{8}$

Weights Sharing





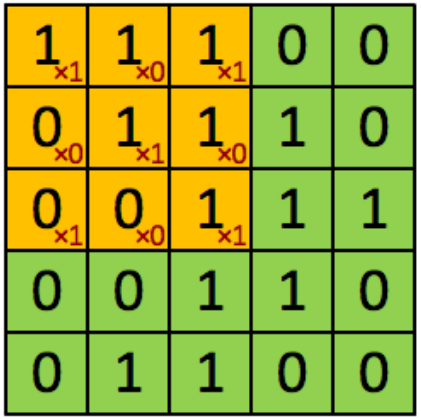
Suppose there are 1M hidden units:

Left: $1000\times1000\times1M = 10^{12}$

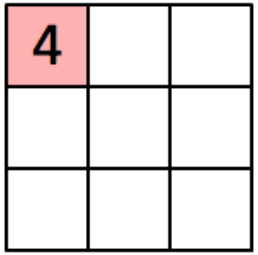
Right: $10\times10\times1 = 10^{2}$

Convolutions

- Suppose you want to learned 9 features from a 5x5 image.
  - With Fully Connected Neural Networks:
    $5 \times 5 \times 9 = 225$
  - With Locally Connected Neural Networks:
    $3 \times 3 \times 9 = 81$
  - With Weights Sharing:
    $3 \times 3 \times 1 = 9$

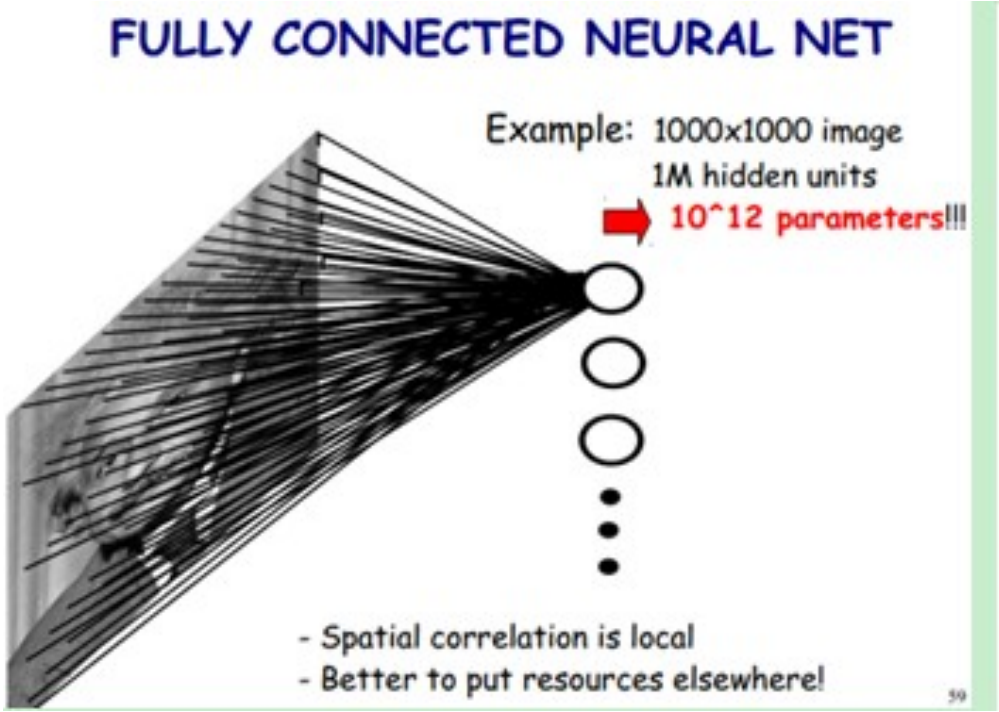


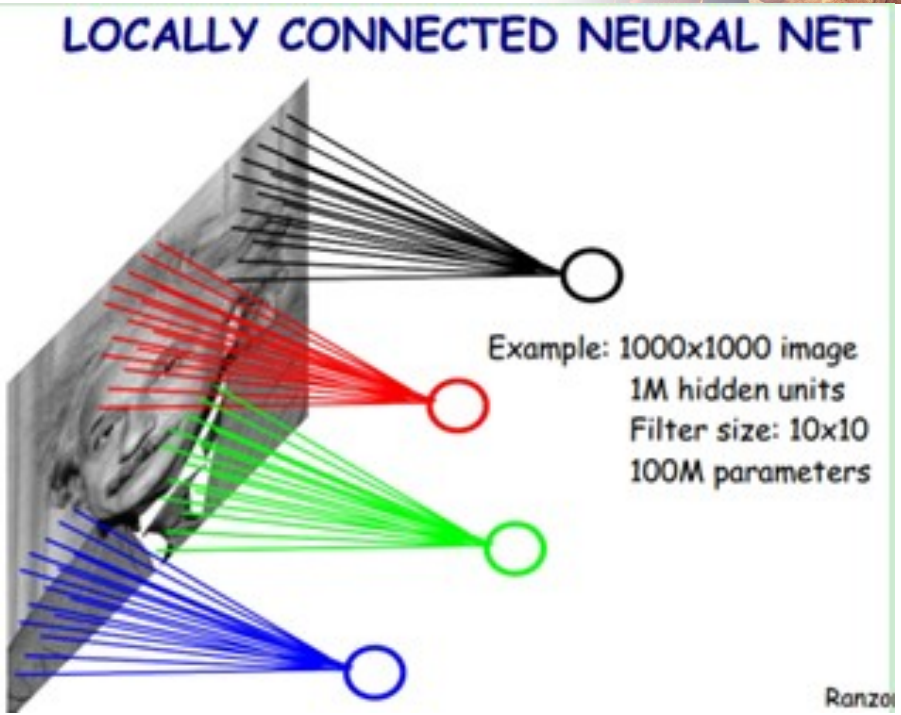
Image



Convolved Feature

Is 1 Hidden Unit Enough ? No !



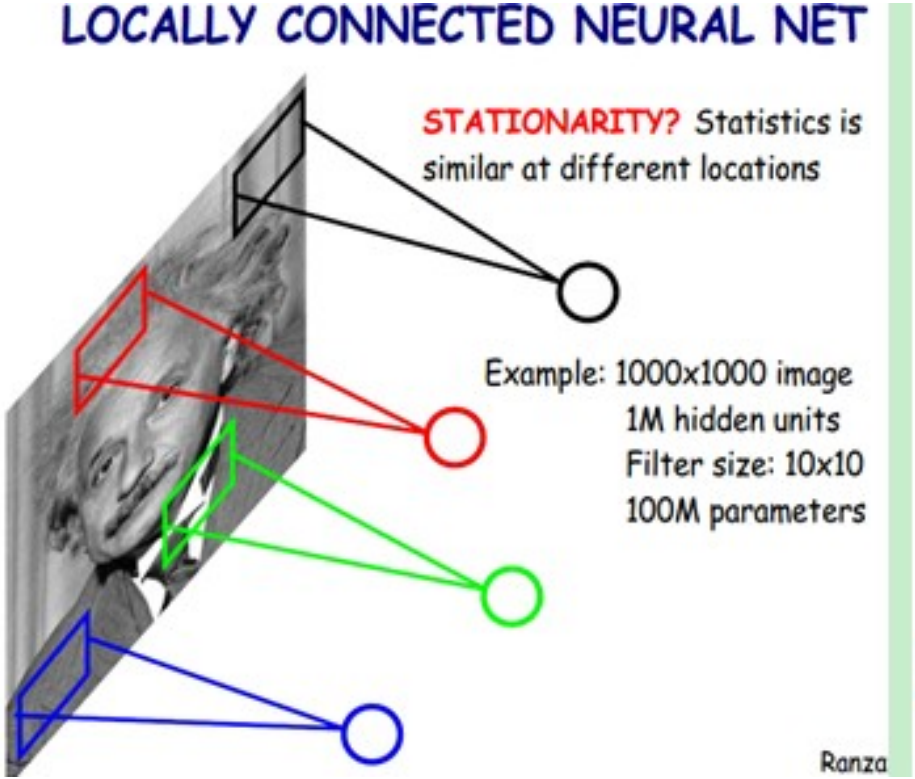


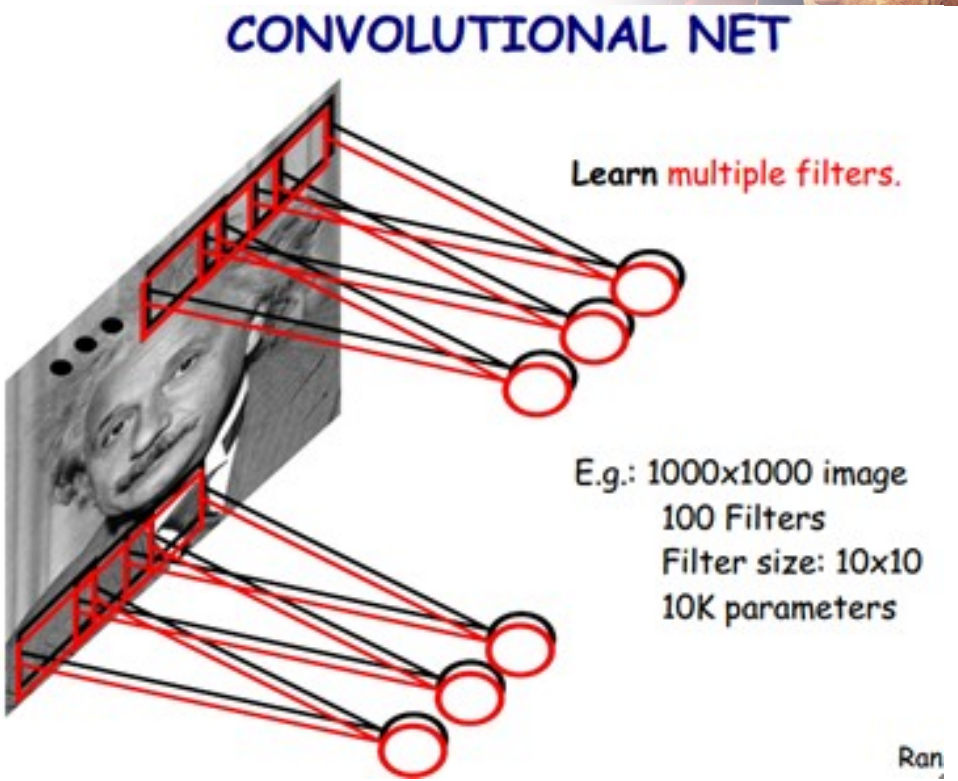
Suppose there are 1M hidden units:

Left: $1000\times1000\times1M = 10^{12}$

Right: $10\times10\times1 = 10^{2}$

Multiple Kernels Convolution



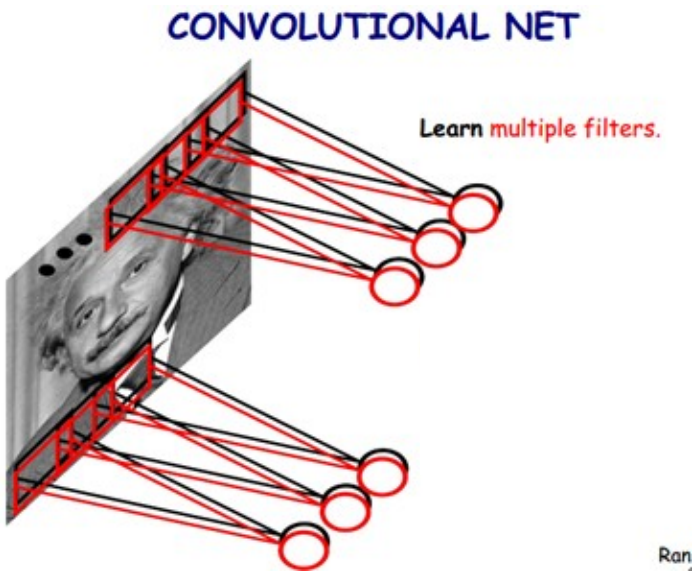


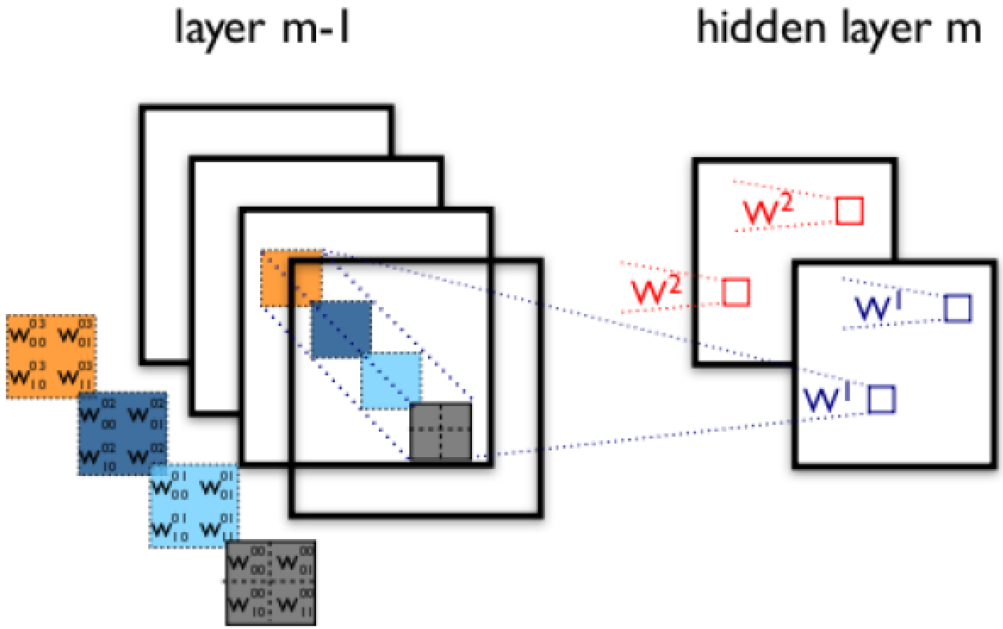
Left: $10\times10\times1M(\text{特征数}) = 10^8$

