

Lecture 3: 卷积神经网络

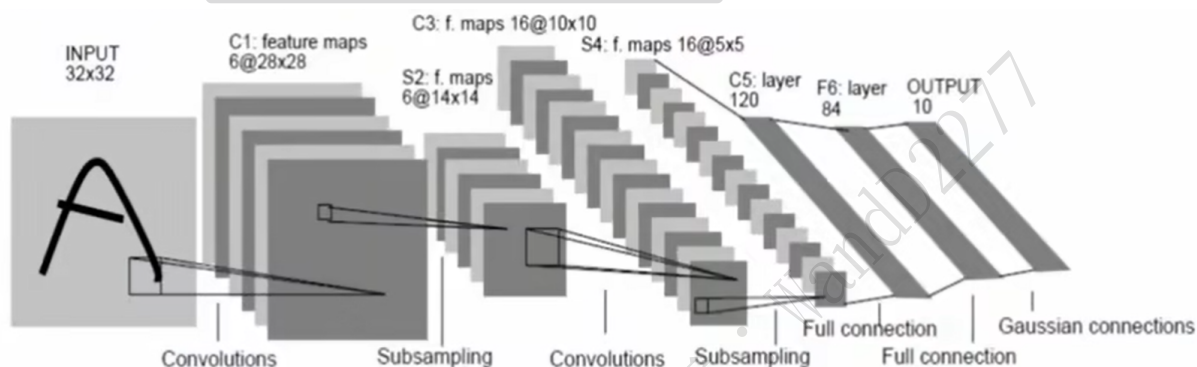
卷积神经网络基础

引入

概念：卷积神经网络的出现

- 概念：

- 卷积神经网络 (Convolutional Neural Network, CNN) 在上世纪 90 年代提出；



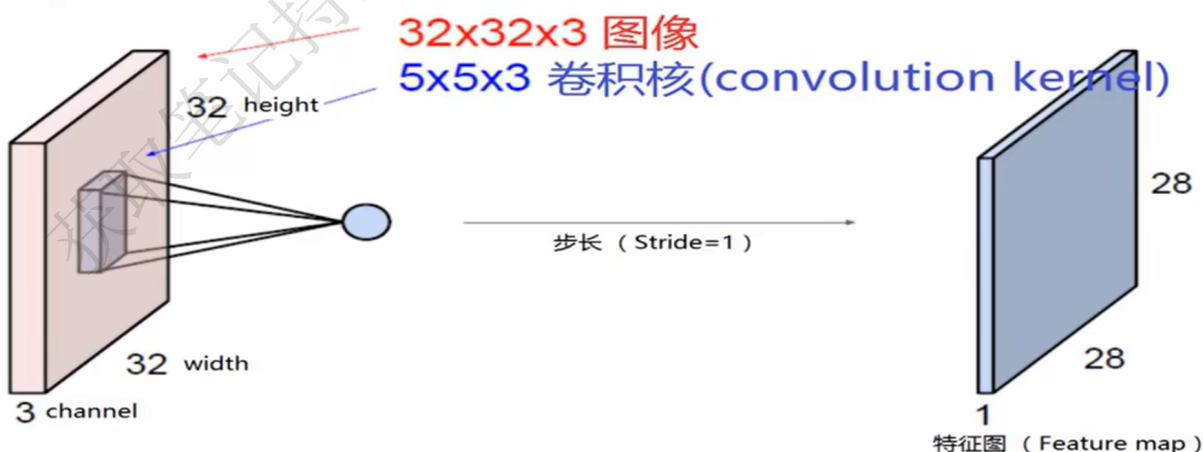
- 本质：

- 卷积神经网络，改变了之前由手工设计卷积核的方式，变成了使用大量样本、自动学习卷积核；

基础概念

概念：卷积神经网络基础概念

- 概念：图像



- 假设有一个 32×32 大小的彩色图像，彩色图像有 RGB 三个值；
- 因此是一个 $32 \times 32 \times 3$ 的立方体；
- 概念：卷积核 - convolution kernel
 - 图像大小为 $32 \times 32 \times 3$ ，卷积核大小为 $5 \times 5 \times 3$ ；
 - 这个卷积核将会作用在这个图像上，它会以此作用在这个图像的各个部分，小的这个可以在图像上任意移动，每移动到一个地方、将会输出一个数，这个数是：卷积核上的数字核

图像上被卷积核覆盖的区域的数组分别相乘再求和，得到一个数字；

- 概念：卷积过程

- 可以将图像卷积看成全连接网络的权值共享：

$$\begin{array}{|c|c|c|} \hline x1 & x2 & x3 \\ \hline x4 & x5 & x6 \\ \hline x7 & x8 & x9 \\ \hline \end{array} * \begin{array}{|c|c|} \hline w1 & w2 \\ \hline w3 & w4 \\ \hline \end{array} + \begin{array}{|c|c|} \hline b1 & b2 \\ \hline b3 & b4 \\ \hline \end{array} = \begin{array}{|c|c|} \hline p1 & p2 \\ \hline p3 & p4 \\ \hline \end{array}$$

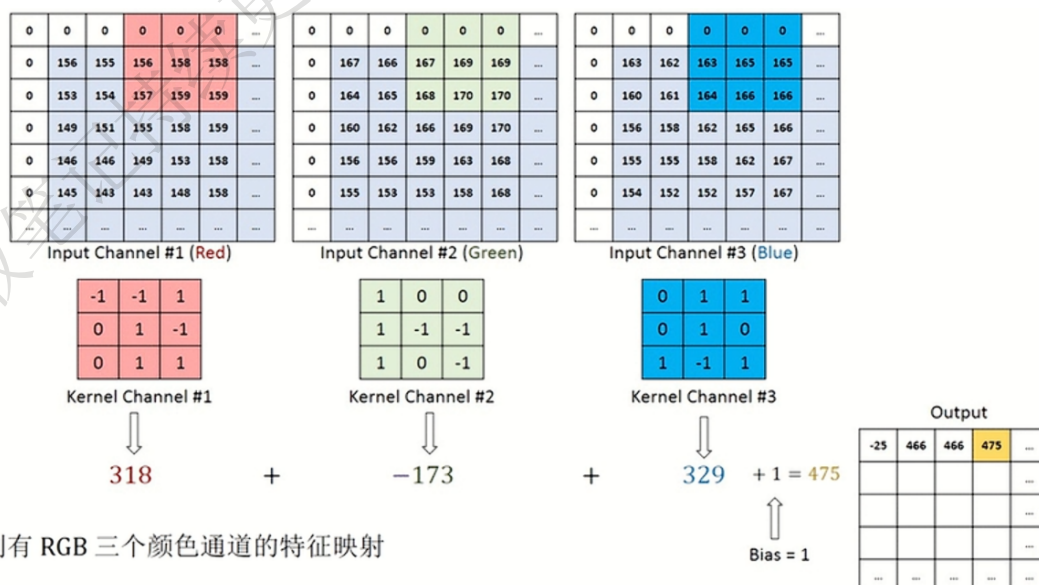
- $p1 = w1 \cdot x1 + w2 \cdot x2 + w3 \cdot x4 + w4 \cdot x5 + b1$
- $p2 = w1 \cdot x2 + w2 \cdot x3 + w3 \cdot x5 + w4 \cdot x6 + b2$
- $p3 = w1 \cdot x4 + w2 \cdot x5 + w3 \cdot x7 + w4 \cdot x9 + b3$
- $p4 = w1 \cdot x5 + w2 \cdot x6 + w3 \cdot x8 + w4 \cdot x9 + b4$
- 其中还要加上 b_1 到 b_2 的偏置；
- 在卷完一次之后、还需要往旁边移动一个单位，继续卷积；