Right: $10\times10\times100$（特征数） $= 10^4$

Multiple Kernels Convolution

- 对于一张100×100的图片
  - One Kernel: $10\times10\times100 = 10^4$　　4 Kernels: $10^4\times4$
  - 2nd-Convolution: $2\times2\times4\times2 = 32$





Pooling

- If we use all the extracted features with a classifier, this can be computationally challenging.
  - images of size 96x96 pixels;
  - suppose we want to learn 400 features over 8x8 inputs;

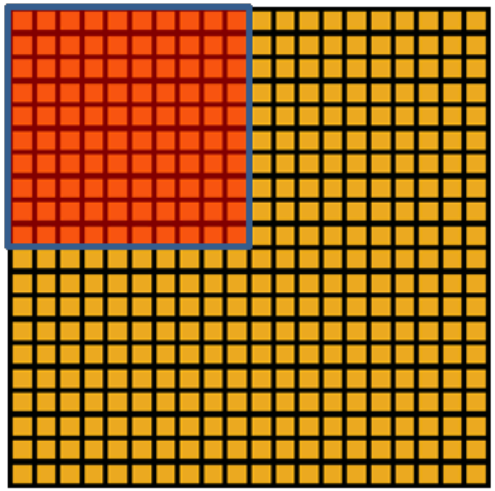
  Then we have to compute

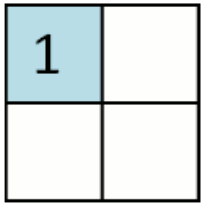
  $$(96-8+1)*(96-8+1)*400=3,168,400 \text{ features}$$

  Learning a classifier with inputs having 3+ million features can be unwieldy, and can also be prone to over-fitting.
- Pooling: to describe a large image, we can aggregate statistics of these features at various locations.

Pooling

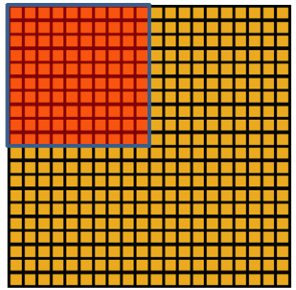
- Subsampling: “mean pooling” or “max pooling”
  - one could compute the mean (or max) value of a particular feature over a region of the image.
  - These summary statistics are much lower in dimension and can also improve results (less over-fitting).



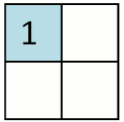


Convolutional Neural Networks

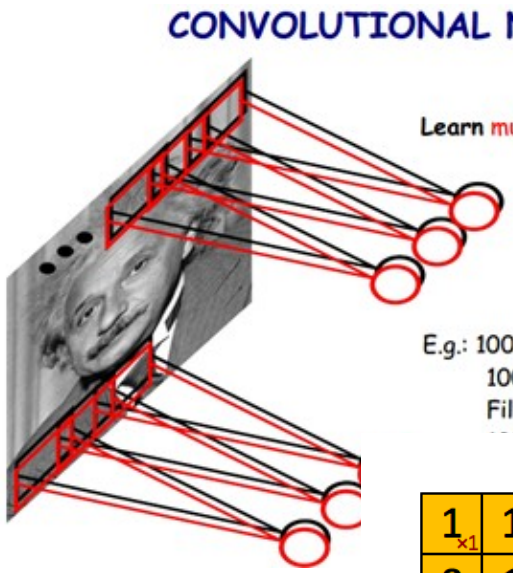
- Composed of
  - Convolution Layers
  - Pooling Layers

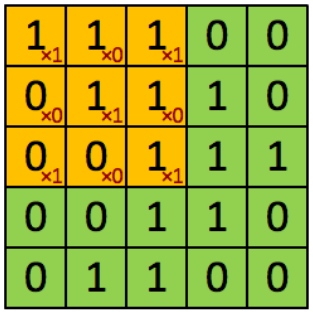


Convolved feature

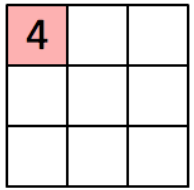


Pooled feature





Image



Convolved Feature

CNN的前向传播与后向传播

CNN前向传播

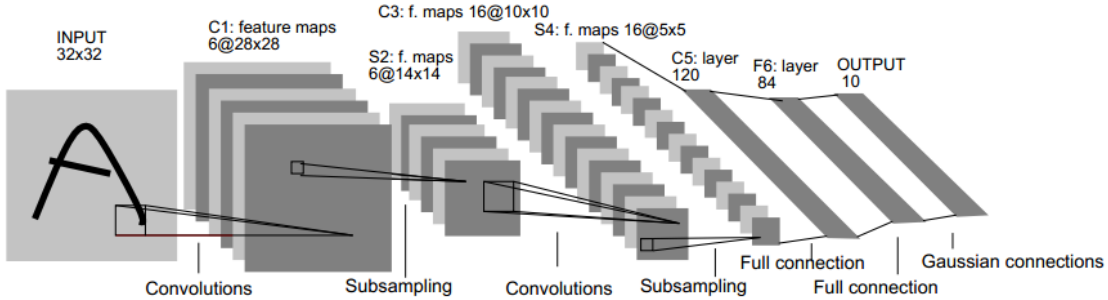
- Forward Propagation
  - ℓ 表示当前层，x^{ℓ}表示当前层输出， b^{ℓ}当前层偏置

$$\mathbf{x}^{\ell} = f(\mathbf{u}^{\ell}), \quad \text{with} \quad \mathbf{u}^{\ell} = \mathbf{W}^{\ell}\mathbf{x}^{\ell-1} + \mathbf{b}^{\ell}$$

$$f(x) = (1 + e^{-\beta x})^{-1}$$

或

$$f(x) = a\tanh(bx)$$



CNN前向传播

- 代价函数E
  - 以 平方误差 代价函数 为例

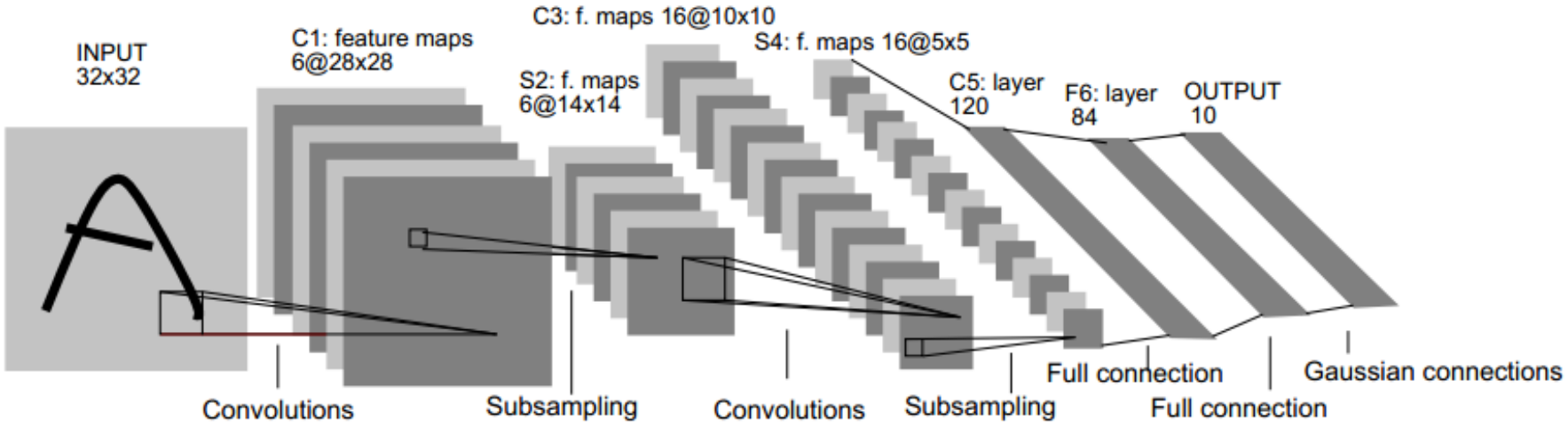
$$E^N = \frac{1}{2}\sum_{n=1}^{N}\sum_{k=1}^{c}(t_k^n - y_k^n)^2$$

  - t_k^n 第n个训练样本 对应的 输出层第K个神经元上的样本标签
  - y_k^n 第n个训练样本 对应的 输出层第K个神经元上的实际输出
- 单个样本的代价函数：

$$E^n = \frac{1}{2}\sum_{k=1}^{c}(t_k^n - y_k^n)^2 = \frac{1}{2}\|\mathbf{t}^n - \mathbf{y}^n\|_2^2.$$

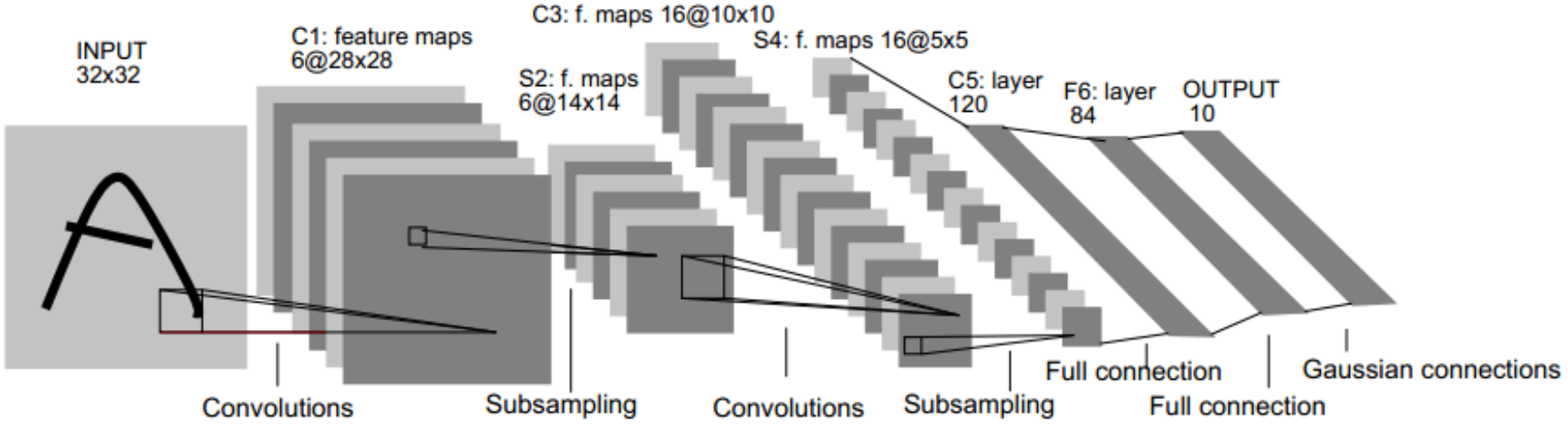
LeNet-5

- C1层是一个卷积层
  - 6个特征图，每个特征图中的每个神经元与输入中5*5的邻域相连，特征图大小为28*28，
  - 每个卷积神经元的参数数目：5*5=25个unit参数和一个bias参数，
  - 连接数目： (5*5+1)*6*(28*28)=122,304个连接
  - 参数共享：每个特征图内共享参数，因此参数总数：共(5*5+1)*6=156个参数



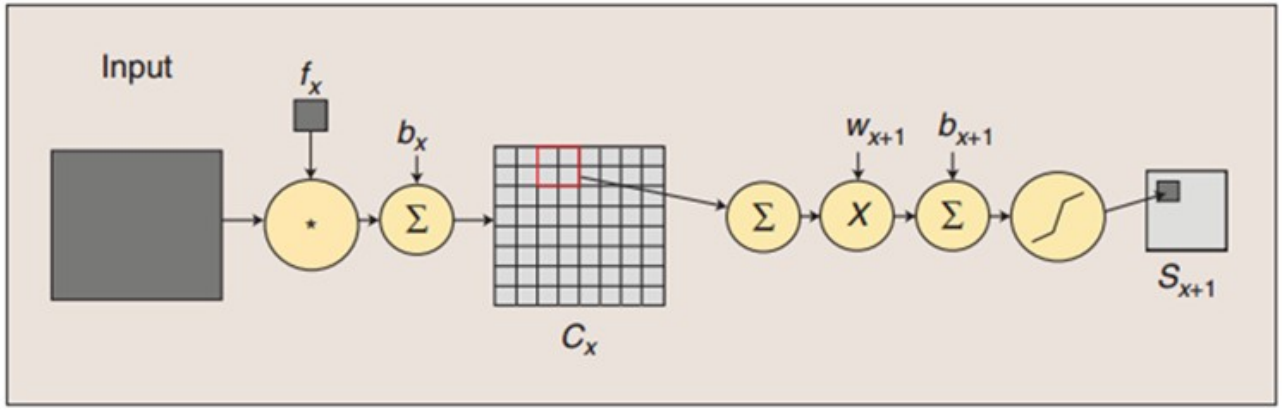
LeNet-5

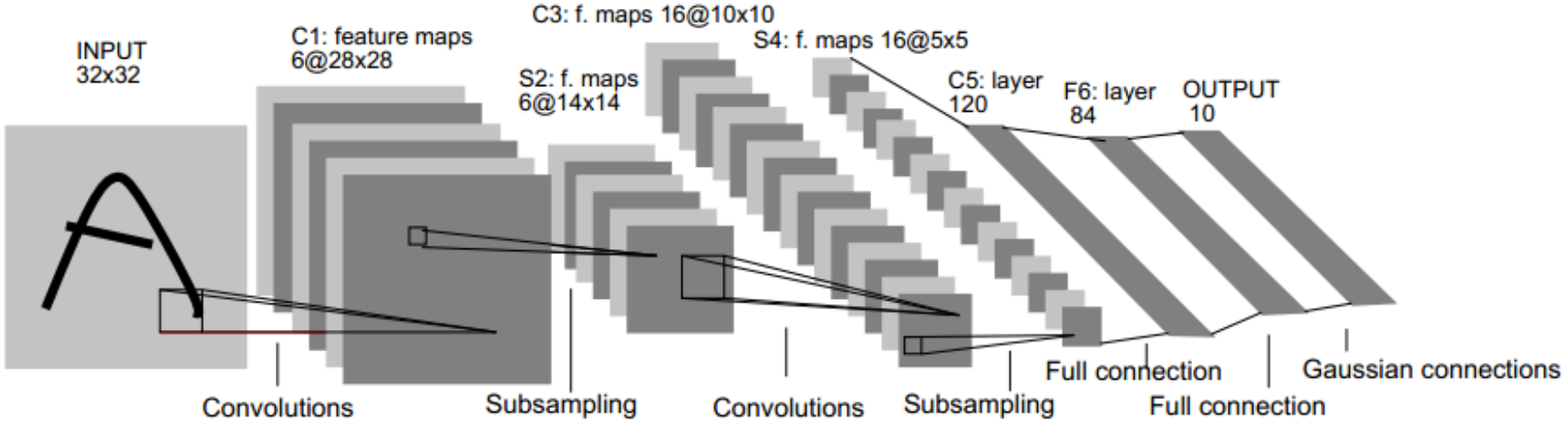
- S2层是一个下采样层
  - 6个14*14的特征图，每个图中的每个单元与C1特征图中的一个2*2邻域相连接，不重叠。因此，S2中每个特征图的大小是C1中特征图大小的1/4.
  - S2层每个单元的4个输入相加，乘以一个可训练参数w，再加上一个可训练偏置b，结果通过sigmoid函数计算。
  - 连接数：（2*2+1）*1*14*14*6 = 5880个
  - 参数共享：每个特征图内共享参数，因此有(2*2+1)*6=30个可训练参数



LeNet-5

- LeCun的表示法

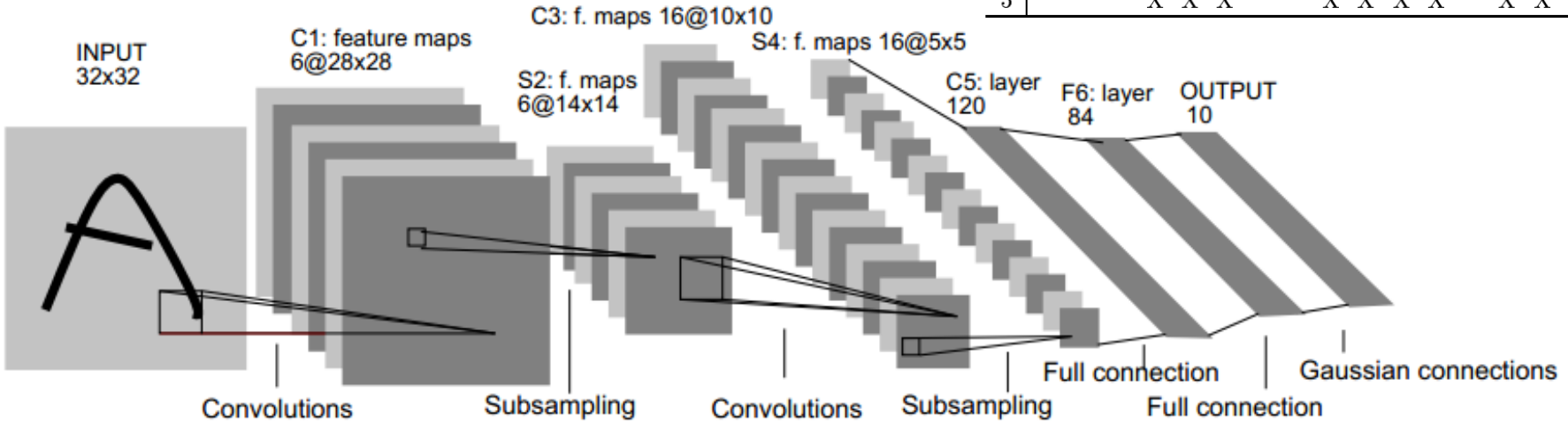




LeNet-5

- C3层是一个卷积层
  - 16个卷积核，得到16张特征图，特征图大小为10*10；
  - 每个特征图中的每个神经元与S2中某几层的多个5*5的邻域相连；

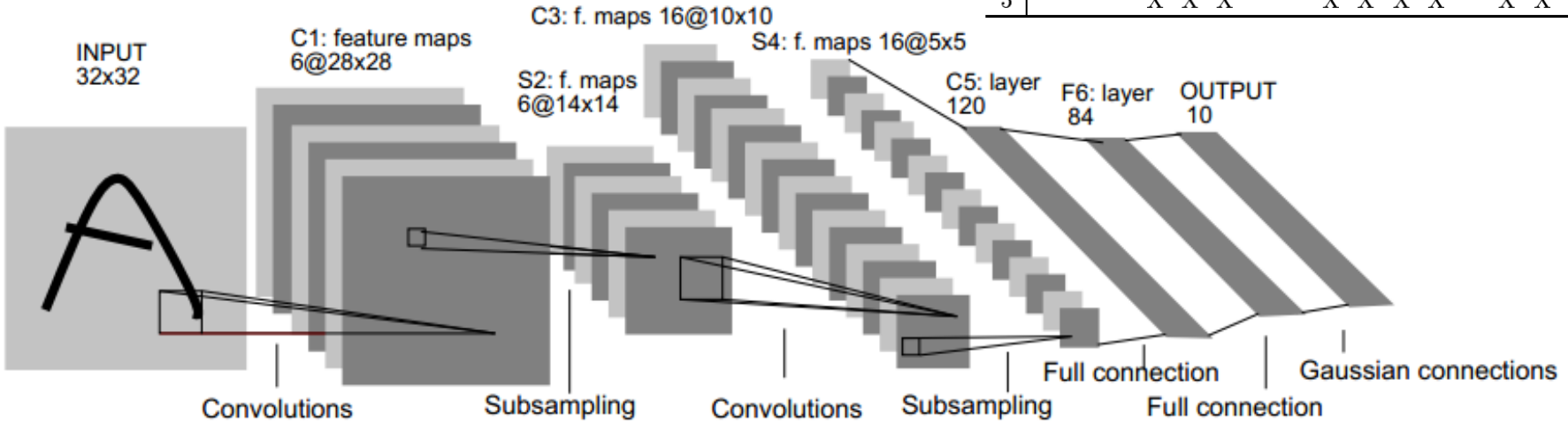
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X



LeNet-5

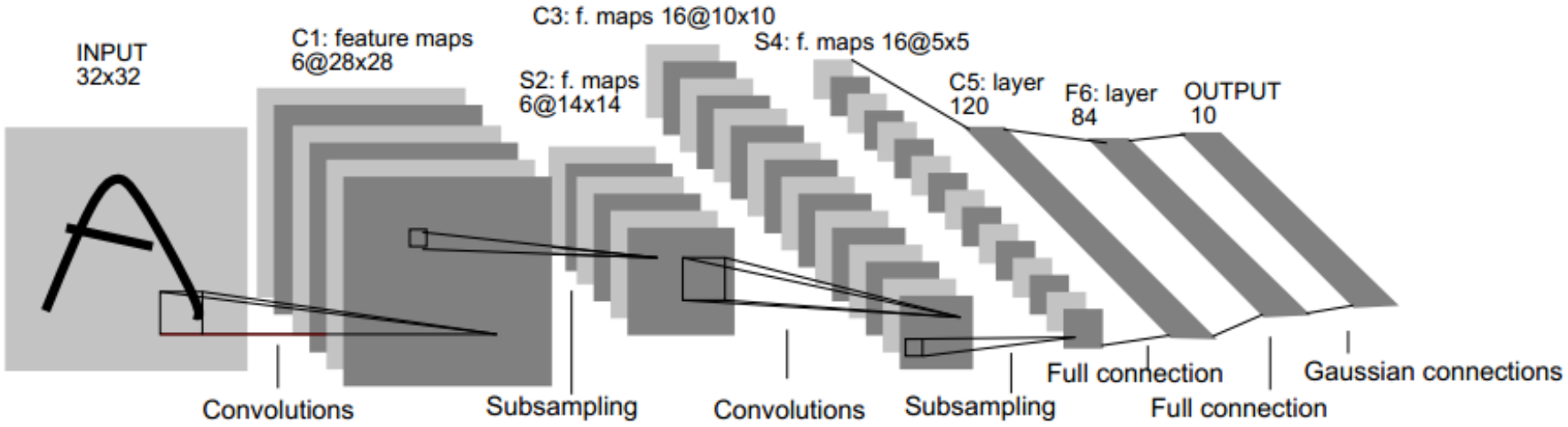
- C3层是一个卷积层
  - 16个卷积核，得到16张特征图，特征图大小为10*10；
  - 每个特征图中的每个神经元与S2中某几层的多个5*5的邻域相连；
    - 例如，对于C3层第0张特征图，其每一个节点与S2层的第0张特征图，第1张特征图，第2张特征图，总共3个5x5个节点相连接。

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X



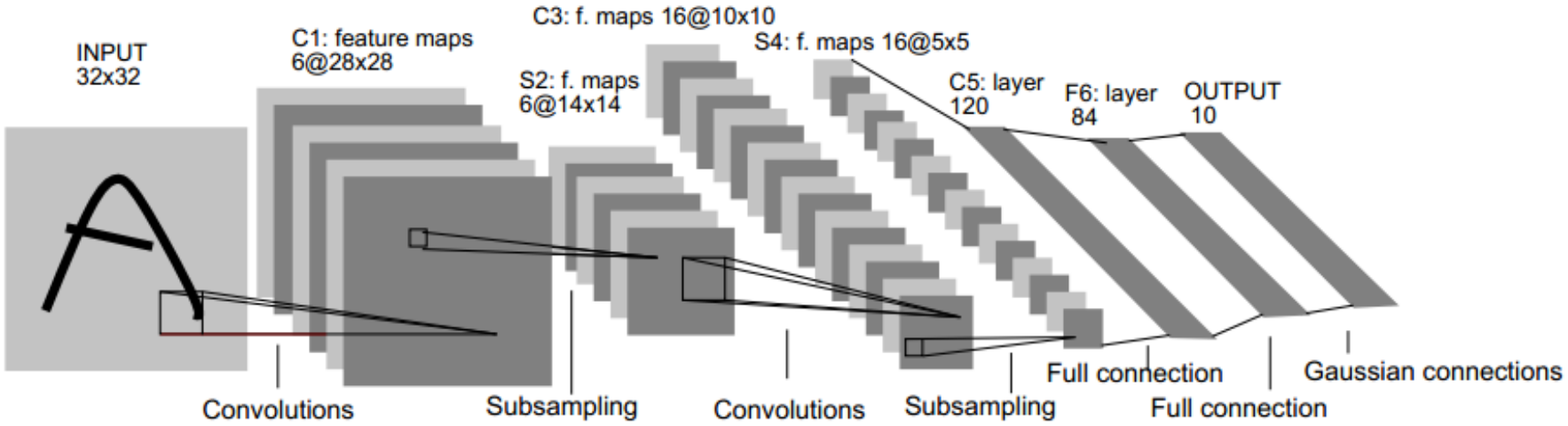
LeNet-5

- S4层是一个下采样层
  - 由16个5*5大小的特征图构成，特征图中的每个单元与C3中相应特征图的2*2邻域相连接；
  - 连接数：（2*2+1）*5*5*16=2000个
  - 参数共享：特征图内共享参数，每张特征图中的每个神经元需要1个因子和一个偏置，因此有 2*16 个可训练参数



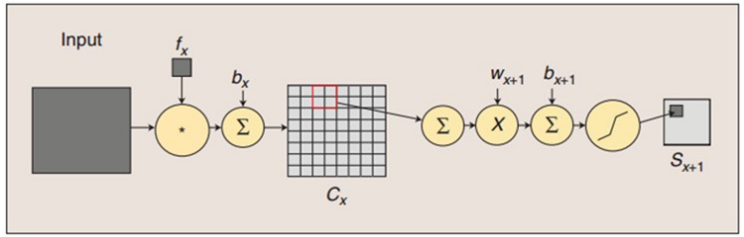
LeNet-5

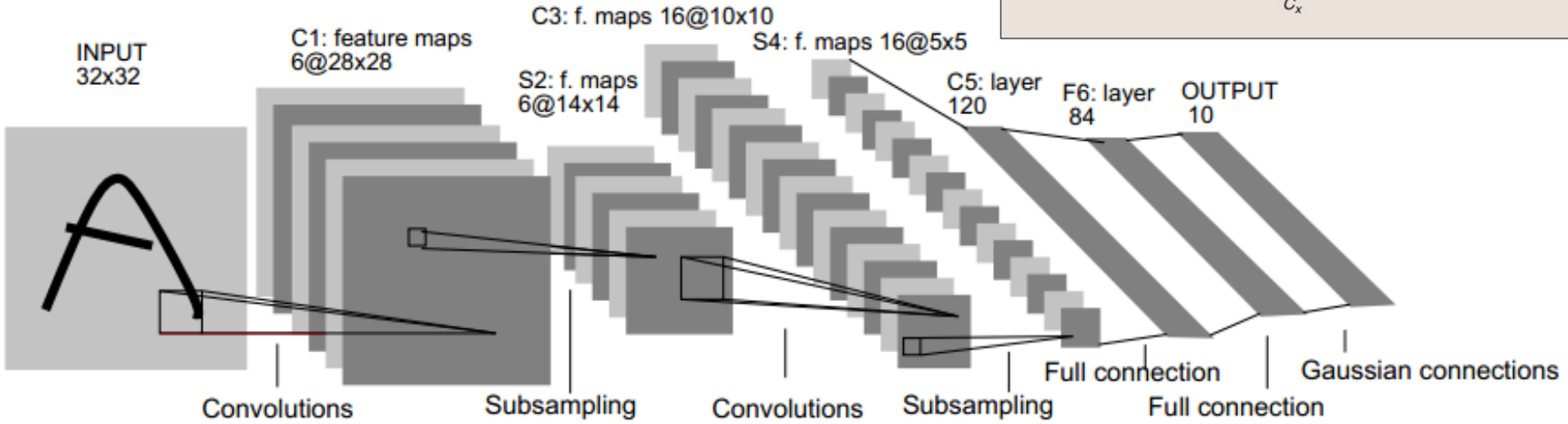
- C5层
  - 120个神经元，可以看作120个特征图，每张特征图的大小为1*1
  - 每个单元与S4层的全部16个单元的5*5邻域相连（S4和C5之间的全连接）
  - 连接数=可训练参数：（5*5*16+1）*120=48120个