- 概念：步长

- 在完成上一次卷积之后，下一次卷积、卷积核要移动多远；
- 假设 $\text{Stride}=[1,1]$ ，即每次往右边移动一格，然后移动到顶部之后、再从头开始，往下移动一格、再往右移动；
- 假设 $\text{Stride}=[2,2]$ ，即每次往右边移动两格；
- 最后卷积得到的结果，叫做特征图；

- 概念：通道

- 深度有时候也称为数据的“通道” channel；
- 卷积核通常与它所处理的数据/输入数据拥有同样的通道数；



- 概念：特征图

- 假设：

图像大小, (M, N)
卷积核大小 (m, n)
步长 (u, v)

- 因此，卷积得到的特征图每个点为：

$$\text{点1: } 1 - m$$

$$\text{点2: } (1 + u) \sim m + u$$

:

$$\text{点K: } 1 + (k - 1)u \sim m + (k - 1)u \leq M$$

- 因此：

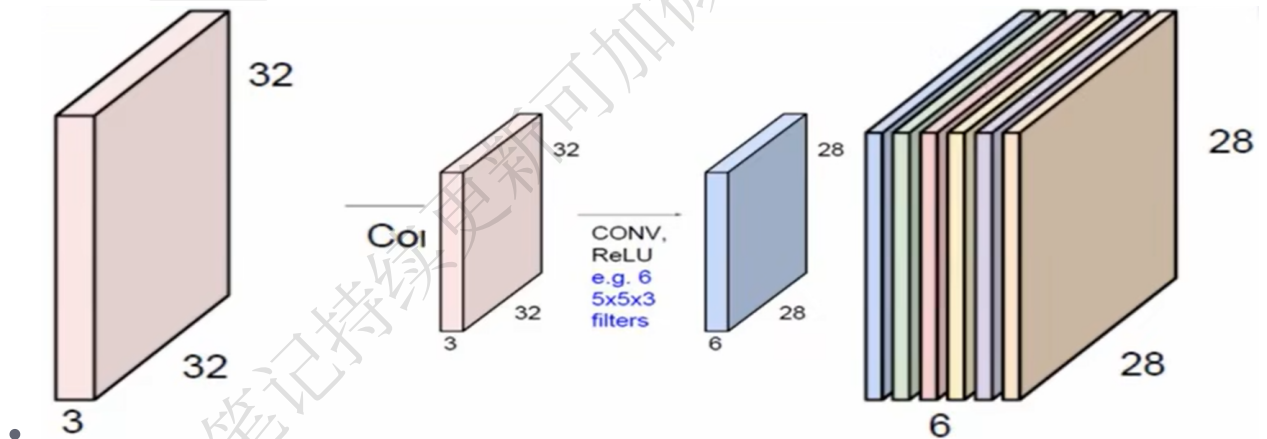
$$k \leq \frac{M - m}{u} + 1; \quad L \leq \frac{N - n}{v} + 1$$

- 概念：补零 Padding

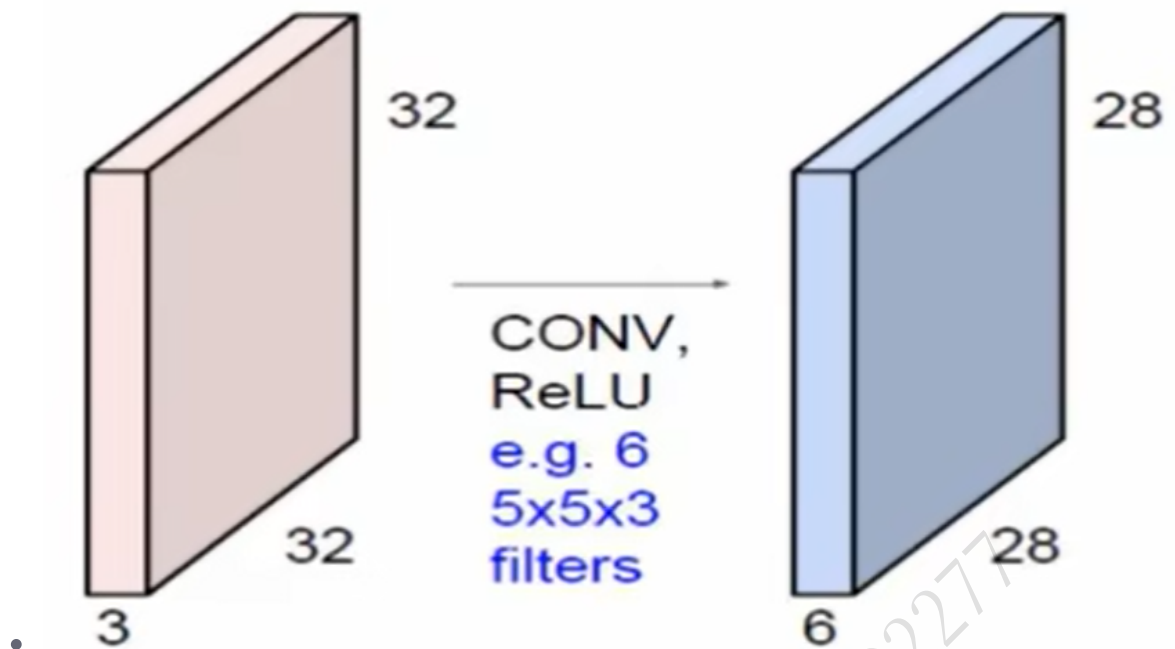
- 如果步长很大，可能会导致卷积核无法卷积到图像的部分区域。因此需要通过 Padding 的方式将图像扩大，从而使得卷积核可以卷积图像的全部区域；
- 问题：最多可以补多少个零？
 - 假设卷积核为 2×2 ，则最多 Padding 一圈；

- 概念：多卷积核

- 如果我们用 6 个卷积核，就能获得 6 个特征图 (feature map)；
- 你也可以把产生的 6 个特征图看成一个新的“图像”，其 height, width, channel 数目分别是 28, 28, 6；