LeNet-5

- F6层
  - 有84个单元（之所以选这个数字的原因来自于输出层的设计），与C5层全相连。
  - F6层计算输入向量和权重向量之间的点积，再加上一个偏置。
  - 连接数=可训练参数：（120+1）*84=10164
  - 84：stylized image：7*12





LeNet-5

- 输出层采用欧式径向基函数（Euclidean Radial Basis Function）单元
  - 给定一个输入模式，损失函数应能使得F6的配置与RBF参数向量（即模式的期望分类）足够接近。
  - 每类一个单元，每个单元连接84个输入；每个输出RBF单元计算输入向量和参数向量之间的欧式距离。
  - RBF输出可以被理解为F6层配置空间的高斯分布的【-log-likelihood】

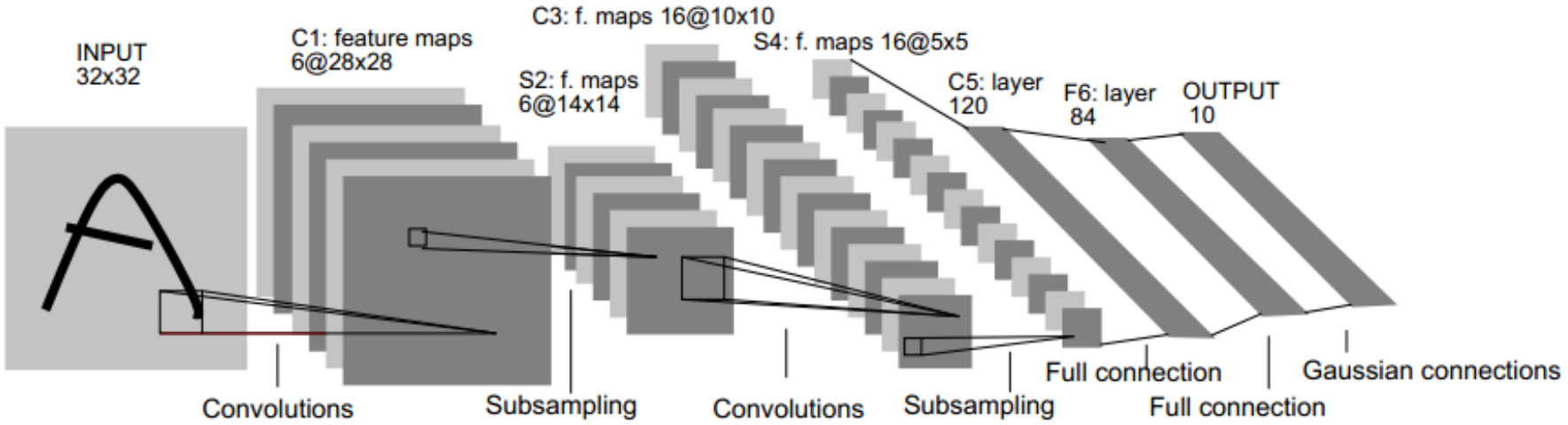


Table of contents

卷积神经网络

卷积函数

卷积函数，是关于两个函数的函数设存在原始函数为：$f(x)$；
设存在对原始函数的输出进行权重修正的函数：$w(t-x)$；则：

$$s(t) = \int_0^t f(x)w(t-x)dx$$

该卷积操作通常表示为：

$$s(t) = (f * w)(t)$$

其中：

- $w(t-x)$ 表示对“$f(x)$ 在 x 点的值”进行加权的权值；
- 它是 t 的函数，t 是与 x 同一维度的自变量，$t-x$ 表示当前 t 点对 x 点的距离；
- 也就是说，$w(t-x)$ 是一个随“t 点对 x 点的距离”而变化的函数；

卷积函数

在离散的情况下，可以把卷积函数写成：

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

在 x 为多维数据的情况下，如对于二维数据（灰度图像），可以将上式重写为：

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n)$$

其中：

- 函数 I 表示输入，$I(m,n)$ 为图像在 (m,n) 点的灰度值;
- $K(i-m, j-n)$ 表示对“图像在 (m,n) 点的灰度值 $I(m,n)$”的加权值；
- $i-m$（或 $j-n$）表示 i 点到 m 点（或 j 点到 n 点）的距离；
- 加权值 $K(i-m, j-n)$ 是“i 点到 m 点（或 j 点到 n 点）的距离”的函数；

卷积函数

对公式：

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n)$$

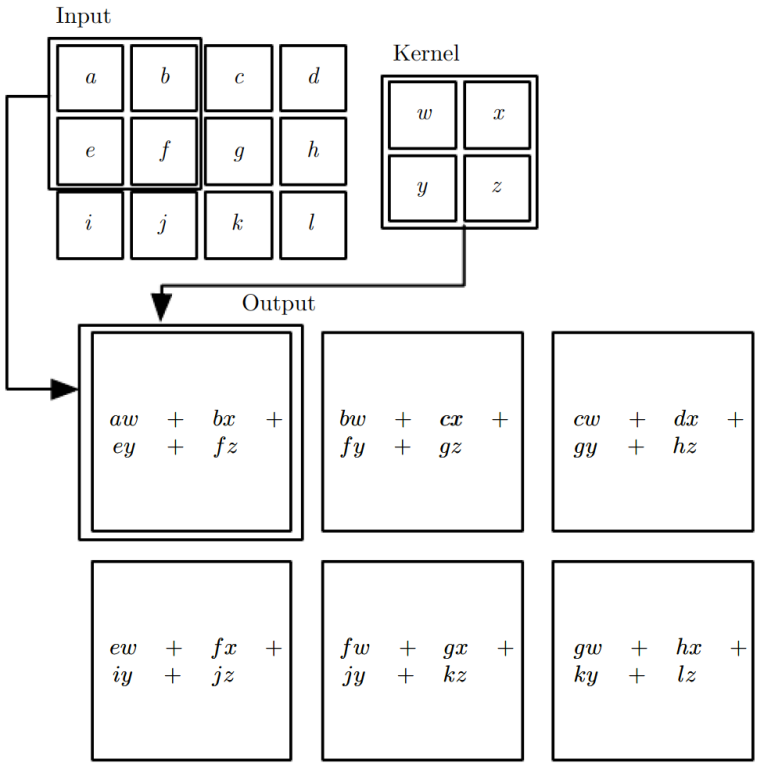
进行近似等价重写，得到 Cross-Correlation（互相关）公式：

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n)$$

这就是我们常见的“卷积”公式！其中：

- $I(i+m, j+n)$ 表示以 (i,j) 为起点，以 (m,n) 为宽度和高度的输入区域的灰度值矩阵；
- $K(m,n)$ 表示宽度为 m，高度为 n 的卷积核（$m \times n$ 的矩阵）；
- $S(i,j)$ 表示“以 (i,j) 为起点，宽度为 m，高度为 n 的灰度值矩阵”经过卷积核 K 进行卷积计算的值；
- S 的大小由 (i,j) 的最大值确定，也可以说，由输入区域 I 和卷积核 K 共同决定；

卷积运算



$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n)$$

$$S(0,0) = (I * K)(0,0) = \sum_m \sum_n I(0+m, 0+n) K(m,n)$$

$$S(1,0) = (I * K)(1,0) = \sum_m \sum_n I(1+m, 0+n) K(m,n)$$

$$S(2,0) = (I * K)(2,0) = \sum_m \sum_n I(2+m, 0+n) K(m,n)$$

$$S(0,1) = (I * K)(0,1) = \sum_m \sum_n I(0+m, 1+n) K(m,n)$$

$$S(1,1) = (I * K)(1,1) = \sum_m \sum_n I(1+m, 1+n) K(m,n)$$

$$S(2,1) = (I * K)(2,1) = \sum_m \sum_n I(2+m, 1+n) K(m,n)$$

卷积运算

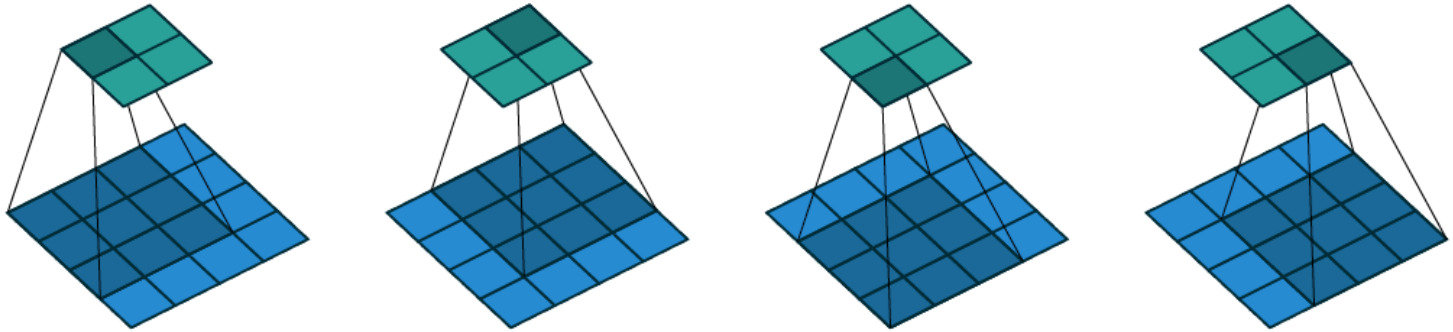


Figure 2.1: (No padding, no strides) Convolving a 3×3 kernel over a 4×4 input using unit strides (i.e., $i = 4$, $k = 3$, $s = 1$ and $p = 0$).

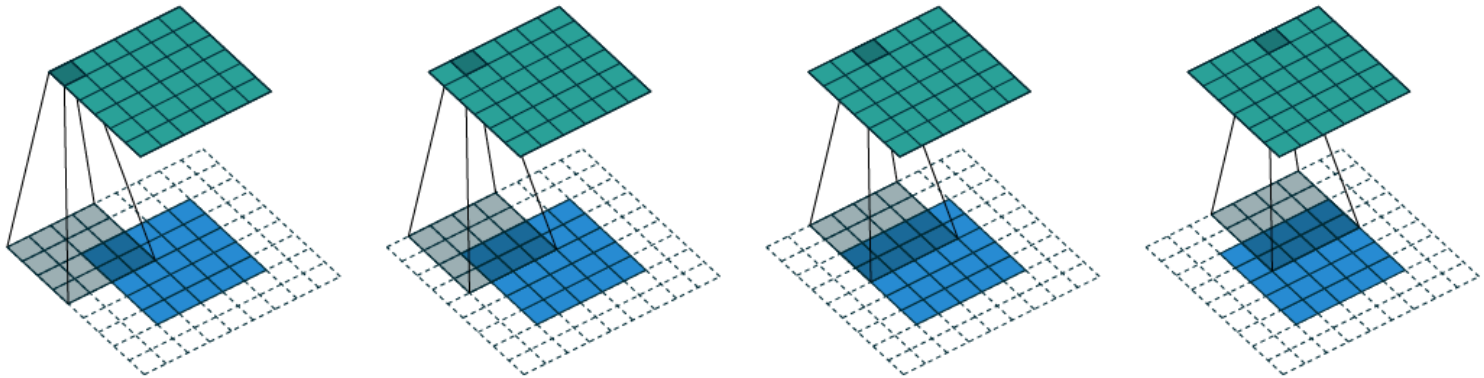


Figure 2.2: (Arbitrary padding, no strides) Convolving a 4×4 kernel over a 5×5 input padded with a 2×2 border of zeros using unit strides (i.e., $i = 5$, $k = 4$, $s = 1$ and $p = 2$).

卷积运算

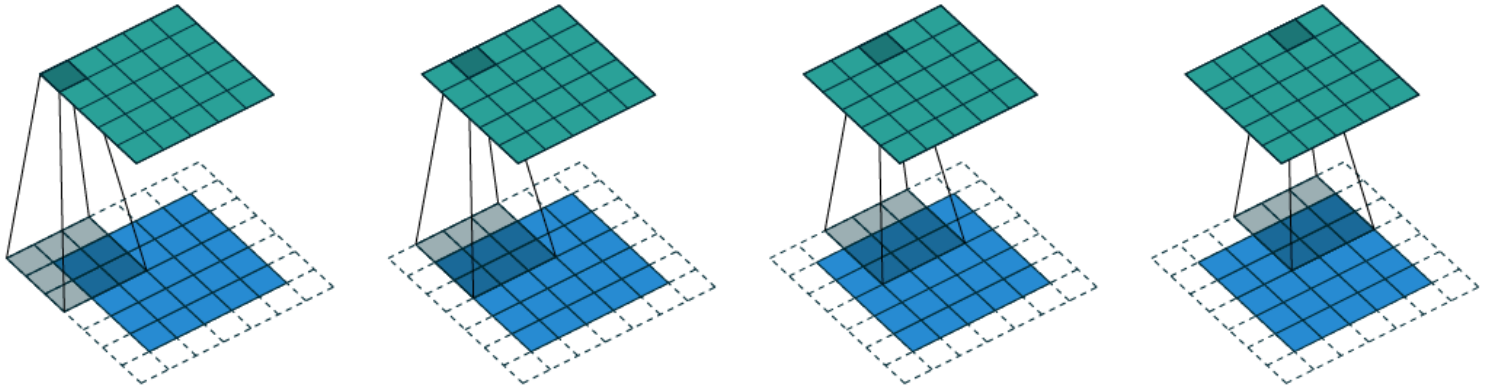


Figure 2.3: (Half padding, no strides) Convolving a 3×3 kernel over a 5×5 input using half padding and unit strides (i.e., $i = 5$, $k = 3$, $s = 1$ and $p = 1$).

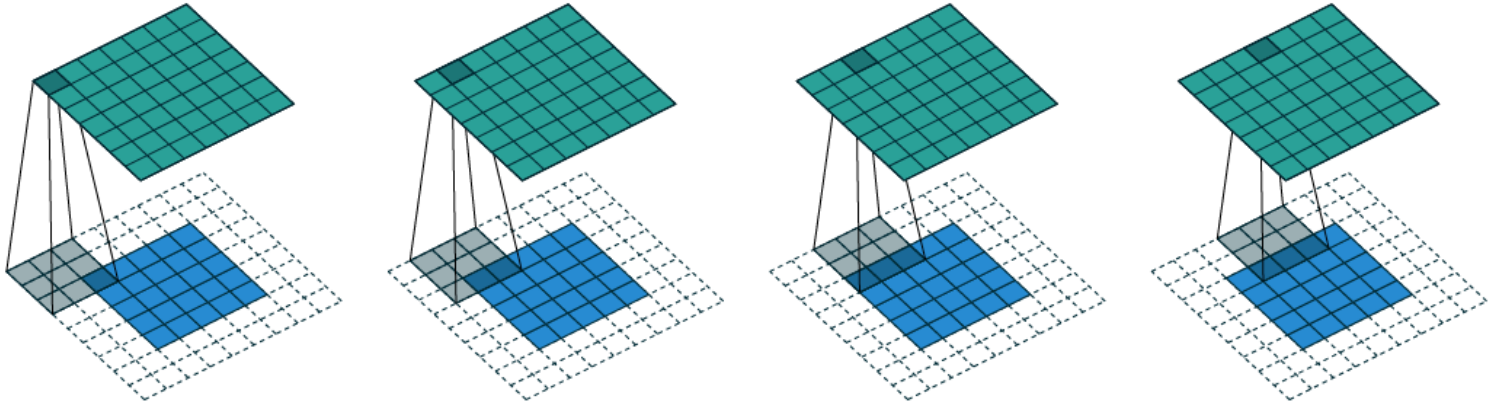


Figure 2.4: (Full padding, no strides) Convolving a 3×3 kernel over a 5×5 input using full padding and unit strides (i.e., $i = 5$, $k = 3$, $s = 1$ and $p = 2$).

卷积运算的物理含义

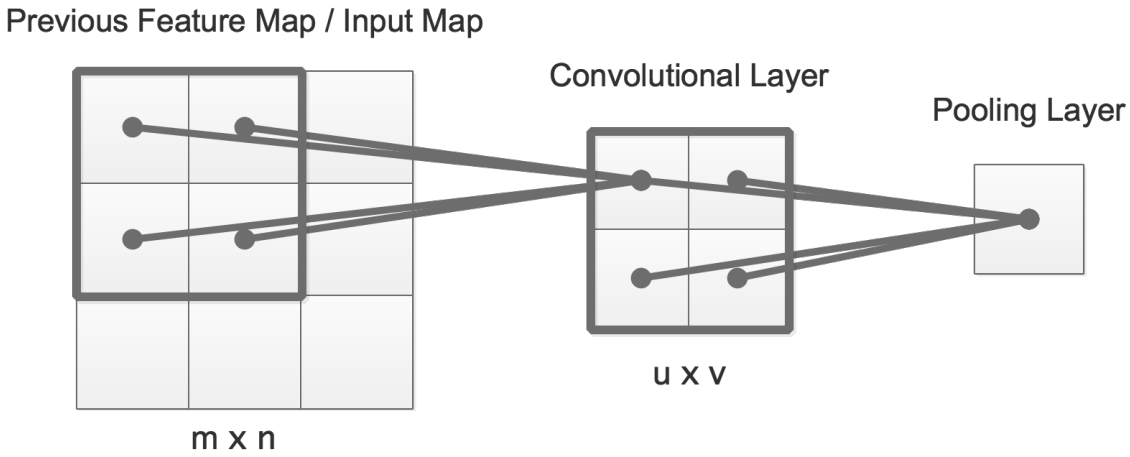
$$K_{horizontal\_high\_magnitude} = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

(a) Lenna

(b) Horizontal edge

(c) Vertical edge

卷积层的前向计算



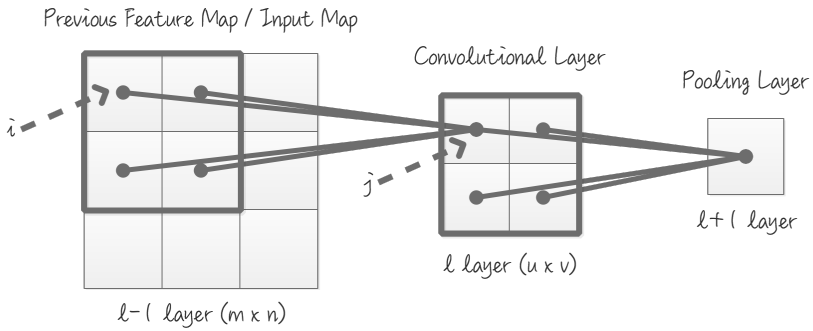
$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n)$$

可以写成：

$$z_{(u,v)}^{(l)} = \sum_{(m,n) \in M_{(u,v)}} a_{(m,n)}^{(l-1)} * k_{(u,v)(m,n)}^{(l)} + b_{(u,v)}^{(l)}$$

$$a_{(u,v)}^{(l)} = f(z_{(u,v)}^{(l)})$$

卷积层的前向计算



$$z_{(u,v)}^{(l)} = \sum_{(m,n)\in M_{(u,v)}} a_{(m,n)}^{(l-1)} * k_{(u,v)(m,n)}^{(l)} + b_{(u,v)}^{(l)}$$

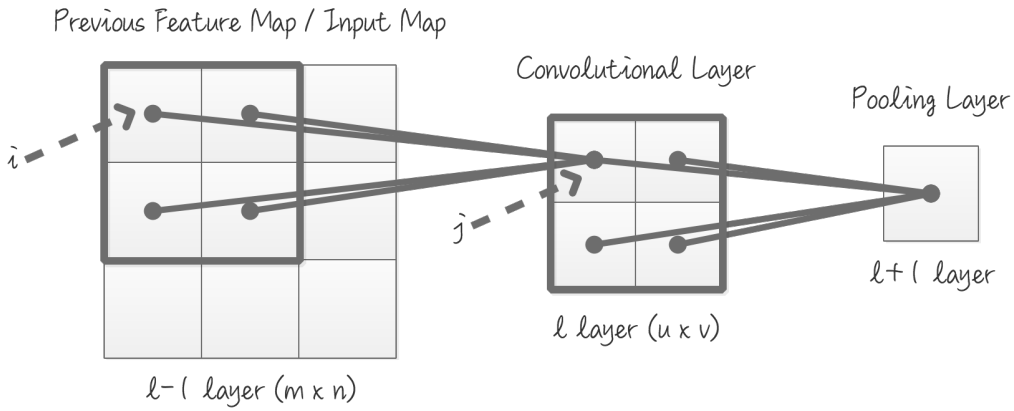
$$a_{(u,v)}^{(l)} = f(z_{(u,v)}^{(l)})$$

对上式进行简化：

$$z_j^{(l)} = \sum_{i\in M_j} a_i^{(l-1)} * k_{ji}^{(l)} + b_j^{(l)} \qquad a_j^{(l)} = f(z_j^{(l)})$$

上式中 $i \in M_j$ 表示：i 在与卷积层节点 j 所对应的输入窗口 M_j 中；

Pooling 层的前向计算



$$z_k^{(l+1)} = \beta_k^{(l+1)} down_{j \in M_k}(a_j^{(l)}) + b_k^{(l+1)} \qquad a_k^{(l+1)} = f_{pooling}(z_k^{(l+1)})$$

其中：

- $down(\cdot)$ 为下采样函数，$j \in M_k$ 表示：j 在与 Pooling 层节点 k 所对应的卷积层的窗口 M_k 中；
- 常见的下采样函数如：取平均 (Mean Pooling) 或取最大值 (Max Pooling)；
- 常数参数 $\beta_k^{(l+1)}$ 可以取 1；偏置参数 $b_k^{(l+1)}$ 可以取 0；

卷积网络的权重计算

$$w_{ji}^{(l)} = w_{ji}^{(l)} - \alpha \frac{\partial J(W,b)}{\partial w_{ji}^{(l)}} = w_{ji}^{(l)} - \alpha \delta_j^{(l)} a_i^{(l-1)}$$

$$= w_{ji}^{(l)} - \alpha \left(\sum_{k=1}^{n_{l+1}} \delta_k^{(l+1)} w_{kj}^{(l+1)} f'(z_j^{(l)}) \right) a_i^{(l-1)}$$

对照上式，对 $k_{ji}^{(l)}$ 进行求解：首先，在不考虑权值共享的前提下：

$$k_{ji}^{(l)} = k_{ji}^{(l)} - \alpha \frac{\partial J(W,b)}{\partial k_{ji}^{(l)}} = k_{ji}^{(l)} - \alpha \frac{\partial J(W,b)}{\partial z_j^{(l)}} \frac{\partial z_j^{(l)}}{\partial k_{ji}^{(l)}}$$

$$= k_{ji}^{(l)} - \alpha \frac{\partial J(W,b)}{\partial z_j^{(l)}} a_i^{(l-1)} = k_{ji}^{(l)} - \alpha \delta_j^{(l)} a_i^{(l-1)}$$

接下来，关键看 $\delta_j^{(l)}$ 怎么计算；

情况一：当前层之后为 Pooling 层

对照以前的推导方法：因为 $z_k^{(l+1)} = \sum_{j=1}^{n_{l+1}} w_{kj}^{(l+1)} a_j^{(l)} + b^{(l+1)}$ 所以可以选择从 $z_k^{(l+1)}$ 开始进行对 $z_j^{(l)}$ 进行求导。但此处$z_k^{(l+1)}$为何物呢？

情况一：假设，当前层之后为 Pooling 层；

则 $z_k^{(l+1)}$ 为 Pooling 层的激活函数的输入值：

$$\delta_j^{(l)} = \frac{\partial J(W,b)}{\partial z_j^{(l)}} = \frac{\partial J(W,b)}{\partial z_k^{(l+1)}} \frac{\partial z_k^{(l+1)}}{\partial a_j^{(l)}} \frac{\partial a_j^{(l)}}{\partial z_j^{(l)}} = \delta_k^{(l+1)} \frac{\partial z_k^{(l+1)}}{\partial a_j^{(l)}} f'(z_j^{(l)})$$

注意，其中：

$$z_k^{(l+1)} = \beta_k^{(l+1)} down_{j \in M_k}(a_j^{(l)}) + b_k^{(l+1)}$$

情况一：当前层之后为 Pooling 层

对上文的公式进行分析：

- $\frac{\partial z_k^{(l+1)}}{\partial a_j^{(l)}}$ 表达了在计算 $z_k^{(l+1)}$ 的过程中 $a_j^{(l)}$（其中 j 为：与 Pooling 层节点 k 所对应的卷积层的窗口 M_k 中的元素）的“贡献程度”；
- 因此，算式 $\frac{\partial z_k^{(l+1)}}{\partial a_j^{(l)}}$ 的计算中，$\beta_k^{(l+1)}$ 可以保留，即：

$$\delta_j^{(l)} = \left(\beta_k^{(l+1)} \delta_k^{(l+1)} f'(z_j^{(l)})\right) \frac{\partial z_k^{(l+1)}}{\partial a_j^{(l)}}$$

- 为保证可计算性和计算效率，可以用一个上采样函数 $up(\cdot)$ 来替代 $\frac{\partial z_k^{(l+1)}}{\partial a_j^{(l)}}$ 实现：根据“与 Pooling 层节点 k 所对应的卷积层的窗口 M_k 中的不同神经元输出的贡献程度”，对如上等式中的已知部分 $\left(\delta_k^{(l+1)} \beta_k^{(l+1)} f'(z_j^{(l)})\right)$ 进行分配；

情况一：当前层之后为 Pooling 层

上采样函数 $up(\cdot)$ 的选取：

- 若 Pooling 层的下采样函数采用 Mean Pooling，则该上采样函数可以取：

$$up(x) \equiv \frac{x \otimes 1_{n \times n}}{n \times n}$$

其中，$\otimes$ 为 Kronecker 乘积。

- 若 Pooling 层的下采样函数采用 Max Pooling，则该上采样函数可以通过记录 Max 值的来源位置，来实现；
- 在计算过程中，要保持“分配后的各个 $\delta_j^{(l)}$ 的和”与“Pooling 层已算得的 $\delta_k^{(l+1)}$”相等。