- 简化画法：



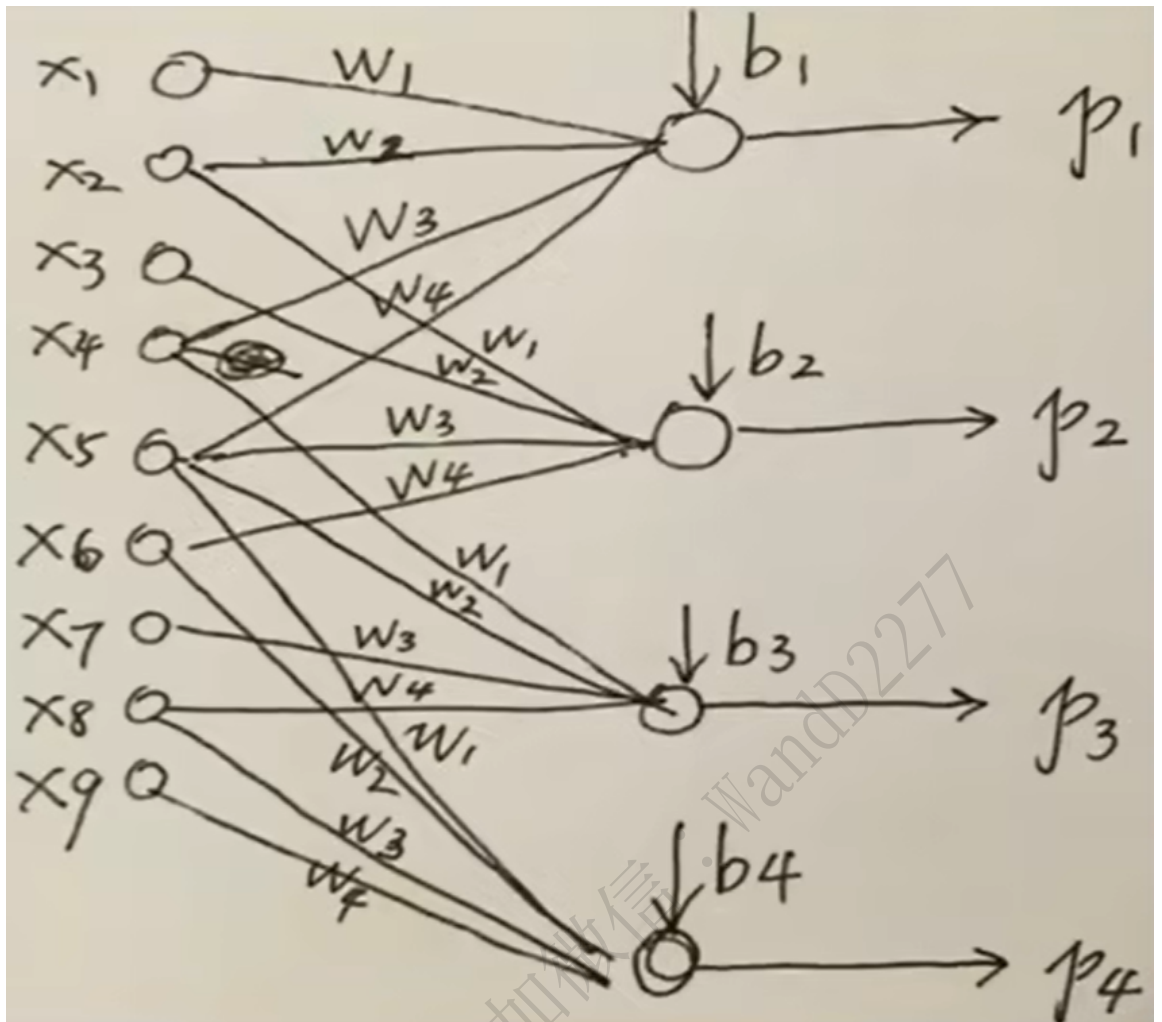
卷积神经网络与多层神经网络

概念：卷积神经网络与多层神经网络

- 概念：
 - 卷积神经网络：

$$\begin{array}{|c|c|c|} \hline x1 & x2 & x3 \\ \hline x4 & x5 & x6 \\ \hline x7 & x8 & x9 \\ \hline \end{array} * \begin{array}{|c|c|} \hline w1 & w2 \\ \hline w3 & w4 \\ \hline \end{array} + \begin{array}{|c|c|} \hline b1 & b2 \\ \hline b3 & b4 \\ \hline \end{array} = \begin{array}{|c|c|} \hline p1 & p2 \\ \hline p3 & p4 \\ \hline \end{array}$$

- 对应的传统神经网络形式：



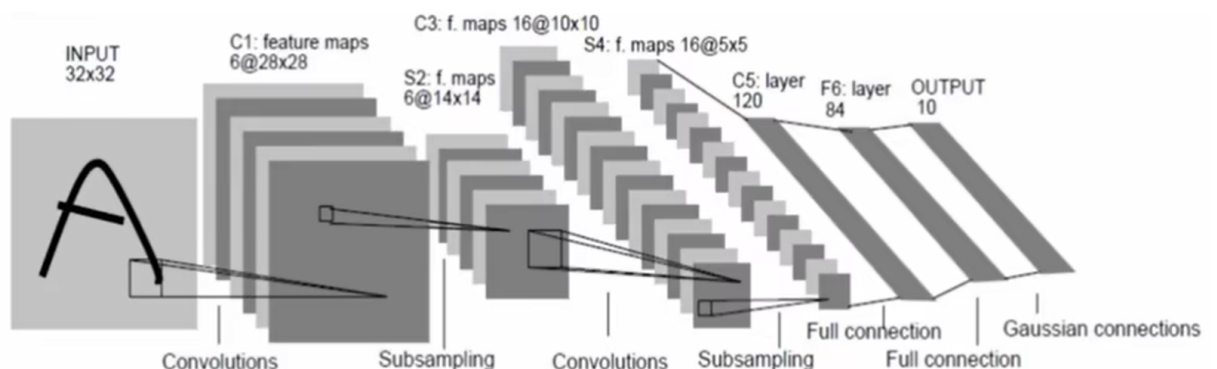
- 区别：传统神经网络当中，每一个神经元核每一个输入 x 都相连接。但卷积神经网络当中之和部分向连接；
- 概念：更本质的区别 - 共享权重
 - W_1 被四个神经元共享，其他的 W 同样；
 - 在后向传播时，

LeNet 卷积神经网络

模型分析

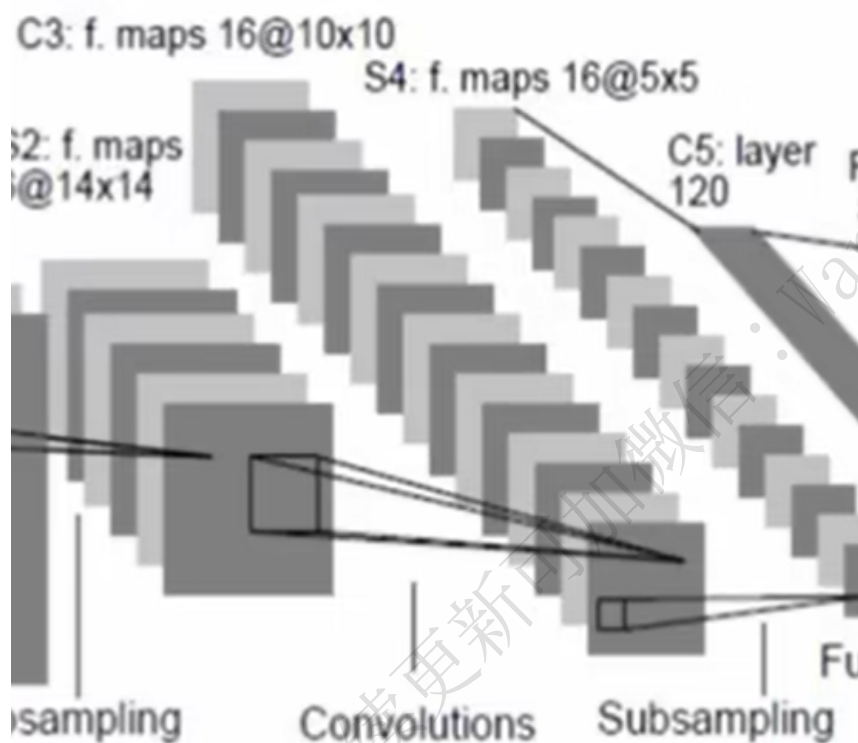
概念： LeNet 卷积神经网络

- 结构：

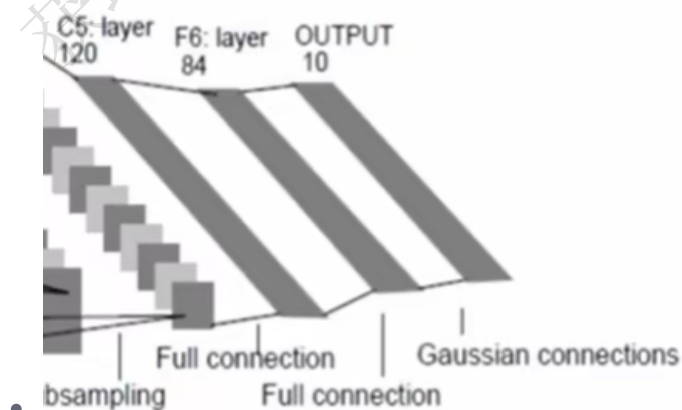


- 概念：SubSampling 层
 - 也称之为 降采样层；

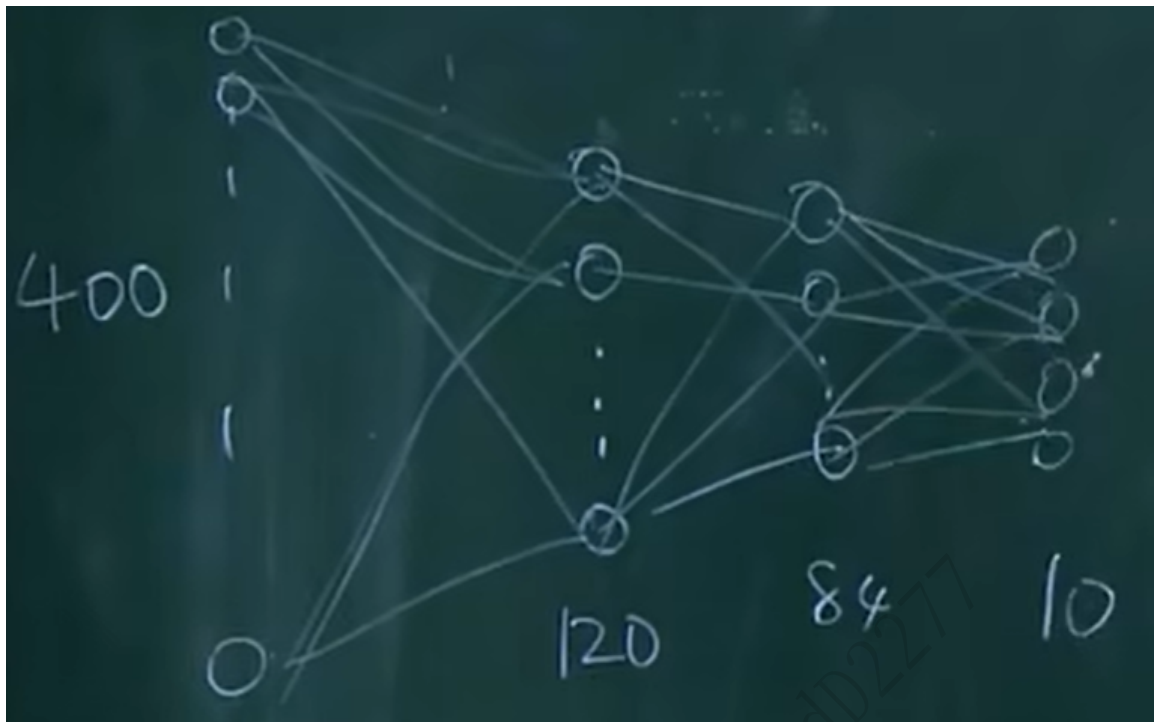
- 比如：在上图中，对 $28 \times 28 \times 6$ 的特征图中，每个 2×2 的矩阵进行一个求和平均，算成一个数；
- 通过 SubSampling 层通过每个 2×2 的矩阵进行一个求和平均，变成 $14 \times 14 \times 6$ 的层；
- 概念：非线性层
 - 非线性层不会改变图像的大小，因此未在上面标注出来；
 - 最常用的非线性函数是 ReLU；
 - 注意：图上的每一个线性变换之后，都必须接上一个非线性变换。因为如果两个线性变换接在一起，实际上等于是可以使用一个线性变换来替代；
- 分析：从 $14 \times 14 \times 6$ 的 Subsampling - 卷积 $10 \times 10 \times 16$
 - 用 16 个 $5 \times 5 \times 6$ 的卷积核，Stride = 1，作用到 $14 \times 14 \times 6$ 的特征图上；



- 概念：全连接层 - Full Connection
 - 从这一层，特征图完全展开；
 - 在 S4 中有 $5 \times 5 \times 16 = 400$ 个数；
 - 下一层的神经元个数为 120 个，将 400 个数核 120 个神经元全连接；



- 这一层的参数个数为 $(400+1) \times 120$ ，其中的 +1 为偏置；
- 然后再和 84 个神经元全连接，从 84 连接 10 个神经元；



- 在到达 10 之后，得到了 y ，使用 $\text{SoftMax}(z)=p$
- 最后使用 $E = -\sum_{i=1}^{10} Y_i \log(p_i)$ (交叉熵) 作为损失函数；

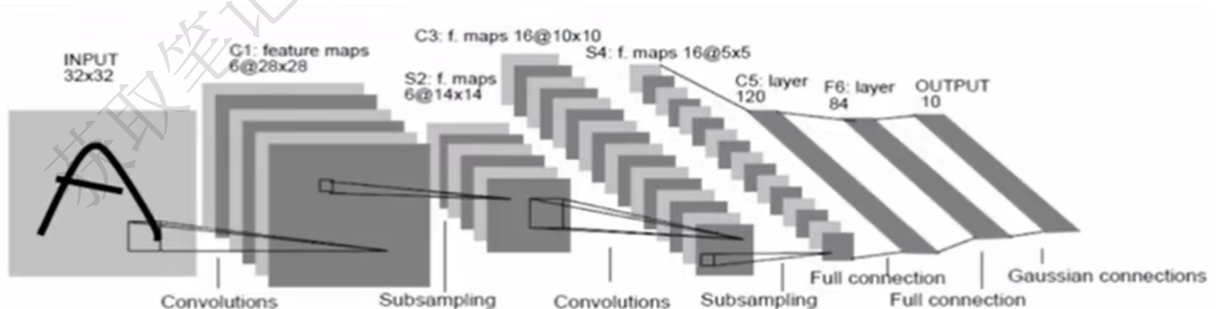
• 总结：

- 1. 卷积 + 池化 = 特征提取；
- 2. 全连接层 = 分类器；
- 3. 浅层的卷积神经网络得到的是比较细节的信息；
- 4. 深层的卷积神经网络得到的是比较整体、轮廓的信息；

参数个数

分析：参数个数

• 概念：



参数个数计算：

第1层(convolutional layer): $(5*5+1)*6=156$

第2层(subsampling layer): 0

第3层(convolutional layer): $(5*5*6+1)*16=2416$

第4层(subsampling layer): 0

第5层(fully connected layer): $(5*5*16+1)*120=48120$

第6层(fully connected layer): $(120+1)*84=10164$

第7层(fully connected layer): $(84+1)*10=850$

参数总数：61,706

- 其中，大部分的参数是在全连接层；
- 总结：
 - 整个网络的计算速度，取决于卷积层；
 - 整个网络的参数个数，取决于全连接层；

使用 Caffe 实现 LENET-5

代码：使用 Caffe 实现 LENET-5

- 代码：create_lmdb.sh

```
DATA=/home/hhj/caffe-master/examples/mnist
BUILD=/home/hhj/caffe-master/build/tools
```

```
rm -rf $DATA/mnist_train_lmdb
rm -rf $DATA/mnist_test_lmdb
```

```
$BUILD/convert_imageset --shuffle \
--resize_height=28 --resize_width=28 \
$DATA/ \
$DATA/training.txt $DATA/mnist_train_lmdb
```

```
$BUILD/convert_imageset --shuffle \
--resize_height=28 --resize_width=28 \
$DATA/ \
$DATA/testing.txt $DATA/mnist_test_lmdb
```