情况一：当前层之后为 Pooling 层

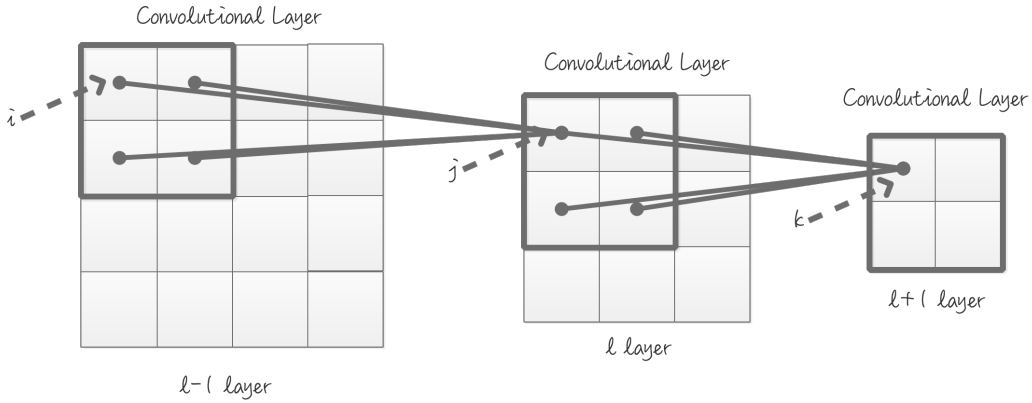
1	3
2	4

1	1	3	3
1	1	3	3
2	2	4	4
2	2	4	4

0.25	0.25	0.75	0.75
0.25	0.25	0.75	0.75
0.5	0.5	1	1
0.5	0.5	1	1

0	0	0	3
0	1	0	0
0	0	0	0
0	2	0	4

情况二：当前层之后为卷积层



情况二：假设当前层之后为卷积层；

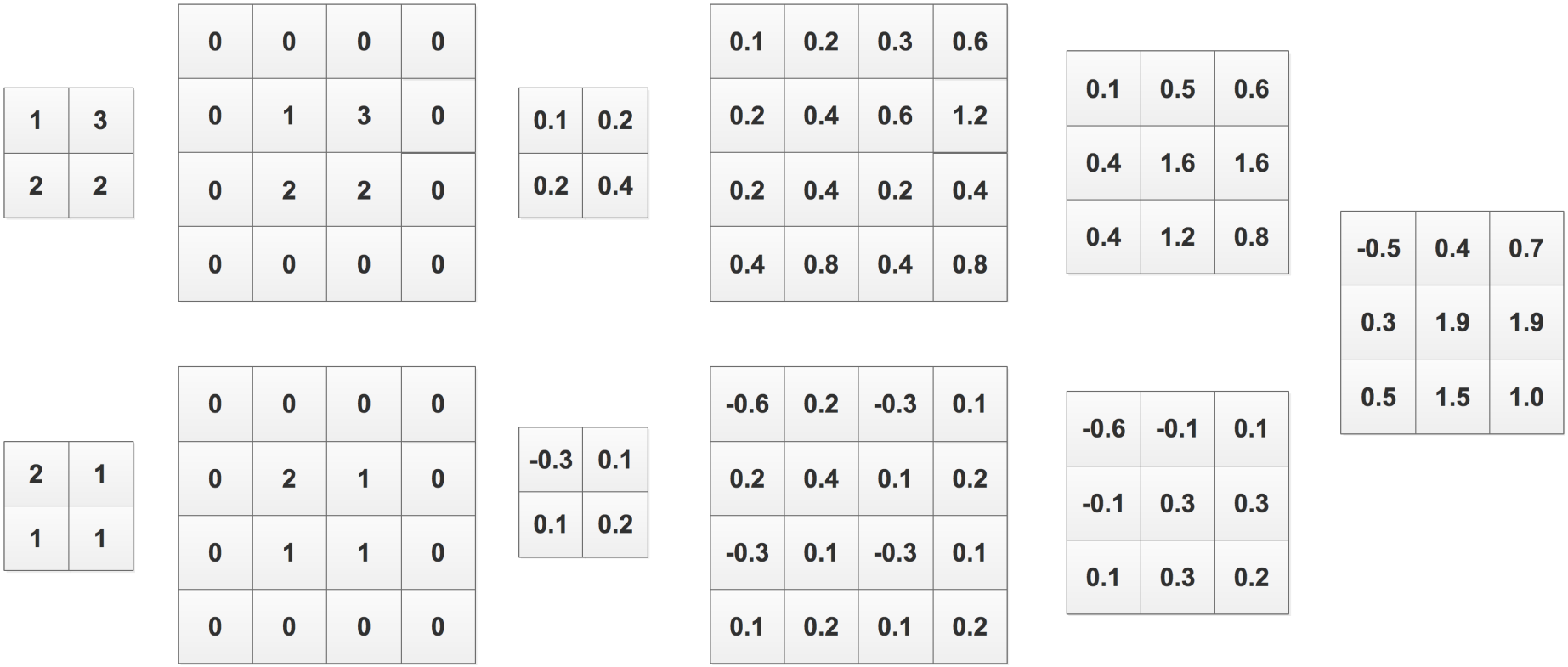
则，$z_k^{(l+1)}$ 为下一卷积层激活函数的输入值：

$$\delta_j^{(l)} = \frac{\partial J(W,b)}{\partial z_j^{(l)}} = \sum_K \sum_{k \in C_k \in K} \frac{\partial J(W,b)}{\partial z_k^{(l+1)}} \frac{\partial z_k^{(l+1)}}{\partial a_j^{(l)}} \frac{\partial a_j^{(l)}}{\partial z_j^{(l)}}$$

$$= \sum_K \sum_{k \in C_k \in K} \delta_k^{(l+1)} k_{kj}^{(l+1)} f'(z_j^{(l)})$$

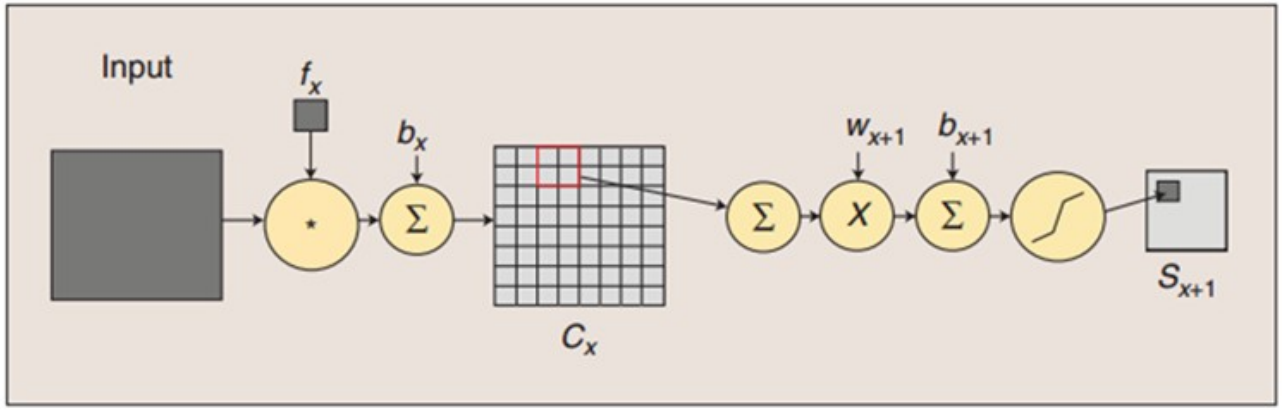
其中：C_k 为卷积运算中包含 $z_j^{(l)}$ 的 $l+1$ 层中的神经元的集合；K 为卷积核的数量；

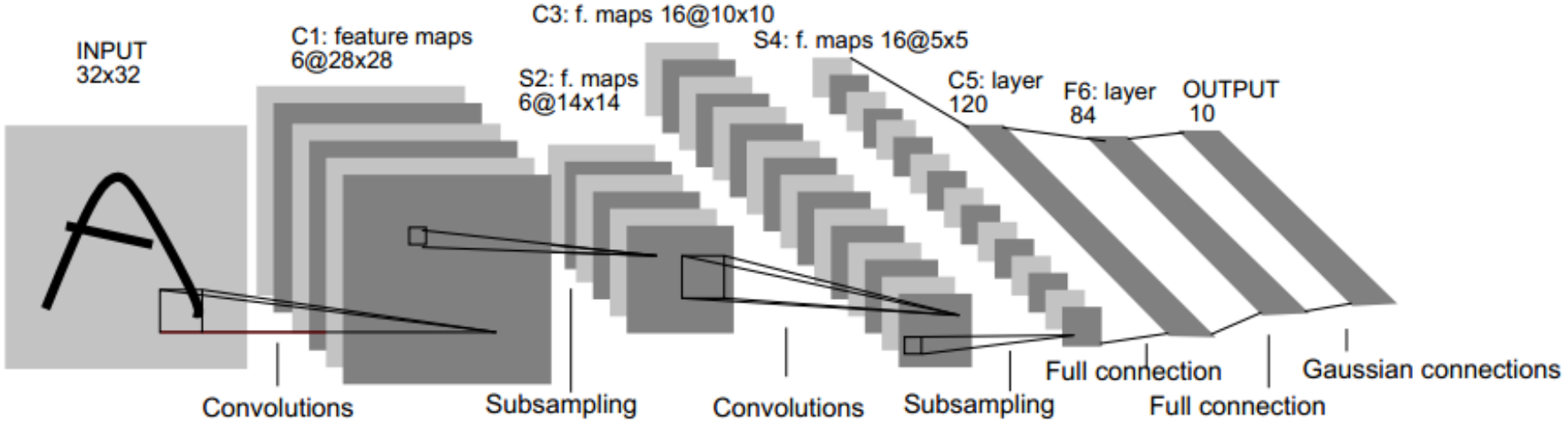
卷积网络的权重计算


1 3
2 2
0 0 0 0
0 1 3 0
0 2 2 0
0 0 0 0
0.1 0.2
0.2 0.4
0.1 0.2 0.3 0.6
0.2 0.4 0.6 1.2
0.2 0.4 0.2 0.4
0.4 0.8 0.4 0.8
0.1 0.5 0.6
0.4 1.6 1.6
0.4 1.2 0.8
-0.5 0.4 0.7
0.3 1.9 1.9
0.5 1.5 1.0
2 1
1 1
0 0 0 0
0 2 1 0
0 1 1 0
0 0 0 0
-0.3 0.1
0.1 0.2
-0.6 0.2 -0.3 0.1
0.2 0.4 0.1 0.2
-0.3 0.1 -0.3 0.1
0.1 0.2 0.1 0.2
-0.6 -0.1 0.1
-0.1 0.3 0.3
0.1 0.3 0.2

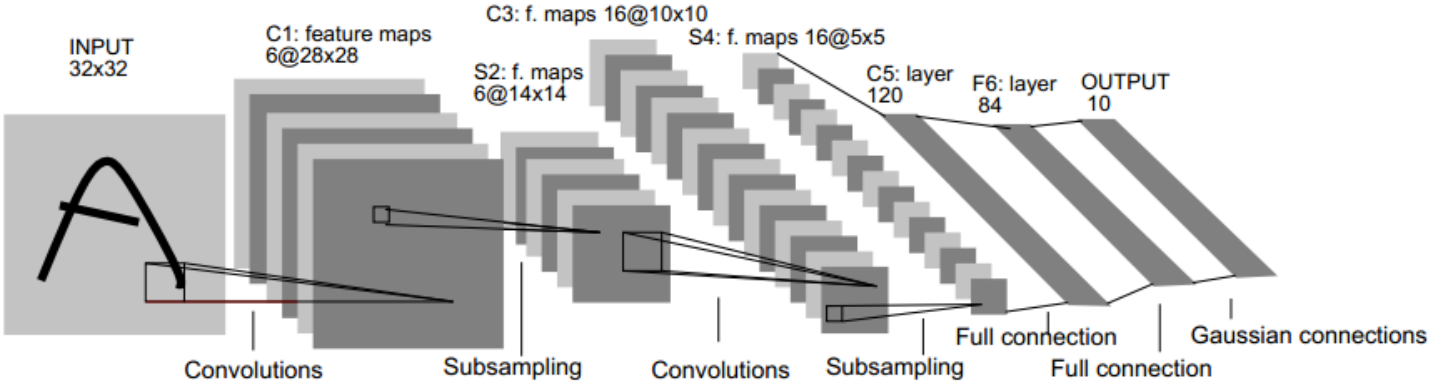
LeNet-5

- LeCun的表示法





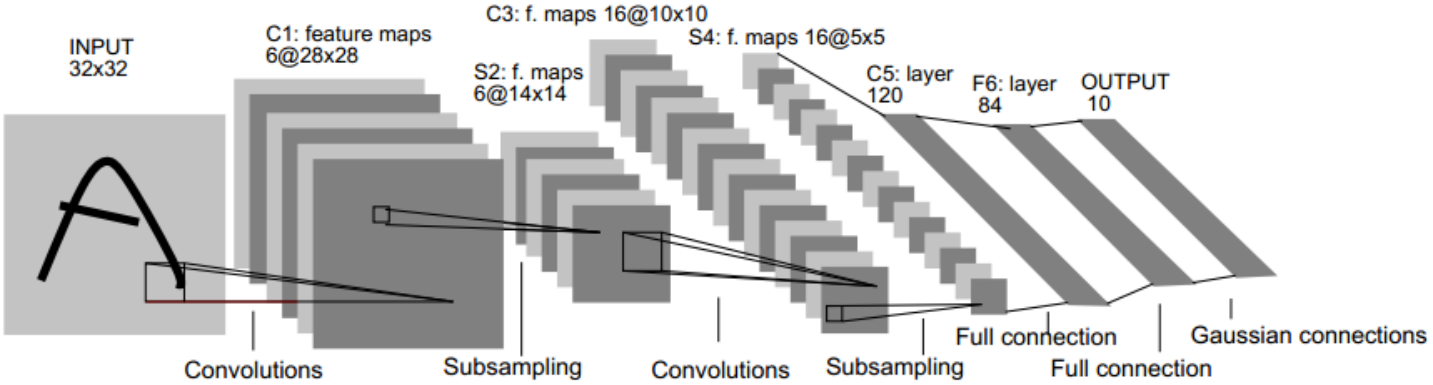
情况 3：卷积层与前一层的连接方式未定



- 设α_{ij} 为 第j个卷积核 与 前一层第i个特征图 之间的连接强度；
  则 卷积层的前向传播计算方式为：

$$\mathbf{x}_j^{\ell} = f\left(\sum_{i=1}^{N_{in}} \alpha_{ij}(\mathbf{x}_i^{\ell-1} * \mathbf{k}_i^{\ell}) + b_j^{\ell}\right) \qquad \sum_i \alpha_{ij} = 1, \quad \text{and} \quad 0 \le \alpha_{ij} \le 1.$$