- 制定数据和编译的工具的目录；
- 然后，第一步是把所有的图片变成 Caffe 看得懂的形式；
- Caffeine 把这些图片弄好后，打包弄进来训练；
- 代码：compute_mean.sh
 - 算出训练图像的均值：


```
#!/usr/bin/env sh
# This script converts the mnist data into lmdb/leveldb format,
# depending on the value assigned to $BACKEND.
set -e

DATA=/home/hhj/caffe-master/examples/mnist
BUILD=/home/hhj/caffe-master/build/tools

rm -rf $DATA/mean.binaryproto

$BUILD/compute_image_mean $DATA/mnist_train_lmdb $DATA/mean.binaryproto $@
```
- 代码：train_lenet.sh

```
#!/usr/bin/env sh
set -e
```

```
BUILD=/home/hhj/caffe-master/build/tools
DATA=/home/hhj/caffe-master/examples/mnist
$BUILD/caffe train --solver=$DATA/lenet_solver.prototxt $@
```

- 代码: `lenet_solver.prototxt`

```
net: "examples/mnist/lenet_train_test.prototxt"
# test_iter specifies how many forward passes the test should
carry out.
# In the case of MNIST, we have test batch size 100 and 100 test
iterations,
# covering the full 10,000 testing images.
test_iter: 100
# Carry out testing every 500 training iterations.
test_interval: 500
# The base learning rate, momentum and the weight decay of the
network.
base_lr: 0.01
momentum: 0.0
weight_decay: 0.0005
```

```
# The learning rate policy
lr_policy: "inv"
gamma: 0.0001
power: 0.75
# Display every 100 iterations
display: 100
# The maximum number of iterations
max_iter: 10000
# snapshot intermediate results
snapshot: 5000
snapshot_prefix: "examples/mnist/lenet_rmsprop"
# solver mode: CPU or GPU
solver_mode: GPU
type: "RMSProp"
rms_decay: 0.98
```

- `test interval`: 每隔 500 次训练, 进行以此测试;
- 学习率: `base_lr` - 0.01

AlexNet

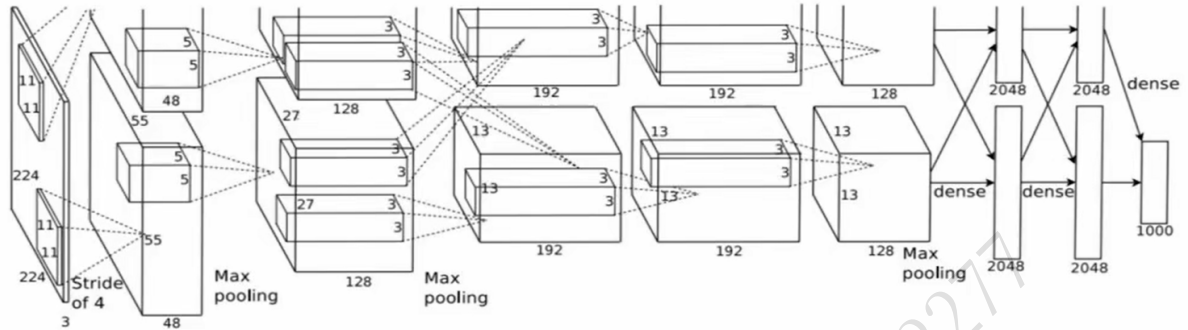
AlexNet 基本概念

概念: AlexNet 基本概念

- 概念:
 - ImageNet 包含超过 120 万张彩色图片, 属于 1000 个不同类别, 这是目前为止最大的图像识别数据库;

- Alex Krizhevsky 等人构建了一个包含 65 万多个神经元，待估计参数超过 6000 万的大规模网络，这一网络被称为 AlexNet；
- 在第二层输入的 96 个卷积核，上面 48 个是一个 GPU，下面是另一个 GPU；
- 经过一个卷积层和池化层，最后经过若干全连接层，得到 1000 维的向量；
- 网络结构上、和 LENET 没有本质的区别；

• 结构：



• 分析：

- 第一层：224*224 的图片。11*11 的卷积核，一共有 16 个卷积核；
- 第二层：经过 Max Pooling；
- 第三层：卷积...

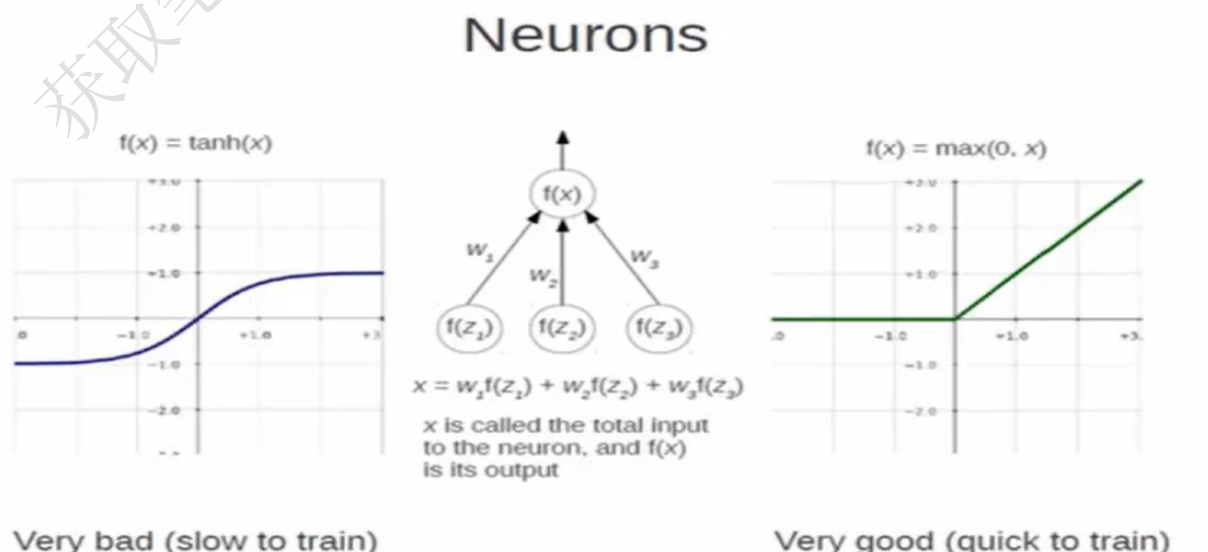
• 结果：

- Alex、Krizhevsky 等人在 ImageNet 的测试集上获得了 37.5% 的 Top1 错误率 (即正确的类不是测试中排名第1的类的百分率)和 16.4% 的 Top5 错误率 (即正确的类不是测试中排名前五的类的百分率)，这比第二名 45.7% 的 Top1 错误率和 25.8% 的 Top5 错误率高出了 8 个百分点；
- 考虑到第二名与第三名的差距只有 1 个百分点左右，这个结果可以说是遥遥领先；