情况 3：卷积层与前一层的连接方式未定



- 对于一个固定的卷积核而言，α_{ij}（即）α_i 的计算：

$$\frac{\partial E}{\partial \alpha_i} = \frac{\partial E}{\partial u^\ell}\frac{\partial u^\ell}{\partial \alpha_i} = \sum_{u,v}\left(\boldsymbol{\delta}^\ell \circ \left(\mathbf{x}_i^{\ell-1} * \mathbf{k}_i^\ell\right)\right)_{uv}$$

情况 3：卷积层与前一层的连接方式未定

- 进而，可以将 α_i 表示为 另一个值 c_i 的softmax函数：

$$\alpha_{ij} = \frac{\exp(c_{ij})}{\sum_k \exp(c_{kj})}.$$

  - 从而使 α_i 的约束条件得到加强，这时：

$$\frac{\partial \alpha_k}{\partial c_i} = \delta_{ki}\alpha_i - \alpha_i\alpha_k$$

$$\frac{\partial E}{\partial c_i} = \sum_k \frac{\partial E}{\partial \alpha_k}\frac{\partial \alpha_k}{\partial c_i} = \alpha_i\left(\frac{\partial E}{\partial \alpha_i} - \sum_k \frac{\partial E}{\partial \alpha_k}\alpha_k\right).$$

情况 3：卷积层与前一层的连接方式未定

- 进而，为了加强 α_i 的稀疏性，可以加入如下的规则化项 $\Omega(\boldsymbol{\alpha})$ 在此处键入公式。：

$$\tilde{E}^n = E^n + \lambda \sum_{i,j} |(\boldsymbol{\alpha})_{ij}|$$

$$\frac{\partial \Omega}{\partial c_i} = \sum_k \frac{\partial \Omega}{\partial \alpha_k} \frac{\partial \alpha_k}{\partial c_i} = \lambda \left(|\alpha_i| - \alpha_i \sum_k |\alpha_k| \right)$$

$$\frac{\partial \tilde{E}^n}{\partial c_i} = \frac{\partial E^n}{\partial c_i} + \frac{\partial \Omega}{\partial c_i}.$$

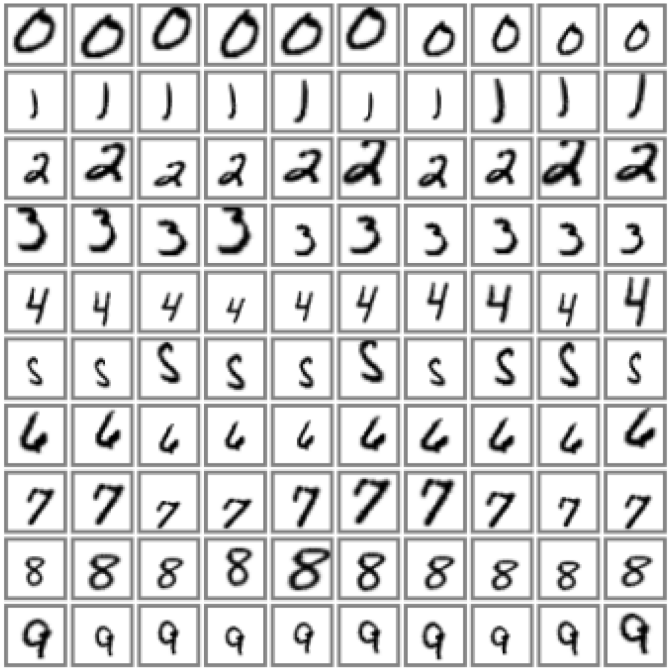
Results

- 人眼辨识的错误率约为：5.1%
- 2010年Alex Krizhevsky的CNN，错误率为15.3%
- 2012年微软研究团队，错误率已降低至4.94%
- 2015年Google团队，错误率降至：4.82%
- 2016年微软团队，图像分类错误率降低至3.57%；

LeNet-5 on MNIST

60,000 original datasets

Test error: 0.95%



540,000 artificial distortions

+ 60,000 original

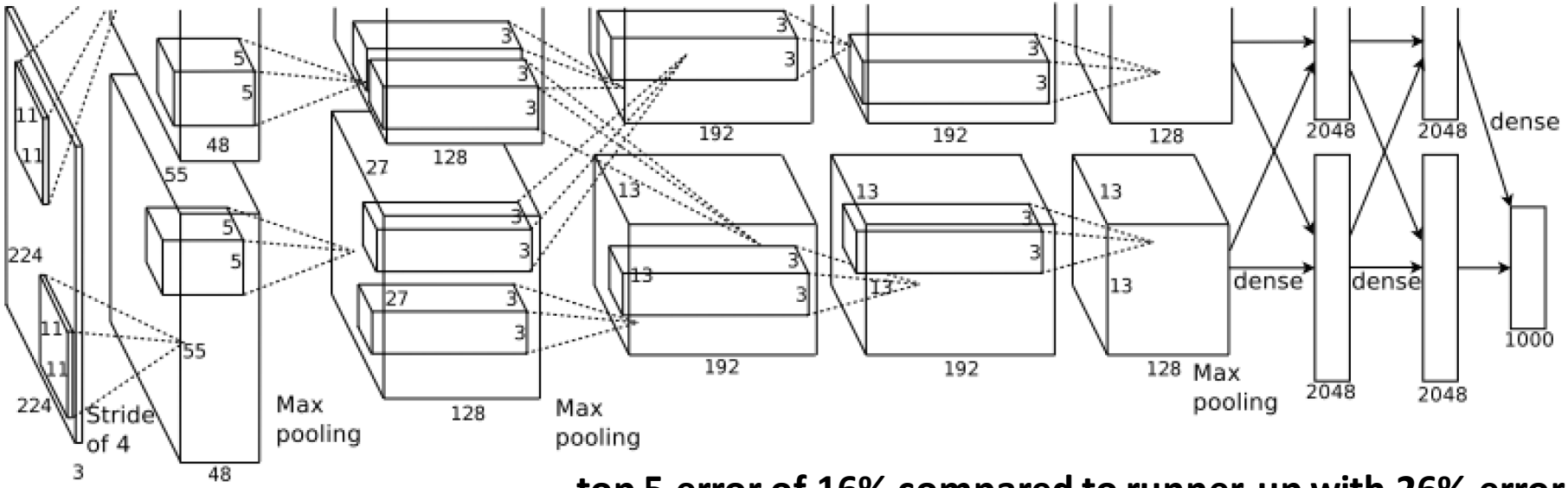
Test error: 0.8%

错误识别分析


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

2012，Alex Net

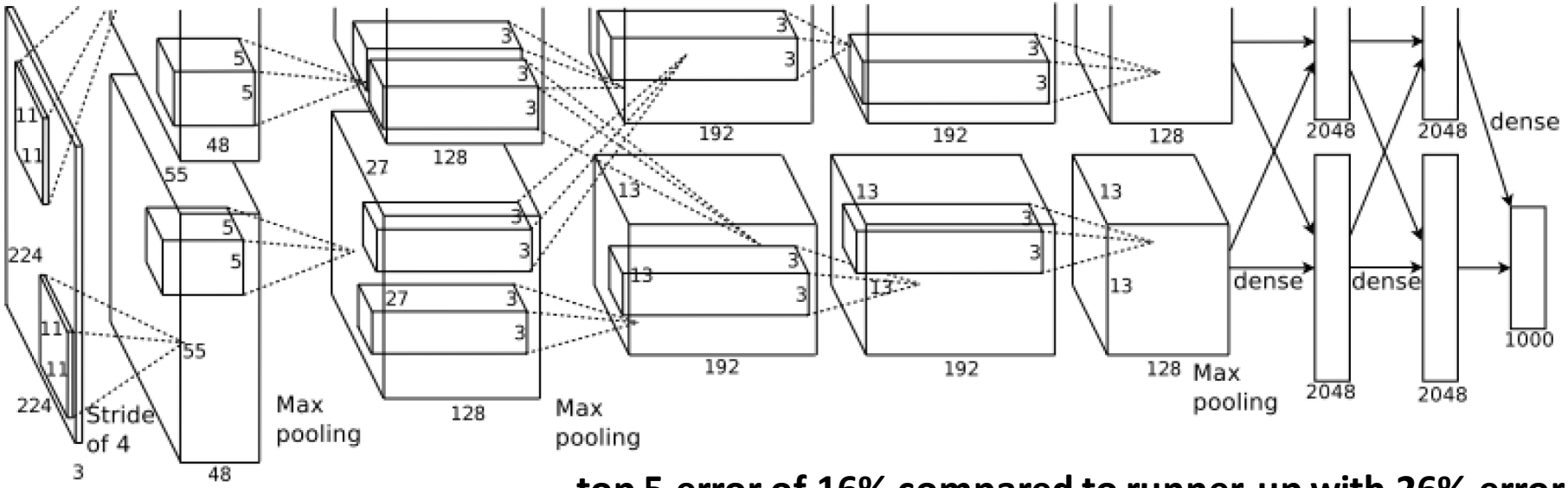
- 输入层：224×224大小的图片，3通道（RGB）
- 第一层卷积：5×5大小，跨度4，卷积核96个，每个GPU上48个。
- 第一层Maxpooling：窗口大小2×2



top 5 error of 16% compared to runner-up with 26% error

2012，Alex Net

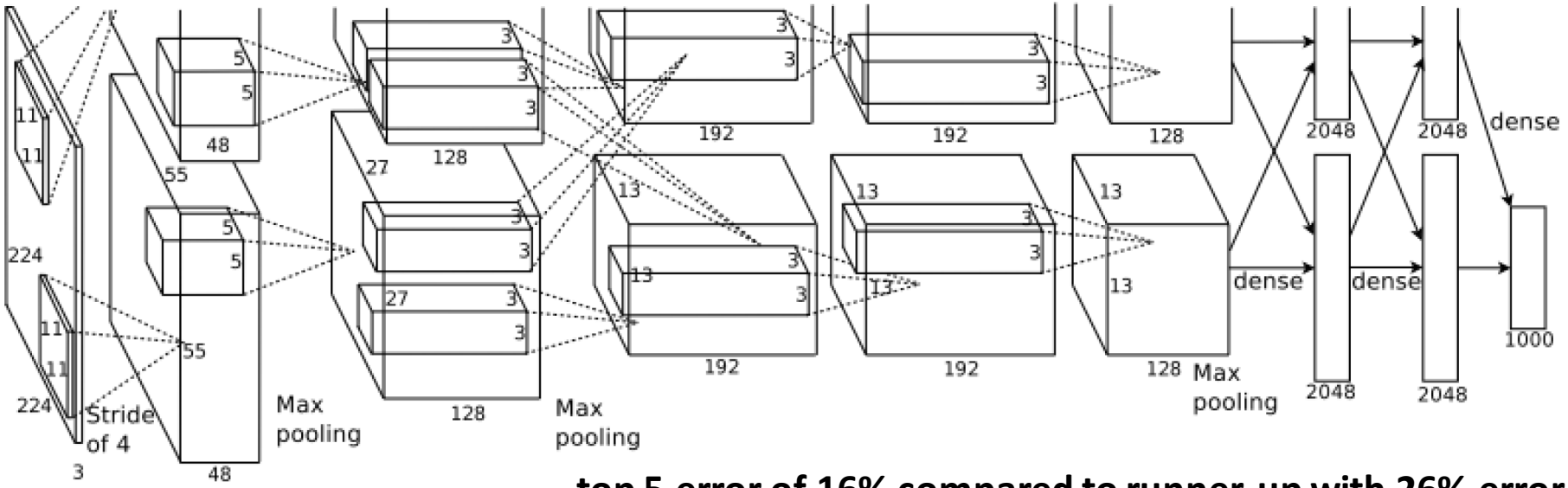
- 第二层卷积：3×3卷积核256个，每个GPU上128个。
- 第二层Maxpooling：窗口大小2×2
- 第三层卷积：与上层全连接，3*3的卷积核 384个，每个GPU 192个。