AlexNet 带来的改进

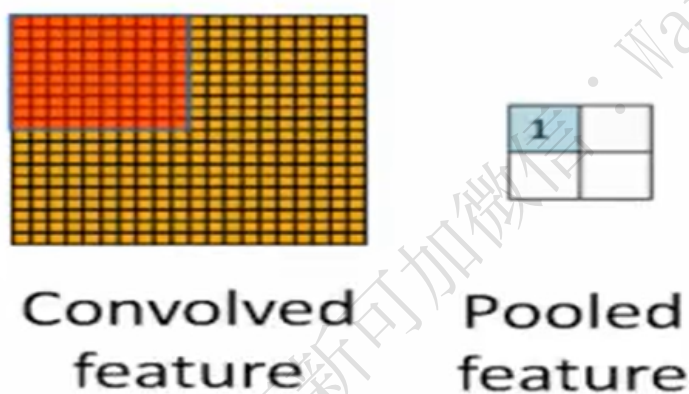
分析：ALexNet 和 LeNet 的区别

- 改进一：以 ReLU 函数代替 sigmoid 或 tanh 函数



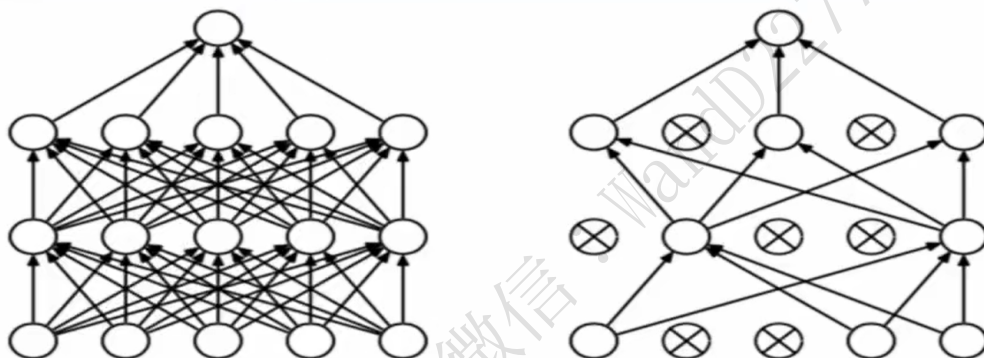
- 如果神经元的个数非常多、模型特别复杂的时候，使用 Sigmoid 将会出现问题；

- 因为在后向传播时、如果每一个神经元都激活，都往后传输值，在神经网络很大时将会产生问题；
- 而如果使用 ReLU，则可以保证在每一层当中，激活的神经元个数只有一半左右；
- 因此可以更加有效的控制收敛的速度，使得收敛性变得更好；
- 并且，因为 ReLU 在大于零时、其梯度始终为 1，因此使得其在向前传播时可以传更深远的影响；
- 改进二：使用池化技术
 - 概念：池化
 - Pooling
 - 含义：
 - 图像中的相邻像素倾向于具有相似的像素值，因此通过卷积层相邻的像素输出的也是具有相似的值。这意味着卷积层当中输出的大量值都是冗余的；
 - 因此可以使用池化，意思是把邻近的像素作为一个“池子”来重新考虑；
 - 如图所示，左边所有红色的像素值可以看做是一个“池子”经过池化操作后，变成右边的一个蓝色像素；



- 在 AlexNet 中，提出了最大池化 (Max Pooling) 的概念，即对每一个邻近像素组成的“池子”，选取像素最大值作为输出；
- 在 LeNet 中，池化的像素是不重叠的，而在 AlexNet 中进行的是有重叠的池化；
- 实践表明，有重叠的最大池化能够很好的克服过拟合问题，提升系统性能；
- 补充：
 - 前向这样计算池化时，往后传递时、使用的是赢者通吃的方式，取最大的那个值、直接把梯度值传递回去；
- 分析：
 - 为什么池化技术可以使得性能更好？
 - 1. 做了降采样；
 - 当参数很多的时候，时刻要考虑收敛的问题；
 - 使用池化时，除了最大值之外，其他值都不会被考虑，从而让激活的神经元更少，因此可以更好的收敛；
 - 2. 做了一个非线性的操作；
 - 求所有值的最大值，是一个非线性的操作；
 - 因为如果全是线性的变化，功能会有所重合；
- 举例：
 - 最大池化：更多的保留图片的纹理特征和局部特征；

- 平均池化：更多的保留图片的背景信息和全局特征；
- 改进三：随即丢弃
 - 概念：
 - 随机丢弃 (Drop out)；
 - 为了避免系统参数更新过快导致过拟合，每次利用训练样本更新参数时候，在设定的层次里面，随机的“丢弃一定比例的神经元，被丢弃的神经元将不参加训练过程，输入和输出该神经元的权重系数也不做更新；
 - 这样每次训练时训练的网络架构都不一样，而这些不同的网络架构却分享共同的权重系数；
 - 实验表明，随机丢弃技术减缓了网络收敛速度，也以大概率避免了过拟合的发生；
 - 分析：
 - 图示：



- 假设这三层、每一层都做丢弃；
- Dropout 做法是，对每一层，每次训练时以概率 p 丢弃一些神经元，这样每次训练的网络都不一样；
- 训练结束后的测试流程，要用完整的网络结构，同时对该层的所有的参数 (w, b) 都要乘以 $(1-p)$

- 代码：

```
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.argmax(y_conv, 1), tf.argmax(y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))

sess.run(tf.global_variables_initializer())
```

```
for i in range(10000):
    batch = mnist.train.next_batch(50)
    if i%100 == 0:
        train_accuracy = accuracy.eval(feed_dict={x:batch[0], y_:
        batch[1], keep_prob: 1.0})
        print "step %d, training accuracy %g"%(i, train_accuracy)
        train_step.run(feed_dict={x: batch[0], y_: batch[1], keep_prob:
        0.5})
```

识别率97.5%

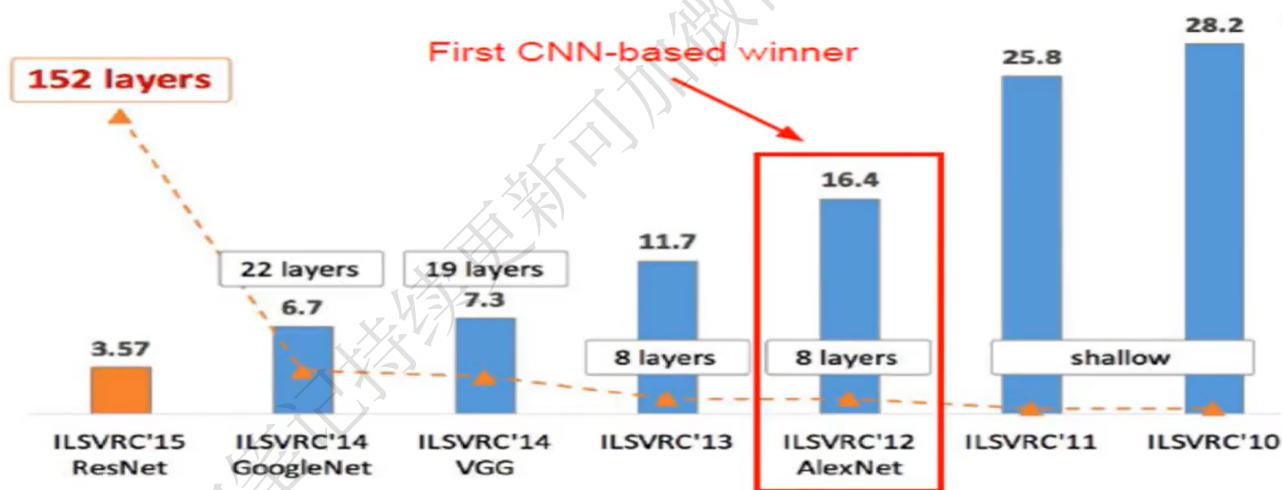
```
print "test accuracy %g"%accuracy.eval(feed_dict={
    x: mnist.test.images, y_: mnist.test.labels, keep_prob: 1.0})
```