top 5 error of 16% compared to runner-up with 26% error

2012，Alex Net

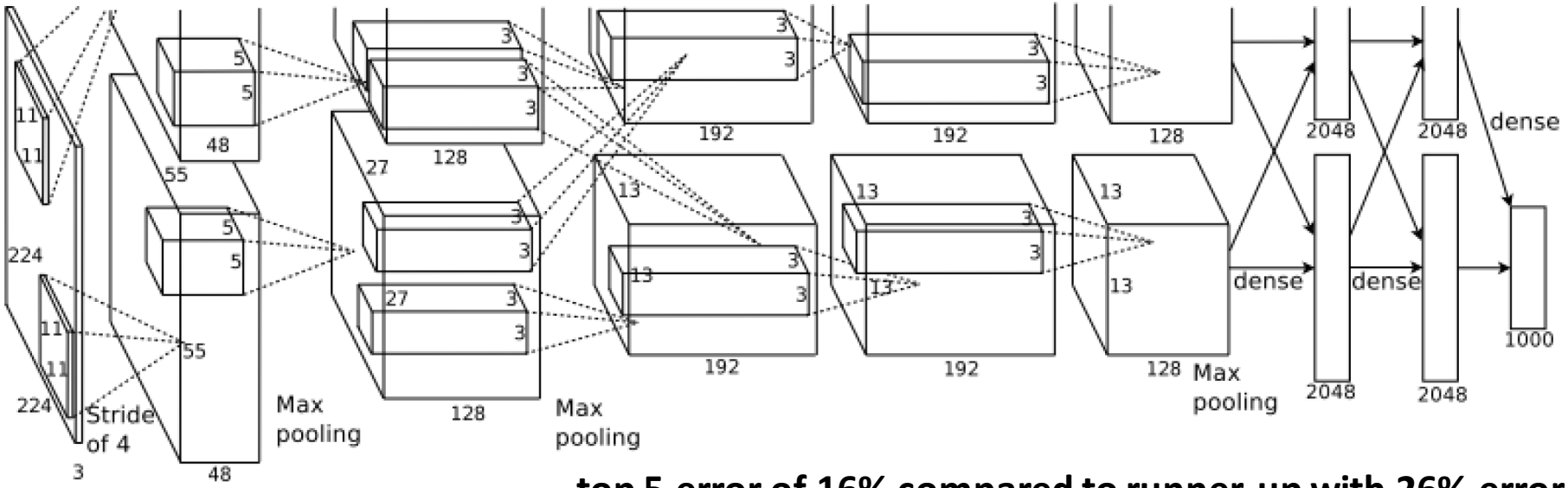
- 第四层卷积：3×3的卷积核384个，两个GPU各192个；
- 第五层卷积：3×3的卷积核256个，两个GPU上个128个；
- 第五层Maxpooling：窗口大小2×2



top 5 error of 16% compared to runner-up with 26% error

2012，Alex Net

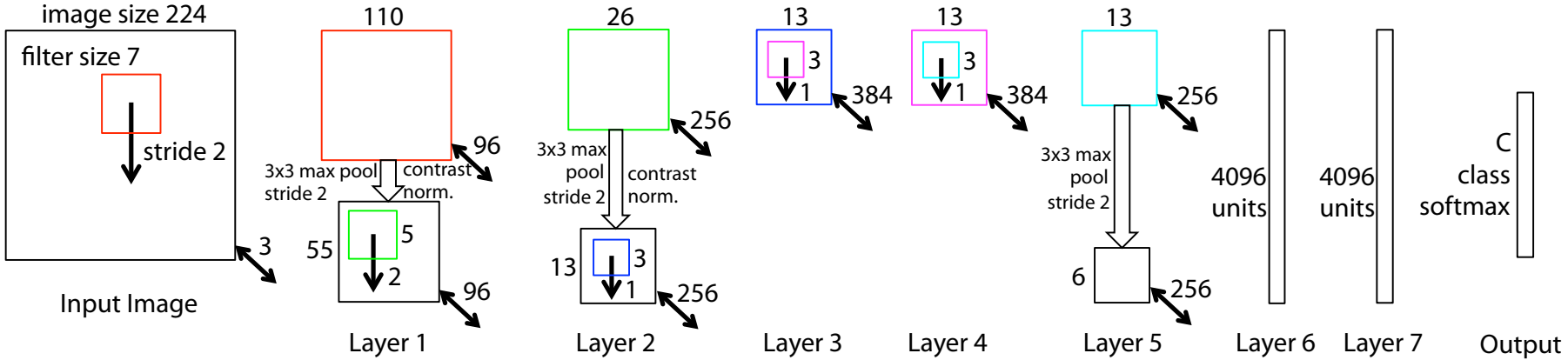
- 第一全连接：将第五层输出连接为一个一维向量作为输入，共4096维；
- 第二全连接：连接上层输入4096维；
- Softmax输出层：1000单元，每一维为图片属于该类别的概率。



top 5 error of 16% compared to runner-up with 26% error

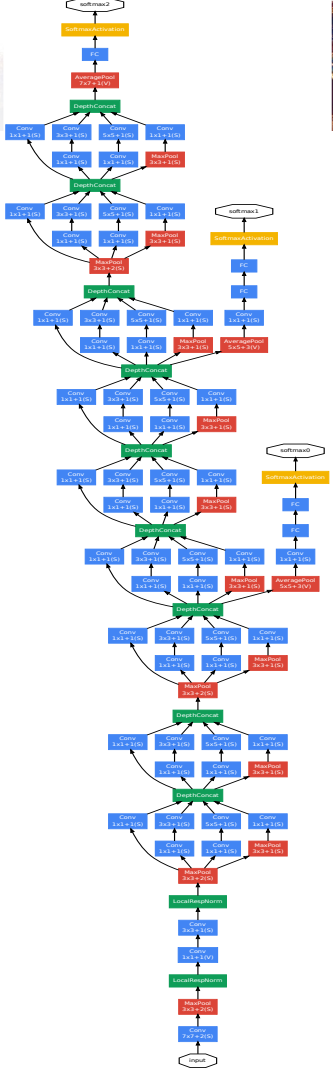
2013 ZF Net

- Visualizing and Understanding Convolutional Networks



2014 , GoogLeNet

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								



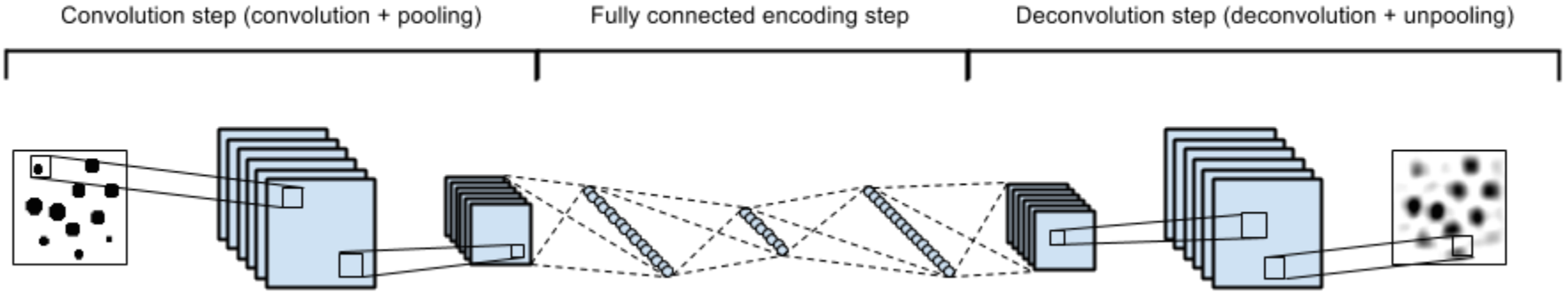
2014，VGGNet.

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					

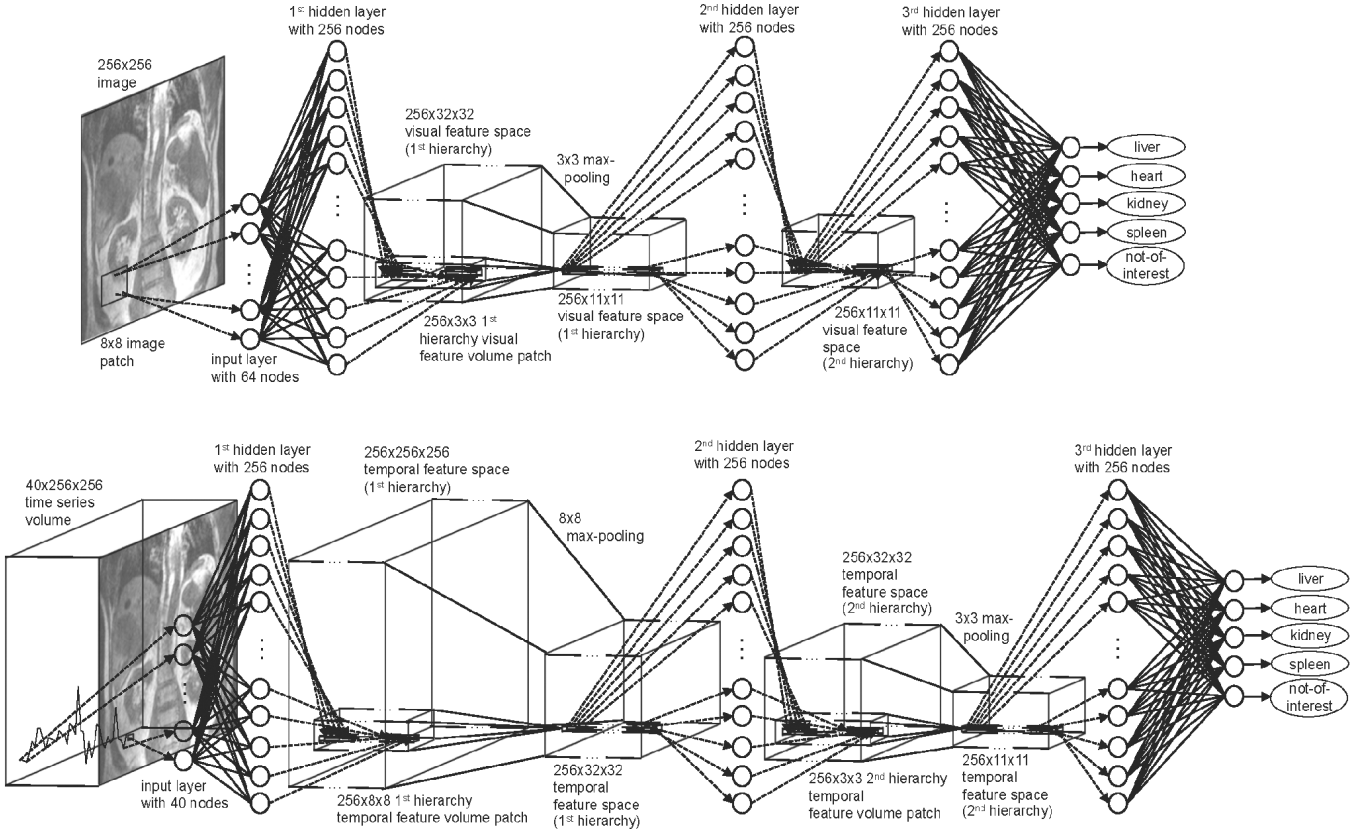
2015，ResNet

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Convolution & AutoEncoder



Convolution & AutoEncoder



2014/12/9 Network In Network

Table 4: Test set error rates for MNIST of various methods.

Method	Test Error
2-Layer CNN + 2-Layer NN [11]	0.53%
Stochastic Pooling [11]	0.47%
NIN + Dropout	0.47%
Conv. maxout + Dropout [8]	**0.45%**

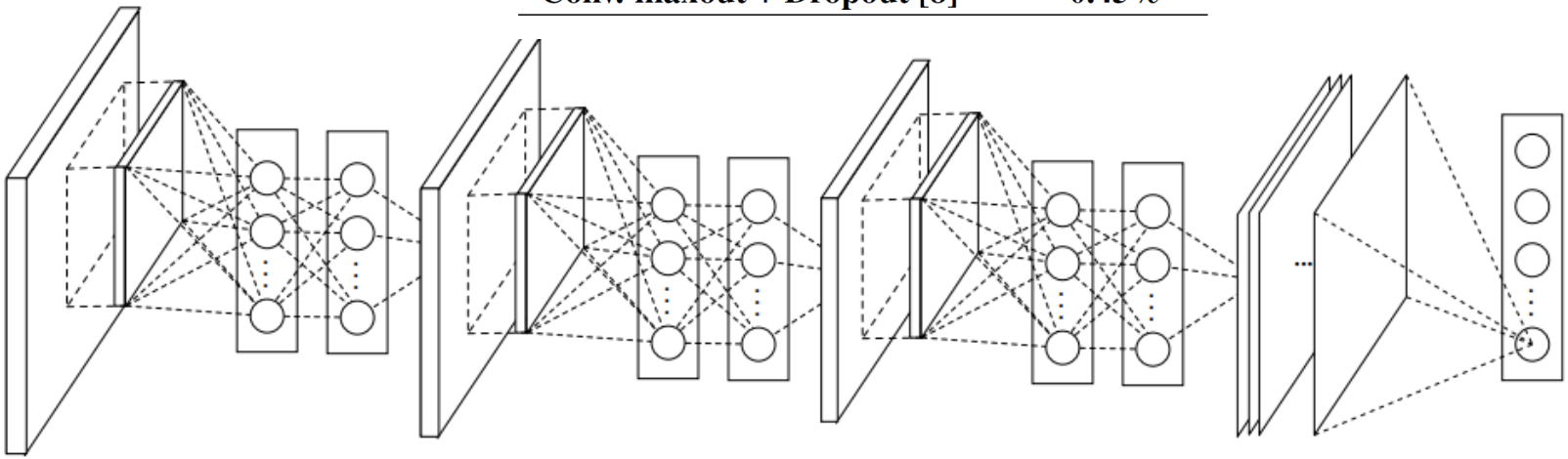


Figure 2: The overall structure of Network In Network. In this paper the NINs include the stacking of three mlpconv layers and one global average pooling layer.

Thanks.