- 每一次训练要丢弃一定的值；
- 效果：
 - 可以加快网络的收敛，其所做的事情和池化的效果本质上是一样的，即每一次只改变一定的参数；

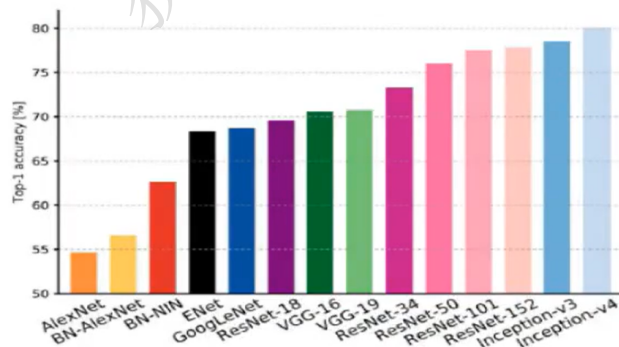
- 并且进行随机丢弃时，进行训练时、每一层不断变化，相当于是同时在训练一大堆网络，然后将一大堆网络平均、合到一起，作为完整的结果；
- 改进四：增加训练样本
 - 概念：
 - 尽管 ImageNet 的训练样本数量有超过 120 万幅图片，但相对于 6 亿待估计参数来说，训练图像仍然不够；
 - Alex 等人采用了多种方法增加训练样本，包括：
 - (1) 将原图水平翻转；
 - (2) 将 $256 * 256$ 的图像随机选取 $224 * 224$ 的片段作为输入图像；
 - 运用上面两种方法的组合，可以将一幅图像变为 2048 幅图像；
 - 还可以对每幅图片引入一定的噪声，构成新的图像；
 - 这样做可以较大规模增加训练样本，避免由于训练样本不够造成的性能损失；
 - 改进五：使用 GPU 加速训练过程
 - 采用 2 片 GTX 580 GPU 对训练过程进行加速，由于 GPU 强大的并行计算能力，使得训练过程的时间缩短数十倍，哪怕这样，训练时间仍然用了六天；

其他常见网络结构

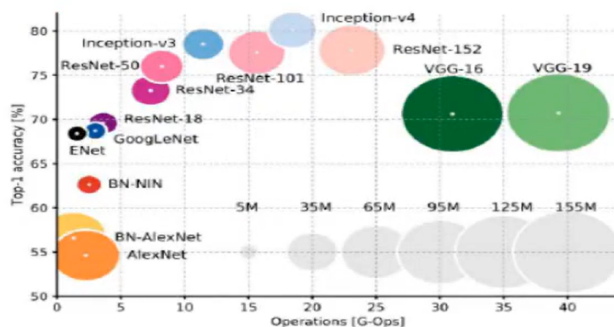
概念：近年流行的网络结构



各种不同网络在 IMAGENET 上的结果



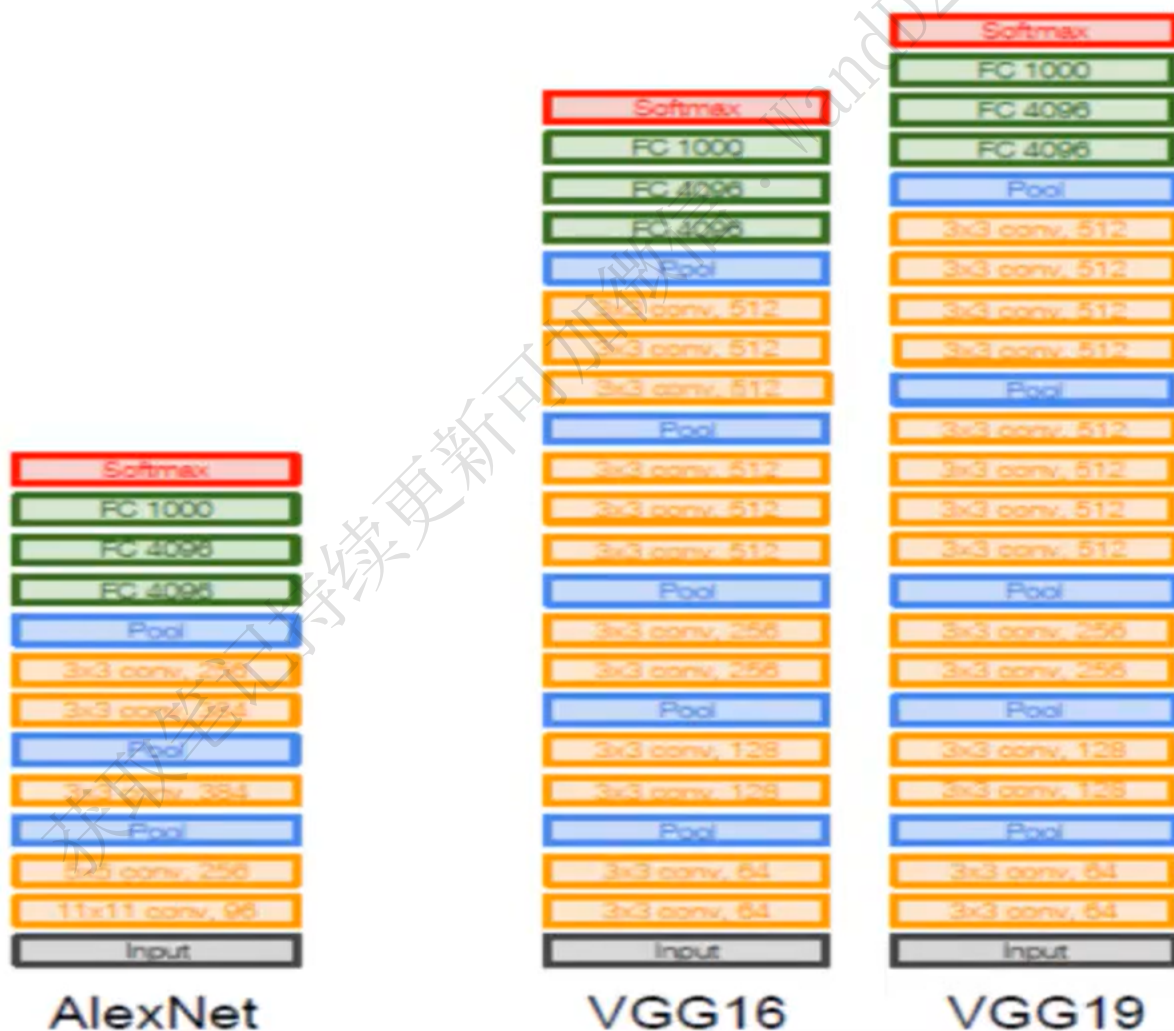
不同网络识别率比较



不同网络计算量和识别率的联合比较

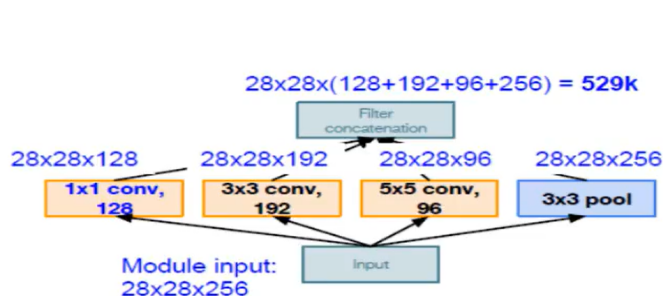
概念: VGGNet

- 引入:
 - 为何要将 2 个 3×3 卷积核叠到一起?
 - 因为 2 个叠到一起的 3×3 卷积核, 感受野 (Receptive Field) 是 7×7 , 大致可以替代 7×7 卷积核的作用;
 - 但这样做可以**使参数更少**, 参数比例大致为 18:49;
- 概念: 感受野
 - 将每一个神经元、推到你最原始的那副图上时, 其占据几个神经元;
 - 比如有三个层, Layer3 中的一个点是经过了两次卷积之后、在原本的 Layer1 中 7×7 的大小的图像, 其在经过两次卷积后、变成了 Layer3 中的一个点;
- 结果:
 - 这样导致的结果时、其计算的速度更加的慢, 但识别率提升了;
- 图示:

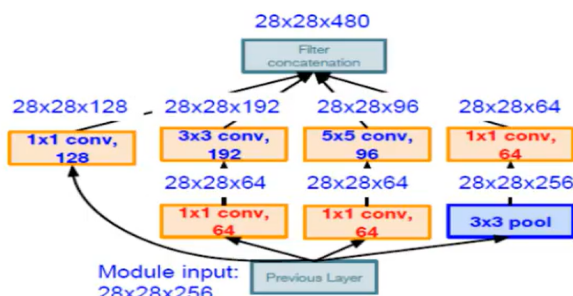


GoogleNet

概念: GoogleNet



最初的Inception结构



改进的Inception结构

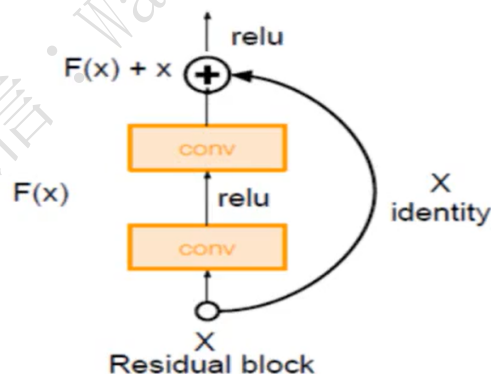
ResNet - 残差网络

概念: ResNet - 残差网络

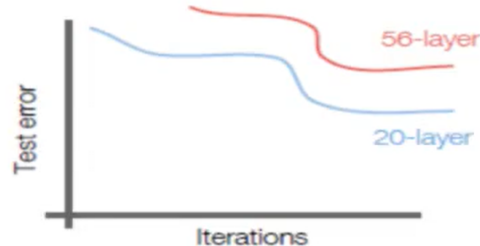
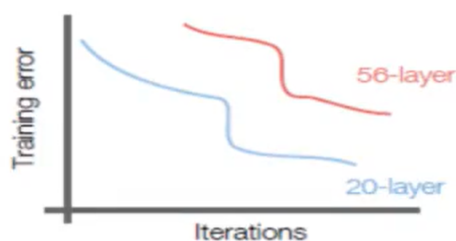
• 概念:

- 把前面层次的输入, 直接加到后面去;
- 如果不把前面层的输入加到前面去, 层次变得很深后、可能不收敛;

(1) 152层
(2) ILSVRC'15冠军,
(3.57 TOP 5 ERROR)
(3) 加入了前向输入机制, 将前面层获得的特征图作为监督项输入到后面层。用这样的方法使深层网络训练能够收敛。



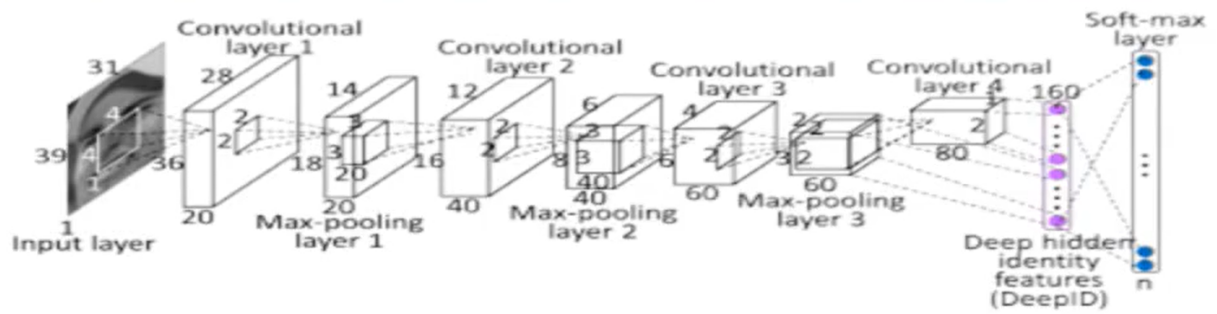
- 但是如果直接将前面层次的内容输入到后面, 可以使得收敛速度变快;
- 分析:
 - (1) 作者首先发现了, 训练一个浅层的网络, 无论在训练集还是在测试集上, 都比深层网络表现好, 而且是在训练各个阶段持续的表现好;
 - (2) 在这个例子中, 既然 20 层的网络表现比 56 层的好, 那么大不了另外 36 层什么事情也不做, 于是产生了将浅层网络的输出直接加到后面层的想法;



卷积神经网络的应用

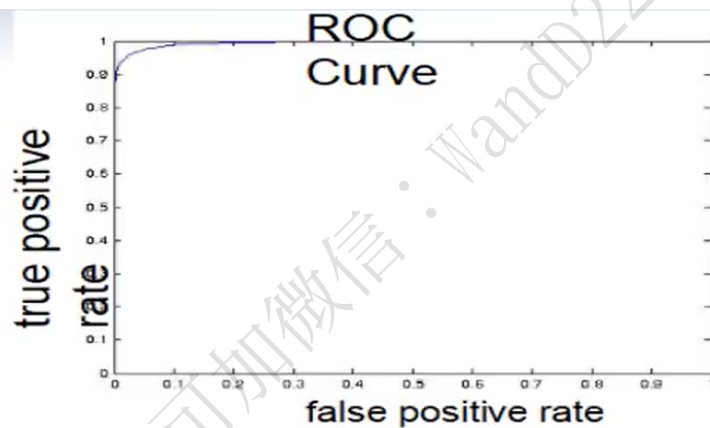
举例: 卷积神经网络的应用

- 2015 年, DeepID



Sun Y., Wang X. and Tang X., [Deep learning face representation from 10,000 classes](#). In Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, pp. 1891-1898, 2014.

这一系统在LFW (Labeled Faces in the Wild)数据库上获得了99.15%的人脸辨识率，超过了人用肉眼在LFW上的辨识率97.52%。这也是第一个声称表现超过人的人脸识别系统。



在身份证场景下的ROC图

识别率：在LFW上，我们 0.992 vs DeepID 0.975

时间复杂度：
在在瑞星微 RK3288 上硬件实现 ~20-30ms/帧, 实时.

未来规划：

- 实际场景的更高识别率
- 结合具体硬件对卷积神经网络的优化

• 补充：Transfer Learning - 迁移学习

- 比如现在要做身份证识别，但是现在只有几百万张欧美人脸的图片；
- 此时可以使用欧美人连图片进行训练，训练完之后，再使用十万张亚洲人脸的图片、对这个模型进行调优，使得其能够识别亚洲人的人脸；
- 然后再使用几万张身份证的图片、再对其进行调优，使得其能够识别身份证照片上的亚洲人的人脸；
- 即用训练好的网络、做特征体的提取器；